
Esercitazioni con classi di memoria

Filippo Cugini

Esempio: campi d'azione

Consideriamo un programma che illustri alcuni aspetti relativi ai campi d'azione con:

- variabili globali
- variabili locali automatiche
- e variabili locali `static`

Esempio: campi d'azione

```
#include <stdio.h>

void useLocal(void); // prototipo di funzione
void useStaticLocal(void); // prototipo di funzione
void useGlobal(void); // prototipo di funzione

int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main

    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione
```

Esempio: campi d'azione

```
printf("local x in outer scope of main is %d\n", x);

useLocal(); // ha una x locale automatica
useStaticLocal(); // ha una x locale statica
useGlobal(); // usa una x globale
useLocal(); // reinizializza una x locale
useStaticLocal(); // x locale statica conserva valore
useGlobal(); // x globale conserva il suo valore

printf("\nlocal x in main is %d\n", x);
}
```

Esempio: campi d'azione

```
// reinizializza la variabile locale x durante ogni
// chiamata

void useLocal(void)
{
    int x = 25; // inizializzata ogni volta

    printf("\nlocal x in useLocal is %d ", x);
    puts("after enter useLocal");

    ++x;

    printf("local x in useLocal is %d ", x);
    puts("before exiting useLocal");
}
```

Esempio: campi d'azione

```
// inizializza la variabile statica locale x solo la
// prima volta che la funzione e' chiamata;
// il valore di x e' conservato tra una chiamata
// e l'altra a questa funzione

void useStaticLocal (void)
{
    static int x = 50; // inizializza x una sola volta

    printf("\nlocal static x is %d ", x);
    puts("on entering useStaticLocal");

    ++x;

    printf("local static x is %d ", x);
    puts("on exiting useStaticLocal");
}
```

Esempio: campi d'azione

```
// useGlobal modifica la variabile globale x
// in ogni chiamata

void useGlobal (void)
{
    printf("\nglobal x is %d ", x);
    puts("on entering useGlobal");

    x *= 10;

    printf("global x is %d on exiting useGlobal\n", x);
}
```

Esempio: campi d'azione

Output:

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```


Campo d'azione globale

Una variabile globale `x` è definita e inizializzata a 1

Questa variabile globale è nascosta in un qualunque blocco (o funzione) in cui è definita una variabile denominata anch'essa `x`

```
#include <stdio.h>

void useLocal(void); // prototipo di funzione
void useStaticLocal(void); // prototipo di funzione
void useGlobal(void); // prototipo di funzione

int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main
```



Campo d'azione globale

Nella funzione `main` una variabile locale `x` è definita e inizializzata a 5

Questa variabile viene quindi stampata per mostrare che la `x` globale è nascosta in `main`

```
#include <stdio.h>
//...
int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main
    printf("local x in outer scope of main is %d\n", x);
}
```

```
local x in outer scope of main is 5
```


Campo d'azione globale

In seguito, in `main` è definito un nuovo blocco con un'altra variabile locale `x` inizializzata a 7

Questa variabile viene stampata per mostrare che essa nasconde `x` nel blocco esterno di `main`

```
int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main

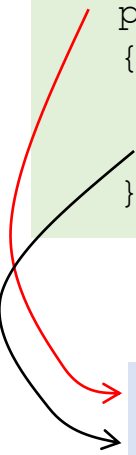
    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione
```



Campo d'azione globale

```
int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main

    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione
```



The diagram consists of two curved arrows. A red arrow originates from the first `printf` statement in the code block and points to the first line of the output box. A black arrow originates from the `printf` statement inside the inner scope's curly braces and points to the second line of the output box.

```
local x in outer scope of main is 5
local x in inner scope of main is 7
```

Campo d'azione globale

La variabile `x` con valore 7 viene automaticamente eliminata quando si esce dal blocco, dopodiché la variabile locale `x` nel blocco esterno di `main` viene ristampata per mostrare che non è più nascosta

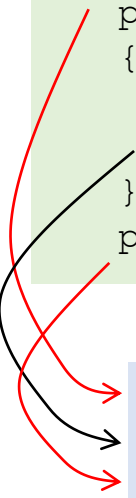
```
int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main

    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione
    printf("local x in outer scope of main is %d\n", x);
}
```

Campo d'azione globale

```
int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main

    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione
    printf("local x in outer scope of main is %d\n", x);
}
```



```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
```

Campo d'azione nelle funzioni

La funzione `useLocal` definisce una variabile automatica `x` e la inizializza a 25

```
void useLocal(void)
{
    int x = 25; // inizializzata ogni volta
    printf("\nlocal x in useLocal is %d ", x);
    puts("after enter useLocal");
    ++x;
    printf("local x in useLocal is %d ", x);
    puts("before exiting useLocal");
}
```


Campo d'azione nelle funzioni

Quando `useLocal` è chiamata, la variabile viene stampata, incrementata e stampata di nuovo prima che si esca dalla funzione

```
void useLocal(void)
{
    int x = 25; // inizializzata ogni volta
    printf("\nlocal x in useLocal is %d ", x);
    puts("after enter useLocal");
    ++x;
    printf("local x in useLocal is %d ", x);
    puts("before exiting useLocal");
}
```


Campo d'azione nelle funzioni

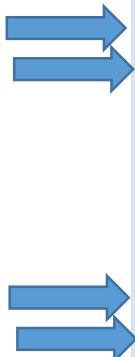
Output:



```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

Ogni volta che questa funzione
useLocal è chiamata, la variabile
automatica x è reinizializzata a 25



```
local x in useLocal è chiamata, la variabile
local x in automatica x è reinizializzata a 25
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

La funzione `useStaticLocal` definisce una variabile statica `x` e la inizializza a 50

Ricordate che la memoria per le variabili `static` è allocata e inizializzata solo una volta prima che il programma inizi l'esecuzione

```
void useStaticLocal (void) {  
    static int x = 50; // inizializzata x una sola volta  
    printf("\nlocal static x is %d ", x);  
    puts("on entering useStaticLocal");  
    ++x;  
    printf("local static x is %d ", x);  
    puts("on exiting useStaticLocal");  
}
```

Campo d'azione nelle funzioni


Le variabili locali dichiarate come `static` mantengono i loro valori anche quando sono fuori dal campo d'azione

Quando `useStaticLocal` è chiamata, `x` viene stampata, incrementata e ristampata prima dell'uscita dalla funzione

```
void useStaticLocal (void) {  
    static int x = 50; // inizializzata x una sola volta  
    printf("\nlocal static x is %d ", x);  
    puts("on entering useStaticLocal");  
    ++x;  
    printf("local static x is %d ", x);  
    puts("on exiting useStaticLocal");  
}
```

Campo d'azione nelle funzioni

Output:



```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

Nella chiamata successiva di questa funzione `useStaticLocal` la variabile locale `static x` conterrà il valore 51 incrementato precedentemente

```
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

La funzione `useGlobal` non definisce alcuna variabile

Pertanto, quando essa si riferisce alla variabile `x`, viene usata la `x` globale


```
void useGlobal (void)
{
    printf("\nglobal x is %d ", x);
    puts("on entering useGlobal");
    x *= 10;
    printf("global x is %d on exiting useGlobal\n", x);
}
```

Campo d'azione nelle funzioni

Quando `useGlobal` è chiamata, la variabile globale viene stampata, moltiplicata per 10 e ristampata prima dell'uscita dalla funzione

```
void useGlobal (void)
{
    printf("\nglobal x is %d ", x);
    puts("on entering useGlobal");
    x *= 10;
    printf("global x is %d on exiting useGlobal\n", x);
}
```


Campo d'azione nelle funzioni



```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

La volta successiva che la funzione useGlobal è chiamata, la variabile globale ha ancora il suo valore modificato, 10

```
local x
local x
local x
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is 5
```

Campo d'azione nelle funzioni

Infine, il programma stampa la variabile locale `x` di nuovo in `main` per mostrare che nessuna delle chiamate di funzione ha modificato il valore di `x`, poiché le funzioni si riferiscono tutte a variabili in altri campi d'azione

```
#include <stdio.h>

void useLocal(void); // prototipo di funzione
void useStaticLocal(void); // prototipo di funzione
void useGlobal(void); // prototipo di funzione

int x = 1; // variabile globale
int main(void)
{
    int x = 5; // variabile locale per main ←

    printf("local x in outer scope of main is %d\n", x);
    { // inizio di un nuovo campo d'azione
        int x = 7; // variabile locale nel campo d'azione
        printf("local x in inner scope of main is %d\n", x);
    } // fine del nuovo campo d'azione

    printf("local x in outer scope of main is %d\n", x);

    useLocal(); // ha una x locale automatica
    useStaticLocal(); // ha una x locale statica
    useGlobal(); // usa una x globale
    useLocal(); // reinizializza una x locale
    useStaticLocal(); // x locale statica conserva valore
    useGlobal(); // x globale conserva il suo valore

    printf("\nlocal x in main is %d\n", x); ←
}
```

Campo d'azione nelle funzioni

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal
global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal
local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal
global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
local x in main is ⑤
```

