

# Pascal: esempi di programmazione



Problemi elementari di programmazione .....	2252
Somma tra due numeri positivi .....	2253
Moltiplicazione di due numeri positivi attraverso la somma 2254	
Divisione intera tra due numeri positivi .....	2256
Elevamento a potenza .....	2257
Radice quadrata .....	2260
Fattoriale .....	2261
Massimo comune divisore .....	2263
Numero primo .....	2264
Scansione di array .....	2266
Ricerca sequenziale .....	2266
Ricerca binaria .....	2269
Algoritmi tradizionali .....	2271
Bubblesort .....	2272
Torre di Hanoi .....	2275
Quicksort .....	2276
Permutazioni .....	2280

Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Lo scopo di questi esempi è solo didattico, utilizzando forme non ottimizzate per la velocità di esecuzione.

Problemi elementari di programmazione .....	2252
Somma tra due numeri positivi .....	2253
Moltiplicazione di due numeri positivi attraverso la somma 2254	
Divisione intera tra due numeri positivi .....	2256
Elevamento a potenza .....	2257
Radice quadrata .....	2260
Fattoriale .....	2261
Massimo comune divisore .....	2263
Numero primo .....	2264
Scansione di array .....	2266
Ricerca sequenziale .....	2266
Ricerca binaria .....	2269
Algoritmi tradizionali .....	2271
Bubblesort .....	2272
Torre di Hanoi .....	2275
Quicksort .....	2276
Permutazioni .....	2280

## Problemi elementari di programmazione



In questa sezione vengono mostrati alcuni algoritmi elementari portati in Pascal.

## Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione [62.3.1](#). <<

```
(* ===== *)
(* Somma.pas                                     *)
(* Somma esclusivamente valori positivi.        *)
(* ===== *)
program Sommare;

var      x      : integer;
         y      : integer;
         z      : integer;

(* ===== *)
(* somma( <x>, <y> )                             *)
(* ----- *)
function somma( x : integer; y : integer ) : integer;

var      z      : integer;
         i      : integer;

begin

    z := x;

    for i := 1 to y do begin
        z := z+1;
    end;

    somma := z;

end;

(* ===== *)
(* Inizio del programma.                       *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
```

```

Readln( y );

z := somma( x, y );

Write( x, ' + ', y, ' = ', z );

end.

(* ===== *)

```

In alternativa si può tradurre il ciclo **‘for’** in un ciclo **‘while’**:

```

function somma( x : integer; y : integer ) : integer;

var
    z      : integer;
    i      : integer;

begin

    z := x;
    i := 1;

    while i <= y do begin
        z := z+1;
        i := i+1;
    end;

    somma := z;

end;

```

Moltiplicazione di due numeri positivi attraverso la somma



Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione [62.3.2](#).

```

(* ===== *)
(* Moltiplica.pas                               *)
(* Moltiplica esclusivamente valori positivi.   *)
(* ===== *)
program Moltiplicare;

var
    x      : integer;

```

```

        y      : integer;
        z      : integer;

(* ===== *)
(* moltiplica( <x>, <y> ) *)
(* ----- *)
function moltiplica( x : integer; y : integer ) : integer;

var      z      : integer;
        i      : integer;

begin

    z := 0;

    for i := 1 to y do begin
        z := z+x;
    end;

    moltiplica := z;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := moltiplica( x, y );

    Write( x, ' * ', y, ' = ', z );

end.

(* ===== *)

```

In alternativa si può tradurre il ciclo **‘for’** in un ciclo **‘while’**:

```

function moltiplica( x : integer; y : integer ) : integer;

var      z          : integer;
         i          : integer;

begin

    z := 0;
    i := 1;

    while i <= y do begin
        z := z+x;
        i := i+1;
    end;

    moltiplica := z;

end;

```

## Divisione intera tra due numeri positivi

<<

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione [62.3.3](#).

```

(* ===== *)
(* Dividi.pas                               *)
(* Divide esclusivamente valori positivi.   *)
(* ===== *)
program Dividere;

var      x          : integer;
         y          : integer;
         z          : integer;

(* ===== *)
(* dividi( <x>, <y> )                          *)
(* ----- *)
function dividi( x : integer; y : integer ) : integer;

var      z          : integer;
         i          : integer;

begin

```

```

z := 0;
i := x;

while i >= y do begin
    i := i - y;
    z := z+1;
end;

dividi := z;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := dividi( x, y );

    Write( x, ' / ', y, ' = ', z );

end.
(* ===== *)

```

## Elevamento a potenza

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione [62.3.4](#). «

```

(* ===== *)
(* Exp.pas *)
(* Eleva a potenza. *)
(* ===== *)
program Potenza;

var    x        : integer;

```

```

        y      : integer;
        z      : integer;

(* ===== *)
(* exp( <x>, <y> ) *)
(* ----- *)
function exp( x : integer; y : integer ) : integer;

var      z      : integer;
        i      : integer;

begin

    z := 1;

    for i := 1 to y do begin
        z := z * x;
    end;

    exp := z;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := exp( x, y );

    Write( x, ' ** ', y, ' = ', z );

end.
(* ===== *)

```

In alternativa si può tradurre il ciclo **for** in un ciclo **while**:



```

(* ===== *)
(* exp( <x>, <y> ) *)
(* ----- *)
function exp( x : integer; y : integer ) : integer;

var      z      : integer;
         i      : integer;

begin

    z := 1;
    i := 1;

    while i <= y do begin
        z := z * x;
        i := i+1;
    end;

    exp := z;

end;

```

È possibile usare anche un algoritmo ricorsivo:

```

function exp( x : integer; y : integer ) : integer;

begin

    if x = 0 then
        begin
            exp := 0;
        end
    else if y = 0 then
        begin
            exp := 1;
        end
    else
        begin
            exp := ( x * exp(x, y-1) );
        end
    ;

end;

```

# Radice quadrata



Il problema della radice quadrata è descritto nella sezione [62.3.5](#).

```
(* ===== *)
(* Radice.pas                                     *)
(* Radice quadrata.                             *)
(* ===== *)
program RadiceQuadrata;

var      x      : integer;
         z      : integer;

(* ===== *)
(* radice( <x> )                                  *)
(* ----- *)
function radice( x : integer; ) : integer;

var      z      : integer;
         t      : integer;
         ciclo   : boolean;

begin

    z := 0;
    t := 0;
    ciclo := TRUE;

    while ciclo do begin

        t := z * z;

        if t > x then
            begin
                z := z-1;
                radice := z;
                ciclo := FALSE;
            end
        ;

        z := z+1;

    end;

end;
```

```

end;

(* ===== *)
(* Inizio del programma.                               *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il numero intero positivo: ' );
    Readln( x );

    z := radice( x );

    Writeln( 'La radice di ', x, ' e'' ', z );

end.
(* ===== *)

```

## Fattoriale

Il problema del fattoriale è descritto nella sezione [62.3.6](#).

```

(* ===== *)
(* Fact.pas                                           *)
(* Fattoriale.                                       *)
(* ===== *)
program Fattoriale;

var    x      : integer;
       z      : integer;

(* ===== *)
(* fact( <x> )                                       *)
(* ----- *)
function fact( x : integer ) : integer;

var    i      : integer;

begin

    i := x - 1;

    while i > 0 do begin

```

```

        x := x * i;
        i := i-1;

    end;

    fact := x;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il numero intero positivo: ' );
    Readln( x );

    z := fact( x );

    Writeln( 'Il fattoriale di ', x, ' e'' ', z );

end.

(* ===== *)

```

In alternativa, l' algoritmo si può tradurre in modo ricorsivo:

```

function fact( x : integer ) : integer;

begin

    if x > 1 then
        begin
            fact := ( x * fact( x - 1 ) )
        end
    else
        begin
            fact := 1
        end
    end;

end;

```

## Massimo comune divisore

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione [62.3.7](#). «

```

(* ===== *)
(* MCD.pas                                     *)
(* Massimo Comune Divisore.                   *)
(* ===== *)
program MassimoComuneDivisore;

var    x      : integer;
       y      : integer;
       z      : integer;

(* ===== *)
(* mcd( <x>, <y> )                               *)
(* ----- *)
function mcd( x : integer; y : integer ) : integer;

begin

    while x <> y do begin

        if x > y then
            begin
                x := x - y;
            end

```

```

        else
            begin
                y := y - x;
            end
        ;

    end;

    mcd := x;

end;

(* ===== *)
(* Inizio del programma.                               *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := mcd( x, y );

    Write( 'Il massimo comune divisore tra ', x, ' e ', y, ' e'' ', z );

end.

(* ===== *)

```

## Numero primo



Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione [62.3.8](#).

```

(* ===== *)
(* Primo.pas                                           *)
(* ===== *)
program NumeroPrimo;

var    x        : integer;

```

```

(* ===== *)
(* primo( <x> ) *)
(* ----- *)
function primo( x : integer ) : boolean;

var      np      : boolean;
         i       : integer;
         j       : integer;

begin

    np := TRUE;
    i := 2;

    while (i < x) AND np do begin

        j := x / i;
        j := x - (j * i);

        if j = 0 then
            begin
                np := FALSE;
            end
        else
            begin
                i := i+1;
            end
        ;

    end;

    primo := np;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci un numero intero positivo: ' );
    Readln( x );

```

```

if primo( x ) then
  begin
    Writeln( 'E'' un numero primo' );
  end
else
  begin
    Writeln( 'Non e'' un numero primo' );
  end
;

end.
(* ===== *)

```

## Scansione di array

«

In questa sezione vengono mostrati alcuni algoritmi, legati alla scansione degli array, portati in Pascal.

Per semplicità, gli esempi mostrati fanno uso di array dichiarati globalmente, che come tali sono accessibili alle procedure e alle funzioni senza necessità di farne riferimento all'interno delle chiamate.

### Ricerca sequenziale

«

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione [62.4.1](#).

```

(* ===== *)
(* RicercaSeq.pas                               *)
(* Ricerca sequenziale.                         *)
(* ===== *)
program RicercaSequenziale;

const  DIM      = 100;

var    lista    : array[1..DIM] of integer;
       x        : integer;

```



```

    i      : integer;
    z      : integer;

(* ===== *)
(* ricercaseq( <x>, <ele-inf>, <ele-sup> ) *)
(* ----- *)
function ricercaseq( x : integer; a : integer; z : integer ) : integer;

var      i      : integer;

begin

    (* ----- *)
    (* Se l'elemento non viene trovato, il valore -1 segnala *)
    (* l'errore. *)
    (* ----- *)
    ricercaseq := -1;

    (* ----- *)
    (* Scandisce l'array alla ricerca dell'elemento. *)
    (* ----- *)
    for i := a to z do begin

        if x = lista[i] then
            begin
                ricercaseq := i;
            end
        ;

    end;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin

```

```

        z := DIM;
    end
;

Writeln( 'Inserire i valori dell''array' );

for i := 1 to z do begin
    Write( 'elemento ', i:2, ': ' );
    Readln( lista[i] );
end;

Writeln( 'Inserire il valore da cercare' );
Readln( x );

i := ricercaseq( x, 1, z );

Writeln( 'Il valore cercato si trova nell''elemento', i );

end.
(* ===== *)

```

Esiste anche una soluzione ricorsiva che viene mostrata nella subroutine seguente:

```

function ricercaseq( x : integer; a : integer; z : integer ) : integer;

begin

  if a > z then
    begin
      (* ----- *)
      (* La corrispondenza non è stata trovata. *)
      (* ----- *)
      ricercaseq := -1;
    end
  else if x = lista[a] then
    begin
      ricercaseq := a;
    end
  else
    begin
      ricercaseq := ricercaseq( x, a+1, z);
    end
  ;

end;

```

## Ricerca binaria

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione [62.4.2](#).

```

(* ===== *)
(* RicercaBin.pas *)
(* Ricerca binaria. *)
(* ===== *)
program RicercaBinaria;

const  DIM      = 100;

var    lista    : array[1..DIM] of integer;
       x        : integer;
       i        : integer;
       z        : integer;

(* ===== *)
(* ricercabin( <x>, <ele-inf>, <ele-sup> ) *)

```

```

(* ----- *)
function ricercabin( x : integer; a : integer; z : integer ) : integer;

var      m      : integer;

begin

    (* ----- *)
    (* Determina l'elemento centrale.                *)
    (* ----- *)
    m := ( a + z ) / 2;

    if m < a then
        begin

            (* ----- *)
            (* Non restano elementi da controllare.    *)
            (* ----- *)
            ricercabin := -1;
        end
    else if x < lista[m] then
        begin

            (* ----- *)
            (* Si ripete la ricerca nella parte inferiore. *)
            (* ----- *)
            ricercabin := ricercabin( x, a, m-1 );
        end
    else if x > lista[m] then
        begin

            (* ----- *)
            (* Si ripete la ricerca nella parte superiore. *)
            (* ----- *)
            ricercabin := ricercabin( x, m+1, z );
        end
    else
        begin

            (* ----- *)
            (* m rappresenta l'indice dell'elemento cercato. *)
            (* ----- *)
            ricercabin := m;
        end
    end
end

```

```

;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

  Writeln( 'Inserire il numero di elementi.' );
  Writeln( DIM, ' al massimo.' );
  Readln( z );

  if z > DIM then
    begin
      z := DIM;
    end
  ;

  Writeln( 'Inserire i valori dell''array' );

  for i := 1 to z do begin
    Write( 'elemento ', i:2, ': ' );
    Readln( lista[i] );
  end;

  Writeln( 'Inserire il valore da cercare' );
  Readln( x );

  i := ricercabin( x, 1, z );

  Writeln( 'Il valore cercato si trova nell''elemento', i );

end.
(* ===== *)

```

## Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in Pascal. <<

# Bubblesort



Il problema del Bubblesort è stato descritto nella sezione [62.5.1](#). Viene mostrata prima una soluzione iterativa e successivamente la funzione **'bsort'** in versione ricorsiva.

```
(* ===== *)
(* BSort.pas                                     *)
(* ===== *)
program BubbleSort;

const   DIM       = 100;

var     lista     : array[1..DIM] of integer;
        i         : integer;
        z         : integer;

(* ===== *)
(* bsort( <ele-inf>, <ele-sup> )                 *)
(* ----- *)
procedure bsort( a : integer; z : integer );

var     scambio  : integer;
        j         : integer;
        k         : integer;

begin

    (* ----- *)
    (* Inizia il ciclo di scansione dell'array.   *)
    (* ----- *)
    for j := a to ( z-1 ) do begin

        (* ----- *)
        (* Scansione interna dell'array per collocare nella *)
        (* posizione j l'elemento giusto.                *)
        (* ----- *)
        for k := ( j+1 ) to z do begin

            if lista[k] < lista[j] then
                begin

                    (* ----- *)
                    (* Scambia i valori.                *)
                    (* ----- *)
```

```

                (* ----- *)
                scambio := lista[k];
                lista[k] := lista[j];
                lista[j] := scambio;
            end
        ;

    end;
end;

end;

(* ===== *)
(* Inizio del programma.                               *)
(* ----- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    bsort( 1, z );

    Writeln( 'Array ordinato:' );

    for i := 1 to z do begin
        Write( lista[i] );
    end;

end.

```

```
(* ===== *)
```

## Segue la procedura 'bsort' in versione ricorsiva:

```
procedure bsort( a : integer; z : integer );

var      scambio : integer;
        k       : integer;

begin

  if a < z then
    begin

      (* ----- *)
      (* Scansione interna dell'array per collocare nella *)
      (* posizione j l'elemento giusto. *)
      (* ----- *)
      for k := ( a+1 ) to z do begin

        if lista[k] < lista[a] then
          begin

            (* ----- *)
            (* Scambia i valori. *)
            (* ----- *)
            scambio := lista[k];
            lista[k] := lista[a];
            lista[a] := scambio;
          end
        ;

      end;

      bsort( a+1, z );

    end
  ;

end;
```



# Torre di Hanoi



Il problema della torre di Hanoi è descritto nella sezione [62.5.3](#).

```
(* ===== *)
(* Hanoi.pas                                     *)
(* Torre di Hanoi.                             *)
(* ===== *)
program TorreHanoi;

var      n          : integer;
        p1         : integer;
        p2         : integer;

(* ===== *)
(* hanoi( <n>, <p1>, <p2> )                      *)
(* ----- *)
procedure hanoi( n : integer; p1 : integer; p2 : integer );

begin

    if n > 0 then
        begin
            hanoi( n-1, p1, 6-p1-p2 );

            Writeln(
                'Muovi l''anello ', n:1,
                ' dal piolo ', p1:1,
                ' al piolo ', p2:1
            );

            hanoi( n-1, 6-p1-p2, p2 );
        end
    ;

end;

(* ===== *)
(* Inizio del programma.                       *)
(* ----- *)
begin

    Writeln;
    Write( 'Inserisci il numero di anelli: ' );
```

```

Readln( n );
Write( 'Inserisci il piolo iniziale: ' );
Readln( p1 );
Write( 'Inserisci il piolo finale: ' );
Readln( p2 );

hanoi( n, p1, p2 );

end.
(* ===== *)

```

## Quicksort



L'algoritmo del Quicksort è stato descritto nella sezione [62.5.4](#).

```

(* ===== *)
(* QSort.pas *)
(* ===== *)
program QuickSort;

const   DIM      = 100;

var     lista    : array[1..DIM] of integer;
        i       : integer;
        z       : integer;

(* ===== *)
(* part( <ele-inf>, <ele-sup> ) *)
(* ----- *)
function part( a : integer; z : integer ) : integer;

var     scambio : integer;
        i       : integer;
        cf      : integer;
        loop1   : boolean;
        loop2   : boolean;
        loop3   : boolean;

begin

        (* ----- *)
        (* Si assume che a sia inferiore a z. *)
        (* ----- *)

```

```

i := a+1;
cf := z;

(* ----- *)
(* Inizia il ciclo di scansione dell'array.          *)
(* ----- *)
loop1 := TRUE;
while loop1 do begin

    loop2 := TRUE;
    while loop2 do begin

        (* ----- *)
        (* Sposta i a destra.                            *)
        (* ----- *)
        if ( lista[i] > lista[a] ) OR ( i >= cf ) then
            begin
                loop2 := FALSE;
            end
        else
            begin
                i := i+1;
            end
        end
        ;

    end;

    loop3 := TRUE;
    while loop3 do begin

        (* ----- *)
        (* Sposta cf a sinistra.                          *)
        (* ----- *)
        if lista[cf] <= lista[a] then
            begin
                loop3 := FALSE;
            end
        else
            begin
                cf := cf-1;
            end
        end
        ;

    end;

end;

```

```

    if cf <= i then
        begin

            (* ----- *)
            (* è avvenuto l'incontro tra i e cf. *)
            (* ----- *)
            loop1 := FALSE;
        end
    else
        begin

            (* ----- *)
            (* Vengono scambiati i valori. *)
            (* ----- *)
            scambio := lista[cf];
            lista[cf] := lista[i];
            lista[i] := scambio;

            i := i+1;
            cf := cf-1;
        end
    ;
end;

(* ----- *)
(* A questo punto, lista[a..z] è stata ripartita e cf è la *)
(* collocazione finale. *)
(* ----- *)
scambio := lista[cf];
lista[cf] := lista[a];
lista[a] := scambio;

(* ----- *)
(* In questo momento, lista[cf] è un elemento (un valore) nella *)
(* posizione giusta. *)
(* ----- *)
part := cf

end;

(* ===== *)
(* quicksort( <ele-inf>, <ele-sup> ) *)
(* ----- *)

```

```

procedure quicksort( a : integer; z : integer );

var      cf      : integer;

begin

    if z > a then
        begin
            cf := part( a, z );
            quicksort( a, cf-1 );
            quicksort( cf+1, z );
        end
    ;

end;

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    quicksort( 1, z );

    Writeln( 'Array ordinato:' );

    for i := 1 to z do begin
        Write( lista[i] );

```

```

end;

end.
(* ===== *)

```

## Permutazioni

<<

L'algorithmo ricorsivo delle permutazioni è descritto nella sezione [62.5.5](#).

```

(* ===== *)
(* Permuta.pas *)
(* ===== *)
program Permutazioni;

const   DIM      = 100;

var     lista    : array[1..DIM] of integer;
        i        : integer;
        z        : integer;

(* ===== *)
(* permuta( <ele-inf>, <ele-sup>, <elementi-totali> ) *)
(* ----- *)
function permuta( a : integer; z : integer; elementi : integer ) : integer;

var     scambio : integer;
        k        : integer;
        i        : integer;

begin

    (* ----- *)
    (* Se il segmento di array contiene almeno due elementi, *)
    (* si procede. *)
    (* ----- *)
    if ( z-a ) >= 1 then
        begin

            (* ----- *)
            (* Inizia il ciclo di scambi tra l'ultimo elemento e *)
            (* uno degli altri contenuti nel segmento di array. *)

```

```

    (* ----- *)
    k := z;
    while k >= a do begin

        (* ----- *)
        (* Scambia i valori. *)
        (* ----- *)
        scambio := lista[k];
        lista[k] := lista[z];
        lista[z] := scambio;

        (* ----- *)
        (* Esegue una chiamata ricorsiva per permutare un *)
        (* segmento più piccolo dell'array. *)
        (* ----- *)
        permuta( a, z-1, elementi );

        (* ----- *)
        (* Scambia i valori. *)
        (* ----- *)
        scambio := lista[k];
        lista[k] := lista[z];
        lista[z] := scambio;

        k := k-1;

    end;
end
else
begin

    (* ----- *)
    (* Visualizza la situazione attuale dell'array. *)
    (* ----- *)
    for i := 1 to elementi do begin
        Write( lista[i]:4 );
    end;
    Writeln;

end
;

end;

```

```

(* ===== *)
(* Inizio del programma. *)
(* ----- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    permuta( 1, z, z );

end.

(* ===== *)

```