

# COLLEGAMENTO E CARICAMENTO

## INTRODUZIONE

Quando s'inizializza il sistema, un processo, il *command.com* (la shell) parte ed attende un input. Quando è digitata una riga, il *command.com* fa partire un nuovo processo mediante la funzione 4B00H, alloca memoria, scrive un PSP per il nuovo programma all'offset zero della memoria allocata, carica il nuovo programma e gli passa il controllo aspettando finché non abbia finito.

MS-DOS ha due tipi di file binari, che danno origine a due tipi differenti di processi.

## FILE CON ESTENSIONE .COM (CORE IMAGE)

È un semplice file eseguibile, esso non ha intestazione ed ha un solo segmento. Il file eseguibile è un'immagine esatta byte per byte del codice eseguibile; il file è caricato nella memoria così com'è ed è eseguito, non è rilocabile, è compatto e caricato velocemente. Questo tipo di modello di processo proviene dal CP/M. Un processo generato da un file .COM ha un segmento di codice, più dati e più stack lungo non più di 64K, e *command.com* gli alloca tutta la memoria disponibile e inizializza SP = 0100H byte in cima al segmento da 64K. Se un .COM che non ha ancora rilasciato parte della memoria fornitagli dal sistema, ne richiede altra, riceverà quell'occupata dalla parte residente di *command.com* che dovrà essere ricaricato a processo ultimato. Per minimizzare questo rischio un programma .COM deve ridurre il blocco che il sistema gli ha allocato inizialmente (INT 4AH) e rilasciare tutta la memoria allocata.

tiny.com

```
DOSSEG
.MODEL TINY
.DATA
Stringa DB "Ciao, mondo",13,10,'$'
.CODE
ORG 100H
BEGIN:  MOV DX,OFFSET Stringa
        MOV AH,9
        INT 21H
        MOV AX,4C00H
        INT 21H
END BEGIN
```

tiny.lst

```
1          DOSSEG
2      0000      .MODEL TINY
3      0000      .DATA
4      0000      43 69 61 6F 2C 20 6D+ Stringa DB  "Ciao, mondo",13,10,'$'
5          6F 6E 64 6F 0D 0A 24
6      000E      .CODE
7          ORG 100H
8      0100      BA 0000r BEGIN:  MOV DX,OFFSET Stringa
9      0103      B4 09      MOV AH,9
10     0105      CD 21      INT 21H
11     0107      B8 4C00      MOV AX,4C00H
12     010A      CD 21      INT 21H
13     END BEGIN
```

Symbol Table

Symbol Name	Type	Value	Cref (defined at #)
??DATE	Text	"01/05/01"	
??FILENAME	Text	"tiny"	"
??TIME	Text	"02:54:32"	

```

??VERSION          Number 030A
@32BIT             Text    0                #2
@CODE              Text    DGROUP          #2
@CODESIZE         Text    0                #2
@CPU               Text    0101H
@CURSEG           Text    _TEXT          #3 #6
@DATA              Text    DGROUP          #2
@DATASIZE         Text    0                #2
@FILENAME         Text    TINY
@INTERFACE        Text    00H             #2
@MODEL            Text    1                #2
@STACK            Text    DGROUP          #2
@WORDSIZE         Text    2                #3 #6
BEGIN              Near   DGROUP:0100     #8 13
STRINGA           Byte   DGROUP:0000     #4 8
Groups & Segments Bit Size  Align Combine Class      Cref (defined at #)
DGROUP           Group
  _DATA          16 000E    Word Public  DATA      #2 #3
  _TEXT          16 010C    Word Public  CODE       #2 #6

```

### tiny.ref

```

Turbo CREF Version 2.0 01/05/01 02:55:50 Page 1
Global Symbol Name      Cref (defined at #)

```

### tiny.map

```

Start      Stop          Length      Name          Class
00000H     0010BH     0010CH     _TEXT         CODE
0010CH     00119H     0000EH     _DATA         DATA

```

```

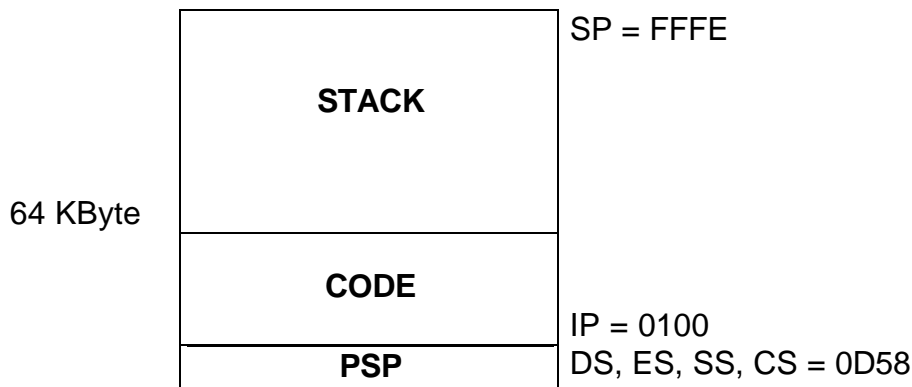
C:\MIC1\ESERCIZI\7>debug tiny.com

```

```

-R
AX=0000 BX=0000 CX=001A DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0D58 ES=0D58 SS=0D58 CS=0D58 IP=0100 NV UP EI PL NZ NA PO NC

```



Sicuramente il valore esadecimale dei segmenti sul vostro PC sarà diverso da quello di quest'esempio, questo perché MS-DOS carica *debug* (e qualsiasi altro programma eseguibile) a partire dal primo segmento libero trovato in memoria. Gli eseguibili di tipo .COM presentano la caratteristica di avere i quattro registri di segmento allineati sul primo segmento trovato libero in memoria da MS-DOS.

```

C:\MIC1\ESERCIZI\7>debug tiny.com 01234

```

```

-D 0000L100
0D5C:0000  CD 20 FF 9F 00 9A F0 FE-1D F0 4F 03 34 07 8A 03  . . . . .O.4...
0D5C:0010  34 07 17 03 34 07 23 07-01 01 01 00 02 FF FF FF  4...4.#.....
0D5C:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 01 0D 4C 01  . . . . .L.
0D5C:0030  14 0C 14 00 18 00 5C 0D-FF FF FF FF 00 00 00 00  . . . . .\.....
0D5C:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00  . . . . .
0D5C:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 30 31 32  .!. . . . .012
0D5C:0060  33 34 20 20 20 20 20 20-00 00 00 00 00 20 20 20  34 . . . . .

```

```

0D5C:0070 20 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00 .....
0D5C:0080 06 20 30 31 32 33 34 0D-6D 20 30 31 32 33 34 0D . 01234.m 01234.
0D5C:0090 62 32 0D 4E 54 5C 73 79-73 74 65 6D 33 32 3B 43 b2.NT\system32;c
0D5C:00A0 3A 5C 57 49 4E 4E 54 3B-43 3A 5C 57 49 4E 4E 54 :\WINNT;c:\WINNT
0D5C:00B0 5C 53 79 73 74 65 6D 33-32 5C 57 62 65 6D 3B 63 \System32\Wbem;c
0D5C:00C0 3A 5C 6A 64 6B 5C 62 69-6E 3B 63 3A 5C 62 6F 72 :\jdk\bin;c:\bor
0D5C:00D0 6C 61 6E 64 63 5C 62 69-6E 3B 0D 00 00 00 00 00 landc\bin;.....
0D5C:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0D5C:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

```

I primi 256 byte (da 0000H a 00FFH, 16 paragrafi) di ogni processo MS-DOS formano un blocco speciale, chiamato **PSP (Program Segment Prefix)**, ed anche quest'idea è stata presa direttamente dal CP/M. Il PSP è costruito dal sistema operativo al momento della creazione del processo. Per i file .COM, esso occupa una parte di spazio degli indirizzi del processo e può essere indirizzato utilizzando gli indirizzi da zero a 255; per questa ragione, tutti i processi .COM iniziano da 100H e non dall'indirizzo zero. Il PSP contiene la dimensione del programma, un pointer al blocco di ambiente, l'indirizzo del gestore di CTRL-C, la stringa di comando, un pointer al PSP del genitore, la tabella dei descrittori dei file e tutta una serie di informazioni destinate a MS-DOS, per controllare il programma in esecuzione; creato da INT 4B00H. I tre interrupt che iniziano all'offset 0AH permettono al processo padre di eseguire il processo figlio. L'indirizzo del gestore di CTRL-C si può cambiare da programma. Quando l'utente digita CTRL-C per interrompere il processo corrente, è chiamato questo gestore. Il buffer della coda di comando inizia all'offset 80H e può contenere al massimo 127 byte, il primo byte riporta il numero di caratteri introdotti dopo il comando, escluso il CR che è in ogni caso memorizzato nel buffer. I caratteri introdotti da tastiera sono, quindi, memorizzati a partire dall'offset 81H, sono tolti il nome del programma chiamato ed eventuali comandi di ridirezione di I/O. Può essere sfruttato vantaggiosamente ogni volta che un programma contiene dei parametri passati insieme al nome del programma stesso. Il PSP è creato automaticamente dalla function `Execute Program; Load Overlay`.

OFFSET	SIZE	CONTENUTI
00H	2	INT 20 (CODICE OPERATIVO CDH)
02H	2	SEGMENTO FINE BLOCCO ALLOCAZIONE
04H	1	RISERVATO
05H	5	CHIAMATA ALLA VECCHIA FUNZIONE DOS
0AH	4	CS:IP INT 22H (PROGRAM TERMINATE ADDRESS)
0EH	4	CS:IP INT 23H (CTRL-C)
12H	4	CS:IP INT 24H (CRITICAL ERROR)
16H	16	PARENT PROGRAM SEGMENT PREFIX
2CH	2	POINTER ALL'ENVIRONMENT BLOCK (SET DI MS-DOS)
2EH	22	RISERVATO
50H	2	INT 21H
52H	0A	RISERVATO
5CH	10	FCB1
6CH	14	FCB2
80H	1	LUNGHEZZA CODA COMANDO
81H	7F	CODA COMANDO

### EB (Environment Block)

Contiene una copia dedicata al programma dell'ambiente DOS. Precede sempre il programma, non possiede dimensione fissa, ma è fissata al momento del caricamento; contiene tutte le variabili della shell nella forma `variabile = valore`. Si visualizza/memorizza con SET, ogni entry dell'ambiente è una stringa ASCII chiusa da un byte uguale a zero: questo formato è chiamato ASCIIZ (che termina con il carattere ASCII 0).

```
C:\>set
ALLUSERSPROFILE=C:\Documents and Settings\All Users
APPDATA=C:\Documents and Settings\ADMINISTRATOR\Dati applicazioni
classpath=C:\mic1\MicEmulator13042001\ijvmclass.zip;C:\mic1\MicEmulator13042001
classes.zip;%classpath%
CommonProgramFiles=C:\Programmi\File comuni
COMPUTERNAME=XXXXXXXXX
ComSpec=C:\WINNT\system32\cmd.exe
HOMEDRIVE=C:
HOMEPATH=\
LOGONSERVER=\\XXXXXXXXX
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Os2LibPath=C:\WINNT\system32\os2\dll;
Path=C:\WINNT\system32;C:\WINNT\System32\Wbem;c:\jdk\bin;c:\borlandc\bin;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 2 Stepping 1, AuthenticAMD
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0201
ProgramFiles=C:\Programmi
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\DOCUME~1\ADMINI~1\IMPOST~1\Temp
TMP=C:\DOCUME~1\ADMINI~1\IMPOST~1\Temp
USERDOMAIN= XXXXXXXXX
USERNAME=Administrator
USERPROFILE=C:\Documents and Settings\ADMINISTRATOR
windir=C:\WINNT
```

## FILE CON ESTENSIONE .EXE (EXECUTABLE)

Un processo creato da uno di questi file può avere un segmento di codice, uno di dati, uno di stack e molti segmenti extra così come ritiene necessario. A differenza dei file .COM, i file .EXE contengono informazioni di rilocazione, in modo che possono essere rilocati al volo quando sono caricati. Il sistema operativo riconosce la differenza tra i file guardando i primi due byte e non guardando l'estensione del nome di file. I file .EXE sono rilocati prima del PSP, così l'indirizzo zero è il primo byte prima del PSP e questa strategia evita di perdere 256 byte dello spazio degli indirizzi.

### 1.asm

```
DOSSEG
.MODEL SMALL
STACK 200H
.DATA
Stringa DB "Ciao, mondo",13,10,'$'
.CODE

BEGIN:
MOV AX,@data
MOV DS,AX
MOV DX,OFFSET Stringa
MOV AH,9
INT 21H
MOV AX,4C00H
INT 21H

END BEGIN
```

### 1.lst

```
Turbo Assembler Version 3.1 30/04/01 23:09:41 Page 1 1.ASM
1          DOSSEG
2          0000          .MODEL SMALL
3          0000          STACK 200H
4          0000          .DATA
5          0000  43 69 61 6F 2C 20 6D+ Stringa DB  "Ciao, mondo",13,10,'$'
6          6F 6E 64 6F 0D 0A 24
7          000E          .CODE
8          0000          BEGIN:
9          0000  B8 0000s          MOV AX,@data
10         0003  8E D8          MOV DS,AX
11         0005  BA 0000r          MOV DX,OFFSET Stringa
12         0008  B4 09          MOV AH,9
13         000A  CD 21          INT 21H
14         000C  B8 4C00          MOV AX,4C00H
15         000F  CD 21          INT 21H
16         END BEGIN
```

#### Symbol Table

Symbol Name	Type	Value	Cref (defined at #)
??DATE	Text	"30/04/01"	
??FILENAME	Text	"1 "	
??TIME	Text	"23:09:41"	
??VERSION	Number	030A	
@32BIT	Text	0	#2
@CODE	Text	_TEXT	#2 #2 #7
@CODESIZE	Text	0	#2
@CPU	Text	0101H	
@CURSEG	Text	_TEXT	#4 #7
@DATA	Text	DGROUP	#2 9
@DATASIZE	Text	0	#2
@FILENAME	Text	1	
@INTERFACE	Text	00H	#2
@MODEL	Text	2	#2
@STACK	Text	DGROUP	#2
@WORDSIZE	Text	2	#4 #7

Groups & Segments	Bit	Size	Align	Combine	Class	Cref (defined at #)
BEGIN	Near				_TEXT:0000	#8 16
STRINGA	Byte				DGROUP:0000	#5 11
DGROUP	Group					#2 2 9
STACK	16	0200	Para	Stack	STACK	#3
_DATA	16	000E	Word	Public	DATA	#2 #4
_TEXT	16	0011	Word	Public	CODE	#2 2 #7 7

## 1.ref

Turbo CREF Version 2.0 30/04/01 23:11:48 Page 1  
Global Symbol Name Cref (defined at #)

## 1.map

Start	Stop	Length	Name	Class
00000H	00020H	00021H	_TEXT	CODE
00030H	0003DH	0000EH	_DATA	DATA
00040H	0023FH	00200H	STACK	STACK

Program entry point at 0000:0010

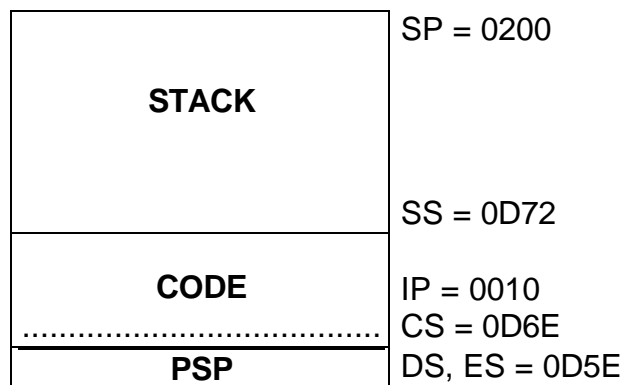
1.EXE = 256 (PSP) + 62 (codice) + 256 (rilocazione) = 574 byte

Gli eseguibili .EXE presentano la caratteristica di mantenere separate, anche fisicamente, le tre aree logiche di CODE, DATA e STACK del programma per sfruttare pienamente le capacità di gestione segmentata della memoria.

C:\MIC1\ESERCIZI\7>debug 1.exe

-R

AX=0000 BX=0000 CX=003E DX=0000 SP=0200 BP=0000 SI=0000 DI=0000  
DS=0D5E ES=0D5E SS=0D72 CS=0D6E IP=0010 NV UP EI PL NZ NA PO NC



0D5E:0000 = DS, ES:0000  
0D5E:0000 = DS, ES:00FF (PSP)  
0D5E:0100 = CS:0000 (0D5E0 + 0100 = 0F6E0)  
0D5E:0110 = CS:003E (CX = 3E = 62)  
0D5E:0140 = SS:0000 (0D5E0 + 0140 = 0D720)  
0D5E:0140 = SS:0200

C:\MIC1\ESERCIZI\7>DEBUG 1.EXE 01234

-D 0000L100

```

0D5F:0000  4D 5A 3E 00 02 00 01 00-20 00 21 00 FF FF 04 00  MZ.....O.6...
0D5F:0010  00 02 00 00 10 00 00 00-3E 00 00 00 01 FF FF FF  6...6.%.
0D5F:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 03 0D 4C 01  .....L.
0D5F:0030  16 0C 14 00 18 00 5F 0D-FF FF FF FF 00 00 00 00  ....._.....
0D5F:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
0D5F:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 30 31 32  ..!.....012
0D5F:0060  33 34 20 20 20 20 20 20-00 00 00 00 20 20 20  34 .....
0D5F:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00  .....
0D5F:0080  06 20 30 31 32 33 34 0D-31 32 33 34 0D 33 34 0D  . 01234.1234.34.
0D5F:0090  62 32 0D 4E 54 5C 73 79-73 74 65 6D 33 32 3B 43  b2.NT\system32;c
0D5F:00A0  3A 5C 57 49 4E 4E 54 3B-43 3A 5C 57 49 4E 4E 54  :\WINNT;c:\WINNT
0D5F:00B0  5C 53 79 73 74 65 6D 33-32 5C 57 62 65 6D 3B 63  \System32\Wbem;c

```

```

0D5F:00C0 3A 5C 6A 64 6B 5C 62 69-6E 3B 63 3A 5C 62 6F 72 : \jdk\bin;c:\bor
0D5F:00D0 6C 61 6E 64 63 5C 62 69-6E 3B 0D 00 00 00 00 00 landc\bin;.....
0D5F:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0D5F:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....

```

Quando un programma è caricato sono richieste informazioni per il calcolo della memoria necessaria e l'assegnazione dei valori iniziali ai diversi registri. Queste informazioni si trovano nella struttura **file header**, che è costruita dal linker ed è collocata all'inizio dei file di tipo .EXE. In fase di caricamento il sistema prova innanzi tutto ad allocare la memoria indicata da MAXALLOC, se questo non è possibile, cerca allora di assegnare tutta la memoria disponibile, purché questa abbia almeno le dimensioni indicate in MINALLOC, se anche questo tentativo non ha successo, il caricamento è interrotto. Per questo motivo MAXALLOC riceve spesso il valore FFFFH, in modo da assicurare che il sistema operativo allochi al processo tutta la memoria disponibile.

OFFSET	CONTENUTI
00H	Identificatore del tipo di file 4DH, 5AH, <M,Z>, Mark Zbikowski "Piccolo, ma onnipresente monumento posto a ricordo di se stesso"
02H	Numero di byte contenuti nell'ultima pagina, lunghezza file modulo 512
04H	Lunghezza del file in pagine di 512 byte, header compreso
06H	Numero dei riferimenti presenti nella tabella di rilocazione
08H	Lunghezza file header in paragrafi da 16 byte
0AH	Numero minimo di paragrafi necessari al programma: MINALLOC
0CH	Numero massimo di paragrafi richiesti dal programma: MAXALLOC
0EH	Valore iniziale di SS da rilocare
10H	Valore iniziale di SP
12H	Somma di controllo sulle word del file: CRC
14H	Valore iniziale di IP
16H	Valore iniziale di CS da rilocare
18H	Offset del primo elemento della tabella di rilocazione
1AH	Numero di overlay, è uguale a zero per la parte residente del programma
1CH	Riservato
variabile	Tabella di rilocazione
variabile	Riservato
variabile	Programma e dati

Al file header fa seguito una tabella di rilocazione (RELOCATION TABLE) per il calcolo degli indirizzi assoluti in funzione del punto di caricamento ed infine il codice compilato di programma vero e proprio. Dato che è pronto per il caricamento, un file di tipo .EXE è chiamato anche modulo di caricamento (load module). La tabella di rilocazione è formata da un numero variabile di oggetti da rilocare, ognuno dei quali è formato da due campi: un offset (2 byte) e un numero di segmento, riferiti a una word che richiede modifica prima che il controllo sia passato al modulo caricato.

1. La parte formattata dello header è trasferita in memoria. (1BH bytes).
2. Una porzione di memoria allocata in rapporto alla dimensione del modulo di caricamento e di MINALLOC e MAXALLOC. MS-DOS tenta di allocare FFFFH paragrafi, con lo scopo di ottenere la dimensione del massimo blocco disponibile. Se essa è minore di MINALLOC e MAXALLOC, sarà segnalato un errore. Se il blocco è maggiore di MAXALLOC, MS-DOS allocherà un blocco pari a (MAXALLOC + dimensione del blocco da caricare), altrimenti il sistema allocherà il blocco libero più grosso.
3. Il PSP costruito nella parte inferiore della memoria allocata.
4. La dimensione del modulo di caricamento (load module) valutata in base alle informazioni contenute nei campi 04H-05H, 08H-09h e 02H-03H dello header; il loader

localizza in memoria il segmento appropriato dove caricare il load module (start segment).

5. Il load module è trasferito in memoria a partire dallo start segment.
6. La tabella di rilocazione è trasferita in un'area di lavoro in RAM.
7. Per ogni elemento della tabella di rilocazione, si aggiunge il valore dello start segment al campo segmento. Il valore calcolato, più il contenuto del campo offset, un puntatore a una word nel load module alla quale aggiungere il valore dello start segment; il nuovo valore della word è infine riscritto nell'immagine in memoria del load module. (Questo procedimento è la RILCAZIONE).
8. I registri SS e SP sono inizializzati come indicato nei campi 0EH-0FH e 10H-11H; il loader incrementa poi SS del valore dello start segment. I registri ES e DS sono inizializzati con l'indirizzo di segmento del PSP. Infine il valore dello start segment aggiunto al valore iniziale per CS indicato nello header. Il risultato, insieme al valore iniziale per IP (campo 14H-15H) il punto iniziale CS:IP al quale trasferire il controllo per iniziare l'esecuzione del programma.



## ROUTINE

### 2.asm

```
DOSSEG
.MODEL SMALL
STACK 200H
.DATA
Stringa DB "Ciao, mondo",13,10
.CODE
BEGIN PROC NEAR
MOV AX,@data
MOV DS,AX
MOV DX,OFFSET Stringa
CALL STAMPA
MOV AX,4C00H
INT 21H
BEGIN ENDP
STAMPA PROC NEAR
PUSH AX
MOV AH,09H
INT 21H
POP AX
RET
STAMPA ENDP
END BEGIN
```

### 2.lst

```
1          DOSSEG
2      0000      .MODEL SMALL
3      0000      STACK 200H
4      0000      .DATA
5      0000      43 69 61 6F 2C 20 6D+ Stringa DB "Ciao, mondo",13,10
6      6F 6E 64 6F 0D 0A
7      000D      .CODE
8      0000      BEGIN PROC NEAR
9      0000      B8 0000s      MOV AX,@data
10     0003      8E D8      MOV DS,AX
11     0005      BA 0000r      MOV DX,OFFSET Stringa
12     0008      E8 0005      CALL STAMPA
13     000B      B8 4C00      MOV AX,4C00H
14     000E      CD 21      INT 21H
15     0010      BEGIN ENDP
16     0010      STAMPA PROC NEAR
17     0010      50      PUSH AX
18     0011      B4 09      MOV AH,09H
19     0013      CD 21      INT 21H
20     0015      58      POP AX
21     0016      C3      RET
22     0017      STAMPA ENDP
23          END BEGIN
```

### Symbol Table

Symbol Name	Type	Value	Cref (defined at #)
??DATE	Text	"01/05/01"	
??FILENAME	Text	"2 "	
??TIME	Text	"01:58:50"	
??VERSION	Number	030A	
@32BIT	Text	0	#2
@CODE	Text	_TEXT	#2 #2 #7
@CODESIZE	Text	0	#2
@CPU	Text	0101H	
@CURSEG	Text	_TEXT	#4 #7
@DATA	Text	DGROUP	#2 9
@DATASIZE	Text	0	#2
@FILENAME	Text	2	

```

@INTERFACE      Text    00H          #2
@MODEL         Text    2            #2
@STACK         Text    DGROUP       #2
@WORDSIZE      Text    2            #4 #7
BEGIN          Near    _TEXT:0000   #8 23
STAMPA         Near    _TEXT:0010   12 #16
STRINGA        Byte   DGROUP:0000  #5 11
Groups & Segments Bit Size    Align Combine    Class      Cref (defined at #)
DGROUP         Group
  STACK        16  0200    Para  Stack      STACK      #3
  _DATA        16  000D    Word  Public     DATA      #2 #4
  _TEXT        16  0017    Word  Public     CODE       #2 2 #7 7

```

## 2.ref

Turbo CREF Version 2.0 01/05/01 01:58:58 Page 1  
Global Symbol Name Cref (defined at #)

## 2.map

```

Start          Stop           Length        Name           Class
00000H         00026H         00027H        _TEXT          CODE
00030H         0003CH         0000DH        _DATA          DATA
00040H         0023FH         00200H        STACK          STACK
Program entry point at 0000:0010

```

## funz.c

```

#include <stdio.h>
#include <conio.h>
int Min (int a, int b);
int main (void)
{ int x=10,y=20;int z=20,k=30;
  clrscr();
  printf("Il minimo vale: %d\n", Min(x,y));
  printf("Il minimo vale: %d\n", Min(z,k));
  getch();return (0);
}
int Min (int a, int b)
{ if (a<b) return a;else return b;}

```

## funz.asm

```

_TEXT segment byte public use16 'CODE'
_TEXT ends
DGROUP      group _DATA,_BSS
            assume     cs:_TEXT,ds:DGROUP
_DATA segment word public use16 'DATA'
d@ label byte
d@w label word
_DATA ends
_BSS segment word public use16 'BSS'
b@ label byte
b@w label word
_BSS ends
_TEXT segment byte public use16 'CODE'
; int main (void)
assume      cs:_TEXT
_main proc near
push bp
mov bp,sp
sub sp,8
; { int x=10,y=20;int z=20,k=30;
mov word ptr [bp-2],10
mov word ptr [bp-4],20
mov word ptr [bp-6],20
mov word ptr [bp-8],30
; clrscr();

```

```

    call near ptr _clrscr
;   printf("Il minimo vale: %d\n", Min(x,y));
    push word ptr [bp-4]
    push word ptr [bp-2]
    call near ptr _Min
    add sp,4
    push ax
    push offset DGROUP:s@
    call near ptr _printf
    add sp,4
;   printf("Il minimo vale: %d\n", Min(z,k));
    push word ptr [bp-8]
    push word ptr [bp-6]
    call near ptr _Min
    add sp,4
    push ax
    push offset DGROUP:s@+20
    call near ptr _printf
    add sp,4
;   getch();return (0);
    call near ptr _getch
    xor ax,ax
; }
    leave
    ret
_main endp
; int Min (int a, int b)
    assume cs:_TEXT
_Min proc near
    push bp
    mov bp,sp
    mov dx,word ptr [bp+4]
    mov bx,word ptr [bp+6]
; { if (a<b) return a;else return b;}
    cmp dx,bx
    jge short @2@86
    mov ax,dx
    pop bp
    ret
@2@86:
    mov ax,bx
    pop bp
    ret
_Min endp
_TEXT ends
_DATA segment word public use16 'DATA'
s@ label byte
    db 'Il minimo vale: %d'
    db 10
    db 0
    db 'Il minimo vale: %d'
    db 10
    db 0
_DATA ends
_TEXT segment byte public use16 'CODE'
_TEXT ends
    public _main
    public _Min
    extrn _getch:near
    extrn _clrscr:near
    extrn _printf:near
_s@ equ s@
end

```

# MACRO

## 3.asm

```
DOSSEG
.MODEL SMALL
STACK 200H
.DATA
Stringa DB "Ciao, mondo",0DH,0AH,0
.CODE

BEGIN:
messag      MACRO etich
              LEA DX,etich
              MOV     AH,9
              INT     21H
endm

MOV AX,@data
MOV DS,AX
messag Stringa
MOV AX,4C00H
INT 21H

END BEGIN
```

## 3.lst

```
1          DOSSEG
2      0000      .MODEL SMALL
3      0000      STACK 200H
4      0000      .DATA
5      0000  43 69 61 6F 2C 20 6D+ Stringa DB "Ciao, mondo",0DH,0AH,0
6          6F 6E 64 6F 0D 0A 00
7      000E      .CODE
8      0000      BEGIN:
9          messag  MACRO etich
10             LEA DX,etich
11             MOV     AH,9
12             INT     21H
13         endm
14      0000  B8 0000s      MOV AX,@data
15      0003  8E D8      MOV DS,AX
16             messag Stringa
1  17      0005  BA 0000r      LEA DX,Stringa
1  18      0008  B4 09      MOV AH,9
1  19      000A  CD 21      INT 21H
20      000C  B8 4C00      MOV AX,4C00H
21      000F  CD 21      INT 21H
22             END BEGIN
```

### Symbol Table

Symbol Name	Type	Value	Cref (defined at #)
??DATE	Text	"01/05/01"	
??FILENAME	Text	"3 "	
??TIME	Text	"04:29:04"	
??VERSION	Number	030A	
@32BIT	Text	0	#2
@CODE	Text	_TEXT	#2 #2 #7
@CODESIZE	Text	0	#2
@CPU	Text	0101H	
@CURSEG	Text	_TEXT	#4 #7
@DATA	Text	DGROUP	#2 14
@DATASIZE	Text	0	#2
@FILENAME	Text	3	
@INTERFACE	Text	00H	#2
@MODEL	Text	2	#2
@STACK	Text	DGROUP	#2
@WORDSIZE	Text	2	#4 #7
BEGIN	Near	_TEXT:0000	#8 22
STRINGA	Byte	DGROUP:0000	#5 17

```

Macro Name Cref (defined at #)
MESSAG      #9  16
Groups & Segments Bit Size Align      Combine      Class      Cref (defined at #)
DGROUP      Group
  STACK      16  0200 Para      Stack      STACK      #3
  _DATA      16  000E Word      Public     DATA      #2  #4
  _TEXT      16  0011 Word      Public     CODE       #2  2  #7  7

```

### 3.ref

Turbo CREF Version 2.0 01/05/01 04:29:10 Page 1  
Global Symbol Name Cref (defined at #)

### 3.map

```

Start      Stop      Length      Name      Class
00000H     00020H     00021H     _TEXT     CODE
00030H     0003DH     0000EH     _DATA     DATA
00040H     0023FH     00200H     STACK     STACK
Program entry point at 0000:0010

```

### macro.c

```

#include <stdio.h>
#include <conio.h>
#define Min(a,b) ((a<b) ? a:b)
int main (void)
{ int x=10,y=20;int z=20,k=30;
  clrscr();
  printf("Il minimo vale: %d\n", Min(x,y));
  printf("Il minimo vale: %d\n", Min(z,k));
  getch();return (0);
}

```

### macro.asm

```

_TEXT segment byte public use16 'CODE'
_TEXT ends
DGROUP      group _DATA,_BSS
            assume      cs:_TEXT,ds:DGROUP
_DATA segment word public use16 'DATA'
d@ label byte
d@w label word
_DATA ends
_BSS segment word public use16 'BSS'
b@ label byte
b@w label word
_BSS ends
_TEXT segment byte public use16 'CODE'
; int main (void)
assume      cs:_TEXT
_main proc near
push bp
mov bp,sp
sub sp,4
push si
push di
; { int x=10,y=20;int z=20,k=30;
mov si,10
mov di,20
mov word ptr [bp-2],20
mov word ptr [bp-4],30
; clrscr();
call near ptr _clrscr
; printf("Il minimo vale: %d\n", Min(x,y));
cmp si,di
jge short @1@86
mov ax,si

```

```

        jmp     short @1@114
@1@86:   mov     ax,di
@1@114:  push   ax
        push  offset DGROUP:s@
        call  near ptr _printf
        add   sp,4
        ;    printf("Il minimo vale: %d\n", Min(z,k));
        mov   ax,word ptr [bp-2]
        cmp   ax,word ptr [bp-4]
        jge   short @1@170
        jmp   short @1@198
@1@170:  mov     ax,word ptr [bp-4]
@1@198:  push   ax
        push  offset DGROUP:s@+20
        call  near ptr _printf
        add   sp,4
        ;    getch();return (0);
        call  near ptr _getch
        xor   ax,ax
        ;    }
        pop   di
        pop   si
        leave
        ret
_main   endp
_TEXT  ends
_DATA  segment word public use16 'DATA'
s@     label byte
        db   'Il minimo vale: %d'
        db   10
        db   0
        db   'Il minimo vale: %d'
        db   10
        db   0
_DATA  ends
_TEXT  segment byte public use16 'CODE'
_TEXT  ends
        public      _main
        extrn _getch:near
        extrn _clrscr:near
        extrn _printf:near
_s@    equ   s@
end

```

## PROGRAMMAZIONE MODULARE C-ASM

### showtot.c

```
#include <stdio.h>
#include <conio.h>
extern int StartingValue;
extern int DoTotal (void);
int Repetitions;
int main (void)
{ int i;clrscr();Repetitions=10; StartingValue = 2;
  printf("Valore = %d\n", DoTotal());
  getch();return (0);
}
```

### show\_1.asm

```
DOSSEG
.MODEL SMALL
.DATA
  EXTRN  _Repetitions:WORD      ;definito esternamente
  PUBLIC _StartingValue        ;disponibile per altri moduli
_StartingValue DW 0
.DATA?
RunningTotal   DW ?
.CODE
PUBLIC  _DoTotal
_DoTotal      PROC
  mov     cx,[_Repetitions]      ;# di conteggi da eseguire
  mov     ax,[_StartingValue]
  mov     [RunningTotal],ax      ;imposta il valore iniziale
TotalLoop:
  inc     [RunningTotal]         ;RunningTotal++
  loop   TotalLoop
  mov     ax,[RunningTotal]      ;valore di ritorno
  ret
_DoTotal      ENDP
END
```

### show\_1.lst

```
1          DOSSEG
2      0000          .MODEL  SMALL
3      0000          .DATA
4          EXTRN    _Repetitions:WORD
5          PUBLIC  _StartingValue
6      0000  0000    _StartingValue DW 0
7      0002          .DATA?
8      0000  ?????    RunningTotal   DW ?
9      0002          .CODE
10         PUBLIC  _DoTotal
11      0000          _DoTotal      PROC
12      0000  8B 0E 0000e    mov cx,[_Repetitions]
13      0004  A1 0000r    mov ax,[_StartingValue]
14      0007  A3 0000r    mov [RunningTotal],ax
15      000A          TotalLoop:
16      000A  FF 06 0000r    inc [RunningTotal]
17      000E  E2 FA          loop  TotalLoop
18      0010  A1 0000r    mov ax,[RunningTotal]
19      0013  C3          ret
20      0014          _DoTotal      ENDP
21      0014          END
```

### Symbol Table

Symbol Name	Type	Value	Cref (defined at #)
??DATE	Text	"01/05/01"	
??FILENAME	Text	"show_1"	"
??TIME	Text	"10:43:42"	
??VERSION	Number	030A	

@32BIT	Text	0				#2			
@CODE	Text	_TEXT				#2	#2	#9	
@CODESIZE	Text	0				#2			
@CPU	Text	0101H							
@CURSEG	Text	_TEXT				#3	#7	#9	
@DATA	Text	DGROUP				#2			
@DATASIZE	Text	0				#2			
@FILENAME	Text	SHOW_1							
@INTERFACE	Text	00H				#2			
@MODEL	Text	2				#2			
@STACK	Text	DGROUP				#2			
@WORDSIZE	Text	2				#3	#7	#9	
RUNNINGTOTAL	Word	DGROUP:0000				#8	14	16	18
TOTALLOOP	Near	_TEXT:000A				#15	17		
_DOTOTAL	Near	_TEXT:0000				10	#11		
_REPETITIONS	Word	DGROUP:---- Extern				#4	12		
_STARTINGVALUE	Word	DGROUP:0000				5	#6	13	
Groups & Segments	Bit	Size	Align	Combine	Class	Cref (defined at #)			
DGROUP	Group					#2	2		
_BSS	16	0002	Word	Public	BSS	#7			
_DATA	16	0002	Word	Public	DATA	#2	#3		
_TEXT	16	0014	Word	Public	CODE	#2	2	#9	9

### show\_1.ref

Turbo CREF Version 2.0 01/05/01 10:43:53 Page 1

Global Symbol Name	Cref (defined at #)
_DOTOTAL	# SHOW_1.ASM: 10 #11
_REPETITIONS	SHOW_1.ASM: 4 12
_STARTINGVALUE	# SHOW_1.ASM: 5 #6 13

# PROGRAMMAZIONE MODULARE ASM

## modulia2.asm

```
D_SEG SEGMENT
ascii db 8H DUP (?)
D_SEG ENDS
S_SEG SEGMENT STACK
db 200 dup ("stack")
S_SEG ENDS
C_SEG SEGMENT
ASSUME CS:C_SEG, DS:D_SEG, SS:S_SEG
INCLUDE servizio
EXTRN var1:BYTE, var2:WORD
EXTRN modulo:FAR
start: mov ax,SEG D_SEG
mov DS,ax
ciclo: CALL tastiera
CMP AL,'Q' ;se premi Q esce
JE fine
MOV BX, SEG var1 ;punta al segmento di var1 e var2
MOV ES, BX
MOV ES:[var1], AL
MOV ES:[var2], OFFSET ascii
MOV ES:[(var2+2H)], SEG ascii
CALL modulo
CALL video ;visualizzazione del carattere ASCII
MOV AL, 20H ;ritorno a capo
CALL video
MOV SI, 0H
ripeti: MOV AL, DS:[ascii+SI] ;prepara visualizzazione cod ASCII
CALL video
INC SI
CMP SI, 8H ;verifica se ho visualizzato 8 caratteri ASCII
JB ripeti ;..se no ripeti
MOV AL, 0DH ;vai a capo
CALL video
MOV AL, 0AH ;salta una riga
CALL video
JMP ciclo
fine: JMP dos
C_SEG ends
end start
```

## servizio

```
; Contenuto Del File servizio
tastiera PROC FAR
PUSH BX
MOV BX,AX
MOV AH,00H
INT 16H
MOV AH,BH
POP BX
RET
tastiera ENDP
video PROC FAR
PUSH AX
PUSH DX
MOV AH,02H
MOV DL,AL
INT 21H
POP DX
POP AX
RET
video ENDP
dos LABEL FAR
```

```
MOV AH,4CH
INT 21H
```

### modulia2.lst

```

1      0000          D_SEG  SEGMENT
2      0000  08*(??)      ascii  db      8H DUP (?)
3      0008          D_SEG  ENDS
4      0000          S_SEG  SEGMENT      STACK
5      0000  C8*(73 74   61 63 6B)      db      200      dup ("stack")
6      03E8          S_SEG  ENDS
7      0000          C_SEG  SEGMENT
8          ASSUME CS:C_SEG, DS:D_SEG,      SS:S_SEG
9          INCLUDE servizio
1     10          ; Contenuto Del File servizio
1     11
1     12      0000          tastiera      PROC      FAR
1     13      0000  53          PUSH      BX
1     14      0001  8B D8      MOV       BX,AX
1     15      0003  B4 00      MOV       AH,00H
1     16      0005  CD 16      INT      16H
1     17      0007  8A E7      MOV       AH,BH
1     18      0009  5B          POP       BX
1     19      000A  CB          RET
1     20      000B          tastiera      ENDP
1     21
1     22      000B          video      PROC      FAR
1     23      000B  50          PUSH      AX
1     24      000C  52          PUSH      DX
1     25      000D  B4 02      MOV       AH,02H
1     26      000F  8A D0      MOV       DL,AL
1     27      0011  CD 21      INT      21H
1     28      0013  5A          POP       DX
1     29      0014  58          POP       AX
1     30      0015  CB          RET
1     31      0016          video      ENDP
1     32
1     33      0016          dos      LABEL      FAR
1     34      0016  B4 4C      MOV       AH,4CH
1     35      0018  CD 21      INT      21H
1     36
1     37          ;utility      ENDS
1     38
1     39
40          EXTRN      var1:BYTE, var2:WORD
41          EXTRN      modulo:FAR
42
43      001A  B8 0000s      start:  mov      ax,SEG D_SEG
44      001D  8E D8          mov      DS,ax
45      001F  0E E8 FFDD      ciclo:  CALL     tastiera
46      0023  3C 51          CMP     AL,'Q'      ;se premi Q esce
47      0025  74 44          JE      fine
48      0027  BB 0000s          MOV     BX, SEG var1
49      002A  8E C3          MOV     ES, BX
50      002C  26: A2 0000e          MOV     ES:[var1], AL
51      0030  26: C7 06 0000e  +      MOV     ES:[var2], OFFSET ascii
52      0000r
53      0037  26: C7 06 0002e  +      MOV     ES:[(var2+2H)], SEG ascii
54      0000s
55      003E  0E E8 0000e          CALL   modulo
56      0042  0E E8 FFC5          CALL   video
;visualizzazione del carattere ASCII
57      0046  B0 20          MOV     AL, 20H
58      0048  0E E8 FFBF          CALL   video
59      004C  BE 0000          MOV     SI, 0H
60      004F  8A 84 0000r      ripeti: MOV     AL, DS:[ascii+SI]
```

```

61      0053  0E E8 FFB4          CALL    video
62      0057  46                   INC     SI
63      0058  83 FE 08          CMP     SI, 8H
64      005B  72 F2          JB     ripeti
65      005D  B0 0D          MOV    AL, 0DH          ;vai a capo
66      005F  0E E8 FFA8        CALL    video
67      0063  B0 0A          MOV    AL, 0AH
68      0065  0E E8 FFA2        CALL    video
69      0069  EB B4          JMP    ciclo
70      006B  EB A9          fine:  JMP    dos
71      006D                    C_SEG  ends
72                    end start

```

#### Symbol Table

Symbol Name	Type	Value	Cref (defined at #)			
??DATE	Text	"01/05/01"				
??FILENAME	Text	"modulia2"				
??TIME	Text	"11:11:11"				
??VERSION	Number	030A				
@CPU	Text	0101H				
@CURSEG	Text	C_SEG	#1 #4 #7			
@FILENAME	Text	MODULIA2				
@WORDSIZE	Text	2	#1 #4 #7			
ASCII	Byte	D_SEG:0000	#2 51 53 60			
CICLO	Near	C_SEG:001F	#45 69			
DOS	Far	C_SEG:0016	#33 70			
FINE	Near	C_SEG:006B	47 #70			
MODULO	Far	C_SEG:---- Extern	#41 55			
RIPETI	Near	C_SEG:004F	#60 64			
START	Near	C_SEG:001A	#43 72			
TASTIERA	Far	C_SEG:0000	#12 45			
VAR1	Byte	C_SEG:---- Extern	#40 48 50			
VAR2	Word	C_SEG:---- Extern	#40 51 53			
VIDEO	Far	C_SEG:000B	#22 56 58 61 66 68			
Groups & Segments	Bit	Size	Align	Combine	Class	Cref (defined at #)
C_SEG	16	006D	Para	none		#7 8
D_SEG	16	0008	Para	none		#1 8 43
S_SEG	16	03E8	Para	Stack		#4 8

#### modulia2.ref

Turbo CREF Version 2.0 01/05/01 11:11:26 Page 1

Global Symbol Name	Cref (defined at #)
MODULO	MODULIA2.ASM: 11 23
VAR1	MODULIA2.ASM: 10 18 20
VAR2	MODULIA2.ASM: 10 21 22

#### modulia2.map

Start	Stop	Length	Name	Class
00000H	00007H	00008H	D_SEG	
00010H	003F7H	003E8H	S_SEG	
00400H	0046CH	0006DH	C_SEG	
00470H	004A4H	00035H	C_SEG	
004B0H	004B4H	00005H	VAR	

Program entry point at 0040:001°

#### modulib2.asm

```

var      SEGMENT
Assume CS:C_seg, DS:VAR
        PUBLIC  var1, var2
var1     db     ?
var2     dw     2 DUP (?)
var      ENDS
C_SEG    SEGMENT
        PUBLIC modulo
modulo   PROC    FAR

```

```

PUSH    AX
PUSH    BX
PUSH    SI
PUSH    DS
PUSH    ES
MOV     AX, SEG var
MOV     DS, AX
MOV     AL, DS:[var1]
MOV     BX, DS:[var2]
MOV     ES, DS:[(var2+2)]
MOV     SI, 0H
ciclo: TEST    AL, 80H
        JZ     zero
        MOV    byte ptr ES:[BX+SI], '1'
        JMP    avanti
zero:   MOV    byte ptr ES:[BX+SI], '0'
avanti: SHL    AL, 1
        INC    SI
        CMP    SI, 8H
        JB     ciclo
        POP    ES
        POP    DS
        POP    SI
        POP    BX
        POP    AX
        RET
modulo ENDP
C_SEG  ENDS
        END

```

#### modulib2.lst

```

1      0000          var      SEGMENT
2
3
4      0000  ??          var1    db      ?
5      0001  02*(????)   var2    dw      2 DUP (?)
6      0005
7      0000
8
9      0000          modulo   PUBLIC modulo
10     0000  50          modulo  PROC    FAR
11     0001  53          PUSH    BX
12     0002  56          PUSH    SI
13     0003  1E          PUSH    DS
14     0004  06          PUSH    ES
15     0005  B8 0000s    MOV     AX, SEG var
16     0008  8E D8       MOV     DS, AX
17     000A  A0 0000r    MOV     AL, DS:[var1]
18     000D  8B 1E 0001r    MOV     BX, DS:[var2]
19     0011  8E 06 0003r    MOV     ES, DS:[(var2+2)]
20     0015  BE 0000       MOV     SI, 0H
21     0018  A8 80          ciclo: TEST    AL, 80H
22     001A  74 07          JZ     zero
23     001C  26: C6 00  31    MOV    byte ptr ES:[BX+SI], '1'
24     0020  EB 05 90          JMP    avanti
25     0023  26: C6 00  30    zero: MOV    byte ptr ES:[BX+SI], '0'
26     0027  D0 E0          avanti: SHL    AL, 1
27     0029  46          INC    SI
28     002A  83 FE 08       CMP    SI, 8H
29     002D  72 E9          JB     ciclo
30     002F  07          POP    ES
31     0030  1F          POP    DS
32     0031  5E          POP    SI
33     0032  5B          POP    BX
34     0033  58          POP    AX

```

```

35      0034  CB                      RET
36      0035                      modulo ENDP
37      0035                      C_SEG  ENDS
38                                END

```

Symbol Table

Symbol Name	Type	Value	Cref (defined at #)			
??DATE	Text	"01/05/01"				
??FILENAME	Text	"modulib2"				
??TIME	Text	"11:11:19"				
??VERSION	Number	030A				
@CPU	Text	0101H				
@CURSEG	Text	C_SEG	#1 #7			
@FILENAME	Text	MODULIB2				
@WORDSIZE	Text	2	#1 #7			
AVANTI	Near	C_SEG:0027	24 #26			
CICLO	Near	C_SEG:0018	#21 29			
MODULO	Far	C_SEG:0000	8 #9			
VAR1	Byte	VAR:0000	3 #4 17			
VAR2	Word	VAR:0001	3 #5 18 19			
ZERO	Near	C_SEG:0023	22 #25			
Groups & Segments	Bit	Size	Align	Combine	Class	Cref (defined at #)
C_SEG	16	0035	Para	none		2 #7
VAR	16	0005	Para	none		#1 2 15

modulib2.ref

Turbo CREF Version 2.0 01/05/01 11:11:35 Page 1

Global Symbol Name	Cref (defined at #)
MODULO	# MODULIB2.ASM: 8 #9
VAR1	# MODULIB2.ASM: 3 #4 17
VAR2	# MODULIB2.ASM: 3 #5 18 19

## LIBRERIE PERSONALI

La soluzione di collocare in una libreria un certo numero di macro e routine di utilizzo frequente (che il programmatore può invocare tramite un semplice richiamo, evitando così di dover codificare ogni macro e routine esplicitamente all'interno del programma stesso) ottimizza il tempo di sviluppo del programma.

Una libreria di macro e routine contiene solo codice sorgente, per cui l'unica operazione da eseguire è quella di salvare il file ASCII, al termine della scrittura delle macro.

Per includere nel programma la libreria di macro si utilizza la direttiva

### **Include**

bisogna assemblare tutte le volte che si riassume il programma principale, inserisce tutto il file anche se si usa una sola macro o routine, non esiste privatezza dei sorgenti.

Le macro e routine realizzate nel corso dello sviluppo di un programma si rivelano essere sottoprogrammi di utilità generale che possono quindi essere riutilizzati da molti altri programmi.

In questi casi è conveniente costruire delle librerie di macro e routine già assemblate da poter richiamare, all'occorrenza, all'interno dei programmi, lasciando al linker l'incombenza di effettuare i dovuti collegamenti.

Le librerie di macro e routine sono quindi una collezione di sottoprogrammi raggruppate solitamente per argomenti omogenei, possono essere:

.OBJ: i moduli sono separati per cui si evitano compilazioni inutili, si ha la privatezza dei sorgenti, ma la compilazione inserisce tutto il file anche se si usa una sola routine.

Tlib: è un'utility che gestisce librerie di file .OBJ individuali, utile per creare una nuova libreria da un gruppo di .OBJ;

aggiungere .OBJ o altri .LIB ad una libreria esistente;

rimuovere, sostituire, elencare ed estrarre .OBJ da una libreria esistente.

Per costruire una libreria di routine in assembler è sufficiente raggruppare le routine stesse in un unico file sorgente, rispettando il seguente formalismo.

```
CODE SEGMENT
    PUBLIC nome_procl, nome_proc2, ..., nome_procn
    ASSUME CS:CODE
nome_procl PROC FAR
    ...
nome_procl ENDP
nome_proc2 PROC FAR
    ...
nome_proc2 ENDP
...
nome_procn PROC FAR
    ...
nome_procn ENDP
CODE ENDS
    END
```

Una volta completata la scrittura del codice delle macro e routine, basta convertire il file sorgente nel corrispondente file oggetto.

Per costruire una libreria di funzioni in BorlandC è sufficiente raggruppare le funzioni stesse in un unico file sorgente, rispettando il seguente formalismo.

```
void func1 (void { printf("funzione 1\n"); } /*test.c*/  
void func2 (void { printf("funzione 2\n"); }
```

### Ambiente a linea di comando

- c:\>bcc -c test; per creare il modulo oggetto TEST.OBJ
- c:\>tlib provalib +test per creare la libreria di nome PROVALIB.LIB;
- c:\>tlib provalib, provalib.lst per ottenere un listato della libreria PROVALIB.LIB;
- c:\>tilib -+test sostituisce TEST.OBJ con una nuova copia.

```
#include <stdio.h> /*prova.c*/  
#include <conio.h>  
#include "prova.h"  
int main (void)  
{  
    clrscr();  
    func1();  
    func2();  
    getch();  
    return(0);  
}
```

```
void func1(void); /*prova.h*/  
void func2(void);
```

### Ambiente IDE

prova.prj = prova + provalib

### Ambiente a linea di comando

c:\> bcc prova provalib

UBERTINI MASSIMO

<http://www.ubertini.it>  
massimo@ubertini.it

Dip. Informatica Industriale  
I.T.I.S. "Giacomo Fauser"  
Via Ricci, 14  
28100 Novara Italy

tel. +39 0321482411  
fax +39 0321482444

<http://www.fauser.edu>  
<http://www.fauser.edu/fau/sistem.htm>  
massimo@fauser.edu

Massimo Ubertini