DIRETTIVE DI PREPROCESSORE

- **elaborazione di macro**
- **inclusione di file sorgente**
- **compilazione condizionale (consente di compilare porzioni di codice sorgente in dipendenza del valore di una espressione aritmetica)**

Tutte le direttive iniziano con # (diesis detto anche cancelletto)

```
#include <stdio.h>
#include <string.h>
 /* se il nome di una macro é molto lungo  \
 può andare su più linee a patto di terminare la linea con un \ (backslash) */
#define ESEMPIO "Questa definizione \
non serve a niente, ma solo a esemplificare come andare \
a capo quando la definizione é troppo lunga."
int main(void)
{int j=0; char s[] = "prova";
# define LUNG_MAX 10
/* le #define, a differenza delle #include, possono comparire ovunque nel programma */
if (strlen(s) < LUNG_MAX )
printf("La stringa s é %s e la sua lunghezza é minore di LUNG_MAX = %d.\n",s, LUNG_MAX);
if (strlen(ESEMPIO) > LUNG_MAX)
printf("La stringa ESEMPIO = \"%s\"  é troppo lunga.\n",ESEMPIO);
return 0;
}
```

OUTPUT:

```
La stringa s é prova e la sua lunghezza é minore di LUNG_MAX =
10.
La stringa ESEMPIO = "Questa definizione non serve a niente, ma
solo a esemplificare come andare a capo quando la definizione é
troppo lunga."  é troppo lunga.
```

Questioni di *STILE* nell'elaborazione di macro (e non)

- PREMETTERE le direttive del preprocessore alle altre istruzioni ne facilita l'individuazione

- Usare le MAIUSCOLE per i nome delle macro è una convenzione comune che ne facilita l'individuazione

- Usare SEMPRE nomi che evocano il significato degli oggetti nominati in modo accurato
  (Nomi non ben scelti possono indurre errori!!)

- Usare le macro per definire delle COSTANTI SIMBOLICHE facilita eventuali modifiche riadattive del programma

L’applicazione dei consigli di stile all’esempio precedente

```
#include <stdio.h>
#include <string.h>
#define ESEMPIO "Questa definizione \
non serve a niente, ma solo a esemplificare come andare \
a capo quando la definizione é troppo lunga.”
 # define LUNG_MAX 10
*/ # define LUNG_MAX 10  /* LUNG_MAX in un altro contesto   potrebbe essere 100!*/

int main(void)
{int j=0;
char s[] = "prova";
if (strlen(s) < LUNG_MAX )
printf("La stringa s é %s e la sua lunghezza é minore di LUNG_MAX = %d.\n",s, LUNG_MAX);
if (strlen(ESEMPIO) > LUNG_MAX)
printf("La stringa ESEMPIO = \"%s\"  é troppo lunga.\n",ESEMPIO);
return 0;
}
```

COSTANTI SIMBOLICHE PREDEFINITE

Questa macro

__DATE__ è la data nel momento in cui il file è compilato

__FILE__ è il nome del file che viene compilato

__LINE__ è il numero della linea che si sta compilando,
ottenuto prima dell'inclusione di ogni file header

__TIME__ è il momento in cui il file è compilato

__STDC__ vale un intero ≠0 se il compilatore è un compilatore ANSI C

ESEMPIO DI USO DI COSTANTI SIMBOLICHE PREDEFINITE

```
#include <stdio.h>

int main(void)
{
if( __STDC__ )
printf("è un compilatore  ISO C (ANSI C) \n");
else printf("non è un compilatore  ISO C \n");
printf("oggi %s, alle %s\n", __DATE__ , __TIME__ );
printf("nome del file = %s, siamo alla linea: %d\n",  __FILE__ , __LINE__ );
return 0;
}
```

OUTPUT:

```
è un compilatore  ANSI C
oggi Sep 25 2002, alle 17:46:33
nome del file = MacroPredef.c, siamo alla linea: 9
```

MACRO CON PARAMETRI

```
#include <stdio.h>
#define TO_LOWER(c) ((c) + ('a' - 'A'))

int main(void)
{char p = 'F',q;
printf("\nIl carattere in p è %c\n",p);
q = TO_LOWER(p);
printf("\nIl carattere in q è %c\n",q);
return 0;
}
```

OUTPUT:

```
Il carattere in p è F

Il carattere in q è f
```

Se **c >= 'A' && c <= 'Z', c - 'A'** calcola la distanza di c dal primo carattere maiuscolo e aggiungendo 'a' otteniamo il valore ASCI del carattere minuscolo corrispondente, altrimenti il risultato non è prevedibile.

ATTENZIONE ALLE PARENTESI!

```
#include <stdio.h>
/* le diverse definizioni di square illustrano la necessità delle parentesi
*nelle definizioni di macro con argomenti */
#define square(a) ((a)*(a))
#define square2(a) (a*a)
#define square3(a) a*a
int main(void)
{int j=0;
j = 2 * square(3+4);
printf(" \nQui j = 2* square(3+4) = 2*((3+4)*(3+4))= %d\n",j);

j = 2 * square2(3+4);
printf("\nQui j = 2* square2(3+4) = 2*(3+4*3+4) = 2*(3+(4*3)+4)= %d\n",j);

j = 2 * square3(3+4);
printf("\n Qui j = 2* square3(3+4) = 2*3+4*3+4=(2*3) + (4*3) + 4=%d\n",j);
return 0;}
```

OUTPUT:

```
Qui j = 2* square(3+4) = 2*((3+4)*(3+4)) = 98

Qui j =  2* square2(3+4) = 2*(3+4*3+4) = 2*(3+(4*3)+4) = 38

Qui j = 2* square3(3+4) = 2*3+4*3+4=(2*3) + (4*3) + 4 = 22
```

ATTENZIONE AGLI EFFETTI COLLATERALI!

```
#include <stdio.h>
/* la seguente definizione illustra il risultato di effetti collaterali non voluti
*nelle definizioni di macro con argomenti */
#define MIN(a,b)  ((a) < (b) ? (a) : (b))

 int main(void)

 {int j=0, p = 3, q = 7, r = 4,t = 8;


 j = MIN(p+q,r+t);
 printf("\nQui j = ((p+q) < (r+t) ? (p+q): (r+t)) e il suo valore, corretto, é   %d\n",j);

 j = MIN(p++,r);
 printf("\nQui j = min(p++,r)=((p++) < (r) ? (p++): (r)) e il suo valore é   %d\n",j);
 printf("\nIl  valore di p é %d (e non è quello che si vuole)  e quello di r é %d \n",p,r);
 return 0;}
```

OUTPUT:

```
Qui j = ((p+q) < (r+t) ? (p+q): (r+t) e il suo valore, corretto, é   10

Qui j = min(p++,r)=((p++) < (r) ? (p++): (r) e il suo valore é   4

Il  valore di p é 5 (e non è detto sia quello che si vuole)…
```

ATTENZIONE AGLI ANNIDAMENTI!

```
#include <stdio.h>
/* esempio di comportamento errato dovuto all'annidamento /
di macro con effetti collaterali*/

#define IS_UPPER(c) ((c) >='A' && (c) <='Z')

int main(void)
{char ch ;
while (IS_UPPER(ch = getchar()))
printf("il carattere prelevato é %c\n",ch);
/* IS_UPPER(ch = getchar()) = ((getchar()) >='A' && (getchar()) <='Z') */
/* per cui il primo carattere è confrontato con 'A' e
/* allora un altro carattere è prelevato e confrontato con 'Z'*/
return 0;}
```

INPUT: ABCDEF **OUTPUT:**

```
il carattere prelevato é B
il carattere prelevato é D
il carattere prelevato é F
```

Versione corretta del precedente esempio

```
#include <stdio.h>
#define IS_UPPER(c) ((c) >='A' && (c) <='Z')

int main(void)
{char ch ;
while ((ch = getchar()) != EOF && IS_UPPER(ch))
printf("il carattere prelevato é %c\n",ch);
return 0;}
```

INPUT: ABCDE

OUTPUT:

```
il carattere prelevato é A
il carattere prelevato é B
il carattere prelevato é C
il carattere prelevato é D
il carattere prelevato é E
```

MACRO, in ANSI C _tolower()

```
#define TO_LOWER(c) ((c) + ('a' - 'A'))
```

FUNZIONE ANSI C

```
char to_lower(char c)
{if (c >= 'A' && c <= 'Z' )
(return c + ('a' - 'A'); else return c;}
```

VANTAGGI MACRO

• Sono state introdotte perché la loro valutazione è più veloce delle funzioni, infatti si evitano le complesse operazioni collegate alla chiamata di una funzione (con i moderni calcolatori e compilatori la differenza è sempre meno rilevabile)

•C'è un controllo sul numero degli argomenti, (come nel caso di funzioni con prototipo)

•Gli argomenti non hanno un tipo, quindi si adattano a diversi tipi di dato. (non c'è il controllo sui tipi (type checking))

SVANTAGGI MACRO

• Effetti collaterali negli argomenti

• Le macro sono sostituite ovunque nel testo quindi il codice risultante può essere significativamente più lungo di un codice equivalente che usa una funzione (compilata una volta sola),

• debugging più complesso perchè il codice oggetto è più distante dal sorgente

CONCLUSIONE

SE NON CI SONO VINCOLI STRINGENTI DI COMPLESSITA', MEGLIO EVITARE LE MACRO CON PARAMETRI E USARE AL LORO POSTO DELLE FUNZIONI

Ma alcune funzioni di libreria di I/O sono implementate come macro:

getchar, isdigit, putchar ...