Parte iv

# COBOL

Ogni manuale COBOL tradizionale riporta una premessa che cita le origini del linguaggio e le fonti a cui si fa riferimento. Questo tipo di premessa ha soprattutto un valore storico e con tale spirito viene inserita qui.

Il testo seguente è una traduzione tratta dalla pubblicazione *COBOL*, edita dalla Conferenza sui linguaggi dei sistemi di elaborazione dati, CODASYL (*Conference on data system languages*), stampata a cura dell'ufficio stampa del governo degli Stati Uniti d'America.

> «*Questa pubblicazione si basa sul sistema COBOL sviluppato nel 1959 da un comitato composto da utenti governativi e costruttori di elaboratori. Le organizzazioni che hanno preso parte ai lavori iniziali sono state:*
>
> > *Air Material Command, U.S. Air Force;*
> > *Bureau of Standards, U.S. Department of Commerce;*
> > *Burroughs Corporation;*
> > *David Tylor Model Basin, Bureau of Ships, U.S. Navy;*
> > *Electronic Data processing Division, Minneapolis-Honeywell Regulator Company;*
> > *International Business Machines Corporation;*
> > *Radio Corporation of America;*
> > *Sylvania Electric Products, Inc.;*
> > *UNIVAC Division of Sperry Rand Corporation.*
>
> *Oltre alle suddette organizzazioni, le seguenti altre partecipano ai lavori del Gruppo di revisione:*
>
> > *Allstate Insurance Company;*
> > *The Bendix Corporation, Computer Division;*
> > *Control Data Corporation;*
> > *E.I. du Pont de Nemours and Company;*
> > *General Electric Company;*
> > *General Motors Corporation;*
> > *The National Cash Register Company;*
> > *Philco Corporation;*
> > *Standard Oil Company (New Jersey);*
> > *United States Steel Corporation.*
>
> *Questo manuale COBOL è risultato dalla collaborazione fra tutte le organizzazioni citate.*
>
> *Nessuna garanzia, espressa o tacita, è fornita dal comitato o dai singoli collaboratori, circa l'esattezza e il buon funzionamento del sistema di programmazione e del linguaggio. Inoltre, sia il comitato, sia i suoi collaboratori, non si assumono alcuna responsabilità in ordine a quanto esposto.*
>
> *È ragionevole attendersi che molti perfezionamenti e aggiunte vengano fatte al COBOL. Si farà ogni sforzo per assicurare che miglioramenti e correzioni siano apportate con criteri di continuità, tenendo debito conto degli investimenti effettuati dagli utenti nel settore della programmazione. Tuttavia, tali garanzie potranno essere efficacemente mantenute soltanto da coloro che apporteranno perfezionamento o correzioni.*
>
> *Sono state predisposte apposite procedure per l'aggiornamento del COBOL. Le richieste di informazioni circa tali procedure e sulle modalità per proporre modifiche dovranno essere inoltrate al comitato esecutivo della Conferenza sui linguaggi dei sistemi di elaborazione dati.*
>
> *Gli autori e i titolari dei diritti di autore e di riproduzione del materiale così protetto, usato nel presente manuale:*

*FLOW-MATIC (marchio depositato dalla Sperry Rand Corporation) - Sistema di programmazione per i calcolatori UNIVAC® I e II, Data Automation Systems © 1958, 1959 Sperry Rand Corporation; IBM Commercial Traslator, Codice F28-8013, © 1959 IBM; FACT, DSI 27A5260-2760 © 1960 della Minneapolis-Honeywell; hanno esplicitamente autorizzato l'uso di tale materiale, in tutto o in parte, nelle specifiche del COBOL. Tale autorizzazione si estende alla riproduzione e all'uso delle specifiche COBOL in manuali di programmazione o in pubblicazioni analoghe.*

*Qualsiasi organizzazione che intenda riprodurre il rapporto COBOL e le specifiche iniziali in tutto o in parte, usando idee ricavate da tale rapporto o utilizzando il rapporto stesso come elemento base per un manuale di istruzione o per qualsiasi altro scopo, è libera di farlo. Tuttavia, si richiede a tutte queste organizzazioni di riprodurre la presente sezione, come parte dell'introduzione. Coloro che invece utilizzano brevi citazioni, come nelle rassegne dei nuovi libri, sono pregati di citare la fonte ma non di riprodurre l'intera sezione.*»

Successivamente alla Conferenza sui linguaggi dei sistemi di elaborazione dati, CODASYL, il compito di definire lo standard del linguaggio COBOL è stato preso dall'istituto ANSI (*American national standards institute*), che chiede nuovamente di citare la fonte nei manuali di tale linguaggio. Il testo seguente è citato in lingua originale.

«*Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgement paragraphs in their entirety as part of the preface to any such publication (any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgement of the source, but need not quote the acknowledgement):*

*COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.*

*No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.*

*The authors and copyright holders of the copyrighted material used herein have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.*

*FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac++ I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM, FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.*»

Il linguaggio COBOL nasce nel 1959, come linguaggio standard per l'amministrazione degli uffici e il nome sta per *Common business oriented language*, descrivendo precisamente il suo obiettivo.

L'origine così lontana del linguaggio COBOL è responsabile della prolissità della sua sintassi e dei vincoli di forma nella struttura che il programma sorgente deve avere. Tuttavia, questo linguaggio è eccezionale nella gestione dei dati, avvicinandosi alle funzionalità di un sistema di gestione di basi di dati (ovvero un DBMS).



Il linguaggio COBOL è nato da un comitato di utenti e di produttori di elaboratori, con lo scopo di rimanere uniforme, il più possibile, su tutte le piattaforme. Generalmente si considera, correttamente, che il C rappresenti l'esempio di linguaggio di programmazione standard per definizione, ma i contesti sono differenti: il linguaggio C serve a consentire la migrazione di un sistema operativo da una macchina all'altra, mentre il linguaggio COBOL è fatto per consentire la migrazione di programmi applicativi su architetture fisiche e sistemi operativi differenti.

Il linguaggio COBOL è fatto per poter funzionare su sistemi operativi che possono anche essere privi di qualunque astrazione dell'hardware; pertanto, una porzione apposita nella struttura del sorgente è riservata alla dichiarazione delle unità fisiche per lo scambio dei dati (la divisione '**`ENVIRONMENT DIVISION`**'). Utilizzando il COBOL in un ambiente abbastanza evoluto, quanto può esserlo un sistema Unix, molte informazioni diventano inutili e implicite, ma il fatto che con questo linguaggio ci sia la possibilità di operare con la maggior parte degli elaboratori fabbricati dal 1959 in poi, lo rende speciale e degno di apprezzamento per lungo tempo.

Il linguaggio COBOL ha subito nel tempo diverse revisioni, indicate generalmente attraverso l'anno di edizione; un punto di riferimento abbastanza comune è lo standard del 1985, a cui questo manuale, in parte, cerca di adeguarsi. Per quanto riguarda l'ente ISO, è disponibile lo standard ISO 1989.

## 72.1 Caratteristiche del linguaggio

Il linguaggio COBOL si basa convenzionalmente sulla lingua inglese ed è composto sommariamente da ***parole***, ***istruzioni***, gruppi di istruzioni, ***paragrafi*** e ***sezioni***.

### 72.1.1 Organizzazione del programma in forma sorgente

Ogni programma COBOL deve contenere quattro divisioni, anche se queste dovessero essere vuote, rispettando l'ordine seguente:

1. '**`IDENTIFICATION DIVISION`**'
2. '**`ENVIRONMENT DIVISION`**'
3. '**`DATA DIVISION`**'
4. '**`PROCEDURE DIVISION`**'

La divisione '**`IDENTIFICATION DIVISION`**' serve a identificare il programma. Vi si possono includere informazioni generali, come il nome del programma stesso, la data di edizione, la data di compilazione, il nome dell'elaboratore per il quale è stato scritto e altre annotazioni.

La divisione '**`ENVIRONMENT DIVISION`**' specifica le apparecchiature usata e i file che servono al programma.

La divisione '**`DATA DIVISION`**' contiene la descrizione dei file e dei record relativi, creati o utilizzati dal programma, assieme a tutte le altre variabili e costanti che servono al programma.

La divisione '**`PROCEDURE DIVISION`**' specifica il procedimento elaborativo da applicare ai dati.

Le «azioni» descritte nel programma COBOL sono espresse in termini di istruzioni, che possono essere riunite in gruppi di istruzioni e poi in paragrafi.

### 72.1.2 Insieme dei caratteri

I compilatori tradizionali del linguaggio COBOL adottano, a seconda dei casi, il codice ASCII o il codice EBCDIC per la rappresentazione interna dei caratteri; inoltre, in un programma sorgente si può usare soltanto un insieme ristretto di simboli, con l'eccezione del contenuto delle costanti alfanumeriche, che invece è abbastanza libero.

Tabella 72.1. I simboli disponibili nel linguaggio, in generale.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| '0'..'9' | cifre numeriche | 'A'..'Z' | lettere maiuscole dell'alfabeto inglese (latino) |
| ' ' | spazio | | |
| '+' | segno più | '-' | segno meno o trattino |
| '*' | asterisco | '/' | barra obliqua |
| '$' | dollaro o segno di valuta | ',' | virgola |
| ';' | punto e virgola | '.' | punto fermo |
| '(' | parentesi aperta | ')' | parentesi chiusa |
| '<' | minore | '>' | maggiore |

Si osservi che il segno di valuta, rappresentato normalmente dal dollaro, può essere ridefinito e rappresentato da un altro simbolo.

Tabella 72.2. Caratteri usati per l'interpunzione.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| ' ' | spazio bianco | | |
| ',' | virgola | ';' | punto e virgola |
| '.' | punto fermo | '"' | virgolette |
| '(' | parentesi aperta | ')' | parentesi chiusa |

Tabella 72.3. Caratteri usati per formulare le parole.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| 'A'..'Z' | lettere alfabetiche maiuscole, senza accenti | '0'..'9' | cifre numeriche |
| '-' | trattino | | |

Tabella 72.4. Caratteri usati come operatori aritmetici.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| '+' | addizione | '-' | sottrazione |
| '*' | moltiplicazione | '/' | divisione |
| '(' | aperta parentesi | ')' | chiusa parentesi |

Tabella 72.5. Caratteri usati nelle relazioni.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| '=' | uguale a | | |
| '<' | minore di | '>' | maggiore di |

Si osservi che, al contrario di tanti altri linguaggi, nati però in momenti successivi, il COBOL non prevede l'uso del trattino basso ('_').

### 72.1.3 Struttura del linguaggio

Il testo di un programma sorgente COBOL è costruito con stringhe di caratteri e separatori, secondo le regole descritte nelle sezioni successive.

#### 72.1.3.1 Separatori

Un separatore è una stringa composta da uno o più caratteri di interpunzione, rispettando le regole seguenti. Si osservi che queste regole non si applicano al contenuto delle costanti non numeriche (le stringhe letterali) e naturalmente non si applicano ai commenti.

- La virgola e il punto e virgola sono separatori, tranne quando appaiono nel modello di definizione di una variabile ('**PICTURE**'), dove invece sono trattati come parte del modello stesso. La virgola e il punto e virgola, se usati come separatori, possono essere impiegati al posto dello spazio.
- Un punto fermo, seguito da uno spazio, è un separatore. Il punto fermo può apparire soltanto dove ciò è permesso esplicitamente dalle regole grammaticali del linguaggio.
- Le parentesi tonde, usate in coppia, aperta e chiusa, sono separatori. Possono essere usate per delimitare indici, espressioni aritmetiche e condizioni.



- Le virgolette sono separatori. Le virgolette di apertura devono essere precedute da uno spazio o da una parentesi aperta; le virgolette di chiusura devono essere seguite, alternativamente da: uno spazio, una virgola, un punto e virgola, un punto fermo oppure una parentesi chiusa.

  Le virgolette possono apparire solo in coppia, per delimitare costanti alfanumeriche, tranne quando le costanti continuano nella riga successiva.
- Lo spazio usato come separatore può precedere o seguire tutti gli altri separatori, tranne nei casi previsti dalle altre regole grammaticali del linguaggio. Uno spazio compreso tra una coppia di virgolette è una costante alfanumerica e non costituisce un separatore.

I caratteri di interpunzione che appaiono all'interno di un modello di definizione di una variabile ('**PICTURE**') o di una costante numerica, non sono considerati caratteri di interpunzione, piuttosto sono simboli usati per caratterizzare il modello relativo o la costante (le regole per la dichiarazione di un modello di definizione di una variabile sono descritte nella sezione 72.9).

I modelli di definizione delle variabili sono delimitati solo dallo spazio, dalla virgola, dal punto e virgola o dal punto fermo.

#### 72.1.3.2 Stringhe: «character-string»

Nei modelli sintattici, una stringa di caratteri (*character-string*) può essere: un carattere o una sequenza di caratteri contigui, che forma una parola per il linguaggio COBOL; il modello di definizione di una variabili ('**PICTURE**'); un commento. Una stringa di caratteri di questi contesti è delimitata da separatori.

#### 72.1.3.3 Parole

Una «parola» per il linguaggio COBOL è una stringa composta al massimo da 30 caratteri, che può essere:

- una parola definita dall'utente, ovvero *user-defined word*;
- un nome di sistema, ovvero *system-name*;
- una parola riservata, ovvero *reserved word*.

Le parole riservate o di sistema non possono essere utilizzate per fini diversi, pertanto non possono essere ridefinite dall'utente.

#### 72.1.3.4 Parole definite dall'utente

Una parola definita dall'utente è una parola COBOL che deve essere fornita per soddisfare la sintassi di un'istruzione. Tale parola può essere composta utilizzando soltanto le lettere alfabetiche maiuscole, le cifre numeriche e il trattino ('-'), tenendo conto che il trattino non può trovarsi all'inizio o alla fine di tali parole. Si osservi che in alcuni casi le parole sono costituite esclusivamente da cifre numeriche, mentre in tutti gli altri, le parole devono iniziare con una lettera alfabetica.

Tabella 72.6. Classificazione parziale delle parole definite dall'utente.

| Definizione tradizionale | Descrizione |
|---|---|
| *condition-name* | Il «nome di condizione» è un nome al quale viene assegnato un valore o un insieme di valori o un intervallo di valori, scelti fra tutti quelli che una variabile può assumere. La variabile stessa viene chiamata «variabile di condizione». I nomi di condizione vengono definiti nella divisione '**DATA DIVISION**'. Un nome di condizione può essere usato solo nelle espressioni condizionali, dove viene trattato come un'abbreviazione di una condizione di relazione. Il valore restituito dal nome di condizione è *Vero* se il valore della variabile di condizione associata è uguale a uno di quei valori che sono stati assegnati al nome di condizione. |
| *data-name* | Si tratta del nome di una variabile descritta nella divisione '**DATA DIVISION**'. Una variabile di questo tipo rappresenta normalmente un componente che non può essere suddiviso ulteriormente. |
| *file-name* | Si tratta del nome di un file descritto all'interno della divisione '**DATA DIVISION**' e può appartenere sia alla sezione '**FD**' (*File description*), sia alla sezione '**SD**' (*Sort description*). |
| *index-name* | Si tratta del nome di un indice associato a una certa tabella, usato per selezionare una voce dalla tabella stessa. Un nome di questo tipo si dichiara nella divisione '**DATA DIVISION**'. |
| *level-number* | Si tratta di un numero che indica la posizione nella struttura gerarchica di un record logico, oppure di un numero speciale che rappresenta convenzionalmente delle proprietà speciali di una variabile. Il numero di livello è espresso esclusivamente con una o due cifre numeriche; inoltre, i numeri che vanno da 01 a 49 indicano la posizione in un record, mentre i numeri 66, 77 e 88 identificano proprietà speciali. Normalmente il numero di livello si scrive sempre utilizzando due cifre, aggiungendo eventualmente uno zero iniziale. Il numero di livello si usa nella divisione '**DATA DIVISION**'. |
| *library-name* | Si tratta di un nome che serve a individuare una libreria di sorgenti COBOL, da usare per importare codice contenuto in altri file. |
| *mnemonic-name* | Si tratta di un nome che fa riferimento a qualcosa, che dipende dall'ambiente in cui si vuole compilare o eseguire il programma. Questo tipo di parole si usa nella divisione '**ENVIRONMENT DIVISION**', precisamente nel paragrafo '**SPECIAL-NAMES**', con lo scopo di poter sostituire facilmente tali associazioni, senza intervenire in altre parti del programma. |
| *paragraph-name* | Si tratta del nome che dichiara l'inizio di un paragrafo nella divisione '**PROCEDURE DIVISION**'. |
| *program-name* | Si tratta del nome del programma sorgente, come specificato nella divisione '**IDENTIFICATION DIVISION**'. |
| *record-name* | Si tratta del nome di un record di un file. Associando idealmente il file a una tabella di dati, il record equivale alla riga di tale tabella. La dichiarazione dei record avviene nella divisione '**DATA DIVISION**'. |
| *section-name* | Si tratta del nome che delimita l'inizio di una sezione nella divisione '**PROCEDURE DIVISION**'. |
| *text-name* | Si tratta del nome di identificazione di un componente all'interno della libreria di sorgenti. |

Tutte le parole definite dall'utente, a esclusione dei numeri di livello, possono appartenere soltanto a uno dei vari raggruppamenti previsti e devono essere uniche; tuttavia, in alcuni casi è prevista la possibilità di «qualificare» dei nomi, che non sono univoci, in modo da attribuirli al loro contesto preciso (sezione 72.8.3).

### 72.1.3.5 Parole riservate

Le parole riservate sono quelle parole del linguaggio che fanno parte di un elenco prestabilito e che hanno un significato speciale. Queste parole sono classificate in gruppi in base al loro utilizzo.



Tabella 72.7. Classificazione sintetica delle parole riservate.

| Classificazione | Descrizione |
|---|---|
| parole chiave | Una parola chiave è una parola riservata la cui presenza è richiesta all'interno di un'istruzione (al contrario di altre che possono essere usate soltanto per migliorare l'estetica o la leggibilità delle istruzioni). Le parole chiave devono essere inserite per specificare quel tipo di istruzione. |
| parole opzionali | Una parola opzionale è una parola riservata facoltativa, che si può usare nelle istruzioni per facilitarne la lettura. La parola opzionale non è obbligatoria, ma se usata va applicata secondo la sintassi prevista. |
| registri speciali | Una registro speciale identifica un'area di memoria con funzioni speciali. I registri speciali dipendono generalmente dalle caratteristiche del compilatore e non sono standard. |
| costanti figurative | Una costante figurativa è un nome che identifica un certo valore costante, come alternativa alla rappresentazione letterale. |
| parole di caratteri speciali | Alcuni caratteri speciali, nell'ambito del contesto appropriato, possono essere rappresentati attraverso parole particolari. Si tratta precisamente degli operatori di relazione. |

### 72.1.3.6 Costanti figurative

Per fare riferimento a valori costanti specifici si possono usare alcune parole riservate, note come costanti figurative. Di queste parole chiave esistono sia versioni al singolare, sia al plurale, ma rappresentano sempre la stessa cosa, ovvero un valore singolo o un valore ripetuto, in base al contesto.

Tabella 72.8. Costanti figurative.

| Nome | Descrizione |
|---|---|
| `ZERO`<br>`ZEROS`<br>`ZEROES` | Rappresenta il valore numerico zero o la stringa '**0**' ripetuta più volte. |
| `SPACE`<br>`SPACES` | Rappresenta uno o più spazi bianchi. |
| `HIGH-VALUE`<br>`HIGH-VALUES` | Rappresenta uno o più caratteri con un «valore massimo», in base a qualche criterio, legato alla sequenza di collazione (*collating sequence*) o alla codifica. Generalmente si tratta del valore $FF_{16}$. |
| `LOW-VALUE`<br>`LOW-VALUES` | Rappresenta uno o più caratteri con un «valore minimo», in base a qualche criterio, legato alla sequenza di collazione (*collating sequence*) o alla codifica. Generalmente si tratta del valore $00_{16}$. |
| `QUOTE`<br>`QUOTES` | Rappresenta una o più virgolette. Questa costante figurativa non può sostituire le virgolette che delimitano le costanti alfanumeriche. |
| `ALL` ***valore*** | Rappresenta la ripetizione indefinita del valore indicato. Tale valore può essere specificato anche attraverso una costante letterale o una costante figurativa. |

### 72.1.3.7 Parole di caratteri speciali

Gli operatori di relazione si possono rappresentare con i simboli previsti ('**<**', '**>**', e '**=**') oppure attraverso parole speciali, ovvero «parole di caratteri speciali», note come *special character word*. La tabella successiva riepiloga l'uso degli operatori di relazione, in tutte le loro forme.

Tabella 72.9. Modelli sintattici per l'uso degli operatori di relazione.

| Operatore | Descrizione |
|---|---|
| `IS [NOT] GREATER THEN` | maggiore di, non maggiore di |
| `IS [NOT] >` | maggiore di, non maggiore di |

| Operatore | Descrizione |
|---|---|
| `IS [NOT] LESS THEN` | minore di, non minore di |
| `IS [NOT] <` | minore di, non minore di |
| `IS [NOT] EQUAL TO` | uguale a, diverso da |
| `IS [NOT] =` | uguale a, diverso da |
| `IS GREATER THAN OR EQUAL TO` | maggiore o uguale a |
| `IS >=` | maggiore o uguale a |
| `IS LESS THAN OR EQUAL TO` | minore o uguale a |
| `IS <=` | minore o uguale a |

### 72.1.3.8 Rappresentazione delle costanti

Le costanti possono essere stringhe di caratteri, il cui valore è implicito in base ai caratteri di cui sono composte, oppure sono costanti figurative, che rappresentano un valore in base al significato verbale che hanno. **Una costante può essere di tipo numerico o alfanumerico e non sono previsti altri tipi.**

Una costante numerica letterale è una stringa composta da cifre numeriche ed eventualmente anche dai segni '**+**', '**-**' e dal punto per la separazione tra la parte intera e la parte decimale (a meno che il punto sia da sostituire con la virgola, avendone scambiato le funzionalità con un'istruzione apposita). Una costante numerica deve contenere almeno una cifra e ha una dimensione massima di cifre che dipende dal compilatore.

Una costante numerica non può contenere più di un segno. Se viene usato il segno, questo deve collocarsi nella posizione più a sinistra; se non appare alcun segno, il valore si intende positivo.

Una costante numerica non può contenere più di un punto decimale e può apparire in qualunque posizione. Se non viene usato il punto decimale, la costante rappresenta un numero intero.

Se nel paragrafo '**SPECIAL-NAMES**' della divisione '**ENVIRONMENT DIVISION**' è specificata la dichiarazione '**DECIMAL-POINT IS COMMA**', la rappresentazione dei valori numerici avviene scambiando il significato del punto e della virgola (in pratica secondo le convenzioni europee).

Una costante alfanumerica è una stringa di caratteri delimitata da virgolette. La stringa può contenere qualsiasi carattere previsto dalla codifica utilizzata dal compilatore; in generale è ammesso almeno l'uso delle lettere minuscole dell'alfabeto latino.

Per rappresentare le virgolette ('**"**') all'interno di una stringa si usa il concatenamento con la costante figurativa '**QUOTE**', come nell'esempio seguente:

```
000000 DISPLAY "Il file ", QUOTE, "mio.txt", QUOTE,
000000         " e' impegnato!".
```

Una costante alfanumerica deve contenere almeno un carattere all'interno delle virgolette. La lunghezza massima di un valore alfanumerico dipende dal compilatore, ma in generale dovrebbe essere garantita la rappresentazione di almeno 200 caratteri.

### 72.1.4 Notazione sintattica

I manuali COBOL adottano generalmente una forma particolare di notazione per descriverne la sintassi, a cui si adegua anche questo.

Nella sintassi le «parole chiave», secondo la definizione del COBOL, sono rappresentate sottolineate, a indicare la loro obbligatorietà, mentre le parole facoltative non sono sottolineate. Nell'esempio seguente, le parole '**IF**', '**NOT**', '**NUMERIC**' e '**ALPHABETIC**' sono parole chiave, mentre la parola '**IS**' è facoltativa:



```
                                /           \
                                | NUMERIC   |
IF identifier IS [NOT]          <           >
                                | ALPHABETIC |
                                \           /
```

Tutte le parole scritte con lettere minuscole rappresentano delle metavariabili sintattiche che devono essere espresse dal programmatore in quella posizione. Nell'esempio precedente appare una sola metavariabile denominata '**identifier**'.

Le parentesi graffe servono a rappresentare la scelta tra alternative differenti. Nell'esempio precedente si deve scegliere tra due parole chiave: '**NUMERIC**' o '**ALPHABETIC**'.

Le parentesi quadre rappresentano parti opzionali di un'istruzione; tuttavia si osservi che non si tratta di «parole facoltative», secondo la definizione del linguaggio COBOL, perché l'uso o meno di tali porzioni di codice implica un risultato differente dell'istruzione.

La presenza di tre punti consecutivi indica che i dati che precedono la notazione possono essere ripetuti successivamente, in funzione delle esigenze del problema che si intende risolvere.

```
MOVE identifier-1 TO identifier-2 ...
```

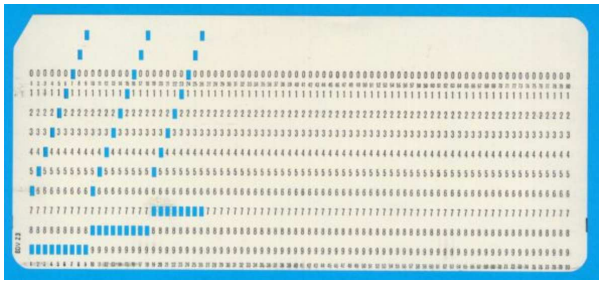
Nell'esempio mostrato, i puntini di sospensione indicano che si possono inserire più variabili (precisamente ciò che è rappresentato come ***identifier-2***). In questo caso, il contenuto della prima variabile viene copiato all'interno di tutte quelle che sono annotate dopo la parola chiave '**TO**'.

Quando appare il punto fermo nello schema sintattico, l'istruzione reale deve contenerlo nella stessa posizione relativa.

## 72.2 Modulo di programmazione

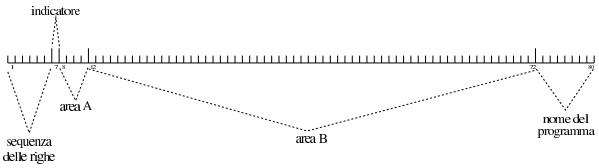
Il linguaggio COBOL nasce quando l'inserimento dei dati in un elaboratore avveniva principalmente attraverso schede perforate, pertanto, da questo derivano delle limitazioni nel modo in cui vanno scritte le sue istruzioni.

Figura 72.13. La scheda perforata classica, da 80 colonne.



Il modulo di programmazione (*coding form*) era un foglio quadrettato che conteneva la guida per la scrittura di un programma, da passare poi a una persona che si incaricava di perforare le schede, copiando il testo di tale modulo. Attualmente strumenti del genere non si usano più, tuttavia occorre sapere che le direttive vanno scritte in uno spazio di colonne prestabilito.

Figura 72.14. Suddivisione delle colonne.




In pratica, il codice COBOL si scrive in un file di testo di 80 colonne, rispettando le convenzioni descritte nella tabella successiva.

Tabella 72.15. Colonne riservate nel codice COBOL.

| Colonne | Utilizzo |
|---|---|
| 1..6 | Le prime sei colonne servono a indicare un numero di sequenza delle righe. Il numero può essere discontinuo, purché progressivo. Generalmente si utilizzava una sequenza maggiore dell'unità, per consentire l'inserzione successiva di righe ulteriori, che si sarebbero tradotte nell'aggiunta di schede, senza dover perforare nuovamente tutto. |
| 7 | La settima colonna serve a inserire un simbolo «indicatore». Generalmente si tratta dell'asterisco, per specificare che si tratta di una riga di commento, del trattino per la continuazione delle stringhe, oppure di una barra obliqua per richiedere un salto pagina in fase di stampa del sorgente. |
| 8..11 | Le colonne dall'ottava all'undicesima rappresentano l'«area A», nella quale devono iniziare le dichiarazioni più importanti. |
| 12..72 | Le colonne dalla dodicesima alla settantaduesima rappresentano l'«area B», nella quale si mettono tutte le direttive che non possono partire dall'area A. |
| 73..80 | Le ultime otto colonne sono riservate per inserire un'etichetta facoltativa di identificazione del programma. |

### 72.2.1 Indicatore

La settima colonna serve per diverse funzioni, distinte in base alla presenza di un simbolo speciale; se in questa colonna si trova uno spazio, la riga viene usata per le funzioni normali. La tabella successiva riepiloga i simboli che possono apparire nella settima colonna e come questi dovrebbero essere interpretati dal compilatore.

| Indicatore | Descrizione |
|---|---|
| `$` | Il dollaro viene usato per specificare delle opzioni in fase di compilazione. |
| `*` | L'asterisco stabilisce che, la riga in cui appare, contiene un commento che il compilatore deve ignorare completamente. Il commento può essere collocato solo nello spazio complessivo dell'area A e B, ovvero dalla colonna 8 alla colonna 72. |
| `/` | La barra obliqua serve a richiedere un salto pagina quando gli strumenti di compilazione vengono usati per stampare il sorgente. Ciò che dovesse apparire nell'area A e B di una riga che ha la barra obliqua nella settima colonna viene considerato come un commento. |
| `D` | La lettera '**D**' serve a indicare al compilatore che la riga in questione deve essere presa in considerazione solo se l'opzione '**WITH DEBUGGING**' viene utilizzata nel paragrafo '**SOURCE COMPUTER**'; in caso contrario la riga deve essere trattata come un commento. |
| `-` | Un trattino indica che, sulla riga precedente, l'ultima parola o costante non è completa, ma continua sulla riga in cui appare il trattino stesso. |

Per quanto riguarda la continuazione di parole e di costanti numeriche su più righe, il troncamento può avvenire in qualsiasi punto, mettendo un trattino nella settima colonna della riga successiva, continuando lì la parola o la costante, a partire dalla colonna 12 fino alla colonna 72 (area B). Gli spazi finali nella riga interrotta e quelli iniziali della riga che riprende, vengono ignorati.

Le costanti alfanumeriche delimitate da virgolette, si separano in modo differente. Sulla riga spezzata, si considerano tutte le informazioni dalle virgolette di apertura fino alla colonna 72 inclusa, mentre nella riga successiva, la costante deve riprendere aggiungendo altre virgolette di apertura.



Si osservi che ci sono compilatori che si limitano a riconosce solo l'asterisco per i commenti, ignorando tutto il resto. Per questo motivo, è bene evitare l'uso di ogni altro simbolo in questa colonna, quando si vuole scrivere un programma abbastanza compatibile, tenendo conto che si può evitare la continuazione nella riga successiva, perché le istruzioni possono collocarsi su più righe senza spezzare le parole, mentre le costanti alfanumeriche si possono dividere in porzioni più piccole da concatenare.

### 72.2.2 Area A e area B

Le intestazioni dei paragrafi, delle sezioni e delle divisioni devono iniziare nell'area A. L'intestazione di una divisione consiste nel nome della divisione ('**IDENTIFICATION**', '**ENVIRONMENT**', '**DATA**' o '**PROCEDURE**'), seguito da uno spazio bianco e dalla parola '**DIVISION**', seguita a sua volta da un punto fermo. L'intestazione di una sezione consiste di un nome di sezione seguito da uno spazio bianco e dalla parola '**SECTION**', seguita a sua volta da un punto fermo. L'intestazione di un paragrafo consiste di un nome di paragrafo seguito da un punto fermo e da uno spazio bianco; il primo gruppo di istruzioni del paragrafo può apparire anche sulla stessa riga.

All'interno delle divisioni '**IDENTIFICATION DIVISION**' e '**ENVIRONMENT DIVISION**', le sezioni e i paragrafi sono fissi e sono ammessi solo i nomi previsti espressamente, mentre nella divisione '**PROCEDURE DIVISION**' i nomi dei paragrafi e delle sezioni sono stabiliti liberamente.

All'interno della divisione '**DATA DIVISION**', le sezioni '**FD**' e '**SD**', così come i numeri di livello 01 e 77, devono iniziare nell'area A, mentre gli altri numeri di livello devono iniziare nell'area B.

Nell'area B inizia tutto quello che non può iniziare nell'area A.

### 72.2.3 Interpunzione

La scrittura di un programma COBOL è sottoposta alle regole seguenti che riguardano l'uso dei caratteri di interpunzione.

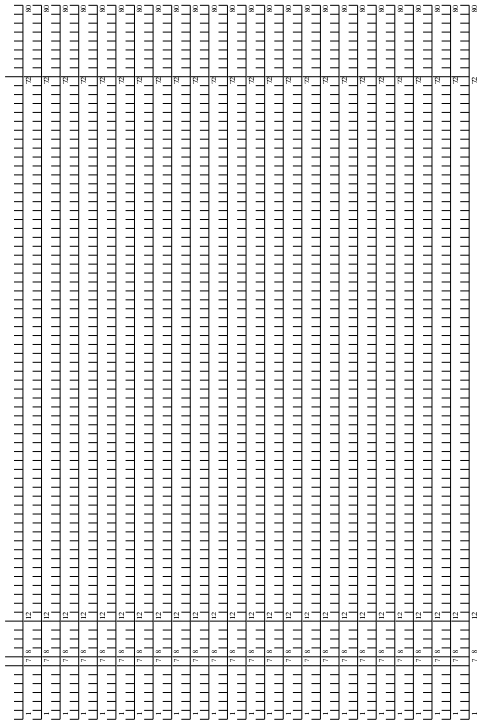
- Un gruppo di istruzioni termina con un punto seguito da uno spazio bianco. Un punto può apparire in un'altra posizione solo se fa parte di una costante alfanumerica, se si tratta del punto decimale di una costante numerica o se viene usato in un modello di definizione di una variabile ('**PICTURE**').
- Una virgola può essere usata fra le istruzioni per facilitare la leggibilità del programma; diversamente, una virgola può apparire solo dove indicato nello schema sintattico. **L'uso delle virgole non è obbligatorio.**
- Il punto e virgola può essere usato al posto della virgola.
- Uno spazio delimita sempre una parola o una costante, a meno che tale spazio sia parte di una costante alfanumerica. Lo spazio inteso come delimitatore può essere ridondante; inoltre, quando il testo di un'istruzione termina esattamente alla fine dell'area B (colonna 72), lo spazio successivo viene a mancare.

Listato 72.17. Un esempio preso da un programma sorgente, dove si mette in evidenza come si continua una parola o una costante alfanumerica nella riga successiva.

```
000100 IDENTIFICATION DIVISION.                                         ESEMPIO0
000200                                                                  ESEMPIO0
000300 PROGRAM-ID.   ESEMPIO0.                                          ESEMPIO0
000400 AUTHOR.       DANIELE GIACOMINI.                                 ESEMPIO0
000500 DATE-WRITTEN. 2005-02-13.                                        ESEMPIO0
000600                                                                  ESEMPIO0
000700                                                                  ESEMPIO0
000800 ENVIRONMENT DIVISION.                                            ESEMPIO0
000900                                                                  ESEMPIO0
001000 INPUT-OUTPUT SECTION.                                            ESEMPIO0
001100                                                                  ESEMPIO0
001200 FILE-CONTROL.                                                    ESEMPIO0
001300                                                                  ESEMPIO0
001400     SELECT FILE-DA-LEGGERE ASSIGN TO DISK                        ESEMPIO0
001500                            ORGANIZATION IS SEQUENTIAL.          ESEMPIO0
001600                                                                  ESEMPIO0
001700 DATA DIVISION.                                                   ESEMPIO0
001800                                                                  ESEMPIO0
```

```
001900 FILE SECTION.                                                    ESEMPIO0
002000                                                                  ESEMPIO0
002100 FD  FILE-DA-LEGGERE                                              ESEMPIO0
002200     LABEL RECORD IS STANDARD                                     ESEMPIO0
002300     VALUE OF FILE-ID IS "input.txt".                             ESEMPIO0
002400                                                                  ESEMPIO0
002500 01  RECORD-DA-LEGGERE PIC X(30).                                ESEMPIO0
002600                                                                  ESEMPIO0
002700 WORKING-STORAGE SECTION.                                         ESEMPIO0
002800 01  EOF               PIC 9   VALUE ZERO.                       ESEMPIO0
002900/                                                                 ESEMPIO0
003000 PROCEDURE DIVISION.                                              ESEMPIO0
003100*                                                                 ESEMPIO0
003200* Qui inizia il paragrafo "MAIN".                                 ESEMPIO0
003300*                                                                 ESEMPIO0
003400 MAIN.                                                            ESEMPIO0
003500     OPEN INPUT FILE-DA-LEG                                       ESEMPIO0
003600-               GERE.                                             ESEMPIO0
003700     READ FILE-DA-LEG                                             ESEMPIO0
003800-         GERE                                                    ESEMPIO0
003900          AT END                                                  ESEMPIO0
004000               MOVE 1 TO EOF.                                     ESEMPIO0
004100     DISPLAY "Ho aperto il file input.txt e sto per emettere il suESEMPIO0
004200-            "o contenuto sullo schermo:".                        ESEMPIO0
004300     PERFORM LETTURA UNTIL EOF = 1.                               ESEMPIO0
004400     CLOSE FILE-DA-LEGGERE.                                       ESEMPIO0
004500                                                                  ESEMPIO0
004600     STOP RUN.                                                    ESEMPIO0
004700*                                                                 ESEMPIO0
004800* Qui inizia un altro paragrafo.                                  ESEMPIO0
004900*                                                                 ESEMPIO0
005000 LETTURA.                                                         ESEMPIO0
005100     DISPLAY RECORD-DA-LEGGERE.                                   ESEMPIO0
005200     READ FILE-DA-LEGGERE                                         ESEMPIO0
005300          AT END                                                  ESEMPIO0
005400               MOVE 1 TO EOF.                                     ESEMPIO0
005500/                                                                 ESEMPIO0
```

Figura 72.18. Esempio di un modulo di programmazione COBOL.



## 72.3 Divisione «IDENTIFICATION DIVISION»

La divisione '**IDENTIFICATION DIVISION**' costituisce la prima parte di un programma COBOL. Il suo scopo è quello di contenere delle informazioni sul programma, secondo una classificazione ben stabilita. Le informazioni tipiche che si inseriscono in questa divisione sono il nome del programma (nome che non coincide necessariamente con il nome del file che contiene il sorgente), il nome dell'autore, la data di scrittura del programma, la data di compilazione.



### 72.3.1 Struttura

La struttura della divisione '**IDENTIFICATION DIVISION**' è sintetizzabile nello schema sintattico seguente:

```
 IDENTIFICATION DIVISION.

[PROGRAM-ID. program-name].

[AUTHOR. [comment-entry]...].

[INSTALLATION. [comment-entry]...].

[DATE-WRITTEN. [comment-entry]...].

[DATE-COMPILED. [comment-entry]...].

[SECURITY. [comment-entry]...].
```

La divisione deve iniziare scrivendo '**IDENTIFICATION DIVISION**' a partire dall'area A, ricordando di aggiungere il punto fermo finale.

Tutti i nomi di paragrafo di questa divisione devono iniziare nell'area A e devono terminare con un punto fermo.

Il nome del programma (***program-name***) deve essere una parola COBOL e serve a identificare il programma sorgente, ma non corrisponde necessariamente al nome del file su disco che contiene il sorgente.

Le voci di commento (***comment-entry***), secondo lo schema sintattico, possono essere costituite da una sequenza qualunque di caratteri e possono occupare anche più righe, senza bisogno di indicare il simbolo di continuazione nella settima colonna, avendo cura però di utilizzare per tali voci solo l'area B e di terminarle comunque con un punto fermo.

La data di compilazione è, o dovrebbe essere, posta automaticamente dal compilatore, quando è prevista la stampa del sorgente da parte di questo strumento.

A parte il caso della data di compilazione, che dovrebbe essere fornita dal compilatore, tutte le altre informazioni rimangono invariate.

### 72.3.2 Codifica della divisione

Il listato successivo dà un'idea di come può essere codificata la divisione '**IDENTIFICATION DIVISION**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PROVA-INTESTAZIONE.
000300 AUTHOR.        DANIELE GIACOMINI.
000400 INSTALLATION.  NANOLINUX IV,
000500                TINYCOBOL 0.61,
000600                OPENCOBOL 0.31.
000700 DATE-WRITTEN.  2005-02-14.
000800 DATE-COMPILED.
000900 SECURITY.      SEGRETISSIMO, LIVELLO III.
001000*
001100 ENVIRONMENT DIVISION.
001200 DATA DIVISION.
001300 PROCEDURE DIVISION.
001400 MAIN.
001500     DISPLAY "CIAO A TUTTI!".
001600     STOP RUN.
```

## 72.4 Divisione «ENVIRONMENT DIVISION»

La divisione '**ENVIRONMENT DIVISION**' costituisce la seconda parte di un programma COBOL. La divisione si compone di due sezioni: '**CONFIGURATION SECTION**' e '**INPUT-OUTPUT SECTION**'.

La sezione '**CONFIGURATION SECTION**' serve per indicare delle informazioni relative all'elaboratore usato per la compilazione del programma sorgente e a quello nel quale deve essere eseguito il programma, una volta compilato; inoltre, questa sezione permette di

stabilire delle sostituzioni, come nel caso della virgola al posto del punto per separare la parte intera di un numero dalla parte decimale.

La sezione '**INPUT-OUTPUT SECTION**' serve per associare i file usati dal programma con le unità fisiche relative, a indicare le caratteristiche di tali file e a stabilire altri aspetti dello scambio di dati.

### 72.4.1 Struttura

La struttura della divisione '**ENVIRONMENT DIVISION**' è sintetizzabile nello schema sintattico seguente:

```
   ENVIRONMENT DIVISION.
.--                                      --.
|  CONFIGURATION SECTION.                  |
|                                          |
| [SOURCE-COMPUTER. source-computer-entry]. |
|                                          |
| [OBJECT-COMPUTER. object-computer-entry]. |
|                                          |
| [SPECIAL-NAMES. special-names-entry].    |
'--                                      --'
.--                                         --.
|  INPUT-OUTPUT SECTION.                      |
|                                             |
|  FILE-CONTROL. file-control-entry...        |
|                                             |
| [I-O-CONTROL. input-output-control-entry...]. |
'--                                         --'
```

### 72.4.2 Sezione «CONFIGURATION SECTION»

La sezione '**CONFIGURATION SECTION**' contiene le informazioni sul sistema usato per la compilazione del programma (nel paragrafo '**SOURCE-COMPUTER**'), il sistema nel quale il programma deve essere eseguito (nel paragrafo '**OBJECT-COMPUTER**') e il paragrafo '**SPECIAL-NAMES**' che consente di effettuare alcune sostituzioni a dei valori che altrimenti resterebbero al loro stato predefinito.

```
 CONFIGURATION SECTION.

[SOURCE-COMPUTER. source-computer-entry].

[OBJECT-COMPUTER. object-computer-entry].

[SPECIAL-NAMES. special-names-entry].
```

#### 72.4.2.1 Paragrafo «SOURCE-COMPUTER»

Il paragrafo '**SOURCE-COMPUTER**' identifica l'elaboratore presso il quale si intende compilare il programma. Si utilizza secondo lo schema sintattico seguente:

```
SOURCE-COMPUTER. computer-name [WITH DEBUGGING MODE].
```

Al posto della metavariabile ***computer-name*** deve essere indicata una parola COBOL, che serve solamente a titolo informativo nel sorgente. Se si specifica l'opzione '**DEBUGGING MODE**' si richiede al compilatore di prendere in considerazione, nel sorgente, tutte le righe annotate con la lettera '**D**' nella settima colonna e le istruzioni '**USE FOR DEBUGGING**', che altrimenti verrebbero semplicemente ignorate.

#### 72.4.2.2 Paragrafo «OBJECT-COMPUTER»

Il paragrafo '**OBJECT COMPUTER**' identifica l'elaboratore presso il quale deve essere utilizzato il programma, una volta compilato. Lo schema sintattico per l'utilizzo di questo paragrafo è quello seguente:

```
OBJECT-COMPUTER. computer-name... .
```

Il nome dell'elaboratore (*computer name*) deve essere una parola COBOL e ha un significato puramente informativo. Alla fine dell'indicazione dell'ultimo nome, deve apparire un punto fermo.



#### 72.4.2.3 Paragrafo «SPECIAL-NAMES»

Il paragrafo '**SPECIAL-NAMES**' serve ad associare un valore a dei nomi prestabiliti, quando si vuole che la funzione loro associata sia diversa da quella predefinita, oppure ad attribuire un «nome mnemonico» a un nome usato dal compilatore, che però non fa parte dello standard. Le dichiarazioni che possono apparire in questo paragrafo dipendono molto dalle caratteristiche del compilatore; quello che si vede nello schema sintattico seguente è il minimo che dovrebbe essere disponibile nella maggior parte dei casi:

```
SPECIAL-NAMES.

    implementor-name IS mnemonic-name

    [CURRENCY SIGN IS literal]

    [DECIMAL-POINT IS COMMA].
```

Si utilizza la dichiarazione '**CURRENTY SIGN IS**' per fissare il simbolo predefinito da usare come segno di valuta; si usa la dichiarazione '**DECIMAL-POINT IS COMMA**' per rappresentare i valori numerici secondo la forma europea, dove la virgola indica la separazione tra la parte intera e quella decimale.

Il segno di valuta può essere costituito da un solo carattere e sono molto pochi i simboli che si possono usare. Per la precisione, sono esclusi tutti i simboli che invece possono essere usati nei modelli di definizione delle variabili oltre a quelli che si usano come delimitatori. In linea di massima sono da escludere: tutte le cifre numeriche (da '**0**' a '**9**'); lo spazio; le lettere alfabetiche '**A**', '**B**', '**C**', '**D**', '**J**', '**L**', '**N**', '**P**', '**R**', '**S**', '**V**', '**X**', '**Z**'; i caratteri speciali '*****', '**+**', '**-**', '**,**', '**.**', '**;**', '**%**', '**(**', '**)**', '**"**', '**?**'.

Si osservi che anche nel modello di definizione di una variabile ('**PICTURE**'), quando si usa la dichiarazione '**DECIMAL-POINT IS COMMA**', il punto e la virgola si scambiano i ruoli.

L'esempio seguente mostra un pezzo di programma in cui si vede l'uso di queste opzioni. Per la precisione, si assegna la lettera «E» per rappresentare la valuta:

```
000000 ENVIRONMENT DIVISION.
000000 CONFIGURATION SECTION.
000000 SPECIAL-NAMES. DECIMAL-POINT IS COMMA
000000                CURRENCY SIGN IS "E".
```

L'attribuzione di un nome mnemonico a una parola non standard che però fa parte delle funzionalità specifiche del compilatore utilizzato, consente di limitare a questa sezione le modifiche per l'adattamento del programma a un compilatore che ha funzioni simili, ma descritte da parole diverse. Nell'esempio seguente, compilabile con OpenCOBOL, si sostituisce la parola '**CONSOLE**' con '**STANDARD-INPUT**', per identificare la fonte dei dati in ingresso per l'istruzione '**ACCEPT**':

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-ACCEPT.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-27.
000500*
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SOURCE-COMPUTER.
000900     OPENCOBOL.
001000 SPECIAL-NAMES.
001100     CONSOLE IS STANDARD-INPUT.
001200*
001300 DATA DIVISION.
001400*
001500 WORKING-STORAGE SECTION.
001600 77  MESSAGGIO            PIC X(30).
001700*
001800 PROCEDURE DIVISION.
```

```
001900*
002000 MAIN.
002100     DISPLAY "INSERISCI IL MESSAGGIO".
002200     ACCEPT MESSAGGIO FROM STANDARD-INPUT.
002300     DISPLAY "HAI INSERITO: ", MESSAGGIO.
002400*
002500     STOP RUN.
002600*
```

Nell'esempio appena mostrato sono evidenziate le righe più importanti per la comprensione del meccanismo; si può comprendere che l'istruzione '**ACCEPT**' avrebbe potuto essere scritta semplicemente così:

```
002200     ACCEPT MESSAGGIO FROM CONSOLE.
```

Tuttavia, avendo utilizzato il nome mnemonico '**STANDARD-INPUT**', se con un altro compilatore la console fosse identificata dalla sigla '**SPO**' (*Supervisory printer output*, come avveniva nel COBOL CMS (*Computer management system* della Burroughs negli anni 1980), basterebbe modificare la dichiarazione iniziale:

```
001000 SPECIAL-NAMES.
001100     SPO IS STANDARD-INPUT.
```

Per chiarezza, è il caso di sottolineare che '**STANDARD-INPUT**' ha valore per il compilatore solo in quanto viene dichiarato come nome mnemonico, dal momento che il linguaggio, nella sua veste ufficiale, non prevede la gestione dei flussi standard dei sistemi Unix.

### 72.4.3 Sezione «INPUT-OUTPUT SECTION»

La sezione '**INPUT-OUTPUT SECTION**' si suddivide in due paragrafi: '**FILE-CONTROL**' e '**I-O-CONTROL**'. Il paragrafo '**FILE-CONTROL**' specifica l'organizzazione e l'accesso dei file che vengono usati dal programma e le informazioni correlate a tali file; il paragrafo '**I-O-CONTROL**' serve a specificare informazioni aggiuntive sui file già dichiarati nell'altro paragrafo.

```
INPUT-OUTPUT SECTION.

FILE-CONTROL. file-control-entry...

[I-O-CONTROL. input-output-control-entry...].
```

#### 72.4.3.1 Paragrafo «FILE-CONTROL»

Il paragrafo '**FILE-CONTROL**' serve a dichiarare i file utilizzati dal programma e a definire alcune loro caratteristiche. Tutti i file dichiarati nel paragrafo '**FILE-CONTROL**' devono essere descritti nella divisione '**DATA DIVISION**'; nello stesso modo, tutti i file descritti nella divisione '**DATA DIVISION**', devono essere dichiarati nel paragrafo '**FILE-CONTROL**'.

Il linguaggio COBOL prevede una gestione dei file molto sofisticata, anche se non è detto che i compilatori mettano a disposizione sempre tutte le funzionalità standard. Si distinguono generalmente i tipi, in base alla loro «organizzazione», come sintetizzato nella tabella successiva.

Per il linguaggio COBOL i file sono sempre composti da record, pertanto l'accesso a un file si riferisce sempre a dei record.

Tabella 72.31. Classificazione dei file in base all'organizzazione.

| Organizzazione | Descrizione |
|---|---|
| sequenziale | Il file sequenziale consente un accesso ai record in modo seriale, dal primo all'ultimo. Generalmente, si dichiara un accesso sequenziale ai file quando l'unità di memorizzazione nella quale sono memorizzati è per sua natura sequenziale, come per i nastri magnetici. |
| relativa (*relative*) | Si tratta di un file ad accesso diretto, dove i record si possono raggiungere specificandone il numero, a partire da uno, avendo anche la possibilità di richiedere qualche spostamento relativo rispetto al record attuale. |



| Organizzazione | Descrizione |
|---|---|
| a indice | Si tratta di un file associato a un indice dei record. Attraverso l'indice è possibile raggiungere direttamente i record associati, senza bisogno di eseguire delle scansioni di ricerca. |

L'organizzazione del file definisce le potenzialità di accesso, ma in generale sono disponibili diverse varianti nel modo particolare di accedere ai record.

Il paragrafo '**FILE CONTROL**' si articola in dichiarazioni '**SELECT**', una per ogni file, secondo lo schema sintattico sintetico seguente:

```
FILE-CONTROL.

    SELECT file-name ASSIGN TO hardware-name [altre-opzioni].

    ...
```

Il modo in cui l'istruzione '**SELECT**' si articola, dipende dall'organizzazione del file e dal metodo di accesso specifico che si vuole attuare sullo stesso. Nella logica originale del linguaggio, in questa fase non viene ancora indicato il nome del file reale, secondo il sistema operativo, perché generalmente per questa informazione si agisce nella divisione '**DATA DIVISION**'; tuttavia, spesso il compilatore permette, o richiede, di specificare il nome del file reale proprio nell'istruzione '**SELECT**'.

#### 72.4.3.2 File fisici e file «logici»

L'organizzazione di un file è una caratteristica immutabile, che stabilisce, oltre che le potenzialità di accesso, anche la sua forma fisica «reale», ovvero quella che viene gestita attraverso l'astrazione del sistema operativo.

L'organizzazione sequenziale è quella più semplice, dove normalmente i record logici del linguaggio corrispondono esattamente al contenuto del file fisico che li contiene.

L'organizzazione relativa richiede la capacità di abbinare delle informazioni ai record logici, per esempio per poter annotare che un record è stato cancellato. Per fare questo, il compilatore può inserire tutte le informazioni necessarie in un file solo, oppure può avvalersi di due file reali: uno per i dati, l'altro per le informazioni sui record.

L'organizzazione a indice richiede tutte le funzionalità di quella relativa, con l'aggiunta di poter gestire l'accesso in base a una o più chiavi. Nei compilatori COBOL attuali, è molto probabile che tutte le informazioni necessarie vengano gestite in un file fisico soltanto, ma originariamente era frequente l'uso di un file per i dati e di altri file per le chiavi (uno per ogni chiave).

In base a questa premessa, si deve intendere che un file che viene creato con una certa organizzazione, può essere usato solo con quella; inoltre, si può contare sul fatto che un file creato con un programma realizzato attraverso un certo compilatore COBOL, non può essere utilizzato con un programma generato con un altro.

Di fronte a questo problema di compatibilità dei dati, i file organizzati in modo sequenziale sono sempre l'unica garanzia per un trasferimento dei dati. D'altra parte, negli anni in cui il linguaggio COBOL aveva il suo massimo splendore, i nastri magnetici rappresentavano l'unità di memorizzazione «standard» tra le varie architetture proprietarie.

#### 72.4.3.3 Istruzione «SELECT» per i file sequenziali

Lo schema sintattico semplificato per l'istruzione '**SELECT**', da usare nel paragrafo '**FILE-CONTROL**', per dichiarare un file sequenziale è quello che si può vedere nella figura successiva:

```
                          /                  \
                          |   hardware-name   |
SELECT file-name ASSIGN TO <                   >
                          | literal-file-name |
                          \                  /

    .--               .--   --. --.
    |                 | AREA  |   |
    | RESERVE integer |       |   |
    |                 | AREAS |   |
    '--               '--   --' --'

    [ ORGANIZATION IS [LINE] SEQUENTIAL ]

    [ ACCESS MODE IS SEQUENTIAL ]

    [ FILE STATUS IS data-name ].
```

Il file sequenziale può essere letto o scritto soltanto in modo sequenziale, a partire dall'inizio. Se l'unità di memorizzazione che lo contiene è sequenziale per sua natura, come avviene per un nastro o un lettore di schede perforate, si può avere solo una fase di lettura o una fase di scrittura, senza la possibilità di mescolare le due operazioni, mentre se si dispone di un'unità di memorizzazione ad accesso diretto, come nel caso di un disco, si può leggere e poi sovrascrivere lo stesso record.

Nello schema sintattico, la metavariabile ***file-name*** deve essere sostituita con il nome che si vuole attribuire al file nell'ambito del programma (non si tratta del nome che questo ha eventualmente per il sistema operativo). La metavariabile ***hardware-name*** va sostituita con un nome che serve a identificare l'unità di memorizzazione che contiene il file; questo nome dipende dal compilatore ma generalmente si mette '**DISK**' per indicare un file su disco. Altri nomi per la metavariabile ***hardware-name*** potrebbero essere: '**TAPE**', '**PRINTER**', '**PUNCH**', '**READER**' (gli ultimi due sarebbero un perforatore e un lettore di schede).

Il linguaggio COBOL è fatto per poter essere adattato a sistemi operativi molto diversi. In un sistema Unix, l'accesso alle unità di memorizzazione avviene attraverso dei file di dispositivo, pertanto, a seconda del compilatore, potrebbe anche essere superfluo dichiarare il tipo di unità di memorizzazione in questo modo, anche se in passato il linguaggio obbligava a farlo. Proprio per questo motivo, ci sono compilatori che, al posto di indicare il tipo di unità fisica attraverso un nome prestabilito, richiedono di mettere subito il percorso del file a cui si vuole fare riferimento, nonostante il linguaggio preveda per questo una dichiarazione separata nella divisione '**DATA DIVISION**'. In questo senso, nello schema sintattico appare la possibilità di indicare una stringa alfanumerica con il percorso del file (***literal-file-name***).

Nella dichiarazione '**RESERVE** ***integer***', la metavariabile ***integer*** rappresenta un numero intero di record da usare come memoria tampone. Se non si usa questa dichiarazione che, come si vede dallo schema sintattico, è facoltativa, viene usata la dimensione predefinita.

La dichiarazione '**ORGANIZATION IS SEQUENTIAL**' è facoltativa e sottintesa; tuttavia va osservato il significato che assume quando si aggiunge la parola '**LINE**'. In generale, il linguaggio COBOL considera i file come composti da record di dimensione uniforme. Quando però si vuole lavorare con i file di testo, le righe di questi file sono suddivise in base alla presenza del codice di interruzione di riga (che può cambiare da un sistema operativo all'altro). Volendo considerare in COBOL le righe di un file di testo pari a dei record di dimensione variabile, occorre aggiungere l'opzione '**LINE**', così da chiarire che si tratta sì di un'organizzazione sequenziale, ma di un file suddiviso in «righe».

La dichiarazione '**ACCESS MODE IS SEQUENTIAL**' è facoltativa, perché l'accesso a un file organizzato in modo sequenziale può essere solo sequenziale.



La dichiarazione '**FILE STATUS IS** ***data-name***' consente di indicare una variabile (da specificare nella sezione '**WORKING-STORAGE SECTION**' della divisione '**DATA DIVISION**') da usare eventualmente per conoscere lo stato dell'ultima operazione svolta sul file. Questa variabile deve poter rappresentare un valore di due caratteri (il modello di definizione della variabile deve essere '**XX**') e quando contiene il valore zero indica che l'ultima operazione è stata eseguita con successo (si vedano le tabelle 72.48 e 72.49, che appaiono alla fine del capitolo).

Il punto fermo che conclude l'istruzione '**SELECT**' appare una volta sola, alla fine; tutta l'istruzione deve risiedere nell'area B.

Viene mostrato un esempio completo di un programma COBOL che legge un file sequenziale:

Listato 72.34. Programma elementare che legge un file sequenziale.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ESEMPIO-SEQUENZIALE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               TINYCOBOL 0.61.
000600 DATE-WRITTEN. 2005-02-16.
000700 ENVIRONMENT DIVISION.
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100     SELECT FILE-NAME ASSIGN TO DISK
001200                      ORGANIZATION IS SEQUENTIAL
001300                      ACCESS MODE IS SEQUENTIAL
001400                      FILE STATUS IS DATA-NAME.
001500*
001600 DATA DIVISION.
001700 FILE SECTION.
001800 FD  FILE-NAME
001900     LABEL RECORD IS STANDARD
002000     VALUE OF FILE-ID IS "input.seq".
002100 01  RECORD-NAME  PIC X(20).
002200 WORKING-STORAGE SECTION.
002300 01  DATA-NAME    PIC XX.
002400 PROCEDURE DIVISION.
002500 MAIN.
002600     OPEN INPUT  FILE-NAME.
002700     DISPLAY "FILE STATUS: ", DATA-NAME.
002800     PERFORM READ-FILE UNTIL DATA-NAME NOT = ZERO.
002900     CLOSE FILE-NAME.
003000     STOP RUN.
003100 READ-FILE.
003200     READ FILE-NAME.
003300     DISPLAY "FILE STATUS: " DATA-NAME, " RECORD: ",
003400             RECORD-NAME.
```

Il file indicato come '**FILE-NAME**' è associato in pratica al file '`input.seq`'. Si può supporre che questo file abbia il contenuto seguente, senza alcun codice di interruzione di riga:

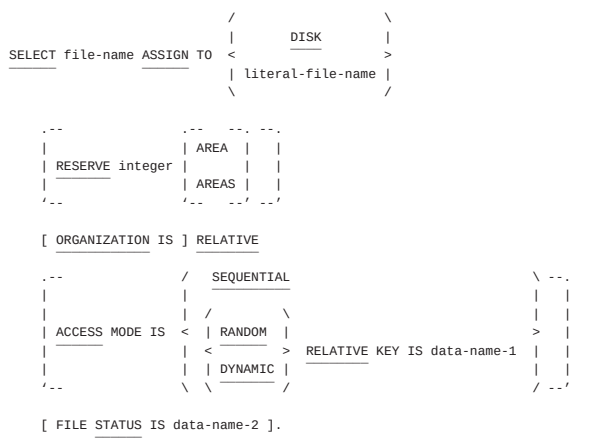
```
aaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbb↩
↪ccccccccccccccccccddddddddddddddddddddd
```

Eseguendo il programma dell'esempio si potrebbe ottenere il testo seguente attraverso lo schermo:

```
FILE STATUS: 00
FILE STATUS: 00 RECORD: aaaaaaaaaaaaaaaaaaaa
FILE STATUS: 00 RECORD: bbbbbbbbbbbbbbbbbbbb
FILE STATUS: 00 RECORD: ccccccccccccccccccc
FILE STATUS: 00 RECORD: ddddddddddddddddddd
FILE STATUS: 10 RECORD: ddddddddddddddddddd
```

### 72.4.3.4 Istruzione «SELECT» per i file relativi

Lo schema sintattico semplificato per l'istruzione '**SELECT**', da usare nel paragrafo '**FILE-CONTROL**', per dichiarare un file organizzato in modo «relativo» è quello che si può vedere nella figura successiva:




Il file organizzato in modo relativo può essere utilizzato secondo un accesso sequenziale, oppure facendo riferimento ai record per numero, considerando che il primo ha proprio il numero uno. Quando si individuano i record per numero, si distinguono due modalità di accesso: diretto (‘**RANDOM**’) e dinamico (‘**DYNAMIC**’). L’accesso diretto richiede che per ogni operazione l’indicazione del numero del record a cui si vuole fare riferimento, mentre con l’accesso dinamico è anche possibile eseguire delle operazioni di lettura sequenziali (‘**READ NEXT**’).

L’organizzazione relativa, oltre alle operazioni di lettura e scrittura, prevede la cancellazione dei record, che comunque possono essere rimpiazzati successivamente attraverso un’operazione di scrittura. Si osservi comunque che un record che risulta essere stato cancellato, non può essere letto.

Osservando lo schema sintattico si può intuire che la prima parte dell’istruzione ‘**SELECT**’ funzioni nello stesso modo di un file organizzato sequenzialmente; la differenza più importante riguarda la definizione del tipo di unità di memorizzazione che, date le caratteristiche dei file organizzati in modo relativo, deve consentire un accesso diretto ai dati.

La dichiarazione ‘**RESERVE** ***integer***’ si usa nello stesso modo del file organizzato sequenzialmente.

L’indicazione dell’organizzazione, attraverso la dichiarazione ‘**ORGANIZATION IS RELATIVE**’ è obbligatoria, anche se probabilmente è sufficiente scrivere soltanto ‘**RELATIVE**’.

Se non viene specificata la dichiarazione ‘**ACCESS MODE**’, si intende che l’accesso debba avvenire in modo sequenziale, altrimenti vale quanto indicato espressamente. Se l’accesso richiesto è diretto o dinamico, è necessario indicare quale variabile usare per specificare il numero del record, nella posizione occupata nello schema sintattico dalla metavariabile ***data-name-1*** (da specificare ulteriormente nella sezione ‘**WORKING-STORAGE SECTION**’ della divisione ‘**DATA DIVISION**’).

La dichiarazione ‘**FILE STATUS IS** ***data-name-2***’ funziona nello stesso modo descritto a proposito dei file organizzati in modo sequenziale.

Il punto fermo che conclude l’istruzione ‘**SELECT**’ appare una volta sola, alla fine; tutta l’istruzione deve risiedere nell’area B.

Viene mostrato un esempio completo di un programma COBOL che legge un file relativo, ad accesso diretto, scandendo sequenzialmente il numero del record:

Listato 72.38. Programma elementare che legge un file relativo, ad accesso diretto.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ESEMPIO-RELATIVO-DIRETTO.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-08.
000500 ENVIRONMENT DIVISION.
000600 INPUT-OUTPUT SECTION.
000700*
000800 FILE-CONTROL.
000900     SELECT MIO-FILE ASSIGN TO "input.rel"
001000                     ORGANIZATION IS RELATIVE
001100                     ACCESS MODE IS RANDOM
001200                     RELATIVE KEY IS N-RECORD
001300                     FILE STATUS IS STATO-DEL-FILE.
001400*
001500 DATA DIVISION.
001600 FILE SECTION.
001700 FD  MIO-FILE
001800     LABEL RECORD IS STANDARD.
001900 01  MIO-RECORD  PIC X(20).
002000 WORKING-STORAGE SECTION.
002100 77  N-RECORD             PIC 9999 COMP VALUE IS ZERO.
002200 77  STATO-DEL-FILE       PIC XX.
002300 PROCEDURE DIVISION.
002400 MAIN.
002500     OPEN INPUT MIO-FILE.
002600     DISPLAY "FILE STATUS: ", STATO-DEL-FILE.
002700     PERFORM READ-FILE UNTIL STATO-DEL-FILE NOT = ZERO.
002800     CLOSE MIO-FILE.
002900     STOP RUN.
003000 READ-FILE.
003100     ADD 1 TO N-RECORD.
003200     READ MIO-FILE
003300          INVALID KEY DISPLAY "INVALID KEY!".
003400     DISPLAY "FILE STATUS: " STATO-DEL-FILE,
003500             " RECORD: ", N-RECORD, " ", MIO-RECORD.
003600
```



Il file indicato come ‘**MIO-FILE**’ è associato in pratica al file ‘`input.rel`’. Si può supporre che questo file sia composto dall’elenco seguente di record logici:

1. ‘**aaaaaaaaaaaaaaaaaaaa**’
2. ‘**bbbbbbbbbbbbbbbbbbbb**’
3. ‘**ccccccccccccccccccc**’
4. ‘**ddddddddddddddddddddd**’

Eseguendo il programma dell’esempio si potrebbe ottenere il testo seguente attraverso lo schermo:

```
FILE STATUS: 00
FILE STATUS: 00 RECORD: 0001 aaaaaaaaaaaaaaaaaaaa
FILE STATUS: 00 RECORD: 0002 bbbbbbbbbbbbbbbbbbbb
FILE STATUS: 00 RECORD: 0003 ccccccccccccccccccc
FILE STATUS: 00 RECORD: 0004 ddddddddddddddddddddd
INVALID KEY!
FILE STATUS: 23 RECORD: 0005 ddddddddddddddddddddd
```

Segue un altro esempio completo per la lettura di un file relativo, utilizzando un accesso dinamico, partendo dal primo record e selezionando i successivi attraverso la richiesta del prossimo:

Listato 72.40. Programma elementare che legge un file relativo, ad accesso dinamico.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ESEMPIO-RELATIVO-DINAMICO.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-08.
000500 ENVIRONMENT DIVISION.
000600 INPUT-OUTPUT SECTION.
000700*
000800 FILE-CONTROL.
000900     SELECT MIO-FILE ASSIGN TO "input.rel"
001000                     ORGANIZATION IS RELATIVE
001100                     ACCESS MODE IS DYNAMIC
001200                     RELATIVE KEY IS N-RECORD
001300                     FILE STATUS IS STATO-DEL-FILE.
001400*
001500 DATA DIVISION.
001600 FILE SECTION.
001700 FD  MIO-FILE
001800     LABEL RECORD IS STANDARD.
001900 01  MIO-RECORD  PIC X(20).
002000 WORKING-STORAGE SECTION.
002100 77  N-RECORD             PIC 9999 COMP VALUE IS 1.
002200 77  STATO-DEL-FILE       PIC XX.
```

```
002300 PROCEDURE DIVISION.
002400 MAIN.
002500     OPEN INPUT MIO-FILE.
002600     DISPLAY "FILE STATUS: ", STATO-DEL-FILE.
002700     READ MIO-FILE
002800          INVALID KEY DISPLAY "INVALID KEY!".
002900     PERFORM READ-FILE UNTIL STATO-DEL-FILE NOT = ZERO.
003000     CLOSE MIO-FILE.
003100     STOP RUN.
003200 READ-FILE.
003300     DISPLAY "FILE STATUS: " STATO-DEL-FILE,
003400             " RECORD: ", N-RECORD, " ", MIO-RECORD.
003500     READ MIO-FILE NEXT RECORD
003600          AT END DISPLAY "END OF FILE!".
003700
```

Il file che viene letto è lo stesso dell'esempio precedente e il risultato si dovrebbe ottenere, si può vedere così:

```
FILE STATUS: 00
FILE STATUS: 00 RECORD: 0001 aaaaaaaaaaaaaaaaaaaa
FILE STATUS: 00 RECORD: 0002 bbbbbbbbbbbbbbbbbbbb
FILE STATUS: 00 RECORD: 0003 cccccccccccccccccccc
FILE STATUS: 00 RECORD: 0004 dddddddddddddddddddd
END OF FILE!
```

### 72.4.3.5 Istruzione «SELECT» per i file a indice

Lo schema sintattico semplificato per l'istruzione '**SELECT**', da usare nel paragrafo '**FILE-CONTROL**', per dichiarare un file organizzato a indici è quello che si può vedere nella figura successiva:

```
                          /                  \
                          |       DISK       |
SELECT file-name ASSIGN TO  <                  >
                          | literal-file-name |
                          \                  /

    .--               .--   --. --.
    |                 | AREA  |   |
    | RESERVE integer |       |   |
    |                 | AREAS |   |
    '--               '--   --' --'

    [ ORGANIZATION IS ] INDEXED

    .--               /              \ --.
    |                 |  SEQUENTIAL  |   |
    |                 |              |   |
    | ACCESS MODE IS  <  RANDOM      >   |
    |                 |              |   |
    |                 |  DYNAMIC     |   |
    '--               \              / --'

    RECORD KEY IS data-name-1 [WITH DUPLICATES]

    [ ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES] ]...

    [ FILE STATUS IS data-name-3 ].
```

Un file organizzato a indice è un file che consente un accesso diretto ai record in base a una chiave costituita da una porzione del record stesso. A titolo di esempio, si può immaginare un file contenente i dati anagrafici dei dipendenti di un'azienda, che in una posizione precisa dei record riporta il numero di matricola di ognuno; in tal modo, il numero di matricola può essere usato per definire la chiave di accesso ai record.

Il file organizzato a indice può disporre di una o più chiavi di accesso e può essere consentita o meno la presenza di record con chiavi uguali.

Rispetto ai file organizzati sequenzialmente o in modo relativo, lo schema sintattico per i file organizzati a indice ha le dichiarazioni '**RECORD KEY**' e '**ALTERNATE RECORD KEY**' per poter specificare la chiave o le chiavi di accesso. Le metavariabili '**data-name-1**' e '**data-name-2**' devono essere nomi di porzioni di record, come dichiarato nella divisione '**DATA DIVISION**', in corrispondenza della descrizione del record stesso. Naturalmente, l'opzione '**WITH DUPLICATES**' serve a dichiarare l'intenzione di gestire chiavi uguali su più record.



### 72.4.3.6 Riordino e fusione

Oltre ai file comuni, per i quali si stabilisce un'organizzazione e un tipo di accesso, sono previsti dei file da usare soltanto per ottenere un riordino o una fusione (*sort*, *merge*). Per questi file occorre una dichiarazione apposita con l'istruzione '**SELECT**', secondo lo schema sintattico seguente:

```
                                          /                  \
                                          |       DISK       |
SELECT sort-merge-file-name ASSIGN TO     <                  > .
                                          | literal-file-name |
                                          \                  /
```

Viene proposto un esempio di riordino di file, nel quale, in particolare, si dichiarano i nomi dei file su disco, direttamente nell'istruzione '**SELECT**':

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ORDINA.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-ORDINARE     ASSIGN TO "input.seq".
001300     SELECT FILE-ORDINATO        ASSIGN TO "output.seq".
001400     SELECT FILE-PER-IL-RIORDINO ASSIGN TO "sort.tmp".
001500*
001600 DATA DIVISION.
001700*
001800 FILE SECTION.
001900*
002000 FD  FILE-DA-ORDINARE.
002100 01  RECORD-DA-ORDINARE           PIC X(80).
002200*
002300 FD  FILE-ORDINATO.
002400 01  RECORD-ORDINATO              PIC X(80).
002500*
002600 SD  FILE-PER-IL-RIORDINO.
002700*
002800 01  RECORD-PER-IL-RIORDINO.
002900     02  CHIAVE-ORDINAMENTO       PIC X(10).
003000     02  FILLER                   PIC X(70).
003100*
003200 PROCEDURE DIVISION.
003300*
003400 MAIN.
003500     SORT FILE-PER-IL-RIORDINO,
003600          ON ASCENDING KEY CHIAVE-ORDINAMENTO,
003700          USING FILE-DA-ORDINARE,
003800          GIVING FILE-ORDINATO.
003900*
004000     STOP RUN.
004100*
```

Come si può vedere, si vuole ordinare il file '`input.seq`' per generare il file '`output.seq`', ordinato. Per fare questo, si usa un file intermedio, denominato '`sort.tmp`'. Al termine dell'operazione, non dovrebbe rimanere traccia del file intermedio.

Si osservi che non si rende necessaria l'apertura dei file coinvolti per portare a termine l'operazione.

L'esempio seguente riguarda la fusione: si hanno i file '`input-1.seq`' e '`input-2.seq`' ordinati e si vuole ottenere il file '`output.seq`' con la somma dei record, mantenendo l'ordinamento:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   MERGE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
```

```
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-INPUT-1         ASSIGN TO "input-1.seq".
001300     SELECT FILE-INPUT-2         ASSIGN TO "input-2.seq".
001400     SELECT FILE-OUTPUT          ASSIGN TO "output.seq".
001500     SELECT FILE-PER-LA-FUSIONE  ASSIGN TO "merge.tmp".
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-INPUT-1
002200 01  RECORD-1                    PIC X(80).
002300*
002400 FD  FILE-INPUT-2
002500 01  RECORD-2                    PIC X(80).
002600*
002700 FD  FILE-OUTPUT
002800 01  RECORD-OUTPUT               PIC X(80).
002900*
003000 SD  FILE-PER-LA-FUSIONE.
003100*
003200 01  RECORD-PER-LA-FUSIONE.
003300     02  CHIAVE-ORDINAMENTO      PIC X(10).
003400     02  FILLER                  PIC X(70).
003500*
003600 PROCEDURE DIVISION.
003700*
003800 MAIN.
003900     MERGE FILE-PER-LA-FUSIONE
004000           ON ASCENDING KEY CHIAVE-ORDINAMENTO,
004100           USING FILE-INPUT-1,
004200                 FILE-INPUT-2,
004300           GIVING FILE-OUTPUT.
004400*
004500     STOP RUN.
004600*
```

Si osservi che esistono compilatori COBOL, di buona qualità, che però non offrono le funzionalità di riordino e di fusione, oppure non in modo completo. È frequente l'assenza della funzione per la fusione dei file ordinati.

### 72.4.3.7 Paragrafo «I-O-CONTROL»

Il paragrafo '**I-O-CONTROL**' è opzionale e il suo scopo è quello di specificare l'utilizzo comune delle aree di memoria centrale adibite alla gestione dei file.

```
I-O-CONTROL.

    .--      .--          --.                                  --.
    |        |  RECORD      |                                    |
    |        |              |                                    |
    |  SAME  |  SORT        |  AREA FOR file-name-1 [file-name-2]...  |... .
    |        |              |                                    |
    |        |  SORT-MERGE  |                                    |
    '--      '--          --'                                  --'
```

L'utilità dell'utilizzo del paragrafo '**I-O-CONTROL**' dipende molto dal compilatore, che potrebbe anche limitarsi a ignorare l'istruzione '**SAME**…**AREA**', in tutto o solo in parte. Tuttavia, quando l'istruzione '**SAME**…**AREA**' viene presa in considerazione, ci sono delle conseguenze nell'accesso ai file, che bisogna conoscere.

Per cominciare: si intuisce dallo schema sintattico che l'istruzione '**SAME**…**AREA**' inizia nell'area B del modulo di programmazione, si vede che il punto fermo è richiesto solo alla fine del gruppo di istruzioni '**SAME**…**AREA**', inoltre sono evidenti quattro possibilità:

```
SAME AREA FOR file-name-1 [file-name-2]... .

SAME RECORD AREA FOR file-name-1 [file-name-2]... .

SAME SORT AREA FOR file-name-1 [file-name-2]... .

SAME SORT-MERGE AREA FOR file-name-1 [file-name-2]... .
```

Utilizzando la prima forma dell'istruzione '**SAME AREA**', si intende richiedere al compilatore che la gestione dei file elencati sia fatta condividendo tutto quello che si può condividere nella memoria centrale. Così facendo, nell'ambito del gruppo specificato, solo un file può essere aperto simultaneamente; inoltre, se si utilizzano più istruzioni '**SAME AREA**', un file può appartenere soltanto a uno di questi raggruppamenti.



Utilizzando l'istruzione '**SAME RECORD AREA**' si richiede al compilatore di gestire lo spazio della memoria tampone (dei record) di un gruppo di file in modo comune. Così facendo, la lettura di un record di un file del gruppo, comporta il fatto che gli stessi dati siano disponibili come se fossero stati letti da tutti gli altri file del gruppo. I file di un gruppo definito con questa istruzione possono essere aperti simultaneamente, ma le operazioni di accesso ai dati non possono essere simultanee; inoltre, un file può appartenere a un solo raggruppamento di questo tipo.

Teoricamente, i file indicati in un raggruppamento con l'istruzione '**SAME AREA**' possono apparire anche in un raggruppamento con l'istruzione '**SAME RECORD AREA**', ma in tal caso deve trattarsi di tutti quelli che appartengono al primo di questi due (tutti quelli in '**SAME AREA**' devono essere parte di quello in '**SAME RECORD AREA**'). Inoltre, questo fatto comporta che i file che si trovano anche in '**SAME AREA**' non possono essere aperti simultaneamente.

Nei manuali COBOL classici si sottolinea il fatto che la condivisione dei record offra dei vantaggi in velocità e in risparmio di memoria; in particolare si suggerisce in tali manuali la possibilità di dichiarare nel dettaglio uno solo dei record del gruppo, oppure la possibilità di ridefinire i record cambiando il punto di vista (il record rispetto a quello di un altro). Tuttavia, considerata la potenza elaborativa degli elaboratori attuali, dal momento che esiste comunque la possibilità di ridefinire la suddivisione di un record, l'uso di questo paragrafo diventa sconsigliabile, se non altro per le complicazioni che si creano nell'interpretazione umana del programma sorgente.

Le istruzioni '**SAME SORT AREA**' e '**SAME SORT-MERGE AREA**' sono equivalenti e consentono di condividere la memoria utilizzata per i file che servono specificatamente per il riordino o la fusione. Premesso che in questi raggruppamenti non possono apparire file che appartengono a un gruppo definito come '**SAME AREA**', è invece possibile inserire anche nomi di file che non sono stati dichiarati per l'ordinamento o la fusione, ma la loro presenza fa sì che questi file non possano essere aperti quando invece lo sono quelli che si utilizzano proprio per tale scopo.

I file dichiarati con l'indicatore '**SD**' nella sezione '**FILE SECTION**' servono per portare a termine le operazioni di riordino e di fusione, ma si avvalgono di file in ingresso e di file in uscita, che vengono dichiarati normalmente con l'indicatore '**FD**'. Tutti i file coinvolti in un procedimento di riordino e di fusione, non devono essere aperti esplicitamente durante questa fase.

Tabella 72.48. Codici di due caratteri sullo stato dei file ('**FILE STATUS**'), secondo lo standard del 1985: il significato del primo dei due caratteri.

| Codice | Descrizione |
|---|---|
| 0*x* | L'ultimo accesso al file si è concluso sostanzialmente con successo. |
| 1*x* | Si è verificato un tentativo di leggere oltre la fine del file. |
| 2*x* | Si è verificato un errore riferito alla chiave di accesso di un file organizzato a indici. |
| 3*x* | Si è verificato un errore che impedisce di accedere ulteriormente al file. |
| 4*x* | Si è verificato un errore «logico», dovuto a una sequenza errata nelle operazioni o al tentativo di eccedere rispetto ai limiti stabiliti. |
| 9*x* | Si tratta di errori diversi, stabiliti senza uno standard precisato da chi ha realizzato il compilatore. |

Tabella 72.49. Codici di due caratteri sullo stato dei file (‘**FILE STATUS**’), secondo lo standard del 1985: significato dettagliato.

| Codice | Organizzazione sequenziale | Organizzazione relativa | Organizzazione a indici |
|---|---|---|---|
| 00 | Operazione eseguita con successo. | idem | idem |
| 02 | -- | -- | L’operazione ha avuto successo, ma è stata scoperta una chiave doppia: la lettura di un record evidenzia che è disponibile un altro record con la stessa chiave; la scrittura di un record risulta avere una chiave già presente in altri. |
| 04 | La lunghezza del record letto non corrisponde a quella che dovrebbe avere. | idem | idem |
| 05 | Il tentativo di aprire un file opzionale mancante è risultato nella sua creazione e apertura successiva. | idem | idem |
| 07 | Il file non si trova su nastro e le opzioni specifiche per tale tipo di unità, contenute nei comandi di apertura o di chiusura del file, sono state ignorate. | -- | -- |
| 10 | Si è verificato un tentativo di leggere oltre la fine del file, oppure di leggere un file opzionale che non risulta presente. | -- | -- |
| 14 | -- | La dimensione in record del file è più grande della capacità della variabile usata come indice. Questo tipo di errore si può manifestare quando si tenta una lettura sequenziale che dovrebbe incrementare automaticamente il valore dell’indice, ma si trova a non poterlo fare. | -- |
| 21 | -- | -- | Si è verificato un errore di sequenza nelle chiavi durante un accesso sequenziale al file. |
| 22 | -- | Si è verificato un tentativo di scrivere un record già esistente (senza prima averlo cancellato). | Si è verificato un tentativo di scrivere un record con chiave doppia, quando ciò non è consentito. |
| 23 | -- | Il record richiesto non esiste. | idem |
| 24 | -- | Tentativo di scrittura oltre il limite della dimensione consentita, oppure tentativo di scrittura sequenziale di un record che ha un numero più grande della capacità della variabile usata come chiave. | Tentativo di scrittura oltre il limite della dimensione consentita. |
| 30 | Errore permanente senza altre indicazioni. | idem | idem |
| 34 | Si è verificato un errore dovuto a un tentativo di scrittura oltre il limite fisico del file. | -- | -- |
| 35 | Non è stato possibile aprire un file indispensabile. | idem | idem |
| 36 | L’operazione richiesta non è gestita dall’unità che contiene il file. | idem | idem |
| 38 | È stata tentata l’apertura di un file che risulta essere stato chiuso con l’opzione ‘**LOCK**’. | idem | idem |
| 39 | È stata tentata l’apertura di un file, le cui caratteristiche reali sono incompatibili con quelle dichiarate nel programma. | idem | idem |
| 41 | È stato aperto un file che risulta essere già aperto. | idem | idem |
| 42 | È stato chiuso un file che non risultava essere aperto. | idem | idem |
| 43 | Non è stato eseguito un comando ‘**READ**’ prima del comando ‘**REWRITE**’. | Durante un accesso sequenziale, non è stato eseguito un comando ‘**READ**’ prima del comando ‘**REWRITE**’ o del comando ‘**DELETE**’. | Durante un accesso sequenziale, non è stato eseguito un comando ‘**READ**’ prima del comando ‘**REWRITE**’ o del comando ‘**DELETE**’. |
| 44 | Si è verificato un problema legato alla dimensione del record. | idem | idem |
| 46 | Durante un accesso sequenziale in lettura, si è verificato un errore, successivo a un altro tentativo fallito di lettura. | idem | idem |
| 47 | Tentativo di lettura di un file che non risulta essere aperto per questo tipo di accesso. | idem | idem |
| 48 | Tentativo di scrittura di un file che non risulta essere aperto per questo tipo di accesso. | idem | idem |

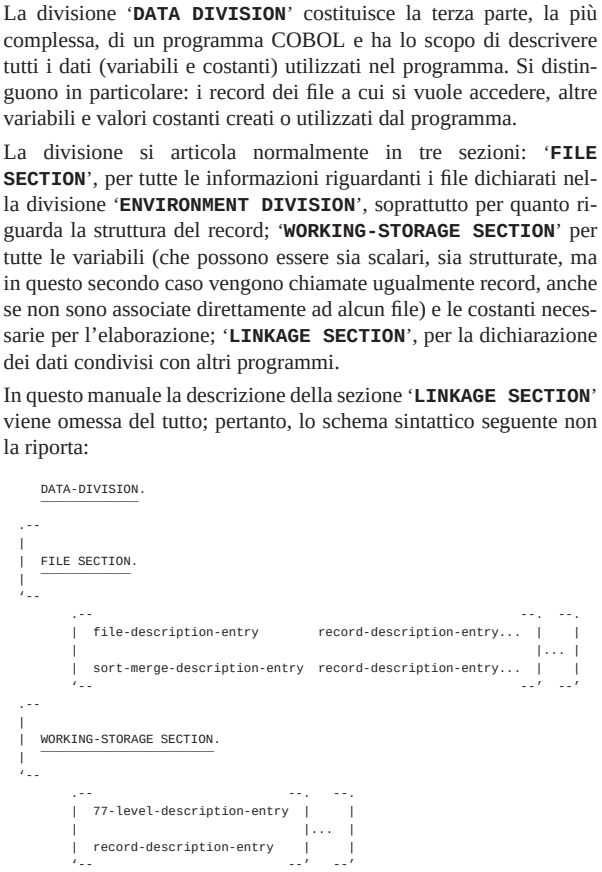
| Codice | Organizzazione sequenziale | Organizzazione relativa | Organizzazione a indici |
|---|---|---|---|
| 49 | Tentativo di riscrittura di un file che non risulta essere aperto per questo tipo di accesso. | idem | idem |

## 72.5 Divisione «DATA DIVISION»

La divisione '**DATA DIVISION**' costituisce la terza parte, la più complessa, di un programma COBOL e ha lo scopo di descrivere tutti i dati (variabili e costanti) utilizzati nel programma. Si distinguono in particolare: i record dei file a cui si vuole accedere, altre variabili e valori costanti creati o utilizzati dal programma.

La divisione si articola normalmente in tre sezioni: '**FILE SECTION**', per tutte le informazioni riguardanti i file dichiarati nella divisione '**ENVIRONMENT DIVISION**', soprattutto per quanto riguarda la struttura del record; '**WORKING-STORAGE SECTION**' per tutte le variabili (che possono essere sia scalari, sia strutturate, ma in questo secondo caso vengono chiamate ugualmente record, anche se non sono associate direttamente ad alcun file) e le costanti necessarie per l'elaborazione; '**LINKAGE SECTION**', per la dichiarazione dei dati condivisi con altri programmi.

In questo manuale la descrizione della sezione '**LINKAGE SECTION**' viene omessa del tutto; pertanto, lo schema sintattico seguente non la riporta:




Sulla base della terminologia usata nello schema sintattico, si può intuire il fatto che per il linguaggio COBOL, il termine record ha un significato particolare: si tratta di una variabile strutturata, che pertanto può essere scomposta in campi, in modo più o meno articolato. In questo senso, il contenuto della sezione '**WORKING-STORAGE SECTION**' viene suddiviso in due tipi di dichiarazioni: variabili scalari non suddivisibili (la metavariabile ***77-level-description-entry***) e variabili strutturate, ovvero record. Naturalmente, una variabile strutturata (dichiarata come record) può essere gestita e usata tranquillamente come se fosse uno scalare puro e semplice, ma questo fatto ha eventualmente delle ripercussioni nell'efficienza del programma che si ottiene dalla compilazione.

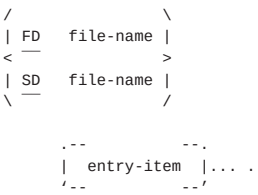
### 72.5.1 Sezione «FILE SECTION»

La sezione '**FILE SECTION**' ha lo scopo di definire le caratteristiche fisiche dei file e la struttura dei record. Tradizionalmente sarebbe in questa sezione che si specifica il nome o il percorso dei file in base al sistema operativo in cui si deve utilizzare il programma, salvo il caso in cui il compilatore voglia questa indicazione direttamente nella divisione '**ENVIRONMENT DIVISION**', precisamente nell'istruzione '**SELECT**' della sezione '**FILE CONTROL**'.

La descrizione di un file nella sezione '**FILE SECTION**' inizia con l'***indicatore di livello*** '**FD**' o '**SD**', a seconda che si tratti di un file



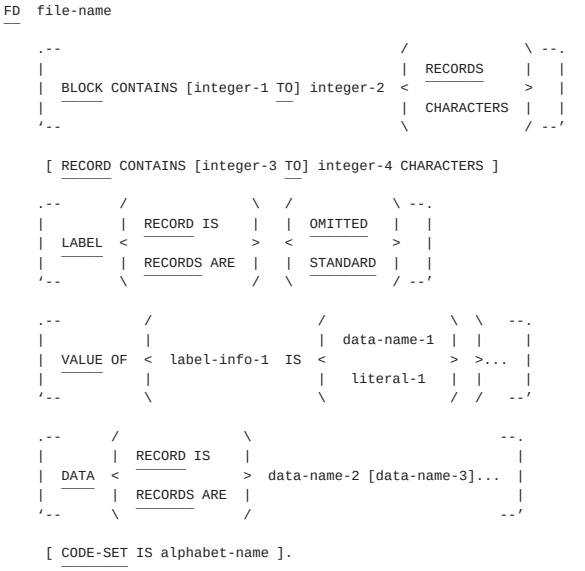
«normale» o di un file da usare per le operazioni di riordino e fusione. Si osservi che queste due istruzioni iniziano nell'area A del modulo di programmazione, continuando poi nell'area B, ma è importante sottolineare che già il nome del file, evidenziato nello schema sintattico con il nome ***file-name***, deve iniziare nell'area B:




Dopo ogni indicatore di livello '**FD**' o '**SD**' deve apparire la dichiarazione della variabile strutturata che rappresenta il record del file; tale dichiarazione inizia con il livello 01.

#### 72.5.1.1 Indicatore di livello «FD»

I file comuni, ovvero quelli che non sono stati dichiarati esplicitamente per eseguire delle operazioni di riordino o di fusione, si descrivono nella sezione '**FILE SECTION**' con l'indicatore di livello '**FD**' (*File description*), che in pratica è un'istruzione singola. Si ricordi che il nome del file che segue la parola chiave '**FD**' deve iniziare nell'area B del modulo di programmazione:




Si osservi che, a seconda del compilatore e del sistema operativo per il quale il programma viene compilato, diverse dichiarazioni inserite nell'indicatore di livello '**FD**' potrebbero essere ignorate in pratica.

#### 72.5.1.2 Indicatore di livello «SD»

I file da usare specificatamente per il riordino o la fusione, si descrivono nella sezione '**FILE SECTION**' con l'indicatore di livello '**SD**' (*Sort description*), che in pratica è un'istruzione singola. Si ricordi che il nome del file che segue la parola chiave '**SD**' deve iniziare nell'area B:

```
SD  file-name

     [ RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]

.--          /                    /              \  \   --.
|            |                    |  data-name-1  |  |     |
|  VALUE OF  <  label-info-1  IS  <               >  >...  |
|            |                    |   literal-1   |  |     |
'--          \                    \               /  /   --'

.--      /              \                              --.
|        |  RECORD IS    |                                |
|  DATA  <               >  data-name-2 [data-name-3]...  | .
|        |  RECORDS ARE  |                                |
'--      \               /                              --'
```

### 72.5.1.3 Dichiarazione «BLOCK CONTAINS»

All'interno dell'indicatore di livello '**FD**' è possibile dichiarare la dimensione di un blocco fisico per l'accesso ai record del file a cui si sta facendo riferimento.

In generale, si può contare sul fatto che il sistema operativo sia in grado di gestire in modo trasparente il problema dei blocchi fisici dei dati, rispetto ai record «logici» utilizzati dai programmi; tuttavia, ci possono essere contesti in cui il programma che si genera deve provvedere da solo ad accedere all'unità di memorizzazione, pertanto in questi casi conviene dichiarare nel programma la dimensione del blocco di dati da usare per la comunicazione con l'unità stessa. Storicamente la definizione del blocco consente di gestire meglio l'utilizzo di un'unità a nastro; in altre situazioni, come per esempio con un lettore o perforatore di schede, il blocco può contenere un solo record.

```
                                          /              \
                                          |  RECORDS     |
BLOCK CONTAINS [integer-1 TO] integer-2   <              >
                                          |  CHARACTERS  |
                                          \              /
```

Omettendo questa dichiarazione, si intende lasciare al compilatore o al sistema operativo il compito di determinare un valore predefinito valido.

L'unità di misura del blocco dipende dalla parola usata, o non usata, alla fine della dichiarazione: la parola chiave '**RECORDS**' indica che i valori numerici si riferiscono a quantità di record, mentre diversamente si intendono dei «caratteri». Generalmente è da considerare che per caratteri si intendano byte.

Se viene indicato un valore solo (***integer-2***), si intende che il blocco possa avere soltanto quella dimensione, altrimenti, si intende dire al compilatore che c'è la possibilità di usare blocchi che hanno una dimensione minima (***integer-1***) e una massima (***integer-2***).

### 72.5.1.4 Dichiarazione «DATA RECORD»

La dichiarazione '**DATA RECORD**', che riguarda sia l'indicatore di livello '**FD**', sia '**SD**', è superata e generalmente viene ignorata dai compilatori. Il suo scopo è quello di dichiarare il nome di una o più variabili strutturate che descrivono il record del file. Questa dichiarazione è superata soprattutto perché il record viene comunque indicato successivamente attraverso la dichiarazione di una variabile strutturata apposita.

```
      /              \
      |  RECORD IS    |
DATA  <               >  data-name-2 [data-name-3]...
      |  RECORDS ARE  |
      \               /
```

I nomi da inserire al posto delle metavariabili ***data-name*** dello schema sintattico devono corrispondere a nomi di record (variabili strutturate) descritti con il numero di livello 01. La presenza di più di uno di questi nomi nella dichiarazione '**DATA**' implica che i record del file possono avere una struttura e una dimensione differente.



### 72.5.1.5 Dichiarazione «LABEL RECORD»

A seconda del tipo di unità di memorizzazione dei dati, ci può essere la necessità di aggiungere ai record delle informazioni per poterne poi gestire l'accesso. Il linguaggio COBOL prevede la possibilità di dover gestire direttamente questo meccanismo di etichettatura dei record, ma generalmente i sistemi operativi attuali dovrebbero rendere questo meccanismo trasparente, togliendo al programma COBOL l'onere di doversene occupare.

La dichiarazione '**LABEL RECORD**' servirebbe per stabilire se siano da gestire le «etichette» dei record, oppure se questa funzione non debba essere considerata dal programma. Attualmente, tale dichiarazione è superata e generalmente i compilatori si limitano a ignorarla:

```
       /              \   /              \
       |  RECORD IS    |   |  OMITTED     |
LABEL  <               >   <              >
       |  RECORDS ARE  |   |  STANDARD    |
       \               /   \              /
```

Dovendo o volendo inserire questa dichiarazione, in caso di dubbio la forma '**LABEL RECORD IS STANDARD**' dovrebbe essere quella più adatta, anche se non è più compito del programma occuparsi delle etichette. Di per sé, l'omissione di questa dichiarazione comporta, per il compilatore che dovesse volerla, proprio l'utilizzo della forma standard.

### 72.5.1.6 Dichiarazione «RECORD CONTAINS»

La dichiarazione '**RECORD CONTAINS**', che riguarda sia l'indicatore di livello '**FD**', sia '**SD**', permette di specificare la dimensione del record:

```
RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
```

Come si può intuire, se si indica un valore solo, si intende che il record abbia una dimensione fissa, altrimenti si prevede un intervallo di valori: da un minimo a un massimo.

Generalmente, i compilatori si limitano a ignorare questa dichiarazione, perché le informazioni che porta sono già incluse nella variabile strutturata che descrive il record stesso, pertanto è sufficiente associare più variabili strutturate nella dichiarazione '**DATA RECORD**'.

### 72.5.1.7 Dichiarazione «CODE-SET»

La dichiarazione '**CODE-SET**' riguarda i file a organizzazione sequenziale e serve a specificare l'insieme di caratteri con cui tale file è codificato. Tradizionalmente, questa istruzione è servita per gestire dati in formato EBCDIC, in contrapposizione al codice ASCII, o viceversa.

```
CODE-SET IS alphabet-name
```

Al posto della metavariabile ***alphabet-name*** va inserita una parola che definisce l'insieme di caratteri del file, secondo le aspettative del compilatore utilizzato.

### 72.5.1.8 Dichiarazione «VALUE OF»

La dichiarazione '**VALUE OF**' consente, in un certo senso, di assegnare dei valori a delle voci legate alle caratteristiche del file. La cosa più importante che si potrebbe fare è di specificare il file da utilizzare secondo ciò che richiede il sistema operativo. Per esempio, se si tratta di un file su disco e il sistema operativo richiede di indicare anche i dischi per nome, il compilatore dovrebbe prevedere qui una voce appropriata.

```
         /                    /              \  \
         |                    |  data-name-1  |  |
VALUE OF  <  label-info-1  IS  <               >  >...
         |                    |   literal-1   |  |
         \                    \               /  /
```

Le voci che si possono dichiarare qui possono essere di ogni tipo, con la possibilità di abbinare un valore costante (una stringa alfanumerica), oppure una variabile il cui contenuto viene poi modificato in fase elaborativa.

L'estratto seguente di un programma COBOL, scritto per il compilatore TinyCOBOL, mostra l'uso della voce '**FILE-ID**' per dichiarare il nome del file da utilizzare:

```
001000 FILE-CONTROL.
001100     SELECT FILE-NAME ASSIGN TO DISK
001200                      ORGANIZATION IS SEQUENTIAL
001300                      ACCESS MODE IS SEQUENTIAL
001400                      FILE STATUS IS DATA-NAME.
001600 DATA DIVISION.
001700 FILE SECTION.
001800 FD  FILE-NAME
001900     LABEL RECORD IS STANDARD
002000     VALUE OF FILE-ID IS "input.seq".
002100 01  RECORD-NAME  PIC X(20).
002200 WORKING-STORAGE SECTION.
002300 01  DATA-NAME    PIC XX.
```

### 72.5.1.9 Descrizione del record

Dopo ogni indicatore di livello ('**FD**' o '**SD**') si deve descrivere il record attraverso una variabile strutturata, che si dichiara con quelli che sono noti come ***livelli***. I livelli sono in pratica delle dichiarazioni che costituiscono ognuna delle istruzioni singole, ma in tal caso, a differenza delle istruzioni comuni, iniziano con un numero: il numero di livello.

Il livello 01 è obbligatorio e dichiara il nome della variabile strutturata che descrive il record nella sua interezza; qualunque numero superiore serve a descrivere una porzione inferiore del record, con la possibilità di scomposizioni successive. I numeri di livello che possono essere usati per questo scopo sono limitati all'intervallo da 01 a 49, tenendo conto che, a parte l'obbligo di iniziare da 01, i livelli inferiori possono utilizzare incrementi superiori all'unità. Si osservi l'esempio seguente che contiene un estratto dalla sezione '**FILE SECTION**':

```
001600 DATA DIVISION.
001700 FILE SECTION.
001800 FD  SALES-FILE
001830     LABEL RECORD IS STANDARD
001860     VALUE OF FILE-ID IS "sales".
001900 01  SALES-RECORD.
002000     05  SALES-VENDOR-NAME      PIC X(20).
002100     05  SALES-VALUE            PIC S9(6).
002200     05  SALES-NUMBER           PIC X(13).
002300     05  SALES-TYPE             PIC X.
002400     05  SALES-VENDOR-REGION    PIC X(17).
002500     05  SALES-VENDOR-CITY      PIC X(20).
002600     05  SALES-COMMENTS         PIC X(60).
```

Il file individuato dal nome '**SALES-FILE**' si compone di record a cui si può fare riferimento con la variabile strutturata '**SALES-RECORD**'. Il record si suddivide in sette campi con caratteristiche diverse. Il record nella sua interezza corrisponde al livello 01, evidenziato dalla sigla '**01**' che si trova nell'area A del modulo di programmazione. Come si vede nel livello 01 dell'esempio, la variabile strutturata che rappresenta tutto il record viene solo nominata, senza altre indicazioni, perché la sua dimensione si determina dalla somma dei campi che contiene.

I numeri di livello, mano a mano che si annidano in sottolivelli successivi, devono crescere: non è importante se il numero cresce di una o di più unità. Tradizionalmente, i livelli vengono incrementati con un passo maggiore di uno, per facilitare la modifica del sorgente quando dovesse presentarsi l'esigenza di ristrutturare i livelli.

Per comprendere meglio il senso della descrizione del record attraverso il sistema dei livelli, conviene dare un'occhiata allo schema successivo:



| registrazione n. | data operazione | | | carico | | scarico | | descrizione operazione |
|---|---|---|---|---|---|---|---|---|
| | anno | mese | giorno | quantità caricata | costo unitario | quantità scaricata | valore unitario di scarico | |

Quello che appare nello schema vuole rappresentare il record di un file da usare per memorizzare carichi e scarichi di un magazzino. Si può osservare inizialmente un campo per numerare le registrazioni (ogni registrazione occupa un record), successivamente, appare la data dell'operazione suddivisa in tre parti (anno, mese e giorno), quindi viene indicato il carico, suddividendo la quantità caricata e il costo unitario di carico, quindi lo scarico, anche questo diviso in quantità scaricata e valore unitario di scarico, infine appare un campo descrittivo dell'operazione. Un record di questo tipo potrebbe essere descritto utilizzando i livelli nel modo seguente:

```
000000 01  RECORD-MOVIMENTI-DI-MAGAZZINO.
000000     10  MM-NUMERO-REGISTRAZIONE          PIC 99999.
000000     10  MM-DATA-REGISTRAZIONE.
000000         20  MM-DATA-REGISTRAZIONE-ANNO   PIC 9999.
000000         20  MM-DATA-REGISTRAZIONE-MESE   PIC 99.
000000         20  MM-DATA-REGISTRAZIONE-GIORNO PIC 99.
000000     10  MM-CARICO.
000000         20  MM-CARICO-QUANTITA           PIC 9(8)V999.
000000         20  MM-CARICO-COSTO-UNITARIO     PIC 999999V99.
000000     10  MM-SCARICO.
000000         20  MM-SCARICO-QUANTITA          PIC 9(8)V999.
000000         20  MM-SCARICO-VALORE-UNITARIO   PIC 999999V99.
000000     10  MM-DESCRIZIONE                   PIC X(200).
```

Come si può comprendere dall'esempio e come già accennato in precedenza, per le porzioni di record che non si scompongono ulteriormente, si devono specificare le dimensioni, sommando le quali si ottiene la dimensione dei vari raggruppamenti e infine del record complessivo. La sintassi per rappresentare i livelli si potrebbe semplificare in questa fase nel modo seguente, dove però non si usa la notazione standard del linguaggio COBOL:

***nn*** ***nome-campo*** [PIC[TURE] [IS] ***modello_della_variabile*** [***opzioni***]].

Ciò che non è stato descritto fino a questo punto è la parte di dichiarazione successiva al nome del campo, che inizia con la parola chiave '**PICTURE**', spesso abbreviata soltanto con '**PIC**'. Ciò che appare qui serve a definire il modello della variabile, ovvero la sua dimensione e le sue caratteristiche.

Il modello di definizione della variabile è una stringa che va composta seguendo regole precise. Con questo modello si specifica se la variabile è di tipo numerico o alfanumerico, la sua dimensione, la presenza eventuale di una virgola (ovviamente per i valori numerici), il segno ed eventualmente una maschera di trasformazione. Dopo il modello di definizione della variabile possono apparire delle opzioni, in forma di dichiarazioni ulteriori, che servono a precisare la modalità con cui la variabile deve essere rappresentata internamente alla memoria centrale.

Quando si dichiara una variabile numerica, è importante chiarire quale rappresentazione deve avere. A seconda del compilatore, la variabile numerica potrebbe essere gestita in forma binaria oppure in forma BCD (*Binary coded decimal*), che a sua volta può essere «normale», dove ogni cifra occupa un byte, oppure *packed*, dove ogni cifra occupa mezzo byte (4 bit, noto anche come nibble). Questa caratteristica della variabile si definisce con le dichiarazioni opzionali che seguono il modello di definizione della variabile.

Il modo in cui si dichiara il modello di definizione della variabile è descritto nella sezione 72.9, mentre per una visione complessiva del modo in cui si dichiara una variabile, si deve consultare la sezione 72.6; tuttavia, in questa fase si può cominciare ugualmente a interpretare l'esempio mostrato in precedenza, osservando in particolare i campi seguenti:

- il campo '**MM-NUMERO-REGISTRAZIONE**' può contenere un numero intero senza segno di cinque cifre: da zero a 99999;
- il campo '**MM-CARICO-QUANTITA**' può contenere un numero senza segno con otto cifre per la parte intera e tre cifre per la parte decimale;
- il campo '**MM-COSTO-UNITARIO**' può contenere un numero senza segno con sei cifre per la parte intera e due cifre per la parte decimale;
- il campo '**MM-DESCRIZIONE**' può contenere caratteri alfanumerici di qualunque tipo (nell'ambito di una rappresentazione in byte), per una dimensione di 200 caratteri.

Nell'esempio del magazzino si può notare che tutti i nomi usati per individuare le varie componenti del record sono unici, ma oltre a questo è stata usata l'accortezza di mettere un prefisso ('**MM-**') per distinguerli rispetto a campi di altri file che potrebbero avere una struttura del record simile. Tuttavia, non è strettamente necessario che tali nomi siano univoci per tutto il programma, perché è prevista la possibilità di qualificarli in modo gerarchico. La qualificazione è descritta nella sezione 72.8.3.

Esiste anche la possibilità di ridefinire la struttura di un record, assegnando un nome alternativo a un certo livello che si vuole descrivere diversamente. Si osservi l'esempio seguente:

```
000000 01  MIO-RECORD.
000000     02  CAMPO-A                          PIC X(20).
000000     02  RIDEFINITO-A REDEFINES CAMPO-A.
000000         03  DATA.
000000             04  ANNO                     PIC 9999.
000000             04  MESE                     PIC 99.
000000             04  GIORNO                   PIC 99.
000000         03  DESCRIZIONE                  PIC X(12).
000000     02  CAMPO-B …
…
```

Nell'esempio si vede un record denominato '**MIO-RECORD**', che inizialmente è composto dal campo '**CAMPO-A**' fatto per contenere 20 caratteri. Questo campo viene ridefinito nella riga successiva con il nome '**RIDEFINITO-A**', che si articola in sottocampi, con i quali si vuole descrivere in modo alternativo la variabile '**CAMPO-A**'. In base al contesto si intende che i primi otto caratteri possano essere interpretati come le cifre numeriche di una data (anno, mese e giorno), individuando il resto come una descrizione non meglio qualificabile.

Generalmente, la ridefinizione di un campo che non è suddiviso è di scarsa utilità, mentre è più interessante quando si applica a campi che hanno già una suddivisione, che però si vuole gestire anche in modo differente:

```
000000 01  MIO-RECORD.
000000     02  A
000000         03  B                            PIC X(10).
000000         03  C                            PIC X(10).
000000     02  D REDEFINES A.
000000         03  E                            PIC X(5).
000000         03  F                            PIC X(10).
000000         03  G                            PIC X(5).
000000     02  H …
…
```

In questo caso, il campo '**A**' è composto complessivamente da 20 caratteri, a cui si accede con i campi '**B**' e '**C**' per i primi 10 e gli ultimi 10 rispettivamente. La ridefinizione successiva, consente di accedere a una porzione centrale, a cavallo dei campi '**B**' e '**C**', con il campo '**F**'.

### 72.5.2 Sezione «WORKING-STORAGE SECTION»

La sezione '**WORKING-STORAGE SECTION**' serve a dichiarare le variabili, strutturate o scalari, utilizzate dal programma, che non si riferiscono direttamente alla descrizione dei record dei file:



```
WORKING-STORAGE SECTION.

      .--                          --.
      |  77-level-description-entry  |
      |                              |...
      |  record-description-entry    |
      '--                          --'
```

A differenza della sezione '**FILE SECTION**', oltre alla dichiarazione di variabili strutturate, è possibile dichiarare delle variabili scalari (non suddivisibili), utilizzando il livello speciale numero 77.

```
000000 WORKING-STORAGE SECTION.
000000 01  DATA-DA-SCOMPORRE.
000000     02  ANNO              PIC 9999.
000000     02  MESE              PIC 99.
000000     02  GIORNO            PIC 99.
000000 77  FINE-DEL-FILE         PIC 9.
000000 77  A                     PIC X(10).
000000 77  B                     PIC 9999V99.
```

Il livello 77 viene dichiarato mettendo il numero relativo nella colonna dell'area A del modulo di programmazione, così come si fa per il livello 01; nello stesso modo, il nome della variabile scalare si scrive nell'area B. L'esempio che appare sopra dovrebbe essere sufficiente a comprendere l'uso della sezione '**WORKING-STORAGE SECTION**', tenendo conto che vale quanto descritto a proposito delle variabili strutturate che descrivono i record nella sezione '**FILE SECTION**', compresa la ridefinizione.

La dichiarazione di una variabile scalare con il livello 77 consente di specificare dei tipi numerici binari (come '**USAGE IS INDEX**'), per i quali non si può prevedere la dimensione in modo standard. L'uso di questi tipi numerici speciali non è ammesso nei campi di una variabile scalare descrittiva di un record.

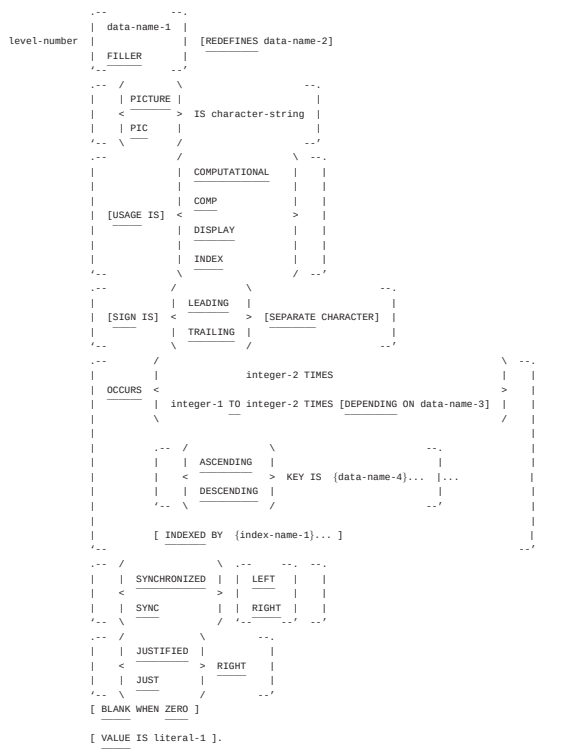
### 72.5.3 Altri livelli speciali

Oltre ai livelli che servono a descrivere le variabili strutturate (da 01 a 49) e le variabili scalari (77), sono previsti due livelli speciali: 66 e 88. Questi livelli speciali servono a definire dei raggruppamenti di variabili appartenenti alla stessa struttura o a definire dei «nomi di condizione».

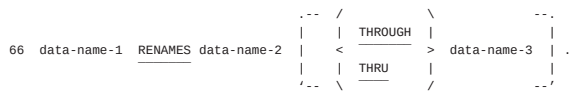
La descrizione di questi ulteriori livelli speciali viene fatta nella sezione 72.8.

## 72.6 Descrizione delle variabili

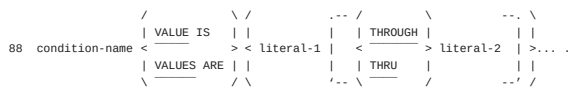
Lo schema sintattico seguente descrive la dichiarazione delle variabili per i livelli da 01 a 49 e per il livello 77:




Formato per il livello 66:




Formato per il livello 88:




Nel primo schema, l'indicazione del nome della variabile, che si vede in alternativa alla parola chiave '**FILLER**', deve apparire immediatamente dopo il numero di livello; inoltre, se si usa la parola chiave '**REDEFINES**', questa segue immediatamente il nome della variabile. Il resto delle opzioni può essere inserito con l'ordine che si preferisce, rispettando comunque lo schema e l'eventuale interdipendenza che c'è tra le opzioni stesse.

L'opzione che definisce il modello di definizione della variabile, introdotta dalla parola chiave '**PICTURE**', deve essere usata per ogni variabile che non si scompone ulteriormente e non può essere usata negli altri casi. Nello stesso modo, le opzioni '**SYNCHRONIZED**', '**JUSTIFIED**' e '**BLANK WHEN ZERO**' si possono usare solo per le variabili terminali (che non si scompongono).

Il valore iniziale delle variabili, a eccezione del tipo '**INDEX**', si specifica con l'opzione introdotta dalla parola chiave '**VALUE**'. Quando non si usa o non si può usare questa opzione, il valore iniziale delle variabile non è conosciuto e non può essere previsto.

La descrizione dei nomi di condizione (livello 88) è disponibile nella sezione 72.8.1, mentre per quella riguardante i raggruppamenti alternativi di variabili si può consultare la sezione 72.8.2.

### 72.6.1 Oggetto della dichiarazione

La dichiarazione di una variabile (livelli da 01 a 49 e 77) avviene indicandone il nome subito dopo il numero di livello. Il nome della variable può non essere univoco nell'ambito del programma, pur-



ché sia possibile individuare correttamente la variabile attraverso la qualificazione (sezione 72.8.3).

Al posto di indicare il nome di una variabile, è possibile mettere la parola chiave '**FILLER**', quando il campo a cui fa riferimento non viene mai utilizzato direttamente. Naturalmente, l'uso di questa parola chiave non è ammissibile in un livello '**77**', perché non avrebbe senso dichiarare una variabile scalare, indipendente da qualunque altra, senza avere poi la possibilità di utilizzarla nel programma.

### 72.6.2 Ridefinizione di una variabile

L'opzione '**REDEFINES**' può essere usata soltanto quando si dichiara una variabile, nell'ambito dei livelli da 01 a 49 (si esclude quindi l'uso della parola chiave '**FILLER**'), allo scopo di ridefinire la struttura di un'altra variabile che si trova allo stesso livello.

```
000000 01  MIO-RECORD.
000000     02  A
000000         03  B                              PIC X(10).
000000         03  C                              PIC X(10).
000000     02  D REDEFINES A.
000000         03  E                              PIC X(5).
000000         03  F                              PIC X(10).
000000         03  G                              PIC X(5).
000000     02  H …
…
```

L'esempio mostra che il campo '**A**' è composto complessivamente da 20 caratteri, a cui si accede con i campi '**B**' e '**C**' per i primi 10 e gli ultimi 10 rispettivamente. La ridefinizione successiva, consente di accedere a una porzione centrale, a cavallo dei campi '**B**' e '**C**', con il campo '**F**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   PICTURE-REDEFINES.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-A.
001200     02  P        PICTURE X(10)      USAGE IS DISPLAY.
001300     02  Q        PICTURE X(10)      USAGE IS DISPLAY.
001400 01  RECORD-B REDEFINES RECORD-A.
001500     02  R        PICTURE X(5)       USAGE IS DISPLAY.
001600     02  S        PICTURE X(10)      USAGE IS DISPLAY.
001700     02  T        PICTURE X(5)       USAGE IS DISPLAY.
001800*
001900 PROCEDURE DIVISION.
002000*
002100 MAIN.
002200     MOVE "ABCDEFGHIJKLMNOPQRST" TO RECORD-A.
002300     DISPLAY "P: ", P.
002400     DISPLAY "Q: ", Q.
002500     DISPLAY "R: ", R.
002600     DISPLAY "S: ", S.
002700     DISPLAY "T: ", T.
002800*
002900     STOP RUN.
003000*
```

Questo esempio ulteriore mostra un piccolo programma completo, che dimostra il funzionamento della ridefinizione, visualizzando le variabili associate a '**RECORD-A**' e a '**RECORD-B**', che ridefinisce il primo. Se si compila questo programma con TinyCOBOL o con OpenCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente:

```
P: ABCDEFGHIJ
Q: KLMNOPQRST
R: ABCDE
S: FGHIJKLMNO
T: PQRST
```

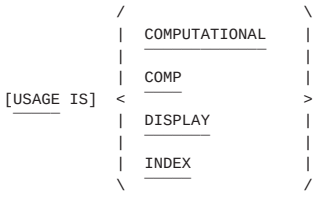
### 72.6.3 Opzione «PICTURE»

Attraverso l'opzione '**PICTURE**', che è obbligatoria per le variabili che non si scompongono in livelli inferiori, si definisce precisamente come è fatta la variabile. L'argomento dell'opzione è una stringa che descrive la variabile.

Il modo in cui si rappresenta il modello delle variabili è molto articolato e viene descritto nella sezione 72.9.

### 72.6.4 Opzione «USAGE»

Per il linguaggio COBOL, il contenuto di una variabile può essere solo di due tipi: alfanumerico o numerico. Una variabile dichiarata per contenere valori alfanumerici, utilizza un formato a «caratteri» (normalmente si tratta della codifica ASCII o EBCDIC), mentre per i valori numerici ci sono delle alternative. Attraverso l'opzione '**USAGE**' si può specificare il modo in cui la variabile deve contenere i dati.




Il formato definito dalla parola chiave '**DISPLAY**' corrisponde a un'organizzazione a caratteri. Si tratta evidentemente del formato necessario per le variabili alfanumeriche, ma può essere usato anche per le variabili numeriche.

Il formato '**COMPUTATIONAL**', abbreviabile anche con '**COMP**', può essere usato soltanto per i valori numerici e richiede l'utilizzo di 4 bit per ogni cifra numerica e anche per il segno (se specificato).

Il formato '**INDEX**' serve a dichiarare una variabile speciale, che può rappresentare soltanto numeri interi senza segno. Non si può associare un modello di definizione della variabile quando il formato è '**INDEX**', perché la dimensione è fissa e dipende dal compilatore. L'uso di queste variabili è anche limitato a certi contesti (di solito ne è ammesso l'uso negli indici delle tabelle ed eventualmente nei contatori delle iterazioni) e queste variabili si dichiarano soltanto con un livello 77 in modo da essere svincolati da una struttura a record.

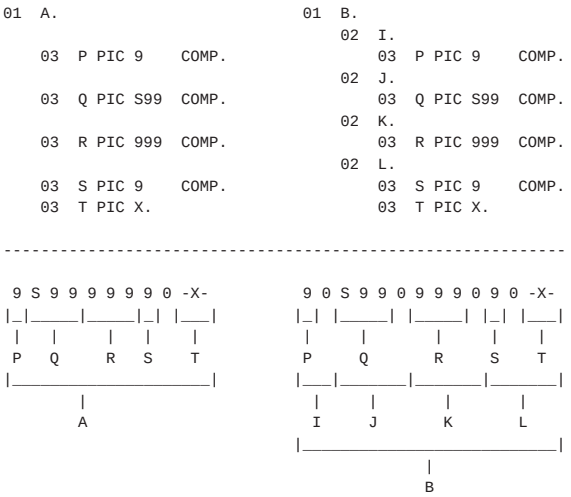
L'opzione '**USAGE**' va usata principalmente nella descrizione di variabili terminali; teoricamente si può indicare l'opzione '**USAGE**' anche per variabili che si suddividono in campi e in tal caso i livelli gerarchicamente inferiori ereditano l'opzione da quelli superiori. Tuttavia, è meglio evitare questa pratica e limitarsi a usare l'opzione '**USAGE**' soltanto sui campi terminali, tenendo conto che così i campi composti sono implicitamente di tipo '**DISPLAY**'.

Volendo assegnare a una variabile strutturata un tipo diverso da '**DISPLAY**', si otterrebbe solamente di assegnare questa caratteristica ai livelli inferiori, perché una variabile strutturata, ammesso che la si voglia utilizzare direttamente, nel suo complesso, può funzionare soltanto come '**DISPLAY**'. Come conseguenza, una variabile strutturata può essere formata soltanto da un numero intero di byte.

Quando si dichiarano su uno stesso livello variabili numeriche organizzate secondo il formato '**COMPUTATIONAL**', queste utilizzano regolarmente lo spazio loro assegnato, condividendo eventualmente i byte che dovessero trovarsi a cavallo tra una e l'altra; tuttavia, nell'ambito della variabile composta che contiene questi livelli, i dati devono occupare una quantità di byte intera, pertanto si può perdere eventualmente un po' di spazio alla fine.



Figura 72.75. Confronto fra due variabili strutturate.



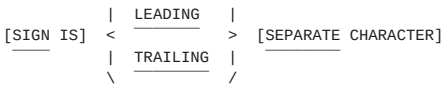

La figura dovrebbe essere sufficiente per capire come viene utilizzata la memoria per la rappresentazione delle variabili '**COMPUTATIONAL**'. Si può osservare che la variabile strutturata '**A**' ha soltanto uno spreco di una cifra '**COMPUTATIONAL**', ovvero 4 bit, mentre la variabile '**B**', avendo un'organizzazione differente nella sua struttura sottostante, spreca più spazio.

Molti compilatori COBOL considerano anche il tipo numerico '**COMPUTATIONAL-3**', o '**COMP-3**'. Si tratta di una variante del tipo '**COMPUTATIONAL**', con l'indicazione del segno o dell'assenza del segno, nel gruppo di 4 bit meno significativo (quello più a destra).

La maggior parte dei manuali COBOL sottolinea il fatto che per eseguire delle elaborazioni numeriche (dei calcoli matematici di qualunque tipo) è bene che le variabili utilizzate siano di tipo '**COMPUTATIONAL**', perché se le variabili sono ti tipo '**DISPLAY**', prima di poter essere utilizzate devono essere convertite.

### 72.6.5 Opzione «SIGN»

Quando si dichiarano variabili numeriche che prevedono l'indicazione del segno, è possibile stabilire in che modo e in quale posizione deve trovarsi, con l'opzione '**SIGN**':

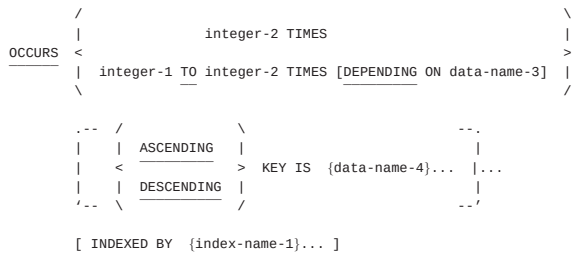



Le variabili numeriche di tipo '**DISPLAY**', in condizioni normali, incorporano il segno nel byte più significativo. Quando si vuole richiedere che il segno occupi un byte tutto per sé, è necessario usare la parola chiave '**SEPARATE**'. Per le variabili di tipo '**COMPUTATIONAL**' il segno occupa sempre uno spazio separato rispetto a quello delle cifre numeriche.

Se si utilizza la parola chiave '**LEADING**', il segno viene collocato a sinistra (e di norma questo è il comportamento predefinito); se invece si usa la parola chiave '**TRAILING**', il segno viene collocato nella posizione più a destra.

### 72.6.6 Opzione «OCCURS»

La parola chiave '**OCCURS**' introduce un gruppo di informazioni che consentono di indicare che la variabile a cui fanno riferimento viene ripetuta secondo certe modalità. Attraverso questo meccanismo si creano quelle che per il COBOL sono delle tabelle.



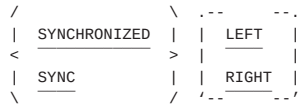

La variabile ricorrente di una tabella può ripetersi per un numero fisso di elementi (***integer-2***), oppure per un numero variabile, nell'ambito di un intervallo stabilito (da ***integer-1*** a ***integer-2***), sotto il controllo di un'altra variabile (***data-name-3***).

Se l'insieme degli elementi della tabella dichiarata con l'opzione '**OCCURS**' è ordinato in base a una chiave, questa può essere specificata (***index-name-1***); inoltre, l'indice per accedere agli elementi può essere dichiarato contestualmente già in questa fase (***index-name-1***).

Per maggiori dettagli, si veda la sezione 72.7, dedicata solo alle tabelle del COBOL.

### 72.6.7 Opzione «SYNCHRONIZED»

L'opzione '**SYNCHRONIZED**' avrebbe lo scopo di allineare una variabile numerica nei limiti di una o più «parole», secondo l'architettura usata nell'elaboratore per il quale è realizzato il compilatore.

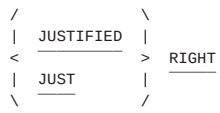



Teoricamente, l'uso dell'opzione '**SYNCHRONIZED**' avrebbe lo scopo di facilitare le elaborazioni numeriche, ma si creano delle complicazioni nel modo di determinare la dimensione effettiva delle variabili, soprattutto quando si vogliono ridefinire.

Per scrivere programmi COBOL compatibili tra i vari sistemi operativi e le architetture fisiche degli elaboratori, è meglio evitare l'uso di variabili '**SYNCHRONIZED**'; tuttavia, se si preferisce comunque usare questa funzione, diventa necessario consultare il manuale specifico del proprio compilatore.

### 72.6.8 Opzione «JUSTIFIED RIGHT»

Le variabili alfabetiche o alfanumeriche possono essere dichiarate con l'opzione '**JUSTIFIED**', per fare in modo che ricevano i dati allineandoli a destra:




Normalmente, quando si «invia» una stringa in una variabile alfanumerica, la copia inizia da sinistra a destra: se la variabile ricevente è più piccola della stringa, questa viene troncata alla destra; se la variabile ricevente è più grande, si aggiungono alla destra degli spazi.

Quando si usa l'opzione '**JUSTIFIED**' per una variabile (ricevente), la copia di una stringa avviene con un allineamento opposto, pertanto il troncamento avviene a sinistra e così anche l'aggiunta degli spazi ulteriori.



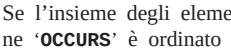
### 72.6.9 Opzione «BLANK WHEN ZERO»

L'opzione '**BLANK WHEN ZERO**' si può utilizzare solo per le variabili numeriche scalari, dove ogni cifra utilizza un byte intero. L'opzione fa sì che gli zeri anteriori vengano sostituiti da spazi, a meno che il modello di definizione della variabile preveda diversamente.




### 72.6.10 Opzione «VALUE»

Convenzionalmente, una variabile che viene dichiarata nei livelli da 01 a 49 e 77, non ha inizialmente un valore prestabilito, ma solo casuale. Per stabilire un valore da attribuire a una variabile nel momento della sua creazione, si usa l'opzione '**VALUE**':




La costante che nello schema sintattico è indicata come ***literal-1***, è il valore che viene attribuito inizialmente.

Si osservi che è possibile stabilire un valore iniziale per una variabile, soltanto quando si tratta di qualcosa che viene dichiarato nella sezione '**WORKING-STORAGE SECTION**', perché nella sezione '**FILE SECTION**' ciò non è possibile e non avrebbe senso.

L'opzione '**VALUE**' si usa anche per la dichiarazione dei nomi di condizione, ma in tal caso la funzione di questa opzione ha un valore differente e non c'è più la discriminazione tra le sezioni in cui si può utilizzare.

### 72.6.11 Opzione «RENAMES»

L'opzione '**RENAMES**', che si usa nel livello 66, permette di dichiarare delle variabili che rappresentano un raggruppamento di altre variabili, appartenenti alla stessa struttura, purché queste siano adiacenti. Nella sezione 72.8.2 viene mostrata la dichiarazione dei raggruppamenti.

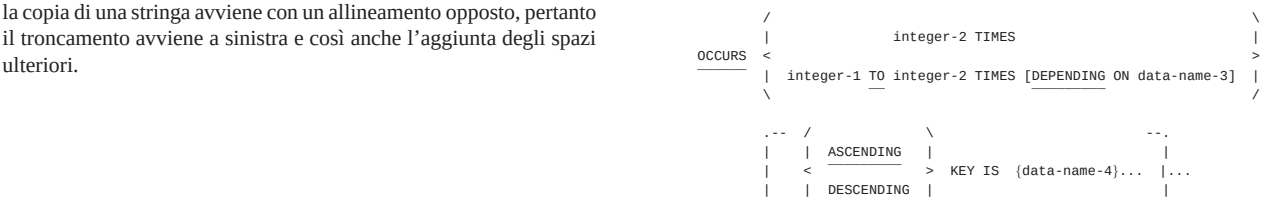
## 72.7 Tabelle

Il linguaggio COBOL offre la gestione di array attraverso la definizione di variabili multiple, all'interno di variabili strutturate (record); tuttavia, la denominazione usata nel COBOL per queste rappresentazioni dei dati è di ***tabella***.




### 72.7.1 Dichiarazione di una tabella

Si dichiara che un campo è composto da più elementi dello stesso tipo aggiungendo in coda l'opzione '**OCCURS *n* TIMES**'. Lo schema sintattico completo dell'opzione è il seguente:



```
'--  \  ___________  /                    --'

[ INDEXED BY  {index-name-1}... ]
```

Le tabelle più semplici sono quelle che hanno un numero fisso di elementi. Si osservi l'esempio seguente:

```
000000 01  A.
000000     02  B         PIC 9999.
000000     02  C                        OCCURS 10 TIMES.
000000         03  D     PIC X(10).
000000         03  E     PIC 99         OCCURS  7 TIMES.
000000     02  F         PIC X(10).
```

Nell'esempio viene dichiarata una variabile strutturata denominata '**A**', che si articola nelle variabili '**B**', '**C**' e '**F**'. La variabile '**C**' è ripetuta per 10 volte e si articola ogni volta nella variabile '**D**' e nella variabile '**E**'. La variabile '**E**' si ripete per sette volte.

La variabile '**E**' è una tabella a due dimensioni, perché è inclusa nelle ripetizioni della variabile '**C**', mentre la variabile '**C**' è una tabella a una sola dimensione.

È evidente che per fare riferimento ai valori contenuti nelle tabelle sia necessario utilizzare un indice.

L'opzione '**OCCURS**' si può utilizzare per tutte le variabili dichiarate con un numero di livello da 02 a 49. In pratica vengono esclusi i livelli 01 (record), 66 (usato per il raggruppamento delle variabili), 77 (usato esclusivamente per le variabili scalari) e 88 (nomi di condizione).

Lo standard del 1974 del linguaggio COBOL pone come limite un massimo di tre dimensioni per le tabelle.

### 72.7.2 Riferimento al contenuto di una tabella

Per fare riferimento a un elemento di una tabella, nelle istruzioni della divisione '**PROCEDURE DIVISION**' si usa una forma descritta dallo schema sintattico seguente:

```
data-name (subscript-1 [subscript-2 [subscript-3...]])
```

In pratica, si scrive il nome della variabile ripetuta, seguita dall'indice o dagli indici tra parentesi tonde. Il primo indice riguarda la prima dimensione, intesa come quella più esterna; l'ultimo riguarda l'annidamento più interno.

L'indice è un numero intero positivo che va da uno fino al massimo della dimensione che lo riguarda. Seguendo l'esempio apparso nella sezione precedente, '**E (1 7)**' rappresenta la settima occorrenza della variabile '**E**' nell'ambito della prima della variabile '**C**'. Pertanto, il nome da usare per indicare l'elemento è quello della variabile più interna che si vuole individuare, mentre gli indici partono dalla posizione più esterna.

Si noti che è convenzione comune inserire delle virgole per separare gli indici, anche se si tratta di una forma di rappresentazione facoltativa.

Viene mostrato un altro esempio di tabella a tre dimensioni:

```
000000 01  ENCICLOPEDIA.
000000     05  VOLUME                   OCCURS 10 TIMES.
000000         10  TITOLO-VOLUME            PIC X(30).
000000         10  PARTE                OCCURS 20 TIMES.
000000             15  TITOLO-PARTE         PIC X(30).
000000             15  CAPITOLO         OCCURS 30 TIMES.
000000                 20  TITOLO-CAPITOLO  PIC (30).
000000                 20  TESTO            PIC (200).
```

Si tratta di una variabile strutturata che serve a contenere delle informazioni su un'enciclopedia. L'elemento '**VOLUME (5)**' contiene le informazioni su tutto il volume quinto; l'elemento



'**TITOLO-VOLUME (5)**' contiene il titolo del volume quinto; l'elemento '**TITOLO-PARTE (5, 3)**' contiene il titolo della terza parte del volume quinto; l'elemento '**TESTO (5, 3, 25)**' contiene il testo del venticinquesimo capitolo contenuto nella terza parte del quinto volume. Naturalmente, in questo esempio si intende che la numerazione delle parti ricominci da uno all'inizio di ogni volume; così si intende che all'inizio di ogni parte la numerazione dei capitoli riprenda da uno.

### 72.7.3 Indice

L'indice di una tabella può essere indicato attraverso una costante numerica, una variabile numerica a cui sia stato attribuito preventivamente un valore appropriato o attraverso un'espressione elementare che risulta in un numero intero appropriato.

Quando si usa una variabile per la gestione di un indice, è possibile ed è consigliabile che il tipo numerico di tale variabile sia '**INDEX**'. In pratica, nella sezione '**WORKING-STORAGE SECTION**' un indice potrebbe essere dichiarato come nell'esempio seguente, dove se ne vedono due, il primo, denominato '**INDICE**', è dichiarato come variabile scalare di livello 77, il secondo, denominato '**INDICE-C**', è sempre costituito da una variabile scalare, che però fa parte di una variabile strutturata:

```
000000 77  INDICE           USAGE IS INDEX.
000000
000000 01  A.
000000     02  B            PIC X(30).
000000     02  INDICE-C     USAGE IS INDEX.
```

Si può osservare che questo tipo di variabile numerica non prevede la definizione della sua dimensione che è stabilita invece dal compilatore, in base alle caratteristiche dell'elaboratore e del sistema operativo.

In alternativa, **l'indice può essere dichiarato contestualmente alla dichiarazione della variabile ricorrente**; in tal caso, il compilatore può aggiungere dei controlli tali da impedire che si possa assegnare alla variabile un valore al di fuori dell'ambito della dimensione della tabella:

```
000000 01  ENCICLOPEDIA.
000000     05  VOLUME                   OCCURS 10 TIMES
000000                                  INDEX IS IND-VOLUME.
000000         10  TITOLO-VOLUME            PIC X(30).
000000         10  PARTE                OCCURS 20 TIMES
000000                                  INDEX IS IND-PARTE.
000000             15  TITOLO-PARTE         PIC X(30).
000000             15  CAPITOLO         OCCURS 30 TIMES
000000                                  INDEX IS IND-CAPITOLO.
000000                 20  TITOLO-CAPITOLO  PIC (30).
000000                 20  TESTO            PIC (200).
```

Qui viene ripreso e modificato un esempio già apparso in una sezione precedente. La differenza consiste nell'assegnare l'indice a ogni variabile ricorrente. Pertanto si hanno gli indici: '**IND-VOLUME**', '**IND-PARTE**' e '**IND-CAPITOLO**'.

Come accennato, si può fare riferimento a un elemento di una tabella indicando un indice costituito da un'espressione matematica elementare. Si parla in questo caso di indici relativi, perché si possono sommare o sottrarre dei valori a partire da una posizione di partenza. Per esempio, supponendo che la variabile '**I**' contenga il numero 15, l'elemento indicato come '**ELE (I - 4)**' corrisponderebbe alla notazione '**ELE (11)**'; nello stesso modo, l'elemento indicato come '**ELE (I + 4)**' corrisponderebbe alla notazione '**ELE (19)**'.

### 72.7.4 Tabelle di dimensione variabile

Teoricamente, è possibile dichiarare l'occorrenza di una variabile per una quantità variabile di elementi; si usa in tal caso la forma '**OCCURS *m* TO *n* TIMES**'. A seconda del compilatore, può essere obbligatorio, o facoltativo, specificare il nome di una variabile che controlla dinamicamente la quantità massima di elementi:

```
OCCURS  integer-1 TO integer-2 TIMES [DEPENDING ON data-name-3]
```

Viene mostrato l'esempio di un programma completo, che serve ad accumulare in una tabella alcuni dati personali. Sono previsti un massimo di 60 elementi e la quantità effettiva di elementi è controllata dalla variabile '**UTENTI-MAX**':

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1150.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-24.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-UTENTI.
001200     02  UTENTE           OCCURS 1 TO 60 TIMES
001300                          DEPENDING ON UTENTI-MAX
001400                          INDEXED BY IND-UTENTE.
001500         03  COGNOME      PIC X(30).
001600         03  NOME         PIC X(30).
001700         03  NOTA         PIC X(200).
001800 77  UTENTI-MAX           USAGE IS INDEX.
001900 77  EOJ                  PIC 9   VALUE ZERO.
002000 77  RISPOSTA             PIC XX.
002100*
002200 PROCEDURE DIVISION.
002300*------------------------ LIVELLO 0 -----------------
002400 MAIN.
002500     PERFORM INSERIMENTO-DATI
002600             VARYING IND-UTENTE FROM 1 BY 1
002700             UNTIL EOJ = 1.
002800     PERFORM SCANSIONE
002900             VARYING IND-UTENTE FROM 1 BY 1
003000             UNTIL IND-UTENTE > UTENTI-MAX.
003100*
003200     STOP RUN.
003300*------------------------ LIVELLO 1 -----------------
003400 INSERIMENTO-DATI.
003500     MOVE IND-UTENTE TO UTENTI-MAX.
003600     DISPLAY "INSERISCI IL COGNOME: ".
003700     ACCEPT COGNOME (IND-UTENTE).
003800     DISPLAY "INSERISCI IL NOME: ".
003900     ACCEPT NOME (IND-UTENTE).
004000     DISPLAY "INSERISCI UNA NOTA DESCRITTIVA: ".
004100     ACCEPT NOTA (IND-UTENTE).
004200*
004300     IF IND-UTENTE >= 60
004400       THEN
004500           MOVE 1 TO EOJ;
004600       ELSE
004700           DISPLAY "VUOI CONTINUARE? SI O NO",
004800           ACCEPT RISPOSTA;
004900           IF RISPOSTA = "SI"
005000             THEN
005100                 DISPLAY "OTTIMO!";
005200             ELSE
005300                 MOVE 1 TO EOJ.
005400*-------------------------------------------------------
005500 SCANSIONE.
005600     DISPLAY COGNOME (IND-UTENTE), " ",
005700             NOME (IND-UTENTE), " ",
005800             NOTA (IND-UTENTE).
005900*
```

### 72.7.5 Tabelle ordinate

Se si devono utilizzare i dati in una tabella per eseguire una ricerca al suo interno (utilizzando l'istruzione '**SEARCH**' nella divisione '**PROCEDURE DIVISION**'), se si può essere certi che le informazioni contenute siano ordinate secondo una certa chiave, lo si può specificare nella dichiarazione:

```
        /                                                   \
        |              integer-2 TIMES                      |
OCCURS  <                                                   >
        |  integer-1 TO integer-2 TIMES [DEPENDING ON data-name-3]  |
        \                                                   /

        .--  /             \                          --.
        |    | ASCENDING   |                           |
        |    <             >  KEY IS  {data-name-4}...  |...
        |    | DESCENDING  |                           |
        '--  \             /                          --'

        [ INDEXED BY  {index-name-1}... ]
```

La metavariabile ***data-name-4*** dello schema sintattico rappresenta una variabile contenuta nell'elemento ricorrente; attraverso la parola chiave '**ASCENDING**' si intende dichiarare che la tabella è ordinata, lessicograficamente, in modo ascendente, secondo il contenuto di quella variabile, se invece si usa la parola chiave '**DESCENDING**', si intende un ordinamento decrescente.

È possibile specificare più chiavi di ordinamento successive, nel caso si vogliano abbinare chiavi secondarie di ordinamento.

Sia chiaro che la tabella deve già risultare ordinata secondo le chiavi specificate, altrimenti le istruzioni '**SEARCH**' della divisione '**PROCEDURE DIVISION**' danno risultati errati o falliscono semplicemente. Naturalmente, all'interno del programma è possibile prevedere un procedimento di riordino, da eseguire prima di utilizzare delle istruzioni '**SEARCH**'.

L'esempio seguente mostra l'indicazione della chiave di ordinamento, costituita precisamente dalla variabile '**COGNOME**', che deve risultare ascendente in fase di ricerca:

```
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-UTENTI.
001200     02  UTENTE           OCCURS 1 TO 60 TIMES
001300                          DEPENDING ON UTENTI-MAX
001350                          ASCENDING KEY IS COGNOME
001400                          INDEXED BY IND-UTENTE.
001500         03  COGNOME      PIC X(30).
001600         03  NOME         PIC X(30).
001700         03  NOTA         PIC X(200).
001800 77  UTENTI-MAX           USAGE IS INDEX.
```

### 72.7.6 Scansione delle tabelle

Il linguaggio COBOL prevede un'istruzione apposita per facilitare la scansione delle tabelle. Si tratta di '**SEARCH**', che ha due modalità di funzionamento, a seconda che si voglia eseguire una ricerca sequenziale o una ricerca binaria. Naturalmente, la ricerca sequenziale si presta alla scansione di una tabella i cui dati non sono ordinati, mentre nel secondo caso devono esserlo.

Viene mostrato l'esempio di un programma completo che inizia con l'inserimento di dati all'interno di una tabella, quindi esegue una ricerca sequenziale al suo interno:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SEARCH.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-12.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 01  RECORD-UTENTI.
001400     02  UTENTE           OCCURS 60 TIMES
001500                          INDEXED BY IND-UTENTE.
001600         03  COGNOME      PIC X(30).
001700         03  NOME         PIC X(30).
001800         03  NOTA         PIC X(200).
001900 77  EOJ                  PIC 9   VALUE ZERO.
002000 77  RISPOSTA             PIC XX.
002100 77  RICERCA              PIC X(30).
002200*
002300 PROCEDURE DIVISION.
002400*------------------------ LIVELLO 0 -----------------
002500 MAIN.
002600     PERFORM INSERIMENTO-DATI
002700             VARYING IND-UTENTE FROM 1 BY 1
```

```
002800             UNTIL EOJ = 1.
002900     MOVE 0 TO EOJ.
003000     PERFORM SCANSIONE UNTIL EOJ = 1.
003100*
003200     STOP RUN.
003300*------------------------ LIVELLO 1 -----------------
003400 INSERIMENTO-DATI.
003500     DISPLAY IND-UTENTE, " INSERISCI IL COGNOME: ".
003600     ACCEPT COGNOME (IND-UTENTE).
003700     DISPLAY IND-UTENTE, " INSERISCI IL NOME: ".
003800     ACCEPT NOME (IND-UTENTE).
003900     DISPLAY IND-UTENTE,
003950             " INSERISCI UNA NOTA DESCRITTIVA: ".
004000     ACCEPT NOTA (IND-UTENTE).
004100*
004200     IF IND-UTENTE >= 60
004300       THEN
004400           MOVE 1 TO EOJ;
004500       ELSE
004600           DISPLAY "VUOI CONTINUARE? SI O NO",
004700           ACCEPT RISPOSTA;
004800           IF RISPOSTA = "SI"
004900             THEN
005000                 NEXT SENTENCE;
005100             ELSE
005200                 MOVE 1 TO EOJ.
005300*-------------------------------------------------------
005400 SCANSIONE.
005500     DISPLAY "INSERISCI IL COGNOME DA CERCARE:".
005600     ACCEPT RICERCA.
005700     IF RICERCA = SPACES
005800       THEN
005900           MOVE 1 TO EOJ;
006000       ELSE
006100           SET IND-UTENTE TO 1,
006200           SEARCH UTENTE
006300                 AT END
006400                       DISPLAY "IL COGNOME CERCATO ",
006500                               "NON SI TROVA NELLA ",
006550                               "TABELLA: ",
006600                               QUOTE RICERCA QUOTE;
006700                 WHEN COGNOME (IND-UTENTE) = RICERCA
006800                     DISPLAY "IL COGNOME ", RICERCA,
006900                             "SI TROVA NELLA ",
006950                             "POSIZIONE ",
007000                             IND-UTENTE.
007100*
```

Nell'esempio sono evidenziate le righe in cui si dichiara la tabella e quelle che eseguono la scansione. Si deve osservare che prima dell'istruzione '**SEARCH**', l'indice deve essere collocato manualmente nella posizione iniziale.

L'esempio seguente mostra una variante del programma già descritto, in cui si vuole eseguire una ricerca binaria. Perché la ricerca possa avere successo, la tabella deve essere dichiarata con una dimensione variabile di elementi, inoltre non è più necessario impostare il valore iniziale dell'indice, prima della scansione.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SEARCH-KEY.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-12.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 01  RECORD-UTENTI.
001400     02  UTENTE           OCCURS 1 TO 60 TIMES
001500                           DEPENDING ON UTENTI-MAX
001600                           ASCENDING KEY IS COGNOME
001700                           INDEXED BY IND-UTENTE.
001800         03  COGNOME      PIC X(30).
001900         03  NOME         PIC X(30).
002000         03  NOTA         PIC X(200).
002100 77  UTENTI-MAX           USAGE IS INDEX.
002200 77  EOJ                  PIC 9   VALUE ZERO.
```



```
002300 77  RISPOSTA             PIC XX.
002400 77  RICERCA              PIC X(30).
002500*
002600 PROCEDURE DIVISION.
002700*------------------------ LIVELLO 0 -----------------
002800 MAIN.
002900     PERFORM INSERIMENTO-DATI
003000             VARYING IND-UTENTE FROM 1 BY 1
003100             UNTIL EOJ = 1.
003200     MOVE 0 TO EOJ.
003300     PERFORM SCANSIONE UNTIL EOJ = 1.
003400*
003500     STOP RUN.
003600*------------------------ LIVELLO 1 -----------------
003700 INSERIMENTO-DATI.
003800     MOVE IND-UTENTE TO UTENTI-MAX.
003900     DISPLAY IND-UTENTE, " INSERISCI IL COGNOME: ".
004000     ACCEPT COGNOME (IND-UTENTE).
004100     DISPLAY IND-UTENTE, " INSERISCI IL NOME: ".
004200     ACCEPT NOME (IND-UTENTE).
004300     DISPLAY IND-UTENTE,
004350             " INSERISCI UNA NOTA DESCRITTIVA: ".
004400     ACCEPT NOTA (IND-UTENTE).
004500*
004600     IF IND-UTENTE >= 60
004700       THEN
004800           MOVE 1 TO EOJ;
004900       ELSE
005000           DISPLAY "VUOI CONTINUARE? SI O NO",
005100           ACCEPT RISPOSTA;
005200           IF RISPOSTA = "SI"
005300             THEN
005400                 NEXT SENTENCE;
005500             ELSE
005600                 MOVE 1 TO EOJ.
005700*-------------------------------------------------------
005800 SCANSIONE.
005900     DISPLAY "INSERISCI IL COGNOME DA CERCARE:".
006000     ACCEPT RICERCA.
006100     IF RICERCA = SPACES
006200       THEN
006300           MOVE 1 TO EOJ;
006400       ELSE
006600           SEARCH ALL UTENTE
006700                 AT END
006800                       DISPLAY "IL COGNOME CERCATO ",
006900                               "NON SI TROVA NELLA ",
006950                               "TABELLA: ",
007000                               QUOTE RICERCA QUOTE;
007100                 WHEN COGNOME (IND-UTENTE) = RICERCA
007200                     DISPLAY "IL COGNOME ", RICERCA,
007300                             "SI TROVA NELLA ",
007350                             "POSIZIONE ",
007400                             IND-UTENTE.
007500*
```

La ricerca binaria richiede che gli elementi della tabella siano ordinati in base alla chiave primaria; pertanto, si presume che l'inserimento dei cognomi avvenga tenendo conto dell'ordine lessicografico.

## 72.8 Nomi di condizione, raggruppamento e qualificazione

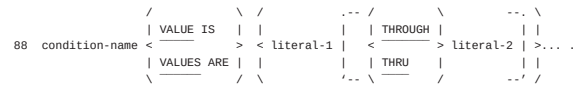
Per rappresentare in modo immediato una condizione che verifichi la corrispondenza tra il contenuto di una variabile e un insieme di valori prestabiliti, si possono utilizzare i ***nomi di condizione***. I nomi di condizione sono sostanzialmente delle costanti che descrivono un'espressione condizionale di questo tipo e si dichiarano con il numero di livello 88.

Le variabili possono essere raggruppate diversamente, purché si trovino all'interno di una stessa variabile strutturata e siano adiacenti.

I nomi dati alle variabili e ad altri oggetti del programma, in certe situazioni possono richiedere la «qualificazione», che consiste nello specificare il contesto al quale si riferiscono.

### 72.8.1 Nomi di condizione

Attraverso il livello 88 è possibile definire una costante speciale, con lo scopo di rappresentare la possibilità che una variabile contenga certi valori. In pratica, si dichiara una variabile, con i livelli da 01 a 49 o anche con il livello 77. A questa variabile si abbinano una o più costanti dichiarate con il livello 88, che descrivono la possibilità che la variabile a cui si riferiscono contenga un certo insieme di valori. Le costanti di livello 88, dichiarate in questo modo, si usano poi nelle condizioni.




Nello schema sintattico si vede soltanto la dichiarazione del livello 88, ma si deve tenere conto che si tratta di istruzioni che vanno collocate a partire dalla riga successiva a quella di dichiarazione della variabile a cui si riferiscono.

La parola chiave '**THROUGH**' o '**THRU**' si usa per specificare un intervallo di valori (dalla costante ***literal-1*** a ***literal-2***. Il valore singolo o l'intervallo di valori, può essere seguito da altri.

Nell'esempio seguente, si vede la dichiarazione della variabile '**VERNICIATURA**' che può contenere una stringa di 30 caratteri (alfanumerica). A questa variabile si associano due nomi di condizione, '**VERNICE-CHIARA**' e '**VERNICE-SCURA**', che servono a definire due gruppi di colori, descritti per nome. Da questo si intuisce che nella parte procedurale del programma venga attribuito alla variabile '**VERNICIATURA**' il nome di un colore (scritto con lettere maiuscole); poi, per verificare il tipo di colore si può usare uno di questi nomi di condizione, per esprimere il fatto che la variabile contenga uno dei nomi del gruppo a cui quel nome fa riferimento.

```
000000 01  VERNICIATURA     PIC X(30).
000000     88  VERNICE-CHIARA   "ARANCIO", "GIALLO", "VERDE",
000000                          "AZZURRO",  "GRIGIO",  "BIANCO".
000000     88  VERNICE-SCURA    "NERO", "MARRONE", "ROSSO",
000000                          "BLU",  "VIOLA".
```

L'esempio seguente descrive la variabile '**CODICE**' che può contenere una sola cifra numerica. A questa variabile si associano dei nomi di condizione, che descrivono raggruppamenti diversi delle cifre che possono essere contenute nella variabile.

```
000000     02  CODICE           PIC 9.
000000         88  PARI         0, 2, 4, 6, 8.
000000         88  DISPARI      1, 3, 5, 7, 9.
000000         88  BASSO        0 THRU 4.
000000         88  ALTO         5 THRU 9.
```

Nella divisione '**PROCEDURE DIVISION**' potrebbero apparire righe come quelle successive, per verificare che la variabile '**CODICE**' contenga certi valori:

```
000000     IF PARI
000000     THEN
000000         PERFORM …;
000000     ELSE
000000         …
```

In questo modo si evita di scrivere un'espressione condizionale complessa come nell'esempio seguente:

```
000000     IF CODICE IS EQUAL TO 0
000000        OR CODICE IS EQUAL TO 2
000000        OR CODICE IS EQUAL TO 4
000000        OR CODICE IS EQUAL TO 6
000000        OR CODICE IS EQUAL TO 8
000000     THEN
000000         PERFORM …
000000     ELSE
000000         …
```

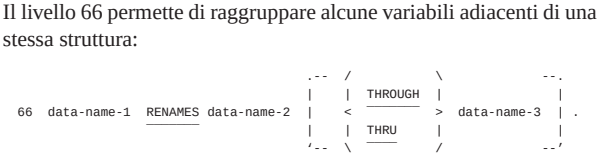
I nomi di condizione si possono associare a variabili che hanno un contenuto alfabetico, alfanumerico e numerico, ma nell'ultimo caso, deve trattarsi di valori rappresentati in forma di stringhe di caratteri (pertanto sono escluse le rappresentazioni compatte che usano quattro bit per cifra e quelle binarie).

Un nome di condizione può essere associato a una variabile che costituisce una tabella, oppure che è contenuta in una struttura tabellare. Per fare riferimento al nome di condizione nella cella giusta, occorre utilizzare gli indici, così come si fa normalmente in queste situazioni.



### 72.8.2 Raggruppamento

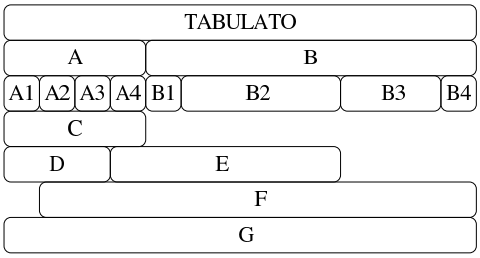
Il livello 66 permette di raggruppare alcune variabili adiacenti di una stessa struttura:




Dopo il numero di livello 66 viene dichiarato il nome del raggruppamento, quindi, dopo la parola chiave '**RENAMES**', si indica il nome della prima variabile; eventualmente, dopo la parola chiave '**THROUGH**' o '**THRU**', si indica l'ultima variabile da riunire. Si osservi l'esempio seguente:

```
000000 01  TABULATO.
000000     05  A.
000000         10  A1  PIC X.
000000         10  A2  PIC XXX.
000000         10  A3  PIC XX.
000000         10  A4  PIC XX.
000000     05  B.
000000         10  B1  PIC XX.
000000         10  B2  PIC X(16).
000000         10  B3  PIC X(8).
000000         10  B4  PIC X.
000000 66  C RENAMES A.
000000 66  D RENAMES A1 THRU A3.
000000 66  E RENAMES A4 THRU B2.
000000 66  F RENAMES A2 THRU B.
000000 66  G RENAMES A  THRU B.
```

Il disegno successivo, anche se non proporzionato rispetto alla dimensione delle variabili, mostra a cosa si riferiscono i vari raggruppamenti:




### 72.8.3 Qualificazione

I nomi usati nel linguaggio COBOL, per le variabili e per la delimitazione di porzioni del contenuto della divisione '**PROCEDURE DIVISION**' hanno valore per tutta l'estensione del programma; pertanto, in generale, non possono esserci nomi doppi e non si creano ambiti di funzionamento ristretti.

Dal momento che i nomi usati per dichiarare le variabili, oltre che le sezioni e i paragrafi della divisione '**PROCEDURE DIVISION**', hanno una lunghezza limitata normalmente a soli 30 caratteri, in un programma abbastanza grande si può porre il problema di riuscire a scrivere nomi abbastanza mnemonici e univoci. Per questo, è possibile riutilizzare certi nomi, purché poi questi siano indicati attraverso la qualificazione del loro contesto. Il contesto che consente di qualificare un nome è costituito da una gerarchia di nomi; si osservi l'esempio seguente:

```
001700 FILE SECTION.
001800 FD  SALES-FILE
001830     LABEL RECORD IS STANDARD
001860     VALUE OF FILE-ID IS "sales".
001900 01  SALES-RECORD.
002000     05  VENDOR-NAME      PIC X(20).
002100     05  VALUE            PIC S9(6).
002200     05  NUMBER           PIC X(13).
002300     05  TYPE             PIC X.
002400     05  VENDOR-REGION    PIC X(17).
002500     05  VENDOR-CITY      PIC X(20).
002600     05  COMMENTS         PIC X(60).
```

Si vede la dichiarazione del file '**SALES-FILE**', dove la variabile '**SALES-RECORD**' ne rappresenta il record, che a sua volta è suddiviso in una serie di campi. La variabile '**TYPE**', di questo esempio, appartiene gerarchicamente alla variabile '**SALES-RECORD**', che a sua volta appartiene al file '**SALES-FILE**'.

Supponendo che in questo programma, per qualche ragione, ci sia un'altra variabile con il nome '**TYPE**', si potrebbe individuare quella abbinata all'esempio specificando che si tratta della variabile '**TYPE**' di '**SALES-RECORD**'; ma volendo supporre che ci siano anche diverse variabili '**SALES-RECORD**', contenenti un campo denominato '**TYPE**', occorrerebbe indicare la variabile '**TYPE**' di '**SALES-RECORD**' di '**SALES-FILE**'.

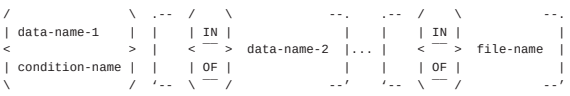
Nella divisione '**PROCEDURE DIVISION**' può succedere qualcosa di simile, quando si usa una suddivisione delle procedure in sezioni e paragrafi. In questo modo, i nomi dei paragrafi potrebbero richiedere una qualificazione, specificando a quale sezione appartengono.

Ciò che consente di qualificare un nome, in modo sufficiente a renderlo univoco, è il ***qualificatore***.

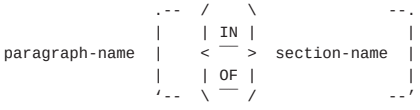
Quando nella divisione '**PROCEDURE DIVISION**' si usa un nome che richiede una qualificazione, si usa indifferentemente la parola chiave '**IN**' oppure '**OF**':

```
… TYPE IN SALES-RECORDS …
… TYPE IN SALES-RECORDS OF SALES-FILE …
```

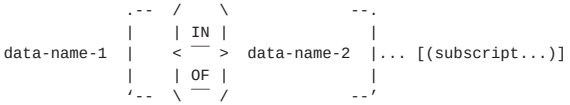
Segue lo schema sintattico per qualificare una variabile:

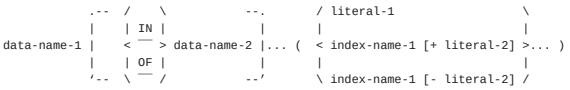



Segue lo schema sintattico per qualificare un paragrafo all'interno della divisione '**PROCEDURE DIVISION**':




Quando si deve usare la qualificazione per individuare un elemento di una tabella, gli indici tra parentesi tonde appaiono alla fine, dopo la qualificazione stessa. Seguono due modelli sintattici alternativi che descrivono il modo di rappresentare un elemento di una tabella, tenendo conto anche della qualificazione:




Le regole che stabiliscono la possibilità o meno di usare la qualificazione, sono intuitive e non vengono descritte qui nel dettaglio. La regola generale da seguire è quella della leggibilità del programma



sorgente, che richiede di usare la qualificazione solo quando questa è utile per migliorare la chiarezza dello stesso.

## 72.9 Modello di definizione della variabile

Il COBOL gestisce soltanto due tipi di dati: numerici e alfanumerici. Tuttavia, nel momento in cui si dichiara il modello, le variabili scalari (ovvero quelle che non sono scomposte) si distinguono in classi e in categorie, secondo lo schema che si vede nella tabella successiva.

Tabella 72.109. Classi e categorie delle variabili COBOL.

| Livello di suddivisione | Classe | Categoria |
|---|---|---|
| variabile scalare (non suddivisibile) | numerica | numerica |
| | alfanumerica | numerica *edited* |
| | | alfanumerica *edited* |
| | | alfanumerica |
| variabile strutturata (*record*) | alfanumerica | alfanumerica |

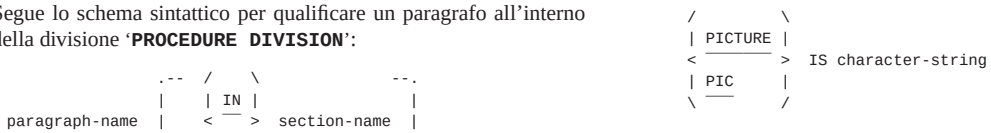
Le variabili che appartengono alla ***classe numerica*** contengono dei valori che si possono utilizzare per eseguire delle espressioni numeriche; mentre quelle che appartengono alla ***classe alfanumerica*** si possono utilizzare soltanto come sequenze di caratteri.

Si osservi che una variabile strutturata è sempre solo alfanumerica, ma ciò non toglie che i campi che contiene possano essere di una classe e di una categoria differente.

Le variabili che appartengono alle categorie *edited*, «modificate», servono per ricevere un valore, numerico o alfanumerico, da modificare nell'atto stesso della ricezione. Una variabile di questo tipo fa sempre parte della classe alfanumerica, perché, una volta ricevuti i dati e modificati in base al modello di definizione della variabile, questi sono semplicemente una sequenza di caratteri senza più alcun valore numerico.

### 72.9.1 Dichiarazione del modello di definizione della variabile

Il modello di definizione della variabile è introdotto dalla parola chiave '**PICTURE**', o '**PIC**':




La metavariabile ***character-string*** costituisce il modello vero e proprio, che può essere composto da un massimo di 30 caratteri, anche se in questo modo si possono comunque rappresentare modelli di variabili di dimensioni molto più grandi.

Si osservi che l'indicazione di un modello di definizione della variabile è obbligatorio per tutte le variabili scalari (pertanto riguarda potenzialmente i livelli da 01 a 49 e 77), mentre non è consentito per le variabili strutturate.

La stringa che costituisce il modello di definizione della variabile è composta di simboli che descrivono ognuno le caratteristiche di un carattere, di una cifra, di un segno, oppure inseriscono qualcosa che nel dato originale è assente, ma spesso possono essere ripetuti mettendo un numero intero tra parentesi:

***x*(*n*)**

Una notazione del genere indica che il simbolo ***x*** va inteso come se fosse ripetuto ***n*** volte; per esempio, '**9(3)V9(5)**' è equivalente a '**999V99999**'. È attraverso questo meccanismo che si possono de-

scrivere variabili molto grandi, pur avendo un modello di definizione limitato a soli 30 caratteri.

Nelle sezioni successive vengono descritti i simboli utilizzabili per i modelli di definizione delle variabili scalari. Lo standard del linguaggio COBOL annovera più possibilità di quelle indicate, ma i compilatori non si comportano sempre allo stesso modo e spesso ignorano l'uso di alcuni simboli, oppure li interpretano in modo inesatto. Questo problema riguarda soprattutto i simboli che applicano delle modifiche al valore ricevuto nella variabile, che quindi vanno usati con estrema parsimonia se si vuole scrivere un programma abbastanza compatibile.

### 72.9.2 Variabili alfanumeriche

Sono variabili alfanumeriche pure e semplici quelle che possono rappresentare qualunque carattere delle codifica prevista, limitatamente al fatto che la rappresentazione è in forma di byte (ASCII o EBCDIC). Il modello di definizione di una variabile alfanumerica prevede simboli speciali che limitano l'inserimento di soli caratteri alfabetici o di sole cifre numeriche, ma in pratica, succede spesso che i compilatori non applichino alcuna restrizione.

Tabella 72.111. Simboli del modello di definizione di una variabile alfanumerica.

| Simbolo | Descrizione |
|---|---|
| `9` | Rappresenta una cifra numerica. |
| `A` | Rappresenta una lettera dell'alfabeto latino, non accentata, maiuscola o minuscola. |
| `X` | Rappresenta un carattere qualsiasi. |

Si osservi che il modello di definizione di una variabile alfanumerica deve contenere almeno un simbolo '**X**' o '**A**', altrimenti, un modello che contenga soltanto il simbolo '**9**' verrebbe interpretato come numerico.

A titolo di esempio, viene mostrato un piccolo programma con tre variabili scalari alfanumeriche, aventi modelli diversi, abbinate ognuna a una variabile strutturata. Alle variabili scalari viene assegnato lo stesso valore, in modo da poter confrontare come questo valore viene inteso e rappresentato.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   PICTURE-ALPHANUMERIC.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-23.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-A.
001200     02  A        PICTURE X(15)        USAGE IS DISPLAY.
001300 01  RECORD-B.
001400     02  B        PICTURE 9(5)A(5)X(5) USAGE IS DISPLAY.
001500 01  RECORD-C.
001600     02  C        PICTURE A(5)X(5)9(5) USAGE IS DISPLAY.
001700*
001800 PROCEDURE DIVISION.
001900*
002000 MAIN.
002100     MOVE " 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO A.
002200     MOVE " 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO B.
002300     MOVE " 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO C.
002400     DISPLAY "SOURCE VALUE IS: ", QUOTE,
002500             " 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ", QUOTE.
002600     DISPLAY "PICTURE: X(15)         VALUE: ", QUOTE, A, QUOTE,
002700             "  DATA: ", QUOTE, RECORD-A, QUOTE.
002800     DISPLAY "PICTURE: 9(5)A(5)X(5)  VALUE: ", QUOTE, B, QUOTE,
002900             "  DATA: ", QUOTE, RECORD-B, QUOTE.
003000     DISPLAY "PICTURE: A(5)X(5)9(5)  VALUE: ", QUOTE, C, QUOTE,
003100             "  DATA: ", QUOTE, RECORD-C, QUOTE.
003200*
```

Compilando il programma con TinyCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente, dove si può vedere che l'uso dei simboli '**A**' e '**9**' non comporta alcuna differenza di funzionamento rispetto a '**X**'; tuttavia, un compilatore più sofisticato potrebbe segnalare qualche tipo di errore:

```
SOURCE VALUE IS: " 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ"
PICTURE: X(15)         VALUE: " 1234567890ABCD"  DATA: " 1234567890ABCD"
PICTURE: 9(5)A(5)X(5)  VALUE: " 1234567890ABCD"  DATA: " 1234567890ABCD"
PICTURE: A(5)X(5)9(5)  VALUE: " 1234567890ABCD"  DATA: " 1234567890ABCD"
```

### 72.9.3 Variabili alfanumeriche modificate

Il modello di definizione di una variabile alfanumerica può contenere simboli che applicano una modifica al valore stesso. La modifica avviene nel momento in cui il valore viene ricevuto.

Tabella 72.114. Simboli del modello di definizione di una variabile alfanumerica che descrivono una modifica del valore ricevuto.

| Simbolo | Descrizione |
|---|---|
| `B` | Richiede l'inserimento di uno spazio. |
| `0` | Richiede l'inserimento di uno zero. |
| `/` | Se il compilatore lo permette, inserisce una barra obliqua. |
| `,` | Se il compilatore lo permette, inserisce una virgola. Si osservi comunque che non può apparire la virgola come simbolo conclusivo del modello, perché in tal caso assumerebbe il significato di un delimitatore. |
| | Teoricamente, ogni altro simbolo che non abbia un significato preciso per la realizzazione dei modelli di definizione delle variabili, dovrebbe essere aggiunto tale e quale; in pratica, molto dipende dalle caratteristiche del compilatore. |

A titolo di esempio, viene mostrato un piccolo programma con due variabili scalari alfanumeriche, aventi modelli diversi, abbinate ognuna a una variabile strutturata. Alle variabili scalari viene assegnato lo stesso valore, in modo da poter confrontare come questo viene inteso e rappresentato. Nell'esempio si tenta in particolare di inserire in un modello una barra obliqua e una virgola.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   PICTURE-ALPHANUMERIC-EDITED.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-23.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-A.
001200     02  A        PICTURE X(15)           USAGE IS DISPLAY.
001300 01  RECORD-B.
001400     02  B        PICTURE ABX09,/X(8) USAGE IS DISPLAY.
001500*
001600 PROCEDURE DIVISION.
001700*
001800 MAIN.
001900     MOVE "ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO A.
002000     MOVE "ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO B.
002100     DISPLAY "SOURCE VALUE IS: ", QUOTE,
002200             "ABCDEFGHIJKLMNOPQRSTUVWXYZ", QUOTE.
002300     DISPLAY "PICTURE: X(15)        VALUE: ", QUOTE, A, QUOTE,
002400             "  DATA: ", QUOTE, RECORD-A, QUOTE.
002500     DISPLAY "PICTURE: ABX09,/X(8)  VALUE: ", QUOTE, B, QUOTE,
002600             "  DATA: ", QUOTE, RECORD-B, QUOTE.
002700*
```

Compilando il programma con TinyCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente:

```
SOURCE VALUE IS: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
PICTURE: X(15)        VALUE: "ABCDEFGHIJKLMNO"  DATA: "ABCDEFGHIJKLMNO"
PICTURE: ABX09,/X(8)  VALUE: "A B0C,/DEFGHIJK"  DATA: "A B0C,/DEFGHIJK"
```

### 72.9.4 Variabili numeriche

Le variabili numeriche pure e semplici, sono quelle il cui valore può essere usato per calcolare delle espressioni numeriche. Nel modello di definizione di una variabile di questo tipo, possono apparire solo simboli che descrivono la dimensione del valore rappresentabile, distinguendo la parte intera da quella decimale e specificando eventualmente la presenza del segno.

Tabella 72.117. Simboli del modello di definizione di una variabile numerica.

| Simbolo | Descrizione |
|---|---|
| `9` | Rappresenta una cifra numerica singola. |
| `V` | Rappresenta la separazione tra la parte intera e la parte decimale del numero e di conseguenza può apparire una sola volta nel modello. Questo simbolo non incrementa la dimensione della variabile, in quanto serve solo a sapere dove si intende che debba trovarsi la separazione tra la parte intera e quella decimale, ai fini dei calcoli che si possono eseguire. Se questo simbolo appare alla fine del modello, è come se non fosse inserito, perché la sua presenza non aggiunge alcuna informazione. |
| `S` | Se si utilizza questo simbolo, può essere collocato soltanto all'estrema sinistra del modello, allo scopo di specificare che il numero è provvisto di segno. A seconda del modo in cui l'informazione numerica viene codificata nella variabile (‘**`DISPLAY`**’, ‘**`COMPUTATIONAL`**’ e altre opzioni eventuali), il segno può occupare una posizione separata oppure no. |
| `P` | Il simbolo ‘**P**’ può essere usato soltanto alla sinistra o alla destra del modello e può apparire ripetuto, tenendo conto che se appare alla sinistra è incompatibile con il simbolo ‘**V**’. Questo simbolo non incrementa la dimensione della variabile, ma serve a far sì che il valore contenuto nella variabile sia interpretato in modo differente. **L'uso di questo simbolo è sconsigliabile**, perché altera il comportamento della variabile in un modo che non è facile da interpretare correttamente. |

A titolo di esempio, viene mostrato un piccolo programma con cinque variabili scalari numeriche, aventi modelli diversi, abbinate ognuna a una variabile strutturata. Alle variabili scalari viene assegnato lo stesso valore, in modo da poter confrontare come questo valore viene inteso e rappresentato. Nell'esempio appare anche l'uso del simbolo ‘**P**’ a dimostrazione della difficoltà che questo comporta nell'interpretare l'esito degli assegnamenti alle variabili.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   PICTURE-NUMERIC.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-22.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-A.
001200     02  A        PICTURE 99999       USAGE IS DISPLAY.
001300 01  RECORD-B.
001400     02  B        PICTURE 999V99      USAGE IS DISPLAY.
001500 01  RECORD-C.
001600     02  C        PICTURE S999V99     USAGE IS DISPLAY.
001700 01  RECORD-D.
001800     02  D        PICTURE 999V99PP    USAGE IS DISPLAY.
001900 01  RECORD-E.
002000     02  E        PICTURE PP99999     USAGE IS DISPLAY.
002100*
002200 PROCEDURE DIVISION.
002300*
002400 MAIN.
002500     MOVE -1234.5678 TO A.
002600     MOVE -1234.5678 TO B.
002700     MOVE -1234.5678 TO C.
002800     MOVE -1234.5678 TO D.
002900     MOVE -1234.5678 TO E.
```



```
003000     DISPLAY "SOURCE VALUE IS -1234.5678".
003100     DISPLAY "PICTURE: 99999     VALUE: ", A,
003200             "     DATA: ", RECORD-A.
003300     DISPLAY "PICTURE: 999V99    VALUE: ", B,
003400             "    DATA: ", RECORD-B.
003500     DISPLAY "PICTURE: S999V99   VALUE: ", C,
003600             "   DATA: ", RECORD-C.
003700     DISPLAY "PICTURE: 999V99PP  VALUE: ", D,
003800             "  DATA: ", RECORD-D.
003900     DISPLAY "PICTURE: PP99999   VALUE: ", E,
004000             "  DATA: ", RECORD-E.
004100     STOP RUN.
004200*
```

Compilando il programma con TinyCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente:

```
SOURCE VALUE IS -1234.5678
PICTURE: 99999     VALUE: 01234     DATA: 01234
PICTURE: 999V99    VALUE: 234.56    DATA: 23456
PICTURE: S999V99   VALUE: -234.56   DATA: 2345O
PICTURE: 999V99PP  VALUE: 004.5678  DATA: 45678
PICTURE: PP99999   VALUE: .0078000  DATA: 78000
```

Facendo la stessa cosa con OpenCOBOL:

```
SOURCE VALUE IS -1234.5678
PICTURE: 99999     VALUE: 01234     DATA: 01234
PICTURE: 999V99    VALUE: 234.56    DATA: 23456
PICTURE: S999V99   VALUE: -234.56   DATA: 2345v
PICTURE: 999V99PP  VALUE: 004.5678  DATA: 45678
PICTURE: PP99999   VALUE: .0078000  DATA: 78000
```

Si osservi che nell'esempio le variabili scalari numeriche sono state dichiarate con l'opzione ‘**`USAGE IS DISPLAY`**’, che comunque sarebbe stata implicita, in modo da assicurare la visibilità del contenuto leggendo il livello 01.

### 72.9.5 Variabili numeriche modificate

Il modello di definizione di una variabile fatta per ricevere un valore numerico, può contenere simboli che applicano una modifica all'apparenza del valore. Nel momento in cui una variabile del genere riceve un valore, ciò che la variabile fornisce è un'informazione alfanumerica, che non può più essere usata per elaborazioni matematiche; al massimo, una variabile del genere può ricevere il risultato di un'espressione che generi un valore numerico.

Tabella 72.121. Alcuni simboli del modello di definizione di una variabile numerica che descrivono una trasformazione in formato alfanumerico.

| Simbolo | Descrizione |
|---|---|
| `B` | Richiede l'inserimento di uno spazio. |
| `0` | Richiede l'inserimento di uno zero. |
| `Z` | Si usa soltanto nella parte sinistra del modello e richiede di indicare uno spazio al posto di una cifra zero. |
| `*` | Si usa soltanto nella parte sinistra del modello e richiede di indicare un asterisco al posto di una cifra zero. |
| `'` | In condizioni normali inserisce una virgola letterale. Si osservi che non può apparire la virgola come simbolo conclusivo del modello, perché in tal caso assumerebbe il valore di un delimitatore. |
| `.` | In condizioni normali rappresenta il punto di separazione tra la parte intera e la parte decimale. Si osservi che in un modello che prevede la modifica, non si può usare il simbolo ‘**V**’. |
| `+` `-` | Il segno ‘**+**’ o il segno ‘**-**’ indicano la posizione in cui si vuole appaia il segno del numero; se il segno viene ripetuto, le occorrenze successive alla prima rappresentano implicitamente la posizione di una cifra numerica, con soppressione degli zeri anteriori. Usando il segno ‘**-**’ si dovrebbe ottenere la rappresentazione di un numero senza segno, quando questo ha un valore positivo. |

| Simbolo | Descrizione |
|---|---|
| `CR`<br><br>`DB` | Questi simboli, che occupano due caratteri, vengono inseriti tali e quali nella rappresentazione della variabile. Sono utili queste sigle per la contabilità dei paesi di lingua inglese (*credit*, *debit*); a ogni modo è bene ricordare che non sono influenzate dal segno del valore numerico che viene ricevuto nella variabile. |
| `$` | Si usa per mostrare il simbolo del dollaro, quando la valuta prevista nel sorgente del programma è quella predefinita. |
| `/` | Se il compilatore lo permette, inserisce una barra obliqua. |
| | Teoricamente, ogni altro simbolo che non abbia un significato preciso per la realizzazione dei modelli di definizione delle variabili, dovrebbe essere aggiunto tale e quale; in pratica, molto dipende dalle caratteristiche del compilatore. |

Come esempio viene mostrato un piccolo programma con alcune variabili scalari numeriche modificate (*edited*), aventi modelli diversi, abbinate ognuna a una variabile strutturata. Alle variabili scalari viene assegnato lo stesso valore, in modo da poter confrontare come questo valore viene inteso e rappresentato.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   PICTURE-NUMERIC-EDITED.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-A.
001200     02  A        PICTURE S9(10)V9(5)   USAGE IS DISPLAY.
001300 01  RECORD-B.
001400     02  B        PICTURE +Z(9)9.9(5)   USAGE IS DISPLAY.
001500 01  RECORD-C.
001600     02  C        PICTURE CR+Z(7)9.9(5) USAGE IS DISPLAY.
001700 01  RECORD-D.
001800     02  D        PICTURE +Z(7)9.9(5)DB USAGE IS DISPLAY.
001900 01  RECORD-E.
002000     02  E        PICTURE *(9)9.9(5)+   USAGE IS DISPLAY.
002100 01  RECORD-F.
002200     02  F        PICTURE +*(9)9.9(4)$  USAGE IS DISPLAY.
002300 01  RECORD-G.
002400     02  G        PICTURE +*(9)9,9(4)$  USAGE IS DISPLAY.
002500 01  RECORD-H.
002600     02  H        PICTURE -(10)9,9(4)$  USAGE IS DISPLAY.
002700 01  RECORD-I.
002800     02  I        PICTURE +(10)9,9(4)$  USAGE IS DISPLAY.
002900*
003000 PROCEDURE DIVISION.
003100*
003200 MAIN.
003300     MOVE +123456.789 TO A.
003400     MOVE +123456.789 TO B.
003500     MOVE +123456.789 TO C.
003600     MOVE +123456.789 TO D.
003700     MOVE +123456.789 TO E.
003800     MOVE +123456.789 TO F.
003900     MOVE +123456.789 TO G.
004000     MOVE +123456.789 TO H.
004100     MOVE +123456.789 TO I.
004200     DISPLAY "SOURCE VALUE IS: +123456.789".
004300     DISPLAY "PICTURE: S9(10)V9(5)    VALUE: ", A,
004400             "  DATA: ", RECORD-A.
004500     DISPLAY "PICTURE: +Z(9)9.9(5)    VALUE: ", B,
004600             "  DATA: ", RECORD-B.
004700     DISPLAY "PICTURE: CR+Z(7)9.9(5)  VALUE: ", C,
004800             "  DATA: ", RECORD-C.
004900     DISPLAY "PICTURE: +Z(7)9.9(5)DB  VALUE: ", D,
005000             "  DATA: ", RECORD-D.
005100     DISPLAY "PICTURE: *(9)9.9(5)+    VALUE: ", E,
005200             "  DATA: ", RECORD-E.
005300     DISPLAY "PICTURE: +*(9)9.9(4)$   VALUE: ", F,
005400             "  DATA: ", RECORD-F.
005500     DISPLAY "PICTURE: +*(9)9,9(4)$   VALUE: ", G,
005600             "  DATA: ", RECORD-G.
005700     DISPLAY "PICTURE: -(10)9,9(4)$   VALUE: ", H,
005800             "  DATA: ", RECORD-H.
```



```
005900     DISPLAY "PICTURE: +(10)9,9(4)$   VALUE: ", I,
006000             "  DATA: ", RECORD-I.
006100     STOP RUN.
006200*
```

Compilando il programma con TinyCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente:

```
SOURCE VALUE IS: +123456.789
PICTURE: S9(10)V9(5)    VALUE:  0000123456.78900  DATA: 00001234567890{
PICTURE: +Z(9)9.9(5)    VALUE: +    123456.78900  DATA: +    123456.78900
PICTURE: CR+Z(7)9.9(5)  VALUE: CR+  123456.78900  DATA: CR+  123456.78900
PICTURE: +Z(7)9.9(5)DB  VALUE: +  123456.78900DB  DATA: +  123456.78900DB
PICTURE: *(9)9.9(5)+    VALUE: ****123456.78900+  DATA: ****123456.78900+
PICTURE: +*(9)9.9(4)$   VALUE: +****123456.7890$  DATA: +****123456.7890$
PICTURE: +*(9)9,9(4)$   VALUE: +********12,3456$  DATA: +********12,3456$
PICTURE: -(10)9,9(4)$   VALUE:          12,3456$  DATA:          12,3456$
PICTURE: +(10)9,9(4)$   VALUE:         +12,3456$  DATA:         +12,3456$
```

Tra i vari risultati, si può osservare che la virgola è stata interpretata come un segno senza un ruolo preciso, pertanto si colloca semplicemente prima delle ultime quattro cifre, secondo la previsione del modello.

Intervenendo nella sezione '**`CONFIGURATION SECTION`**' è possibile invertire il ruolo del punto e della virgola, nella rappresentazione dei numeri; nello stesso modo, è possibile attribuire un simbolo differente per la valuta. L'esempio seguente è una variante di quello appena mostrato, con le modifiche necessarie per questo scopo. Si osservi che come simbolo di valuta è stata scelta la lettera «E».

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PICTURE-NUMERIC-EDITED-BIS.
000300 AUTHOR.        DANIELE GIACOMINI.
000400 DATE-WRITTEN.  2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SPECIAL-NAMES. DECIMAL-POINT IS COMMA
000900                CURRENCY SIGN IS "E".
001000*
001100 DATA DIVISION.
001200*
001300 WORKING-STORAGE SECTION.
001400 01  RECORD-A.
001500     02  A        PICTURE S9(10)V9(5)   USAGE IS DISPLAY.
001600 01  RECORD-B.
001700     02  B        PICTURE +Z(9)9.9(5)   USAGE IS DISPLAY.
001800 01  RECORD-C.
001900     02  C        PICTURE CR+Z(7)9.9(5) USAGE IS DISPLAY.
002000 01  RECORD-D.
002100     02  D        PICTURE +Z(7)9.9(5)DB USAGE IS DISPLAY.
002200 01  RECORD-E.
002300     02  E        PICTURE *(9)9.9(5)+   USAGE IS DISPLAY.
002400 01  RECORD-F.
002500     02  F        PICTURE +*(9)9.9(4)E  USAGE IS DISPLAY.
002600 01  RECORD-G.
002700     02  G        PICTURE +*(9)9,9(4)E  USAGE IS DISPLAY.
002800 01  RECORD-H.
002900     02  H        PICTURE -(10)9,9(4)E  USAGE IS DISPLAY.
003000 01  RECORD-I.
003100     02  I        PICTURE +(10)9,9(4)E  USAGE IS DISPLAY.
003200*
003300 PROCEDURE DIVISION.
003400*
003500 MAIN.
003600     MOVE +123456,789 TO A.
003700     MOVE +123456,789 TO B.
003800     MOVE +123456,789 TO C.
003900     MOVE +123456,789 TO D.
004000     MOVE +123456,789 TO E.
004100     MOVE +123456,789 TO F.
004200     MOVE +123456,789 TO G.
004300     MOVE +123456,789 TO H.
004400     MOVE +123456,789 TO I.
004500     DISPLAY "SOURCE VALUE IS: +123456.789".
004600     DISPLAY "PICTURE: S9(10)V9(5)    VALUE: ", A,
004700             "  DATA: ", RECORD-A.
004800     DISPLAY "PICTURE: +Z(9)9.9(5)    VALUE: ", B,
004900             "  DATA: ", RECORD-B.
005000     DISPLAY "PICTURE: CR+Z(7)9.9(5)  VALUE: ", C,
005100             "  DATA: ", RECORD-C.
005200     DISPLAY "PICTURE: +Z(7)9.9(5)DB  VALUE: ", D,
005300             "  DATA: ", RECORD-D.
005400     DISPLAY "PICTURE: *(9)9.9(5)+    VALUE: ", E,
```

```
005500             "  DATA: ", RECORD-E.
005600     DISPLAY "PICTURE: +*(9)9.9(4)E   VALUE: ", F,
005700             "  DATA: ", RECORD-F.
005800     DISPLAY "PICTURE: +*(9)9,9(4)E   VALUE: ", G,
005900             "  DATA: ", RECORD-G.
006000     DISPLAY "PICTURE: -(10)9,9(4)E   VALUE: ", H,
006100             "  DATA: ", RECORD-H.
006200     DISPLAY "PICTURE: +(10)9,9(4)E   VALUE: ", I,
006300             "  DATA: ", RECORD-I.
006400     STOP RUN.
006500*
```

Compilando il programma con TinyCOBOL, l'avvio dell'eseguibile che si ottiene genera il risultato seguente:

```
SOURCE VALUE IS: +123456.789
PICTURE: S9(10)V9(5)    VALUE:  0000123456,78900  DATA: 00001234567890{
PICTURE: +Z(9)9.9(5)    VALUE: +         1.23456  DATA: +         1.23456
PICTURE: CR+Z(7)9.9(5)  VALUE: CR+       1.23456  DATA: CR+       1.23456
PICTURE: +Z(7)9.9(5)DB  VALUE: +       1.23456DB  DATA: +       1.23456DB
PICTURE: *(9)9.9(5)+    VALUE: ********1.23456+  DATA: ********1.23456+
PICTURE: +*(9)9.9(4)E   VALUE: ********12.3456E  DATA: ********12.3456E
PICTURE: +*(9)9,9(4)E   VALUE: +****123456,7890E  DATA: +****123456,7890E
PICTURE: -(10)9,9(4)E   VALUE:      123456,7890E  DATA:      123456,7890E
PICTURE: +(10)9,9(4)E   VALUE:     +123456,7890E  DATA:     +123456,7890E
```

Questa volta si può osservare che nel modello è il punto che perde il suo significato, apparendo nel risultato soltanto nella posizione prevista, allineando le cifre numeriche originali alla destra.

## 72.10 Note sull'utilizzo dell'insieme di caratteri universale con il COBOL

Lo standard COBOL del 1985 prevede sostanzialmente che si possano gestire informazioni alfanumeriche composte di simboli rappresentabili in byte, appartenenti pertanto al codice ASCII o EBCDIC. Con l'introduzione dell'insieme di caratteri universale, si pone il problema di gestire codifiche di tipo diverso, ben più grandi del solito byte. Purtroppo, però, le soluzioni adottate non sono semplici e lineari come nel passato.

### 72.10.1 Stringhe letterali

Le stringhe letterali che devono contenere simboli al di fuori del codice ASCII o EBCDIC, devono essere delimitate in modo differente. Generalmente sono disponibili due forme:

N"*stringa_letterale*"

NX"*stringa_esadecimale*"

La prima forma riguarda una stringa letterale composta secondo la forma codificata del carattere prevista dal compilatore (UTF-8, UTF-16 o altro); la seconda, ammesso che sia disponibile, viene espressa attraverso cifre esadecimali. Si osservi, però, che per poter esprimere una stringa del genere in forma esadecimale, occorre sapere in che modo il compilatore la interpreta, dato che dipende dalla forma codificata del carattere adottata.

### 72.10.2 modello di definizione delle variabili

Nel modello di definizione di una variabile, la lettera '**N**' rappresenta un carattere espresso secondo la codifica universale; la lettera «N» sta per *National*. Pertanto, si aggiunge anche una voce nuova all'opzione '**USAGE**': '**USAGE IS NATIONAL**'.

A seconda della forma codificata del carattere adottata dal compilatore, cambia la dimensione di una variabile del genere. Se si utilizzano codifiche del tipo UTF-8, che hanno una lunghezza variabile, può diventare impossibile stabilire in anticipo la dimensione in byte corrispondente. Anche per questo motivo, è improbabile che si possa usare lo standard UTF-8 con il COBOL.



### 72.10.3 Costanti figurative

Tra le costanti figurative, '**HIGH-VALUES**' e '**LOW-VALUES**' perdono di significato, se associate a una variabile dichiarata come '**USAGE IS NATIONAL**'.

## 72.11 Divisione «PROCEDURE DIVISION»

La divisione '**PROCEDURE DIVISION**' costituisce la quarta e ultima parte di un programma sorgente COBOL. La divisione si può suddividere in paragrafi, oppure in sezioni contenenti eventualmente dei paragrafi. All'interno delle sezioni o dei paragrafi, si inseriscono le istruzioni che descrivono la procedura del programma.

Le istruzioni sono inserite a gruppi, terminanti con un punto fermo, seguito da uno spazio; le istruzioni singole, che non costituiscono un gruppo autonomo, possono essere separate graficamente attraverso dei separatori (la virgola, il punto e virgola, la parola '**THEN**').

Alcune istruzioni, quando non costituiscono un gruppo autonomo, possono collocarsi solo alla fine del gruppo. Si tratta precisamente di '**GO TO**' e di '**STOP RUN**'.

La divisione può articolarsi in tre modi diversi; quello che si vede descritto nello schema segue è il più semplice, perché non fa uso delle sezioni:

```
[PROCEDURE DIVISION.

{paragraph-name.
     [sentence]...}...
```

Se si usano le sezioni, i paragrafi devono essere contenuti tutti all'interno di sezioni:

```
[PROCEDURE DIVISION.

/ section-name SECTION [segment-number]. \
|                                        |
< {paragraph-name.                       >...
|      [sentence]...}...                 |
\                                        /
```

Eventualmente ci può essere un gruppo iniziale di sezioni speciali; in tal caso, è obbligatorio suddividere il resto del programma in sezioni:

```
[PROCEDURE DIVISION.

 DECLARATIVES.

 / section-name SECTION [segment-number]. \
 |                                        |
 |     USE statement                      |
 <                                        >...
 | {paragraph-name.                       |
 |      [sentence]...}...                 |
 \                                        /
 END DECLARATIVES.

/ section-name SECTION [segment-number]. \
|                                        |
< {paragraph-name.                       >...
|      [sentence]...}...                 |
\                                        /
```

Il primo gruppo di istruzioni a essere eseguito è quello che si trova nel primo paragrafo della prima sezione; escludendo quelli inseriti in un blocco '**DECLARATIVES**'. In condizioni normali, la sequenza dei gruppi di istruzioni eseguiti prosegue con quelli successivi, salvo quando si incontrano istruzioni speciali che richiedono esplicitamente di modificare questo tipo di flusso.

### 72.11.1 Gruppo di sezioni «DECLARATIVES»

Quando all'inizio della divisione '**PROCEDURE DIVISION**' appare la parola chiave '**DECLARATIVES**', che inizia dall'area A del modulo di programmazione, le sezioni dichiarate fino alla riga dove appare

‘**END DECLARATIVES**’ (sempre a partire dall’area A), non vengono eseguite normalmente, ma solo al verificarsi di certe condizioni.

```
DECLARATIVES.

/ section-name SECTION [segment-number]. \
|                                        |
|     USE statement                      |
<                                        >...
| {paragraph-name.                       |
|      [sentence]...}...                 |
\                                        /
END DECLARATIVES.
```

Ogni sezione di questo gruppo speciale, inizia con una o più istruzioni ‘**USE**’, prima di procedere con dei paragrafi contenenti altre istruzioni. L’istruzione ‘**USE**’ serve ad abbinare l’esecuzione della sezione (a partire dal primo dei suoi paragrafi), a condizione che si verifichi una certa condizione:

```
                                          / {file-name}... \
                                          |                |
                  /           \           | INPUT          |
                  | EXCEPTION |           |                |
USE AFTER STANDARD <           > PROCEDURE ON < OUTPUT       >
                  | ERROR     |           |                |
                  \           /           | I-O            |
                                          |                |
                                          \ EXTEND         /
```

Tenendo conto che le parole chiave ‘**EXCEPTION**’ e ‘**ERROR**’ del modello sono equivalenti, si intende che questa istruzione serve ad attivare la sezione che la contiene se si verifica una condizione di errore, che non sia stato gestito diversamente all’interno del programma, riguardante: un certo file (***file-name***), un file qualunque aperto in lettura (‘**INPUT**’), scrittura (‘**OUTPUT**’), lettura e scrittura (‘**I-O**’) o in estensione (‘**EXTEND**’).

Viene mostrato l’esempio di un piccolo programma completo, che ha lo scopo di leggere un file (‘`input.txt`’) e di mostrarne il contenuto sullo schermo:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-DECLARATIVES.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-26.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "input.txt"
001300                            ORGANIZATION IS
001350                                LINE SEQUENTIAL
001400                            FILE STATUS IS
001450                                STATO-FILE-DA-LEGGERE.
001500*
001600 DATA DIVISION.
001700*
001800 FILE SECTION.
001900*
002000 FD  FILE-DA-LEGGERE.
002100 01  RECORD-DA-LEGGERE            PIC X(79).
002200*
002300 WORKING-STORAGE SECTION.
002400 77  STATO-FILE-DA-LEGGERE        PIC XX.
002500*
002600 PROCEDURE DIVISION.
002700*
002800 DECLARATIVES.
002900 FILE-ACCESS-ERROR SECTION.
003000     USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
003100 FILE-ACCESS-ERROR-RECOVERY.
003200     DISPLAY "FILE ACCESS ERROR: ",
003250             STATO-FILE-DA-LEGGERE.
003300     STOP RUN.
003400 END DECLARATIVES.
```



```
003500*
003550 MAIN-PROCEDURE SECTION.
003600 MAIN.
003700     OPEN INPUT FILE-DA-LEGGERE.
003800     PERFORM LETTURA UNTIL 0 = 1.
003900     CLOSE FILE-DA-LEGGERE.
004000*
004100     STOP RUN.
004200*
004300 LETTURA.
004400     READ FILE-DA-LEGGERE.
004500     DISPLAY RECORD-DA-LEGGERE.
004600*
```

Si può osservare nel programma che il ciclo di lettura non termina mai, perché la condizione ‘**0 = 1**’ non si può avverare. Così facendo, dato che la lettura non prevede alcun controllo del superamento della fine del file, si verifica un errore che viene preso in considerazione dalla sezione ‘**FILE-ACCESS-ERROR**’.

Compilando il programma con OpenCOBOL, l’avvio dell’eseguibile che si ottiene genera un risultato simile a quello seguente:

```
aaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbb
ccccccccccccccccc
FILE ACCESS ERROR: 10
```

In pratica, alla fine del file termina la visualizzazione del suo contenuto e si ottiene un messaggio di errore, come organizzato nella sezione ‘**FILE-ACCESS-ERROR**’.

### 72.11.2 Sezioni e segmenti

Le sezioni della divisione ‘**PROCEDURE DIVISION**’, oltre al nome possono indicare un numero di segmento, che può andare da zero a 99.

```
section-name SECTION [segment-number].
```

Il numero di segmento serve a raggruppare tutte le sezioni con lo stesso numero in uno stesso segmento, allo scopo di sapere, quale parte del programma deve rimanere simultaneamente nella memoria centrale durante il funzionamento.

Si dividono precisamente due tipi di segmenti: quelli fissi, con numeri da 00 a 49, e quelli indipendenti, da 50 a 99. I segmenti numerati fino al numero 49 devono rimanere sempre in memoria, mentre gli altri devono esserci solo per il tempo necessario al loro funzionamento. Per questa ragione, le sezioni dichiarate nella zona ‘**DECLARATIVES**’, possono essere associate soltanto a segmenti fissi (da 00 a 49).

Naturalmente, questa possibilità di segmentare il programma dipende dal compilatore, che potrebbe limitarsi semplicemente a ignorare il numero di segmento.

### 72.11.3 Gruppi di istruzioni e istruzioni condizionali

Un gruppo di istruzioni si evidenzia per la presenza del punto fermo conclusivo (seguito da uno spazio). Le istruzioni che non costituiscono gruppi singoli possono essere separate, oltre che con lo spazio, con la virgola, il punto e virgola, e con la parola ‘**THEN**’.

Le istruzioni condizionali sono quelle che alterano la sequenza normale dell’esecuzione delle istruzioni, sulla base della verifica di una condizione. L’istruzione condizionale tipica è ‘**IF**’, ma molte altre istruzioni prevedono delle parole opzionali per descrivere un’azione da compiere al verificarsi di una certa condizione.

```
                /                  \  .--     /                  \ --.
                | { statement-1 }... |  |      | { statement-2 }... |   |
IF condition-1  <                  >  | ELSE <                  >   |
                | NEXT SENTENCE      |  |      | NEXT SENTENCE      |   |
                \                    /  '--    \                    / --'
```

Quello che si vede sopra è lo schema sintattico dell’istruzione ‘**IF**’,

che incorpora a sua volta altre istruzioni. Naturalmente, le istruzioni incorporate possono contenere altre istruzioni condizionali annidate; in ogni caso, non è possibile suddividere una struttura del genere in gruppi di istruzioni più piccoli, pertanto il punto fermo finale può apparire solo alla fine della struttura più esterna.

```
000000 IF ALTEZZA IS GREATER THAN 190
000000   THEN
000000       DISPLAY "LA PERSONA E' MOLTO ALTA!",
000000       PERFORM PERSONA-MOLTO-ALTA;
000000   ELSE
000000       IF ALTEZZA IS GREATER THAN 170
000000         THEN
000000             DISPLAY "LA PERSONA E' ABBASTANZA ALTA.",
000000             PERFORM PERSONA-ALTA;
000000         ELSE
000000             DISPLAY "LA PERSONA HA UN'ALTEZZA ",
000000                     "MEDIA O BASSA".
```

L'esempio mostra un'istruzione '**IF**' annidata, dove sono stati usati i vari separatori disponibili, per facilitare la lettura: la parola '**THEN**' non fa parte dell'istruzione, ma introduce qui le istruzioni da eseguire nel caso la condizione si avveri; la virgola viene usata per terminare le istruzioni singole, mentre il punto e virgola si usa per concludere quelle istruzioni dopo le quali si passa all'alternativa (introdotta dalla parola chiave '**ELSE**').

Il punto fermo finale è molto importante, perché rappresenta l'unico modo per stabilire dove finisca tutta la struttura, dal momento che nel linguaggio non è previsto l'uso di parole come «end if».

### 72.11.4 Sezioni, paragrafi e qualificazione

Quando la parte procedurale del programma si suddivide in sezioni, i nomi dei paragrafi devono essere univoci soltanto nell'ambito della sezione in cui vengono dichiarati.

Quando si deve fare riferimento al nome di un paragrafo che non è unico nel programma, si deve usare la qualificazione per distinguere a quale sezione si sta facendo riferimento; eccezionalmente, se si tratta della sezione in cui ci si trova già, la qualificazione è implicita.

La qualificazione si ottiene aggiungendo la parola '**OF**', oppure '**IN**', seguita dal nome della sezione.

```
                /    \
                | IN |
paragraph-name < ‾‾ > section-name
                | OF |
                \ ‾‾ /
```

### 72.11.5 Espressioni aritmetiche

L'espressione aritmetica è ciò che si traduce in un valore numerico, eventualmente attraverso l'uso di operatori. Gli operatori aritmetici disponibili nel linguaggio COBOL sono molto pochi, limitando le possibilità alle quattro operazioni.

È importante osservare che gli operatori aritmetici, tranne nel caso delle parentesi, vanno separati dai loro argomenti; diversamente, il segno '**-**' verrebbe confuso come carattere che compone una parola. Per esempio, '**A - B**' è un'espressione che rappresenta una sottrazione, mentre '**A-B**' è una parola.

Tabella 72.137. Espressioni aritmetiche.

| Espressione | Descrizione |
|---|---|
| + *x* | Non modifica il segno di ***x***. |
| - *x* | Inverte il segno di ***x***. |
| *x* + *y* | Somma i due operandi. |
| *x* - *y* | Sottrae da ***x*** il valore di ***y***. |
| *x* * *y* | Moltiplica i due operandi. |



| Espressione | Descrizione |
|---|---|
| *x* / *y* | Divide il primo operando per il secondo. |
| ( … ) | Cambia la precedenza stabilendo che quanto contenuto tra parentesi va calcolato prima di ciò che si trova all'esterno. |

L'ordine di precedenza nelle espressioni aritmetiche è quello consueto: prima gli operatori unari, che si applicano a un operando singolo, poi la moltiplicazione e la divisione, quindi la somma e la sottrazione.

### 72.11.6 Espressioni condizionali

Nel linguaggio COBOL si distinguono diversi tipi di espressioni condizionali elementari, che vengono descritte nelle sezioni successive. Le espressioni elementari, a loro volta, si possono combinare in espressioni composte, con l'uso di operatori booleani ed eventualmente con l'aiuto di parentesi tonde per modificare l'ordine di valutazione.

#### 72.11.6.1 Condizioni di relazione

Le condizioni di relazione stabiliscono un confronto tra due valori, che possono essere rappresentati da variabili, costanti o da espressioni aritmetiche. Segue lo schema sintattico:

```
                      / IS [NOT] GREATER THAN \
                      |    ‾‾‾    ‾‾‾‾‾‾‾    |
                      | IS [NOT] LESS THAN    |
/ identifier-1      \ |    ‾‾‾    ‾‾‾‾       | / identifier-2      \
|                   | | IS [NOT] EQUAL TO     | |                   |
< literal-1         > <    ‾‾‾    ‾‾‾‾‾      > < literal-2         >
|                   | | IS [NOT] >            | |                   |
\ arith-expression-1 / |    ‾‾‾    ‾          | \ arith-expression-2 /
                      | IS [NOT] <            |
                      |    ‾‾‾    ‾           |
                      \ IS [NOT] =            /
                           ‾‾‾    ‾
```

Tabella 72.139. Significato degli operatori di relazione.

| Operatore | Descrizione |
|---|---|
| IS [NOT] GREATER THEN | maggiore di, non maggiore di |
| IS [NOT] > | maggiore di, non maggiore di |
| IS [NOT] LESS THEN | minore di, non minore di |
| IS [NOT] < | minore di, non minore di |
| IS [NOT] EQUAL TO | uguale a, diverso da |
| IS [NOT] = | uguale a, diverso da |
| IS GREATER THAN OR EQUAL TO | maggiore o uguale a |
| IS >= | maggiore o uguale a |
| IS LESS THAN OR EQUAL TO | minore o uguale a |
| IS <= | minore o uguale a |

Quando gli operandi sono entrambi numerici, indipendentemente dal fatto che la loro rappresentazione sia in forma di «indice» ('**INDEX**'), compatta ('**COMPUTATIONAL**') o in forma di byte ('**DISPLAY**'), il confronto si basa sul valore numerico che esprimono, tenendo conto del segno, se c'è, considerando positivi i valori senza segno.

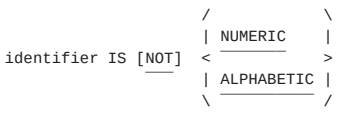
Quando si confrontano operandi alfanumerici, o quando anche uno solo è di tipo alfanumerico, il confronto avviene in modo lessicografico (in base all'ordinamento previsto dalla codifica adottata).

### 72.11.6.2 Condizioni di classe

La condizione di classe serve a stabilire se l'operando a cui si applica è numerico o alfabetico. È numerico un operando che è composto soltanto di cifre da '**0**' a '**9**', con il segno eventuale; è alfabetico un operando composto soltanto dal lettere alfabetiche ed eventualmente da spazi.

La condizione di classe si utilizza solo per verificare il contenuto di variabili che sono state dichiarate con una rappresentazione in byte ('**USAGE IS DISPLAY**').

Segue lo schema sintattico per esprimere la condizione di classe:




Naturalmente, se si usa la parola chiave '**NOT**', si intende invertire il significato della condizione.

### 72.11.6.3 Nomi di condizione

I nomi di condizione, che si dichiarano nella divisione '**DATA DIVISION**' con il numero di livello 88, servono a descrivere il confronto della variabile a cui si riferiscono con i valori che rappresentano.

Supponendo di avere dichiarato il nome di condizione '**PARI**' nel modo seguente:

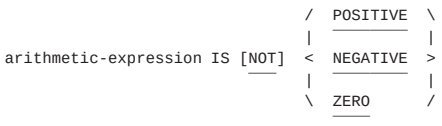
```
000000     02  CODICE          PIC 9.
000000         88  PARI        0, 2, 4, 6, 8.
000000         88  DISPARI     1, 3, 5, 7, 9.
000000         88  BASSO       0 THRU 4.
000000         88  ALTO        5 THRU 9.
```

Nella divisione '**PROCEDURE DIVISION**' potrebbero apparire righe come quelle successive, per verificare che la variabile '**CODICE**' contenga un valore pari:

```
000000     IF PARI
000000     THEN
000000         PERFORM …;
000000     ELSE
000000         …
```

### 72.11.6.4 Condizioni di segno

La condizione di segno permette di stabilire se un'espressione aritmetica (e può essere anche solo una costante o una variabile numerica) è positiva, negativa o se vale esattamente zero:




### 72.11.6.5 Condizioni composte e negate

Attraverso gli operatori booleani comuni, si possono definire delle condizioni composte, oppure negate. Si utilizzano le parole chiave '**AND**', '**OR**' e '**NOT**' per esprimere gli operatori booleani noti con lo stesso nome. Con l'ausilio delle parentesi tonde si possono modificare le precedenze nella valutazione delle espressioni.



Il linguaggio COBOL prevede una forma abbreviata per esprimere delle condizioni di relazione composte. Si osservi l'espressione seguente:

```
A > B OR A > C OR A < D
```

Questa potrebbe essere abbreviata così:

```
A > B OR > C OR < D
```

Tuttavia, si comprende che l'abbreviazione comporta maggiore difficoltà interpretativa nella lettura umana del programma sorgente.

### 72.11.7 Avverbi comuni

Per «avverbi comuni» qui si intendono delle parole chiave che possono far parte di varie istruzioni, fornendo però lo stesso tipo di funzionalità.

Tabella 72.144. Alcuni avverbi comuni.

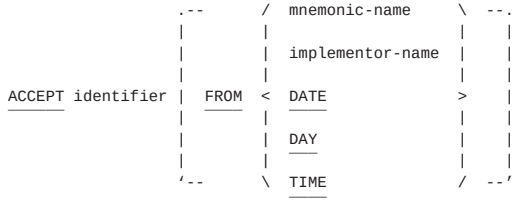
| Avverbio | Descrizione |
|---|---|
| `ROUNDED` | Quando una variabile deve ricevere il risultato di un'espressione aritmetica e non ha la possibilità di rappresentare in modo esatto i decimali, con l'uso dell'avverbio '**ROUNDED**' si chiede di assegnare un valore arrotondato nella cifra meno significativa. |
| `SIZE ERROR` | Quando una variabile deve ricevere il risultato di un'espressione aritmetica e non ha la possibilità di rappresentare alcune cifre più significative, si verifica un errore, che può essere espresso dalla condizione '**SIZE ERROR**'. La verifica di questa condizione implica l'indicazione di un'istruzione da eseguire se si verifica questo tipo di errore. |
| `CORRESPONDING` | Alcune istruzioni prevedono l'uso dell'avverbio '**CORRESPONDING**' per fare riferimento a variabili strutturate che contengono campi con lo stesso nome, pur non avendo la stessa struttura. Generalmente si usa per assegnare un gruppo di variabili a un altro gruppo, con lo stesso nome, in un'altra struttura. |

## 72.12 Istruzioni della divisione «PROCEDURE DIVISION»

Nelle sezioni successive sono raccolti e descritti, in ordine alfabetico, i modelli sintattici delle istruzioni principali del linguaggio COBOL, da usare nella divisione '**PROCEDURE DIVISION**'.

### 72.12.1 Istruzione «ACCEPT»

L'istruzione '**ACCEPT**' permette di ricevere un'informazione da una fonte esterna al programma e di assegnarla a una variabile. Generalmente si usa questa istruzione per consentire all'utente l'inserimento di un valore attraverso il terminale che sta utilizzando, ma, a seconda delle possibilità offerte dal compilatore, può servire anche per l'acquisizione di altri dati.




La fonte di dati per l'istruzione '**ACCEPT**' può essere dichiarata attraverso un nome mnemonico, definito nel paragrafo '**SPECIAL-NAMES**' della sezione '**INPUT-OUTPUT SECTION**' (se-

zione 72.4.3), un nome particolare che dipende da funzionalità speciali del compilatore, oppure un nome che fa riferimento alla data o all'orario attuale (le parole chiave '**DATE**', '**DAY**', '**TIME**').

Tabella 72.146. Parole chiave standard per definire l'accesso alle informazioni data-orario.

| Parola chiave | Descrizione |
|---|---|
| DATE | Fornisce la data attuale, formata da sei cifre numeriche, secondo la forma ***yymmdd*** (due cifre per l'anno, due per il mese e due per il giorno). |
| DAY | Fornisce il giorno attuale, formato da cinque cifre numeriche, secondo la forma ***yyddd*** (due cifre per l'anno e tre cifre per il giorno, espresso in quantità di giorni trascorsi dall'inizio dell'anno). |
| TIME | Fornisce l'orario attuale, formato da otto cifre numeriche, secondo la forma ***hhmmsscc*** (due cifre per l'ora, due per i minuti, due per i secondi e due per i centesimi di secondo). |

L'esempio seguente dimostra il funzionamento e l'utilizzo di queste parole chiave standard:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-ACCEPT-DATE-TIME.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-27.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 77  MY-DATE      PIC X(30).
001200 77  MY-DAY       PIC X(30).
001300 77  MY-TIME      PIC X(30).
001400*
001500 PROCEDURE DIVISION.
001600*
001700 MAIN.
001800     ACCEPT MY-DATE FROM DATE.
001900     ACCEPT MY-DAY  FROM DAY.
002000     ACCEPT MY-TIME FROM TIME.
002100     DISPLAY "DATE: ", MY-DATE.
002200     DISPLAY "DAY:  ", MY-DAY.
002300     DISPLAY "TIME: ", MY-TIME.
002400*
002500     STOP RUN.
002600*
```

Avviando questo programma il giorno 27 gennaio 2005, alle ore 13:30.45, si dovrebbe ottenere il risultato seguente:

```
DATE: 050227
DAY:  05058
TIME: 13304500
```

Tabella 72.149. Parole chiave non standard.

| Parola chiave | Descrizione |
|---|---|
| CONSOLE<br><br>SYSIN | Quando non si specifica la fonte dei dati per l'istruzione '**ACCEPT**', si intende il terminale dal quale il programma è stato avviato; spesso, i compilatori considerano l'uso della parola chiave '**CONSOLE**', o di '**SYSIN**', come sinonimo di questo comportamento, anche se è quello predefinito. |
| COMMAND-LINE | I compilatori per i sistemi Unix consentono spesso di accedere al contenuto della riga di comando usata per avviare il programma, attraverso l'uso di questa parola chiave. |

L'esempio successivo dimostra l'uso di un nome mnemonico per dichiarare l'origine dei dati. Sono evidenziate le righe più



significative:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-ACCEPT.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-27.
000500*
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SOURCE-COMPUTER.
000900     OPENCOBOL.
001000 SPECIAL-NAMES.
001100     CONSOLE IS STANDARD-INPUT.
001200*
001300 DATA DIVISION.
001400*
001500 WORKING-STORAGE SECTION.
001600 77  MESSAGGIO            PIC X(30).
001700*
001800 PROCEDURE DIVISION.
001900*
002000 MAIN.
002100     DISPLAY "INSERISCI IL MESSAGGIO".
002200     ACCEPT MESSAGGIO FROM STANDARD-INPUT.
002300     DISPLAY "HAI INSERITO: ", MESSAGGIO.
002400*
002500     STOP RUN.
002600*
```

### 72.12.2 Istruzione «ADD»

L'istruzione '**ADD**' consente di eseguire delle somme. Sono previsti diversi formati per l'utilizzo di questa istruzione.

```
     /             \
     | identifier-1 |
ADD  <             >...  TO { identifier-n  [ROUNDED] }...
     | literal-1    |
     \             /

    [ ON SIZE ERROR  imperative-statement ]
```

Nello schema sintattico appena mostrato, si vede che dopo la parola chiave '**ADD**' si elencano delle costanti o variabili con valore numerico, da sommare assieme, sommando poi quanto ottenuto al contenuto delle variabili specificate dopo la parola chiave '**TO**'. L'opzione '**ROUNDED**' richiede di eseguire un arrotondamento se la variabile ricevente non può rappresentare in modo esatto il valore; l'opzione '**SIZE ERROR**' serve a eseguire un'istruzione nel caso una delle variabili riceventi non possa accogliere la porzione più significativa del valore ottenuto dalla somma. Si osservi l'esempio seguente:

```
000000     ADD 1, 2, 3, TO A.
```

Supponendo che la variabile '**A**', prima della somma contenga il valore 10, dopo la somma contiene il valore 16 (1+2+3+10).

```
     /             \  /             \
     | identifier-1 |  | identifier-2 |
ADD  <             >  <             >...
     | literal-1    |  | literal-2    |
     \             /  \             /

    GIVING { identifier-n  [ROUNDED] }...

    [ ON SIZE ERROR  imperative-statement ]
```

Quando al posto della parola chiave '**TO**', si usa '**GIVING**', la somma dei valori che precede tale parola chiave viene assegnata alle variabili indicate dopo, senza tenere in considerazione il loro valore iniziale nella somma. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'. Si osservi l'esempio seguente:

```
000000     ADD 1, 2, 3, GIVING A.
```

Qualunque sia il valore iniziale della variabile '**A**', dopo la somma

questa contiene il valore 6 (1+2+3).

```
      /              \
      | CORR          |
ADD   <               >  identifier-1  TO  identifier-2  [ROUNDED]
      | CORRESPONDING |
      \              /

    [ ON SIZE ERROR  imperative-statement ]
```

In questo ultimo caso, la somma fa riferimento a variabili strutturate, dove i campi della prima variabile devono essere sommati ai campi della seconda variabile che hanno lo stesso nome della prima. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'.

### 72.12.3 Istruzione «CLOSE»

Attraverso l'istruzione '**CLOSE**' si può chiudere un file aperto. Questa istruzione non riguarda i file definiti esplicitamente per le funzionalità di riordino e fusione del COBOL, perché questi non vengono aperti. La sintassi dell'istruzione può essere più o meno ricca, a seconda delle estensioni che offre il compilatore; tuttavia, lo schema seguente si adatta alla maggior parte delle situazioni:

```
       /              .--      /           \  --.  \
       |              |        | NO REWIND |    |  |
CLOSE  <  file-name-1  |  WITH  <           >    |  > ...
       |              |        | LOCK      |    |  |
       \              '--      \           /  --'  /
```

Il file indicato viene chiuso, eventualmente con delle opzioni. Se si tratta di un file sequenziale a nastro, si può utilizzare l'opzione '**NO REWIND**', con la quale si vuole evitare che il nastro venga riavvolto automaticamente dopo la chiusura, così da poter accedere eventualmente a un file successivo, già esistente o da creare sullo stesso nastro. L'opzione '**LOCK**' serve a impedire che il file possa essere riaperto nel corso del funzionamento del programma.

Nel caso si utilizzino dei nastri, quelli che il programma ha chiuso senza riavvolgere, vengono comunque riavvolti alla conclusione del programma stesso; inoltre, alla conclusione del programma vengono chiusi automaticamente i file che sono rimasti ancora aperti.

### 72.12.4 Istruzione «COMPUTE»

L'istruzione '**COMPUTE**' consente di calcolare un'espressione aritmetica, assegnando il risultato a una o più variabili:

```
COMPUTE  { identifier  [ROUNDED] }... = arithmetic-expression

    [ ON SIZE ERROR  imperative-statement ]
```

La variabile che nello schema sintattico appare con il nome ***identifier*** deve essere scalare e di tipo numerico, anche se può contenere una maschera di modifica. Possono essere indicate più variabili a sinistra del segno '**=**' e ognuna riceve una copia del risultato dell'espressione alla destra.

L'opzione '**ROUNDED**' serve a richiedere un arrotondamento se la variabile ricevente non può rappresentare il risultato con la stessa precisione ottenuta dal calcolo dell'espressione; l'opzione '**SIZE ERROR**' consente di richiamare un'istruzione nel caso una delle variabili riceventi non fosse in grado di contenere la parte più significativa del valore ottenuto calcolando l'espressione.

```
000000     COMPUTE D = A * B + C.
```

L'esempio mostra che si vuole assegnare alla variabile '**D**' il risultato dell'espressione '**A * B + C**' (A moltiplicato B, sommato a C).

### 72.12.5 Istruzione «DELETE»

L'istruzione '**DELETE**' cancella un record logico da un file organizzato in modo relativo o a indice (sono esclusi i file organizzati in modo sequenziale).



```
DELETE  file-name  RECORD  [INVALID KEY imperative-statement]
```

Per poter cancellare un record è necessario che il file sia stato aperto in lettura e scrittura ('**I-O**').

Se il file viene utilizzato con un accesso sequenziale, l'opzione '**INVALID KEY**' non è applicabile e non deve essere scritta nell'istruzione. Inoltre, utilizzando un accesso sequenziale, prima di eseguire un'istruzione '**DELETE**' è necessario che il puntatore del record sia stato posizionato attraverso un'istruzione '**READ**'. L'istruzione '**READ**' deve precedere immediatamente l'istruzione '**DELETE**', che così può cancellare il record appena letto.

Quando il file viene utilizzato con un accesso diretto ('**RANDOM**') o dinamico ('**DYNAMIC**'), l'opzione '**INVALID KEY**' è obbligatoria, a meno di avere dichiarato un'azione alternativa, in caso di errore, nella zona di istruzioni definite come '**DECLARATIVES**', all'inizio della divisione '**PROCEDURE DIVISION**'. Per individuare il record da cancellare, si fa riferimento alla chiave, come specificato dalla dichiarazione '**RECORD KEY**', associata al file in questione. Se si tenta di cancellare un record indicando una chiave che non esiste, si ottiene l'errore che fa scattare l'esecuzione dell'istruzione associata all'opzione '**INVALID KEY**'.

Dipende dal compilatore il modo in cui viene trattato effettivamente il record da cancellare: questo potrebbe essere sovrascritto con un valore prestabilito, oppure potrebbe essere semplicemente segnato per la cancellazione; in ogni caso, il record non viene cancellato fisicamente dal file.

Quando si accede al file attraverso un indice, bisogna considerare che la cancellazione può provocare la comparsa di record con chiavi doppie, se la cancellazione implica la sovrascrittura del record con un certo valore; inoltre, se il file contiene record con chiavi doppie, la cancellazione di un record specificando la sua chiave, può portare a cancellare quello sbagliato. Pertanto, in presenza di file a indice con chiavi doppie, conviene usare un accesso sequenziale per individuare in modo esatto il record da cancellare.

### 72.12.6 Istruzione «DISPLAY»

L'istruzione '**DISPLAY**' consente di emettere un messaggio attraverso un dispositivo che consenta di farlo. Generalmente, se usata senza opzioni, la visualizzazione avviene attraverso il terminale dal quale è stato avviato il programma.

```
        /            \     .--      /                  \  --.
        | literal    |     |        | implementor-name |    |
DISPLAY <            >...  |  UPON  <                  >    |
        | identifier |     |        | mnemonic-name    |    |
        \            /     '--      \                  /  --'
```

Osservando lo schema sintattico si vede che dopo la parola chiave '**DISPLAY**' si possono mettere delle costanti letterali o dei nomi di variabile. Questi elementi possono rappresentare sia valori alfanumerici, sia numerici (tuttavia, il compilatore potrebbe rifiutarsi di accettare delle variabili di tipo '**INDEX**'): è il compilatore che provvede a eseguire le conversioni necessarie. L'elenco di costanti o di variabili viene concatenato prima della visualizzazione.

L'aggiunta dell'opzione '**UPON**' consente di specificare dove deve essere emesso il messaggio. Si può indicare una parola chiave definita dal compilatore, che identifica qualche tipo di dispositivo, oppure un nome mnemonico, da specificare nel paragrafo '**SPECIAL-NAMES**' della sezione '**INPUT-OUTPUT SECTION**' (sezione 72.4.3).

Tabella 72.161. Parole chiave non standard.

| Parola chiave | Descrizione |
|---|---|
| CONSOLE<br><br>SYSOUT | Quando non si specifica la fonte dei dati per l'istruzione '**ACCEPT**', si intende il terminale dal quale il programma è stato avviato; spesso, i compilatori considerano l'uso della parola chiave '**CONSOLE**', o di '**SYSOUT**', come sinonimo di questo comportamento, anche se è quello predefinito. |

L'esempio successivo mostra un uso abbastanza comune dell'istruzione '**DISPLAY**':

```
000000     DISPLAY "ATTENZIONE: ", A, " + ", B, " = ", C.
```

L'esempio mostra in particolare il concatenamento che si vuole ottenere. Si ricorda che non è importante se le variabili utilizzate nell'istruzione sono alfanumeriche o numeriche, perché è il compilatore che provvede a convertire tutto nel modo più appropriato al tipo di dispositivo che deve emettere il messaggio.

### 72.12.7 Istruzione «DIVIDE»

L'istruzione '**DIVIDE**' consente di eseguire delle divisioni, fornendone il risultato ed eventualmente il resto. Sono previsti diversi formati per l'utilizzo di questa istruzione.

```
        /             \
        | identifier-1 |
DIVIDE  <             >  INTO  { identifier-2 [ROUNDED] }...
        | literal-1    |
        \             /

    [ ON SIZE ERROR  imperative-statement ]
```

Nello schema sintattico appena mostrato, si vede che dopo la parola chiave '**DIVIDE**' viene indicato un valore, in forma costante o attraverso una variabile; questo valore viene diviso per la variabile indicata dopo la parola chiave '**INTO**' e il risultato viene assegnato alla stessa variabile che funge da divisore. Se appaiono più variabili dopo la parola '**INTO**', la divisione viene ripetuta per ognuna di quelle, assegnando rispettivamente il risultato.

L'opzione '**ROUNDED**' richiede di eseguire un arrotondamento se la variabile ricevente non può rappresentare in modo esatto il valore; l'opzione '**SIZE ERROR**' serve a eseguire un'istruzione nel caso una delle variabili riceventi non possa accogliere la porzione più significativa del valore ottenuto dalla somma. Si osservi l'esempio seguente:

```
000000     DIVIDE 100 INTO A.
```

Supponendo che la variabile '**A**', prima della divisione contenga il valore 5, dopo l'operazione contiene il valore 20 (100/5). Si potrebbe scrivere la stessa cosa utilizzando l'istruzione '**COMPUTE**':

```
000000     COMPUTE A = 100 / A.
```

Lo schema sintattico successivo mostra l'utilizzo di '**DIVIDE**' in modo da non alterare i valori utilizzati come divisori:

```
        /             \  /      \  /             \
        | identifier-1 |  | INTO |  | identifier-2 |
DIVIDE  <             >  <      >  <              >
        | literal-1    |  | BY   |  | literal-2    |
        \             /  \      /  \             /

    GIVING identifier-3  [ROUNDED]

    [ REMAINDER  identifier-4  [ROUNDED] ]

    [ ON SIZE ERROR  imperative-statement ]
```

Nella forma appena mostrata, dove le parole '**INTO**' e '**BY**' sono equivalenti, la divisione avviene immettendo il risultato dell'operazione nella variabile indicata dopo la parola '**GIVING**'. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'. Si osservi l'esempio seguente che ripete sostanzialmente l'esempio già mostrato in precedenza:

```
000000     DIVIDE 100 BY 5 GIVING A.
```

Utilizzando l'opzione '**REMAINDER**', si fa in modo che il resto della divisione venga inserito nella variabile che segue tale parola. Tuttavia, si osservi che per resto si intende ciò che rimane moltiplicando il quoziente ottenuto (***identifier-3***) per il divisore (***identifier-2*** o ***literal-2***), sottraendo poi questo valore ottenuto dal dividendo (***identifier-1*** o ***literal-1***). Si osservi l'esempio che segue:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-DIVIDE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-27.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 77  A            PIC 9(10)V99.
001200 77  B            PIC 9(10)V99.
001400*
001500 PROCEDURE DIVISION.
001600*
001700 MAIN.
001800     DIVIDE 100 BY 3 GIVING A REMAINDER B.
001900     DISPLAY "100 / 3 = ", A, " CON IL RESTO DI ", B.
002000*
002100     STOP RUN.
002200*
```

Una volta compilato questo programma, se viene messo in funzione si dovrebbe ottenere il risultato seguente, che dovrebbe chiarire di che tipo di resto si parla con questa istruzione:

```
100 / 3 = 0000000033.33 CON IL RESTO DI 0000000000.01
```

### 72.12.8 Istruzione «EXIT»

L'istruzione '**EXIT**' serve a concludere anticipatamente l'esecuzione di un gruppo di paragrafi, attraverso un'istruzione '**PERFORM**'. L'istruzione '**EXIT**' deve essere usata da sola, all'interno di un paragrafo tutto per sé:

```
paragraph-name

    EXIT.
```

Si osservi che un programma COBOL scritto in modo ordinato non dovrebbe avere bisogno di questa istruzione.

```
000000     PERFORM UNO THRU TRE.
000000     …
000000 UNO.
000000     …
000000 DUE.
000000     …
000000     IF …
000000       THEN
000000           GO TO TRE.
000000     …
000000 TRE.
000000     EXIT.
000000 QUATTRO.
000000     …
```

L'esempio appena mostrato serve a dare un'idea del significato dell'istruzione '**EXIT**': la chiamata iniziale con l'istruzione '**PERFORM**' richiede l'esecuzione sequenziale dei paragrafi da '**UNO**' a '**TRE**', ma nel paragrafo '**DUE**' si verifica una condizione e al suo avverarsi si esegue un salto ('**GO TO**') al paragrafo '**TRE**', che conclude comunque la chiamata principale.

Come già accennato, dal momento che l'uso dell'istruzione '**EXIT**' implica l'utilizzo di '**GO TO**', che notoriamente complica la com-

prensibilità di un programma in modo eccessivo, entrambe queste istruzioni sono da evitare accuratamente.

### 72.12.9 Istruzione «GO TO»

L'istruzione '**GO TO**' consente di saltare all'inizio di un paragrafo specificato, senza ritorno. Sono previsti due modi di utilizzo:

```
GO TO  procedure-name
```

Oppure:

```
GO TO  { procedure-name }... DEPENDING ON identifier
```

Nel primo caso, l'esecuzione dell'istruzione passa il controllo al paragrafo indicato; nel secondo, viene scelto il paragrafo a cui passare il controllo in base al valore indicato dopo la parola '**DEPENDING**'. Il valore in questione deve essere un numero intero, rappresentato attraverso una variabile (altrimenti non ci sarebbe motivo di usarlo), dove il valore uno rappresenta il primo paragrafo nominato dopo le parole '**GO TO**' e il valore ***n*** rappresenta il paragrafo ***n***-esimo dello stesso elenco.

L'utilizzo dell'istruzione '**GO TO**' complica la lettura di un programma sorgente COBOL e, secondo il parere di molti, andrebbe abolita. Si veda a questo proposito: Edsger W. Dijkstra, *Go To Statement Considered Harmful*, 1968, *http://www.acm.org/classics/oct95/* , *http://www.cs.utsa.edu/~wagner/CS3723/nogoto/harm2.html* e altri indirizzi.

### 72.12.10 Istruzione «IF»

L'istruzione '**IF**' consente di eseguire un gruppo di istruzioni solo se si verifica una condizione, o se questa non si verifica. Il formato di questa istruzione è visibile nello schema seguente:


Le istruzioni che seguono immediatamente la condizione (***statement-1***), vengono eseguite se la condizione si avvera; le istruzioni del gruppo che segue la parola '**ELSE**' vengono eseguite se la condizione non si avvera. Le istruzioni del primo e del secondo gruppo, possono contenere altre istruzioni '**IF**'.

Si osservi che la parola '**THEN**' è un separatore, ma viene usata spesso per migliorare la lettura di un'istruzione '**IF**':

```
000000 IF ALTEZZA IS GREATER THAN 190
000000   THEN
000000       DISPLAY "LA PERSONA E' MOLTO ALTA!",
000000       PERFORM PERSONA-MOLTO-ALTA;
000000   ELSE
000000       IF ALTEZZA IS GREATER THAN 170
000000         THEN
000000             DISPLAY "LA PERSONA E' ABBASTANZA ALTA.",
000000             PERFORM PERSONA-ALTA;
000000         ELSE
000000             DISPLAY "LA PERSONA HA UN'ALTEZZA ",
000000                     "MEDIA O BASSA".
```

L'esempio mostra un'istruzione '**IF**' che ne contiene un'altra dopo la parola '**ELSE**'. Si può osservare che il punto fermo che conclude il gruppo di istruzioni appare solo alla fine della prima istruzione '**IF**' e costituisce l'unico modo per poter comprendere dove finisce tutta la struttura. Si osservi che la rappresentazione della struttura con dei rientri appropriati serve per individuare facilmente i livelli di annidamento esistenti.

Data la particolarità di questo esempio, i rientri potrebbero essere gestiti in modo diverso, per sottolineare la presenza di una serie di condizioni alternative ('**ELSE IF**'):



```
000000 IF ALTEZZA IS GREATER THAN 190
000000 THEN
000000     DISPLAY "LA PERSONA E' MOLTO ALTA!",
000000     PERFORM PERSONA-MOLTO-ALTA;
000000 ELSE IF ALTEZZA IS GREATER THAN 170
000000 THEN
000000     DISPLAY "LA PERSONA E' ABBASTANZA ALTA.",
000000     PERFORM PERSONA-ALTA;
000000 ELSE
000000     DISPLAY "LA PERSONA HA UN'ALTEZZA MEDIA O BASSA".
```

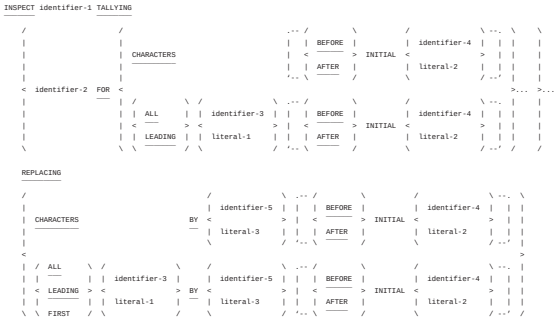
### 72.12.11 Istruzione «INSPECT»

L'istruzione '**INSPECT**' consente di scandire una variabile contenente una stringa alfanumerica, allo scopo di contare alcuni caratteri o di rimpiazzare alcuni dei caratteri della stringa. Sono previsti tre schemi sintattici per l'uso di questa istruzione, per il conteggio, la sostituzione, oppure per entrambe le cose simultaneamente.

**Conteggio dei caratteri**


**Sostituzione di caratteri**


**Conteggio e sostituzione**




In tutti gli schemi sintattici, la variabile indicata dopo la parola '**INSPECT**', che viene annotata come ***identifier-1***, deve contenere una stringa di caratteri, da scandire.

L'opzione '**BEFORE**' o '**AFTER**', permette di individuare una posizione nella stringa, da prendere come limite finale, o come punto iniziale, per l'elaborazione. In pratica, la variabile ***identifier-4***, o la costante letterale ***literal-2***, serve a rappresentare una sottostringa (anche un solo carattere), che all'interno della stringa complessiva si trova per prima (a partire da sinistra); se si usa la parola '**BEFORE**', l'elaborazione deve avvenire nella parta iniziale della stringa, fino a quella sottostringa di riferimento esclusa; se si usa la parola '**AFTER**', l'elaborazione deve avvenire nella parta finale della stringa, subito dopo quella sottostringa. Naturalmente, se la sottostringa indicata non esiste nella stringa, è come se l'opzione '**BEFORE**' o '**AFTER**' non fosse stata aggiunta.

Con il primo schema sintattico, si vogliono contare i caratteri della stringa che soddisfano certe condizioni. Il conteggio viene eseguito incrementando il valore contenuto nella variabile indicata nello schema come ***identifier-2***, che deve essere numerica. Si osservi che la

variabile non viene azzerata automaticamente, pertanto il suo valore iniziale viene sommato al conteggio eseguito.

Il conteggio può riguardare tutti i caratteri della stringa o della porzione iniziale o finale selezionata, utilizzando la parola '**CHARACTERS**'. Si osservi l'esempio successivo che utilizza solo questo tipo di conteggio.

Listato 72.180. Programma elementare che scandisce una stringa e conta i caratteri contenuti.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-INSPECT-TALLYING-1.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-15.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 77  STRINGA-DI-CARATTERI PIC X(30).
001400 77  CONTATORE-1          PIC 99 VALUE IS 0.
001500 77  CONTATORE-2          PIC 99 VALUE IS 0.
001600 77  CONTATORE-3          PIC 99 VALUE IS 0.
001700*
001800 PROCEDURE DIVISION.
001900*------------------------ LIVELLO 0 -----------------
002000 MAIN.
002100     MOVE "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123"
002200          TO STRINGA-DI-CARATTERI.
002300     INSPECT STRINGA-DI-CARATTERI
002400             TALLYING CONTATORE-1
002500             FOR CHARACTERS,
002600             TALLYING CONTATORE-2
002700             FOR CHARACTERS BEFORE INITIAL "H",
002800             TALLYING CONTATORE-3
002900             FOR CHARACTERS AFTER INITIAL "H".
003000     DISPLAY "CONTATORI: ", CONTATORE-1, " ",
003100             CONTATORE-2, " ", CONTATORE-3.
003200     STOP RUN.
```

L'esempio appena mostrato utilizza un'istruzione '**INSPECT**' per contare tre cose in una stringa, con una sola scansione: i caratteri contenuti in tutta la stringa; i caratteri fino alla comparsa della prima lettera «H»; i caratteri che si trovano dopo la lettera «H»:

```
            30 caratteri
    <--------------------------->
    ABCDEFGHIJKLMNOPQRSTUVWXYZ0123
    <-----> <-------------------->
  7 caratteri      22 caratteri
```

Compilando l'esempio e avviando il programma eseguibile che si ottiene, si dovrebbe vedere il risultato seguente:

```
CONTATORI: 30 07 22
```

Con la parola '**ALL**' si intendono contare tutte le corrispondenze con una certa sottostringa (***identifier-3*** o ***literal-1***), contenuta nella stringa complessiva o nella porzione specificata successivamente. Con la parola '**LEADING**', si vogliono contare solo le corrispondenze che avvengono in modo contiguo, purché inizino dal principio della zona di interesse.

Listato 72.183. Programma elementare che scandisce una stringa e conta i caratteri che corrispondono a delle sottostringhe.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-INSPECT-TALLYING-2.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-15.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
```



```
001300 77  STRINGA-DI-CARATTERI PIC X(30).
001400 77  CONTATORE-1          PIC 99 VALUE IS 0.
001500 77  CONTATORE-2          PIC 99 VALUE IS 0.
001600 77  CONTATORE-3          PIC 99 VALUE IS 0.
001700*
001800 PROCEDURE DIVISION.
001900*------------------------ LIVELLO 0 -----------------
002000 MAIN.
002100     MOVE "ABCDEFGHIAAAABBBBCCCCDDDDEEEEF"
002200          TO STRINGA-DI-CARATTERI.
002300     INSPECT STRINGA-DI-CARATTERI
002400             TALLYING CONTATORE-1
002500             FOR ALL "E",
002600             TALLYING CONTATORE-2
002700             FOR LEADING "A" AFTER INITIAL "I",
002800             TALLYING CONTATORE-3
002900             FOR LEADING "B" BEFORE INITIAL "I".
003000     DISPLAY "CONTATORI: ", CONTATORE-1, " ",
003100             CONTATORE-2, " ", CONTATORE-3.
003200     STOP RUN.
```

In questo esempio viene cercata la corrispondenza con tutte le lettere «E»; le lettere «A» adiacenti che iniziano a partire dalla prima apparizione della lettera «I»; le lettere «B» adiacenti e iniziali, che si trovano prima di quella stessa lettera «I».

```
            5 lettere «E»
         ------------------------
         |                   ||||
    ABCDEFGHIAAAABBBBCCCCDDDDEEEEF
           |\\\\
           | ´´´´´´4 lettere «A» adiacenti e iniziali
           |
   lettera «I» di riferimento

 Non ci sono lettere «B» adiacenti e iniziali
 prima del riferimento.
```

Compilando l'esempio e avviando il programma eseguibile che si ottiene, si dovrebbe vedere il risultato seguente:

```
CONTATORI: 05 04 00
```

Il secondo schema sintattico mostra l'uso di '**INSPECT**' per rimpiazzare delle sottostringhe. L'interpretazione dello schema è simile a quella del conteggio, con la differenza che si aggiunge la parola chiave '**BY**', che ha alla sinistra la sottostringa da rimpiazzare e alla destra il suo nuovo valore. Quando si usa la parola '**CHARACTERS**', si intende rimpiazzare tutta la stringa (o tutta la porzione prima o dopo un certo riferimento), con qualcosa con un carattere; le parole '**ALL**' e '**LEADING**' funzionano sostanzialmente come nel conteggio, riferendosi a tutte le sottostringhe di un certo tipo o a tutte le sottostringhe iniziali e adiacenti, dello stesso tipo. In questo schema, si aggiunge la parola '**FIRST**', che identifica solo una prima corrispondenza, non ripetuta.

Listato 72.186. Programma che scandisce una stringa e sostituisce alcuni suoi contenuti. Il programma sfrutta un'estensione al linguaggio standard, che permette di eseguire più sostituzioni in una sola istruzione '**INSPECT**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-INSPECT-REPLACING.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-15.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 77  STRINGA-DI-CARATTERI PIC X(30).
001400*
001500 PROCEDURE DIVISION.
001600*------------------------ LIVELLO 0 -----------------
001700 MAIN.
001800     MOVE "AAAAAABBBBBBCCCCCCDDDDDDEEEEEE"
001900          TO STRINGA-DI-CARATTERI.
```

```
002000     INSPECT STRINGA-DI-CARATTERI REPLACING
002100             CHARACTERS BY "X" AFTER INITIAL "DDD",
002200             LEADING "BB" BY "YZ"
002250                     AFTER INITIAL "AAAAAA",
002300             FIRST "C" BY "W",
002400             ALL "C" BY "P".
002500     DISPLAY STRINGA-DI-CARATTERI.
002600     STOP RUN.
```

L'esempio appena mostrato sfrutta un'estensione al linguaggio tradizionale, in modo da ottenere più sostituzioni con una sola passata. L'esempio fatto in questo modo permette di capire cosa succede in queste situazioni particolari.

```
AAAAAABBBBBBCCCCCCDDDDDDEEEEEE
                  XXXXXXXXX  CHARACTERS BY "X" AFTER INITIAL "DDD"
      YZYZYZ                 LEADING "BB" BY "YZ" AFTER INITIAL "AAAAAA"
           W                 FIRST "C" BY "W"
            PPPPP            ALL "C" BY "P"
AAAAAAYZYZYZWPPPPPDDDXXXXXXXXX
```

Compilando l'esempio e avviando il programma eseguibile che si ottiene, si dovrebbe vedere il risultato seguente che rappresenta soltanto il contenuto finale della variabile elaborata:

```
AAAAAAYZYZYZWPPPPPDDDXXXXXXXXX
```

### 72.12.12 Istruzione «MOVE»

L'istruzione '**MOVE**' copia o assegna un valore in una o più variabili di destinazione. Sono disponibili due modi di usare questa istruzione:

```
      /             \
      | identifier-1 |
MOVE  <             >  TO  { identifier-2 }...
      | literal-1    |
      \             /
```

Oppure:

```
      /               \
      | CORRESPONDING |
MOVE  <               >  identifier-1  TO  { identifier-2 }...
      | CORR          |
      \               /
```

Nel primo caso, ciò che appare dopo la parola chiave '**MOVE**' può essere il nome di una variabile, oppure una costante. Il valore contenuto nella variabile o rappresentato dalla costante, viene copiato in tutte le variabili indicate dopo la parola '**TO**', rispettando eventualmente le regole di modifica stabilite dai modelli di definizione delle variabili.

Nel secondo caso, avendo aggiunto la parola '**CORRESPONDING**' (o soltanto '**CORR**'), si copia il contenuto di una variabile strutturata in una o più variabili strutturate, abbinando però i campi aventi lo stesso nome. In pratica, con il secondo schema si vogliono copiare i campi della prima variabile strutturata che hanno gli stessi nomi di quelli contenuti nella seconda variabile strutturata. Diversamente, per una copia di una variabile strutturata in altre variabili, mantenendo inalterata la struttura originale dei dati, si usa il primo schema sintattico.

È bene ricordare che in alcuni casi la copia dei dati non può essere eseguita; per esempio non si può assegnare a una variabile numerica un'informazione alfanumerica (tenendo conto che una variabile numerica che contiene delle regole di modifica, all'atto della sua lettura offre un'informazione alfanumerica).

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-MOVE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-28.
000500
000600 ENVIRONMENT DIVISION.
000700
000800 DATA DIVISION.
000900
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-1.
001200     02 A         PIC 999V99.
001300     02 B         PIC X(10).
001400     02 C         PIC 99999.
001500
001600 01  RECORD-2.
001700     02 C         PIC 9999999.
001800     02 B         PIC X(12).
001900     02 A         PIC 9999V999.
002000
002100 PROCEDURE DIVISION.
002200
002300 MAIN.
002400     MOVE 123.45       TO A OF RECORD-1.
002500     MOVE "ABCDEFGHIJ" TO B OF RECORD-1.
002600     MOVE 12345        TO C OF RECORD-1.
002700     DISPLAY "RECORD-1: ", RECORD-1.
002800     DISPLAY "    A:    ", A OF RECORD-1.
002900     DISPLAY "    B:    ", B OF RECORD-1.
003000     DISPLAY "    C:    ", C OF RECORD-1.
003100
003200     MOVE RECORD-1 TO RECORD-2.
003300     DISPLAY "RECORD-2: ", RECORD-2
003400     DISPLAY "    A:    ", A OF RECORD-2.
003500     DISPLAY "    B:    ", B OF RECORD-2.
003600     DISPLAY "    C:    ", C OF RECORD-2.
003700
003800     MOVE CORRESPONDING RECORD-1 TO RECORD-2.
003900     DISPLAY "RECORD-2: ", RECORD-2
004000     DISPLAY "    A:    ", A OF RECORD-2.
004100     DISPLAY "    B:    ", B OF RECORD-2.
004200     DISPLAY "    C:    ", C OF RECORD-2.
004300
004400     STOP RUN.
```

L'esempio mostra un programma in cui ci sono due variabili strutturate, contenenti campi, simili, con lo stesso nome, ordinati in modo differente. Dopo aver assegnato dei valori ai campi della prima variabile, il contenuto della variabile viene copiato nella seconda; successivamente, viene ripetuta la copia in modo corrispondente.

Se si compila il programma con OpenCOBOL e si avvia ciò che si ottiene, si dovrebbe vedere un risultato simile a quello seguente, dove si può notare la differenza tra un tipo di copia e l'altra:

```
RECORD-1: 12345ABCDEFGHIJ12345
    A:    123.45
    B:    ABCDEFGHIJ
    C:    12345
RECORD-2: 12345ABCDEFGHIJ12345
    A:    5   .000
    B:    CDEFGHIJ1234
    C:    12345A
RECORD-2: 0012345ABCDEFGHIJ  0123450
    A:    0123.450
    B:    ABCDEFGHIJ
    C:    0012345
```

Si osservi che una variabile di tipo '**INDEX**' non può essere usata con l'istruzione '**MOVE**'. Per assegnare un valore a una tale variabile occorre servirsi dell'istruzione '**SET**'.

### 72.12.13 Istruzione «MULTIPLY»

L'istruzione '**MULTIPLY**' consente di eseguire delle moltiplicazioni. Sono previsti due diversi formati per l'utilizzo di questa istruzione.

```
          /             \
          | identifier-1 |
MULTIPLY  <             >  BY  { identifier-2 [ROUNDED] }...
          | literal-1    |
          \             /

    [ ON SIZE ERROR  imperative-statement ]
```

Nello schema sintattico appena mostrato, si vede che dopo la parola chiave '**MULTIPLY**' viene indicato un valore, in forma costante o attraverso una variabile; questo valore viene moltiplicato per la variabile indicata dopo la parola chiave '**BY**' e il risultato viene assegnato alla stessa variabile che funge da moltiplicatore. Se appaiono

più variabili dopo la parola '**BY**', la moltiplicazione viene ripetuta per ognuna di quelle, assegnando rispettivamente il risultato.

L'opzione '**ROUNDED**' richiede di eseguire un arrotondamento se la variabile ricevente non può rappresentare in modo esatto il valore; l'opzione '**SIZE ERROR**' serve a eseguire un'istruzione nel caso una delle variabili riceventi non possa accogliere la porzione più significativa del valore ottenuto dalla somma. Si osservi l'esempio seguente:

```
000000     MULTIPLY 100 BY A.
```

Supponendo che la variabile '**A**', prima della divisione contenga il valore 5, dopo l'operazione contiene il valore 500 (100×5). Si potrebbe scrivere la stessa cosa utilizzando l'istruzione '**COMPUTE**':

```
000000     COMPUTE A = 100 * A.
```

Lo schema sintattico successivo mostra l'utilizzo di '**MULTIPLY**' in modo da non alterare i valori utilizzati come moltiplicatori:

```
          /             \       /             \
          | identifier-1 |       | identifier-2 |
MULTIPLY  <             >   BY  <             >
          | literal-1   |       | literal-2    |
          \             /       \             /

    GIVING identifier-3  [ROUNDED]

    [ ON SIZE ERROR  imperative-statement ]
```

Nella forma appena mostrata, la moltiplicazione avviene immettendo il risultato dell'operazione nella variabile indicata dopo la parola '**GIVING**'. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'. Si osservi l'esempio seguente che ripete sostanzialmente l'esempio già mostrato in precedenza:

```
000000     MULTIPLY 100 BY 5 GIVING A.
```

### 72.12.14 Istruzione «OPEN»

L'istruzione '**OPEN**' serve ad aprire un file, o un gruppo di file, specificando la modalità di accesso. Quando l'accesso a un file richiede l'esecuzione di alcune procedure meccaniche preliminari, questa istruzione serve a eseguirle. L'istruzione '**OPEN**' non riguarda i file dichiarati esplicitamente per il riordino e la fusione.

```
      /  INPUT   { file-name  [ WITH NO REWIND ] }...  \
      |                                                |
      |  OUTPUT  { file-name  [ WITH NO REWIND ] }...  |
OPEN  <                                                >...
      |  I-O     { file-name }...                      |
      |                                                |
      \  EXTEND  { file-name }...                      /
```

Dopo la parola chiave '**OPEN**' inizia l'elenco dei file che si vogliono aprire, cominciando con la parola chiave che definisce la modalità di accesso desiderata: '**INPUT**' richiede un accesso in lettura; '**OUTPUT**' un accesso in scrittura; '**I-O**' un accesso in lettura e scrittura; '**EXTEND**' un accesso in estensione (scrittura).

Il tipo di accesso consentito dipende dall'organizzazione dei file o dalla modalità di accesso; nelle versioni più vecchie del linguaggio, l'apertura in estensione ('**EXTEND**') può essere usata soltanto per i file sequenziali; l'apertura in lettura e scrittura ('**I-O**') richiede che il file sia collocato in un'unità di memorizzazione ad accesso diretto, come nel caso dei dischi.

L'opzione '**NO REWIND**' si riferisce al riavvolgimento automatico del nastro, che riguarda, evidentemente, solo file sequenziali su unità ad accesso sequenziale, che possono richiedere un'operazione di riavvolgimento. Se si usa questa opzione, si intende evitare che il nastro venga riavvolto automaticamente alla chiusura del file stesso. Per i file su disco, o comunque su unità ad accesso diretto, anche se si tratta di file con organizzazione sequenziale, questa opzione non deve essere usata.



Quando un file viene aperto (con questa istruzione) è possibile accedervi secondo la modalità prevista, con le istruzioni appropriate. L'apertura va eseguita una sola volta e la chiusura (con l'istruzione '**CLOSE**') dichiara la conclusione delle operazioni con quel file. Se un file deve essere riaperto all'interno del programma, probabilmente perché vi si vuole accedere secondo una modalità differente, o per altri motivi, è necessario che alla chiusura non sia utilizzata l'opzione '**lock**', che altrimenti impedirebbe di farlo.

L'apertura in lettura che si ottiene con la parola chiave '**READ**' serve ad accedere a un file esistente in modo da poter leggere il suo contenuto; l'apertura fa sì che la posizione relativa, iniziale, all'interno del file, corrisponda al primo record logico. Se il file non esiste, si presenta una condizione di errore.

L'apertura in scrittura che si ottiene con la parola chiave '**OUTPUT**' serve a **creare** un file, ma se il file esiste già, questo viene azzerato completamente.

L'apertura in lettura e scrittura che si ottiene con la parola chiave '**I-O**' serve a permettere l'accesso a un file esistente, sia per leggere i dati, sia per modificarli. La posizione relativa iniziale è quella del primo record logico.

L'apertura in estensione che si ottiene con la parola chiave '**EXTEND**', può essere utilizzata soltanto con file sequenziali e serve a consentire l'aggiunta di record a partire dalla fine del file iniziale. Pertanto, il puntatore relativo iniziale si trova dopo la fine dell'ultimo record logico e l'utilizzo di questo file avviene nello stesso modo di un'apertura in scrittura, con la differenza che il contenuto precedente non viene cancellato.

Se il file che viene aperto è associato a una variabile indicata con l'opzione '**FILE STATUS**' nell'istruzione '**SELECT**' (nella sezione '**FILE-CONTROL**' di '**ENVIRONMENT DIVISION**'), il valore di tale variabile viene aggiornato.

Tabella 72.199. File con organizzazione sequenziale (e accesso sequenziale).

| Istruzione | Apertura in lettura ('**INPUT**') | Apertura in scrittura ('**OUTPUT**') | Apertura in lettura e scrittura ('**I-O**') | Apertura in estensione ('**EXTEND**') |
|---|---|---|---|---|
| READ | X | | X | |
| WRITE | | X | | X |
| REWRITE | | | X | |

Tabella 72.200. File con organizzazione relativa o a indice, con accesso sequenziale.

| Istruzione | Apertura in lettura ('**INPUT**') | Apertura in scrittura ('**OUTPUT**') | Apertura in lettura e scrittura ('**I-O**') |
|---|---|---|---|
| READ | X | | X |
| WRITE | | X | |
| REWRITE | | | X |
| START | X | | X |
| DELETE | | | X |

Tabella 72.201. File con organizzazione relativa o a indice, con accesso diretto (*random*).

| Istruzione | Apertura in lettura (‘**INPUT**’) | Apertura in scrittura (‘**OUTPUT**’) | Apertura in lettura e scrittura (‘**I-O**’) |
|---|---|---|---|
| READ | X | | X |
| WRITE | | X | X |
| REWRITE | | | X |
| START | | | |
| DELETE | | | X |

Tabella 72.202. File con organizzazione relativa o a indice, con accesso dinamico.

| Istruzione | Apertura in lettura (‘**INPUT**’) | Apertura in scrittura (‘**OUTPUT**’) | Apertura in lettura e scrittura (‘**I-O**’) |
|---|---|---|---|
| READ | X | | X |
| WRITE | | X | X |
| REWRITE | | | X |
| START | X | | X |
| DELETE | | | X |

### 72.12.15 Istruzione «PERFORM»

L’istruzione ‘**PERFORM**’ consente di eseguire un gruppo di istruzioni, contenute all’interno di sezioni o di paragrafi della divisione ‘**PROCEDURE DIVISION**’, riprendendo poi il funzionamento nell’istruzione successiva.

Sono disponibili schemi sintattici diversi, perché la chiamata di queste procedure può essere gestita in maniere differenti. In effetti, questa istruzione è il mezzo con cui realizzare delle iterazioni, normali e con enumerazione, pertanto si rende necessaria questa flessibilità da parte dell’istruzione ‘**PERFORM**’.

Nelle sezioni successive vengono descritte le varie forme di utilizzo dell’istruzione ‘**PERFORM**’, per livelli successivi di complessità. Si tenga conto che la spiegazione riguardo al funzionamento per un certo livello, riguarda anche quelli più complessi successivi.

#### 72.12.15.1 Chiamata semplice

```
                          .--  /          \                 --.
                          |    |  THROUGH  |                   |
PERFORM  procedure-name-1  |    <           >  procedure-name-2  |
                          |    |  THRU     |                   |
                          '--  \          /                 --'
```

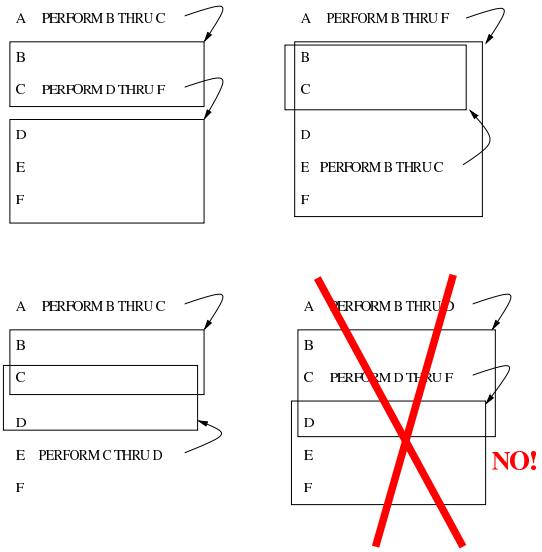
Secondo la forma di utilizzo più semplice dell’istruzione ‘**PERFORM**’, la chiamata esegue una volta sola l’intervallo di procedure indicate. Per procedure qui si intendono dei paragrafi, oppure delle sezioni intere della divisione ‘**PROCEDURE DIVISION**’.

Se si indica soltanto un nome (di paragrafo o di sezione), si intende eseguire solo la procedura relativa; se si indica la parola ‘**THROUGH**’ o ‘**THRU**’ seguita da un altro nome, si intendono eseguire tutti i paragrafi o tutte le sezioni dal primo al secondo nome incluso.

Il fatto che la chiamata di una procedura avvenga in modo così libero, implica la necessità di stabilire delle restrizioni alle chiamate annidate: una procedura, o un insieme di procedure chiamate attraverso l’istruzione ‘**PERFORM**’, possono contenere delle chiamate annidate. Queste chiamate interne, per poter essere eseguite correttamente, devono riguardare delle procedure più interne, oppure completamente esterne.



Figura 72.204. Schematizzazione delle forme di annidamento consentite e di quella non consentita (sbarrata).




La figura mostra schematicamente i vari modi in cui le istruzioni ‘**PERFORM**’ possono annidarsi, o possono in qualche modo riguardare le stesse porzioni di codice. L’ultimo esempio, in basso a destra, non è ammissibile perché la chiamata dei paragrafi da ‘**D**’ a ‘**F**’ verrebbe interrotta alla conclusione del paragrafo ‘**D**’, con il rientro dalla prima istruzione ‘**PERFORM**’.

#### 72.12.15.2 Chiamata ripetuta un certo numero di volte

```
                          .--  /          \                 --.
                          |    |  THROUGH  |                   |
PERFORM  procedure-name-1  |    <           >  procedure-name-2  |
                          |    |  THRU     |                   |
                          '--  \          /                 --'
        /                 \
        |  identifier-1   |
        <                 >  TIMES
        |  integer-1      |
        \                 /
```

Aggiungendo allo schema già visto un numero intero, espresso sia in forma costante, sia attraverso una variabile, seguito dalla parola ‘**TIMES**’, si intende ottenere a ripetizione della chiamata del gruppo di procedure indicato per quella quantità di volte.

Se il valore numerico indicato è pari a zero, oppure si tratta di un numero negativo, la chiamata delle procedure viene ignorata semplicemente.

#### 72.12.15.3 Chiamata ripetuta con condizione di uscita

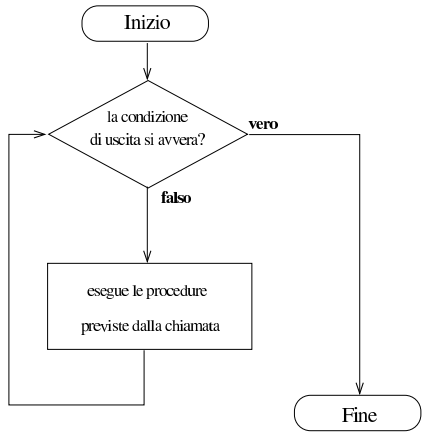
```
                          .--  /          \                 --.
                          |    |  THROUGH  |                   |
PERFORM  procedure-name-1  |    <           >  procedure-name-2  |
                          |    |  THRU     |                   |
                          '--  \          /                 --'

        UNTIL  condition-1
```

Quando nell’istruzione ‘**PERFORM**’ compare la parola chiave ‘**UNTIL**’, seguita da una condizione, si intende eseguire il gruppo di procedure indicate ripetutamente, fino a quando la condizione specificata restituisce il valore *Falso*.

La condizione di uscita viene verificata prima di eseguire ogni iterazione, pertanto, se risulta *Vero* all’inizio, le procedure non vengono eseguite.

Rispetto ai linguaggi di programmazione comuni, il COBOL attribuisce alla parola '**UNTIL**' un significato opposto, anche se logico: «si esegue il ciclo fino a quanto si verifica la condizione». Il problema è che nel senso comune ciò significa che il ciclo va ripetuto in quanto la condizione continua ad avverarsi, mentre secondo il senso del COBOL il ciclo va ripetuto fino a quando si verifica la condizione di uscita, nel senso che il verificarsi della condizione di uscita fa terminare il ciclo.

Figura 72.207. Diagramma di flusso dell'istruzione '**PERFORM**' iterativa con una condizione di uscita.




### 72.12.15.4 Chiamata ripetuta con condizione di uscita e incremento di contatori

«

```
                        .--  /          \                --.
                        |    | THROUGH  |                  |
PERFORM  procedure-name-1  |    <          >  procedure-name-2  |
                        |    | THRU     |                  |
                        '--  \          /                --'

          /             \       / identifier-3 \     /             \
          | identifier-2 |       |              |     | identifier-4 |
VARYING   <              >  FROM  < index-name-2 >  BY  <              >
          | index-name-1 |       |              |     | literal-2    |
          \             /       \ literal-1    /     \             /

        UNTIL  condition-1

/         /             \       / identifier-6 \     /             \ \
|         | identifier-5 |       |              |     | identifier-7 | |
|  AFTER  <              >  FROM  < index-name-4 >  BY  <              > |
<         | index-name-3 |       |              |     | literal-4    | >...
|         \             /       \ literal-3    /     \             / |
|                                                                    |
\       UNTIL  condition-2                                           /
```

Con l'aggiunta della parola chiave '**VARYING**', si intende gestire un contatore numerico (rappresentato nello schema da ***identifier-2*** o da ***index-name-1***, che pertanto può essere una variabile numerica o un indice di una tabella), specificando il valore di partenza dopo la parola '**FROM**', l'incremento a ogni ciclo dopo la parola '**BY**' e la condizione di uscita dopo la parola '**UNTIL**'.

Possono essere gestiti più contatori, con un limite che dipende dal compilatore. A ogni modo, per aggiungere un contatore si usa la parola '**AFTER**', che ne introduce la descrizione, così come per la parola '**VARYING**'.

Il contatore che viene incrementato a ogni ciclo, è quello più interno, ovvero quello descritto dall'ultima parola '**AFTER**'. Quando per quel contatore si verifica la condizione di uscita, viene incrementato il contatore del livello precedente (la penultima parola '**AFTER**' o direttamente '**VARYING**' in mancanza di quella) e azzerato quello interno.

Il ciclo termina quando sono scattate tutte le condizioni di uscita dei vari contatori.

Il linguaggio non pone vincoli alla gestione dei contatori indicati nell'istruzione '**PERFORM**', che possono essere alterati durante l'esecuzione delle procedure chiamate dall'istruzione stessa e in qualche modo possono contaminarsi tra di loro. Sta evidentemente al programmatore evitare di creare confusione nel programma, osservando anche che la sequenza esatta delle operazioni di incremento e azzeramento dei contatori cambia leggermente da uno standard all'altro del linguaggio.



Figura 72.209. Diagramma di flusso dell'istruzione '**PERFORM**' iterativa con l'incremento di un solo contatore.

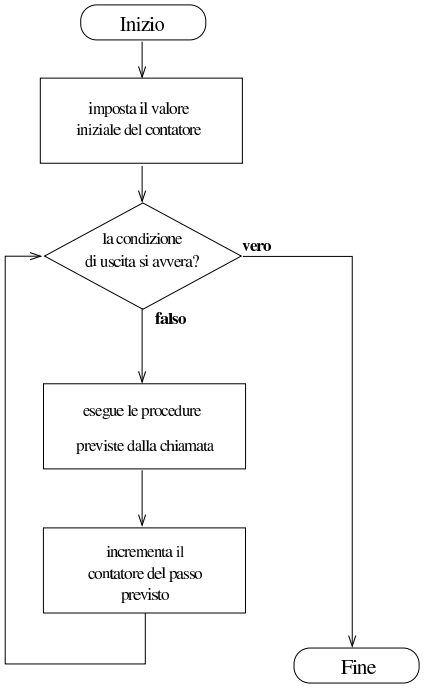
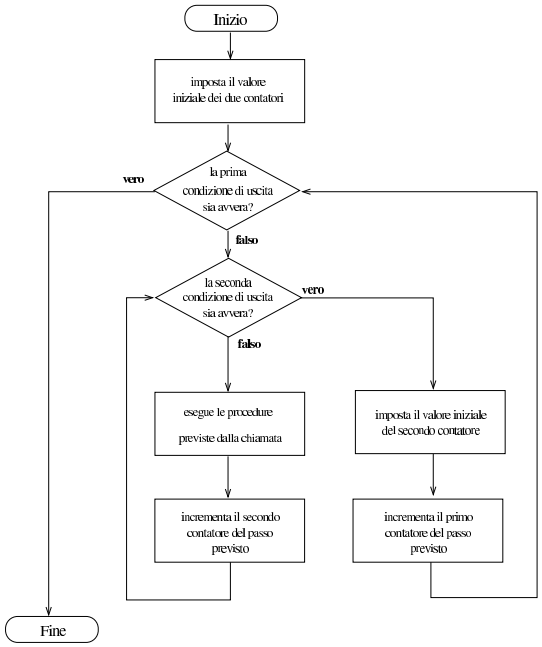



Figura 72.210. Diagramma di flusso dell'istruzione '**PERFORM**' iterativa con la gestione di due contatori.




L'esempio seguente mostra in modo molto semplice la gestione di tre contatori, che scandiscono valori interi da zero a due, senza fare nulla altro di particolare.

Listato 72.211. Programma chiama un paragrafo incrementando tre contatori.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-PERFORM.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-17.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 77  CONTATORE-1          PIC 99.
001400 77  CONTATORE-2          PIC 99.
001500 77  CONTATORE-3          PIC 99.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM VISUALIZZA-CONTATORI
002100             VARYING CONTATORE-1 FROM 0 BY 1
002200                     UNTIL CONTATORE-1 >= 2,
002300             AFTER   CONTATORE-2 FROM 0 BY 1
002400                     UNTIL CONTATORE-2 >= 2,
002500             AFTER   CONTATORE-3 FROM 0 BY 1
002600                     UNTIL CONTATORE-3 >= 2.
002700*
002800     STOP RUN.
002900*------------------------ LIVELLO 1 -----------------
003000 VISUALIZZA-CONTATORI.
003100     DISPLAY CONTATORE-1, " ", CONTATORE-2, " ",
003150             CONTATORE-3.
003200*
```

Una volta compilato questo programma, avviando ciò che si ottiene, si può vedere il risultato seguente:

```
00 00 00
00 00 01
00 01 00
00 01 01
01 00 00
01 00 01
01 01 00
01 01 01
```

### 72.12.16 Istruzione «READ»

L'istruzione '**READ**' serve a ottenere un record logico da un file, che risulta essere già stato aperto, in modo tale da consentire la lettura ('**INPUT**' o '**I-O**'). Sono disponibili formati diversi per l'utilizzo di questa istruzione, che dipendono dall'organizzazione del file a cui si accede.

**Organizzazione sequenziale, relativa e a indice**

```
READ file-name [NEXT] RECORD  [ INTO identifier ]

    [ AT END  { imperative-statement }... ]
```

**Organizzazione relativa**

```
READ file-name RECORD  [ INTO identifier ]

    [ INVALID KEY { imperative-statement }... ]
```

**Organizzazione a indice**

```
READ file-name RECORD  [ INTO identifier ]

    [ KEY IS  data-name ]

    [ INVALID KEY { imperative-statement }... ]
```

In tutti gli schemi sintattici che riguardano l'istruzione '**READ**', si può vedere che viene indicato immediatamente il nome del file (già aperto) che si vuole leggere. Successivamente, appare una parola chiave opzionale, '**INTO**', che precede il nome di una variabile; se viene specificata questa informazione, si intende fare in modo che il record logico ottenuto dal file, oltre che essere disponibile nella variabile strutturata dichiarata appositamente per questo, dopo l'indicatore di livello '**FD**' relativo, sia anche copiato in un'altra variabile. Inoltre, le istruzioni imperative (***imperative-statement***) che si possono inserire dopo le parole '**AT END**' e '**INVALID KEY**', servono a dichiarare cosa deve fare il programma nel caso la lettura fallisca per qualche motivo.

Se il file che viene letto è associato a una variabile indicata con l'opzione '**FILE STATUS**' nell'istruzione '**SELECT**' (nella sezione '**FILE-CONTROL**' di '**ENVIRONMENT DIVISION**'), il valore di tale variabile viene aggiornato.

Nel caso di un file a cui si accede sequenzialmente, si applica il primo schema sintattico. In questo caso l'istruzione '**READ**' fornisce il record attuale e sposta in avanti il puntatore al record, in modo che una lettura successiva fornisca il prossimo record. Quando l'accesso è dinamico e si vuole leggere un file in modo sequenziale, occorre aggiungere l'opzione '**NEXT**', per richiedere espressamente l'avanzamento al record successivo.

Quando si accede sequenzialmente, oppure in modo dinamico ma specificando che si richiede il record successivo, si può verificare un errore che consiste nel tentativo di leggere oltre la fine del file. Se ciò accade e se è stata specificata l'opzione '**AT END**', vengono eseguite le istruzioni che seguono tali parole.

La lettura sequenziale di un file relativo, comporta l'aggiornamento del valore della «chiave relativa», ovvero di quanto specificato con la dichiarazione '**RELATIVE KEY**' dell'istruzione '**SELECT**'.

La lettura sequenziale può essere applicata anche a un file organizzato a indice; in tal caso, la sequenza di lettura corrisponde a quella della chiave principale.

Quando si accede in modo diretto ai record all'interno di un file relativo, si utilizza il secondo schema sintattico, per ottenere il record specificato dal numero contenuto nella variabile che funge da chiave (come specificato nell'istruzione '**SELECT**', attraverso la dichiarazione '**RELATIVE KEY**'). Se un record con quel numero non esiste, si verifica la condizione controllata dall'opzione '**INVALID KEY**' e il programma esegue le istruzioni che questa controlla.

Il terzo formato sintattico si usa per i file organizzati a indice, con accesso diretto, in base alla chiave specificata. La chiave in questione è quella primaria, salvo specificarla nell'istruzione '**READ**' con l'opzione '**KEY IS**'. La chiave cercata deve essere scritta in corrispondenza del campo che la contiene, all'interno del record dichiarato dopo l'indicatore di livello '**FD**' relativo al file, secondo le specifiche dell'istruzione '**SELECT**' ('**RECORD KEY**', o '**ALTERNATE RECORD KEY**'). Se la lettura avviene con successo, si ottiene il record che contiene quella chiave; altrimenti si verifica la condizione controllata dall'opzione '**INVALID KEY**' e le istruzioni relative vengono eseguite.

La lettura ad accesso diretto di un file a indice, consente di ottenere il primo record che soddisfa la corrispondenza con la chiave cercata; se sono presenti record con chiavi doppie, le altre corrispondenze devono essere raggiunte attraverso una lettura sequenziale.

Listato 72.216. Programma elementare che legge un file sequenziale, ad accesso sequenziale.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-READ-SEQ.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               TINYCOBOL 0.61,
000600               OPENCOBOL 0.31.
000700 DATE-WRITTEN. 2005-03-12.
000800*
000900 ENVIRONMENT DIVISION.
001000*
001100 INPUT-OUTPUT SECTION.
001200*
```

```
001300 FILE-CONTROL.
001400*
001500     SELECT FILE-DA-LEGGERE ASSIGN TO "input.seq"
001600                            ORGANIZATION IS SEQUENTIAL.
001700*
001800 DATA DIVISION.
001900*
002000 FILE SECTION.
002100*
002200 FD  FILE-DA-LEGGERE
002300     LABEL RECORD IS STANDARD.
002400*
002500 01  RECORD-DA-LEGGERE PIC X(30).
002600*
002700 WORKING-STORAGE SECTION.
002800 01  EOF               PIC 9   VALUE ZERO.
002900*
003000 PROCEDURE DIVISION.
003100*------------------------ LIVELLO 0 -----------------
003200 MAIN.
003300     OPEN INPUT FILE-DA-LEGGERE.
003400     READ FILE-DA-LEGGERE
003500          AT END
003600               MOVE 1 TO EOF.
003700     PERFORM LETTURA UNTIL EOF = 1.
003800     CLOSE FILE-DA-LEGGERE.
003900*
004000     STOP RUN.
004100*------------------------ LIVELLO 1 -----------------
004200 LETTURA.
004300     DISPLAY RECORD-DA-LEGGERE.
004400     READ FILE-DA-LEGGERE
004500          AT END
004600               MOVE 1 TO EOF.
004700*
```

Listato 72.217. Programma elementare che legge un file sequenziale, ad accesso dinamico. Le differenze rispetto all'esempio precedente sono evidenziate.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-READ-DYN.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               TINYCOBOL 0.61,
000600               OPENCOBOL 0.31.
000700 DATE-WRITTEN. 2005-03-12.
000800*
000900 ENVIRONMENT DIVISION.
001000*
001100 INPUT-OUTPUT SECTION.
001200*
001300 FILE-CONTROL.
001400*
001500     SELECT FILE-DA-LEGGERE ASSIGN TO "input.seq"
001600                            ORGANIZATION IS SEQUENTIAL
001700                            ACCESS MODE IS DYNAMIC.
001800*
001900 DATA DIVISION.
002000*
002100 FILE SECTION.
002200*
002300 FD  FILE-DA-LEGGERE
002400     LABEL RECORD IS STANDARD.
002500*
002600 01  RECORD-DA-LEGGERE PIC X(30).
002700*
002800 WORKING-STORAGE SECTION.
002900 01  EOF               PIC 9   VALUE ZERO.
003000*
003100 PROCEDURE DIVISION.
003200*------------------------ LIVELLO 0 -----------------
003300 MAIN.
003400     OPEN INPUT FILE-DA-LEGGERE.
003500     READ FILE-DA-LEGGERE
003600          AT END
003700               MOVE 1 TO EOF.
003800     PERFORM LETTURA UNTIL EOF = 1.
003900     CLOSE FILE-DA-LEGGERE.
004000*
004100     STOP RUN.
004200*------------------------ LIVELLO 1 -----------------
004300 LETTURA.
004400     DISPLAY RECORD-DA-LEGGERE.
004500     READ FILE-DA-LEGGERE NEXT RECORD
004600          AT END
004700               MOVE 1 TO EOF.
004800*
```



### 72.12.17 Istruzione «REWRITE»

L'istruzione '**REWRITE**' consente si sovrascrivere un record logico all'interno di un file, purché questo risieda all'interno di un'unità che consente un accesso diretto ai dati (le unità sequenziali come i nastri sono escluse). Per utilizzare l'istruzione '**REWRITE**' il file deve essere stato aperto in lettura e scrittura ('**I-O**'); inoltre, il record deve avere una dimensione fissa.

**File organizzati in modo sequenziale**

```
REWRITE record-name  [ FROM identifier ]
```

**File organizzati in modo da consentire un accesso diretto**

```
REWRITE record-name  [ FROM identifier ]

    [ INVALID KEY { imperative-statement }... ]
```

Gli schemi sintattici mostrati hanno in comune la prima parte: il nome della variabile che fa riferimento al record, serve a individuare implicitamente il file a cui si fa riferimento; la variabile indicata dopo la parola '**FROM**', permette di copiare tale variabile su quella del record, prima di procedere alla sovrascrittura, come se si usasse l'istruzione '**MOVE**' prima di '**REWRITE**':

```
MOVE identifier TO record-name;

REWRITE record-name

    [ INVALID KEY { imperative-statement }... ]
```

Quando si utilizza l'istruzione '**REWRITE**' con un file aperto in modo sequenziale, prima è necessario che sia stata eseguita una lettura del record che si vuole sovrascrivere; la lettura implica la selezione del record. Nel caso particolare di un accesso sequenziale a un file con indice, oltre che leggere preventivamente il record da sovrascrivere, occorre accertarsi che la riscrittura mantenga la stessa chiave, altrimenti la riscrittura non avviene e si attiva invece l'opzione '**INVALID KEY**' (con l'esecuzione delle istruzioni che questa controlla). Oltre a questo, se il file prevede l'esistenza di una chiave secondaria e non sono ammesse chiavi doppie, se il record da sovrascrivere contiene una chiave secondaria già esistente in un altro, si ottiene, anche in questo caso, l'attivazione dell'opzione '**INVALID KEY**'.

Quando l'istruzione '**REWRITE**' si applica a file aperti attraverso un accesso diretto, dinamico o con chiave, la sovrascrittura non richiede più di procedere prima a una lettura del record, perché è sufficiente indicarlo tramite il numero ('**RELATIVE KEY**') oppure attraverso la chiave primaria. In tal caso, la condizione '**INVALID KEY**' si verifica quando il numero del record o la chiave primaria non corrispondono a nulla di già esistente nel file. Nel caso particolare dei file con indice, la condizione '**INVALID KEY**' si avvera anche quando, non essendo previste chiavi doppie, si tenta di modificare un record, immettendo però una chiave secondaria (non quella primaria) già esistente in un altro.

Listato 72.221. Programma elementare che legge un file sequenziale, ad accesso sequenziale, che quando incontra un record contenente lettere «A», lo sostituisce con lettere «Z».

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-READ-SEQ-REWRITE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               TINYCOBOL 0.61,
000600               OPENCOBOL 0.31.
```

```
000700 DATE-WRITTEN. 2005-03-12.
000800*
000900 ENVIRONMENT DIVISION.
001000*
001100 INPUT-OUTPUT SECTION.
001200*
001300 FILE-CONTROL.
001400*
001500     SELECT FILE-DA-MODIFICARE ASSIGN TO "input.seq"
001600            ORGANIZATION IS SEQUENTIAL.
001700*
001800 DATA DIVISION.
001900*
002000 FILE SECTION.
002100*
002200 FD  FILE-DA-MODIFICARE
002300     LABEL RECORD IS STANDARD.
002400*
002500 01  RECORD-DA-MODIFICARE PIC X(30).
002600*
002700 WORKING-STORAGE SECTION.
002800 01  EOF               PIC 9   VALUE ZERO.
002900*
003000 PROCEDURE DIVISION.
003100*------------------------ LIVELLO 0 -----------------
003200 MAIN.
003300     OPEN I-O FILE-DA-MODIFICARE.
003400     READ FILE-DA-MODIFICARE
003500          AT END
003600               MOVE 1 TO EOF.
003700     PERFORM LETTURA-RISCRITTURA UNTIL EOF = 1.
003800     CLOSE FILE-DA-MODIFICARE.
003900*
004000     STOP RUN.
004100*------------------------ LIVELLO 1 -----------------
004200 LETTURA-RISCRITTURA.
004300     IF RECORD-DA-MODIFICARE
004350         = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
004400     THEN
004500         MOVE "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"
004600              TO RECORD-DA-MODIFICARE,
004700         REWRITE RECORD-DA-MODIFICARE.
004800     READ FILE-DA-MODIFICARE
004900          AT END
005000               MOVE 1 TO EOF.
005100*
```

## 72.12.18 Istruzione «SEARCH»

L'istruzione '**SEARCH**' scandisce una tabella alla ricerca di un elemento che soddisfi una condizione, più o meno articolata, posizionando l'indice della tabella stessa in corrispondenza dell'elemento trovato. Sono disponibili due schemi sintattici: il primo serve per scandire le tabelle in modo sequenziale; il secondo serve per scandire delle tabelle ordinate, attraverso una ricerca binaria.

**Ricerca sequenziale**

```
                    .--        /               \ --.
                    |          |  identifier-2 |   |
SEARCH  identifier-1  |  VARYING  <              >   |
                    |          |  index-name-1 |   |
                    '--        \               / --'
    [ AT END  { imperative-statement-1 }... ]

    /                   /                               \  \
    |                   |  { imperative-statement-2 }... |  |
    <  WHEN  condition-1  <                               >  >...
    |                   |  NEXT SENTENCE                 |  |
    \                   \                               /  /
```

**Ricerca binaria per tabelle ordinate**

```
SEARCH ALL  identifier-1  [ AT END  { imperative-statement-1 }... ]

                    /                               \
                    |  { imperative-statement-2 }... |
    WHEN  condition-1  <                              >
                    |  NEXT SENTENCE                 |
                    \                               /
```

In entrambi i formati di utilizzo dell'istruzione '**SEARCH**', la variabile indicata come ***identifier-1*** deve essere stata dichiarata con l'opzione '**OCCURS**' e con l'opzione '**INDEXED BY**' (pertanto è obbligatorio che gli sia stato attribuito un indice in modo esplicito). Nel caso del secondo formato, che si utilizza per una ricerca binaria, è obbligatorio che la variabile indicata come ***identifier-1*** sia stata dichiarata con l'opzione '**KEY IS**', che sta a specificare il fatto che la tabella è ordinata in base a una certa chiave.

L'opzione '**AT END**' di entrambi gli schemi sintattici precede una o più istruzioni da eseguire nel caso la ricerca fallisca.

La parola chiave '**WHEN**' precede una condizione, che deve essere soddisfatta per lo scopo della ricerca, dopo la quale vengono eseguite le istruzioni successive (***imperative-statement-2***). Quando la scansione avviene in modo sequenziale, secondo il primo formato, la condizione può essere espressa in modo abbastanza libero, inoltre si possono indicare condizioni differenti e gruppi diversi di istruzioni da eseguire; quando invece la ricerca avviene in modo ordinato (ricerca binaria), ci può essere una sola condizione, che verifichi la corrispondenza della chiave con il valore cercato (se ci sono chiavi secondarie, si combinano le condizioni con l'operatore '**AND**').

La condizione di una ricerca in una tabella ordinata (ricerca binaria) deve rispettare i limiti dello schema sintattico seguente, dove le metavariabili ***data-name*** sono le chiavi di ordinamento, che vanno indicate con gli indici necessari:

```
/              /              \  /  identifier-3        \  \
|              |  IS EQUAL TO  |  |                     |  |
|  data-name-1  <               >  <  literal-1          >  |
<              |  IS =          |  |                     |  >
|              \     -         /  \  arith-expression-1  /  |
|                                                           |
\  condition-name-1                                         /

    .--                                                        --.
    |      /             /             \ / identifier-4       \ \   |
    |      |             | IS EQUAL TO | |                     | |   |
    |      | data-name-2 <               > < literal-2          > |   |
    |  AND <             | IS =         | |                     | >   |...
    |      |             \     -        / \ arith-expression-2 / |   |
    |      |                                                     |   |
    |      \ condition-name-2                                    /   |
    '--                                                        --'
```

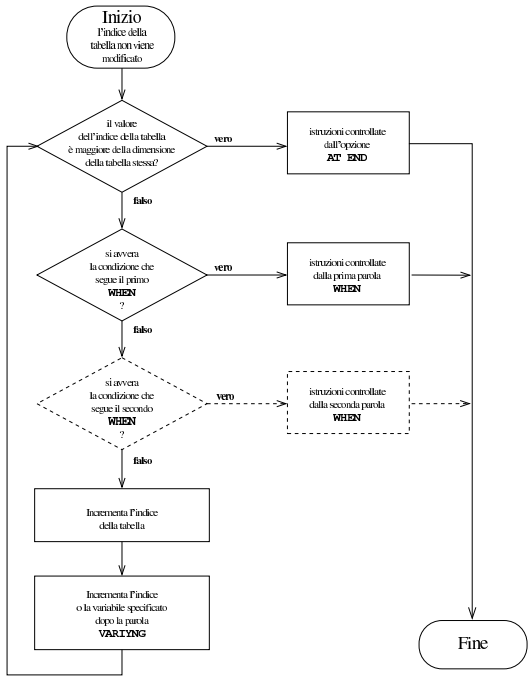
### 72.12.18.1 Ricerca sequenziale

La ricerca sequenziale con l'istruzione '**SEARCH**', inizia dal valore che si trova già ad avere l'indice, proseguendo fino a soddisfare una delle condizioni, oppure fino alla fine degli elementi. Pertanto, se l'indice dovesse avere un valore maggiore del numero degli elementi della tabella, l'istruzione terminerebbe immediatamente.

L'istruzione '**SEARCH**', usata per una ricerca sequenziale, esegue un ciclo di verifiche delle condizioni poste, quindi incrementa l'indice della tabella e ricomincia i confronti, fino a quando si avvera una delle condizioni, oppure quando la tabella non ha più elementi. Oltre a incrementare l'indice della tabella, può incrementare un altro indice, di un'altra tabella, o semplicemente una variabile numerica, attraverso l'uso dell'opzione '**VARYING**'.

Tradizionalmente, il funzionamento dell'istruzione '**SEARCH**', quando si usa per una scansione sequenziale di una tabella, lo si descrive attraverso un diagramma di flusso, nel quale si immagina di utilizzare due condizioni controllate dalla parola '**WHEN**', come si vede nella figura 72.225.

Figura 72.225. Esecuzione dell'istruzione '**SEARCH**' secondo il formato per la scansione sequenziale. Si mette in evidenza l'uso di due parole '**WHEN**' e si può comprendere come sarebbe con l'aggiunta di altre condizioni del genere.




Viene mostrato l'esempio di un programma completo che inizia con l'inserimento di dati all'interno di una tabella, quindi esegue una ricerca sequenziale al suo interno:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SEARCH.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-12.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 01  RECORD-UTENTI.
001400     02  UTENTE           OCCURS 60 TIMES
001500                           INDEXED BY IND-UTENTE.
001600         03  COGNOME      PIC X(30).
001700         03  NOME         PIC X(30).
001800         03  NOTA         PIC X(200).
001900 77  EOJ                  PIC 9   VALUE ZERO.
002000 77  RISPOSTA             PIC XX.
002100 77  RICERCA              PIC X(30).
002200*
002300 PROCEDURE DIVISION.
002400*------------------------ LIVELLO 0 -----------------
002500 MAIN.
002600     PERFORM INSERIMENTO-DATI
002700             VARYING IND-UTENTE FROM 1 BY 1
002800             UNTIL EOJ = 1.
002900     MOVE 0 TO EOJ.
003000     PERFORM SCANSIONE UNTIL EOJ = 1.
003100*
003200     STOP RUN.
003300*------------------------ LIVELLO 1 -----------------
003400 INSERIMENTO-DATI.
003500     DISPLAY IND-UTENTE, " INSERISCI IL COGNOME: ".
003600     ACCEPT COGNOME (IND-UTENTE).
003700     DISPLAY IND-UTENTE, " INSERISCI IL NOME: ".
003800     ACCEPT NOME (IND-UTENTE).
003900     DISPLAY IND-UTENTE,
003950             " INSERISCI UNA NOTA DESCRITTIVA: ".
004000     ACCEPT NOTA (IND-UTENTE).
```



```
004100*
004200     IF IND-UTENTE >= 60
004300       THEN
004400           MOVE 1 TO EOJ;
004500       ELSE
004600           DISPLAY "VUOI CONTINUARE? SI O NO",
004700           ACCEPT RISPOSTA;
004800           IF RISPOSTA = "SI"
004900             THEN
005000                 NEXT SENTENCE;
005100             ELSE
005200                 MOVE 1 TO EOJ.
005300*--------------------------------------------------
005400 SCANSIONE.
005500     DISPLAY "INSERISCI IL COGNOME DA CERCARE:".
005600     ACCEPT RICERCA.
005700     IF RICERCA = SPACES
005800       THEN
005900           MOVE 1 TO EOJ;
006000       ELSE
006100           SET IND-UTENTE TO 1,
006200           SEARCH UTENTE
006300                 AT END
006400                       DISPLAY "IL COGNOME CERCATO ",
006500                               "NON SI TROVA ",
006500                               "NELLA TABELLA: ",
006600                               QUOTE RICERCA QUOTE;
006700                 WHEN COGNOME (IND-UTENTE) = RICERCA
006800                     DISPLAY "IL COGNOME ", RICERCA,
006900                             "SI TROVA NELLA ",
006950                             "POSIZIONE ",
007000                             IND-UTENTE.
007100*
```

Nell'esempio sono evidenziate le righe in cui si dichiara la tabella e quelle che eseguono la scansione. Si deve osservare che prima dell'istruzione '**SEARCH**', l'indice deve essere collocato manualmente nella posizione iniziale.

### 72.12.18.2 Ricerca in una tabella ordinata

La ricerca che si esegue con l'istruzione '**SEARCH ALL**' richiede che si rispettino alcune condizioni:

- i dati contenuti nella tabella devono risultare ordinati come previsto dalle chiavi già dichiarate;
- i dati contenuti nella tabella devono risultare tutti validi;
- le chiavi a cui si fa riferimento nella condizione di ricerca devono essere sufficienti a raggiungere l'informazione in modo univoco.

È importante considerare correttamente il problema dei dati validi: quando una tabella deve ricevere una quantità imprecisata di dati in elementi separati, questa deve essere stata dichiarata in modo abbastanza grande da poter contenere tutto, ma così facendo si ha la certezza di avere una serie di celle vuote alla fine della tabella stessa. Per evitare che la scansione di ricerca tenga conto anche delle celle vuote, si dichiara la tabella con una quantità «variabile» di celle (con l'opzione '**OCCURS *m* TO *n* TIMES, DEPENDING ON *identifier***'). In realtà, più che trattarsi di una tabella che ha veramente una quantità variabile di celle, si fa in modo di stabilire qual è la dimensione massima, attraverso il controllo di una variabile apposita.

Dato il tipo di ricerca, non fa alcuna differenza il valore iniziale dell'indice della tabella.

L'esempio seguente mostra una variante del programma già descritto a proposito della ricerca sequenziale, modificato in modo da sfruttare una ricerca binaria. Si osservi che non è più necessario impostare il valore iniziale dell'indice, prima della scansione.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SEARCH-KEY.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-12.
```

```
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 01  RECORD-UTENTI.
001400     02  UTENTE           OCCURS 1 TO 60 TIMES
001500                          DEPENDING ON UTENTI-MAX
001600                          ASCENDING KEY IS COGNOME
001700                          INDEXED BY IND-UTENTE.
001800         03  COGNOME      PIC X(30).
001900         03  NOME         PIC X(30).
002000         03  NOTA         PIC X(200).
002100 77  UTENTI-MAX           USAGE IS INDEX.
002200 77  EOJ                  PIC 9   VALUE ZERO.
002300 77  RISPOSTA             PIC XX.
002400 77  RICERCA              PIC X(30).
002500*
002600 PROCEDURE DIVISION.
002700*------------------------ LIVELLO 0 -----------------
002800 MAIN.
002900     PERFORM INSERIMENTO-DATI
003000             VARYING IND-UTENTE FROM 1 BY 1
003100             UNTIL EOJ = 1.
003200     MOVE 0 TO EOJ.
003300     PERFORM SCANSIONE UNTIL EOJ = 1.
003400*
003500     STOP RUN.
003600*------------------------ LIVELLO 1 -----------------
003700 INSERIMENTO-DATI.
003800     MOVE IND-UTENTE TO UTENTI-MAX.
003900     DISPLAY IND-UTENTE, " INSERISCI IL COGNOME: ".
004000     ACCEPT COGNOME (IND-UTENTE).
004100     DISPLAY IND-UTENTE, " INSERISCI IL NOME: ".
004200     ACCEPT NOME (IND-UTENTE).
004300     DISPLAY IND-UTENTE,
004350             " INSERISCI UNA NOTA DESCRITTIVA: ".
004400     ACCEPT NOTA (IND-UTENTE).
004500*
004600     IF IND-UTENTE >= 60
004700       THEN
004800           MOVE 1 TO EOJ;
004900       ELSE
005000           DISPLAY "VUOI CONTINUARE? SI O NO",
005100           ACCEPT RISPOSTA;
005200           IF RISPOSTA = "SI"
005300             THEN
005400                 NEXT SENTENCE;
005500             ELSE
005600                 MOVE 1 TO EOJ.
005700*-----------------------------------------------------
005800 SCANSIONE.
005900     DISPLAY "INSERISCI IL COGNOME DA CERCARE:".
006000     ACCEPT RICERCA.
006100     IF RICERCA = SPACES
006200       THEN
006300           MOVE 1 TO EOJ;
006400       ELSE
006600           SEARCH ALL UTENTE
006700                 AT END
006800                       DISPLAY "IL COGNOME CERCATO ",
006900                               "NON SI TROVA ",
006950                               "NELLA TABELLA: ",
007000                               QUOTE RICERCA QUOTE;
007100                 WHEN COGNOME (IND-UTENTE) = RICERCA
007200                     DISPLAY "IL COGNOME ", RICERCA,
007300                             "SI TROVA ",
007350                             "NELLA POSIZIONE ",
007400                             IND-UTENTE.
007500*
```

### 72.12.19 Istruzione «SET»

L'istruzione '**SET**' permette di attribuire un valore all'indice di una tabella; valore inteso come la posizione all'interno della stessa. Sono disponibili due schemi sintattici: attraverso il primo si attribuisce una posizione determinata; con il secondo si incrementa o si decrementa l'indice di una certa quantità di posizioni.



```
     /             \         / index-name-2 \
     | index-name-1 |         |              |
SET  <              >...  TO  < identifier-2 >
     | identifier-1 |         |              |
     \              /         \ integer-1    /
```

Oppure:

```
                          /       \      /              \
                          |  UP   |      | identifier-3 |
SET  { index-name-3 }...  <       >  BY  <              >
                          |  DOWN |      | integer-2    |
                          \       /      \              /
```

In entrambi gli schemi sintattici, la variabile o le variabili indicate subito dopo la parola chiave '**SET**', sono quelle che rappresentano l'indice di una tabella e devono essere modificate. Nel primo caso, si intende assegnare loro il valore indicato o rappresentato dopo la parola chiave '**TO**', mentre nel secondo caso, l'indice viene incrementato ('**UP**') o diminuito ('**DOWN**') del valore posto dopo la parola chiave '**BY**'.

Quando nell'istruzione si usa una costante numerica, o una variabile numerica normale, deve trattarsi di un valore intero, che può essere senza segno, oppure può avere un segno positivo, con l'eccezione del caso dell'incremento o decremento dell'indice (nel secondo schema), dove può avere senso anche un segno negativo.

Nel primo schema sintattico, non sono ammesse tutte le combinazioni, rispetto a quando sembrerebbe dallo schema stesso. Per prima cosa, il valore che si attribuisce all'indice, deve essere valido nell'ambito della tabella a cui si riferisce; inoltre, valgono gli abbinamenti dello schema successivo. Nello schema si distingue tra variabili intere normali, variabili di tipo indice associate a una tabella e variabili di tipo indice indipendenti.

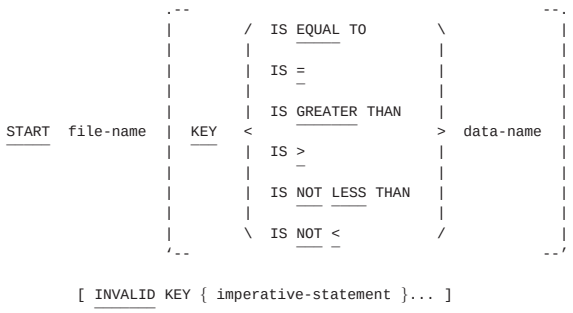
Tabella 72.230. Combinazioni degli operandi nell'istruzione '**SET**'.

| | Variabile ricevente di tipo numerico intero (*integer data item*) | Variabile ricevente di tipo indice associata a una tabella (*index name*) | Variabile ricevente di tipo indice non associata ad alcuna tabella (*index data item*) |
|---|---|---|---|
| Valore assegnato costituito da una costante numerica intera | non ammesso | assegnamento valido | non ammesso |
| Valore assegnato costituito da una variabile numerica intera | non ammesso | assegnamento valido | non ammesso |
| Valore assegnato costituito da una variabile di tipo indice associata a una tabella | assegnamento valido | assegnamento valido | assegnamento valido |
| Valore assegnato costituito da una variabile di tipo indice non associata ad alcuna tabella | non ammesso | assegnamento valido | assegnamento valido |

A seconda delle caratteristiche del compilatore, l'assegnamento di un valore a un indice può richiedere l'esecuzione di una conversione numerica appropriata.

### 72.12.20 Istruzione «START»

L'istruzione '**START**' consente di posizionare il puntatore del record logico di un file relativo o a indice, per il quale sia stato previsto un accesso sequenziale o dinamico.




Il file indicato dopo la parola chiave '**START**' è quello all'interno del quale si vuole posizionare il puntatore del record logico. Come accennato, il file deve essere organizzato in modo relativo o a indice; inoltre, deve essere stato aperto in lettura ('**INPUT**') o in lettura e scrittura ('**I-O**').

La variabile che appare alla fine dello schema sintattico (***data-name***), può avere due significati differenti: se si tratta di un file organizzato in modo relativo, questa deve individuare la variabile definita con la dichiarazione '**RELATIVE KEY**' dell'istruzione '**SELECT**' del file stesso; se si tratta di un file organizzato a indice, deve trattarsi della chiave di ordinamento (dichiarata come '**RECORD KEY**' o '**ALTERNATE RECORD KEY**' nell'istruzione '**SELECT**'), tenendo conto che può trattarsi di una porzione inferiore della chiave stessa, purché questa porzione si trovi a partire dall'inizio (a sinistra) della chiave.

L'opzione '**INVALID KEY**' introduce una o più istruzioni che vengono eseguite nel caso l'istruzione '**START**' fallisca a causa dell'indicazione di una chiave che con combacia secondo il tipo di confronto richiesto.

Nello schema sintattico, la parola chiave '**KEY**' precede un gruppo di parole che servono a stabilire la ***condizione di ricerca***. La corrispondenza con la chiave (costituita dal numero del record o dalla chiave di ordinamento vera e propria) può essere richiesta in modo esatto, oppure attraverso un altro tipo di relazione. Il record che per primo soddisfa la condizione di ricerca, è quello che viene selezionato. Una volta eseguita la selezione, il record potrebbe essere letto con l'istruzione '**READ**'.

Tabella 72.232. Condizione di ricerca.

| Operatore | Descrizione |
|---|---|
| `KEY IS EQUAL TO data-name`<br>`KEY IS = data-name` | la chiave, o la sua porzione, corrisponde esattamente |
| `KEY IS GREATER THAN data-name`<br>`KEY IS > data-name` | la chiave del record è superiore al valore specificato |
| `KEY IS NOT LESS THEN data-name`<br>`KEY IS NOT < THEN data-name` | la chiave del record non è inferiore (è maggiore o uguale) al valore specificato |

La condizione di ricerca (assieme alla parola chiave '**KEY**') e il nome della variabile che ha il ruolo di chiave, possono essere omessi. In tal caso, la ricerca avviene in base alla corrispondenza esatta con il valore che ha la variabile che costituisce la chiave relativa del file, oppure con quello che ha il campo della chiave primaria dello stesso.

Quando la chiave indicata nell'istruzione '**START**' corrisponde a una porzione iniziale della chiave primaria o secondaria del file, il confronto si basa solo su quella porzione di chiave, ignorando il resto; nello stesso modo, se la chiave indicata nell'istruzione è più grande della chiave primaria o di quella secondaria, il confronto si basa solo sulla dimensione della chiave che ha il file effettivamente (che risulta essere più breve).



Comunque sia l'esito della ricerca, l'esecuzione dell'istruzione '**START**', provoca l'aggiornamento della variabile che rappresenta lo stato del file ('**FILE STATUS**').

Listato 72.233. Programma elementare che legge un file relativo, ad accesso sequenziale, partendo dal terzo record.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-READ-SEQ-START.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-13.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "input.rel"
001300                            ORGANIZATION IS RELATIVE
001400                            RELATIVE KEY IS N-RECORD
001500                            ACCESS MODE IS SEQUENTIAL.
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-DA-LEGGERE
002200     LABEL RECORD IS STANDARD.
002300*
002400 01  RECORD-DA-LEGGERE PIC X(30).
002500*
002600 WORKING-STORAGE SECTION.
002700 77  EOF               PIC 9   VALUE ZERO.
002800 77  N-RECORD          PIC 999 VALUE ZERO.
002900*
003000 PROCEDURE DIVISION.
003100*------------------------- LIVELLO 0 -----------------
003200 MAIN.
003300     OPEN INPUT FILE-DA-LEGGERE.
003400     MOVE 3 TO N-RECORD.
003500     START FILE-DA-LEGGERE KEY IS EQUAL TO N-RECORD
003600           INVALID KEY
003700                   MOVE 1 TO EOF.
003800     READ FILE-DA-LEGGERE
003900          AT END
004000                MOVE 1 TO EOF.
004100     PERFORM LETTURA UNTIL EOF = 1.
004200     CLOSE FILE-DA-LEGGERE.
004300*
004400     STOP RUN.
004500*------------------------- LIVELLO 1 -----------------
004600 LETTURA.
004700     DISPLAY RECORD-DA-LEGGERE.
004800     READ FILE-DA-LEGGERE
004900          AT END
005000                MOVE 1 TO EOF.
005100*
```

### 72.12.21 Istruzione «STOP RUN»

L'istruzione '**STOP RUN**' conclude il funzionamento del programma; pertanto, può trovarsi soltanto alla fine di un gruppo di istruzioni.

```
STOP RUN.
```

Storicamente esiste una versione alternativa, ma superata, dell'istruzione '**STOP**', alla quale si associa una costante, allo scopo di mostrare tale valore attraverso il terminale principale. In quella situazione, l'esecuzione del programma veniva sospesa e poteva essere fatta riprendere dall'utente.

Considerato che esiste la possibilità di usare istruzioni come '**DISPLAY**' e '**ACCEPT**', è meglio utilizzare esclusivamente l'istruzione '**STOP RUN**' per l'arresto del programma, senza altre varianti.

### 72.12.22 Istruzione «STRING»

L'istruzione '**STRING**' consente di riempire delle variabili alfanumeriche specificando un punto di inizio, espresso in caratteri.

```
        /  /              \                /  identifier-2  \  \
        |  |  identifier-1  |               |                |  |
STRING  <  <              >... DELIMITED BY <  literal-2     >  >...
        |  |  literal-1    |               |                |  |
        \  \              /                \  SIZE          /  /

    INTO  identifier-3

    [ WITH POINTER  identifier-4 ]

    [ ON OVERFLOW  { imperative-statement-1 }... ]
```

Quello che si mette dopo la parola chiave '**STRING**' è un elenco di valori che si traducono in informazioni alfanumeriche, che vengono considerati come se fossero concatenati tra di loro. Dopo la parola '**DELIMITED**' si deve specificare un modo per delimitare la stringa complessiva indicata a sinistra. Se si usa la parola chiave '**SIZE**', si intende considerare tutta la stringa alfanumerica complessiva, altrimenti, si seleziona solo la parte che si trova a sinistra, prima di ciò che viene indicato come riferimento.

La stringa complessiva, eventualmente ridotta a destra in qualche modo, viene copiata all'interno della variabile indicata dopo la parola '**INTO**'. La stringa viene copiata a partire dalla prima posizione, oppure dalla posizione specificata dal numero indicato dopo la parola '**POINTER**'. Dopo la parola '**POINTER**' va indicata una variabile numerica, che, oltre a indicare la posizione iniziale dell'inserimento della stringa, viene incrementata di conseguenza, per i caratteri che vengono inseriti effettivamente.

Se si utilizza la parola '**OVERFLOW**', le istruzioni che appaiono subito dopo tale parola vengono eseguite se l'inserimento nella variabile di destinazione va oltre la fine della variabile stessa.

L'esempio successivo mostra un piccolo programma completo che compila in più fasi una variabile ricevente. La variabile ricevente contiene inizialmente una serie di simboli '**#**', per consentire di vedere facilmente cosa succede al suo interno, durante le varie fasi.

Listato 72.236. Programma elementare che dimostra il funzionamento di '**STRING**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-STRING.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 INSTALLATION. NANOLINUX IV,
000500               OPENCOBOL 0.31,
000600 DATE-WRITTEN. 2005-03-16.
000700*
000800 ENVIRONMENT DIVISION.
000900*
001000 DATA DIVISION.
001100*
001200 WORKING-STORAGE SECTION.
001300 77  TESTO-RICEVENTE      PIC X(40) VALUE ALL "#".
001400 77  PUNTATORE            PIC 99.
001500*
001600 PROCEDURE DIVISION.
001700*------------------------ LIVELLO 0 -----------------
001800 MAIN.
001900     MOVE 1 TO PUNTATORE.
002000     DISPLAY PUNTATORE, " ", TESTO-RICEVENTE.
002100     STRING "CIAO", SPACE, DELIMITED BY SIZE
002200            INTO TESTO-RICEVENTE
002250            WITH POINTER PUNTATORE.
002300     DISPLAY PUNTATORE, " ", TESTO-RICEVENTE.
002400     STRING "COME STAI?" DELIMITED BY SIZE
002500            INTO TESTO-RICEVENTE
002550            WITH POINTER PUNTATORE.
002600     DISPLAY PUNTATORE, " ", TESTO-RICEVENTE.
002700     MOVE 11 TO PUNTATORE.
002800     STRING "VA LA VITA?" DELIMITED BY SIZE
002900            INTO TESTO-RICEVENTE
002950            WITH POINTER PUNTATORE.
003000     DISPLAY PUNTATORE, " ", TESTO-RICEVENTE.
003100*
003200     STOP RUN.
003300*
```



Dopo aver compilato il programma, eseguendo ciò che si ottiene, di dovrebbe vedere il risultato seguente attraverso il terminale:

```
01 ########################################
06 CIAO ##################################
16 CIAO COME STAI?########################
22 CIAO COME VA LA VITA?##################
```

Come si può vedere leggendo il sorgente del programma, dopo l'inserimento della stringa '**CIAO** ', la variabile usata come puntatore all'interno della variabile di destinazione, si trova a essere già posizionata sulla sesta colonna, in modo che un inserimento ulteriore si trovi già nella posizione necessaria. Dopo, viene riposizionato il puntatore per sovrascrivere la parola «STAI».

### 72.12.23 Istruzione «SUBTRACT»

L'istruzione '**SUBTRACT**' consente di eseguire delle sottrazioni. Sono previsti diversi formati per l'utilizzo di questa istruzione.

```
          /             \  .--           --.
          | identifier-1 |  |  identifier-2  |
SUBTRACT  <             >  |                |...
          | literal-1   |  |  literal-2     |
          \             /  '--           --'

    FROM { identifier-m  [ROUNDED] }...

    [ ON SIZE ERROR  imperative-statement ]
```

Nello schema sintattico appena mostrato, si vede che dopo la parola chiave '**SUBTRACT**' si elencano delle costanti o variabili con valore numerico, che vengono sommate assieme inizialmente, per poi sottrarre tale valore dal contenuto delle variabili specificate dopo la parola chiave '**FROM**'. L'opzione '**ROUNDED**' richiede di eseguire un arrotondamento se la variabile ricevente non può rappresentare in modo esatto il valore; l'opzione '**SIZE ERROR**' serve a eseguire un'istruzione nel caso una delle variabili riceventi non possa accogliere la porzione più significativa del valore ottenuto dalla somma. Si osservi l'esempio seguente:

```
000000     SUBTRACT 1, 2, 3, FROM A.
```

Supponendo che la variabile '**A**', prima della somma contenga il valore 10, dopo la somma contiene il valore 4 (10-1-2-3).

```
          /             \  .-- /             \ --.
          | identifier-1 |  |   | identifier-2 |   |
SUBTRACT  <             >  |   <              >   |...
          | literal-1   |  |   | literal-2    |   |
          \             /  '-- \             / --'

          /             \
          | identifier-3 |
    FROM  <             >  [ROUNDED] ...
          | identifier-3 |
          \             /

    GIVING { identifier-n  [ROUNDED] }...

    [ ON SIZE ERROR  imperative-statement ]
```

Quando si utilizza la parola chiave '**GIVING**', si può indicare un solo valore dopo la parola chiave '**FROM**' e il risultato della sottrazione viene assegnato alle variabili che sono elencate dopo la parola '**GIVING**', senza tenere in considerazione il loro valore iniziale. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'. Si osservi l'esempio seguente:

```
000000     SUBTRACT 1, 2, 3, FROM 10 GIVING A.
```

Qualunque sia il valore iniziale della variabile '**A**', dopo la somma questa contiene il valore 4 (10-1-2-3).

```
          /              \
          | CORR          |
SUBTRACT  <                > identifier-1 FROM identifier-2 [ROUNDED]
          | CORRESPONDING |
          \ ______________ /

    [ ON SIZE ERROR imperative-statement ]
```

In questo ultimo caso, la sottrazione fa riferimento a variabili strutturate, dove i campi della prima variabile devono essere sottratti ai campi della seconda variabile che hanno lo stesso nome della prima. Valgono le stesse considerazioni già fatte a proposito delle opzioni '**ROUNDED**' e '**SIZE ERROR**'.

### 72.12.24 Istruzione «WRITE»

L'istruzione '**WRITE**' scrive un record logico in un file, aperto in modo appropriato. Nel caso di un file organizzato in modo sequenziale, il file può essere aperto in scrittura ('**OUTPUT**') o in estensione ('**EXTEND**'); nel caso di un file ad accesso diretto, organizzato in modo relativo o a indice, questo può essere stato aperto in scrittura ('**OUTPUT**') o in lettura e scrittura ('**I-O**'), inoltre, se si usa l'accesso sequenziale, è consentito anche in caso di apertura in estensione ('**EXTEND**').

L'istruzione '**WRITE**' viene usata con due schemi sintattici alternativi: uno per i file organizzati in modo sequenziale e l'altro per tutti gli altri casi. Il formato adatto ai file sequenziali contiene, in particolare, opzioni specifiche per l'avanzamento della carta di una stampante.

**File organizzati in modo sequenziale**

```
WRITE record-name [ FROM identifier-1 ]

    .--                                                    --.
    |                       /  /             \  .--     --.  \    |
    |                       |  | identifier-2 |  |  LINE   |  |    |
    |                       |  <              >  |         |  |    |
    |  /       \            |  | integer-1    |  |  LINES  |  |    |
    |  | AFTER |            |  \              /  '--     --'  |    |
    |  <       >  ADVANCING  <                                 >    |
    |  | BEFORE |           |  /                \             |    |
    |  \        /            |  | mnemonic-name |             |    |
    |                       |  <               >             |    |
    |                       |  | PAGE          |             |    |
    |                       \  \               /             /    |
    '--                                                    --'
```

**File organizzati in modo relativo e a indice**

```
WRITE record-name [ FROM identifier-1 ]

    [ INVALID KEY { imperative-statement }... ]
```

Gli schemi sintattici mostrati hanno in comune la prima parte: il nome della variabile che fa riferimento al record, serve a individuare implicitamente il file; la variabile indicata dopo la parola opzionale '**FROM**', permette di copiare tale variabile su quella del record, prima di procedere alla scrittura, come se si usasse l'istruzione '**MOVE**' prima di '**WRITE**':

```
MOVE identifier-1 TO record-name;

WRITE record-name

    [ omissis ]
```

Quando la scrittura avviene con successo, il contenuto del record non è più disponibile in memoria, a meno di averne una copia per altri motivi (per esempio a causa dell'utilizzo dell'opzione '**FROM**').

La scrittura di un file organizzato in modo sequenziale implica l'utilizzo del primo schema sintattico. Nello schema sintattico non è previsto il controllo di alcuna condizione di errore, che comunque potrebbe verificarsi, quando per qualche ragione non è possibile scrivere nel file. Le opzioni '**AFTER ADVANCING**' e '**BEFORE ADVANCING**', servono rispettivamente per richiedere un avanzamento preventivo o successivo alla scrittura. Per un file di dati, non ha significato l'uso di tali opzioni, che invece servono precisamente per la stampa, o per la creazione di file di testo (destinati eventualmente alla stampa). L'avanzamento può essere specificato in un numero intero di righe (***identifier-2*** o ***integer-1***), oppure richiedendo un salto pagina, con la parola chiave '**PAGE**'. Il nome mnemonico che può essere indicato in alternativa alla parola chiave '**PAGE**' può servire per attribuire un nome alternativo proprio alla parola '**PAGE**', oppure a fare riferimento a un'altra parola chiave (alternativa a '**PAGE**'), che si riferisce a caratteristiche speciali, legate alla stampa, che il proprio compilatore può gestire.

Si osservi che un file organizzato in modo sequenziale, per il quale abbiano senso le opzioni di avanzamento, è bene che sia stato dichiarato con l'opzione '**LINE SEQUENTIAL**'.

Il secondo schema sintattico può essere usato per i file che non hanno un'organizzazione sequenziale. In questo caso vengono a mancare i controlli di avanzamento della riga o della pagina, ma si aggiunge la verifica di un errore di scrittura, attraverso l'opzione '**INVALID KEY**', dopo la quale appaiono le istruzioni da eseguire in caso di problemi.

Nel caso di file organizzati in modo relativo, ad accesso sequenziale, il comportamento è lo stesso che si avrebbe con un file sequenziale puro e semplice, con la differenza che la variabile designata a contenere il numero del record viene impostata automaticamente e che si può verificare la condizione controllata dall'opzione '**INVALID KEY**' se si tenta di espandere il file oltre i limiti imposti esternamente al programma. Se invece questo tipo di file viene usato con un accesso diretto o dinamico, il numero del record (inserito nella variabile definita con la dichiarazione '**RELATIVE KEY**' dell'istruzione '**SELECT**' del file stesso) deve essere indicato espressamente: se il numero indicato corrisponde a un record già esistente, oppure se si tenta di scrivere oltre i limiti stabiliti esternamente al programma, si ottiene la condizione di errore controllata dall'opzione '**INVALID KEY**'.

Nel caso di file organizzati a indice, l'inserimento dei record avviene tenendo conto delle chiavi previste per questo. In linea di principio, le chiavi non devono essere doppie; pertanto, il tentativo di inserire un record che contiene una chiave già esistente nel file (primaria o secondaria che sia), provoca un errore che può essere controllato attraverso l'opzione '**INVALID KEY**'. Naturalmente, se nella dichiarazione delle chiavi è stato stabilito che possono anche essere doppie, tale errore non si verifica e la scrittura avviene con successo.

Un file organizzato a indice può essere scritto utilizzando un accesso sequenziale, ma in tal caso, la scrittura deve avvenire rispettando l'ordine crescente della chiave primaria, altrimenti si verifica un errore che si può controllare con l'opzione '**INVALID KEY**'.

L'utilizzo dell'istruzione '**WRITE**' implica l'aggiornamento della variabile che rappresenta lo stato del file ('**FILE STATUS**').

## 72.13 Riordino e fusione

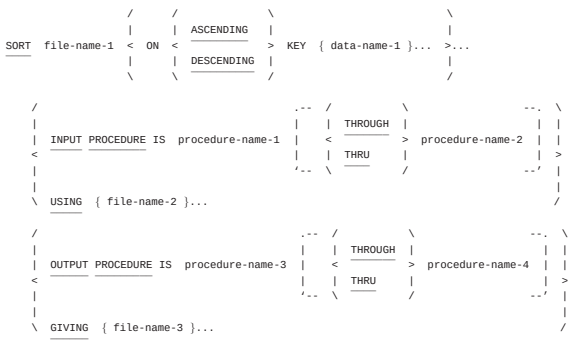
Il riordino e la fusione del contenuto dei file sono gestite normalmente attraverso funzionalità speciali del linguaggio COBOL. Si utilizzano in particolare file dichiarati con l'indicatore di livello '**SD**' nella sezione '**FILE SECTION**', per svolgere la funzione di riordino o di fusione, mentre i file da ordinare o da fondere, assieme al risultato dell'ordinamento o della fusione, possono essere file normali organizzati secondo le esigenze del programma.

I file che prendono parte alle operazioni di riordino e di fusione, **non devono essere aperti o chiusi** durante tali operazioni.

### 72.13.1 Riordino

Il riordino di un file, con l'istruzione '**SORT**' del COBOL, richiede in linea di massima il coinvolgimento di tre file: il file che formalmente serve come appoggio per svolgere la funzione di ordinamento, dichiarato nella sezione '**FILE SECTION**' con l'indicatore di livello

‘**SD**’; un file per i dati in ingresso da ordinare; un file per accogliere il risultato del procedimento di ordinamento.




Il file che nello schema sintattico appare nominato come ***file-name-1***, è quello che deve essere dichiarato nella sezione ‘**FILE SECTION**’ con l’indicatore di livello ‘**SD**’. Dopo la parola ‘**ASCENDING**’ si indica un elenco di chiavi di ordinamento crescenti; dopo la parola ‘**DESCENDING**’ si indica un elenco di chiavi di ordinamento decrescenti. Le chiavi di ordinamento sono campi del record del file ***file-name-1*** e la possibilità di indicare più chiavi serve a definire una gerarchia di ordinamento quando se ne crea la necessità. In pratica, la presenza di più chiavi fa sì che in presenza di chiavi doppie a un certo livello gerarchico, permetta di distinguere l’ordine dei record utilizzando anche le chiavi di livello inferiore.

Nell’ordinamento di un file, la presenza di record con tutte le chiavi previste doppie, ha l’unico inconveniente di non poter stabilire quale sequenza effettiva ottengono tali record dopo l’ordinamento.

Il file da ordinare può essere costituito dal nome che appare dopo la parola ‘**USING**’, oppure può essere generato da un gruppo di procedure del programma, specificate dopo le parole ‘**INPUT PROCEDURE**’. Il file indicato dopo la parola ‘**USING**’ è un file dichiarato normalmente, con l’organizzazione e l’accesso desiderati.

Il file che risulta dall’ordinamento può essere costituito dal nome che appare dopo la parola ‘**GIVING**’, oppure può essere letto da un gruppo di procedure del programma, specificate dopo le parole ‘**OUTPUT PROCEDURE**’. Il file indicato dopo la parola ‘**GIVING**’ è un file dichiarato normalmente, con l’organizzazione e l’accesso desiderati.

La gestione dei dati in ingresso o in uscita, attraverso delle procedure, viene descritto in altre sezioni; per il momento viene mostrato un esempio di ordinamento tipico, che coinvolge il file per il riordino, più due file per i dati (in ingresso e in uscita).

Listato 72.247. Programma elementare che dimostra il funzionamento di ‘**SORT**’.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SORT.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-02-25.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-ORDINARE     ASSIGN TO "input.seq".
001300     SELECT FILE-ORDINATO        ASSIGN TO "output.seq".
001400     SELECT FILE-PER-IL-RIORDINO ASSIGN TO "sort.tmp".
001500*
001600 DATA DIVISION.
001700*
001800 FILE SECTION.
001900*
002000 FD  FILE-DA-ORDINARE.
002100 01  RECORD-DA-ORDINARE           PIC X(10).
```



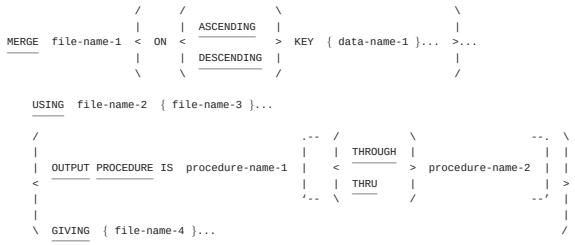
```
002200*
002300 FD  FILE-ORDINATO.
002400 01  RECORD-ORDINATO              PIC X(10).
002500*
002600 SD  FILE-PER-IL-RIORDINO.
002700*
002800 01  RECORD-PER-IL-RIORDINO.
002900     02  CHIAVE-ORDINAMENTO       PIC X(5).
003000     02  FILLER                   PIC X(5).
003100*
003200 PROCEDURE DIVISION.
003300*------------------------- LIVELLO 0 -----------------
003400 MAIN.
003500     SORT FILE-PER-IL-RIORDINO,
003600          ON ASCENDING KEY CHIAVE-ORDINAMENTO,
003700          USING FILE-DA-ORDINARE,
003800          GIVING FILE-ORDINATO.
003900*
004000     STOP RUN.
004100*
```

Nell’esempio, il file usato per ottenere il riordino è ‘`sort.tmp`’; il file da ordinare è ‘`input.seq`’ (organizzato in modo sequenziale); il file ordinato che si ottiene è ‘`output.seq`’ (anche questo organizzato in modo sequenziale). Come chiave di ordinamento si prendono in considerazione i primi cinque byte del record.

Lo schema sintattico consentirebbe l’indicazione di più file da ordinare e di più file ordinati da generare. Nel primo caso, i dati dei vari file vengono raccolti assieme e considerati parte di un file unico da ordinare; i file ordinati da generare, invece, rappresentano copie dello stesso risultato ordinato.

### 72.13.2 Fusione

La fusione di due o più file, con l’istruzione ‘**MERGE**’ del COBOL, richiede la presenza di due o più file ordinati nello stesso modo, che si vogliono mettere insieme in un solo file ordinato. Per compiere questa funzione, si aggiunge un file ulteriore, dichiarato nella sezione ‘**FILE SECTION**’ con l’indicatore di livello ‘**SD**’.




La prima parte dello schema sintattico va interpretata nello stesso modo di quello per il riordino; dove ***file-name-1*** è il file che deve essere dichiarato nella sezione ‘**FILE SECTION**’ con l’indicatore di livello ‘**SD**’ e le variabili indicate dopo le parole ‘**ASCENDING**’ o ‘**DESCENDING**’ sono le chiavi di ordinamento previste nei file in ingresso.

Successivamente si può osservare nello schema sintattico che sono previsti soltanto file in ingresso, dopo la parola ‘**USING**’, tenendo in considerazione il fatto che devono essere almeno due. Questi file **devono risultare già ordinati** secondo le chiavi previste, altrimenti il risultato della fusione non è prevedibile.

Il risultato della fusione può essere costituito dal nome che appare dopo la parola ‘**GIVING**’, oppure può essere letto da un gruppo di procedure del programma, specificate dopo le parole ‘**OUTPUT PROCEDURE**’. Il file indicato dopo la parola ‘**GIVING**’ è un file dichiarato normalmente, con l’organizzazione e l’accesso desiderati.

La gestione dei dati in uscita, attraverso delle procedure, viene descritto in altre sezioni; per il momento viene mostrato un esempio di fusione tipico, si hanno i file ‘`input-1.seq`’ e ‘`input-2.seq`’ ordinati, si vuole ottenere il file ‘`output.seq`’ con la somma dei record, mantenendo l’ordinamento:

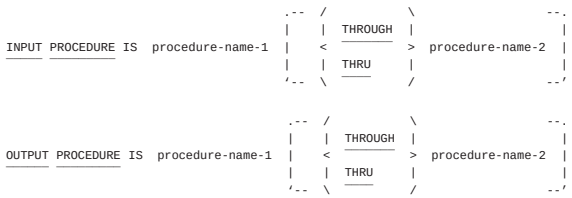
Listato 72.249. Programma elementare che dimostra il funzionamento di '**MERGE**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-MERGE.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-18.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-INPUT-1 ASSIGN TO "input-1.seq".
001300     SELECT FILE-INPUT-2 ASSIGN TO "input-2.seq".
001400     SELECT FILE-OUTPUT  ASSIGN TO "output.seq".
001500     SELECT FILE-PER-LA-FUSIONE
001550                         ASSIGN TO "merge.tmp".
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-INPUT-1
002200 01  RECORD-1                    PIC X(10).
002300*
002400 FD  FILE-INPUT-2
002500 01  RECORD-2                    PIC X(10).
002600*
002700 FD  FILE-OUTPUT
002800 01  RECORD-OUTPUT               PIC X(10).
002900*
003000 SD  FILE-PER-LA-FUSIONE.
003100*
003200 01  RECORD-PER-LA-FUSIONE.
003300     02  CHIAVE-ORDINAMENTO      PIC X(5).
003400     02  FILLER                  PIC X(5).
003500*
003600 PROCEDURE DIVISION.
003700*------------------------ LIVELLO 0 -----------------
003800 MAIN.
003900     MERGE FILE-PER-LA-FUSIONE
004000           ON ASCENDING KEY CHIAVE-ORDINAMENTO,
004100           USING FILE-INPUT-1,
004200                 FILE-INPUT-2,
004300           GIVING FILE-OUTPUT.
004400*
004500     STOP RUN.
004600*
```

Lo schema sintattico consentirebbe l'indicazione di più file ordinati da generare: se viene indicato più di un file per raccogliere il risultato della fusione, questi ottengono lo stesso contenuto; i file in sé possono essere differenti, se possiedono una diversa organizzazione.

### 72.13.3 Gestire i dati in ingresso o in uscita attraverso delle procedure

Nelle istruzioni '**SORT**' e '**MERGE**', a seconda dei casi, esiste la possibilità di specificare un gruppo di procedure con le forme seguenti:




Queste procedure sono da intendere come un intervallo di sezioni o di paragrafi della divisione '**PROCEDURE DIVISION**', da ***procedure-name-1*** a ***procedure-name-2***. Questa porzione di sezione o di paragrafi deve però rispettare delle condizioni: deve servire esclusivamente per lo scopo del riordino o della fusione; non può contenere chiamate a procedure esterne; non può essere usata nel programma per fini differenti.



In generale, se si intendono usare delle procedure per generare dati da ordinare, leggere i dati ordinati o fusi, conviene gestire la divisione '**PROCEDURE DIVISION**' in sezioni. L'esempio seguente mostra proprio una sezione che potrebbe essere usata per leggere il risultato di un file ordinato o fuso:

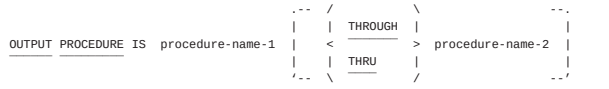
Listato 72.252. Esempio di sezione da usare come procedura di uscita di '**SORT**' o '**MERGE**'.

```
004100 MOSTRA-FILE-ORDINATO SECTION.
004200 INIZIO.
004300     PERFORM MOSTRA-RECORD UNTIL EOF = 1.
004400     GO TO FINE.
004500 MOSTRA-RECORD.
004600     RETURN FILE-PER-IL-RIORDINO RECORD
004700           AT END MOVE 1 TO EOF,
004800                  DISPLAY "FINE DEL FILE ORDINATO".
004900     IF EOF = 0
005000     THEN
005100         DISPLAY RECORD-PER-IL-RIORDINO.
005200 FINE.
005300     EXIT.
```

Nell'esempio si vede anche l'uso del famigerato '**GO TO**', allo scopo di uscire dalla sezione dopo l'esecuzione del ciclo di chiamate al paragrafo '**MOSTRA-RECORD**', dal momento che l'istruzione '**EXIT**', secondo lo standard, deve trovarsi da sola in un paragrafo.

### 72.13.4 Lettura del risultato dell'ordinamento o della fusione attraverso una procedura

Quando si usano le istruzioni '**SORT**' o '**MERGE**', invece di generare un file ordinato o fuso, è possibile leggere il risultato dell'ordinamento o della fusione, specificando la chiamata di un intervallo di procedure (paragrafi o sezioni):




Nell'ambito dell'intervallo di procedure chiamato, occorre usare l'istruzione '**RETURN**' per leggere questi dati dal file di riordino o di fusione:

```
RETURN file-name-1 [NEXT] RECORD  [ INTO identifier ]

    AT END  { imperative-statement }...
```

L'istruzione '**RETURN**' funziona a tutti gli effetti come l'istruzione '**READ**' di un file sequenziale, dove il file indicato è precisamente quello che appare nell'istruzione '**SORT**' o '**MERGE**' chiamante, con la stessa metavariabile.

Listato 72.255. Esempio di lettura del risultato di un ordinamento attraverso una procedura.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   TEST-SORT-4.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-18.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-ORDINARE     ASSIGN TO "input.seq".
001300     SELECT FILE-PER-IL-RIORDINO ASSIGN TO "sort.tmp".
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-ORDINARE.
002000 01  RECORD-DA-ORDINARE           PIC X(10).
002100*
002200 SD  FILE-PER-IL-RIORDINO.
002300*
002400 01  RECORD-PER-IL-RIORDINO.
```

```
002500     02  CHIAVE-ORDINAMENTO       PIC X(5).
002600     02  FILLER                   PIC X(5).
002700
002800 WORKING-STORAGE SECTION.
002900 77  EOF                          PIC 9 VALUE 0.
003000*
003100 PROCEDURE DIVISION.
003200*------------------------ LIVELLO 0 -----------------
003300 MAIN SECTION.
003400 INIZIO.
003500     SORT FILE-PER-IL-RIORDINO,
003600          ON ASCENDING KEY CHIAVE-ORDINAMENTO,
003700          USING FILE-DA-ORDINARE,
003800          OUTPUT PROCEDURE IS MOSTRA-FILE-ORDINATO.
003900*
004000     STOP RUN.
004100*
004200*--------------------- SORT-MERGE PROCEDURE ----------
004300 MOSTRA-FILE-ORDINATO SECTION.
004400 INIZIO.
004500     PERFORM MOSTRA-RECORD UNTIL EOF = 1.
004600     GO TO FINE.
004700 MOSTRA-RECORD.
004800     RETURN FILE-PER-IL-RIORDINO RECORD
004900            AT END MOVE 1 TO EOF,
005000                   DISPLAY "FINE DEL FILE ORDINATO".
005100     IF EOF = 0
005200     THEN
005300         DISPLAY RECORD-PER-IL-RIORDINO.
005400 FINE.
005500     EXIT.
005600*
```

L'esempio riguarda la visualizzazione di un file ordinato, senza generare il file stesso, ma si applica tale e quale al caso della fusione.

### 72.13.5 Acquisizione dei dati per il riordino da una procedura

Limitatamente al caso del riordino, con l'istruzione '**SORT**', è possibile acquisire i record da riordinare attraverso una procedura:

```
                                  .--  /          \                --.
                                  |    |  THROUGH  |                  |
INPUT PROCEDURE IS  procedure-name-1  |    <          >  procedure-name-2  |
                                  |    |  THRU     |                  |
                                  '--  \          /                --'
```

Nell'ambito dell'intervallo di procedure chiamato, occorre usare l'istruzione '**RELEASE**' per passare formalmente un record. L'istruzione '**RELEASE**' si utilizza e si comporta come l'istruzione '**WRITE**' per i file sequenziali:

```
WRITE  record-name  [ FROM identifier-1 ]
```

Il record è il nome della variabile strutturata corrispondente del file che esegue in pratica l'ordinamento, ovvero quello che nello schema sintattico dell'istruzione '**SORT**' appare come ***file-name-1***.

Listato 72.258. Esempio di acquisizione di record da ordinare attraverso l'inserimento diretto.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    TEST-SORT-3.
000300 AUTHOR.        DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-18.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-PER-IL-RIORDINO ASSIGN TO "sort.tmp".
001300*
001400 DATA DIVISION.
001500*
001600 FILE SECTION.
001700*
001800 SD  FILE-PER-IL-RIORDINO.
001900*
002000 01  RECORD-PER-IL-RIORDINO.
```



```
002100     02  CHIAVE-ORDINAMENTO       PIC X(5).
002200     02  FILLER                   PIC X(5).
002300
002400 WORKING-STORAGE SECTION.
002500 77  EOJ                          PIC 9 VALUE 0.
002600 77  EOF                          PIC 9 VALUE 0.
002700 77  DATI-INSERITI                PIC X(10).
002800*
002900 PROCEDURE DIVISION.
003000*------------------------ LIVELLO 0 -----------------
003100 MAIN SECTION.
003200 INIZIO.
003300     SORT FILE-PER-IL-RIORDINO,
003400          ON ASCENDING KEY CHIAVE-ORDINAMENTO,
003500          INPUT PROCEDURE IS INSERIMENTO-DATI,
003600          OUTPUT PROCEDURE IS MOSTRA-FILE-ORDINATO.
003700*
003800     STOP RUN.
003900*
004000*--------------------- SORT-MERGE PROCEDURE ----------
004100 MOSTRA-FILE-ORDINATO SECTION.
004200 INIZIO.
004300     PERFORM MOSTRA-RECORD UNTIL EOF = 1.
004400     GO TO FINE.
004500 MOSTRA-RECORD.
004600     RETURN FILE-PER-IL-RIORDINO RECORD
004700            AT END MOVE 1 TO EOF,
004800                   DISPLAY "FINE DEL FILE ORDINATO".
004900     IF EOF = 0
005000     THEN
005100         DISPLAY RECORD-PER-IL-RIORDINO.
005200 FINE.
005300     EXIT.
005400*-----------------------------------------------------
005500 INSERIMENTO-DATI SECTION.
005600 INIZIO.
005700     PERFORM INSERISCI-RECORD UNTIL EOJ = 1.
005800     GO TO FINE.
005900 INSERISCI-RECORD.
006000     DISPLAY "INSERISCI UN RECORD DA 10 CARATTERI:".
006100     ACCEPT DATI-INSERITI.
006200     IF DATI-INSERITI = SPACES
006300     THEN
006400         MOVE 1 TO EOJ;
006500     ELSE
006600         MOVE DATI-INSERITI TO RECORD-PER-IL-RIORDINO,
006700         RELEASE RECORD-PER-IL-RIORDINO.
006800 FINE.
006900     EXIT.
007000*
```

L'esempio è completo, in quanto anche il risultato del riordino viene gestito tramite una procedura. Nella fase di inserimento dati, si può osservare che un inserimento nullo (pari all'inserimento di tutti spazi), implica la conclusione di quella fase.

## 72.14 Riferimenti


- Christopher Heng, *Free COBOL compilers and interpreters*, http://www.thefreecountry.com/compilers/cobol.shtml
- *Programming manuals and tutorials, COBOL*, http://www.theamericanprogrammer.com/programming/manuals.cobol.html
- *MPE/iX and HP e3000 Technical Documentation, HP COBOL II/XL*
  - *Programmer's guide*, http://wayback.archive.org/web/2006*/http://docs.hp.com/en/424/31500-90014.pdf
  - *Quick reference guide*, http://wayback.archive.org/web/2006*/http://docs.hp.com/en/425/31500-90015.pdf
  - *Reference manual*, http://wayback.archive.org/web/2006*/http://docs.hp.com/en/426/31500-90013.pdf
- *Compaq COBOL Reference Manual*, http://www.helsinki.fi/atk/unix/dec_manuals/cobv27ua27/cobrm_contents.htm

# Programmare in COBOL

Questo capitolo tratta di casi pratici di programmazione in linguaggio COBOL, con l'intento di recuperare un vecchio lavoro realizzato con il sostegno di Antonio Bernardi, durante i primi anni 1980, utilizzando un elaboratore Burroughs B91.

Figura 73.1. *Mainframe* Burroughs B1900 del 1985: un sogno mai realizzato. La foto originale proviene da *http://www.kiwanja.net/photos.htm* ed è di Ken Banks. La foto viene riprodotta qui con il permesso del suo autore.


## 73.1 Preparazione

Il linguaggio COBOL nasce quando l'inserimento dei dati in un elaboratore avveniva principalmente attraverso schede perforate, pertanto, da questo derivano delle limitazioni nel modo in cui vanno scritte le sue direttive.

### 73.1.1 Problema del modulo di programmazione

Il linguaggio COBOL nasce imponendo dei vincoli al modo di utilizzare gli spazi orizzontali nel file del sorgente. Questi vincoli consentivano di amministrare con un certo criterio la procedura di perforazione e riutilizzo delle schede perforate.

Terminata l'era delle schede perforate, i compilatori hanno cominciato a essere più disponibili e ad accettare codice COBOL scritto senza rispettare i vincoli del modulo di programmazione tradizionale (normalmente viene eliminato l'obbligo della numerazione delle righe e l'area in cui è possibile scrivere le istruzioni si estende per un numero indefinito di colonne, cancellando la funzione della zona identificativa del programma); tuttavia, il suggerimento che qui viene dato è di continuare a usare il modello originale, considerata la particolarità del linguaggio di programmazione, che perderebbe la sua logica estetica. Il listato successivo mostra l'esempio di un programma COBOL molto breve, dove si può vedere l'utilizzo delle varie aree secondo il criterio del modulo di programmazione del linguaggio.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0100.
000300 AUTHOR.       DANIELE GIACOMINI.
```



```
000400 DATE-WRITTEN. 1985-02-12.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.                                  WSS-0000
001100 01  A PIC 9(7).                                          WSS-0000
001200 01  B PIC 9(7).                                          WSS-0000
001300 01  C PIC 9(14).                                         WSS-0000
001400*
001500 PROCEDURE DIVISION.
001600*-----------------------------------------------------------------
001700 MAIN.
001800     DISPLAY "MOLTIPLICAZIONE DI DUE NUMERI".
001900     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
002000     ACCEPT A.
002100     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
002200     ACCEPT B.
002300     COMPUTE C = A * B.
002400     DISPLAY C.
002500*
002600     STOP RUN.
002700*
```

Nell'esempio si può osservare: l'uso dell'asterisco nella settima colonna per indicare un commento; la presenza di direttive che iniziano a dalla colonna ottava e di altre che iniziano dalla colonna dodicesima; l'indicazione di un'etichetta distintiva nelle otto colonne finali ('**`WSS-0000`**'), in corrispondenza di alcune righe (probabilmente per ricordare che quella porzione proviene da un altro programma).

Si osservi che quanto appare nelle ultime otto colonne non ha valore per il linguaggio di programmazione, ma rappresenta un modo per individuare gruppi di righe che possono avere qualche tipo di importanza, oppure qualunque altro tipo di annotazione.

Generalmente, i compilatori consentono di specificare con quale formato viene fornito il file sorgente; la scelta è normalmente tra un formato «fisso» (tradizionale), oppure libero (senza vincoli particolari).

Dal momento che attualmente la numerazione delle righe è divenuta puramente un fatto estetico, ci si può aiutare con uno script per rinumerare il sorgente. Il listato successivo mostra uno script molto semplice, che presuppone di ricevere dallo standard input un file sorgente con i numeri di riga, anche se errati, emettendo lo stesso sorgente attraverso lo standard output, ma con una numerazione progressiva uniforme.

```
#!/bin/sh
#
# cobol-line-renumber.sh INCREMENT < SOURCE_COB > NEW_SOURCE_COB
#
INCREMENT="$1"
LINE=""
NUMBER="0"
NUMBER_FORMATTED=""
#
while read LINE
do
    NUMBER=$(($NUMBER+$INCREMENT))
    NUMBER_FORMATTED=`printf %000006d $NUMBER`
    LINE=`echo "$LINE" | sed s/^[0-9][0-9][0-9][0-9][0-9][0-9]//`
    LINE="$NUMBER_FORMATTED$LINE"
    echo "$LINE"
done
```

In pratica, supponendo che lo script si chiami '**`cobol-line-renumber.sh`**', si potrebbe usare come nell'esempio seguente:

```
$ cobol-line-renumber.sh < sorgente.cob > rinumerato.cob [Invio]
```

#### 73.1.1.1 Compatibilità con i compilatori

I compilatori nati dopo la fine delle schede perforate possono essere più o meno disposti ad accettare la presenza della numerazione delle righe o delle colonne finali di commento. Generalmente questi compilatori consentono di indicare un'opzione che specifica il formato del sorgente; tuttavia si può utilizzare uno script simile a quello seguente, per eliminare le colonne della numerazione delle righe e le colonne descrittive di identificazione del programma:

```
#!/usr/bin/perl
#
# cobol-compile SOURCE_COB SOURCE_COB_NEW
#
use utf8;
binmode (STDOUT, ":utf8");
binmode (STDERR, ":utf8");
binmode (STDIN,  ":utf8");
#
$source=$ARGV[0];
$source_new=$ARGV[1];
$line="";
#
open (SOURCE,     "<:utf8", "$source");
open (SOURCE_NEW, ">:utf8", "$source_new");
#
while ($line = <SOURCE>)
  {
    chomp ($line);
    $line =~ m/^[0-9][0-9][0-9][0-9][0-9][0-9](.*)$/;
    $line = $1;
    if ($line =~ m/^(.{66}).*$/)
      {
        $line = $1;
      }
    print SOURCE_NEW ("$line\n");
  }
close (SOURCE_NEW);
close (SOURCE);
#
```

Eventualmente, se il problema consistesse soltanto nella rimozione del numero di riga, si potrebbe usare uno script molto più semplice:

```
#!/bin/sh
#
# cobol-compile SOURCE_COB SOURCE_COB_NEW
#
SOURCE="$1"
SOURCE_NEW="$2"
cat $SOURCE | sed s/^[0-9][0-9][0-9][0-9][0-9][0-9]//g > $SOURCE_NEW
```

### 73.1.2 Riepilogo di alcuni concetti importanti del linguaggio

In generale, le istruzioni del linguaggio COBOL sono da intendere come frasi scritte in inglese, che terminano con un punto fermo. In certe situazioni, si riuniscono più istruzioni in un'unica «frase», che termina con un punto, ma in tal caso, spesso si usa la virgola e il punto e virgola per concludere le istruzioni singole.

Le istruzioni del linguaggio si compongono in linea di massima di parole chiave, costanti letterali e operatori matematici. Le parole chiave sono scritte usando lettere maiuscole (dell'alfabeto inglese) e il trattino normale ('`-`'). In generale, i simboli che si possono usare nel linguaggio sono abbastanza limitati, con l'eccezione del contenuto delle costanti alfanumeriche letterali, che teoricamente potrebbero contenere qualunque simbolo (escluso quello che si usa come delimitatore) secondo le potenzialità del compilatore particolare.

Tabella 73.6. I simboli disponibili nel linguaggio.

| Simboli | Descrizione | Simboli | Descrizione |
|---|---|---|---|
| '`0`'..'`9`' | cifre numeriche | '`A`'..'`Z`' | lettere maiuscole dell'alfabeto inglese (latino) |
| ' ' | spazio | | |
| '`+`' | segno più | '`-`' | segno meno o trattino |
| '`*`' | asterisco | '`/`' | barra obliqua |
| '`$`' | dollaro o segno di valuta | '`,`' | virgola |
| '`;`' | punto e virgola | '`.`' | punto fermo |
| '`(`' | parentesi aperta | '`)`' | parentesi chiusa |
| '`<`' | minore | '`>`' | maggiore |

Le parole chiave più importanti del linguaggio sono dei «verbi» imperativi, che descrivono un comando che si vuole sia eseguito. Un gruppo interessante di parole chiave è rappresentato dalle «costanti figurative», che servono a indicare verbalmente delle costanti di uso comune. Per esempio, la parola chiave '`ZERO`' rappresenta uno o più zeri, in base al contesto.



Le stringhe sono delimitate da virgolette (apici doppi) e di solito non sono previste forme di protezione per incorporare le virgolette stesse all'interno delle stringhe: per questo occorre suddividere le stringhe, concatenandole con la costante figurativa '`QUOTE`'.

La gestione numerica del COBOL è speciale rispetto ai linguaggi di programmazione comuni, perché le variabili vengono dichiarate con la loro dimensione di cifre esatta, stabilendo anche la quantità di decimali e il modo in cui l'informazione deve essere gestita. In pratica, si stabilisce il modo in cui il valore deve essere rappresentato, lasciando al compilatore il compito di eseguire ogni volta tutte le conversioni necessarie. Sotto questo aspetto, un programma COBOL ha una gestione per i valori numerici molto pesante, quindi più lenta rispetto ad altri linguaggi, dove i valori numerici sono gestiti in base alle caratteristiche fisiche della CPU e le conversioni di tipo devono essere dichiarate esplicitamente.

Le variabili usate nel linguaggio sono sempre globali e come tali vanno dichiarate in una posizione apposita. Tali variabili, salvo situazioni eccezionali, fanno sempre parte di un record, inteso come una raccolta di campi di informazioni. Questa gestione particolare costringe a stabilire esattamente le dimensioni che ogni informazione deve avere se registrata nella memoria di massa (dischi, nastri o altro) o se stampata. In un certo senso, questa caratteristica può impedire o rendere difficile l'uso di una forme di codifica dei caratteri che preveda una dimensione variabile degli stessi, considerato che i record possono essere rimappati, trattando anche valori numerici come insiemi di cifre letterali.

Questo particolare, che non è affatto di poco conto, suggerisce di usare il linguaggio per gestire dati rappresentabili con il codice ASCII tradizionale, ovvero con i primi 127 punti di codifica (da U+0000 a U+007F). Naturalmente sono disponibili compilatori che permettono di superare questo problema, ma in tal caso occorre verificare come vengono gestiti effettivamente i dati.

Le istruzioni COBOL possono essere scritte usando più righe, avendo l'accortezza di continuare a partire dall'area «B»; in generale non c'è bisogno di indicare esplicitamente che l'istruzione sta continuando nella riga successiva, perché si usa il punto fermo per riconoscere la loro conclusione. Tuttavia, in situazioni eccezionali, si può spezzare una parola chiave o anche una stringa letterale; in tal caso, nella settima colonna delle riga che continua, va inserito il segno '`-`', inoltre, se si tratta di una stringa, la sua ripresa va iniziata nuovamente con le virgolette. A ogni modo, considerato che difficilmente si devono scrivere parole chiave molto lunghe e che le stringhe letterali si possono concatenare, è auspicabile che la continuazione nella riga successiva con l'indicatore nella settima colonna sia evitata del tutto.

I commenti nel sorgente si indicano inserendo un asterisco nella settima colonna; se invece si mette una barra obliqua ('`/`') si vuole richiedere un salto pagina, in fase di stampa, ammesso che il compilatore preveda questo.

### 73.1.3 TinyCOBOL

TinyCOBOL[1] è un compilatore COBOL che tende alla conformità con gli standard del 1985. Come per ogni compilatore COBOL ci sono delle differenze rispetto al linguaggio «standard», in particolare è disponibile la possibilità di recepire gli argomenti della riga di comando e di accedere ai flussi standard dei sistemi Unix (standard input, standard output e standard error).

La compilazione di un programma si ottiene attraverso il programma '`htcobol`', che, salvo l'uso dell'opzione '`-F`', si aspetta di trovare un sorgente senza numerazione delle righe e senza il blocco descrittivo finale delle colonne da 73 a 80. In pratica, ciò consentirebbe si disporre di un'area B (per le istruzioni) molto più ampia.

```
htcobol [opzioni] file_sorgente_cobol
```

Il programma '**htcobol**' si aspetta che il file sorgente abbia un nome con un'estensione '.cob' e, in tal caso, l'estensione può anche essere omessa. Se non si specificano opzioni, si ottiene un file eseguibile con lo stesso nome del sorgente, ma senza l'estensione '.cob'.

Tabella 73.7. Alcune opzioni.

| Opzione | Descrizione |
|---|---|
| `-o` ***file*** | Richiede che il file generato dalla compilazione abbia il nome stabilito dall'argomento dell'opzione. |
| `-X` | Richiede che il file sorgente sia scritto senza numerazione delle righe e senza commenti nelle colonne da 73 a 80; tuttavia questa è la modalità di funzionamento predefinita. |
| `-F` | Richiede che il file sorgente sia scritto secondo il formato tradizionale (con la numerazione delle righe e con il limite dell'area «B»). |

Vengono mostrati alcuni esempi.

- `$` **`htcobol -F esempio.cob`** [*Invio*]

  Compila il programma '`esempio.cob`', generando il file eseguibile '**`esempio`**'. Se non vengono riscontrati errori, la compilazione non genera alcun messaggio.

- `$` **`htcobol -F -o programma esempio.cob`** [*Invio*]

  Compila il programma '`esempio.cob`', generando il file eseguibile '**`programma`**'. Se non vengono riscontrati errori, la compilazione non genera alcun messaggio.

### 73.1.4 OpenCOBOL

OpenCOBOL[2] è un compilatore COBOL che genera codice in linguaggio C e si avvale di GCC per arrivare a produrre il file eseguibile finale. In generale si utilizza per la compilazione il programma '**cobc**' che si prende cura di tutti i passaggi necessari:

```
cobc [opzioni] file_sorgente_cobol
```

Tabella 73.8. Alcune opzioni.

| Opzione | Descrizione |
|---|---|
| `-free` | Richiede che il file sorgente sia scritto in formato «libero» (senza i vincoli della numerazione delle righe e senza commenti nelle colonne da 73 a 80. |
| `-fixed` | Richiede che il file sorgente sia scritto secondo il formato tradizionale (con la numerazione delle righe e con il limite tradizionale dell'area «B»). |

L'esempio seguente compila il file '`esempio.cob`' e genera il file eseguibile '**`esempio`**':

`$` **`cobc esempio.cob`** [*Invio*]

## 73.2 Esempi elementari

Qui si raccolgono alcuni esempi elementari di programmi COBOL, risalenti a un lavoro didattico del 1985. Salvo dove indicato in maniera differente, gli esempi mostrati funzionano regolarmente se compilati con OpenCOBOL 0.31.

### 73.2.1 ELM0100: prodotto tra due numeri

Variabili

'**A**' è il moltiplicando;

'**B**' è il moltiplicatore;

'**C**' è il risultato.

Descrizione

Il calcolo viene eseguito attraverso l'istruzione '**COMPUTE**'.



Paragrafo '**MAIN**'

Il programma si svolge unicamente all'interno di questo paragrafo. Il programma riceve dall'esterno i valori per le variabili '**A**' e '**B**', esegue il prodotto tramite l'istruzione '**COMPUTE**' mettendo il risultato nella variabile '**C**'.

Viene visualizzato il contenuto della variabile '**C**' con l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0100.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-12.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  A PIC 9(7).
001200 01  B PIC 9(7).
001300 01  C PIC 9(14).
001400*
001500 PROCEDURE DIVISION.
001600*-------------------------------------------------------
001700 MAIN.
001800     DISPLAY "MOLTIPLICAZIONE DI DUE NUMERI".
001900     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
002000     ACCEPT A.
002100     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
002200     ACCEPT B.
002300     COMPUTE C = A * B.
002400     DISPLAY C.
002500*
002600     STOP RUN.
002700*
```

### 73.2.2 ELM0200: prodotto tra due numeri

Variabili

'**A**' è il moltiplicando;

'**B**' è il moltiplicatore;

'**C**' è il risultato; questa variabile viene inizializzata a zero in fase di dichiarazione.

Descrizione

Il calcolo viene eseguito sommando alla variabile '**C**' la variabile '**A**' per '**B**' volte.

Paragrafo '**MAIN**'

Il programma riceve dall'esterno i valori per le variabili '**A**' e '**B**'. Attraverso l'istruzione '**PERFORM**' viene eseguito il paragrafo '**SOMMA**' per '**B**' volte; al termine di questo ciclo il risultato della moltiplicazione si trova nella variabile '**C**', che viene visualizzato con l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**SOMMA**'

Il paragrafo somma al contenuto della variabile '**C**' il contenuto della variabile '**A**'. Dal momento che questo paragrafo viene eseguito '**B**' volte, la variabile '**C**' finisce con il contenere il risultato del prodotto di «A×B».

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0200.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  A PIC 9(7).
001200 01  B PIC 9(7).
```

```
001300 01  C PIC 9(14)   VALUE ZERO.
001400*
001500 PROCEDURE DIVISION.
001600*------------------------ LIVELLO 0 -----------------
001700 MAIN.
001800     DISPLAY "MOLTIPLICAZIONE DI DUE NUMERI".
001900     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
002000     ACCEPT A.
002100     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
002200     ACCEPT B.
002300     PERFORM SOMMA B TIMES.
002400     DISPLAY C.
002500*
002600     STOP RUN.
002700*------------------------ LIVELLO 1 -----------------
002800 SOMMA.
002900     COMPUTE C = C + A.
003000*
```

### 73.2.3 ELM0300: prodotto tra due numeri

Variabili

'**A**' è il moltiplicando;

'**B**' è il moltiplicatore;

'**C**' è il risultato.

Descrizione

Il calcolo viene eseguito sommando alla variabile '**C**' la variabile '**A**' per '**B**' volte. Per ogni esecuzione di tale somma, la variabile '**B**' viene diminuita di una unità, cosicché il ciclo delle somme viene arrestato quando '**B**' è ormai a zero.

Paragrafo '**MAIN**'

Vengono ricevuti dall'esterno i valori per le variabili '**A**' e '**B**'. Viene eseguito tramite l'istruzione '**PERFORM**' il paragrafo '**SOMMA**' fino a quando la variabile '**B**' raggiunge lo zero. A quel punto la variabile '**C**' contiene il risultato del prodotto, che viene visualizzato con l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**SOMMA**'

Inizialmente viene decrementato di una unità il contenuto della variabile '**B**', quindi viene sommato al contenuto di '**C**' il valore di '**A**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0300.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-04-13.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  A PIC 9(7).
001200 01  B PIC 9(7).
001300 01  C PIC 9(14)   VALUE ZERO.
001400*
001500 PROCEDURE DIVISION.
001600*------------------------ LIVELLO 0 -----------------
001700 MAIN.
001800     DISPLAY "MOLTIPLICAZIONE DI DUE NUMERI".
001900     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
002000     ACCEPT A.
002100     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
002200     ACCEPT B.
002300     PERFORM SOMMA UNTIL B = 0.
002400     DISPLAY C.
002500*
002600     STOP RUN.
002700*------------------------ LIVELLO 1 -----------------
002800 SOMMA.
002900     COMPUTE B = B - 1.
003000     COMPUTE C = C + A.
003100*
```



### 73.2.4 ELM0400: prodotto tra due numeri

Variabili

'**A**' è il moltiplicando;

'**B**' è il moltiplicatore;

'**C**' è il risultato;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo.

Descrizione

Il calcolo viene eseguito sommando alla variabile '**C**' la variabile '**A**' per '**B**' volte. Per ogni esecuzione di tale somma, la variabile '**B**' viene diminuita di una unità, cosicché il ciclo delle somme viene arrestato quando '**B**' è ormai a zero.

Il programma si arresta solo se gli viene dato un comando apposito, altrimenti continua a richiedere altri dati per l'esecuzione di un altro prodotto.

Paragrafo '**MAIN**'

Vengono ricevuti dall'esterno i valori per le variabili '**A**' e '**B**' tramite il paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Viene eseguito tramite l'istruzione '**PERFORM**' il paragrafo '**SOMMA**' ripetutamente, terminando il ciclo quando la variabile '**B**' contiene il valore zero. A quel punto, la variabile '**C**' contiene il risultato del prodotto, che viene visualizzato con l'istruzione '**DISPLAY**'.

Il programma riceve dall'esterno una parola: un '**SI**' o un '**NO**'; se viene fornita la stringa '**SI**' (scritta con lettere maiuscole) il programma azzera il contenuto della variabile '**C**' ed esegue il paragrafo '**INSERIMENTO-DATI**', altrimenti, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-DATI**'

Il paragrafo riceve dall'esterno i valori per le variabili '**A**' e '**B**'.

Paragrafo '**SOMMA**'

Inizialmente viene decrementato di una unità il contenuto della variabile '**B**', quindi viene sommato al contenuto di '**C**' il valore di '**A**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0400.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  A        PIC 9(7).
001200 01  B        PIC 9(7).
001300 01  C        PIC 9(14)   VALUE ZERO.
001400 01  EOJ      PIC 9       VALUE ZERO.
001500 01  RISPOSTA PIC XX.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM INSERIMENTO-DATI.
002100     PERFORM LAVORO UNTIL EOJ = 1.
002200*
002300     STOP RUN.
002400*------------------------ LIVELLO 1 -----------------
002500 LAVORO.
002600     PERFORM SOMMA UNTIL B = 0.
002700     DISPLAY C.
002800*
```

```
002900     DISPLAY "VUOI CONTINUARE? SI O NO".
003000     ACCEPT RISPOSTA.
003100*
003200     IF RISPOSTA = "SI"
003300       THEN
003400           MOVE ZERO TO C,
003500           PERFORM INSERIMENTO-DATI;
003600       ELSE
003700           MOVE 1 TO EOJ.
003800*------------------------ LIVELLO 2 -----------------
003900 INSERIMENTO-DATI.
004000     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
004100     ACCEPT A.
004200     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
004300     ACCEPT B.
004400*-------------------------------------------------------
004500 SOMMA.
004600     COMPUTE B = B - 1.
004700     COMPUTE C = C + A.
004800*
```

### 73.2.5 ELM0500: prodotto tra due numeri

Variabili

'**A**' è il moltiplicando;

'**B**' è il moltiplicatore;

'**C**' è il risultato;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo.

Descrizione

Il calcolo viene eseguito sommando alla variabile '**C**' la variabile '**A**' per '**B**' volte. Il controllo di questa somma viene effettuato da un ciclo '**PERFORM VARYING**' che decrementa di una unità la variabile '**B**', partendo dal suo valore iniziale, fino a quando si riduce a zero, nel qual caso il ciclo si arresta.

Paragrafo '**MAIN**'

Vengono ricevuti dall'esterno i valori per le variabili '**A**' e '**B**' tramite il paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Viene eseguito tramite l'istruzione '**PERFORM**' il paragrafo '**SOMMA**' ripetutamente, decrementando il valore della variabile '**B**', fino a zero, quando il ciclo termina. A quel punto, la variabile '**C**' contiene il risultato del prodotto, che viene visualizzato con l'istruzione '**DISPLAY**'.

Il programma riceve dall'esterno una parola: un '**SI**' o un '**NO**'; se viene fornita la stringa '**SI**' (scritta con lettere maiuscole) il programma azzera il contenuto della variabile '**C**' ed esegue il paragrafo '**INSERIMENTO-DATI**', altrimenti, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-DATI**'

Il paragrafo riceve dall'esterno i valori per le variabili '**A**' e '**B**'.

Paragrafo '**SOMMA**'

Viene sommato al contenuto di '**C**' il valore di '**A**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0500.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  A        PIC 9(7).
```



```
001200 01  B        PIC 9(7).
001300 01  C        PIC 9(14)   VALUE ZERO.
001400 01  EOJ      PIC 9       VALUE ZERO.
001500 01  RISPOSTA PIC XX.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM INSERIMENTO-DATI.
002100     PERFORM LAVORO UNTIL EOJ = 1.
002200*
002300     STOP RUN.
002400*------------------------ LIVELLO 1 -----------------
002500 LAVORO.
002600     PERFORM SOMMA VARYING B FROM B BY -1 UNTIL B = 0.
002700     DISPLAY C.
002800*
002900     DISPLAY "VUOI CONTINUARE? SI O NO".
003000     ACCEPT RISPOSTA.
003100*
003200     IF RISPOSTA = "SI"
003300       THEN
003400           MOVE ZERO TO C,
003500           PERFORM INSERIMENTO-DATI;
003600       ELSE
003700           MOVE 1 TO EOJ.
003800*------------------------ LIVELLO 2 -----------------
003900 INSERIMENTO-DATI.
004000     DISPLAY "INSERISCI IL PRIMO ELEMENTO".
004100     ACCEPT A.
004200     DISPLAY "INSERISCI IL SECONDO ELEMENTO".
004300     ACCEPT B.
004400*-------------------------------------------------------
004500 SOMMA.
004600     COMPUTE C = C + A.
004700*
```

### 73.2.6 ELM0600: inserimento dati in un vettore

Variabili

'**RECORD-ELEMENTI**' è una variabile che si scompone in un array;

'**ELEMENTO**' è l'array che costituisce '**RECORD-ELEMENTI**';

'**INDICE**' è l'indice usato per scandire gli elementi;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo.

Descrizione

Il programma esegue semplicemente un inserimento di dati all'interno degli elementi dell'array, con un accesso libero (bisogna ricordare che l'indice del primo elemento è uno), specificando prima l'indice e poi il valore (il carattere) da attribuire all'elemento.

Paragrafo '**MAIN**'

Viene eseguito una volta il paragrafo '**INSERIMENTO-INDICE**', che serve a ricevere il valore dell'indice di inserimento dall'utente.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Viene visualizzato il valore di tutta la variabile '**RECORD-ELEMENTI**', attraverso l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma riceve dall'esterno il valore per '**ELEMENTO(INDICE)**'.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire, viene eseguito il paragrafo '**INSERIMENTO-INDICE**', altrimenti viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-INDICE**'

Il programma riceve dall'esterno il valore per la variabile '**INDICE**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0600.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 9 TIMES.
001300 01  INDICE        PIC 9.
001400 01  EOJ           PIC 9   VALUE ZERO.
001500 01  RISPOSTA      PIC XX.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM INSERIMENTO-INDICE.
002100     PERFORM LAVORO UNTIL EOJ = 1.
002200     DISPLAY RECORD-ELEMENTI.
002300*
002400     STOP RUN.
002500*------------------------ LIVELLO 1 -----------------
002600 LAVORO.
002700     DISPLAY "INSERISCI I DATI DI UN ELEMENTO ",
002750             "(UN SOLO CARATTERE)".
002800     ACCEPT ELEMENTO(INDICE).
002900*
003000     DISPLAY "VUOI CONTINUARE? SI O NO".
003100     ACCEPT RISPOSTA.
003200*
003300     IF RISPOSTA = "SI"
003400       THEN
003500           PERFORM INSERIMENTO-INDICE;
003600       ELSE
003700           MOVE 1 TO EOJ.
003800*------------------------ LIVELLO 2 -----------------
003900 INSERIMENTO-INDICE.
004000     DISPLAY "INSERISCI L'INDICE".
004100     ACCEPT INDICE.
004200*
```

### 73.2.7 ELM0700: inserimento dati in un vettore

Variabili

'**RECORD-ELEMENTI**' è una variabile che si scompone in un array;

'**ELEMENTO**' è l'array che costituisce '**RECORD-ELEMENTI**';

'**INDICE**' è l'indice usato per scandire gli elementi;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo.

Descrizione

Il programma esegue semplicemente un inserimento di dati all'interno degli elementi dell'array, con un accesso libero (bisogna ricordare che l'indice del primo elemento è uno), specificando prima l'indice e poi il valore (il carattere) da attribuire all'elemento.

Se l'indice che si inserisce è zero, viene richiesto nuovamente di fornire un dato valido.

Paragrafo '**MAIN**'

Viene eseguito paragrafo '**INSERIMENTO-INDICE**', che serve a ricevere il valore dell'indice di inserimento dall'utente, ripetendo l'operazione se il valore fornito è minore o uguale a zero.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.



Viene visualizzato il valore di tutta la variabile '**RECORD-ELEMENTI**', attraverso l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma riceve dall'esterno il valore per '**ELEMENTO(INDICE)**'.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire, dopo l'azzeramento della variabile '**INDICE**' viene eseguito il paragrafo '**INSERIMENTO-INDICE**', ripetutamente, ponendo come condizione di conclusione il fatto che la variabile '**INDICE**' abbia un valore maggiore di zero. Se invece l'utente rinuncia a proseguire, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-INDICE**'

Il programma riceve dall'esterno il valore per la variabile '**INDICE**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0700.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 9 TIMES.
001300 01  INDICE        PIC 9.
001400 01  EOJ           PIC 9   VALUE ZERO.
001500 01  RISPOSTA      PIC XX.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM INSERIMENTO-INDICE UNTIL INDICE > ZERO.
002100     PERFORM LAVORO UNTIL EOJ = 1.
002200     DISPLAY RECORD-ELEMENTI.
002300*
002400     STOP RUN.
002500*------------------------ LIVELLO 1 -----------------
002600 LAVORO.
002700     DISPLAY "INSERISCI I DATI DI UN ELEMENTO ",
002750             "(UN SOLO CARATTERE)".
002800     ACCEPT ELEMENTO(INDICE).
002900*
003000     DISPLAY "VUOI CONTINUARE? SI O NO".
003100     ACCEPT RISPOSTA.
003200*
003300     IF RISPOSTA = "SI"
003400       THEN
003500           MOVE ZERO TO INDICE,
003600           PERFORM INSERIMENTO-INDICE
003650                   UNTIL INDICE > ZERO;
003700       ELSE
003800           MOVE 1 TO EOJ.
003900*------------------------ LIVELLO 2 -----------------
004000 INSERIMENTO-INDICE.
004100     DISPLAY "INSERISCI L'INDICE".
004200     ACCEPT INDICE.
004300*
```

### 73.2.8 ELM0800: inserimento dati in un vettore

Variabili

'**RECORD-ELEMENTI**' è una variabile che si scompone in un array;

'**ELEMENTO**' è l'array che costituisce '**RECORD-ELEMENTI**';

'**INDICE**' è l'indice usato per scandire gli elementi;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo.

Descrizione

Il programma esegue semplicemente un inserimento di dati all'interno degli elementi dell'array, con un accesso libero (bisogna ricordare che l'indice del primo elemento è uno), specificando prima l'indice e poi il valore (il carattere) da attribuire all'elemento.

Se l'indice che si inserisce è zero, viene richiesto nuovamente di fornire un dato valido.

Paragrafo '**MAIN**'

Viene eseguito paragrafo '**INSERIMENTO-INDICE**', che serve a ricevere il valore dell'indice di inserimento dall'utente.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Viene visualizzato il valore di tutta la variabile '**RECORD-ELEMENTI**', attraverso l'istruzione '**DISPLAY**'.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma riceve dall'esterno il valore per '**ELEMENTO(INDICE)**'.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire viene eseguito il paragrafo '**INSERIMENTO-INDICE**', in caso contrario, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-INDICE**'

Il programma riceve dall'esterno il valore per la variabile '**INDICE**', quindi controlla che questo sia diverso da zero; in caso contrario, si ha una chiamata dello stesso paragrafo, in modo ricorsivo.

A causa della caratteristica ricorsiva del paragrafo '**INSERIMENTO-INDICE**', nel programma originale era riportato in un commento: «attenzione! può essere nocivo».

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0800.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-14.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 9 TIMES.
001300 01  INDICE        PIC 9.
001400 01  EOJ           PIC 9   VALUE ZERO.
001500 01  RISPOSTA      PIC XX.
001600*
001700 PROCEDURE DIVISION.
001800*------------------------ LIVELLO 0 -----------------
001900 MAIN.
002000     PERFORM INSERIMENTO-INDICE.
002100     PERFORM LAVORO UNTIL EOJ = 1.
002200     DISPLAY RECORD-ELEMENTI.
002300*
002400     STOP RUN.
002500*------------------------ LIVELLO 1 -----------------
002600 LAVORO.
002700     DISPLAY "INSERISCI I DATI DI UN ELEMENTO",
002800             " (UN SOLO CARATTERE)".
002900     ACCEPT ELEMENTO(INDICE).
003000*
003100     DISPLAY "VUOI CONTINUARE? SI O NO".
003200     ACCEPT RISPOSTA.
003300*
003400     IF RISPOSTA = "SI"
003500       THEN
003600           PERFORM INSERIMENTO-INDICE;
003700       ELSE
003800           MOVE 1 TO EOJ.
003900*------------------------ LIVELLO 2 -----------------
004000 INSERIMENTO-INDICE.
```



```
004100     DISPLAY "INSERISCI L'INDICE".
004200     ACCEPT INDICE.
004300     IF INDICE = 0
004400       THEN
004500           PERFORM INSERIMENTO-INDICE.
004600*
```

### 73.2.9 ELM0900: ricerca sequenziale all'interno di un vettore «

Variabili

'**RECORD-ELEMENTI**' è una variabile usata per accogliere una stringa;

'**ELEMENTO**' è un array che scompone '**RECORD-ELEMENTI**' in caratteri singoli;

'**POSIZIONE**' è l'indice usato per scandire gli elementi della stringa;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo;

'**LETTERA**' è la variabile che contiene la lettera da cercare nella stringa.

Descrizione

Il programma riceve dall'esterno il contenuto di una stringa e di una lettera che dovrebbe essere contenuta nella stringa stessa; successivamente il programma scandisce la stringa come vettore di caratteri e individua la prima posizione in cui appare la lettera cercata.

Paragrafo '**MAIN**'

Viene eseguito paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma esegue il paragrafo '**RICERCA**'.

A questo punto la variabile '**POSIZIONE**' contiene la posizione della lettera contenuta nella variabile '**LETTERA**' e viene visualizzata.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire, viene eseguito il paragrafo '**INSERIMENTO-DATI**', in caso contrario, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-DATI**'

Il programma riceve dall'esterno una stringa da inserire nella variabile '**RECORD-ELEMENTI**' e la lettera da ricercare nella stringa.

Paragrafo '**RICERCA**'

Viene eseguito un paragrafo che non esegue alcunché (l'istruzione '**EXIT**') scandendo l'indice '**POSIZIONE**' a partire da uno, con passo unitario, terminando quando il contenuto di '**ELEMENTO(POSIZIONE)**' coincide con il valore di '**LETTERA**', ovvero quando la posizione della lettera nella stringa è stata trovata.

In pratica, il paragrafo '**EXIT-PARAGRAPH**' è una scusa per utilizzare la scansione dell'istruzione '**PERFORM VARYING**'.

Paragrafo '**EXIT-PARAGRAPH**'

Il paragrafo non fa alcunché.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM0900.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-15.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
```

```
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 60 TIMES.
001300 01  POSIZIONE     PIC 99.
001500 01  EOJ           PIC 9   VALUE ZERO.
001600 01  RISPOSTA      PIC XX.
001700 01  LETTERA       PIC X.
001800*
001900 PROCEDURE DIVISION.
002000*------------------------ LIVELLO 0 -----------------
002100 MAIN.
002200     PERFORM INSERIMENTO-DATI.
002300     PERFORM LAVORO UNTIL EOJ = 1.
002400*
002500     STOP RUN.
002600*------------------------ LIVELLO 1 -----------------
002700 LAVORO.
002800     PERFORM RICERCA.
002900     DISPLAY "LA LETTERA ", LETTERA,
003000             " E' NELLA POSIZIONE ", POSIZIONE.
003100*
003200     DISPLAY "VUOI CONTINUARE? SI O NO".
003300     ACCEPT RISPOSTA.
003400*
003500     IF RISPOSTA = "SI"
003600       THEN
003700           PERFORM INSERIMENTO-DATI;
003800       ELSE
003900           MOVE 1 TO EOJ.
004000*------------------------ LIVELLO 2 -----------------
004100 INSERIMENTO-DATI.
004200     DISPLAY "INSERISCI LA FRASE".
004300     ACCEPT RECORD-ELEMENTI.
004400*
004500     DISPLAY "INSERISCI LA LETTERA DA TROVARE".
004600     ACCEPT LETTERA.
004700*-------------------------------------------------------
004800 RICERCA.
004900     PERFORM EXIT-PARAGRAPH
005000             VARYING POSIZIONE FROM 1 BY 1
005100             UNTIL ELEMENTO(POSIZIONE) = LETTERA.
005200*------------------------ LIVELLO 3 -----------------
005300 EXIT-PARAGRAPH.
005400     EXIT.
005500*
```

### 73.2.10 ELM1000: ricerca sequenziale all'interno di un vettore

«

Variabili

'**RECORD-ELEMENTI**' è una variabile usata per accogliere una stringa;

'**ELEMENTO**' è un array che scompone '**RECORD-ELEMENTI**' in caratteri singoli;

'**POSIZIONE**' è l'indice usato per scandire gli elementi della stringa;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo;

'**LETTERA**' è la variabile che contiene la lettera da cercare nella stringa.

Descrizione

Il programma riceve dall'esterno il contenuto di una stringa e di una lettera che dovrebbe essere contenuta nella stringa stessa; successivamente il programma scandisce la stringa come vettore di caratteri e individua la prima posizione in cui appare la lettera cercata.

Rispetto a '**ELM0900**' la scansione della stringa si arresta anche se non viene trovata alcuna corrispondenza.

Paragrafo '**MAIN**'

Viene eseguito paragrafo '**INSERIMENTO-DATI**'.



Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma esegue il paragrafo '**RICERCA**'.

A questo punto la variabile '**POSIZIONE**' contiene la posizione della lettera contenuta nella variabile '**LETTERA**' e viene visualizzata.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire, viene eseguito il paragrafo '**INSERIMENTO-DATI**', in caso contrario, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-DATI**'

Il programma riceve dall'esterno una stringa da inserire nella variabile '**RECORD-ELEMENTI**' e la lettera da ricercare nella stringa.

Paragrafo '**RICERCA**'

Viene eseguito un paragrafo che non esegue alcunché (l'istruzione '**EXIT**') scandendo l'indice '**POSIZIONE**' a partire da uno, con passo unitario, terminando quando si supera la dimensione della stringa oppure quando il contenuto di '**ELEMENTO(POSIZIONE)**' coincide con il valore di '**LETTERA**', ovvero quando la posizione della lettera nella stringa è stata trovata.

In pratica, il paragrafo '**EXIT-PARAGRAPH**' è una scusa per utilizzare la scansione dell'istruzione '**PERFORM VARYING**'.

Paragrafo '**EXIT-PARAGRAPH**'

Il paragrafo non fa alcunché.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1000.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-15.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 60 TIMES.
001300 01  POSIZIONE     PIC 99.
001400 01  EOJ           PIC 9   VALUE ZERO.
001500 01  RISPOSTA      PIC XX.
001600 01  LETTERA       PIC X.
001700*
001800 PROCEDURE DIVISION.
001900*------------------------ LIVELLO 0 -----------------
002000 MAIN.
002100     PERFORM INSERIMENTO-DATI.
002200     PERFORM LAVORO UNTIL EOJ = 1.
002300*
002400     STOP RUN.
002500*------------------------ LIVELLO 1 -----------------
002600 LAVORO.
002700     PERFORM RICERCA.
002800     DISPLAY "LA LETTERA ", LETTERA,
002900             " E' NELLA POSIZIONE ", POSIZIONE.
003000*
003100     DISPLAY "VUOI CONTINUARE? SI O NO".
003200     ACCEPT RISPOSTA.
003300*
003400     IF RISPOSTA = "SI"
003500       THEN
003600           PERFORM INSERIMENTO-DATI;
003700       ELSE
003800           MOVE 1 TO EOJ.
003900*------------------------ LIVELLO 2 -----------------
004000 INSERIMENTO-DATI.
004100     DISPLAY "INSERISCI LA FRASE".
004200     ACCEPT RECORD-ELEMENTI.
004300*
004400     DISPLAY "INSERISCI LA LETTERA DA TROVARE".
004500     ACCEPT LETTERA.
```

```
004600*-------------------------------------------------
004700 RICERCA.
004800     PERFORM EXIT-PARAGRAPH
004900             VARYING POSIZIONE FROM 1 BY 1
005000             UNTIL POSIZIONE > 60
005100             OR    ELEMENTO(POSIZIONE) = LETTERA.
005200*------------------------ LIVELLO 3 -----------------
005300 EXIT-PARAGRAPH.
005400     EXIT.
005500*
```

### 73.2.11 ELM1100: ricerca sequenziale all'interno di un vettore

Variabili

'**RECORD-ELEMENTI**' è una variabile usata per accogliere una stringa;

'**ELEMENTO**' è un array che scompone '**RECORD-ELEMENTI**' in caratteri singoli;

'**POSIZIONE**' è l'indice usato per scandire gli elementi della stringa;

'**EOJ**' quando assume il valore 1 il programma si arresta;

'**RISPOSTA**' è la variabile che riceve la risposta, un '**SI**' o un '**NO**', per la continuazione o meno con un altro calcolo;

'**LETTERA**' è la variabile che contiene la lettera da cercare nella stringa.

Descrizione

Il programma riceve dall'esterno il contenuto di una stringa e di una lettera che dovrebbe essere contenuta nella stringa stessa; successivamente il programma scandisce la stringa come vettore di caratteri e individua la prima posizione in cui appare la lettera cercata.

Rispetto a '**ELM1000**' si ottiene un avvertimento quando si indica una lettera che non è contenuta nella frase.

Paragrafo '**MAIN**'

Viene eseguito paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, terminando il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Il programma esegue il paragrafo '**RICERCA**'.

A questo punto la variabile '**POSIZIONE**' contiene la posizione della lettera contenuta nella variabile '**LETTERA**': se il valore della posizione supera la dimensione massima dell'array, si ottiene un avvertimento dell'impossibilità di trovare la corrispondenza, altrimenti viene visualizzata la posizione trovata.

Il programma riceve dall'esterno l'assenso o il dissenso riguardo alla continuazione dell'esecuzione; se l'intenzione è di proseguire, viene eseguito il paragrafo '**INSERIMENTO-DATI**', in caso contrario, viene messo il valore uno nella variabile '**EOJ**'.

Paragrafo '**INSERIMENTO-DATI**'

Il programma riceve dall'esterno una stringa da inserire nella variabile '**RECORD-ELEMENTI**' e la lettera da ricercare nella stringa.

Paragrafo '**RICERCA**'

Viene eseguito un paragrafo che non esegue alcunché (l'istruzione '**EXIT**') scandendo l'indice '**POSIZIONE**' a partire da uno, con passo unitario, terminando quando si supera la dimensione della stringa oppure quando il contenuto di '**ELEMENTO(POSIZIONE)**' coincide con il valore di '**LETTERA**', ovvero quando la posizione della lettera nella stringa è stata trovata.

In pratica, il paragrafo '**EXIT-PARAGRAPH**' è una scusa per utilizzare la scansione dell'istruzione '**PERFORM VARYING**'.

Paragrafo '**EXIT-PARAGRAPH**'

Il paragrafo non fa alcunché.



```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1100.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-15.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 DATA DIVISION.
000900*
001000 WORKING-STORAGE SECTION.
001100 01  RECORD-ELEMENTI.
001200     02  ELEMENTO  PIC X   OCCURS 60 TIMES.
001300 01  POSIZIONE     PIC 99.
001400 01  EOJ           PIC 9   VALUE ZERO.
001500 01  RISPOSTA      PIC XX.
001600 01  LETTERA       PIC X.
001700*
001800 PROCEDURE DIVISION.
001900*------------------------ LIVELLO 0 -----------------
002000 MAIN.
002100     PERFORM INSERIMENTO-DATI.
002200     PERFORM LAVORO UNTIL EOJ = 1.
002300*
002400     STOP RUN.
002500*------------------------ LIVELLO 1 -----------------
002600 LAVORO.
002700     PERFORM RICERCA.
002800*
002900     IF POSIZIONE < 61
003000       THEN
003100           DISPLAY "LA LETTERA ", LETTERA,
003200                   " E' NELLA POSIZIONE ", POSIZIONE;
003300       ELSE
003400           DISPLAY "LA LETTERA ", LETTERA,
003500                   " NON E' CONTENUTA NELLA FRASE".
003600*
003700     DISPLAY "VUOI CONTINUARE? SI O NO".
003800     ACCEPT RISPOSTA.
003900*
004000     IF RISPOSTA = "SI"
004100       THEN
004200           PERFORM INSERIMENTO-DATI;
004300       ELSE
004400           MOVE 1 TO EOJ.
004500*------------------------ LIVELLO 2 -----------------
004600 INSERIMENTO-DATI.
004700     DISPLAY "INSERISCI LA FRASE".
004800     ACCEPT RECORD-ELEMENTI.
004900*
005000     DISPLAY "INSERISCI LA LETTERA DA TROVARE".
005100     ACCEPT LETTERA.
005200*-------------------------------------------------
005300 RICERCA.
005400     PERFORM EXIT-PARAGRAPH
005500             VARYING POSIZIONE FROM 1 BY 1
005600             UNTIL POSIZIONE > 60
005700             OR    ELEMENTO(POSIZIONE) = LETTERA.
005800*------------------------ LIVELLO 3 -----------------
005900 EXIT-PARAGRAPH.
006000     EXIT.
006100*
```

### 73.2.12 ELM1300: creazione di un file sequenziale

File

'**FILE-DA-SCRIVERE**' rappresenta il file che viene creato dal programma (il nome del file è '`output.seq`'). Il file è di tipo sequenziale, dove la riga ha una dimensione fissa; non si prevede l'inserimento di un codice di interruzione di riga alla fine delle righe.

Variabili

'**RECORD-DA-SCRIVERE**' è la riga del file da creare;

'**EOJ**' quando assume il valore 1 il programma si arresta.

Descrizione

Il programma riceve dall'esterno il contenuto di ogni riga e di volta in volta lo registra nel file. Il programma termina il lavoro

quando la stringa inserita contiene solo asterischi (almeno 30, pari alla larghezza massima prevista di ogni riga).

Paragrafo '**MAIN**'

Viene aperto in scrittura il file da creare.

Viene eseguito il paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, concludendo il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Viene chiuso il file da creare.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Si controlla se la stringa inserita contiene soltanto asterischi; se è così viene messo il valore uno nella variabile '**EOJ**', altrimenti viene scritta la riga inserita nel file da scrivere e subito dopo viene eseguito nuovamente il paragrafo '**INSERIMENTO-DATI**'.

Paragrafo '**INSERIMENTO-DATI**'

Il paragrafo riceve dall'esterno il contenuto di una riga da registrare nel file, tenendo conto che vengono prese in considerazione al massimo i primi 30 caratteri, pari alla dimensione della variabile che deve accogliere i dati.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1300.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE ASSIGN TO "output.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-SCRIVERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-SCRIVERE PIC X(30).
002300*
002400 WORKING-STORAGE SECTION.
002500 01  EOJ                PIC 9   VALUE ZERO.
002600*
002700 PROCEDURE DIVISION.
002800*------------------------ LIVELLO 0 -----------------
002900 MAIN.
003000     OPEN OUTPUT FILE-DA-SCRIVERE.
003100     PERFORM INSERIMENTO-DATI.
003200     PERFORM LAVORO UNTIL EOJ = 1.
003300     CLOSE FILE-DA-SCRIVERE.
003400*
003500     STOP RUN.
003600*------------------------ LIVELLO 1 -----------------
003700 LAVORO.
003800     IF RECORD-DA-SCRIVERE = ALL "*"
003900       THEN
004000           MOVE 1 TO EOJ;
004100       ELSE
004200           WRITE RECORD-DA-SCRIVERE,
004300           PERFORM INSERIMENTO-DATI.
004400*------------------------ LIVELLO 2 -----------------
004500 INSERIMENTO-DATI.
004600     DISPLAY "INSERISCI IL RECORD".
004700     DISPLAY "PER FINIRE INSERISCI TUTTI ASTERISCHI".
004800     ACCEPT RECORD-DA-SCRIVERE.
004900*
```

Per fare in modo che le righe del file siano concluse come avviene di solito nei file di testo, con un codice di interruzione di riga, occorre specificare nell'istruzione '**SELECT**' un accesso di tipo '**LINE SEQUENTIAL**'.



### 73.2.13 ELM1400: estensione di un file sequenziale

File

'**FILE-DA-SCRIVERE**' rappresenta il file che viene esteso dal programma (il nome del file è '`output.seq`'). Il file è di tipo sequenziale, dove la riga ha una dimensione fissa; non si prevede l'inserimento di un codice di interruzione di riga alla fine delle righe.

Variabili

'**RECORD-DA-SCRIVERE**' è la riga del file da creare;

'**EOJ**' quando assume il valore 1 il programma si arresta.

Descrizione

Il programma riceve dall'esterno il contenuto di ogni riga e di volta in volta lo registra nel file. Il programma termina il lavoro quando la stringa inserita contiene solo asterischi (almeno 30, pari alla larghezza massima prevista di ogni riga).

Paragrafo '**MAIN**'

Viene aperto in scrittura in aggiunta il file da creare.

Viene eseguito il paragrafo '**INSERIMENTO-DATI**'.

Viene eseguito il paragrafo '**LAVORO**' ripetutamente, concludendo il ciclo quando la variabile '**EOJ**' contiene il valore uno.

Viene chiuso il file da creare.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LAVORO**'

Si controlla se la stringa inserita contiene soltanto asterischi; se è così viene messo il valore uno nella variabile '**EOJ**', altrimenti viene scritta la riga inserita nel file da scrivere e subito dopo viene eseguito nuovamente il paragrafo '**INSERIMENTO-DATI**'.

Paragrafo '**INSERIMENTO-DATI**'

Il paragrafo riceve dall'esterno il contenuto di una riga da registrare nel file, tenendo conto che vengono prese in considerazione al massimo i primi 30 caratteri, pari alla dimensione della variabile che deve accogliere i dati.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1400.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE ASSIGN TO "output.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-SCRIVERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-SCRIVERE PIC X(30).
002300*
002400 WORKING-STORAGE SECTION.
002500 01  EOJ                PIC 9   VALUE ZERO.
002600*
002700 PROCEDURE DIVISION.
002800*------------------------ LIVELLO 0 -----------------
002900 MAIN.
003000     OPEN EXTEND FILE-DA-SCRIVERE.
003100     PERFORM INSERIMENTO-DATI.
003200     PERFORM LAVORO UNTIL EOJ = 1.
003300     CLOSE FILE-DA-SCRIVERE.
003400*
003500     STOP RUN.
003600*------------------------ LIVELLO 1 -----------------
003700 LAVORO.
```

```
003800     IF RECORD-DA-SCRIVERE = ALL "*"
003900       THEN
004000           MOVE 1 TO EOJ;
004100       ELSE
004200           WRITE RECORD-DA-SCRIVERE,
004300           PERFORM INSERIMENTO-DATI.
004400*------------------------- LIVELLO 2 -----------------
004500 INSERIMENTO-DATI.
004600     DISPLAY "INSERISCI LA RIGA".
004700     DISPLAY "PER FINIRE INSERISCI TUTTI ASTERISCHI".
004800     ACCEPT RECORD-DA-SCRIVERE.
004900*
```

Per fare in modo che le righe del file siano concluse come avviene di solito nei file di testo, con un codice di interruzione di riga, occorre specificare nell'istruzione '**SELECT**' un accesso di tipo '**LINE SEQUENTIAL**'.

### 73.2.14 ELM1500: lettura di un file sequenziale

File

'**FILE-DA-LEGGERE**' rappresenta il file che viene letto dal programma (il nome del file è '`input.seq`'). Il file è di tipo sequenziale, dove ogni riga ha una dimensione fissa e non si fa affidamento sulla presenza di un codice di interruzione di riga.

Variabili

'**RECORD-DA-LEGGERE**' è la riga del file da leggere;

'**EOF**' quando assume il valore 1 indica che la lettura ha superato la fine del file.

Descrizione

Il programma visualizza il contenuto di un file.

La lettura avviene a blocchi di 30 caratteri, indipendentemente dal fatto che siano presenti dei codici di interruzione di riga. Diversamente, per fare in modo che la lettura sia al massimo di 30 caratteri, ma rispettando anche i codici di interruzione di riga, occorre specificare nell'istruzione '**SELECT**' un accesso di tipo '**LINE SEQUENTIAL**'.

Paragrafo '**MAIN**'

Viene aperto in lettura il file da leggere.

Viene eseguita la lettura di un primo blocco, pari alla dimensione della variabile '**RECORD-DA-LEGGERE**'; se si verifica la condizione '**AT END**', ovvero se il file è vuoto, viene messo il valore uno nella variabile '**EOF**'.

Viene eseguito il paragrafo '**LETTURA**', ripetutamente, utilizzando come condizione di arresto il fatto che la variabile '**EOF**' contenga il valore uno.

Viene chiuso il file da leggere.

Il programma si arresta perché incontra l'istruzione '**STOP RUN**'.

Paragrafo '**LETTURA**'

Viene visualizzata la porzione di file appena letta.

Viene eseguita la lettura del file da leggere; se si verifica la condizione '**AT END**', ovvero se la lettura non ha acquisito alcunché, viene messo il valore uno nella variabile '**EOF**'.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   ELM1500.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1985-02-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "input.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
```



```
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-LEGGERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-LEGGERE PIC X(30).
002300*
002400 WORKING-STORAGE SECTION.
002500 01  EOF               PIC 9   VALUE ZERO.
002600*
002700 PROCEDURE DIVISION.
002800*------------------------- LIVELLO 0 -----------------
002900 MAIN.
003000     OPEN INPUT FILE-DA-LEGGERE.
003100     READ FILE-DA-LEGGERE
003200          AT END
003300                MOVE 1 TO EOF.
003400     PERFORM LETTURA UNTIL EOF = 1.
003500     CLOSE FILE-DA-LEGGERE.
003600*
003700     STOP RUN.
003800*------------------------- LIVELLO 1 -----------------
003900 LETTURA.
004000     DISPLAY RECORD-DA-LEGGERE.
004100     READ FILE-DA-LEGGERE
004200          AT END
004300                MOVE 1 TO EOF.
004400*
```

Figura 73.23. Foto ricordo della festa conclusiva di un corso sul linguaggio COBOL realizzato con l'elaboratore Burroughs B91, presumibilmente tra il 1982 e il 1983. Nell'immagine, l'ingegnere che ha tenuto il corso compila un diploma preparato per scherzo dagli studenti che lo hanno frequentato.

## 73.3 Esempi elementari con i file

Qui si raccolgono alcuni esempi elementari di programmi COBOL per l'accesso ai file, risalenti a un lavoro didattico del 1983. Salvo dove indicato in maniera differente, gli esempi mostrati funzionano regolarmente se compilati con OpenCOBOL 0.31.

### 73.3.1 AGO-83-1: estensione di un file sequenziale

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-1.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE ASSIGN TO "file.seq"
001300            ORGANIZATION IS SEQUENTIAL.
```

```
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-SCRIVERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-SCRIVERE.
002300     02  CODICE-FILE      PIC 9(10) COMP.
002400     02  TESTO            PIC X(75).
002500*
002600 WORKING-STORAGE SECTION.
002700*
002800 01  CAMPI-SCALARI.
002900     02  EOJ              PIC 9     COMP VALUE IS 0.
003000*
003100 PROCEDURE DIVISION.
003200*------------------------ LIVELLO 0 -----------------
003300 MAIN.
003400     OPEN EXTEND FILE-DA-SCRIVERE.
003500     PERFORM INSERIMENTO-DATI UNTIL EOJ = 1.
003600     CLOSE FILE-DA-SCRIVERE.
003700     STOP RUN.
003800*------------------------ LIVELLO 1 -----------------
003900 INSERIMENTO-DATI.
004000     DISPLAY "INSERISCI PRIMA IL CODICE NUMERICO, ",
004050             "POI IL TESTO"
004100     ACCEPT CODICE-FILE.
004200     IF CODICE-FILE = 0
004300       THEN
004400           MOVE 1 TO EOJ,
004500       ELSE
004600           ACCEPT TESTO,
004700           WRITE RECORD-DA-SCRIVERE.
004800*
```

### 73.3.2 AGO-83-2: lettura sequenziale e ricerca di una chiave

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-2.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "file.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-LEGGERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-LEGGERE.
002300     02  CODICE-FILE      PIC 9(10) COMP.
002400     02  TESTO            PIC X(75).
002500*
002600 WORKING-STORAGE SECTION.
002700*
002800 01  CAMPI-SCALARI.
002900     02  EOF              PIC 9     COMP VALUE IS 0.
003000     02  EOJ              PIC 9     COMP VALUE IS 0.
003100     02  CODICE-RECORD    PIC 9(10) COMP VALUE IS 0.
003200*
003300 PROCEDURE DIVISION.
003400*------------------------ LIVELLO 0 -----------------
003500 MAIN.
003600     OPEN INPUT FILE-DA-LEGGERE.
003700     READ FILE-DA-LEGGERE
003800          AT END MOVE 1 TO EOF.
003900     PERFORM DOMANDA UNTIL EOF = 1 OR EOJ = 1.
004000     CLOSE FILE-DA-LEGGERE.
004100     STOP RUN.
```



```
004200*------------------------ LIVELLO 1 -----------------
004300 DOMANDA.
004400     DISPLAY "INSERISCI IL CODICE DEL RECORD, ",
004450             "DI 10 CIFRE"
004500     ACCEPT CODICE-RECORD.
004600     IF CODICE-RECORD = 0
004700       THEN
004800           MOVE 1 TO EOJ.
004900     PERFORM RICERCA UNTIL EOF = 1 OR EOJ = 1.
005000     CLOSE FILE-DA-LEGGERE.
005100     MOVE ZERO TO EOF.
005200     OPEN INPUT FILE-DA-LEGGERE.
005300     READ FILE-DA-LEGGERE
005400          AT END MOVE 1 TO EOF.
005500*------------------------ LIVELLO 2 -----------------
005600 RICERCA.
005700     IF CODICE-FILE = CODICE-RECORD
005800       THEN
005900           DISPLAY CODICE-FILE, " ", TESTO.
006000     READ FILE-DA-LEGGERE
006100          AT END MOVE 1 TO EOF.
006200*
```

### 73.3.3 AGO-83-3: estensione di un file relativo

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-3.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE ASSIGN TO "file.rel"
001300                             ORGANIZATION IS RELATIVE
001400                             ACCESS MODE IS SEQUENTIAL.
001500*
001600 DATA DIVISION.
001700*
001800 FILE SECTION.
001900*
002000 FD  FILE-DA-SCRIVERE
002100     LABEL RECORD IS STANDARD.
002200*
002300 01  RECORD-DA-SCRIVERE.
002400     02  TESTO            PIC X(80).
002500*
002600 WORKING-STORAGE SECTION.
002700*
002800 01  CAMPI-SCALARI.
002900     02  EOJ              PIC 9     COMP VALUE IS 0.
003000*
003100 PROCEDURE DIVISION.
003200*------------------------ LIVELLO 0 -----------------
003300 MAIN.
003400     OPEN EXTEND FILE-DA-SCRIVERE.
003500     PERFORM INSERIMENTO-DATI UNTIL EOJ = 1.
003600     CLOSE FILE-DA-SCRIVERE.
003700     STOP RUN.
003800*------------------------ LIVELLO 1 -----------------
003900 INSERIMENTO-DATI.
004000     DISPLAY "INSERISCI IL TESTO DEL RECORD"
004100     ACCEPT TESTO.
004200     IF TESTO = SPACES
004300       THEN
004400           MOVE 1 TO EOJ,
004500       ELSE
004600           WRITE RECORD-DA-SCRIVERE.
004700*
```

### 73.3.4 AGO-83-4: lettura di un file relativo ad accesso diretto

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-4.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
```

```
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "file.rel"
001300                            ORGANIZATION IS RELATIVE
001400                            ACCESS MODE IS RANDOM
001500                            RELATIVE KEY IS N-RECORD.
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-DA-LEGGERE
002200     LABEL RECORD IS STANDARD.
002300*
002400 01  RECORD-DA-LEGGERE.
002500     02  TESTO            PIC X(80).
002600*
002700 WORKING-STORAGE SECTION.
002800*
002900 01  CAMPI-SCALARI.
003000     02  INVALID-KEY      PIC 9     COMP VALUE IS 0.
003100     02  EOJ              PIC 9     COMP VALUE IS 0.
003200     02  N-RECORD         PIC 9(10) COMP VALUE IS 0.
003300*
003400 PROCEDURE DIVISION.
003500*------------------------ LIVELLO 0 -----------------
003600 MAIN.
003700     OPEN INPUT FILE-DA-LEGGERE.
003800     PERFORM ELABORA UNTIL EOJ = 1.
003900     CLOSE FILE-DA-LEGGERE.
004000     STOP RUN.
004100*------------------------ LIVELLO 1 -----------------
004200 ELABORA.
004300     DISPLAY "INSERISCI IL NUMERO DEL RECORD"
004400     ACCEPT N-RECORD.
004500     IF N-RECORD = 0
004600       THEN
004700           MOVE 1 TO EOJ;
004800       ELSE
004900           PERFORM LEGGI,
005000           IF INVALID-KEY = 1
005100             THEN
005200                 DISPLAY "INVALID KEY";
005300             ELSE
005400                 PERFORM VISUALIZZA.
005500*------------------------ LIVELLO 2 -----------------
005600 VISUALIZZA.
005700     DISPLAY N-RECORD, " ", TESTO.
005800*-------------------------------------------------------
005900 LEGGI.
006000     MOVE ZERO TO INVALID-KEY.
006100     READ FILE-DA-LEGGERE
006200          INVALID KEY
006300                    MOVE 1 TO INVALID-KEY.
006400*
```

### 73.3.5 AGO-83-5: creazione di un file a indice

Questo esempio funziona con il compilatore TinyCOBOL 0.61. In questo caso, vengono creati due file: ‘`file.ind`’ e ‘`file.ind1`’, che insieme costituiscono lo stesso file logico.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-5.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-20.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE
001250            ASSIGN TO "file.ind"
001300            ORGANIZATION IS INDEXED
001400            ACCESS MODE IS SEQUENTIAL
```



```
001500            RECORD KEY IS CHIAVE
001600            ALTERNATE RECORD KEY IS CHIAVE2
001700                              WITH DUPLICATES.
001800*
001900 DATA DIVISION.
002000*
002100 FILE SECTION.
002200*
002300 FD  FILE-DA-SCRIVERE
002400     LABEL RECORD IS STANDARD.
002500*
002600 01  RECORD-DA-SCRIVERE.
002700     02  CHIAVE           PIC X(5).
002800     02  CHIAVE2          PIC X(5).
002900     02  TESTO            PIC X(70).
003000*
003100 WORKING-STORAGE SECTION.
003200*
003300 01  CAMPI-SCALARI.
003400     02  EOJ              PIC 9     COMP VALUE IS 0.
003500*
003600 PROCEDURE DIVISION.
003700*------------------------ LIVELLO 0 -----------------
003800 MAIN.
003900     OPEN OUTPUT FILE-DA-SCRIVERE.
004000     PERFORM INSERIMENTO-DATI UNTIL EOJ = 1.
004100     CLOSE FILE-DA-SCRIVERE.
004200     STOP RUN.
004300*------------------------ LIVELLO 1 -----------------
004400 INSERIMENTO-DATI.
004500     DISPLAY "INSERISCI IL RECORD: ",
004550             "I PRIMI CINQUE CARATTERI ",
004600             "COSTITUISCONO LA CHIAVE PRIMARIA ",
004700             "CHE DEVE ESSERE UNICA"
004800     ACCEPT RECORD-DA-SCRIVERE.
004900     IF RECORD-DA-SCRIVERE = SPACES
005000       THEN
005100           MOVE 1 TO EOJ,
005200       ELSE
005300           WRITE RECORD-DA-SCRIVERE
005400                 INVALID KEY
005500                             DISPLAY "LA CHIAVE ",
005550                                     CHIAVE,
005600                                     " E' DOPPIA,",
005650                                     " OPPURE ",
005700                                     "NON E' VALIDA".
005800*
```

### 73.3.6 AGO-83-6: lettura di un file a indice ad accesso diretto

Questo esempio funziona con il compilatore TinyCOBOL 0.61 e utilizza il file creato con l’esempio precedente.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-6.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE
001250            ASSIGN TO "file.ind"
001300            ORGANIZATION IS INDEXED
001400            ACCESS MODE IS RANDOM
001500            RECORD KEY IS CHIAVE
001600            ALTERNATE RECORD KEY IS CHIAVE2
001700                      WITH DUPLICATES.
001800*
001900 DATA DIVISION.
002000*
002100 FILE SECTION.
002200*
002300 FD  FILE-DA-LEGGERE
002400     LABEL RECORD IS STANDARD.
002500*
002600 01  RECORD-DA-LEGGERE.
```

```
002700     02  CHIAVE           PIC X(5).
002800     02  CHIAVE2          PIC X(5).
002900     02  TESTO            PIC X(70).
003000*
003100 WORKING-STORAGE SECTION.
003200*
003300 01  CAMPI-SCALARI.
003400     02  EOJ              PIC 9     COMP VALUE IS 0.
003500     02  INV-KEY          PIC 9     COMP VALUE IS 0.
003600*
003700 PROCEDURE DIVISION.
003800*------------------------ LIVELLO 0 -----------------
003900 MAIN.
004000     OPEN INPUT FILE-DA-LEGGERE.
004100     PERFORM ELABORAZIONE UNTIL EOJ = 1.
004200     CLOSE FILE-DA-LEGGERE.
004300     STOP RUN.
004400*------------------------ LIVELLO 1 -----------------
004500 ELABORAZIONE.
004600     DISPLAY "INSERISCI LA CHIAVE PRIMARIA".
004700     ACCEPT CHIAVE.
004800     IF CHIAVE = SPACES
004900       THEN
005000           MOVE 1 TO EOJ,
005100       ELSE
005200           PERFORM LEGGI,
005300           IF INV-KEY = 1
005400             THEN
005500                 DISPLAY "INVALID KEY: ", CHIAVE,
005600             ELSE
005700                 DISPLAY CHIAVE, " ", CHIAVE2, " ",
005750                         TESTO.
005800*------------------------ LIVELLO 2 -----------------
005900 LEGGI.
006000     MOVE 0 TO INV-KEY.
006100     READ FILE-DA-LEGGERE
006200          INVALID KEY
006300                     MOVE 1 TO INV-KEY.
006400*
```

### 73.3.7 AGO-83-8: lettura di un file a indice ad accesso dinamico

«

Questo esempio funziona parzialmente con il compilatore TinyCOBOL 0.61 e utilizza il file già predisposto per quello precedente. Si osservi che si fa riferimento alla chiave secondaria del file, in modo da poter contare sulla presenza di chiavi doppie.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-8.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "file.ind"
001300                            ORGANIZATION IS INDEXED
001400                            ACCESS MODE IS DYNAMIC
001500                            RECORD KEY IS CHIAVE2.
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-DA-LEGGERE
002200     LABEL RECORD IS STANDARD.
002300*
002400 01  RECORD-DA-LEGGERE.
002500     02  CHIAVE           PIC X(5).
002600     02  CHIAVE2          PIC X(5).
002700     02  TESTO            PIC X(70).
002800*
002900 WORKING-STORAGE SECTION.
003000*
003100 01  CAMPI-SCALARI.
003200     02  EOJ              PIC 9     COMP VALUE IS 0.
003300     02  EOF              PIC 9     COMP VALUE IS 0.
```



```
003400     02  INV-KEY          PIC 9     COMP VALUE IS 0.
003500     02  END-KEY          PIC 9     COMP VALUE IS 0.
003600     02  CHIAVE-W         PIC X(5).
003700*
003800 PROCEDURE DIVISION.
003900*------------------------ LIVELLO 0 -----------------
004000 MAIN.
004100     OPEN INPUT FILE-DA-LEGGERE.
004200     PERFORM ELABORAZIONE UNTIL EOJ = 1.
004300     CLOSE FILE-DA-LEGGERE.
004400     STOP RUN.
004500*------------------------ LIVELLO 1 -----------------
004600 ELABORAZIONE.
004700     DISPLAY "INSERISCI LA CHIAVE SECONDARIA".
004800     ACCEPT CHIAVE2.
004900     IF CHIAVE2 = SPACES
005000       THEN
005100           MOVE 1 TO EOJ,
005200       ELSE
005300           MOVE CHIAVE2 TO CHIAVE-W,
005400           PERFORM LEGGI,
005500           IF INV-KEY = 1
005600             THEN
005700                 DISPLAY "INVALID KEY: ", CHIAVE2,
005800             ELSE
005900                 PERFORM MOSTRA-LEGGI-NEXT
006000                         UNTIL END-KEY = 1
006100                            OR EOF     = 1.
006200*------------------------ LIVELLO 2 -----------------
006300 LEGGI.
006400     MOVE ZERO TO END-KEY.
006500     MOVE ZERO TO EOF.
006600     MOVE ZERO TO INV-KEY.
006700     READ FILE-DA-LEGGERE
006800          INVALID KEY MOVE 1 TO INV-KEY.
006900*-----------------------------------------------------
007000 MOSTRA-LEGGI-NEXT.
007100     DISPLAY CHIAVE, " ", CHIAVE2, " ", TESTO.
007200     READ FILE-DA-LEGGERE NEXT RECORD
007300          AT END MOVE 1 TO EOF.
007400     IF NOT CHIAVE-W = CHIAVE2
007500       THEN
007600           MOVE 1 TO END-KEY.
007700*
```

### 73.3.8 AGO-83-10: lettura di un file a indice ad accesso dinamico

»

Questo esempio funziona con il compilatore TinyCOBOL 0.61 e utilizza il file già predisposto per quello precedente. In questo caso si ritorna a utilizzare la chiave primaria.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-10.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "file.ind"
001300                            ORGANIZATION IS INDEXED
001400                            ACCESS MODE IS DYNAMIC
001500                            RECORD KEY IS CHIAVE.
001600*
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-DA-LEGGERE
002200     LABEL RECORD IS STANDARD.
002300*
002400 01  RECORD-DA-LEGGERE.
002500     02  CHIAVE           PIC X(5).
002600     02  CHIAVE2          PIC X(5).
002700     02  TESTO            PIC X(70).
002800*
002900 WORKING-STORAGE SECTION.
```

```
003000*
003100 01  CAMPI-SCALARI.
003200     02  EOJ              PIC 9     COMP VALUE IS 0.
003300     02  EOF              PIC 9     COMP VALUE IS 0.
003400     02  INV-KEY          PIC 9     COMP VALUE IS 0.
003500     02  END-KEY          PIC 9     COMP VALUE IS 0.
003600     02  CHIAVE-INIZIALE  PIC X(5).
003700     02  CHIAVE-FINALE    PIC X(5).
003800     02  CHIAVE-SCAMBIO   PIC X(5).
003900*
004000 PROCEDURE DIVISION.
004100*------------------------ LIVELLO 0 -----------------
004200 MAIN.
004300     OPEN INPUT FILE-DA-LEGGERE.
004400     PERFORM ELABORAZIONE UNTIL EOJ = 1.
004500     CLOSE FILE-DA-LEGGERE.
004600     STOP RUN.
004700*------------------------ LIVELLO 1 -----------------
004800 ELABORAZIONE.
004900     DISPLAY "INSERISCI LA CHIAVE PRIMARIA ",
005000             "INIZIALE, POI QUELLA FINALE".
005100     ACCEPT CHIAVE-INIZIALE.
005200     ACCEPT CHIAVE-FINALE.
005300     IF CHIAVE-INIZIALE > CHIAVE-FINALE
005400       THEN
005500           MOVE CHIAVE-INIZIALE TO CHIAVE-SCAMBIO,
005600           MOVE CHIAVE-FINALE   TO CHIAVE-INIZIALE,
005700           MOVE CHIAVE-SCAMBIO  TO CHIAVE-FINALE.
005800     IF CHIAVE-INIZIALE = SPACES
005900       THEN
006000           MOVE 1 TO EOJ,
006100       ELSE
006200           MOVE CHIAVE-INIZIALE TO CHIAVE,
006300           PERFORM LEGGI,
006400           IF INV-KEY = 1
006500             THEN
006600                 DISPLAY "INVALID KEY: ", CHIAVE,
006700             ELSE
006800                 PERFORM MOSTRA-LEGGI-NEXT
006900                         UNTIL END-KEY = 1
007000                            OR EOF     = 1.
007100*------------------------ LIVELLO 2 -----------------
007200 LEGGI.
007300     MOVE ZERO TO END-KEY.
007400     MOVE ZERO TO EOF.
007500     MOVE ZERO TO INV-KEY.
007600     READ FILE-DA-LEGGERE
007700          INVALID KEY MOVE 1 TO INV-KEY.
007800*-----------------------------------------------------
007900 MOSTRA-LEGGI-NEXT.
008000     DISPLAY CHIAVE, " ", CHIAVE2, " ", TESTO.
008100     READ FILE-DA-LEGGERE NEXT RECORD
008200          AT END MOVE 1 TO EOF.
008300     IF CHIAVE > CHIAVE-FINALE
008400       THEN
008500           MOVE 1 TO END-KEY.
008600*
```

### 73.3.9 AGO-83-12: lettura di un file a indice ad accesso dinamico

Questo esempio funziona con il compilatore TinyCOBOL 0.61 e utilizza il file già predisposto per quello precedente. In questo caso si utilizza l'istruzione '**START**' per il posizionamento iniziale.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-12.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-LEGGERE ASSIGN TO "file.ind"
001300                            ORGANIZATION IS INDEXED
001400                            ACCESS MODE IS DYNAMIC
001500                            RECORD KEY IS CHIAVE.
001600*
```



```
001700 DATA DIVISION.
001800*
001900 FILE SECTION.
002000*
002100 FD  FILE-DA-LEGGERE
002200     LABEL RECORD IS STANDARD.
002300*
002400 01  RECORD-DA-LEGGERE.
002500     02  CHIAVE           PIC X(5).
002600     02  CHIAVE2          PIC X(5).
002700     02  TESTO            PIC X(70).
002800*
002900 WORKING-STORAGE SECTION.
003000*
003100 01  CAMPI-SCALARI.
003200     02  EOJ              PIC 9     COMP VALUE IS 0.
003300     02  EOF              PIC 9     COMP VALUE IS 0.
003400     02  INV-KEY          PIC 9     COMP VALUE IS 0.
003500     02  END-KEY          PIC 9     COMP VALUE IS 0.
003600     02  CHIAVE-INIZIALE  PIC X(5).
003700     02  CHIAVE-FINALE    PIC X(5).
003800     02  CHIAVE-SCAMBIO   PIC X(5).
003900*
004000 PROCEDURE DIVISION.
004100*------------------------ LIVELLO 0 -----------------
004200 MAIN.
004300     OPEN INPUT FILE-DA-LEGGERE.
004400     PERFORM ELABORAZIONE UNTIL EOJ = 1.
004500     CLOSE FILE-DA-LEGGERE.
004600     STOP RUN.
004700*------------------------ LIVELLO 1 -----------------
004800 ELABORAZIONE.
004900     DISPLAY "INSERISCI LA CHIAVE PRIMARIA ",
005000             "INIZIALE, POI QUELLA FINALE".
005100     ACCEPT CHIAVE-INIZIALE.
005200     ACCEPT CHIAVE-FINALE.
005300     IF CHIAVE-INIZIALE > CHIAVE-FINALE
005400       THEN
005500           MOVE CHIAVE-INIZIALE TO CHIAVE-SCAMBIO,
005600           MOVE CHIAVE-FINALE   TO CHIAVE-INIZIALE,
005700           MOVE CHIAVE-SCAMBIO  TO CHIAVE-FINALE.
005800     IF CHIAVE-INIZIALE = SPACES
005900       THEN
006000           MOVE 1 TO EOJ,
006100       ELSE
006200           MOVE CHIAVE-INIZIALE TO CHIAVE,
006300           PERFORM START-LEGGI,
006400           IF INV-KEY = 1
006500             THEN
006600                 DISPLAY "INVALID KEY: ", CHIAVE,
006700             ELSE
006800                 PERFORM MOSTRA-LEGGI-NEXT
006900                         UNTIL END-KEY = 1
007000                            OR EOF     = 1.
007100*------------------------ LIVELLO 2 -----------------
007200 START-LEGGI.
007300     MOVE ZERO TO END-KEY.
007400     MOVE ZERO TO EOF.
007500     MOVE ZERO TO INV-KEY.
007600     START FILE-DA-LEGGERE KEY IS NOT < CHIAVE
007700          INVALID KEY MOVE 1 TO INV-KEY.
007800     IF NOT INV-KEY = 1
007900       THEN
008000           PERFORM LEGGI.
008100*-----------------------------------------------------
008200 MOSTRA-LEGGI-NEXT.
008300     DISPLAY CHIAVE, " ", CHIAVE2, " ", TESTO.
008400     PERFORM LEGGI.
008500*------------------------ LIVELLO 3 -----------------
008600 LEGGI.
008700     READ FILE-DA-LEGGERE NEXT RECORD
008800          AT END MOVE 1 TO EOF.
008900     IF CHIAVE > CHIAVE-FINALE
009000       THEN
009100           MOVE 1 TO END-KEY.
009200*
```

### 73.3.10 AGO-83-13: creazione di un file sequenziale con dati da rielaborare

Questo esempio serve a creare un file sequenziale, contenente dei calcoli da eseguire, successivamente, con un altro programma.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-13.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-22.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-DA-SCRIVERE ASSIGN TO "calc.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-SCRIVERE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-SCRIVERE.
002300     02  NUMERO-1         PIC 9(15).
002400     02  TIPO-CALCOLO     PIC X.
002500     02  NUMERO-2         PIC 9(15).
002600     02  FILLER           PIC X.
002700     02  RISULTATO        PIC 9(15).
002800     02  FILLER           PIC X.
002900     02  RESTO            PIC 9(15).
003000     02  NOTE             PIC X(18).
003100*
003200 WORKING-STORAGE SECTION.
003300*
003400 01  CAMPI-SCALARI.
003500     02  EOJ              PIC 9     COMP VALUE IS 0.
003600*
003700 PROCEDURE DIVISION.
003800*------------------------ LIVELLO 0 -----------------
003900 MAIN.
004000     OPEN EXTEND FILE-DA-SCRIVERE.
004100     PERFORM INSERIMENTO-DATI UNTIL EOJ = 1.
004200     CLOSE FILE-DA-SCRIVERE.
004300     STOP RUN.
004400*------------------------ LIVELLO 1 -----------------
004500 INSERIMENTO-DATI.
004600     DISPLAY "INSERISCI, IN SEQUENZA, ",
004650             "IL PRIMO NUMERO, ",
004700             "IL SIMBOLO DELL'OPERAZIONE, ",
004750             "IL SECONDO NUMERO".
004800     ACCEPT NUMERO-1.
004900     ACCEPT TIPO-CALCOLO.
005000     ACCEPT NUMERO-2.
005100     IF NUMERO-1 = 0 AND NUMERO-2 = 0
005150        AND TIPO-CALCOLO = SPACE
005200       THEN
005300           MOVE 1 TO EOJ,
005400       ELSE
005500           WRITE RECORD-DA-SCRIVERE.
005600*
```

### 73.3.11 AGO-83-14: lettura e riscrittura di un file sequenziale

Questo esempio legge e riscrive il file generato con l'esempio precedente, eseguendo i calcoli previsti e mostrando anche il risultato a video.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-14.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
```



```
001100*
001200     SELECT FILE-DA-ELABORARE ASSIGN TO "calc.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-DA-ELABORARE
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-DA-ELABORARE.
002300     02  NUMERO-1         PIC 9(15).
002400     02  TIPO-CALCOLO     PIC X.
002500     02  NUMERO-2         PIC 9(15).
002600     02  UGUALE           PIC X.
002700     02  RISULTATO        PIC 9(15).
002800     02  SEPARAZIONE      PIC X.
002900     02  RESTO            PIC 9(15).
003000     02  NOTE             PIC X(18).
003100*
003200 WORKING-STORAGE SECTION.
003300*
003400 01  CAMPI-SCALARI.
003500     02  EOF              PIC 9     COMP VALUE IS 0.
003600     02  EOJ              PIC 9     COMP VALUE IS 0.
003700*
003800 PROCEDURE DIVISION.
003900*------------------------ LIVELLO 0 -----------------
004000 MAIN.
004100     OPEN I-O FILE-DA-ELABORARE.
004200     READ FILE-DA-ELABORARE
004300          AT END MOVE 1 TO EOF.
004400     PERFORM ELABORAZIONE UNTIL EOF = 1.
004500     CLOSE FILE-DA-ELABORARE.
004600     STOP RUN.
004700*------------------------ LIVELLO 1 -----------------
004800 ELABORAZIONE.
004900     MOVE SPACES TO NOTE.
005000     MOVE ZERO   TO RESTO.
005100     IF      TIPO-CALCOLO = "+"
005200     THEN
005300         COMPUTE RISULTATO = NUMERO-1 + NUMERO-2;
005400     ELSE IF TIPO-CALCOLO = "-"
005500     THEN
005600         COMPUTE RISULTATO = NUMERO-1 - NUMERO-2;
005700     ELSE IF TIPO-CALCOLO = "*"
005800     THEN
005900         COMPUTE RISULTATO = NUMERO-1 * NUMERO-2;
006000     ELSE IF TIPO-CALCOLO = "/"
006100     THEN
006200         DIVIDE NUMERO-1 BY NUMERO-2 GIVING RISULTATO,
006300                REMAINDER RESTO;
006400     ELSE
006500         MOVE ZERO TO RISULTATO,
006600         MOVE "CALCOLO ERRATO" TO NOTE.
006700
006800     MOVE "="   TO UGUALE.
006900     MOVE SPACE TO SEPARAZIONE.
007000     DISPLAY RECORD-DA-ELABORARE.
007100     REWRITE RECORD-DA-ELABORARE.
007200     READ FILE-DA-ELABORARE
007300          AT END MOVE 1 TO EOF.
007400*
```

### 73.3.12 AGO-83-15: estensione di un file sequenziale contenente aggiornamenti successivi

Questo esempio estende un file sequenziale con delle informazioni, che possono essere aggiornate in momenti successivi. I record si considerano contenere la stessa informazione, aggiornata, quando hanno la stessa chiave.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-15.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-22.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
```

```
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-AGGIORNAMENTI ASSIGN TO "agg.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500 DATA DIVISION.
001600*
001700 FILE SECTION.
001800*
001900 FD  FILE-AGGIORNAMENTI
002000     LABEL RECORD IS STANDARD.
002100*
002200 01  RECORD-AGGIORNAMENTI.
002300     02  CHIAVE           PIC X(5).
002400     02  DATI             PIC X(67).
002500     02  ANNO-MESE-GIORNO.
002600         03  ANNO         PIC 9999.
002700         03  MESE         PIC 99.
002800         03  GIORNO       PIC 99.
002900*
003000 WORKING-STORAGE SECTION.
003100*
003200 01  CAMPI-SCALARI.
003300     02  EOJ              PIC 9     COMP VALUE IS 0.
003400*
003500 PROCEDURE DIVISION.
003600*------------------------ LIVELLO 0 -----------------
003700 MAIN.
003800     OPEN EXTEND FILE-AGGIORNAMENTI.
003900     PERFORM INSERIMENTO-DATI UNTIL EOJ = 1.
004000     CLOSE FILE-AGGIORNAMENTI.
004100     STOP RUN.
004200*------------------------ LIVELLO 1 -----------------
004300 INSERIMENTO-DATI.
004400     DISPLAY "INSERISCI IN SEQUENZA: ",
004450             "LA CHIAVE, I DATI DEL ",
004500             "RECORD E LA DATA DI ",
004550             "INSERIMENTO. LA DATA SI ",
004600             "SCRIVE SECONDO IL FORMATO AAAAMMGG".
004700     ACCEPT CHIAVE.
004800     ACCEPT DATI.
004900     ACCEPT ANNO-MESE-GIORNO.
005000     IF CHIAVE = SPACES
005100       THEN
005200           MOVE 1 TO EOJ,
005300       ELSE
005400           WRITE RECORD-AGGIORNAMENTI.
005500*
```

### 73.3.13 AGO-83-16: aggiornamento di un file a indice

Questo esempio utilizza il file sequenziale del programma precedente, per aggiornare i record di un file a indice (che deve essere già esistente). Questo esempio funziona correttamente utilizzando il compilatore TinyCOBOL 0.61.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-16.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-08.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-AGGIORNAMENTI ASSIGN TO "agg.seq"
001300            ORGANIZATION IS SEQUENTIAL.
001400*
001500     SELECT FILE-DA-AGGIORNARE ASSIGN TO "agg.ind"
001600            ORGANIZATION IS INDEXED,
001700            ACCESS MODE IS RANDOM,
001800            RECORD KEY IS CHIAVE-K.
001900*
002000 DATA DIVISION.
002100*
002200 FILE SECTION.
002300*
002400 FD  FILE-AGGIORNAMENTI
```



```
002500     LABEL RECORD IS STANDARD.
002600*
002700 01  RECORD-AGGIORNAMENTI.
002800     02  CHIAVE           PIC X(5).
002900     02  DATI             PIC X(67).
003000     02  ANNO-MESE-GIORNO.
003100         03  ANNO         PIC 9999.
003200         03  MESE         PIC 99.
003300         03  GIORNO       PIC 99.
003400*
003500 FD  FILE-DA-AGGIORNARE
003600     LABEL RECORD IS STANDARD.
003700*
003800 01  RECORD-DA-AGGIORNARE.
003900     02  CHIAVE-K         PIC X(5).
004000     02  DATI             PIC X(67).
004100     02  ANNO-MESE-GIORNO.
004200         03  ANNO         PIC 9999.
004300         03  MESE         PIC 99.
004400         03  GIORNO       PIC 99.
004500*
004600 WORKING-STORAGE SECTION.
004700*
004800 01  CAMPI-SCALARI.
004900     02  EOF              PIC 9     COMP VALUE IS 0.
005000     02  INV-KEY          PIC 9     COMP VALUE IS 0.
005100*
005200 PROCEDURE DIVISION.
005300*------------------------ LIVELLO 0 -----------------
005400 MAIN.
005500     OPEN INPUT FILE-AGGIORNAMENTI.
005600     OPEN I-O   FILE-DA-AGGIORNARE.
005700     PERFORM LEGGI-FILE-AGGIORNAMENTI.
005800     PERFORM ELABORAZIONE
005900             UNTIL EOF = 1.
006000     CLOSE FILE-AGGIORNAMENTI.
006100     CLOSE FILE-DA-AGGIORNARE
006200     STOP RUN.
006300*------------------------ LIVELLO 1 -----------------
006400 ELABORAZIONE.
006500     MOVE ZERO TO INV-KEY.
006600     READ FILE-DA-AGGIORNARE
006700          INVALID KEY
006800                    MOVE 1 TO INV-KEY.
006900     IF INV-KEY = 1
007000       THEN
007100           PERFORM WRITE-FILE-DA-AGGIORNARE;
007200       ELSE
007300           IF ANNO-MESE-GIORNO
007350              OF RECORD-AGGIORNAMENTI
007400              > ANNO-MESE-GIORNO
007450                OF RECORD-DA-AGGIORNARE
007500             THEN
007600                PERFORM REWRITE-FILE-DA-AGGIORNARE.
007700     PERFORM LEGGI-FILE-AGGIORNAMENTI.
007800*-------------------------------------------------
007900 LEGGI-FILE-AGGIORNAMENTI.
008000     READ FILE-AGGIORNAMENTI
008100          AT END MOVE 1 TO EOF.
008200     IF NOT EOF = 1
008300       THEN
008400           MOVE CHIAVE TO CHIAVE-K.
008500*------------------------ LIVELLO 2 -----------------
008600 WRITE-FILE-DA-AGGIORNARE.
008700     WRITE RECORD-DA-AGGIORNARE
008750           FROM RECORD-AGGIORNAMENTI
008800           INVALID KEY
008900                     DISPLAY "ERRORE NON PREVISTO 1".
009000*-------------------------------------------------
009100 REWRITE-FILE-DA-AGGIORNARE.
009200     REWRITE RECORD-DA-AGGIORNARE
009250             FROM RECORD-AGGIORNAMENTI
009300           INVALID KEY
009400                     DISPLAY "ERRORE NON PREVISTO 2".
009500*
```

### 73.3.14 AGO-83-18: fusione tra due file sequenziali ordinati

Il programma seguente richiede la presenza di due file sequenziali, ordinati, denominati rispettivamente '`file-ord-1.seq`' e '`file-ord-2.seq`'. Per creare questi file si può usare il programma

‘**AGO-83-1**’, avendo cura di inserire una sequenza di record ordinati per codice, modificando poi il nome del file, una volta come ‘`file-ord-1.seq`’ e un’altra volta come ‘`file-ord-2.seq`’.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-18.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 1983-06.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-ORD-1 ASSIGN TO "file-ord-1.seq"
001300                       ORGANIZATION IS SEQUENTIAL.
001400     SELECT FILE-ORD-2 ASSIGN TO "file-ord-2.seq"
001500                       ORGANIZATION IS SEQUENTIAL.
001600     SELECT FILE-MERGE ASSIGN TO "file-merge.seq"
001700                       ORGANIZATION IS SEQUENTIAL.
001800*
001900 DATA DIVISION.
002000*
002100 FILE SECTION.
002200*
002300 FD  FILE-ORD-1
002400     LABEL RECORD IS STANDARD.
002500*
002600 01  RECORD-ORD-1.
002700     02  CODICE-1         PIC 9(10) COMP.
002800     02  FILLER           PIC X(75).
002900*
003000 FD  FILE-ORD-2
003100     LABEL RECORD IS STANDARD.
003200*
003300 01  RECORD-ORD-2.
003400     02  CODICE-2         PIC 9(10) COMP.
003500     02  FILLER           PIC X(75).
003600*
003700 FD  FILE-MERGE
003800     LABEL RECORD IS STANDARD.
003900*
004000 01  RECORD-MERGE         PIC X(80).
004100*
004200 WORKING-STORAGE SECTION.
004300*
004400 01  CAMPI-SCALARI.
004500     02  EOF-1            PIC 9     COMP VALUE IS 0.
004600     02  EOF-2            PIC 9     COMP VALUE IS 0.
004700*
004800 PROCEDURE DIVISION.
004900*------------------------ LIVELLO 0 -----------------
005000 MAIN.
005100     OPEN INPUT  FILE-ORD-1.
005200     OPEN INPUT  FILE-ORD-2.
005300     OPEN OUTPUT FILE-MERGE.
005400     PERFORM LETTURA-FILE-ORD-1.
005500     PERFORM LETTURA-FILE-ORD-2.
005600     PERFORM ELABORAZIONE
005700             UNTIL EOF-1 = 1 AND EOF-2 = 1.
005800     CLOSE FILE-MERGE.
005900     CLOSE FILE-ORD-2.
006000     CLOSE FILE-ORD-1.
006100     STOP RUN.
006200*------------------------ LIVELLO 1 -----------------
006300 ELABORAZIONE.
006400     IF      (CODICE-1 <= CODICE-2 AND EOF-1 = 0)
006450             OR EOF-2 = 1
006500     THEN
006600         MOVE RECORD-ORD-1 TO RECORD-MERGE,
006700         WRITE RECORD-MERGE,
006800         PERFORM LETTURA-FILE-ORD-1;
006900     ELSE IF (CODICE-1 > CODICE-2 AND EOF-2 = 0)
006950             OR EOF-1 = 1
007000     THEN
007100         MOVE RECORD-ORD-2 TO RECORD-MERGE,
007200         WRITE RECORD-MERGE,
007300         PERFORM LETTURA-FILE-ORD-2;
007400     ELSE
007500         DISPLAY "ERRORE NON PREVISTO".
007600*------------------------ LIVELLO 2 -----------------
```



```
007700 LETTURA-FILE-ORD-1.
007800     READ FILE-ORD-1
007900          AT END
008000               MOVE 1 TO EOF-1.
008100*-------------------------------------------------------
008200 LETTURA-FILE-ORD-2.
008300     READ FILE-ORD-2
008400          AT END
008500               MOVE 1 TO EOF-2.
008600*
```

### 73.3.15 AGO-83-20: riordino attraverso la fusione

Il programma seguente utilizza un file sequenziale, non ordinato, denominato ‘`file-in.seq`’, per generare il file ‘`file-out.seq`’ ordinato, utilizzando due file temporanei: ‘`file-tmp-1.seq`’ e ‘`file-tmp-2.seq`’. Per creare il file ‘`file-in.seq`’, si può usare il programma ‘**AGO-83-1**’, modificando poi il nome come richiesto in questo esempio.

Nella sezione 62.6.2 viene descritto il problema del riordino ottenuto attraverso la suddivisione in blocchi del file e la fusione successiva.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.   AGO-83-20.
000300 AUTHOR.       DANIELE GIACOMINI.
000400 DATE-WRITTEN. 2005-03-29.
000500*
000600 ENVIRONMENT DIVISION.
000700*
000800 INPUT-OUTPUT SECTION.
000900*
001000 FILE-CONTROL.
001100*
001200     SELECT FILE-IN    ASSIGN TO "file-in.seq"
001300                       ORGANIZATION IS SEQUENTIAL.
001400     SELECT FILE-TMP-1 ASSIGN TO "file-tmp-1.seq"
001500                       ORGANIZATION IS SEQUENTIAL.
001600     SELECT FILE-TMP-2 ASSIGN TO "file-tmp-2.seq"
001700                       ORGANIZATION IS SEQUENTIAL.
001800     SELECT FILE-MERGE ASSIGN TO "file-out.seq"
001900                       ORGANIZATION IS SEQUENTIAL.
002000*
002100 DATA DIVISION.
002200*
002300 FILE SECTION.
002400*
002500 FD  FILE-IN
002600     LABEL RECORD IS STANDARD.
002700*
002800 01  RECORD-IN.
002900     02  CODICE-IN        PIC 9(10) COMP.
003000     02  FILLER           PIC X(75).
003100*
003200 FD  FILE-TMP-1
003300     LABEL RECORD IS STANDARD.
003400*
003500 01  RECORD-TMP-1.
003600     02  CODICE-T1        PIC 9(10) COMP.
003700     02  FILLER           PIC X(75).
003800*
003900 FD  FILE-TMP-2
004000     LABEL RECORD IS STANDARD.
004100*
004200 01  RECORD-TMP-2.
004300     02  CODICE-T2        PIC 9(10) COMP.
004400     02  FILLER           PIC X(75).
004500*
004600 FD  FILE-MERGE
004700     LABEL RECORD IS STANDARD.
004800*
004900 01  RECORD-MERGE.
005000     02  CODICE-MERGE     PIC 9(10) COMP.
005100     02  FILLER           PIC X(75).
005200*
005300 WORKING-STORAGE SECTION.
005400*
005500 01  CAMPI-SCALARI.
005600     02  EOF              PIC 9     COMP VALUE IS 0.
005700     02  EOF-1            PIC 9     COMP VALUE IS 0.
005800     02  EOF-2            PIC 9     COMP VALUE IS 0.
```

```
005900     02  EOB-1            PIC 9     COMP VALUE IS 0.
006000     02  EOB-2            PIC 9     COMP VALUE IS 0.
006100     02  BIFORCAZIONI     PIC 9(10) COMP VALUE IS 0.
006200     02  CODICE-ORIG      PIC 9(10) COMP VALUE IS 0.
006300     02  CODICE-ORIG-1    PIC 9(10) COMP VALUE IS 0.
006400     02  CODICE-ORIG-2    PIC 9(10) COMP VALUE IS 0.
006500     02  SCAMBIO          PIC 9     COMP VALUE IS 0.
006600*
006700 PROCEDURE DIVISION.
006800*------------------------ LIVELLO 0 -----------------
006900 MAIN.
007000     PERFORM COPIA-FILE-MERGE.
007100     PERFORM BIFORCAZIONE.
007200     IF BIFORCAZIONI > 0
007300       THEN
007400           PERFORM FUSIONE,
007500           PERFORM BIFORCAZIONE-E-FUSIONE
007600                   UNTIL BIFORCAZIONI <= 2.
007700     STOP RUN.
007800*------------------------ LIVELLO 1 -----------------
007900 COPIA-FILE-MERGE.
008000     OPEN INPUT  FILE-IN.
008100     OPEN OUTPUT FILE-MERGE.
008200     MOVE ZERO TO EOF.
008300     PERFORM LETTURA-FILE-IN.
008400     PERFORM COPIA-RECORD-FILE-MERGE
008500             UNTIL EOF = 1.
008600     CLOSE FILE-MERGE.
008700     CLOSE FILE-IN.
008800*-------------------------------------------------------
008900 BIFORCAZIONE-E-FUSIONE.
009000     PERFORM BIFORCAZIONE.
009100     PERFORM FUSIONE.
009200*------------------------ LIVELLO 2 -----------------
009300 COPIA-RECORD-FILE-MERGE.
009400     MOVE RECORD-IN TO RECORD-MERGE.
009500     WRITE RECORD-MERGE.
009600     PERFORM LETTURA-FILE-IN.
009700*-------------------------------------------------------
009800 BIFORCAZIONE.
009900     MOVE ZERO TO BIFORCAZIONI.
010000     OPEN INPUT  FILE-MERGE.
010100     OPEN OUTPUT FILE-TMP-1.
010200     OPEN OUTPUT FILE-TMP-2.
010300     MOVE ZERO TO EOF.
010400     MOVE 1 TO SCAMBIO.
010500     PERFORM LETTURA-FILE-MERGE.
010600     IF EOF = 0
010700       THEN
010800           ADD 1 TO BIFORCAZIONI,
010900           MOVE RECORD-MERGE TO RECORD-TMP-1,
011000           WRITE RECORD-TMP-1,
011100           MOVE CODICE-MERGE TO CODICE-ORIG,
011200           PERFORM LETTURA-FILE-MERGE.
011300     PERFORM BIFORCAZIONE-SUCCESSIVA
011400             UNTIL EOF = 1.
011500     CLOSE FILE-TMP-2.
011600     CLOSE FILE-TMP-1.
011700     CLOSE FILE-MERGE.
011800*-------------------------------------------------------
011900 FUSIONE.
012000     OPEN INPUT  FILE-TMP-1.
012100     OPEN INPUT  FILE-TMP-2.
012200     OPEN OUTPUT FILE-MERGE.
012300     MOVE ZERO TO EOF-1.
012400     MOVE ZERO TO EOF-2.
012500     MOVE ZERO TO EOB-1.
012600     MOVE ZERO TO EOB-2.
012700     PERFORM LETTURA-FILE-TMP-1.
012800     IF EOF-1 = 0 AND EOB-1 = 0
012900       THEN
013000           MOVE CODICE-T1 TO CODICE-ORIG-1.
013100     PERFORM LETTURA-FILE-TMP-2.
013200     IF EOF-2 = 0 AND EOB-2 = 0
013300       THEN
013400           MOVE CODICE-T2 TO CODICE-ORIG-2.
013500     PERFORM FUSIONE-SUCCESSIVA
013600             UNTIL EOF-1 = 1 AND EOF-2 = 1.
013700     CLOSE FILE-MERGE.
013800     CLOSE FILE-TMP-2.
013900     CLOSE FILE-TMP-1.
```



```
014000*------------------------ LIVELLO 3 -----------------
014100 BIFORCAZIONE-SUCCESSIVA.
014200     IF CODICE-MERGE >= CODICE-ORIG
014300       THEN
014400           IF SCAMBIO = 1
014500             THEN
014600                 MOVE RECORD-MERGE TO RECORD-TMP-1,
014700                 WRITE RECORD-TMP-1,
014800                 MOVE CODICE-MERGE TO CODICE-ORIG,
014900                 PERFORM LETTURA-FILE-MERGE;
015000             ELSE
015100                 MOVE RECORD-MERGE TO RECORD-TMP-2,
015200                 WRITE RECORD-TMP-2,
015300                 MOVE CODICE-MERGE TO CODICE-ORIG,
015400                 PERFORM LETTURA-FILE-MERGE;
015500       ELSE
015600           ADD 1 TO BIFORCAZIONI,
015700           MOVE CODICE-MERGE TO CODICE-ORIG,
015800           IF SCAMBIO = 1
015900             THEN
016000                 MOVE 2 TO SCAMBIO;
016100             ELSE
016200                 MOVE 1 TO SCAMBIO.
016300*-------------------------------------------------------
016400 FUSIONE-SUCCESSIVA.
016500     PERFORM FUSIONE-BLOCCO
016600             UNTIL EOB-1 = 1 AND EOB-2 = 1.
016700     IF NOT EOF-1 = 1
016800       THEN
016900           MOVE ZERO TO EOB-1.
017000     IF NOT EOF-2 = 1
017100       THEN
017200           MOVE ZERO TO EOB-2.
017300*------------------------ LIVELLO 4 -----------------
017400 FUSIONE-BLOCCO.
017500     IF EOB-1 = 1
017600       THEN
017700           MOVE RECORD-TMP-2 TO RECORD-MERGE,
017800           PERFORM LETTURA-FILE-TMP-2;
017900       ELSE
018000           IF EOB-2 = 1
018100             THEN
018200                 MOVE RECORD-TMP-1 TO RECORD-MERGE,
018300                 PERFORM LETTURA-FILE-TMP-1;
018400             ELSE
018500                 IF CODICE-T1 < CODICE-T2
018600                   THEN
018700                       MOVE RECORD-TMP-1
018750                            TO RECORD-MERGE,
018800                       PERFORM LETTURA-FILE-TMP-1;
018900                       IF EOF-1 = 0 AND EOB-1 = 0
019000                         THEN
019100                             IF CODICE-T1
019150                                >= CODICE-ORIG-1
019200                               THEN
019300                                   MOVE CODICE-T1
019400                                     TO CODICE-ORIG-1;
019500                               ELSE
019600                                   MOVE 1 TO EOB-1;
019700                         ELSE
019800                             NEXT SENTENCE;
019900                   ELSE
020000                       MOVE RECORD-TMP-2
020050                            TO RECORD-MERGE,
020100                       PERFORM LETTURA-FILE-TMP-2;
020200                       IF EOF-2 = 0 AND EOB-2 = 0
020300                         THEN
020400                             IF CODICE-T2
020450                                >= CODICE-ORIG-2
020500                               THEN
020600                                   MOVE CODICE-T2
020700                                     TO CODICE-ORIG-2;
020800                               ELSE
020900                                   MOVE 1 TO EOB-2.
021000     WRITE RECORD-MERGE.
021200*------------------------ LIVELLO 5 -----------------
021300 LETTURA-FILE-IN.
021400     READ FILE-IN
021500          AT END
021600               MOVE 1 TO EOF.
021700*-------------------------------------------------------
```

```
021800 LETTURA-FILE-MERGE.
021900     READ FILE-MERGE
022000          AT END
022100               MOVE 1 TO EOF.
022200*--------------------------------------------------
022300 LETTURA-FILE-TMP-1.
022400     READ FILE-TMP-1
022500          AT END
022600               MOVE 1 TO EOF-1,
022700               MOVE 1 TO EOB-1.
022800*--------------------------------------------------
022900 LETTURA-FILE-TMP-2.
023000     READ FILE-TMP-2
023100          AT END
023200               MOVE 1 TO EOF-2,
023300               MOVE 1 TO EOB-2.
023400*
```

## 73.4 Approfondimento: una tecnica per simulare la ricorsione in COBOL

Questa sezione contiene la ricostruzione di un documento con lo stesso nome, concluso nel mese di giugno del 1985, dopo un periodo di studio sul linguaggio COBOL. Il COBOL è un linguaggio procedurale che offre esclusivamente la gestione di variabili globali, pertanto non consente di realizzare la ricorsione; tuttavia, qui, come esercizio, si descrive una tecnica per arrivare a ottenere un risultato simile alla ricorsione comune.

Si fa riferimento a tre algoritmi noti: torre di Hanoi, quicksort e permutazioni. Questi algoritmi sono descritti nella sezione 62.2.

Al termine è riportata la bibliografia dello studio originale. Tutti gli esempi originali con il linguaggio MPL II sono omessi, anche se nella bibliografia questo linguaggio viene citato.

### 73.4.1 Il concetto di locale e di globale

Niklaus Wirth **[1]** spiega molto bene la differenza tra il concetto di ***locale*** e di ***globale*** all'interno di un programma:

> *Se un oggetto –una costante, una variabile, una procedura, una funzione o un tipo– è significativo solo all'interno di una parte determinata del programma, viene chiamato «locale». Spesso conviene rappresentare questa parte mediante una procedura; gli oggetti locali vengono allora indicati nel titolo della procedura. Dato che le procedure stesse possono essere locali, può accadere che più indicazioni di procedura siano innestate l'una nell'altra.*
>
> *Nell'ambito della procedura si possono quindi riconoscere due tipi di oggetti: gli oggetti «locali» e gli oggetti «non locali». Questi ultimi sono oggetti definiti nel programma (o nella procedura) in cui è inserita la procedura («ambiente» della procedura). Se sono definiti nel programma principale, sono detti «globali». In una procedura il campo di influenza degli oggetti locali corrisponde al corpo della procedura. In particolare, terminata l'esecuzione della procedura, le variabili locali saranno ancora disponibili per indicare dei nuovi valori; chiaramente, in una chiamata successiva della stessa procedura, i valori delle variabili locali saranno diversi da quelli della chiamata precedente.*
>
> *È essenziale che i nomi degli oggetti locali non debbano dipendere dall'ambiente della procedura. Ma, in tal modo, può accadere che un nome «x», scelto per un oggetto locale della procedura «P», sia identico a quello di un oggetto definito nel programma ambiente di «P». Questa situazione però è corretta solo se la grandezza non locale «x» non è significativa per «P», cioè non viene applicata in «P». Si adotta quindi la «regola fondamentale» che «x» denoti entro «P» la grandezza locale e fuori da «P» quella non locale.*



### 73.4.2 La ricorsione

«La ricorsione», come spiegano Ledgard, Nagin e Hueras **[2]**, «è un metodo di definizione in cui l'oggetto della definizione è usato all'interno della definizione». Per esempio si può considerare la seguente definizione della parola «discendente»:

> Un discendente di una persona è il figlio o la figlia di quella persona, o un discendente del figlio o della figlia.

Quindi, come scrive Lawrie Moore **[3]**, un sottoprogramma ricorsivo «è un sottoprogramma che corrisponde direttamente e utilizza una definizione ricorsiva». Ovvero, molto più semplicemente come dicono Aho, Hopcroft e Ullman **4**: «Una procedura che chiama se stessa, direttamente o indirettamente, si dice essere ricorsiva».

Moore **[3]** inoltre aggiunge quanto segue: «La chiamata genera un nuovo blocco di programma, con il suo proprio ambito, il suo proprio spazio di lavoro, la sua propria esistenza virtuale. [...] Questo processo prende luogo al momento dell'esecuzione del programma (run-time). Al momento della compilazione né la macchina, né l'intelligenza umana possono dire quante volte la procedura sarà richiamata al momento dell'esecuzione. Perciò, la creazione di un nuovo blocco di programma al momento dell'esecuzione è un processo dinamico. La creazione ricorsiva di nuovi blocchi di programma è una struttura di programmazione dinamica».

### 73.4.3 Proprietà del linguaggio ricorsivo

La definizione di procedura ricorsiva data da Aho, Hopcroft e Ullman è una condizione necessaria ma non sufficiente perché un linguaggio di programmazione possa definirsi ricorsivo. Infatti, è tale quel linguaggio che oltre a permettere la chiamata di una procedura da parte di se stessa, permette una dichiarazione locale delle variabili, ovvero permette l'allocazione dinamica delle variabili stesse.

Non vi è dubbio che il linguaggio COBOL non sia ricorsivo, eppure ammette che all'interno di un paragrafo si faccia la chiamata dello stesso paragrafo tramite l'istruzione '**`PERFORM`**'. In effetti non si parla di ricorsione proprio perché il COBOL gestisce solo variabili globali.

### 73.4.4 Descrizione della tecnica per simulare la ricorsione in COBOL

Le variabili di scambio di un sottoprogramma possono collegarsi all'esterno, a seconda del contesto del programma, in tre modi: in input, in output o in input-output, a seconda che importi che i dati entrino nel sottoprogramma ma non escano, che i dati escano soltanto oppure che i dati debbano prima entrare e poi uscire modificati.

La pseudocodifica utilizzata per mostrare gli esempi, prima di presentare la trasformazione in COBOL, si rifà al linguaggio MPL II Burroughs, dove le variabili di scambio di una procedura vengono semplicemente nominate a fianco del nome della procedura tra parentesi. Ciò corrisponde a una dichiarazione implicita di quelle variabili con ambito locale e con caratteristiche identiche a quelle usate nelle chiamate relative. In particolare, se nella chiamata vengono usate costanti alfanumeriche, la variabile corrispondente sarà di tipo alfanumerico di lunghezza pari alla costante trasmittente, se di tipo numerico, la variabile corrispondente sarà di tipo numerico opportuno: intero o a virgola mobile.
Quindi, in questo tipo di pseudocodifica non sono permesse le variabili di scambio in output.
Le variabili di scambio di questa pseudocodifica si collegano per posizione.

Il problema della simulazione della ricorsione si risolve utilizzando una pila (*stack*) per ogni variabile locale.
La tecnica è indicata molto semplicemente da Jerrold L. Wagener **[5]**. Una volta determinato a priori qual è il numero massimo di livelli della ricorsione, occorre associare a ogni variabile locale, che non

sia collegata con l'esterno in input-output, una pila con dimensioni pari a quel numero. Quindi, a una variabile scalare viene associato un vettore, a un vettore viene associata una matrice a due dimensioni e così di seguito. L'indice della pila (*stack pointer*) viene indicato con '**SP**'.
La simulazione si divide in due fasi: la prima deve essere effettuata subito prima della chiamata ricorsiva e consiste nella conservazione delle varie pile dei valori delle variabili di scambio che non sono in input-output con un'operazione di inserimento (*push*); la seconda deve essere effettuata subito dopo la chiamata ricorsiva e consiste nel recupero dalle varie pile dei valori originali delle variabili con un'operazione di estrazione (*pop*).

Figura 73.39. Confronto tra una procedura ricorsiva e la sua trasformazione non ricorsiva, attraverso la pseudocodifica.

```
#                              #
# Procedura ricorsiva          # Trasformazione non ricorsiva
#                              #
PROC1 (V, G, Z)                PROC1
    .                              .
    .                              .
    .                              .
    .                              # push
    .                              SP := SP + 1
    .                              SAVEV(SP) := V
    .                              SAVEZ(SP) := Z
    # G è una variabile in         # chiamata
    # input-output                 Z := Z -1
    PROC1 (V, G, Z-1)              PROC1
    .                              # pop
    .                              V := SAVEV(SP)
    .                              Z := SAVEZ(SP)
    .                              SP := SP - 1
    .                              .
    .                              .
    .                              .
END PROC1                      END PROC1
```

È bene precisare che la tecnica è valida solo se all'interno di una procedura ricorsiva tutte le iterazioni che contengono una chiamata (diretta o indiretta) alla stessa procedura sono a loro volta espresse in forma ricorsiva (si veda il problema delle permutazioni).

### 73.4.5 Torre di Hanoi

Segue la descrizione dell'algoritmo attraverso la pseudocodifica in forma ricorsiva. Nella sezione 62.5.3 viene descritto il problema della torre di Hanoi.

| Variabile | Descrizione |
|---|---|
| N | È la dimensione della torre espressa in numero di anelli: gli anelli sono numerati da 1 a '**N**'. |
| P1 | È il numero del piolo su cui si trova inizialmente la pila di '**N**' anelli. |
| P2 | È il numero del piolo su cui deve essere spostata la pila di anelli. |
| 6-P1-P2 | È il numero dell'altro piolo. Funziona così se i pioli sono numerati da 1 a 3. |

```
HANOI (N, P1, P2)
    IF N > 0
        THEN
            HANOI (N-1, P1, 6-P1-P2)
            scrivi: "Muovi l'anello" N "dal piolo" P1 "al piolo" P2
            HANOI (N-1, 6-P1-P2, P2)
    END IF
END HANOI
```

Segue la descrizione della trasformazione in modo tale da simulare la ricorsione.

| Variabile | Descrizione |
|---|---|
| SAVEN | È il vettore utilizzato per conservare il valore di '**N**'. |
| SAVEP1 | È il vettore utilizzato per conservare il valore di '**P1**'. |
| SAVEP2 | È il vettore utilizzato per conservare il valore di '**P2**'. |



| Variabile | Descrizione |
|---|---|
| SP | È l'indice dei vettori usati per salvare i valori (*stack pointer*). |

```
HANOI (N, P1, P2)
    IF N > 0
        THEN
            SP := SP + 1
            SAVEN(SP) := N
            SAVEP2(SP) := P2
            N := N - 1
            P2 := 6 - P1 - P2
            HANOI
            N := SAVEN(SP)
            P2 := SAVEP2(SP)
            SP = SP - 1
            scrivi: "Muovi l'anello" N "dal piolo" P1 "al piolo" P2
            SP := SP + 1
            SAVEN(SP) := N
            SAVEP1(SP) := P1
            N := N - 1
            P1 := 6 - P1 - P2
            HANOI
            N := SAVEN(SP)
            P1 := SAVEP1(SP)
            SP = SP - 1
    END IF
END HANOI
```

Listato 73.44. Soluzione in COBOL del problema della torre di Hanoi, con la simulazione della ricorsione.

```
000600 IDENTIFICATION DIVISION.
000700 PROGRAM-ID.   HC04.
000800 AUTHOR.       DANIELE GIACOMINI.
000900 DATE-WRITTEN. 1984-08-18.
001000
001100
001200 ENVIRONMENT DIVISION.
001300
001400
001500 DATA DIVISION.
001600
001700
001800 WORKING-STORAGE SECTION.
001900
002000 01  RECORD-STACKS.
002100     02  SAVEN  OCCURS 100 TIMES PIC 99.
002200     02  SAVEP1 OCCURS 100 TIMES PIC 9.
002300     02  SAVEP2 OCCURS 100 TIMES PIC 9.
002400
002500 01  STACK-POINTER.
002600     02  SP                      PIC 99 VALUE 0.
002700
002800 01  VARIABILI-SCALARI.
002900     02  N                       PIC 99.
003000     02  P1                      PIC 9.
003100     02  P2                      PIC 9.
003200
003300
003400 PROCEDURE DIVISION.
003500
003600 MAIN.
003700
003800     DISPLAY "INSERISCI LA DIMENSIONE DELLA TORRE".
003900     DISPLAY "(DUE CARATTERI)".
004000     ACCEPT N.
004100
004200     DISPLAY "INSERISCI LA POSIZIONE INIZIALE ",
004250             "DELLA TORRE".
004300     DISPLAY "(UN CARATTERE)".
004400     ACCEPT P1.
004500
004600     DISPLAY "INSERISCI LA DESTINAZIONE DELLA TORRE".
004700     DISPLAY "(UN CARATTERE)".
004800     ACCEPT P2.
004900
005000     PERFORM HANOI.
005100
005200     STOP RUN.
005300
005400 HANOI.
005500
005600     IF N > 0
```

```
005700       THEN
005800*
005900*          push per conservare le variabili di scambio
006000*
006100           COMPUTE SP = SP + 1,
006200           COMPUTE SAVEN(SP) = N,
006300           COMPUTE SAVEP2(SP) = P2,
006400*
006500*          cambiamenti alle variabili di scambio prima
006600*          della chiamata
006700*
006800           COMPUTE N = N - 1,
006900           COMPUTE P2 = 6 - P1 - P2,
007000*
007100*          chiamata della procedura
007200*
007300           PERFORM HANOI,
007400*
007500*          pop per recuperare i valori delle variabili
007550*          di scambio
007600*
007700           COMPUTE P2 = SAVEP2(SP),
007800           COMPUTE N = SAVEN(SP),
007900           COMPUTE SP = SP - 1,
008000
008100           DISPLAY "MUOVI L'ANELLO ", N,
008150                   " DAL PIOLO ", P1,
008200                   " AL PIOLO ", P2,
008300
008400*
008500*          push per conservare le variabili di scambio
008600*
008700           COMPUTE SP = SP + 1,
008800           COMPUTE SAVEN(SP) = N,
008900           COMPUTE SAVEP1(SP) = P1,
009000*
009100*          modifica dei valori delle variabili di
009159*          scambio
009200*
009300           COMPUTE N = N - 1,
009400           COMPUTE P1 = 6 - P1 - P2,
009500*
009600*          chiamata della procedura
009700*
009800           PERFORM HANOI,
009900*
010000*          pop per recuperare i valori delle variabili
010050*          di scambio
010100*
010200           COMPUTE P1 = SAVEP1(SP),
010300           COMPUTE N = SAVEN(SP),
010400           COMPUTE SP = SP - 1.
010500
```

### 73.4.6 Quicksort (ordinamento non decrescente)

«

Segue la descrizione dell'algoritmo attraverso la pseudocodifica in forma ricorsiva; si ricorda che l'algoritmo del Quicksort si risolve con due subroutine: una serve a suddividere il vettore; l'altra esegue le chiamate ricorsive. Nella sezione 62.5.4 viene descritto il problema del Quicksort in modo dettagliato.

| Variabile | Descrizione |
|---|---|
| `LISTA` | L'array da ordinare in modo crescente. |
| `A` | L'indice inferiore del segmento di array da ordinare. |
| `Z` | L'indice superiore del segmento di array da ordinare. |
| `CF` | Sta per «collocazione finale» ed è l'indice che cerca e trova la posizione giusta di un elemento nell'array. |
| `I` | È l'indice che insieme a '**`CF`**' serve a ripartire l'array. |

```
PART (LISTA, A, Z)

    LOCAL I INTEGER
    LOCAL CF INTEGER

    # si assume che A < U
    I := A + 1
    CF := Z

    WHILE TRUE # ciclo senza fine.

        WHILE TRUE

            # sposta I a destra

            IF (LISTA[I] > LISTA[A]) OR I >= CF
                THEN
                    BREAK
                ELSE
                    I := I + 1
            END IF

        END WHILE

        WHILE TRUE

            # sposta CF a sinistra

            IF (LISTA[CF] <= LISTA[A])
                THEN
                    BREAK
                ELSE
                    CF := CF - 1
            END IF

        END WHILE

        IF CF <= I
            THEN
                # è avvenuto l'incontro tra I e CF
                BREAK
            ELSE
                # vengono scambiati i valori
                LISTA[CF] :==: LISTA[I]
                I := I + 1
                CF := CF - 1
        END IF

    END WHILE

    # a questo punto LISTA[A:Z] è stata ripartita e CF è
    # la collocazione di LISTA[A]

    LISTA[CF] :==: LISTA[A]

    # a questo punto, LISTA[CF] è un elemento (un valore)
    # nella giusta posizione

    RETURN CF

END PART
```

```
QSORT (LISTA, A, Z)

    LOCAL CF INTEGER

    IF Z > A
        THEN
            CF := PART (@LISTA, A, Z)
            QSORT (@LISTA, A, CF-1)
            QSORT (@LISTA, CF+1, Z)
    END IF
END QSORT
```

Vale la pena di osservare che l'array viene indicato nelle chiamate in modo che alla subroutine sia inviato un riferimento a quello originale, perché le variazioni fatte all'interno delle subroutine devono riflettersi sull'array originale.

La subroutine '**`QSORT`**' è quella che richiede la trasformazione per la simulazione della ricorsione; tuttavia, anche la subroutine deve essere adattata in modo tale da gestire la variabile '**`CF`**' come variabile globale (non potendo gestire variabili di '**`output`**'). Segue la descrizione di tali adattamenti.

| Variabile | Descrizione |
|---|---|
| SAVEA | È il vettore utilizzato per conservare il valore di '**A**'. |
| SAVEZ | È il vettore utilizzato per conservare il valore di '**Z**'. |
| SP | È l'indice dei vettori usati per salvare i valori (*stack pointer*). |

```
PART (LISTA, A, Z)

    LOCAL I INTEGER

    # si assume che A < U

    I := A + 1
    CF := Z

    WHILE TRUE # ciclo senza fine.

        WHILE TRUE

            # sposta I a destra

            IF (LISTA[I] > LISTA[A]) OR I >= CF
                THEN
                    BREAK
                ELSE
                    I := I + 1
            END IF

        END WHILE

        WHILE TRUE

            # sposta CF a sinistra

            IF (LISTA[CF] <= LISTA[A])
                THEN
                    BREAK
                ELSE
                    CF := CF - 1
            END IF

        END WHILE

        IF CF <= I
            THEN
                # è avvenuto l'incontro tra I e CF
                BREAK
            ELSE
                # vengono scambiati i valori
                LISTA[CF] :==: LISTA[I]
                I := I + 1
                CF := CF - 1
        END IF

    END WHILE

    # a questo punto LISTA[A:Z] è stata ripartita e CF è
    # la collocazione di LISTA[A]

    LISTA[CF] :==: LISTA[A]

    # a questo punto, LISTA[CF] è un elemento (un valore)
    # nella giusta posizione

END PART
```



```
QSORT
    IF Z > A
        THEN
            PART
            SP := SP + 1
            SAVEZ(SP) := Z
            Z := CF - 1
            QSORT
        #   SP := SP - 1
        #   SP := SP + 1
            SAVEA(SP) := A
            A := CF + 1
            QSORT
            A := SAVEA(SP)
            SP := SP - 1
    END IF
END QSORT
```

Listato 73.51. Soluzione in COBOL del problema del Quicksort, con la simulazione della ricorsione. Si osservi che '**CF**' è una parola riservata del linguaggio, pertanto viene sostituita con '**C-F**'.

```
000600 IDENTIFICATION DIVISION.
000700 PROGRAM-ID.   HC06.
000800 AUTHOR.       DANIELE GIACOMINI.
000900 DATE-WRITTEN. 1984-08-22.
001000
001100
001200 ENVIRONMENT DIVISION.
001300
001400
001500 DATA DIVISION.
001600
001700
001800 WORKING-STORAGE SECTION.
001900
002000 01  RECORD-STACKS.
002100     02  SAVEA   OCCURS 100 TIMES PIC 999.
002200     02  SAVEZ   OCCURS 100 TIMES PIC 999.
002300
002400 01  STACK-POINTER.
002500     02  SP                       PIC 999.
002600
002700 01  VARIABILI-SCALARI.
002800     02  C-F                      PIC 999.
002900     02  A                        PIC 999.
003000     02  Z                        PIC 999.
003100     02  TEMP                     PIC X(15).
003200     02  I                        PIC 999.
003300     02  J                        PIC 999.
003400
003500 01  RECORD-TABELLA.
003600     02  TABELLA OCCURS 100 TIMES PIC X(15).
003700
003800 PROCEDURE DIVISION.
003900
004000 MAIN.
004100
004200     DISPLAY "INSERISCI IL NUMERO DI ELEMENTI ",
004250             "DA ORDINARE".
004300     DISPLAY "(TRE CIFRE)".
004400     ACCEPT Z.
004500     IF Z > 100
004600       THEN
004700           STOP RUN.
004800
004900     COMPUTE A = 1.
005000
005100     PERFORM INSERIMENTO-ELEMENTI
005150             VARYING J FROM 1 BY 1
005200             UNTIL   J > Z.
005300
005400     PERFORM QSORT.
005500
005600     PERFORM OUTPUT-DATI VARYING J FROM 1 BY 1
005700                         UNTIL   J > Z.
005800
005900     STOP RUN.
006000
006100
006200 INSERIMENTO-ELEMENTI.
```

```
006300
006400     DISPLAY "INSERISCI L'ELEMENTO ", J,
006450             " DELLA TABELLA".
006500     ACCEPT TABELLA(J).
006600
006700
006800 PART.
006900
007000*
007100*    si assume che A < Z
007200*
007300     COMPUTE I = A + 1.
007400     COMPUTE C-F = Z.
007500
007600     PERFORM PART-TESTA-MAINLOOP.
007700     PERFORM PART-MAINLOOP UNTIL C-F < I
007800                              OR C-F = I.
007900
008000     MOVE TABELLA(C-F) TO TEMP.
008100     MOVE TABELLA(A)   TO TABELLA(C-F).
008200     MOVE TEMP         TO TABELLA(A).
008300
008400
008500 PART-TESTA-MAINLOOP.
008600
008700     PERFORM SPOSTA-I-A-DESTRA
008750             UNTIL TABELLA(I) > TABELLA(A)
008800                   OR I > C-F
008900                   OR I = C-F.
009000
009100     PERFORM SPOSTA-C-F-A-SINISTRA
009200             UNTIL TABELLA(C-F) < TABELLA(A)
009300                OR TABELLA(C-F) = TABELLA(A).
009400
009500
009600 PART-MAINLOOP.
009700
009800     MOVE TABELLA(C-F) TO TEMP.
009900     MOVE TABELLA(I)   TO TABELLA(C-F).
010000     MOVE TEMP         TO TABELLA(I).
010100
010200     COMPUTE I = I + 1.
010300     COMPUTE C-F = C-F - 1.
010400
010500     PERFORM SPOSTA-I-A-DESTRA
010550             UNTIL TABELLA(I) > TABELLA(A)
010600                   OR I > C-F
010700                   OR I = C-F.
010800
010900     PERFORM SPOSTA-C-F-A-SINISTRA
011000             UNTIL TABELLA(C-F) < TABELLA(A)
011100                OR TABELLA(C-F) = TABELLA(A).
011200
011300
011400 SPOSTA-I-A-DESTRA.
011500
011600     COMPUTE I = I + 1.
011700
011800
011900 SPOSTA-C-F-A-SINISTRA.
012000
012100     COMPUTE C-F = C-F - 1.
012200
012300
012400 QSORT.
012500
012600     IF Z > A
012700       THEN
012800*
012900*          le variabili che riguardano PART sono tutte
012950*          in I-O
013000*
013100           PERFORM PART,
013200*
013300*          push
013400*
013500           COMPUTE SP = SP + 1,
013600           COMPUTE SAVEZ(SP) = Z,
013700*
013800*          cambiamenti alle variabili di scambio
013900*
```



```
014000           COMPUTE Z = C-F - 1,
014100*
014200*          chiamata
014300*
014400           PERFORM QSORT,
014500*
014600*          pop
014700*
014800           COMPUTE Z = SAVEZ(SP),
014900           COMPUTE SP = SP - 1,
015000*
015100*          push
015200*
015300           COMPUTE SP = SP + 1,
015400           COMPUTE SAVEA(SP) = A,
015500*
015600*          cambiamenti alle variabili di scambio
015700*
015800           COMPUTE A = C-F + 1,
015900*
016000*          chiamata
016100*
016200           PERFORM QSORT,
016300*
016400*          pop
016500*
016600           COMPUTE A = SAVEA(SP),
016700           COMPUTE SP = SP - 1.
016800
016900
017000 OUTPUT-DATI.
017100
017200     DISPLAY "TABELLA(", J, ") = ", TABELLA(J).
017300
```

### 73.4.7 Permutazioni

La permutazione degli elementi di un vettore si risolve generalmente attraverso un algoritmo iterativo normale; segue la descrizione dell'algoritmo iterativo in forma di pseudocodifica. Nella sezione 62.5.5 viene descritto il problema delle permutazioni in modo dettagliato.

| Variabile | Descrizione |
|---|---|
| `LISTA` | L'array da permutare. |
| `A` | L'indice inferiore del segmento di array da permutare. |
| `Z` | L'indice superiore del segmento di array da permutare. |
| `K` | È l'indice che serve a scambiare gli elementi. |

```
PERMUTA (LISTA, A, Z)

    LOCAL K INTEGER
    LOCAL N INTEGER

    IF (Z - A) >= 1
        # Ci sono almeno due elementi nel segmento di array.
        THEN
            FOR K := Z; K >= A; K--

                LISTA[K] :==: LISTA[Z]

                PERMUTA (LISTA, A, Z-1)

                LISTA[K] :==: LISTA[Z]

            END FOR
        ELSE
            scrivi LISTA
    END IF

END PERMUTA
```

Per esercizio, l'algoritmo iterativo viene trasformato in modo ricorsivo:

```
PERMUTA (LISTA, A, Z)
```

```
    LOCAL K INTEGER
    LOCAL N INTEGER

    SCAMBIO_CHIAMATA_SCAMBIO (LISTA, A, Z, K)
        IF K >= A
            THEN
                LISTA[K] :==: LISTA[Z]
                PERMUTA (LISTA, A, Z-1)
                LISTA[K] :==: LISTA[Z]
                SCAMBIO_CHIAMATA_SCAMBIO (LISTA, A, Z, K - 1)
        END IF
    END SCAMBIO_CHIAMATA_SCAMBIO

    IF Z > A
        THEN
            SCAMBIO_CHIAMATA_SCAMBIO (LISTA, A, Z, Z)
        ELSE
            scrivi LISTA
    END IF

END PERMUTA
```

Segue l'adattamento della pseudocodifica appena mostrata, in modo da simulare la ricorsione, utilizzando variabili globali:

| Variabile | Descrizione |
|---|---|
| SAVEZ | È il vettore utilizzato per conservare il valore di '**Z**'. |
| SAVEK | È il vettore utilizzato per conservare il valore di '**K**'. |
| SP | È l'indice dei vettori usati per salvare i valori (*stack pointer*). |

```
PERMUTA (LISTA, A, Z)

    SCAMBIO_CHIAMATA_SCAMBIO
        IF K >= A
            THEN
                LISTA[K] :==: LISTA[Z]
                SP := SP + 1
                SAVEZ(SP) := Z
                Z := Z - 1
                PERMUTA
                Z := SAVEZ(SP)
                SP := SP - 1
                LISTA[K] :==: LISTA[Z]
                SP := SP + 1
                SAVEK(SP) := K
                K := K - 1
                SCAMBIO_CHIAMATA_SCAMBIO
                K := SAVEK(SP)
                SP := SP - 1
        END IF
    END SCAMBIO_CHIAMATA_SCAMBIO

    IF Z > A
        THEN
            SP := SP + 1
            SAVEK(SP) := K
            K := N
            SCAMBIO_CHIAMATA_SCAMBIO
            K := SAVEK(SP)
            SP := SP - 1
        ELSE
            scrivi LISTA
    END IF
END PERMUTA
```

Listato 73.57. Soluzione in COBOL del problema delle permutazioni, con la simulazione della ricorsione.

```
000600 IDENTIFICATION DIVISION.
000700 PROGRAM-ID.   HC07.
000800 AUTHOR.       DANIELE GIACOMINI.
000900 DATE-WRITTEN. 1985-06-19.
001000
001100
001200 ENVIRONMENT DIVISION.
001300
001400
001500 DATA DIVISION.
001600
```



```
001700
001800 WORKING-STORAGE SECTION.
001900
002000 01  RECORD-STACKS.
002100     02  SAVEZ   OCCURS 100 TIMES PIC 9.
002200     02  SAVEK   OCCURS 100 TIMES PIC 9.
002300
002400 01  STACK-POINTER.
002500     02  SP                       PIC 999.
002600
002700 01  VARIABILI-SCALARI.
002800     02  A                        PIC 9   VALUE 1.
002900     02  Z                        PIC 9.
003000     02  K                        PIC 9.
003100     02  TEMP                     PIC 9.
003200     02  J                        PIC 99.
003300
003400 01  RECORD-LISTA.
003500     02  LISTA   OCCURS 10 TIMES PIC 9.
003600
003700
003800 PROCEDURE DIVISION.
003900
004000 MAIN.
004100
004200     DISPLAY "INSERISCI IL NUMERO DI ELEMENTI ",
004250             "DA PERMUTARE".
004300     DISPLAY "(UNA CIFRA)".
004400     ACCEPT Z.
004500*
004600*    si genera la prima permutazione con numeri in
004650*    ordine crescente
004800*
004900     MOVE SPACES TO RECORD-LISTA.
005000     PERFORM GEN-PRIMA-PERMUTAZIONE
005050             VARYING J FROM 1 BY 1
005100             UNTIL J > Z.
005200
005300     PERFORM PERMUTA.
005400
005500     STOP RUN.
005600
005700
005800 GEN-PRIMA-PERMUTAZIONE.
005900
006000     MOVE J TO LISTA(J).
006100
006200
006300 PERMUTA.
006400
006500     IF Z > A
006600       THEN
006700*
006800*          push
006900*
007000           COMPUTE SP = SP + 1,
007100           COMPUTE SAVEK(SP) = K,
007200*
007300*          chiamata
007400*
007500           COMPUTE K = Z,
007600           PERFORM SCAMBIO-CHIAMATA-SCAMBIO,
007700*
007800*          pop
007900*
008000           COMPUTE K = SAVEK(SP),
008100           COMPUTE SP = SP - 1,
008200
008300       ELSE
008400
008500           DISPLAY RECORD-LISTA.
008600
008700
008800 SCAMBIO-CHIAMATA-SCAMBIO.
008900
009000     IF K >= A
009100       THEN
009200*
009300*      scambio di LISTA(K) con LISTA(Z)
009400*
009500       MOVE LISTA(K) TO TEMP,
```

```
009600       MOVE LISTA(Z) TO LISTA (K),
009700       MOVE TEMP     TO LISTA (Z),
009800*
009900*      push
010000*
010100       COMPUTE SP = SP + 1,
010200       COMPUTE SAVEZ(SP) = Z,
010300*
010400*      chiamata
010500*
010600       COMPUTE Z = Z - 1,
010700       PERFORM PERMUTA,
010800*
010900*      pop
011000*
011100       COMPUTE Z = SAVEZ(SP),
011200       COMPUTE SP = SP - 1,
011300*
011400*      scambio di LISTA(K) con LISTA(Z)
011500*
011600       MOVE LISTA(K) TO TEMP,
011700       MOVE LISTA(Z) TO LISTA (K),
011800       MOVE TEMP     TO LISTA (Z),
011900*
012000*      push
012100*
012200       COMPUTE SP = SP + 1,
012300       COMPUTE SAVEK(SP) = K,
012400*
012500*      chiamata
012600*
012700       COMPUTE K = K - 1,
012800       PERFORM SCAMBIO-CHIAMATA-SCAMBIO,
012900*
013000*      pop
013100*
013200       COMPUTE K = SAVEK(SP),
013300       COMPUTE SP = SP - 1.
013400
```

### 73.4.8 Bibliografia

«

- Wagener J. L., *FORTRAN 77 Principles of Programming*, Wiley, 1980, pagine 228..229. **[5]**
- Knuth D. E., *The Art of Computer Programming - Volume 3 Sorting and Searching*, Addison-Wesley, 1973, capitolo 5.
- Dijkstra E. W., *A Discipline of Programming*, Prentice-Hall, 1976, capitolo 13.

**Il concetto di locale e di globale: ambito delle variabili**

- Wirth N., *Principi di programmazione strutturata*, ISEDI, 1977, capitolo 12. **[1]**
- Moore L., *Foundations of Programming with Pascal*, Ellis Horwood Limited, 1980, capitolo 10. **[3]**
- Ledgard, Nagin, Hueras, *Pascal with Style*, Hayden, 1979, pagine 126..134. **[2]**
- Dijkstra E. W., *A Discipline of Programming*, Prentice-Hall, 1976, capitolo 10.
- Nicholls J. E., *The Structure and Design of Programming Languages*, Addison-Wesley, 1975, capitolo 12.

**La ricorsione**

- Arsac J., *La construction de programmes structures*, DUNOD, 1977, capitoli 2..5.
- Moore L., *Foundations of Programming with Pascal*, Ellis Horwood Limited, 1980, capitolo 14.
- Aho, Hopcroft, Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, pagine 55..60. **[4]**
- Ledgard, Nagin, Hueras, *Pascal with Style*, Hayden, 1979, pagine 134..139.
- Wirth N., *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976, capitolo 3.



- Wagener J. L., *FORTRAN 77 Principles of Programming*, Wiley, 1980, capitolo 11. **[5]**

**I linguaggi**

- Burroughs Corporation, *Computer Management System COBOL*, codice 2007266.
- Burroughs Corporation, *Computer Management System Message Processing Language (MPLII) - reference manual*, codice 2007563.

Figura 73.58. Ultima foto del 1988 di un elaboratore Burroughs B91, prima della sua dismissione completa. Alla destra appaiono le unità a disco; in fondo il B91, preso dal lato posteriore, assieme a un terminale MT. Il materiale infiammabile a cui si riferisce la scritta sull'armadio era una bottiglietta di alcool, utilizzato come solvente per pulire manualmente i dischi (sia le unità rimovibili, sia i piatti del disco fisso) a seguito dei continui atterraggi delle testine. I piatti dei dischi venivano sfruttati fino a quando la traccia iniziale non risultava raschiata completamente, arrivando a volte anche a rimontarli fuori asse, allo scopo di utilizzare una superficie ancora efficiente per tale traccia. Le testine delle unità a disco dovevano compiere un tragitto molto lungo per raggiungere tutte le tracce del disco (con tutti i problemi che ne derivano a causa della dilatazione termica) e spesso il loro motorino si incagliava: per fare riprendere l'attività all'elaboratore occorreva colpire le unità sullo stesso asse delle testine, per sbloccare il loro movimento.

## 73.5 Riferimenti

«

- *TinyCOBOL*, http://tiny-cobol.sourceforge.net
- *OpenCOBOL*, http://www.opencobol.org

[1] **TinyCOBOL** GNU GPL e GNU LGPL

[2] **OpenCOBOL** GNU GPL e GNU LGPL