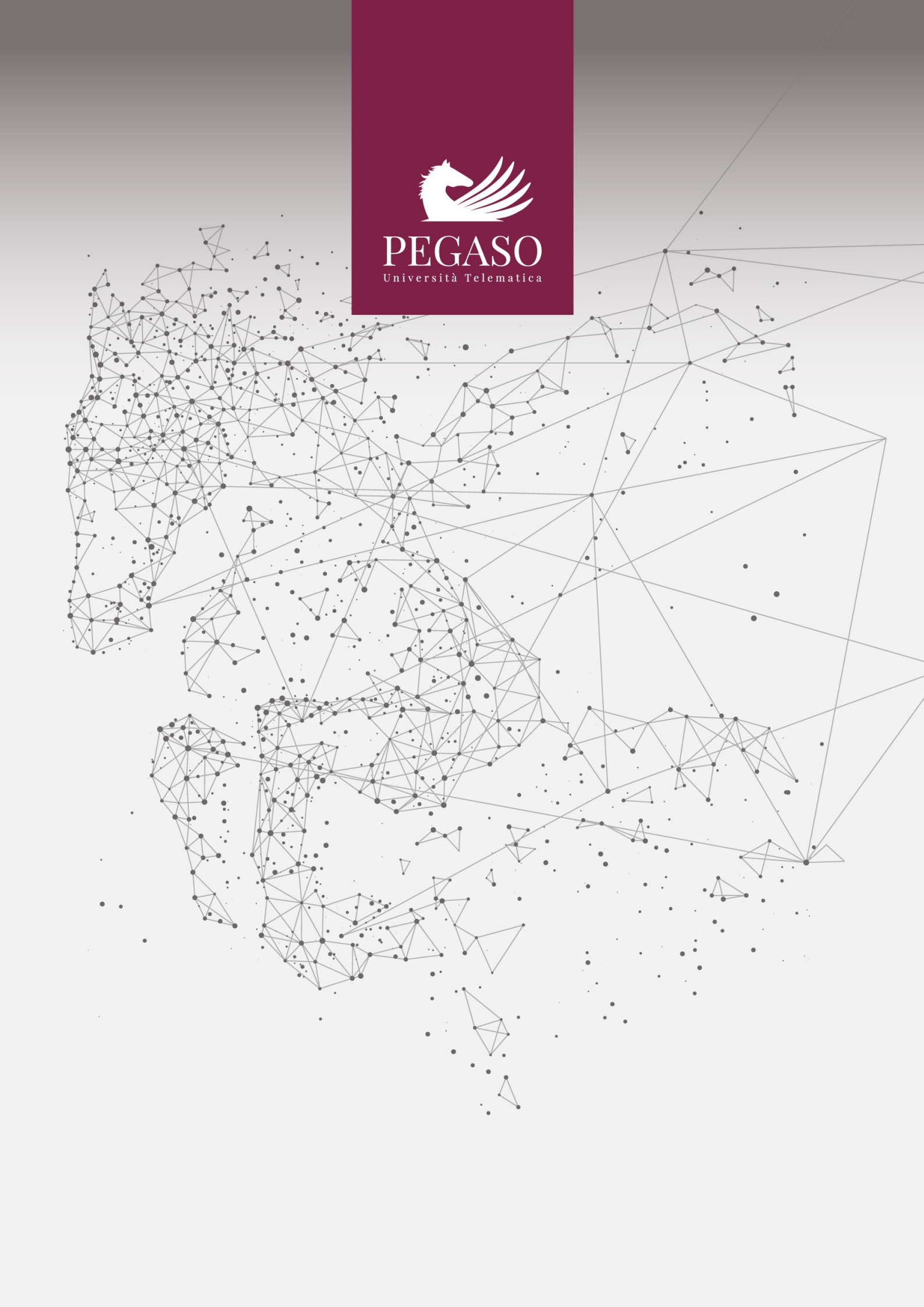
PEGASO
Università Telematica

Indice

1. Classi di memoria

Abbiamo usato identificatori per i nomi delle variabili.

Attributi delle variabili sono:

- nome,
- tipo,
- dimensioni
- e valore.

Usiamo ora gli identificatori anche come nomi per le funzioni definite dall'utente.

In realtà, ogni identificatore in un programma ha altri attributi, quali:

- la classe di memoria,
- la permanenza in memoria,
- il campo di azione
- e il collegamento.

Il C fornisce gli specificatori della classe di memoria:

- `auto`
- `extern`
- `static`
- `_Thread_local` (aggiunto in C11, non lo considereremo)

La classe di memoria di un identificatore determina la sua permanenza in memoria, il suo campo di azione e il suo collegamento.

La permanenza in memoria di un identificatore è il periodo durante il quale l'identificatore esiste nella memoria.

Alcuni esistono per breve tempo, alcuni sono ripetutamente creati e distrutti e altri esistono per l'intera esecuzione del programma.

Il campo di azione (in inglese "scope") di un identificatore è dove l'identificatore può essere menzionato in un programma.

Alcuni si possono menzionare per tutto il programma, altri solo da porzioni di esso.

Il collegamento di un identificatore determina, per un programma con diversi file sorgente, se l'identificatore è conosciuto soltanto nel file sorgente corrente o in qualunque file sorgente con le opportune dichiarazioni.

Iniziamo ad esaminare le classi di memoria e la permanenza in memoria.

Gli specificatori della classe di memoria si suddividono tra permanenza in memoria automatica e permanenza in memoria statica.

La parola chiave auto è usata per dichiarare le variabili con permanenza in memoria automatica.

Le variabili con permanenza in memoria automatica sono create quando il controllo del programma entra nel blocco nel quale sono definite.

Esse esistono finché il blocco è attivo e sono distrutte quando il controllo del programma esce dal blocco.

Variabili locali

Solo le variabili possono avere permanenza in memoria automatica.

Le variabili locali di una funzione (quelle dichiarate nella lista dei parametri o nel corpo della funzione) hanno normalmente permanenza in memoria automatica.

La parola chiave auto dichiara esplicitamente le variabili con permanenza in memoria automatica.

Per impostazione predefinita, le variabili locali hanno permanenza in memoria automatica, così la parola chiave auto si usa raramente.

Per il resto del testo chiameremo le variabili con permanenza in memoria automatica semplicemente variabili automatiche.

Prestazioni

La tecnica di memorizzazione automatica è un mezzo per risparmiare memoria, poiché le variabili automatiche esistono solo quando ce n'è necessità.

Esse sono create quando si entra in una funzione e distrutte quando si esce da essa.

Classe di memoria statica

Le parole chiave `extern e static` si usano nelle dichiarazioni degli identificatori per le variabili e le funzioni con permanenza in memoria statica.

Gli identificatori con permanenza in memoria statica esistono dal momento in cui il programma inizia l'esecuzione fino a che il programma termina.

Per le variabili `static` la memoria è allocata e inizializzata soltanto una volta, prima che il programma inizi l'esecuzione.

Per le funzioni, il nome della funzione esiste quando il programma inizia l'esecuzione.

Tuttavia, anche se le variabili e i nomi di funzione esistono dall'avvio dell'esecuzione del programma, ciò non significa che a questi identificatori si possa accedere per tutta la durata del programma.

La permanenza in memoria e il campo d'azione (dove si può usare un nome) sono aspetti separati.

Vi sono diversi tipi di identificatori con permanenza in memoria statica: gli identificatori esterni (come le variabili globali e i nomi di funzione) e le variabili locali dichiarate con lo specificatore della classe di memoria `static`.

Le variabili globali e i nomi di funzione sono della classe di memoria `extern` per impostazione predefinita.

Le variabili globali sono create ponendo le dichiarazioni di variabile all'esterno della definizione di qualsiasi funzione; esse mantengono i loro valori per tutta l'esecuzione del programma.

Le variabili globali e le funzioni possono essere menzionate da una qualunque funzione definita in seguito alle loro dichiarazioni o definizioni nello stesso file.

Questo giustifica l'uso di prototipi di funzione. Quando includiamo `stdio.h` in un programma che chiama `printf`, il prototipo di funzione è posto all'inizio del nostro file, per rendere noto il nome `printf` al resto del file.

Osservazione di ingegneria del software

Definire una variabile globale invece che locale permette che si verifichino effetti secondari non voluti quando una funzione che non ha necessità di accedere alla variabile la modifica accidentalmente o intenzionalmente.

In generale, le variabili globali vanno evitate, tranne in certe situazioni con critici requisiti di prestazioni.

Osservazione di ingegneria del software

Le variabili usate solo in una data funzione devono essere definite come variabili locali in quella funzione invece che come variabili esterne.

Le variabili locali dichiarate con la parola chiave `static` sono ancora conosciute solo nella funzione in cui sono definite, ma, diversamente dalle variabili automatiche, le variabili locali `static` mantengono il loro valore quando si esce dalla funzione.

La volta successiva che la funzione è chiamata, una variabile locale `static` contiene il valore che aveva prima che si uscisse dalla funzione la volta precedente.

L'istruzione seguente dichiara la variabile locale `count` come `static` e la inizializza a 1.

```
static int count = 1;
```

Se non vengono inizializzate esplicitamente, tutte le variabili numeriche con permanenza in memoria statica sono inizializzate automaticamente a zero.

Le parole chiave `extern` e `static` hanno un significato speciale quando sono esplicitamente applicate a identificatori esterni.

Regole per il campo d'azione

Il campo d'azione di un identificatore è la porzione del programma in cui l'identificatore può essere menzionato.

Ad esempio, quando definiamo una variabile locale in un blocco, essa può essere menzionata solo dopo la sua definizione in quel blocco o nei blocchi annidati all'interno di esso.

I quattro campi d'azione per gli identificatori sono il campo d'azione esteso alla funzione, il campo d'azione esteso al file, il campo d'azione esteso al blocco e il campo d'azione esteso al prototipo di funzione.

Le etichette (identificatori seguiti da due punti, come start:) sono gli unici identificatori con il campo d'azione esteso alla funzione.

Le etichette possono essere usate ovunque nella funzione in cui compaiono, ma non possono essere menzionate al di fuori del corpo della funzione.

Le etichette sono usate nelle istruzioni switch (come le etichette case).

Le etichette sono nascoste nella funzione in cui sono definite.

Questo occultamento (chiamato più formalmente occultamento delle informazioni, in inglese "information hiding") è un mezzo per implementare il principio del minimo privilegio, un principio fondamentale per una buona ingegneria del software.

Nel contesto di un'applicazione, il principio afferma che al codice deve essere garantita soltanto la quantità di privilegi e di accessi necessaria per eseguire il suo compito designato, ma non di più.

Un identificatore dichiarato al di fuori di una qualsiasi funzione ha un campo d'azione esteso al file.

Un tale identificatore è "conosciuto" (cioè accessibile) in tutte le funzioni dal punto in cui l'identificatore è dichiarato fino alla fine del file.

Le variabili globali, le definizioni di funzione e i prototipi di funzione posti al di fuori di una funzione hanno tutti il campo d'azione esteso al file.

Gli identificatori definiti dentro un blocco hanno il campo d'azione esteso al blocco.

Il campo d'azione esteso al blocco termina alla parentesi graffa destra (}) che chiude il blocco.

Le variabili locali definite all'inizio di una funzione hanno il campo d'azione esteso al blocco, come anche i parametri della funzione, anch'essi considerati variabili locali dalla funzione.

Qualsiasi blocco può contenere definizioni di variabili.

Quando i blocchi sono annidati e un identificatore in un blocco esterno ha lo stesso nome di un identificatore in un blocco interno, l'identificatore nel blocco esterno è nascosto finché il blocco interno non termina.

Ciò significa che mentre si esegue nel blocco interno, il blocco interno vede il valore del proprio identificatore locale e non il valore dell'identificatore dal nome identico nel blocco che include lo stesso blocco interno.

Le variabili locali dichiarate `static` hanno ancora il campo d'azione esteso al blocco, anche se esistono da prima che il programma si avvii.

Pertanto, la permanenza in memoria non si ripercuote sul campo d'azione di un identificatore.

Gli unici identificatori con il campo d'azione esteso al prototipo di funzione sono quelli usati nella lista dei parametri del prototipo di funzione.

Come precedentemente accennato, i prototipi di funzione non richiedono nomi nella lista dei parametri: sono necessari solo i tipi.

Se si usa un nome nella lista dei parametri del prototipo di funzione, il compilatore ignora il nome.

Gli identificatori usati in un prototipo di funzione possono essere riutilizzati altrove nel programma senza ambiguità.

Errore comune di programmazione

Usare accidentalmente lo stesso nome per un identificatore in un blocco interno identico a quello usato per un identificatore in un blocco esterno, quando di fatto volete che l'identificatore nel blocco esterno sia attivo per la durata del blocco interno, è causa di malfunzionamenti.

Prevenzione di errori

Evitate i nomi di variabili che nascondono nomi in campi d'azione esterni.

Esempio

Il programma successivo illustra alcuni aspetti relativi ai campi d'azione con variabili globali, variabili locali automatiche e variabili locali `static`.

Una variabile globale `x` è definita e inizializzata a 1 (riga 9).

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

Nella funzione `main` una variabile locale `x` è definita e inizializzata a 5 (riga 13).

Questa variabile viene quindi stampata per mostrare che la x globale è nascosta in main. In seguito, in `main` è definito un nuovo blocco con un'altra variabile locale `x` inizializzata a 7 (riga 18).

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

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.

```
1  // Programma C
2  // Verifica del campo d'azione delle variabili.
3  #include <stdio.h>
4
5  void useLocal(void); // prototipo di funzione
```

```
6  void useStaticLocal(void); // prototipo di funzione
7  void useGlobal(void); // prototipo di funzione
8
9  int x = 1; // variabile globale
10
11 int main(void)
12 {
13    int x = 5; // variabile locale per main
14
15    printf("local x in outer scope of main is %d\n", x);
16
17    { // inizio di un nuovo campo d'azione
18       int x = 7; // variabile locale nel nuovo campo d'azione
19
20       printf("local x in inner scope of main is %d\n", x);
21    } // fine del nuovo campo d'azione
22
23    printf("local x in outer scope of main is %d\n", x);
24
25    useLocal(); // useLocal ha una x locale automatica
26    useStaticLocal(); // useStaticLocal ha una x locale statica
27    useGlobal(); // useGlobal usa una x globale
28    useLocal(); // useLocal reinizializza una x locale
29    useStaticLocal(); //  x locale statica conserva il valore
30    useGlobal(); // la x globale conserva pure il suo valore
31
32    printf("\nlocal x in main is %d\n", x);
33 }
34
35 // reinizializza la variabile locale x durante ogni chiamata
36 void useLocal(void)
37 {
38    int x = 25; // inizializzata ogni volta
39
40    printf("\nlocal x in useLocal is %d after enter useLocal\n", x);
41    ++x;
42    printf("local x in useLocal is %d before exiting useLocal\n", x);
43 }
44
```

```
45  // inizializza la variabile statica locale x solo la
46  // prima volta che la funzione e' chiamata; il valore di x e'
47  // conservato tra una chiamata e l'altra a questa funzione
48  void useStaticLocal(void)
49  {
50     // inizializza x solo una volta
51     static int x = 50;
52
53     printf("\nlocal static x is %d on entering
                                          useStaticLocal\n", x);
54     ++x;
55     printf("local static x is %d on exiting useStaticLocal\n" ,x);
56  }
57
58  //  useGlobal modifica la variabile globale x in ogni chiamata
59  void useGlobal(void)
60  {
61     printf("\nglobal x is %d on entering useGlobal\n", x);
62     x *= 10;
63     printf("global x is %d on exiting useGlobal\n", x);
64  }
```

```
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
```

Il programma definisce tre funzioni che non ricevono argomenti e non restituiscono niente.

La funzione `useLocal` definisce una variabile automatica `x` e la inizializza a 25 (riga 38). Quando `useLocal` è chiamata, la variabile viene stampata, incrementata e stampata di nuovo prima che si esca dalla funzione.

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

La funzione `useStaticLocal` definisce una variabile statica `x` e la inizializza a 50 nella riga 51 (ricordate che la memoria per le variabili `static` è allocata e inizializzata solo una volta prima che il programma inizi l'esecuzione).

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.

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

La funzione `useGlobal` non definisce alcuna variabile.

Pertanto, quando essa si riferisce alla variabile `x`, viene usata la x globale (riga 9).

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

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

Infine, il programma stampa la variabile locale `x` di nuovo in `main` (riga 32) 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.

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.