****

UNA TECNICA PER SIMULARE LA RICORSIONE IN COBOL

APPLICATA A TRE ESEMPI:

TORRE DI HANOI

QUICKSORT

PERMUTAZIONI

****

Giacomini Daniele

Treviso, Giugno 1985

LA LETTERATURA SULLE DEFINIZIONI INERENTI LA RICORSIONE

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 - e' significativo solo all'interno di una determinata parte 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, puo' accadere che piu' 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 e' 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.

E' essenziale che i nomi degli oggetti locali non debbano dipendere dall'ambiente della procedura. Ma, in tal modo, puo' 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 pero' e' corretta solo se la grandezza non locale x non e' significativa per P, cioe' 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".

La ricorsione

"La ricorsione" come spiegano Ledgard, Nagin e Hueras {2} "e' un metodo di definizione in cui l'oggetto della definizione e' usato all'interno della definizione. Per esempio si puo' considerare la seguente definizione della parola 'discendente':

> Un discendente di una persona e' il figlio o la figlia di una persona, o un discendete di un figlio o di una figlia.

Quindi, come scrive Lawrie Moore {3}, un sotto-programma ricorsivo "e' un sotto-programma che corrisponde direttamente ed utilizza una definizione ricorsiva". Ovvero, molto piu' semplicemente come dicono Aho, Hopcroft ed 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 'scope', il suo proprio spazio di lavoro, la sua proria esistenza virtuale. (...) Questo processo prende luogo al momento della esecuzione del programma (run-time). Al momento della compilazione ne' la macchina, ne' l'intelligenza umana possono dire quante volte la procedura sara' richiamata al momento della esecuzione. Percio' la creazione di un nuovo blocco di programma al momento della esecuzione e' un processo dinamico. La creazione ricorsiva di nuovi blocchi di programma e' una struttura di programmazione dinamica".

LE PROPRIETA` DEL LINGUAGGIO RICORSIVO

La definizione di procedura ricorsiva data da Aho, Hopcroft ed Ullman e` una condizione necessaria ma non sufficiente perche` un linguaggio di programmazione possa definirsi ricorsivo; infatti e` 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 non vi e` dubbio che il linguaggio COBOL non sia ricorsivo, eppure ammette che all'interno di un paragrafo si faccia la chiamata dello stesso paragrafo tramite il verbo PERFORM. In effetti non si parla di ricorsione proprio perche` il COBOL gestisce solo variabili 'globali'.

DESCRIZIONE DELLA TECNICA PER SIMULARE LA RICORSIONE IN COBOL

Introduzione

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 sottopramma ma non escano, che i dati escano soltanto oppure che i dati debbano prima entrare e poi uscire modificati.

La pseudocodifica utilizzata

La pseudocodifica utilizzata per mostrare gli esempi, prima di presentare la trasformazione in COBOL, si rifa` al linguaggio MPLII Burroughs (Algol-like) dove le variabili di scambio di una procedura vengono semplicemente nominate a fianco del nome della procedura tra parentesi. Cio` corrisponde ad una dichiarazione implicita di quelle variabili con 'scope' locale e con caratteristiche identiche a quelle usate nelle chiamate relative. In particolare se nella chiamata vengono usate costanti alfanumeriche, la variabile corrispondente sara` di tipo alfanumerico di lunghezza pari alla costante trasmittente, se di tipo numerico, la variabile corrispondente sara` di tipo numerico opportuno: 'integer' o 'float'.

Quindi in questo tipo di pseudocodifica non sono permesse le variabili di scambio in Output.

Le variabili di cambio di questa pseudocodifica si collegano per posizione.

La simulazione con un linguaggio avente solo variabili globali

Il problema della simulazione della ricorsione si risolve utilizzando uno STACK per ogni variabile locale.

La tecnica e` indicata molto semplicemente da Jerrold L. Wagener {5}. Una volta determinato a priori qual'e` il numero massimo di livelli della ricorsione occorre associare ad ogni variabile, locale che non sia collegata con l'esterno in Input-Output, uno stack con dimensione pari a quel numero. Quindi, ad una varabile scalare viene associato un vettore, ad un vettore viene associata una matrice, e cosi` di seguito. L'indice dello stack (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 nei vari stacks dei valori delle variabili di scambio che non sono in Input-Output con una operazione di 'push'; la seconda deve essere effettuata subito dopo la chiamata ricorsiva e consiste nel recupero dai vari stacks dei valori originari delle variabili con una operazione di 'pop'.

Esempio:

Procedura ricorsiva

```
PROCEDURE PROC1 (V, G, Z)
     .
     .
%  commento
     .
     .
     .
     .
%      G e` una variabile in
%      Input-Output
     PROC1 (V, G, Z-1)
     .
     .
     .
     .
     .
     .
     .
END PROC1
```

Trasformazione non ricorsiva

```
 PROCEDURE PROC1
      .
      .
      .
%       push
      SP := SP + 1
      SAVEV(SP) := V
      SAVEZ(SP) := Z
%       call
      Z := Z - 1
      PROC1
%       pop
      V := SAVEV(SP)
      Z := SAVEZ(SP)
      SP := SP - 1
      .
      .
      .
 END PROC1
```

Tengo a precisare che la tecnica e` 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).

ESEMPI

Torre di Hanoi

Descrizione dell'algoritmo; Wagener (5)

"Su una tavola sono infissi tre pioli identici, ed uno contiene N anelli, tutti di diametro differente (ogni anello ha un buco al suo centro in modo da poter essere sopostato da un piolo ad un'altro). L'obbiettivo e` quello di spostare tutti gli anelli su di uno degli altri due pioli, osservando semplicemente le due regole seguenti:

1. Si puo` muovere solo un anello alla volta.
2. Non si puo` mai mettere un anello al di sopra di uno piu` piccolo sullo stesso piolo.

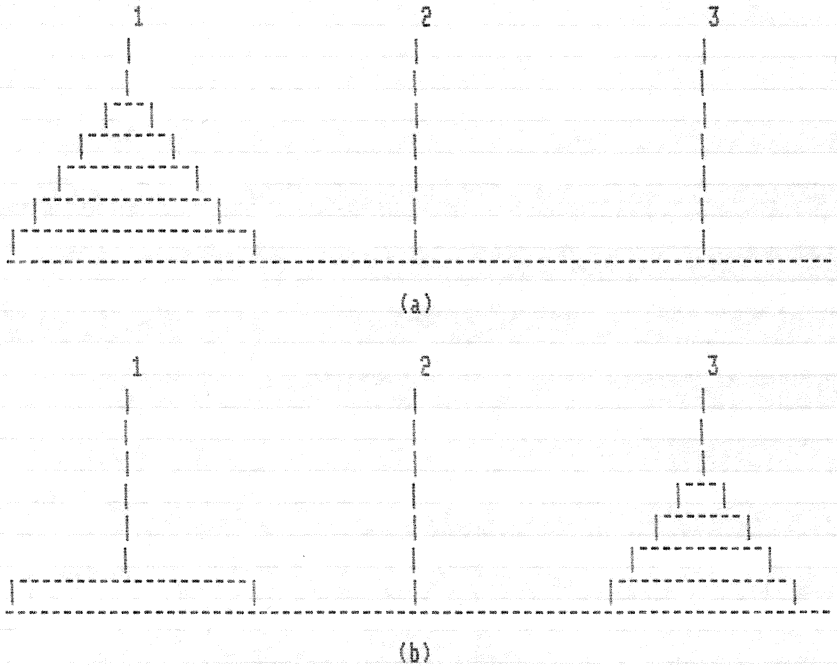


Figura 1. Torre di Hanoi, con N anelli.
(a) Configurazione originale.
(b) Dopo aver spostato N-1 anelli.

Nella figura 1a gli anelli appaiono sul piolo 1; si supponga che essi debbano essere spostati sul piolo 2. Si supponga che gli N-1 anelli superiori vengano spostati in qualche modo (osservando le regole naturalmente) dal piolo 1 al piolo 3, come illustrato in figura 1b. Allora l'N-esimo anello (il piu` largo) puo` essere spostato nel piolo 2, e la pila di N-1 anelli che ora si trova sul piolo 3 puo` essere spostata ancora, questa volta sul piolo 2 al di sopra dell'anello piu` largo che e` appena stato messo li`. Indichiamo con HANOI questo algoritmo. (...).

Se N e` minore di 1 HANOI non fa nulla come dovrebbe dal momento

che cio` significa che non ci sono anelli da muovere. Se N e` 1, le istruzioni contenute nel predicato IF-FI vengono eseguite, ma nessuna delle chiamate ricorsive fa nulla dato che N-1 e` zero. Pertanto in questo caso (N uguale ad 1, e assumendo P1 uguale ad 1 e P2 uguale a 2) il risultato e` semplicemente

MUOVI L'ANELLO 1 DAL PIOLO 1 AL PIOLO 2

che e` corretto per uno stack iniziale consistente di un solo anello. Ora si puo` vedere che l'algoritmo funziona correttamente se N e` uguale a 2. La prima chiamata ricorsiva sposta un anello (N-1 uguale ad 1) dal piolo 1 al piolo 3 (ancora assumendo P1 uguale ad 1 e P2 uguale a 2) e si sa che questa e` la mossa corretta. Viene eseguito l'istruzione di stampa (spostamento dell'anello 2 sul piolo 2). L'output consiste di 3 linee:

MUOVI L'ANELLO 1 DAL PIOLO 1 AL PIOLO 3
MUOVI L'ANELLO 2 DAL PIOLO 1 AL PIOLO 2
MUOVI L'ANELLO 1 DAL PIOLO 3 AL PIOLO 2

Ora che si sa che HANOI funziona correttamente per una pila di due anelli si puo` usare una analisi simile per mostrare che funziona per tre anelli, quindi quatto, cinque, e cosi` di seguito, per qualsiasi valore di N.

Variabili:

N e` la dimensione della torre in numero di anelli: gli anelli sono numerati da 1 a N;

P1 e` il numero del piolo su cui si trova inizialmente la pila di N anelli;

P2 e` il numero del piolo su cui deve essere spostata la pila di anelli;

6-P1-P2 = P3 e` il numero dell'altro piolo (e` evidente essendo i pioli numerati 1, 2 e 3).

Pseudocodifica ricorsiva di HANOI:

```
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)
     FI
END HANOI
```

Codifica in linguaggio MPL II:

```
PROCEDURE MAIN;

$ LIBRARY CONVERSION
$ LIBRARY DECIMAL
$ LIBRARY STRING
$ LIBRARY LM.JR
$ LIBRARY LM.FX
$ LIBRARY LM.IN.N
$ LIBRARY LM.OUT.N

%----------------------------
  DECLARE RECORD.PR.FILE CHARACTER (132);
  REMAP   RECORD.PR.FILE: RECORD.PR.CH(132)  CHARACTER(1);

  FILE PRINTER.FILE WORK.AREA RECORD.PR.FILE;
%----------------------------
    DEVICE     (PRINTER.FILE) := PR;              % ANY PRINTER DEVICE
    RECORD     (PRINTER.FILE) := 132;
    BUFFER     (PRINTER.FILE) := 132;
    NO.BUFFERS (PRINTER.FILE) := 1;
    NO.LABEL   (PRINTER.FILE) := 1;
    CLOSEMODE  (PRINTER.FILE) := RELEASE;
    MYUSE      (PRINTER.FILE) := OUTPUT;
%----------------------------

  PROCEDURE PRINT.MESSAGE (N, P1, P2);
    DECLARE N.CH     CHARACTER (6);
    DECLARE P1.CH    CHARACTER (6);
    DECLARE P2.CH    CHARACTER (6);
    CONVERT (1, N.CH, N);
    CONVERT (1, P1.CH, P1);
    CONVERT (1, P2.CH, P2);
    SMEAR (RECORD.PR.CH, 0, 132, " ", 132);
    SUBSTR (RECORD.PR.FILE, 0, 14) := "MUOVE L'ANELLO";
    SUBSTR (RECORD.PR.FILE, 15, 6) := N.CH;
    SUBSTR (RECORD.PR.FILE, 21, 10) := " DAL POSTO";
    SUBSTR (RECORD.PR.FILE, 32, 6) := P1.CH;
    SUBSTR (RECORD.PR.FILE, 39, 9) := " AL POSTO";
    SUBSTR (RECORD.PR.FILE, 49, 6) := P2.CH;

    WRITE (PRINTER.FILE);
  END PRINT.MESSAGE;
```

```
 PROCEDURE HANOI (N, P1, P2);

     IF N > 0
       THEN
          DO;

            HANOI (N-1, P1, 6-P1-P2);
          % scrivi: "MUOVI L'ANELLO " N "DAL POSTO " P1 "AL POSTO " P2
            PRINT.MESSAGE (N, P1, P2);


            HANOI (N-1, 6-P1-P2, P2);
          END;
 END HANOI;

%-------------------------

  DECLARE N  FIXED;
  DECLARE P1 FIXED;
  DECLARE P2 FIXED;

  OPEN (PRINTER.FILE);

  DISPLAY (" INSERISCI LA DIMENSIONE DELLA TORRE ");
  INPUT.N (N);

  DISPLAY (" INSERISCI IL POSTO ATTUALE DELLA TORRE: 1, 2, 3 ");
  INPUT.N (P1);

  DISPLAY (" INSERISCI IL POSTO IN CUI VUOI PORTARE LA TORRE: 1, 2, 3");
  INPUT.N (P2);

  HANOI (N, P1, P2);

  CLOSE (PRINTER.FILE);

  STOP;
END MAIN;
FINI;
```

Variabili:

SAVEN e` il vettore utilizzato per conservare il valore di N;

SAVEP1 e` il vettore utilizzato per conservare il valore di P1;

SAVEP2 e` il vettore utilizzato per conservare il valore di P2;

SP e` l'indice dei vettori su indicati (stack pointer).

Pseudocodifica che simula la ricorsione:

```
HANOI

     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-1, 6-P1-P2, P2)
          N := SAVEN(SP)
          P1 := SAVEP1(SP)
          SP := SP - 1
     FI
END HANOI
```

Codifica COBOL:

```
000100*************************************************************
000200* PROGRAMMA COBOL HC.04                                     *
000300*                                                           *
000400* DATA:              18/08/84  (GG/MM/AA)                   *
000500*                                                           *
000600* ARGOMENTO: SIMULAZIONE DI RICORSIONE CON TORRE DI HANOI   *
001100*************************************************************
006400
006500
006600 IDENTIFICATION DIVISION.
006700
006800
006900 ENVIRONMENT DIVISION.
007000
007100
007200 DATA DIVISION.
007300
007400
007500 WORKING-STORAGE SECTION.
007600
007700 01  RECORD-STACKS.
007800     02  SAVEN  OCCURS 100 TIMES PIC 99.
007900     02  SAVEP1 OCCURS 100 TIMES PIC 9.
008000     02  SAVEP2 OCCURS 100 TIMES PIC 9.
008100
008200 01  STACK-POINTER.
008300     02  SP                      PIC 99 VALUE 0.
008400
008500 01  VARIABILI-SCALARI.
008600     02  N                       PIC 99.
008700     02  P1                      PIC 9.
008800     02  P2                      PIC 9.
008900
009000
```

```
009100 PROCEDURE DIVISION.
009200
009300 MAIN.
009400
009500     DISPLAY "INSERISCI LA DIMENSIONE DELLA TORRE".
009600     DISPLAY "(DUE CARATTERI)".
009700     ACCEPT N.
009750
009800     DISPLAY "INSERISCI LA POSIZIONE INIZIALE DELLA TORRE".
009900     DISPLAY "(UN CARATTERE: 1 O 2 O 3)".
010000     ACCEPT P1.
010050
010100     DISPLAY "INSERISCI LA DESTINAZIONE DELLA TORRE".
010200     DISPLAY "(UN CARATTERE)".
010300     ACCEPT P2.
010400
010500     PERFORM HANOI.
010600
010700     STOP RUN.
010800
```

```
010900 HANOI.
011000
011100     IF N > 0
011200       THEN
011300*        push per conservare le variabili di scambio
011400          COMPUTE SP = SP + 1,
011500          COMPUTE SAVEN(SP) = N,
011600          COMPUTE SAVEP2(SP) = P2,
011700*        cambiamenti alle variabili di scambio prima della CALL
011800          COMPUTE N = N - 1,
011900          COMPUTE P2 = 6 - P1 - P2,
012000*        call
012100          PERFORM HANOI,
012200*        pop per recuperare i valori delle variabili di scambio
012300          COMPUTE N = SAVEN(SP),
012400          COMPUTE P2 = SAVEP2(SP),
012500          COMPUTE SP = SP - 1,
012600
012700          DISPLAY "MUOVI L'ANELLO " N " DAL PIOLO " P1
012800                  " AL PIOLO " P2,
012850
012900*        push per conservare i valori delle variabili di scambio
013000          COMPUTE SP = SP + 1,
013100          COMPUTE SAVEN(SP) = N,
013200          COMPUTE SAVEP1(SP) = P1,
013300*        modifica dei valori delle variabili di scambio
013400          COMPUTE N = N - 1,
013500          COMPUTE P1 = 6 - P1 - P2,
013600*        call
013700          PERFORM HANOI,
013800*        pop per recuperare i valori delle variabili di scambio
013900          COMPUTE N = SAVEN(SP),
014000          COMPUTE P1 = SAVEP1(SP),
014100          COMPUTE SP = SP - 1.
014200
```

Bibliografia:

Wagener J. L.: "FORTRAN 77 Principles of Programming" {5}
Wiley, 1980, pag. 227.

Moore L.: "Foundations of Programming with Pascal" {3}
Ellis Horwood Limited, 1980, pp. 177-180.

Arsac J.: "La construction de programmes structures"
Dunod, 1977, capitolo 15.

Quicksort (ordinamento non decrescente)

Descrizione dell'algoritmo; Wagener [5]

"Gli elementi di base del 'quicksort' sono schematizzati in figura 2.

TABLE (vettore da ordinare)

```
---- \   partizione dei valori
---- >   minori del 'valore
---- /   allocato'

----      valore allocato

---- \   partizione dei valori
---- >   maggiori del
---- /   'valore allocato'
```

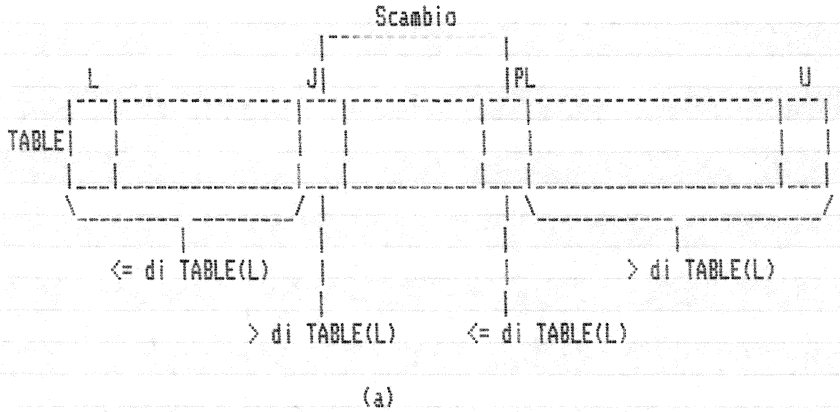
Figura 2. Il concetto base dell'algoritmo del quicksort: partizione del vettore in due gruppi disordinati separati da un valore piazzato correttamente nel suo posto rispetto all'ordinamento.

Un passo attraverso il vettore da ordinare e` sufficiente per piazzare un valore del vettore (diciamo il primo) nella sua posizione finale nel vettore ordinato, ed allo stesso tempo per lasciare tutti i valori minori di questo da una parte (ma disordinatamente) e tutti i valori maggiori dall'altra. Allora delle chiamate ricorsive possono essere usate per ordinare ogn'una di queste due 'partizioni'. (...). Pertanto l'algoritmo essenziale del quicksort e`:

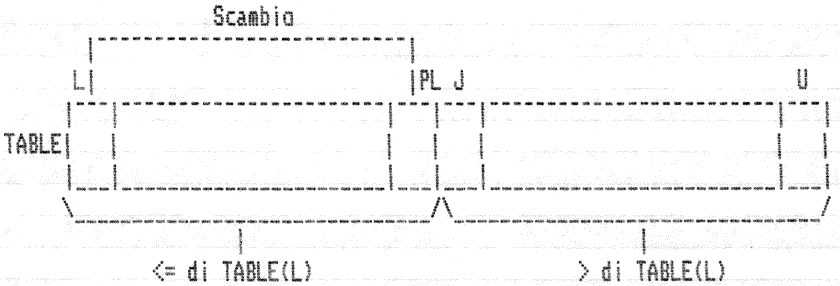
1. Localizzare la posizione (finale) del primo valore, dividendo in questo modo i valori.
2. Ordinare la prima partizione.
3. Ordinare la seconda partizione.

Indichiamo con PARTIT il modulo (subroutine) che esegue il passo 1 - esso determina la posizione propria, PL (Proper Location), dell'elemento TABLE(L) all'interno del vettore TABLE(L:U), mette questo valore in questa posizione (cioe` viene scambiato il valore di TABLE(PL) con quello di TABLE(L)), mette tutti i valori minori di TABLE(PL) nel segmento di vettore TABLE(L:PL-1), e mette tutti i valori maggiori di TABLE(PL) in TABLE(PL+1:U). Indichiamo quindi con QSORT l'algoritmo del quicksort. Assumendo che PARTIT e le chiamate ricorsive di QSORT facciano il loro lavoro correttamente, una analisi informale della correttezza di QSORT potrebbe procedere come segue. Se U non e` maggiore di L allora c'e` solo un elemento (o nessuno) in TABLE(L:U), ed inoltre TABLE(L:U) e` gia` nel suo stato finale. Se U e` maggiore di L allora (per assunzione) PARTIT ripartisce correttamente TABLE(L:U). L'ordinamento separato delle due partizioni (per assunzione eseguito correttamente dalle chiamate ricorsive) completa l'ordinamento di TABLE(L:U).

(...). La figura 3 mostra graficamente la funzione di PARTIT, la parte (a) prima che MAINLOOP sia terminato (MAINLOOP e` il nome di un loop contenuto in PARTIT), e la parte (b) dopo che MAINLOOP e` terminato.



(a)



(b)

Figura 3. La funzione della subroutine PARTIT. (a) Prima del termine di MAINLOOP. (b) Dopo il termine di MAINLOOP.

Praticamente, J, iniziando dal valore L+1, viene 'slittato' verso destra fino a che viene trovato un elemento di TABLE maggiore di TABLE(L), quindi PL viene 'slittato' verso sinistra, iniziando da U, fino a che viene trovato un elemento di TABLE minore o uguale di TABLE(L). Questi valori vengono scambiati e lo slittamento riprende. Cio` continua fino a quando J e PL di 'incontrano', e in quel momento TABLE e` stata ripartita, come mostrato in figura 3b, e PL e` la locazione per il valore di TABLE(L)."

Variabili:

TABLE e` il vettore da ordinare;

L e` l'indice inferiore del segmento di vettore da ordinare;

U e` l'indice superiore del segmento di vettore da ordinare;

PL (proper location) e` l'indice che cerca e trova la giusta posizione di TABLE(L) nel vettore;

Pseudocodifica ricorsiva di QSORT:

```
QSORT (TABLE, L, U)
    IF U > L
      THEN
          VAR PL INTEGER
          PARTIT (TABLE, L, U, PL)
          QSORT  (TABLE, L, PL-1)
          QSORT  (TABLE, PL+1, U)
    FI
END QSORT
```

Variabili:

J e` l'indice che insieme a PL serve a ripartire il vettore TABLE.

Pseudocodifica di PARTIT:

```
PARTIT(TABLE, L, U, PL)

     VAR J INTEGER

     % si assume che L < U

     J := L + 1
     PL := U

     DO MAINLOOP FOREVER
       DO FOREVER  % ...sposta J a destra...
         IF TABLE(J) > TABLE(L) OR J >= PL
           THEN
              EXIT
           ELSE
              J := J + 1
         FI
       END
       DO FOREVER  % ...sposta PL a sinistra...
         IF TABLE(PL) <= TABLE(L)
           THEN
              EXIT
           ELSE
              PL := PL - 1
         FI
       END
       % ...ora esce dal MAINLOOP se TABLE e` completamente ripartita..
       IF PL <= J
         THEN
            EXIT
         ELSE
            %   scambio dei valori tra table(pl) e table(j)
            TABLE(PL) :==: TABLE(J)
            J := J + 1
            PL := PL - 1
       FI
     OD MAINLOOP
     % ...ora TABLE e` stata ripartita, PL e` il posto di TABLE(L)...
     TABLE(PL) :==: TABLE(L)
     % ...ora TABLE(PL) e` nella sua giusta posizione...
END PARTIT
```

Codifica in linguaggio MPL II:

```
PROCEDURE MAIN;


$ LIBRARY CONVERSION
$ LIBRARY DECIMAL
$ LIBRARY STRING
$ LIBRARY LM.JR
$ LIBRARY LM.FX
$ LIBRARY LM.IN.N
$ LIBRARY LM.OUT.N
$ LIBRARY LM.IN.MN
$ LIBRARY LM.OUT.MN
$ LIBRARY LM.XCH


  PROCEDURE PARTIT(TABLE, L, U, PL);


      DECLARE     J FIXED;

      % si assume che L < U

      J := L + 1;
      PL := U;
```

```
    DO MAINLOOP FOREVER;
      DO FOREVER; % ...sposta J a destra...
        IF TABLE((J-1)*2) > TABLE((L-1)*2) OR J >= PL
          THEN
             UNDO;
          ELSE
             J := J + 1;
      END;
      DO FOREVER; % ...sposta PL a sinistra...
        IF TABLE((PL-1)*2) <= TABLE((L-1)*2)
          THEN
             UNDO;
          ELSE
             PL := PL - 1;
      END;
      % ...ora esce dal MAINLOOP se TABLE e' completamente ripartita..
      IF PL <= J
        THEN
           UNDO;
        ELSE
           DO;
             SCAMBIO(TABLE((PL-1)*2), TABLE((J-1)*2));
             J := J + 1;
             PL := PL - 1;
           END;
    END MAINLOOP;
    % ...ora TABLE e' stata ripartita, PL e' il posto di TABLE(L)...
    SCAMBIO (TABLE((PL-1)*2), TABLE((L-1)*2));
    % ...ora TABLE(PL) e' nella sua giusta posizione...


END PARTIT;
```

```
  PROCEDURE QSORT (TABLE, L, U);
    DECLARE PL FIXED;
    IF U > L
      THEN
          DO;
            PARTIT (TABLE, L, U, PL);
            QSORT  (TABLE, L, PL-1);
            QSORT  (TABLE, PL+1, U);
          END;
  END QSORT;

%----------------------------
  DECLARE P                    FIXED;
  DECLARE DIM.TOT              FIXED;

  DISPLAY ("INSERISCI LA DIMENSIONE DEL VETTORE");
  INPUT.N (P);

  DIM.TOT := P * 2;
  DECLARE 1 VETTORE.RECORD     CHARACTER(DIM.TOT),
          2 VETTORE            FIXED;

  IN.MAT.N.1 (VETTORE, P);
  OUT.MAT.N.1 (VETTORE, P);

  QSORT (VETTORE, 1, P);

  OUT.MAT.N.1 (VETTORE, P);

  STOP;
END MAIN;
FINI;
```

Variabili:

SAVEL e` il vettore utilizzato per conservare il valore di L;

SAVEU e` il vettore utilizzato per conservare il valore di U;

SP e` lo stack pointer.

Pseudocodifica non ricorsiva di QSORT:

```
QSORT
    IF U > L
      THEN
          PARTIT
          SP := SP + 1
          SAVEU(SP) := U
          U := PL - 1
          QSORT
          U := SAVEU(SP)
%         SP := SP - 1
%         SP := SP + 1
          SAVEL(SP) := L
          L := PL + 1
          QSORT
          L := SAVEL(SP)
          SP := SP - 1
    FI
END QSORT
```

Codifica COBOL:

```
000300*************************************************************************
000400* PROGRAMMA COBOL HC.06                                                 *
000500*                                                                       *
000600* DATA:              22/08/84  (GG/MM/AA)                               *
000700*                                                                       *
000800* ARGOMENTO: SIMULAZIONE DI RICORSIONE CON QUICKSORT                    *
000900*                                                                       *
001000*************************************************************************
009100
009200
009300
009400 IDENTIFICATION DIVISION.
009500
009600
009700 ENVIRONMENT DIVISION.
009800
009900
010000 DATA DIVISION.
010100
010200
010300 WORKING-STORAGE SECTION.
010400
010500 01  RECORD-STACKS.
010600     02  SAVEL   OCCURS 100 TIMES PIC 999.
010700     02  SAVEU   OCCURS 100 TIMES PIC 999.
010800
010900 01  STACK-POINTERS.
011000     02  SP                       PIC 999.
011100
011200 01  VARIABILI-SCALARI.
011300     02  PL                       PIC 999.
011400     02  L                        PIC 999.
011500     02  U                        PIC 999.
011600     02  TEMP                     PIC X(15).
011700     02  J                        PIC 999.
011800     02  I                        PIC 999.
011900
012000 01  RECORD-TABELLA.
012100     02  TABELLA  OCCURS 100  TIMES PIC X(15).
012200
012300
```

```
012400 PROCEDURE DIVISION.
012500
012600 MAIN.
012700
012800     DISPLAY "INSERISCI IL NUMERO DI ELEMENTI DA ORDINARE".
012900     DISPLAY "(TRE CIFRE)".
013000     ACCEPT U.
013100     IF U > 100
013200       THEN
013300           STOP RUN.
013400
013500     COMPUTE L = 1.
013600
013700     PERFORM INSERIMENTO-ELEMENTI VARYING I FROM 1 BY 1
013800                                  UNTIL I > U.
013900
014000     PERFORM QSORT.
014100
014200     PERFORM OUTPUT-DATI VARYING I FROM 1 BY 1
014300                         UNTIL   I > U.
014400
014500     STOP RUN.
014600
014700
014800 INSERIMENTO-ELEMENTI.
014900
015000     DISPLAY "INSERISCI L'ELEMENTO ", I, " DELLA TABELLA".
015100     ACCEPT TABELLA(I).
015200
015300
015400 PARTIT.
015500
015600*      si assume che L < U
015700
015800     COMPUTE J = L + 1.
015900     COMPUTE PL = U.
016000
016100     PERFORM PARTIT-TESTA-MAINLOOP.
016200     PERFORM PARTIT-MAINLOOP UNTIL PL < J
016300                                OR PL = J.
016400
016500     MOVE TABELLA(PL) TO TEMP.
016600     MOVE TABELLA(L)  TO TABELLA(PL).
016700     MOVE TEMP        TO TABELLA(L).
016800
016900
```

```
017000 PARTIT-TESTA-MAINLOOP.
017100
017200     PERFORM SPOSTA-J-A-DESTRA UNTIL TABELLA(J) > TABELLA(L)
017300                                  OR       J > PL
017400                                  OR       J = PL.
017500
017600     PERFORM SPOSTA-PL-A-SINISTRA UNTIL TABELLA(PL) < TABELLA(L)
017700                                     OR TABELLA(PL) = TABELLA(L).
017800
017900
018000 PARTIT-MAINLOOP.
018100
018200     MOVE TABELLA(PL) TO TEMP.
018300     MOVE TABELLA(J)  TO TABELLA(PL).
018400     MOVE TEMP     TO TABELLA(J).
018500
018600     COMPUTE J = J + 1.
018700     COMPUTE PL = PL - 1.
018800
018900     PERFORM SPOSTA-J-A-DESTRA UNTIL TABELLA(J) > TABELLA(L)
019000                                  OR       J > PL
019100                                  OR       J = PL.
019200
019300     PERFORM SPOSTA-PL-A-SINISTRA UNTIL TABELLA(PL) < TABELLA(L)
019400                                     OR TABELLA(PL) = TABELLA(L).
019500
019600
019700 SPOSTA-J-A-DESTRA.
019800
019900     COMPUTE J = J + 1.
020000
020100
020200 SPOSTA-PL-A-SINISTRA.
020300
020400     COMPUTE PL = PL - 1.
020500
020600
```

```
020700 QSORT.
020800
020900     IF U > L
021000       THEN
021100*           le variabili che riguardano PARTIT sono tutte in I-O
021200           PERFORM PARTIT,
021300*            push
021400           COMPUTE SP = SP + 1,
021500           COMPUTE SAVEU(SP) = U,
021530*            cambiamenti alle variabili di scambio
021600           COMPUTE U = PL - 1,
021650*            call
021700           PERFORM QSORT,
021750*            pop
021800           COMPUTE U = SAVEU(SP),
021900*          compute SP = SP - 1,
021950*            push
022000*          compute SP = SP + 1,
022100           COMPUTE SAVEL(SP) = L,
022110*            cambiamenti alle variabili di scambio
022200           COMPUTE L = PL + 1,
022210*            call
022300           PERFORM QSORT,
022350*            pop
022400           COMPUTE L = SAVEL(SP),
022500           COMPUTE SP = SP - 1.
022600
022700
022800 OUTPUT-DATI.
022900
023000     DISPLAY "TABELLA(", I ") = ", TABELLA(I).
023100
```

Bibliografia:

Wagener J. L.: "FORTRAN 77 Principles of Programming" {5}
Wiley, 1980, pag. 225.

Wirth N.: "Algorithms + Data Structures = Programs"
Prentice-Hall, 1976, pp. 76-82.

Knuth D. E.: "The Art of Computer Programming - Volume 3 / Sorting and Searching"
Addison-Wesley, 1973, pp. 114-123.

Aho A. V., Hopcroft J. E., Ullman J. D.: "The Design and Analysis of Computer Algotithms"
Addison-Wesley, 1974, pp. 92-97.

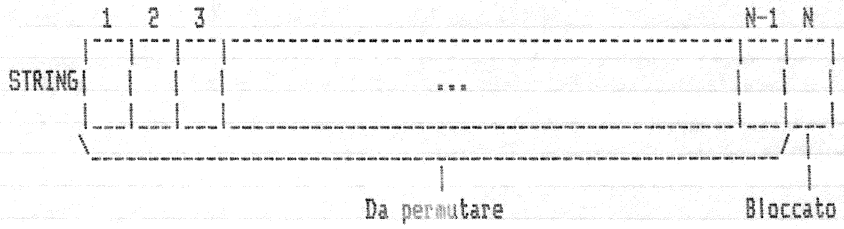
Permutazioni

Descrizione dell'algoritmo; Wagener [5]

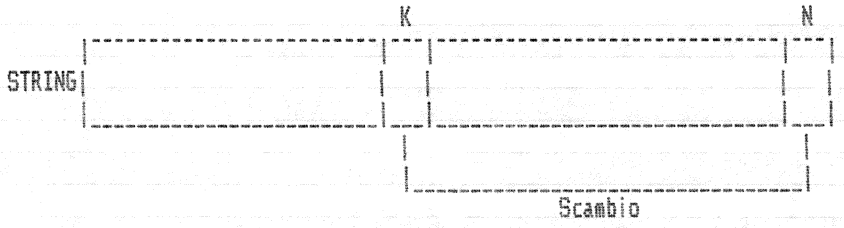
"Il problema e` quello di generare tutte le possibili permutazioni dei caratteri di una stringa. Per esempio la stringa 'PQR' ha sei permutazioni:

PQR
QPR
PRQ
RPQ
RQP
QRP

(...). La figura 4 illustra l'idea generale. La figura 4a mostra che se ci sono N caratteri in una stringa allora alcune delle permutazioni vengono ottenute tenendo 'fisso' l'N-esimo caratere e generando tutte le permutazioni dei primi N-1 caratteri. La figura 4b mostra che allora l'N-esimo carattere puo` essere scambiato con uno dei primi N-1 caratteri, ed allora (a) viene ripetuto. Cio` continua finche` ognuno degli N caratteri originali e` stato usato nella posizione N-esima.



(a)



(b)

Figura 4. Uso della ricorsione per generare tutte le permutazioni di N caratteri. (a) Permutazione dei primi N-1 caratteri, trattenendo l'N-esimo catattere. (b) Scambio dell'N-esimo carattere con il K-esimo.

L'algoritmo risolutivo viene indicato con il nome PERMUT. L'esecuzione dell'algorimo PERMUT da` come riultato la stampa di tutte le possibili permutazioni degli N caratteri. (...). Si supponga che inizialmente N sia uguale ad 1 - allora c'e` solo una permutazione (il carattere stes-

so), ed in tale caso PERMUT stampa semplicemente la stringa e termina, pertanto PERMUT funziona correttamente se N e` uguale ad 1. Per N uguale a 2 ci sono due permutazioni, e PERMUT le stampa entrambe. Per N uguale a 2 il corpo del loop DO-VARYING viene eseguito due volte, la prima per K uguale a 2 e quindi per K uguale ad 1. Ogni volta PERMUT viene chiamato ricorsivamente con N-1 uguale ad 1, e ne risulta semplicemente una stringa stampata. La prima volta i due caratteri sono nel loro ordine originario ('XY') e la seconda volta essi vengono scambiati ('YX'). E` opportuno notare che il secondo scambio riporta la stringa nel suo ordine originario ('XY') al termine dell'algoritmo. Pertanto PERMUT funziona correttamente per N uguale a 2. Per N uguale a 3 il loop DO-VARYING viene eseguito tre volte ogni volta con N-1 uguale a 2 nella chiamata ricorsiva. Dato che da una chiamata di PERMUT con 2 caratteri risultano correttamente stampate due permutazioni, da PERMUT con N uguale a 3 risultano stampate 3*2, ovvero 6 permutazioni. Questo e` il numero esatto delle possibili permutazioni per N uguale a 3, e dato che per ognuna delle tre esecuzioni delle chiamate ricorsive c'e` un carattere diverso nella terza posizione tutte e sei le stringhe stampate sono permutazioni differenti (e di nuovo, alla fine, la stringa viene lasciata nel suo stato originario). Continuando l'analisi in questa maniera, si puo` vedere PERMUT lavorare correttamente per qualsiasi valore di N".

Variabili:

N — e` il numero di elementi da permutare;

STRING — e` il vettore che contiene gli elementi da permutare;

K — e` l'indice che serve a scambiare l'N-esimo elemento.

Procedura PERMUT ricorsiva:

```
PERMUT (STRING, N);

   VAR K INTEGER

   IF N > 1
     THEN
        DO VARYING K FORM N BY -1 WHILE K >= 1
          STRING (K) :==: STRING (N)
          PERMUT (STRING, N-1)
          STRING (K) :==: STRING (N)
        OD
     ELSE
        scrittura STRING
   FI

END PERMUT
```

Codifica in linguaggio MPL II:

```
PROCEDURE MAIN;

$ LIBRARY CONVERSION
$ LIBRARY DECIMAL
$ LIBRARY STRING
$ LIBRARY LM.JR
$ LIBRARY LM.FX
$ LIBRARY LM.IN.N
$ LIBRARY LM.OUT.N
$ LIBRARY LM.XCH

   DECLARE 1 RECORD.STAMPA CHARACTER (10),
           2 STAMPA.CH   CHARACTER (1);

   FILE LISTA WORK.AREA RECORD.STAMPA;

   DECLARE J              FIXED;

   PROCEDURE PERMUT (STRING, N);

      DECLARE K FIXED;

      IF N > 1
        THEN
          DO;
            K := N;
            DO FOREVER;
              IF K < 1
                THEN UNDO;
              SCAMBIO (STRING (K-1), STRING (N-1));
              PERMUT (STRING, N-1);
              SCAMBIO (STRING (K-1), STRING (N-1));
              K := K - 1;
            END;
          END;
       ELSE
          DO;
            J := 0;
            DO FOREVER;
              IF J >= 10
                THEN UNDO;
              STAMPA.CH(J-1+1) := STRING(J-1+1);
              J := J + 1;
            END;
            WRITE (LISTA);
          END;

   END PERMUT;
```

```
    DECLARE 1 RECORD.STRING  CHARACTER (10),
              2 STRING       CHARACTER (1);

    DECLARE N                FIXED;

    DISPLAY ("INSERISCI IL NUMERO DI ELEMENTI");
    INPUT.N (N);

    IF N > 0 AND N <= 10
      THEN
          DO;
            J := 0;

            DO FOREVER;
              IF J >= N
                THEN UNDO;
              CONVERT (0, STRING(J-1+1), J);
              J := J + 1;
            END;

            OPEN (LISTA);

            PERMUT (STRING, N);

            CLOSE (LISTA);

          END;

END MAIN;
FINI;
%----------------------------
    DEVICE     (LISTA) := PR;
    RECORD     (LISTA) := 10;
    BUFFER     (LISTA) := 10;
    NO.BUFFERS (LISTA) := 1;
    NO.LABEL   (LISTA) := 1;
    CLOSEMODE  (LISTA) := RELEASE;
    ACCESSMODE (LISTA) := SEQUENTIAL;
    MYUSE      (LISTA) := OUTPUT;
%----------------------------
```

Pseudocodifica ricorsiva modificata in modo da sostituire il ciclo enumerativo DO-VARYING con una ricorsione:

```
PERMUT (STRING, N)

   SCAMBIO.CHIAMATA.SCAMBIO (STRING, N, K)

      IF K > 0
        THEN
           STRING (K) :==: STRING (N)
           PERMUT (STRING, N-1)
           STRING (K) :==: STRING (N)
           SCAMBIO.CHIAMATA.SCAMBIO (STRING, N, K-1)
      FI

   END SCAMBIO.CHIAMATA.SCAMBIO

   IF N > 1
     THEN
        SCAMBIO.CHIAMATA.SCAMBIO (STRING, N, N)
     ELSE
        scrittura STRING
   FI

END PERMUT
```

Codifica in linguaggio MPL II:

```
PROCEDURE MAIN;

$ LIBRARY CONVERSION
$ LIBRARY DECIMAL
$ LIBRARY STRING
$ LIBRARY LM.JR
$ LIBRARY LM.FX
$ LIBRARY LM.IN.N
$ LIBRARY LM.OUT.N
$ LIBRARY LM.XCH

   DECLARE 1 RECORD.STAMPA CHARACTER (10),
             2 STAMPA.CH   CHARACTER (1);

   FILE LISTA WORK.AREA RECORD.STAMPA;

   DECLARE J               FIXED;

   PROCEDURE PERMUT (STRING, N);

      PROCEDURE SC.CALL.SC (STRING, N, K);
         IF K > 0
           THEN
              DO;
                 SCAMBIO (STRING (K-1), STRING (N-1));
                 PERMUT (STRING, N-1);
                 SCAMBIO (STRING (K-1), STRING (N-1));
                 SC.CALL.SC (STRING, N, K-1);
              END;
      END SC.CALL.SC;

      IF N > 1
        THEN
           SC.CALL.SC (STRING, N, N);
        ELSE
           DO;
             J := 0;
             DO FOREVER;
               IF J >= 10
                 THEN UNDO;
               STAMPA.CH(J-1+1) := STRING(J-1+1);
               J := J + 1;
             END;
             WRITE (LISTA);
           END;

   END PERMUT;
```

```
    DECLARE 1 RECORD.STRING  CHARACTER (10),
              2 STRING       CHARACTER (1);

    DECLARE N                FIXED;

    DISPLAY ("INSERISCI IL NUMERO DI ELEMENTI");
    INPUT.N (N);

    IF N > 0 AND N <= 10
      THEN
          DO;
            J := 0;

            DO FOREVER;
              IF J >= N
                THEN UNDO;
              CONVERT (0, STRING(J-1+1), J);
              J := J + 1;
            END;

            OPEN (LISTA);

            PERMUT (STRING, N);

            CLOSE (LISTA);

          END;

END MAIN;
FINI;
%----------------------------
    DEVICE     (LISTA) := PR;
    RECORD     (LISTA) := 10;
    BUFFER     (LISTA) := 10;
    NO.BUFFERS (LISTA) := 1;
    NO.LABEL   (LISTA) := 1;
    CLOSEMODE  (LISTA) := RELEASE;
    ACCESSMODE (LISTA) := SEQUENTIAL;
    MYUSE      (LISTA) := OUTPUT;
%----------------------------
```

Variabili:

SAVEN e` il vettore utilizzato per conservare il valore di N;

SAVEK e` il vettore utilizzato per conservare il valore di K;

SP e` lo stack pointer.

Pseudocodifica non ricorsiva:

```
PERMUT

   SCAMBIO.CHIAMATA.SCAMBIO

      IF K > 0
        THEN
           STRING (K) :==: STRING (N)
           SP := SP + 1
           SAVEN(SP) := N
           N := N - 1
           PERMUT
           N := SAVEN(SP)
           SP := SP - 1
           STRING (K) :==: STRING (N)
           SP := SP + 1
           SAVEK(SP) := K
           K := K - 1
           SCAMBIO.CHIAMATA.SCAMBIO
           K := SAVEK(SP)
           SP := SP - 1
      FI

   END SCAMBIO.CHIAMATA.SCAMBIO

   IF N > 1
     THEN
        SP := SP + 1
        SAVEK(SP) := K
        K := N
        SCAMBIO.CHIAMATA.SCAMBIO
        K := SAVEK(SP)
        SP := SP - 1
     ELSE
        scrittura STRING
   FI

END PERMUT
```

Codifica COBOL:

```
000400*************************************************************************
000500* PROGRAMMA COBOL HC.07                                                 *
000600*                                                                       *
000700* DATA:              19/06/85  (GG/MM/AA)                               *
000800*                                                                       *
000900* ARGOMENTO: SIMULAZIONE DI RICORSIONE CON PERMUTAZIONI                 *
001000*                                                                       *
001100*************************************************************************
005800
005900
006000
006100 IDENTIFICATION DIVISION.
006200
006300
006400 ENVIRONMENT DIVISION.
006500
006600
006700 DATA DIVISION.
006800
006900
007000 WORKING-STORAGE SECTION.
007100
007200 01  RECORD-STACKS.
007300     02  SAVEN  OCCURS 100 TIMES PIC 9.
007400     02  SAVEK  OCCURS 100 TIMES PIC 9.
007500
007600 01  STACK-POINTERS.
007700     02  SP                      PIC 999.
007800
007900 01  VARIABILI-SCALARI.
008000     02  N                       PIC 9.
008100     02  K                       PIC 9.
008200     02  TEMP                    PIC 9.
008300     02  J                       PIC 99.
008400
008500 01  RECORD-STRING.
008600     02  STRING OCCURS 10 TIMES PIC 9.
008700
```

```
008800 PROCEDURE DIVISION.
008900
009000 MAIN.
009100
009200     DISPLAY "INSERISCI IL NUMERO DI ELEMENTI DA PERMUTARE".
009300     DISPLAY "(UNA CIFRA)".
009400     ACCEPT N.
009500
009600*      si genera la prima permutazione con numeri in ordine
009650*      crescente
009700     MOVE SPACES TO RECORD-STRING.
009800     PERFORM GEN-PRIMA-PERMUTAZIONE VARYING J FROM 1 BY 1
009900                                    UNTIL J > N.
010000
010100     PERFORM PERMUT.
010200
010300     STOP RUN.
010400
010500
010600 GEN-PRIMA-PERMUTAZIONE.
010700
010800     MOVE J TO STRING(J).
010900
011000
011100 PERMUT.
011200
011300     IF N > 1
011400       THEN
011500
011600*           push
011700          COMPUTE SP = SP + 1,
011800          COMPUTE SAVEK(SP) = K,
011900
012000*           call
012100          COMPUTE K = N,
012200          PERFORM SCAMBIO-CHIAMATA-SCAMBIO,
012300
012400*           pop
012500          COMPUTE K = SAVEK(SP),
012600          COMPUTE SP = SP - 1;
012700
012800       ELSE
012900
013000          DISPLAY RECORD-STRING.
013100
013200
```

```
013300 SCAMBIO-CHIAMATA-SCAMBIO.
013400
013500     IF K > 0
013600       THEN
013700
013800*           scambio STRING(K) con STRING(N)
013900           MOVE STRING(K)     TO TEMP,
014000           MOVE STRING(N)     TO STRING(K),
014100           MOVE TEMP          TO STRING(N),
014200
014300*           push
014400           COMPUTE SP = SP + 1,
014500           COMPUTE SAVEN(SP) = N,
014600
014700*           call
014800           COMPUTE N = N - 1,
014900           PERFORM PERMUT,
015000
015100*           pop
015200           COMPUTE N = SAVEN(SP),
015300           COMPUTE SP = SP - 1,
015400
015500*           scambio STRING(K) con STRING(N)
015600           MOVE STRING(K)     TO TEMP,
015700           MOVE STRING(N)     TO STRING(K),
015800           MOVE TEMP          TO STRING(N),
015900
016000*           push
016100           COMPUTE SP = SP + 1,
016200           COMPUTE SAVEK(SP) = K,
016300
016400*           call
016500           COMPUTE K = K - 1,
016600           PERFORM SCAMBIO-CHIAMATA-SCAMBIO,
016700
016800*           pop
016900           COMPUTE K = SAVEK(SP),
017000           COMPUTE SP = SP - 1.
```

Bibliografia:

Wagener J. L.: "FORTRAN 77 Principles of Programming" {5}
Wiley, 1980, pp. 228-229.

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.

BIBLIOGRAFIA

Il concetto di locale e di globale: scope delle variabili

With N.: "Principi di programmazione strutturata" {1}
ISEDI, 1977, capitolo 12.

Moore L.: "Foundations of Programming with Pascal" {3}
Ellis Horwood Limited, 1980, capitolo 10.

Ledgard, Nagin, Hueras: "Pascal with Style" {2}
Hayden, 1979, pp. 126-134.

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" {4}
Addison-Wesley, 1974, pp. 55-60.

Ledgard, Nagin, Hueras: "Pascal with style"
Hayden, 1979, pp. 134-139.

Wirth N.: "Algorithms + Data Structures = Programs"
Prentice-Hall, 1976, capitolo 3.

Wagener J. L.: "FORTRAN 77 Principles of Programming" {5}
Wiley, 1980, capitolo 11.

I linguaggi

Burroughs Corporation: "Computer Management System COBOL - reference manual" codice 2007266.

Burroughs Corporation: "Computer Management System Message Processing Language (MPLII) - reference manual" codice 2007563.

APPENDICE 1

Note particolari sulla Pseudocodifica utilizzata

Il simbolo % (percento) precede un commento e puo` trovarsi anche immediatamente a destra di una istruzione.

Il simbolo :==: indica che si intende scambiare il contenuto delle variabili che si trovano ai suoi lati.

APPENDICE 2

Library MPL II non standard utilizzate

LM.JR

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%LIBRARY :                                                                      %
%   PROCEDURA CHE PERMETTE L' ALLINEAMENTO A DESTRA DI UN NUMERO.               %
%   USO DELLA FUNCTION "SIZE".                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  PROCEDURE SPOSTA.DESTRA (VARIABILE);
    DECLARE  DIMENSIONE      FIXED;
    DIMENSIONE := SIZE (VARIABILE);
    DECLARE  A               FIXED;
    DECLARE  B               FIXED;
    DECLARE  VAR.INPUT       CHARACTER (DIMENSIONE);
    REMAP    VAR.INPUT: ELEM.A(DIMENSIONE) CHARACTER (1);
    DECLARE  VAR.OUTPUT      CHARACTER (DIMENSIONE);
    REMAP    VAR.OUTPUT: ELEM.B(DIMENSIONE) CHARACTER (1);

    VAR.INPUT := VARIABILE;

    A := DIMENSIONE - 1;     %   QUESTO PERCHE' IL VETTORE HA ANCHE
    B := DIMENSIONE - 1;     %   LA POSIZIONE ZERO

    DO FOREVER;
      IF ELEM.A(A) /= " "    %   BLANK
         THEN UNDO;
         ELSE DO;
                IF A = 0
                   THEN DO;
                          VARIABILE := VAR.OUTPUT;
                          RETURN;
                        END;
                   ELSE A :- 1;
              END;
    END;
```

```
   DO FOREVER;
     IF ELEM.A(A) = " " %BLANK
        THEN ELEM.B(B) := "0";
        ELSE ELEM.B(B) := ELEM.A(A);
     IF A = 0
        THEN DO FOREVER;
               IF B = 0
                  THEN DO;
                         VARIABILE := VAR.OUTPUT;
                         RETURN;
                       END;
                  ELSE DO;
                         B :- 1;
                         ELEM.B(B) := "0";
                       END;
             END;
        ELSE DO;
               A :- 1;
               B :- 1;
             END;
   END;
 END SPOSTA.DESTRA;
```

LM.FX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%LIBRARY  MPL:                                                                  %
%   PROCEDURA CHE CONVERTE UN NUMERO ASCII O PACKED IN FIXED                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PROCEDURE CONV.FX (TIPO.CONVERSIONE, VAR.FIXED, VAR.INPUT);

  PROCEDURE CONV (VAR.FIXED, VAR.PACKED);

    REMAP VAR.FIXED: B(16)  BIT(1);

    PROCEDURE SETTAGGIO.BIT (N.BIT, VALORE, PESO);

      IF VALORE = 1
        THEN
          DO;
            B(*N.BIT) := @(1)1@;
            DEC.SUB (VAR.PACKED, PESO);
          END;
        ELSE
          B(*N.BIT) := @(1)0@;

    END SETTAGGIO.BIT;
```

```
IF DEC.COMP (VAR.PACKED, "@3000000000000000@") = 65535
 THEN
     DO;
       DISPLAY ("ERROR**NUMERO < ZERO NON AMMESSO");
       STOP;
     END;
 ELSE
     IF DEC.COMP (VAR.PACKED, "@3000000000065535@") = 1
       THEN
           DO;
             DISPLAY ("ERROR**NUMERO > 65535");
             STOP;
           END;

IF DEC.COMP (VAR.PACKED, "@3000000000032768@") /= 65535
 THEN
     SETTAGGIO.BIT (0, 1, "@3000000000032768@");

IF DEC.COMP (VAR.PACKED, "@3000000000016384@") /= 65535
 THEN
     SETTAGGIO.BIT (1, 1, "@3000000000016384@");

IF DEC.COMP (VAR.PACKED, "@3000000000008192@") /= 65535
 THEN
     SETTAGGIO.BIT (2, 1, "@3000000000008192@");

IF DEC.COMP (VAR.PACKED, "@3000000000004096@") /= 65535
 THEN
     SETTAGGIO.BIT (3, 1, "@3000000000004096@");

IF DEC.COMP (VAR.PACKED, "@3000000000002048@") /= 65535
 THEN
     SETTAGGIO.BIT (4, 1, "@3000000000002048@");

IF DEC.COMP (VAR.PACKED, "@3000000000001024@") /= 65535
 THEN
     SETTAGGIO.BIT (5, 1, "@3000000000001024@");
```

```
  IF DEC.COMP (VAR.PACKED, "@30000000000000512@") /= 65535
    THEN
       SETTAGGIO.BIT (6, 1, "@300000000000000512@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000256@") /= 65535
    THEN
       SETTAGGIO.BIT (7, 1, "@300000000000000256@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000128@") /= 65535
    THEN
       SETTAGGIO.BIT (8, 1, "@300000000000000128@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000064@") /= 65535
    THEN
       SETTAGGIO.BIT (9, 1, "@300000000000000064@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000032@") /= 65535
    THEN
       SETTAGGIO.BIT (10, 1, "@300000000000000032@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000016@") /= 65535
    THEN
       SETTAGGIO.BIT (11, 1, "@300000000000000016@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000008@") /= 65535
    THEN
       SETTAGGIO.BIT (12, 1, "@300000000000000008@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000004@") /= 65535
    THEN
       SETTAGGIO.BIT (13, 1, "@300000000000000004@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000002@") /= 65535
    THEN
       SETTAGGIO.BIT (14, 1, "@300000000000000002@");

  IF DEC.COMP (VAR.PACKED, "@30000000000000001@") /= 65535
    THEN
       SETTAGGIO.BIT (15, 1, "@300000000000000001@");

END CONV;
```

```
  DECLARE VAR.PACKED CHARACTER (8);
  DECLARE VAR.INPUT.SIZE FIXED;

  VAR.INPUT.SIZE := SIZE (VAR.INPUT);

  DECLARE VAR.INPUT.RECO CHARACTER (VAR.INPUT.SIZE);

  VAR.INPUT.RECO := VAR.INPUT;

  VAR.FIXED := 0;

  IF TIPO.CONVERSIONE = 0
     THEN CONV (VAR.FIXED, VAR.INPUT);
     ELSE IF TIPO.CONVERSIONE = 1
          THEN DO;
                 CONVERSION (5, VAR.PACKED, VAR.INPUT);
                 CONV (VAR.FIXED, VAR.PACKED);
               END;
          ELSE DO;
                 DISPLAY ("*ERROR CONV.FX*  TIPO.CONVERSIONE ERRATO");
                 DISPLAY ("*ERROR CONV.FX*  FINE LAVORO FORZATO!");
                 STOP;
               END;

  VAR.INPUT := VAR.INPUT.RECO;

END CONV.FX;
```

LM.IN.N

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   LIBRARY  MPL.                    27/04/84                                   %
%                                                                               %
%   ESEGUE L'ACCEPT DI UN NUMERO FIXED CON GLI ADEGUATI CONTROLLI;              %
%   LA CHIAMATA DEVE ESSERE FATTA COSI': ''INPUT.N (N.FIXED)'' DOVE             %
%   N.FIXED RAPPRESENTA LA VARIABILE FIXED CHE SI VUOLE RIEMPIRE.               %
%                                                                               %
%   QUESTA LIBRARY  FA USO DI:  LM.JR, LM.FX, CONVERSION, DECIMAL               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PROCEDURE INPUT.N (N.FIXED);
  DECLARE N.ASCII     CHARACTER (16);
  DECLARE N.PACKED    CHARACTER (8);

  ACCEPT (N.ASCII);
  SPOSTA.DESTRA (N.ASCII);
  CONVERSION (5, N.PACKED, N.ASCII);

  IF DEC.COMP (N.PACKED, "@3000000000000000@") = @FFFF@
    THEN
        DO;
          DISPLAY ("IL NUMERO INSERITO E' MINORE DI ZERO, RIPETI");
          INPUT.N (N.FIXED);
          RETURN;
        END;
    ELSE
        IF DEC.COMP (N.PACKED, "@3000000000065535@") = 1
          THEN
              DO;
                DISPLAY ("IL NUMERO E' MAGGIORE DI 65535, RIPETI");
                INPUT.N (N.FIXED);
                RETURN;
              END;
          ELSE
              % IL NUMERO E' VALIDO
              ;

  CONV.FX (0, N.FIXED, N.PACKED);

END INPUT.N;
```

LM.OUT.N

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   LIBRARY  MPL.          27/04/84                                     %
%                                                                       %
%   ESEGUE IL DISPLAY DI UN NUMERO FIXED;                               %
%                                                                       %
%   PER CHIAMARE LA PROCEDURA: ''OUTPUT.N(N.FIXED)'' DOVE N.FIXED E'    %
%   IL NUMERO O LA VARIABILE FIXED CHE SI VUOLE VISUALIZZARE SU SPO.    %
%                                                                       %
%   LA LIBRARY  FA USO DI: CONVERSION                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PROCEDURE OUTPUT.N (N.FIXED);

  DECLARE N.ASCII CHARACTER (16);

  CONVERT (1, N.ASCII, N.FIXED);

  DISPLAY (N.ASCII);

END OUTPUT.N;
```

LM.IN.MN

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LIBRARY  MPL                              26/5/84                          %
%                                                                            %
% SONO TRE PROCEDURE CHE ESEGUONO L'INPUT DI UN VETTORE (IN.MAT.N.1)         %
% NUMERICO, L'INPUT DI UNA MATRICE A DUE DIMENSIONI (IN.MAT.N.2)             %
% E L'INPUT PER UNA MATRICE A TRE DIMENSIONI (IN.MAT.N.3).                   %
%                                                                            %
% LA CHIAMATA E' DEL TIPO: IN.MAT.N.3 (TABLE, X, Y, Z)                       %
%                          IN.MAT.N.2 (TABLE, X, Y)                          %
%                          IN.MAT.N.1 (TABLE, X)                             %
% DOVE X, Y E Z SONO LE DIMENSIONI (RIGA, COLONNA, ... ) DELLA TABELLA       %
%                                                                            %
% PRIMA DI QUESTA LIBRARY  SONO NECESSARIE LE SEGUENTI:                      %
%      DECIMAL,                                                              %
%      CONVERSION,                                                           %
%      STRING,                                                               %
%      LM.JR,                                                                %
%      LM.FX,                                                                %
%      LM.IN.N.                                                              %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
PROCEDURE IN.MAT.N.1 (VETT, C);
  % IL VETTORE E' LUNGO C*2 BYTE

  DECLARE K                    FIXED;
  DECLARE K.CH                 CHARACTER (6);
  DECLARE INDICE               FIXED;
  DECLARE MESSAGGIO            CHARACTER (77);
  REMAP MESSAGGIO:MESS.CH(77)  CHARACTER (1);

  K := 0;

  DO FOREVER;
    K := K + 1;
    IF K > C
      THEN
        UNDO;

    % VISUALIZZARE IL MESSAGGIO: "INSEIRSCI L'ELEMENTO VETTORE(K)"

    SMEAR (MESS.CH, 0, 77, " ", 77);   % AZZERAMENTO RECORD MESSAGGIO
    CONVERT(1, K.CH, K);
    SUBSTR(MESSAGGIO, 0, 29) := "INSERISCI L'ELEMENTO VETTORE(";
    SUBSTR(MESSAGGIO, 29, 6) := K.CH;
    SUBSTR(MESSAGGIO, 36, 1) := ")";
    DISPLAY (MESSAGGIO);

    INDICE := 2 * (K - 1);
    INPUT.N (VETT(INDICE));

  END;

END IN.MAT.N.1;
```

```
PROCEDURE IN.MAT.N.2 (MATR, R, C);
 % IL VETTORE E' LUNGO C*R*2 BYTE

 DECLARE M                    FIXED;
 DECLARE M.CH                 CHARACTER (6);
 DECLARE N                    FIXED;
 DECLARE N.CH                 CHARACTER (6);
 DECLARE INDICE               FIXED;
 DECLARE MESSAGGIO            CHARACTER (77);
 REMAP MESSAGGIO:MESS.CH(77)  CHARACTER (1);

 M := 0;

 DO FOREVER;
  M :+ 1;
  IF M > R
   THEN
     UNDO;
  N := 0;
  DO FOREVER;
   N :+ 1;
   IF N > C
    THEN
      UNDO;

   % VISUALIZZARE: "INSEIRSCI L'ELEMENTO MATRICE(M,N)"

   SMEAR (MESS.CH, 0, 77, " ", 77);  % AZZERAMENTO RECORD MESSAGGIO
   CONVERT(1, M.CH, M);
   CONVERT(1, N.CH, N);
   SUBSTR(MESSAGGIO, 0, 29) := "INSERISCI L'ELEMENTO MATRICE(";
   SUBSTR(MESSAGGIO, 29, 6) := M.CH;
   SUBSTR(MESSAGGIO, 36, 1) := ",";
   SUBSTR(MESSAGGIO, 38, 6) := N.CH;
   SUBSTR(MESSAGGIO, 45, 1) := ")";
   DISPLAY (MESSAGGIO);

   INDICE := 2 * ((M - 1) * C + (N - 1));
   INPUT.N (MATR(INDICE));

  END;
 END;

END IN.MAT.N.2;
```

```
PROCEDURE IN.MAT.N.3 (MATR, X, Y, Z);
 % IL VETTORE E' LUNGO X*Y*Z*2 BYTE

 DECLARE M                    FIXED;
 DECLARE M.CH                 CHARACTER (6);
 DECLARE N                    FIXED;
 DECLARE N.CH                 CHARACTER (6);
 DECLARE O                    FIXED;
 DECLARE O.CH                 CHARACTER (6);
 DECLARE INDICE               FIXED;
 DECLARE MESSAGGIO            CHARACTER (77);
 REMAP MESSAGGIO:MESS.CH(77)  CHARACTER (1);

 M := 0;

 DO MLOOP FOREVER;
   M :+ 1;
   IF M > X
     THEN
        UNDO MLOOP;
   N := 0;
   DO NLOOP FOREVER;
     N :+ 1;
     IF N > Y
       THEN
          UNDO NLOOP;
     O := 0;
     DO OLOOP FOREVER;
       O :+ 1;
       IF O > Z
         THEN
            UNDO OLOOP;

       % VISUALIZZARE: "INSEIRSCI L'ELEMENTO MATRICE(X,Y,Z)"

       SMEAR (MESS.CH, 0, 77, " ", 77);  % AZZERAMENTO  MESSAGGIO
       CONVERT(1, M.CH, M);
       CONVERT(1, N.CH, N);
       CONVERT(1, O.CH, O);
       SUBSTR(MESSAGGIO, 0, 29) := "INSERISCI L'ELEMENTO MATRICE(";
       SUBSTR(MESSAGGIO, 29, 6) := M.CH;
       SUBSTR(MESSAGGIO, 36, 1) := ",";
       SUBSTR(MESSAGGIO, 38, 6) := N.CH;
       SUBSTR(MESSAGGIO, 45, 1) := ",";
       SUBSTR(MESSAGGIO, 47, 6) := O.CH;
       SUBSTR(MESSAGGIO, 54, 1) := ")";
       DISPLAY (MESSAGGIO);

       INDICE := 2*((M-1)*Y*Z + (N-1)*Z + (O-1));
       INPUT.N (MATR(INDICE));

     END OLOOP;
```

```
    END NLOOP;
  END MLOOP;

END IN.MAT.N.3;
```

LM.OUT.MN

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LIBRARY  MPL                           26/5/84                       %
%                                                                      %
% SONO TRE PROCEDURE CHE ESEGUONO L'OUTPUT DI UN VETTORE (OUT.MAT.N.1) %
% NUMERICO, L'OUTPUT DI UNA MATRICE A DUE DIMENSIONI (OUT.MAT.N.2)     %
% E L'OUTPUT PER UNA MATRICE A TRE DIMENSIONI (OUT.MAT.N.3).           %
%                                                                      %
% LA CHIAMATA E' DEL TIPO: OUT.MAT.N.3 (TABLE, X, Y, Z)                %
%                          OUT.MAT.N.2 (TABLE, X, Y)                   %
%                          OUT.MAT.N.1 (TABLE, X)                      %
% DOVE X, Y E Z SONO LE DIMENSIONI (RIGA, COLONNA, ... ) DELLA TABELLA %
%                                                                      %
% PRIMA DI QUESTA LIBRARY  SONO NECESSARIE LE SEGUENTI:                %
%      DECIMAL,                                                        %
%      CONVERSION,                                                     %
%      STRING,                                                         %
%      LM.JR,                                                          %
%      LM.FX,                                                          %
%                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 DECLARE RECORD.PR.FILE CHARACTER (132);
 REMAP   RECORD.PR.FILE: RECORD.PR.CH(132)  CHARACTER(1);

 FILE PRINTER.FILE WORK.AREA RECORD.PR.FILE;
%----------------------------
   DEVICE     (PRINTER.FILE) := PR;               % ANY PRINTER DEVICE
   RECORD     (PRINTER.FILE) := 132;
   BUFFER     (PRINTER.FILE) := 132;
   NO.BUFFERS (PRINTER.FILE) := 1;
   NO.LABEL   (PRINTER.FILE) := 1;
   CLOSEMODE  (PRINTER.FILE) := RELEASE;
   MYUSE      (PRINTER.FILE) := OUTPUT;
%----------------------------
```

```
PROCEDURE OUT.MAT.N.1 (VETT, C);
  % IL VETTORE E' LUNGO C*2 BYTE


  DECLARE K                    FIXED;
  DECLARE K.CH                 CHARACTER (6);
  DECLARE VETT.CH              CHARACTER (6);
  DECLARE MESSAGGIO            CHARACTER (132);
  REMAP MESSAGGIO:MESS.CH(132) CHARACTER (1);

  OPEN (PRINTER.FILE);

  % SCRIVI UNA RIGA DI TRATTINI
  SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
  WRITE (PRINTER.FILE);

  K := 0;

  DO FOREVER;
    K := K + 1;
    IF K > C
      THEN
        UNDO;

    % SCRIVI IL VALORE DELL'ELEMENTO  VETTORE(K)

    SMEAR (MESS.CH, 0, 132, " ", 132);  % AZZERAMENTO RECORD MESSAGGIO
    CONVERT(1, K.CH, K);
    CONVERT(1, VETT.CH, VETT((K-1)*2));
    SUBSTR(MESSAGGIO, 0, 24) := "L'ELEMENTO DEL VETTORE (";
    SUBSTR(MESSAGGIO, 24, 6) := K.CH;
    SUBSTR(MESSAGGIO, 31, 4) := ") = ";
    SUBSTR(MESSAGGIO, 36, 6) := VETT.CH;
    RECORD.PR.FILE := MESSAGGIO;
    WRITE (PRINTER.FILE);

  END;

  %SCRIVE UNA RIGA DI TRATTINI
  SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
  WRITE (PRINTER.FILE);

  CLOSE (PRINTER.FILE);

END OUT.MAT.N.1;
```

```
PROCEDURE OUT.MAT.N.2 (MATR, R, C);
 % IL VETTORE E' LUNGO C*R*2 BYTE

 DECLARE M                    FIXED;
 DECLARE M.CH                 CHARACTER (6);
 DECLARE N                    FIXED;
 DECLARE N.CH                 CHARACTER (6);
 DECLARE MATR.CH              CHARACTER (6);
 DECLARE MESSAGGIO            CHARACTER (132);
 REMAP MESSAGGIO:MESS.CH(132) CHARACTER (1);

 OPEN (PRINTER.FILE);

 % SCRIVE UNA RIGA DI TRATTINI
 SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
 WRITE (PRINTER.FILE);

 M := 0;

 DO FOREVER;
  M =+ 1;
  IF M > R
   THEN
     UNDO;
  N := 0;
  DO FOREVER;
   N =+ 1;
   IF N > C
    THEN
      UNDO;

   % SCRIVE L'ELEMENTO MATRICE(M,N)

   SMEAR (MESS.CH, 0, 132, " ", 132);      % AZZERAMENTO MESSAGGIO
   CONVERT(1, M.CH, M);
   CONVERT(1, N.CH, N);
   CONVERT(1, MATR.CH, MATR((M-1)*2*C + (N-1)*2));
   SUBSTR(MESSAGGIO, 0, 26) := "L'ELEMENTO DELLA MATRICE (";
   SUBSTR(MESSAGGIO, 26, 6) := M.CH;
   SUBSTR(MESSAGGIO, 33, 1) := ",";
   SUBSTR(MESSAGGIO, 34, 6) := N.CH;
   SUBSTR(MESSAGGIO, 41, 4) := ") = ";
   SUBSTR(MESSAGGIO, 46, 6) := MATR.CH;

   RECORD.PR.FILE := MESSAGGIO;
   WRITE (PRINTER.FILE);

  END;
END;

% SCRIVE UNA RIGA DI TRATTINI
```

```
    SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
    WRITE (PRINTER.FILE);

    CLOSE (PRINTER.FILE);

END OUT.MAT.N.2;
```

```
PROCEDURE OUT.MAT.N.3 (MATR, X, Y, Z);
 % IL VETTORE E' LUNGO X*Y*Z*2 BYTE

 DECLARE M                    FIXED;
 DECLARE M.CH                 CHARACTER (6);
 DECLARE N                    FIXED;
 DECLARE N.CH                 CHARACTER (6);
 DECLARE O                    FIXED;
 DECLARE O.CH                 CHARACTER (6);
 DECLARE MATR.CH              CHARACTER (6);
 DECLARE MESSAGGIO            CHARACTER (132);
 REMAP MESSAGGIO:MESS.CH(132) CHARACTER (1);

 OPEN (PRINTER.FILE);

 % SCRIVE UNA RIGA DI TRATTINI;
 SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
 WRITE (PRINTER.FILE);

 M := 0;

 DO MLOOP FOREVER;
  M :+ 1;
  IF M > X
    THEN
       UNDO;
  N := 0;
  DO NLOOP FOREVER;
    N :+ 1;
    IF N > Y
      THEN
         UNDO;
    O := 0;
    DO OLOOP FOREVER;
      O :+ 1;
      IF O > Z
        THEN
           UNDO;

      % SCRIVE L'ELEMENTO DELLA MATRICE (X,Y,Z)

      SMEAR (MESS.CH, 0, 132, " ", 132);  % AZZERAMENTO  MESSAGGIO
      CONVERT(1, M.CH, M);
      CONVERT(1, N.CH, N);
      CONVERT(1, O.CH, O);
      CONVERT(1, MATR.CH, MATR((M-1)*2*Y*Z + (N-1)*2*Z + (O-1)*2));
      SUBSTR(MESSAGGIO, 0, 26) := "L'ELEMENTO DELLA MATRICE (";
      SUBSTR(MESSAGGIO, 26, 6) := M.CH;
      SUBSTR(MESSAGGIO, 33, 1) := ",";
      SUBSTR(MESSAGGIO, 35, 6) := N.CH;
      SUBSTR(MESSAGGIO, 42, 1) := ",";
      SUBSTR(MESSAGGIO, 44, 6) := O.CH;
```

```
            SUBSTR(MESSAGGIO, 51, 4) := ") = ";
            SUBSTR(MESSAGGIO, 56, 6) := MATR.CH;

            RECORD.PR.FILE := MESSAGGIO;
            WRITE (PRINTER.FILE);

          END OLOOP;
        END NLOOP;
      END MLOOP;

      %SCRIVI UNA RIGA DI TRATTINI
      SMEAR (RECORD.PR.CH, 0, 132, "-", 132);
      WRITE (PRINTER.FILE);

      CLOSE (PRINTER.FILE);

    END OUT.MAT.N.3;
```

LM.XCH

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LIBRARY  MPL.                           29/5/84                            %
%                                                                            %
% QUESTA LIBRARY  SERVE PER SCAMBIARE IL CONTENUTO DI DUE VARIABILI.         %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 PROCEDURE SCAMBIO (A, B);
   PROCEDURE SCAMBIO.TYPE1 (A, B, SIZEA);
     DECLARE NEW.A CHARACTER (SIZEA);
     NEW.A := A;
     A := B;
     B := NEW.A;
   END SCAMBIO.TYPE1;
   PROCEDURE SCAMBIO.TYPE2 (A, B);
     DECLARE NEW.A FIXED;
     NEW.A := A;
     A := B;
     B := NEW.A;
   END SCAMBIO.TYPE2;
   PROCEDURE SCAMBIO.TYPE3 (A, B);
     DECLARE NEW.A BIT (8);
     NEW.A := A;
     A := B;
     B := NEW.A;
   END SCAMBIO.TYPE3;

   DECLARE SIZEA  FIXED;

   SIZEA := SIZE(A);

   IF TYPE(A) = 1
     THEN
         SCAMBIO.TYPE1 (A, B, SIZEA);
   IF TYPE(A) = 2
     THEN
         SCAMBIO.TYPE2 (A, B);
   IF TYPE(A) = 3
     THEN
         SCAMBIO.TYPE3 (A, B);
   IF TYPE(A) = 4 OR TYPE(A) = 0 OR TYPE(B) = 4 OR TYPE(B) = 0
     THEN
         DO;
           DISPLAY ("ERROR 'SCAMBIO' VARIABILE ");
           STOP;
         END;

 END SCAMBIO;
```