

Il linguaggio C

Puntatori a funzioni ed esempi di
computazioni generiche

I puntatori a funzione

- Consideriamo la funzione :

```
int somma (int x, int y) {  
    return x+y;  
}
```

- se proviamo ad eseguire

```
printf ("%p", somma) ;
```

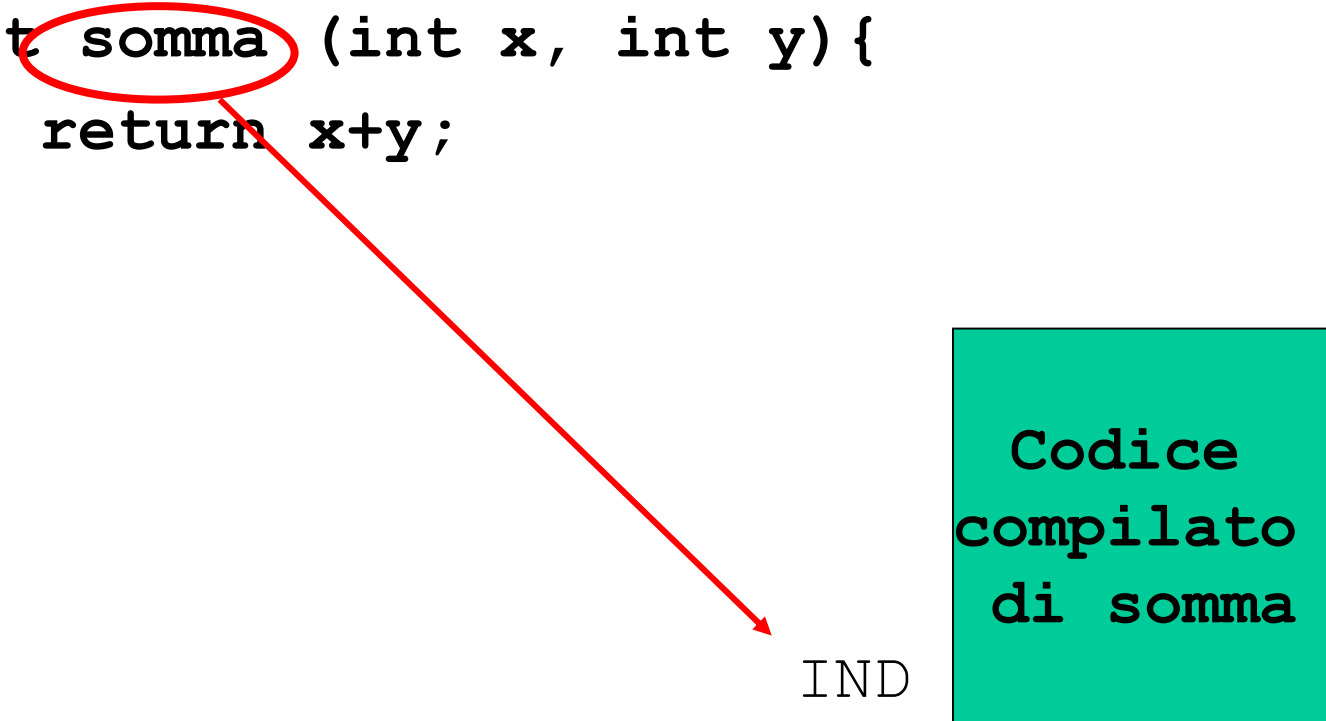
otteniamo un valore esadecimale che rappresenta un indirizzo legale del nostro programma

- ?????????????????????????????????????

I puntatori a funzione (2)

- Consideriamo la funzione :

```
int somma (int x, int y) {  
    return x+y;  
}
```



Codice
compilato
di somma

IND

somma è un puntatore costante
con valore pari a **IND**

I puntatori a funzione (3)

- Consideriamo la funzione :

```
int somma (int x, int y) {  
    return x+y;}  

```

```
/* variabile di tipo funzione  
   (int,int)->int */  

```

```
int (*fun) (int,int);  

```

```
int a;  

```

```
fun = somma;  

```

```
a = fun(3,5);  

```

I puntatori a funzione (4)

- Consideriamo la funzione :

```
int somma (int x, int y) {  
    return x+y;}
```

```
/* variabile di tipo funzione  
   (int,int)->int */
```

```
int (*fun) (int,int);
```

```
int a;
```

```
fun = somma;
```

```
a = fun(3,5);
```

**Ma a che serve
??????????????**

I puntatori a funzione (5)

- Serve a definire funzioni che prendono come argomenti altre funzioni (*di ordine superiore*):

```
void map (int (*fun) (int),
          int x[], int l){
    int i;
    for(i=0;i<l;i++)
        x[i] = fun(x[i]);
}
```

- è un iteratore che applica la funzione **fun** a tutti gli elementi dell'array **x**

I puntatori a funzione (6)

- Esempio di uso della map :

```
int piu_uno (int x) {  
    return x+1;}  

```

```
int quad (int x) {  
    return x*x;}  

```

...

```
int a[3] = {3,4,5};
```

```
map(piu_uno,a,3); /* somma uno a  
tutti gli elementi */
```

```
map(quad,a,2); /* eleva al quadrato  
i primi due elementi */
```

tipo puntatore generico: `void*`

- Può contenere indirizzi di variabili di tipi diversi
 - Non si può dereferenziare
 - È prima necessario effettuare un cast a un tipo noto
 - Es :

```
void * c;
```

```
int a;
```

```
c = &a;
```

```
*c = 5; /* scorretto*/
```

```
*(int *)c = 5; /* corretto*/
```


tipo puntatore generico: **void*** (2)

- Serve a scrivere funzioni ‘polimorfe’ in modo un po’ brutale
- Es :
 - il tipo della malloc() è

```
void * malloc (size_t size);
```
 - quando scrivo

```
int * a;  
a = malloc(10*sizeof(int));
```
 - viene effettuato un cast implicito a (int *)

tipo puntatore generico: `void*` (3)

- Tipi delle altre funzioni di allocazione e deallocazione

```
void * calloc (size_t nmemb, size_t size)
void * realloc (void * ptr, size_t size);
void free (void * ptr);
```

Genericita': esempio qsort

```
void qsort (void* base,  
size_t nmemb,  
size_t size,  
int (*compar) (void*,void*) );
```

- Ordina una array di tipo qualsiasi (**base**) di **nmemb** elementi ognuno di ampiezza **size** utilizzando la funzione **compar** per confrontare gli elementi

Genericita': esempio qsort (2)

- Come si usa qsort ? (vedi man qsort)

```
int array[N];
```

```
int cmpint (void* a, void* b) {  
    int ai = * (int*) a;  
    int bi = * (int*) b;  
    return ai - bi;  
}
```

...

```
qsort(array, N, sizeof(int), cmpint);
```

Genericita' esempio qsort (3)

```
int compar_string (void* a, void* b) {  
    char* ai = *(char **) a;  
    char* bi = *(char **) b;  
    return strcmp(ai,bi) };
```

```
char* string[MAX];
```

```
qsort(string,MAX,sizeof(char*),compar_string);
```

- Ordina l'array **string** usando la funzione **compar_string** passata come argomento

Genericita': esempio qsort (4)

- Come si implementa qsort ? (cenni)

```
void qsort (void* base, size_t nmemb,  
size_t size, int (*compar) (void*,void*) ) {  
/* accedo agli elementi */  
unsigned char * pfirst = (char*) base;  
unsigned char * psecond;  
pfirst; /* puntatore primo elemento */  
psecond = pfirst + size; /*aritmetica dei  
puntatori mi creo il puntatore al secondo  
elemento */  
.....  
/* confronto gli elementi */  
compar((void*)pfirst, (void*)pssecond);
```