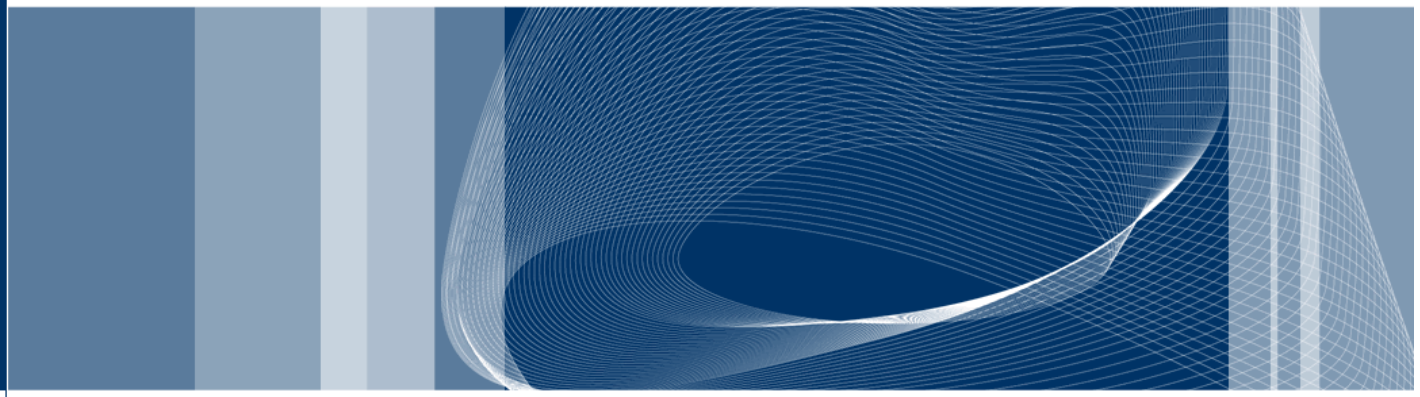
POLITECNICO DI MILANO



Funzioni in C

Fondamenti di Informatica

Daniele Loiacono

Organizzare il codice

- Consideriamo un frammento programma che calcola il massimo di una sequenza di interi letti da tastiera:

```
int i,x,max;
scanf("%d",&max);
for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
}
printf ("Max: %d",max);
…
scanf("%d",&max);
for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
}
printf ("Max: %d",max);
```

Poco leggibile

Scarsa manutenibilità

Nessuna riusabilità

Cosa possiamo fare con le funzioni?

Dare un nome ad una sequenza di istruzioni…

… e poterla *richiamare* facilmente in un programma

Rendere parametrica la sequenza di istruzioni

Calcolare un valore

Funzioni senza parametri/ritorno

```
void nomeFunzione()
{
     <istruzioni>
}
```

```
nomeFunzione();
```

Funzioni senza parametri/ritorno: esempio

```
int main()
{
    int i,x,max;
    scanf("%d",&max);
    for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    printf ("Max: %d",max);
    …
    scanf("%d",&max);
    for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    printf ("Max: %d",max);
    return 0;
}
```

```
void seq_max()
{
  int i,x,max;
  scanf("%d",&max);
  for(i=1;i<10;i++) {
    scanf("%d",&x);
    if (x>max)
      max = x;
  }
  printf ("Max: %d",max)
}
```

Funzioni senza parametri/ritorno: esempio

```
int main()
{

    seq_max();
    …
    seq_max();
    return 0;
}
```

```
void seq_max()
{
  int i,x,max;
  scanf("%d",&max);
  for(i=1;i<10;i++) {
    scanf("%d",&x);
    if (x>max)
      max = x;
   }
   printf ("Max: %d",max)
}
```

Funzioni con parametri in ingresso

```
void nomeFunzione(tipo1 par1,tipo2 par2,
                                  …,tipoN parN) {
      <istruzioni>
}
```

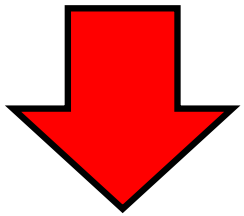
```
nomeFunzione(val1,val2,…,valN);
```

- par1, par2, …, parN sono detti **parametri formali**
- val1, val2, …, valN sono detti **parametri attuali**

Funzioni con parametri in ingresso (2)

- Qual'è la semantica della chiamata di funzione?

```
nomeFunzione(val1,val2,…,valN);
```



```
par1=val1;
par2=val2;
…
parN=valN;
<istruzioni>
```

- I parametri attuali (val1, val2,…,valN) servono esclusivamente per **inizializzare** i parametri formali (par1, par2,…,parN): non c'è quindi *alcun legame* fra parametri attuali e formali durante e dopo l'esecuzione della funzione

Funzioni con parametri in ingresso: esempio

```
int main()
{
    int i,x,max;
    scanf("%d",&max);
    for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    printf ("Max: %d",max);
    …
    scanf("%d",&max);
    for(i=1;i<5;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    printf ("Max: %d",max);
    return 0;
}
```

```
void seq_max(int N)
{
   int i,x,max;
   scanf("%d",&max);
   for(i=1;i<N;i++) {
     scanf("%d",&x);
     if (x>max)
       max = x;
    }
    printf ("Max: %d",max)
}
```

Funzioni con parametri in ingresso: esempio

```
int main()
{
    seq_max(10);
    …
    seq_max(5);
    return 0;
}
```

```
void seq_max(int N)
{
  int i,x,max;
  scanf("%d",&max);
  for(i=1;i<N;i++) {
    scanf("%d",&x);
    if (x>max) max = x;
   }
   printf ("Max: %d",max)
}
```

Funzioni con valore di ritorno

```
tipoR nomeFunzione(tipo1 par1,tipo2 par2,
                                …,tipoN parN) {
      <istruzioni>
      return valore;
}
```

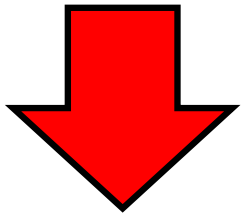
```
valR = nomeFunzione(val1,val2,…,valN);
```

- Se tipoR non è **void**, la chiamata alla funzione *ritornerà* un valore di quel tipo.
- L'utilizzo dell'istruzione **return** all'interno di una funzione termina immediatamente la funzione e ritorna un valore.

Funzioni con valore di ritorno (2)

- Qual'è la semantica della chiamata di funzione?

```
valR = nomeFunzione(val1,val2,…,valN);
```



```
par1=val1;
par2=val2;
…
parN=valN;
<istruzioni>
valR = valore;
```

Funzioni valore di ritorno: esempio

```
int main()
{
    int i,x,max,sum_max=0;
    scanf("%d",&max);
    for(i=1;i<10;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    sum_max = max;
    …
    scanf("%d",&max);
    for(i=1;i<5;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    sum_max = sum_max+max;
    return 0;
}
```

```
int seq_max(int N)
{
  int i,x,max;
  scanf("%d",&max);
  for(i=1;i<N;i++) {
    scanf("%d",&x);
    if (x>max)
      max = x;
  }
  return max;
}
```

Funzioni valore di ritorno: esempio

```
int main()
{
    int sum_max;
    sum_max = seq_max(10) + seq_max(5);
    return 0;
}
```

```
int seq_max(int N)
{
   int i,x,max;
   scanf("%d",&max);
   for(i=1;i<N;i++) {
     scanf("%d",&x);
     if (x>max) max = x;
    }
    return max;
}
```

Dichiarazione e definizione

Dichiarazione del prototipo

- La **dichiarazione** del **prototipo** (o *signature*) della funzione specifica nome, tipo del valore calcolato e parametri.

```
tipoDato nomeFunzione(<parametri>);
```

- Serve ad informare il compilatore che quella funzione sarà usata nel programma.
- Dove devo dichiarare il prototipo di una funzione?
  - la dichiarazione del prototipo deve **precedere** il punto in cui la funzione viene *richiamata* per la prima volta
  - abitualmente le dichiarazioni dei prototipi vengono posizionate all'inizio del programma, prima del main

Definizione della funzione

- La **definizione** specifica sia il prototipo che la sequenza di istruzioni contenute nella funzione.

```
tipoDato nomeFunzione(<parametri>)
{
      <istruzioni>
}
```

- Dove devo definire la funzione?
  - La funzione può essere definita ovunque nel programma
  - Se la definizione della funzione **precede** il punto del programma in cui viene richiamata per la prima volta, la dichiarazione del prototipo può essere omessa
  - Per semplicità si consiglia di utilizzare le dichiarazioni dei prototipi e definire le funzioni dopo il main.

Richiamare una funzione

- Per richiamare una funzione in un programma, è sufficiente scrivere il nome della funzione stessa, seguito da eventuali parametri tra parentesi tonde:

```
nomeFunzione(<parametri>)
```

- Nel caso in cui la funzione calcoli un valore, tale valore verrà sostituito alla chiamata di funzione stessa all'interno del programma.

Cosa succede alle variabili?

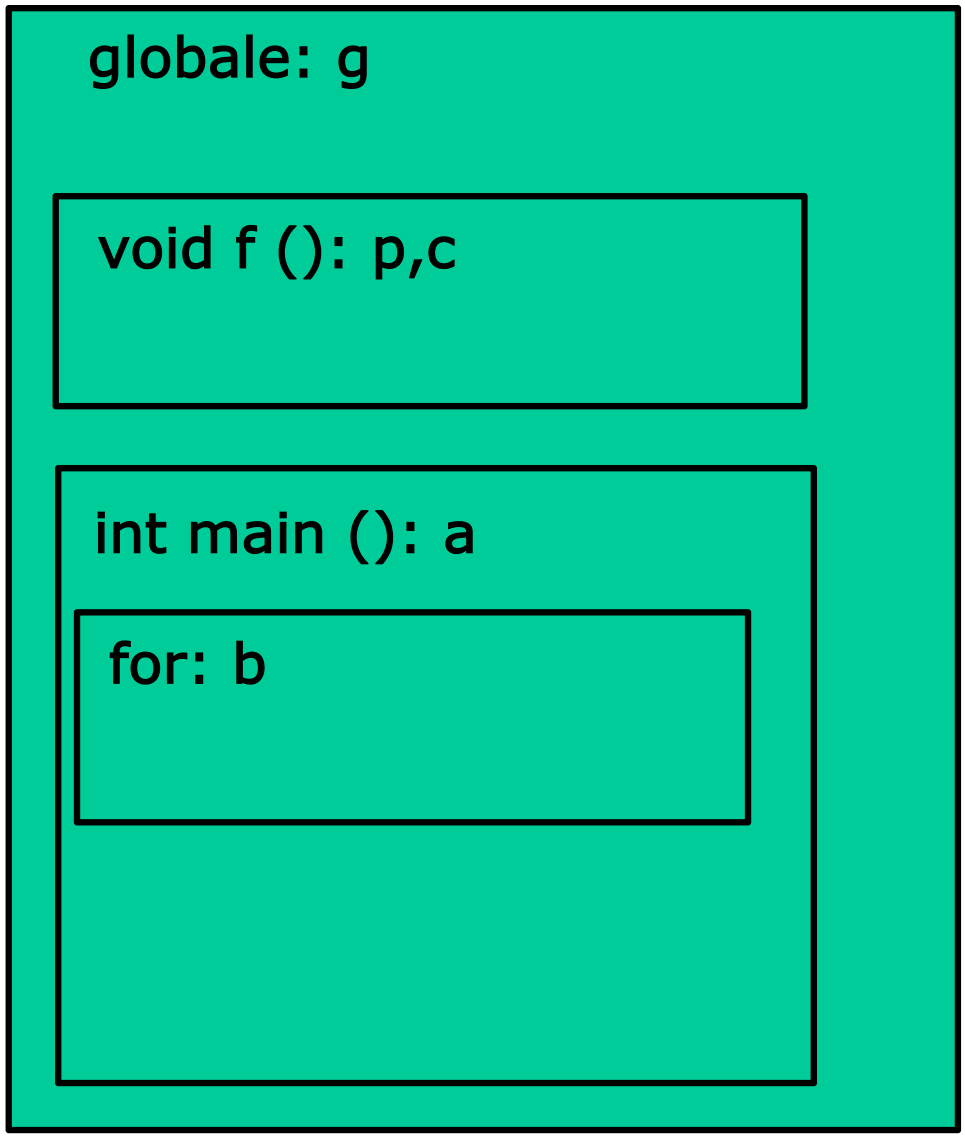
Visibilità delle variabili

```
void f(int p);

float g;
int main()
{
    int a;
    for (a=0; a<5; a++)
    {
        int b;
    }
}
void f(int p)
{
  int c;
}
```

- In quali parti del programma sono **visibili** le variabili g, a, b, c e p?
- Regole di visibilità (scoping) in C:
  - ogni area racchiusa fra {} costituisce un **blocco** (funzione, if, for, while, …)
  - le variabili dichiarate all'interno di un blocco sono dette **locali** (rispetto a quel blocco) ed è visibile solo all'interno di quel blocco
  - le variabili che nono sono dichiarate all'interno di nessun blocco sono dette **globali** e sono visibili ovunque

Visibilità delle variabili: esempio

```
void f(int p);

float g;
int main()
{
    int a;
    for (a=0; a<5; a++)
    {
        int b;
    }
}
void f(int p)
{
  int c;
}
```



Conflitti sui nomi di variabile…

- In C, è possibile dichiarare una variabile locale che abbia **lo stesso** nome di una variabile dichiarata in un altro blocco!
- Cosa succede?
  - se la variabile esterna al blocco **non** è visibile nel blocco, non c'è alcun effetto collaterale
  - se la variabile esterna è invece visibile, questa viene **oscurata** dalla dichiarazione della variabile locale
- Esempio

```
int g;
void f () {
  int a,g; //oscura la variabile globale g!!!
}
int main() {
  int a; //nessun problema: a in f() non è visibile qui
}
```

Vita di una variabile

- Le variabili locali vengono *create* all'atto della loro dichiarazione dentro un blocco
- Quando l'esecuzione blocco è terminata, tutte le variabili locali di quel blocco vengono **distrutte.**
- **Esempio**

```
if (a!=0)
{
   int x;
   scanf("%d",&x);
   x = x/a;
} //qui viene distrutta la variabile x
printf("%d",x);//errore
```

Vita di una variabile… eccezioni

- Le variabili locali che sono dichiarate con la parola chiave **static** mantengono invece il loro valore tra un esecuzione di un blocco e il successivo.
- **Esempio**

```
int ris;
for (int i=0; i<5; i++){
  static int s=0;
  s++;
  ris = s;
}
printf("%d\n",ris); //stampa 5!!!
```

- Attenzione: le **variabili** static sono molto insidiose e difficili da gestire correttamente

Regole di scoping e funzioni

- I parametri formali sono a tutti gli effetti variabili locali di una funzione e seguono, perciò, tutte le regole viste finora:
  - possono oscurare una variabile esterna (ad es. globale)
  - non sono visibili all'esterno
  - vengono distrutte quando la funzione termina la sua esecuzione
- Variabili globali e funzioni
  - Le variabili globali sono visibili e **modificabili** all'interno di una funzione.
  - Questa possibilità comporta la possibilità di creare *effetti secondari* che sono difficili da *prevedere* per chi chiama la funzione.
  - Per evitare questi problemi, è opportuno limitare il più possibile l'uso di variabili globali.

Scoping ed identificatori C

- Le regole di visibilità descritte in precedenza valgono per tutti gli identificatori in C (come tipi di dati e funzioni)

```
typedef enum{red,green,blue} colore;
int f(int a);

int main(){
  typedef enum {vero,falso} booleano;
  colore c=green;
  booleano b=falso;
  int g();
}
int f(int a){
  return a==red && g();
}
int g(){
  return vero;
}
```