

Classi di memorizzazione

Ogni variabile, in C, appartiene a una certa “classe di memorizzazione” (storage class). In C esistono quattro classi di memorizzazione, ma noi ne vedremo solo tre:

1. *variabili locali* (o automatiche)
2. *variabili globali* (o esterne)
3. *variabili statiche*

Variabili locali

Una variabile locale è una variabile che viene dichiarata all'inizio di un blocco (cioè, di un insieme di istruzioni racchiuse fra parentesi graffe { . . . }). In particolare, sono locali le variabili che vengono dichiarate all'inizio di una funzione (compreso il `main`). Nel primo caso diremo che la variabile è *locale al blocco* in cui è dichiarata; nel secondo caso, che è *locale alla funzione* in cui è dichiarata.

Inoltre i parametri formali delle funzioni sono considerati, a tutti gli effetti, variabili locali della funzione.

Esempio

```
void f(int a, char s) {
    int x,y;
    float j[4];

    ....
    for (x=0; x<7; x++) {
        int i;
        char s[20];
        ....
    }

    ...
    {
        int y,z;
        ....
        {
            float j,g;
            char x;
            ....
        }
    }
}
```

The diagram illustrates the call stack for the function `f`. A vertical dashed line represents the stack. Horizontal dashed lines indicate the boundaries of each function call. From top to bottom, the frames are: the caller's frame (labeled `a,s,x,y,j`), the caller's frame (labeled `i,s`), the caller's frame (labeled `y,z`), the caller's frame (labeled `f,j,x`), and the caller's frame (labeled `a,s,x,y,j`).

Variabili globali

Le variabili globali sono quelle dichiarate fuori da tutti i blocchi, solitamente all'inizio del programma:

```
#include <stdio.h>

int i,j;
char v[200];

int f (float a) {
    ....
}

void g (int x, int y) {
    ....
}

...

int main() {
    ....
}
```

Variabili statiche

Le variabili statiche vengono dichiarate prefiggendo `static` alla dichiarazione. Possono essere dichiarate sia all'interno di una funzione, che all'esterno.

```
#include <stdio.h>

int i,j;
static char v[200];

int f (float a) {
    ....
}

void g (int x, int y) {
    static int q;
    ....
}

...

int main() {
    ....
}
```

Regole di visibilità (Scope rules)

Una variabile si dice *visibile* in un certo punto del programma se il suo valore è direttamente noto e modificabile in quel punto del programma.

La regola generale di visibilità è questa:

- le variabili dichiarate fuori da un blocco sono visibili ovunque;
- le variabili dichiarate in un blocco sono visibili solo nel blocco in cui sono dichiarate e in tutti i suoi sottoblocchi.

L'unica eccezione a questa regola è costituita dall'*effetto di ombreggiatura* (shadow effect):

- se in un blocco viene dichiarata una variabile che ha lo stesso nome di un'altra già visibile a quel blocco, quest'ultima viene oscurata da quella locale.

Esempio

```
int x,u;      // Globali

void f(int a, char s) {
    int x,y;
    float j[4];

    ....
    for (x=0; x<7; x++) {
        int i;
        char s[20];
        ....
    }

    ...
    {
        int y,z;
        ....
        {
            float j,g;
            char x;
            ....
        }
    }
}
```

Diagram illustrating the stack frame structure for the function `f`. The stack grows downwards (increasing memory address). The variables are organized as follows:

- Global variables: `x, u` (at the top of the stack).
- Function arguments: `a, s, x, y, j` (located between the third and fourth dashed lines).
- Local variables of `f`: `x, y, j` (located between the first and second dashed lines).
- Local variables of the `for` loop: `i, s` (located between the second and third dashed lines).
- Local variables of the inner block: `y, z` (located between the fourth and fifth dashed lines).
- Local variables of the innermost block: `j, g, x` (located between the fifth and sixth dashed lines).

Inizializzazione

Quale valore assume una variabile dopo che è stata dichiarata, ma prima che le venga assegnato un valore?

- le variabili *globali* e *statiche* vengono inizializzate a zero;
- le variabili *locali* hanno valori imprevedibili, ad eccezione dei parametri formali, che vengono inizializzati con i valori dei parametri attuali passati alla funzione.

Vita e morte delle variabili

Le variabili globali o statiche esistono per tutta la durata del programma. Le variabili locali, al contrario, esistono solo per la durata di esecuzione del blocco in cui sono dichiarate.

Quando il blocco termina la propria esecuzione, le variabili locali a quel blocco spariscono, e il loro valore viene perso.

```
for (i=0; i<20; i++) {  
    int j;  
    j=i+15;  
    printf("%d",j);  
}
```

Differenza fra variabili statiche e non

Le variabili statiche dichiarate al di fuori di una funzione sono analoghe alle variabili globali. L'unica differenza è che esse *non sono visibili da altri file*.

Le variabili statiche dichiarate all'interno di una funzione, invece, hanno una funzione radicalmente diversa dalle variabili locali, perché esistono per tutta la durata del programma.

```
for (i=0; i<20; i++) {  
    static int j;  
    j=i+15;  
    printf("%d",j);  
}
```

Stack

Un modo conveniente per capire il meccanismo con cui il C crea e distrugge le variabili è considerare il funzionamento dello stack.

Le variabili (per lo meno, quelle locali di cui stiamo parlando ora) vengono collocate in una struttura di memoria detta *stack*. Lo stack funziona come una pila: oggetti possono essere messi in cima allo stack, o tolti dalla cima dello stack.

Inizialmente, lo stack contiene le sole variabili globali (questa è però un'astrazione; discuteremo più tardi che cosa succede esattamente). Ogni volta che viene eseguito un blocco (in particolare: una funzione), viene messo in cima allo stack il cosiddetto *record di attivazione* del blocco, che contiene le variabili locali al blocco.

Quando l'esecuzione di un blocco (in particolare: di una funzione) termina, il suo record di attivazione viene cancellato dallo stack.

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {      <---- all'inizio di esecuzione del main
    int x;
    x=1;
    y=2;
    y=f(y,x)+1;
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x ?		LOCALI ALLA FUNZIONE main()

x 0		VARIABILI
y 0		GLOBALI

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;    <---- dopo aver eseguito gli assegnamenti
    y=f(y,x)+1;
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x 1	LOCALI ALLA FUNZIONE main()	

x 0	VARIABILI	
y 2	GLOBALI	

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1; <----(*) viene invocata f con parametri 2 e 1
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x	2	LOCALI ALLA FUNZIONE f()
z	1	punto di ritorno (*)
f	?	

x	1	LOCALI ALLA FUNZIONE main()

x	0	VARIABILI
y	2	GLOBALI

Alla chiamata della funzione `f`, viene allocato il suo record di attivazione, ricordando il punto in cui si dovrà tornare dopo che l'esecuzione sarà finita.

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;      <--- dopo aver eseguito gli assegnamenti
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1; <----(*)
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x	2	LOCALI ALLA FUNZIONE f()
z	4	punto di ritorno (*)
f	2.0	

x	1	LOCALI ALLA FUNZIONE main()

x	0	VARIABILI
y	2	GLOBALI

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f); <--- il valore restituito e' 6
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1; <----(*)
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x	2	LOCALI ALLA FUNZIONE f()
z	4	punto di ritorno (*)
f	2.0	

x	1	LOCALI ALLA FUNZIONE main()

x	0	VARIABILI
y	2	GLOBALI

A questo punto l'esecuzione di `f` termina: il suo record di attivazione viene cancellato, e si riprende l'esecuzione dal punto di ritorno.

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1; <---- Riparte l'esecuzione (valore rest. 6)
    printf("%d %d",x,y);
    return 0;
}
```

Stack:

x	1	LOCALI ALLA FUNZIONE main()

x	0	VARIABILI
y	7	GLOBALI

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1;
    printf("%d %d",x,y); <--- Viene stampato 1 e 7
    return 0;
}
```

Stack:

x 1	LOCALI ALLA FUNZIONE main()	

x 0	VARIABILI	
y 7	GLOBALI	

Esempio

```
#include <stdio.h>
int x,y;

int f (int x, int z) {
    float f;
    z=x+y;
    f=z/2.0;
    return (int)(z+f);
}

int main() {
    int x;
    x=1;
    y=2;
    y=f(y,x)+1;
    printf("%d %d",x,y);
    return 0;          <--- Termina il main
}
```

Stack:

x 0		VARIABILI
y 7		GLOBALI

Ricorsione

Il meccanismo dei record di attivazione è molto importante per la comprensione del funzionamento di un programma C. Ciò è particolarmente vero quando si scrivono funzioni ricorsive.

Una funzione ricorsiva è una funzione che chiama se stessa. L'idea della ricorsione è abbastanza naturale, e compare spesso in molte definizioni matematiche.

Il classico esempio è rappresentato dal fattoriale.

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

Il fattoriale

La definizione ricorsiva di fattoriale si può tradurre in una funzione ricorsiva, come segue:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

diventa

```
int fatt (int n) {  
    if (n==0) return 1;  
    else return n*fatt(n-1);  
}
```

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1);
}

int main() {
    ...
    x=fatt(3); <---- sto per eseguire la funzione
}
```

Stack:

```
|-----|
| x      ? | LOCALI ALLA FUNZIONE main()
| ..altra roba.... |
```

Il fattoriale in esecuzione

```
int fatt (int n) { <---- chiamata la funzione con parametro 3
    if (n==0) return 1;
    else return n*fatt(n-1);
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n 3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x ?	LOCALI ALLA FUNZIONE main()
..altra roba....	

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---n!=0: chiamo di nuovo f
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n 3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x ?	LOCALI ALLA FUNZIONE main()
..altra roba....	

Il fattoriale in esecuzione

```
int fatt (int n) { <---- chiamata la funzione con parametro 2
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**)
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

```
|-----|
| n      2 | LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]
|-----|
| n      3 | LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]
|-----|
| x      ? | LOCALI ALLA FUNZIONE main()
| ..altra roba.... |
```

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**) n!=0: chiamo di nuovo f
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

```
|-----|
| n      2 | LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]
|-----|
| n      3 | LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]
|-----|
| x      ? | LOCALI ALLA FUNZIONE main()
| ..altra roba.... |
```

Il fattoriale in esecuzione

```
int fatt (int n) { <---- chiamata la funzione con parametro 1
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**),(***)
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	1	LOCALI ALLA FUNZIONE fatt() [Ritorno (***)]

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**), (***)
                                n!=0: chiamo di nuovo f
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	1	LOCALI ALLA FUNZIONE fatt() [Ritorno (***)]

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) { <---- chiamata la funzione con parametro 0
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**),(***),(****)
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	0	LOCALI ALLA FUNZIONE fatt() [Ritorno (****)]

n	1	LOCALI ALLA FUNZIONE fatt() [Ritorno (***)]

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1; <--- Stavolta n==0: restituisco 1
                                e torno al punto di ritorno
    else return n*fatt(n-1); <---(**),(***),(****)
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	0	LOCALI ALLA FUNZIONE fatt() [Ritorno (****)]

n	1	LOCALI ALLA FUNZIONE fatt() [Ritorno (***)]

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**),(***),(****)
        ^--- torno qui, calcolo il prodotto
            n*1 (val rest) e ritorno
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	1	LOCALI ALLA FUNZIONE fatt() [Ritorno (***)]

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**),(***)
        ^--- torno qui, calcolo il prodotto
            n*1 (val rest) e ritorno
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n	2	LOCALI ALLA FUNZIONE fatt() [Ritorno (**)]

n	3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x	?	LOCALI ALLA FUNZIONE main()
..altra roba....		

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1); <---(**)
        ^--- torno qui, calcolo il prodotto
           n*2 (val rest) e ritorno
}

int main() {
    ...
    x=fatt(3); <----(*)
}
```

Stack:

n 3	LOCALI ALLA FUNZIONE fatt() [Ritorno (*)]

x ?	LOCALI ALLA FUNZIONE main()
..altra roba....	

Il fattoriale in esecuzione

```
int fatt (int n) {
    if (n==0) return 1;
    else return n*fatt(n-1);
}

int main() {
    ...
    x=fatt(3); <----(*) Torno qui: val. rest 6
}
```

Stack:

```
|-----|
| x          6 | LOCALI ALLA FUNZIONE main()
| ..altra roba.... |
```