

# Macro e Compilazione condizionale)

---



Gianpiero Cabodi  
Dip. Automatica e Informatica  
Politecnico di Torino



# Direttive al pre-compilatore C: #define

---

- Per costanti simboliche (già noto):  
identificatore equivalente a costante

```
#define PI 3.1415926535
```

- Per **"macro"** con argomenti:

```
#define sum2(a,b) ((a)+(b))
```

```
#define min(x,y) ((x)<(y)?(x):(y))
```

```
#define swapInt(x,y) {\n    int t=x; x=y; y=t; }
```



# A cosa servono le macro con argomenti ?

---

Realizzare “funzioni” per sostituzione, invece che chiamata esplicita

- Vantaggio: velocità (in pratica con c'è una funzione)
- Svantaggio: Non c'è compattazione del programma
  - funzione scritta una sola volta e chiamata più volte
  - ogni chiamata di macro sarà sostituita dalle istruzioni interne

Alternativa nelle versioni recenti di C (es. C99 C11)

- inline function



# inline function: esempi

---

```
inline int sum2(int a, int b) {  
    return a+b;  
}
```

```
inline int min(int x, int y) {  
    return (x<y)?x:y;  
}
```

```
inline void swapInt(int *xp, int *yp) {  
    int t = *x;  
    *x=*y; *y=t;  
}
```



# inline function: esempi

---

```
inline int sum2(int a, int b) {  
    return a+b;  
}
```

```
inline int min(int x, int y) {  
    return (x < y ? x : y;  
}
```

```
int  
  
}
```

Si tratta di normali funzioni precedute dalla parola chiave inline

=> CONSIGLIO al compilatore:

“Se utile, EVITARE vera funzione, ma Sostituire con istruzioni”

Spesso scritte nei .h (come le macro)



# Direttive al pre-compilatore C: #define

---

```
#define sum2(a,b) ((a)+(b))
```

```
...
```

```
int i1, i2, i3;
```

```
float f1, f2, f3;
```

```
...
```

```
i3 = sum2(i1,i2);
```

```
f3 = sum2(f1,f2);
```

# Direttive al pre-compilatore C: #define

```
#define sum2  
...
```

```
int i1, i2, i3;  
float f1, f2, f3;  
...  
i3 = sum2(i1, i2);  
f3 = sum2(f1, f2);
```

**sum2 simile a funzione  
ma senza tipo  
(va bene con int e float)**

# Direttive al pre-compilatore C: #define

```
#define sum2(a,b) ((a)+(b))
```

```
...
```

```
int i1, i2, i3;
```

```
float f1, f2, f3
```

```
...
```

```
i3 = sum2(i1,i2);
```

```
f3 = sum2(f1,f2);
```

**Non e' una funzione  
Si tratta di sostituzione**





# Direttive al pre-compilatore C: #define

```
#define sum2(a,b) ((a)+(b))
```

```
...
```

```
int i1, i2, i3;
```

```
float f1, f2, f3
```

```
...
```

```
i3 = ((i1)+(i2))
```

```
f3 = ((f1)+(f2));
```

**Non e' una funzione  
Si tratta di sostituzione**



# Macro: perché le parentesi ?

---

```
#define sum2(a,b) a+b
```

```
...
```

```
int i1, i2, i3;
```

```
float f1, f2, f3;
```

```
...
```

```
i3 = 3*sum2(i1,i2);
```

```
f3 = sum2(f2,f3)/2;
```



# Macro: perché le parentesi ?

```
#define sum2(a,b) a+b
```

```
...
```

```
int i1, i2;
```

```
float f1, f2, f3;
```

```
...
```

```
i3 = 3*sum2(i1, i2);
```

```
f3 = sum2(f2, f3) / 2;
```

**Problema: precedenze operatori**

**i3 = 3\*i1+i2;**

**f3 = f2+f3/2;**



# Macro: perché le parentesi ?

---

```
#define mul2(a,b) (a*b)
```

```
...
```

```
float f1, f2, f3, f4;
```

```
...
```

```
f3 = 2*mul2(f1+f2, f3);
```



# Macro: perché le parentesi ?

```
#define mul2(a,b) (a*b)
```

```
...
```

**Problema: precedenze operatori**  
**f3 = 2\*(f1+f2\*f3);**

```
float f1, f2, f4;
```

```
...
```

```
f3 = 2*mul2(f1+f2, f3);
```



# Macro: perché le parentesi ?

---

```
#define mul2(a,b) ((a)*(b))
```

```
...
```

OK con più parentesi...

```
f3 = 2*((f1+f2)*(f3));
```

```
float f1, f2, f4;
```

```
...
```

```
f3 = 2*mul2(f1+f2, f3);
```



# Macro con espressioni condizionali

---

La Macro NON ha return.

Quindi espressioni condizionali utili

```
#define min(x,y) ((x)<(y)?(x):(y))
```

```
...
```

```
float f1, f2, f3;
```

```
...
```

```
f3 = min(f1, f2) ;
```



# Macro con espressioni condizionali

---

La Macro NON ha return.

Quindi espressioni condizionali utili

```
#define min(x,y) ((x)<(y)?(x):(y))
```

```
...
```

```
float f1, f2, f3;
```

```
...
```

```
f3 = ((f1)<(f2)?(f1):f2);
```





# Macro con espressioni condizionali

La Macro NON ha return.

Quindi espressioni condizionali utili

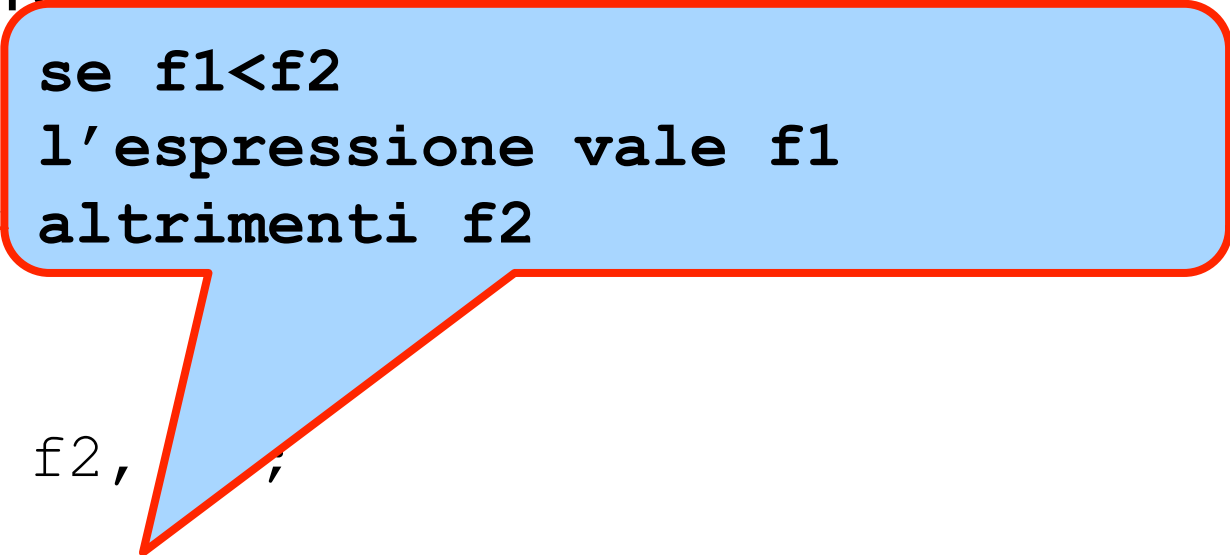
```
#define m
```

```
...
```

```
float f1, f2, ;
```

```
...
```

```
f3 = ((f1)<(f2)?(f1):f2);
```



se  $f1 < f2$   
l'espressione vale  $f1$   
altrimenti  $f2$

# Macro con espressioni condizionali

La Macro M

Quindi esp

```
#define m
```

...

```
float f1, f2,
```

...

```
f3 = ((f1)<(f2)?(f1):f2);
```

equivale a:

```
if (f1<f2)
```

```
    f3 = f1;
```

```
else
```

```
    f3 = f2;
```



# Macro e parametri

---

Parametri né per valore né per riferimento.

## SOSTITUZIONE

- Si possono ri-assegnare

```
#define swapInt(x,y) {\n    int t=x; x=y; y=t; }
```

...

```
int x1, x2;
```

...

```
swapInt(x1,x2);
```



# Macro e parametri

---

Parametri né per valore né per riferimento.

## SOSTITUZIONE

- Si possono ri-assegnare

```
#define swapInt(x, y) {\n    int t=x; x=y; y=t; }
```

...

```
int x1, x2;
```

...

```
{int t=x1; x1=x2; x2=t; };
```



# Direttive al pre-compilatore C: compilazione condizionale

---

Espressione condizionale valutata dal compilatore

```
#define DBG 1  
  
...  
    scanf ("%d", &x) ;  
#if DBG  
    printf ("Ho letto: %d\n", x) ;  
#endif
```

# Direttive al pre-compilatore C: compilazione condizionale

Espressione condizionale

**L'istruzione viene compilata**

```
#define DBG 1
```

```
...
```

```
scanf ("%d", &x)
```

```
printf("Ho letto: %d\n", x);
```

# Direttive al pre-compilatore C: compilazione condizionale

Espressione

L'istruzione NON viene compilata

```
#define DBG 0
```

```
...
```

```
scanf ("%d", &x)
```

```
printf("Ho letto: %d\n", x);
```



# Direttive al pre-compilatore C: varianti

---

`#ifdef, #ifndef`: non è necessario il valore  
(opzionale)

```
#define DBG
```

```
...
```

```
    scanf ("%d", &x) ;
```

```
#ifdef DBG
```

```
    printf ("Ho letto: %d\n", x) ;
```

```
#endif
```



# Direttive al pre-compilatore C: varianti

`#ifdef`, `#ifndef`: non è necessario il valore  
(opzionale)

```
#define DBG
```

```
...
```

```
    scanf("%d", &x);
```

```
#ifdef DBG
```

```
    printf("Ho letto: %d\\n", x);
```

```
#endif
```

**Nessun valore !**