

Variabili locali

- ▶ Il blocco che costituisce il corpo di una funzione/procedura può contenere dichiarazioni di variabili.

Esempio:

- ▶ sono variabili proprie della funzione
- ▶ hanno **tempo di vita** limitato alla durata della chiamata
- ▶ più in generale: un identificatore dichiarato nel corpo di una funzione è detto **locale** alla funzione e **non è visibile all'esterno** della funzione (ad esempio nel `main`), ma solo nel corpo della stessa
- ▶ In realtà, ciò non è altro che un **caso particolare** di regole generali che governano la **visibilità** e il **tempo di vita** degli identificatori di un programma.

Struttura generale di un programma C

- ▶ parte direttiva
- ▶ parte dichiarativa **globale** che comprende (ricordarsi l'acrostico **La Cosa Tre Volte Più Facile**):
 - ▶ dichiarazioni di **Costanti**
 - ▶ dichiarazioni di **Tipi** (li vedremo ...)
 - ▶ dichiarazioni di **Variabili** (**variabili globali**)
 - ▶ prototipi di **Procedure/Funzioni**
- ▶ il programma principale (`main`)
- ▶ le definizioni di funzioni/procedure

Esempio

```
#include <stdio.h>          /* parte direttiva */
#define LUNG 10

int i = 1;                  /* variabili globali */
int j = 2;

int Q(int);                /* prototipi di funzioni e procedure */
void R(float);

main()                      /* programma principale */
{
  int x = 10;
  float y = 3.4;
  char c = 'a';
  x = Q(x);
  R(y);
}

int Q(int v) { ... }      /* definizioni di funzioni e procedure */
void R(float z) { ... }
```

Blocchi

- ▶ il corpo di una funzione/procedura, come il corpo del programma principale, è un **blocco**.
- ▶ In C un blocco è costituito da
 - ▶ una parte dichiarativa (può non esserci)
 - ▶ una parte esecutiva (sequenza di istruzioni)
- ▶ Nel **main** o nel corpo delle funzioni possono comparire diversi blocchi, che possono essere
 - ▶ **annidati**: un blocco è una delle istruzioni di un altro blocco
 - ▶ **paralleli**: blocchi che fanno parte della medesima sequenza di istruzioni

```
{
  int x;
  x = 10;
  {
    int z;
    z = 20 ;
    ...
  }
  ...
}
```

```
{
  int x;
  x = 10;
  ...
  {
    int z;
    z = 20;
    ...
  }
}
```

- ▶ Anche la parte esecutiva del programma principale e di una funzione/procedura è un blocco
- ▶ Gli identificatori dichiarati nella parte dichiarativa di un blocco sono detti **nomi locali** del blocco e devono essere tutti **diversi** tra loro
 - ▶ nel caso di una funzione/procedura, fanno parte dei nomi locali anche gli identificatori utilizzati per i parametri formali

Esempio:

```
{
int x; /* NO! identificatore x dichiarato */
char x; /* due volte nello stesso blocco */
...
}

void p(int x, char y)
{
int x; /* NO! identificatore x già' usato per un parametro formale */
...
}
```

- ▶ In blocchi diversi possono essere utilizzati gli stessi identificatori

Esempio:

```
main()
{
int x; /* x, y: variabili locali del main */
int y;
...
{
char x; /* x: variabile locale del blocco annidato */
...
}
...
}

void p(int x)
{
int y; /*x,y: variabili locali della procedura p */
...
}
```

- ▶ Un programma C può avere una struttura molto complessa a seguito dell'uso di funzioni, procedure e blocchi.
- ▶ È necessario definire regole precise per regolamentare l'uso dei nomi utilizzati all'interno di un programma.
- ▶ A questo scopo introduciamo alcune definizioni utili.
 - Ambiente globale:** è l'insieme di tutti gli elementi (nomi) dichiarati nella parte dichiarativa globale del programma
 - Ambiente locale di una funzione:** è l'insieme di tutti gli elementi (nomi) dichiarati nella parte dichiarativa della funzione e nella sua intestazione
 - Ambiente locale di un blocco:** è l'insieme di tutti gli elementi (nomi) dichiarati nella parte dichiarativa del blocco
- ▶ Quanto detto informalmente in precedenza può essere meglio precisato:
 - ⇒ è possibile dichiarare più volte lo stesso identificatore (anche con significati diversi) purché in ambienti diversi
- ▶ Se ciò evita il proliferare di identificatori, causa il problema di stabilire il significato di un riferimento ad un identificatore in un generico punto del programma

Esempio: Riprendiamo l'esempio precedente

```
main()
{
  int x;      /* x, y: variabili locali del main */
  int y;
  ...
  {
    char x;   /* x: variabile locale del blocco annidato */
    ...
  }
  ...
}

void p(int x)
{
  int y;      /*x,y: variabili locali della procedura p */
  ...
}
```

- ▶ Se in un punto del programma viene eseguita l'istruzione `x = ...`, a quale delle **tre** dichiarazioni di `x` ci si riferisce?
- ▶ Dipende dal punto in cui si trova tale assegnamento e dalle **regole di visibilità** (o regole di **scoping**).

Regole di visibilità

Gli identificatori presenti nell'ambiente

- ▶ **globale** sono visibili in tutte le funzioni e in tutti i blocchi del programma.
N.B. Gli identificatori predefiniti del linguaggio si intendono parte dell'ambiente globale.
- ▶ **locale di una funzione** sono visibili nel corpo della funzione (ivi compresi eventuali blocchi in esso contenuti).
- ▶ **locale di un blocco** sono visibili nella parte esecutiva del blocco (ivi compresi eventuali blocchi in essa contenuti).

Se un identificatore è definito in più punti, la definizione valida è quella dell'ambiente più vicino al punto di utilizzo. In particolare, una variabile locale "nasconde" una eventuale variabile omonima definita nell'ambiente superiore (**shadowing** o oscuramento) per tutto il blocco.

- ▶ Detto altrimenti, l'ambito di visibilità di un identificatore è determinato dalla posizione della sua dichiarazione:
 - ▶ gli identificatori dichiarati all'interno di un blocco hanno ambito di visibilità a livello di blocco
⇒ una variabile dichiarata in un **blocco** è visibile **solo in quel blocco** (compresi eventuali blocchi annidati)
 - ▶ gli identificatori dichiarati all'interno di una **funzione** (compresi quelli nell'intestazione) hanno ambito di visibilità **a livello di funzione**
⇒ una variabile dichiarata in una **funzione** è visibile **solo nel corpo della funzione** (compresi eventuali blocchi annidati)
 - ▶ gli identificatori dichiarati all'esterno delle funzioni e del main hanno ambito di visibilità a livello di programma
⇒ una variabile **globale** è visibile **ovunque** nel programma

Esempio:

```
int x1=10, x2=20;
char c='a';

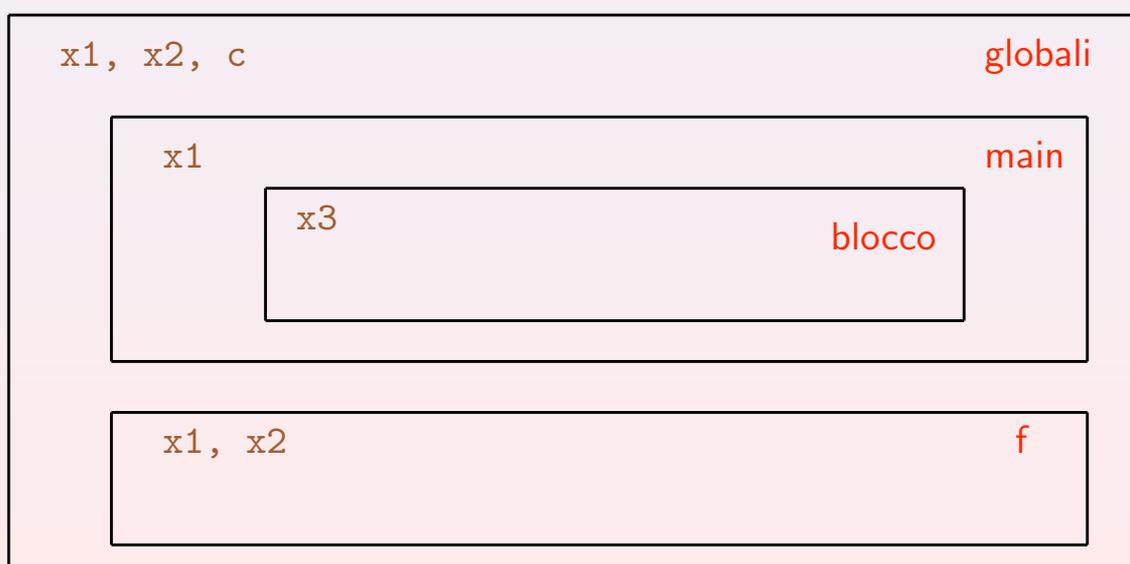
int f(int);

main()
{
int x1=30;    /* nasconde la variabile globale x1 */
x2 = x1+x2;  /* x1 e' quella locale, x2 e' globale */
printf("x1=%d  x2=%d\n", x1, x2);  /* stampa x1=30  x2=50 */
  { int x3=50;
    x1=f(x3); /* x1 e' quella locale al primo blocco */
    printf("x1=%d  x2=%d\n", x1, x2);  /* stampa x1=150  x2=50 */
  }
}

int f(int x1) /* nasconde la variabile globale x1 */
{ int x2;    /* nasconde la variabile globale x2 */
  x2 = x1 + 100; /* x1 e' il parametro formale, x2 la var. locale */
  return x2;
}
```

Rappresentazione Grafica: Modello a contorni

- ▶ Si rappresenta ogni **ambiente** mediante un rettangolo con gli identificatori in esso contenuti.



Durata delle variabili

- ▶ Una variabile ha un suo **tempo di vita**.
 - viene **creata** (ovvero ad essa viene riservata uno spazio di memoria)
 - viene (o può essere) **distrutta** (ovvero viene rilasciato il corrispondente spazio di memoria).
- ▶ Si distinguono due classi di variabili:
 - ▶ variabili **automatiche**: vengono create ogni volta che si entra nel loro ambiente di visibilità e vengono distrutte all'uscita di tale ambiente
 - ▶ es. variabili **locali di un blocco**: vengono create all'ingresso del blocco, {, e distrutte all'uscita dal blocco, }.
 - ▶ es. variabili **locali di una funzione**: vengono create al momento della chiamata e distrutte all'uscita
 - ▶ variabili **statiche**: vengono create una sola volta e vengono distrutte solo al termine dell'esecuzione del programma (non ne faremo uso ...)

Durata delle variabili (cont.)

Nel caso di funzioni/blocchi eseguiti più volte (es. funzione chiamata in punti diversi, blocco all'interno di un ciclo):

le variabili automatiche corrispondenti possono essere associate di volta in volta a locazioni di memoria diverse, quindi

il loro valore **non persiste** tra una esecuzione e la successiva

Gestione della memoria a tempo di esecuzione (run-time)

- ▶ La memoria per
 - ▶ il codice macchina è fissata a tempo di compilazione
 - ▶ i dati (in particolare per le variabili automatiche) cresce e decresce dinamicamente durante l'esecuzione: viene gestita a **pila**
- ▶ Ricordiamo che una **pila** (o **stack**) è una struttura dati con accesso **LIFO**: Last In First Out (ultimo entrato, primo servito). ad es.: pila di piatti da lavare o di pratiche da svolgere.

Pila dei record di attivazione

- ▶ Il sistema gestisce in memoria la **pila dei record di attivazione (RDA)**
 - ▶ per ogni **chiamata di funzione** viene creato un nuovo **RDA** in cima alla pila
 - ▶ al termine della chiamata della funzione il **RDA** viene rimosso dalla pila
- ▶ Ogni **RDA** contiene:
 - ▶ le locazioni di memoria per i parametri formali (se presenti)
 - ▶ le locazioni di memoria per le variabili locali (se presenti)
 - ▶ altre informazioni che non analizziamo
- ▶ Anche gli ambienti locali dei blocchi vengono allocati/deallocati sulla pila.

Pila dei record di attivazione: come funzionano

- ▶ Il primo record di attivazione (RDA) è per il `main()`
- ▶ Ad ogni attivazione viene allocato un RDA
- ▶ Al termine dell'attivazione il record viene rilasciato (liberando la memoria che occupava)
- ▶ La dimensione dell'RDA è nota in fase di compilazione
- ▶ Il numero di attivazioni della funzione non è noto

Esempio:

```
int f(int);
main()
{
    int x, y, z;
    x=10;
    y=20;          /* blocco principale */
    z = f(x);      /* prima chiamata di f */
    {
        int x=50;  /* uscita da f e ingresso nel blocco annidato*/
        y=f(x);   /* seconda chiamata di f */
        z=y;      /* uscita da f */
    }
    ...          /* uscita dal blocco */
}
int f(int a)
{
    int z;
    z = a + 1;
    return z;
}
```

[◀ PUNTO 1](#)[◀ PUNTO 2](#)[◀ PUNTO 3](#)[◀ PUNTO 4](#)[◀ PUNTO 5](#)[◀ PUNTO 6](#)

Evoluzione della pila

x	10
y	20
z	?

► PUNTO 1

Evoluzione della pila

a	10
z	?
x	10
y	20
z	?

► PUNTO 2

Evoluzione della pila

x	50
x	10
y	20
z	11

▶ PUNTO 3

Evoluzione della pila

a	50
z	?
x	50
x	10
y	20
z	11

▶ PUNTO 4

Evoluzione della pila

x	50
x	10
y	51
z	11

► PUNTO 5

Evoluzione della pila

x	10
y	51
z	51

► PUNTO 6

Variabili statiche: un esempio d'uso

- ▶ Una variabile **statica**, una volta creata, rimane in vita per tutto il tempo di esecuzione del programma.

Esempio: `f(void) { static int x; ... }`

- ▶ la variabile viene inizializzata alla prima attivazione della funzione
- ▶ conserva il suo valore tra attivazioni successive
- ▶ è locale, quindi visibile solo all'interno della funzione in cui è dichiarata

Esempio: Funzione che restituisce il numero di volte che è stata attivata.

```
int fun1(void) {
    static int conta = 0;
    /* variabile locale statica visibile solo in fun1;
       contatore del numero di attivazioni di fun1 */
    .....
    conta++;
    return conta;
}
```