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 qua-

le 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 (una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/cobol-line-renumber.sh*).

```
#!/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 conte-

nuto 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 par-

ticolare 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**’.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0100.cob* .

```
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».

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0200.cob* .

```
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**’.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0300.cob* .

```
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**'.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0400.cob* .

```
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**’.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0500.cob* .

```
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**'.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0600.cob* .

```
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**'.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0700.cob* .

```
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».

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0800.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM0900.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM1000.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM1100.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM1300.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM1400.cob* .

```
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**’.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/ELM1500.cob* .

```
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

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-1.cob* .

```
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

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-2.cob* .

```
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

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-3.cob* .

```
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

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-4.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-5.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-6.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-8.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-10.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-12.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-13.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-14.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-15.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-16.cob* .

```
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`’.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-18.cob* .

```
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.

Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/AGO-83-20.cob* .

```
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
#
PROC1 (V, G, Z)
    .
    .
    .
    .
    .
    .
    .
    # G è una variabile in
    # input-output
    PROC1 (V, G, Z-1)
    .
    .
    .
    .
    .
    .
    .
END PROC1
```

```
#
# Trasformazione non ricorsiva
#
PROC1
    .
    .
    .
    # push
    SP := SP + 1
    SAVEV(SP) := V
    SAVEZ(SP) := Z
    # chiamata
    Z := Z -1
    PROC1
    # pop
    V := SAVEV(SP)
    Z := SAVEZ(SP)
    SP := SP - 1
    .
    .
    .
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**'.

Variabile	Descrizione
`SAVEP2`	È il vettore utilizzato per conservare il valore di '**`P2`**'.
`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. Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/HC04.cob* .

```
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**'. Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/HC06.cob* .

```
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. Una copia di questo file dovrebbe essere disponibile presso *allegati/cobol/HC07.cob* .

```
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