Variabili

Una variabile denota una locazione di memoria

Gli attributi principali di una variabile sono:

- Nome
  È un identificatore che serve per nominare la variabile.
- Tipo
  Determina i valori che la variabile può contenere e la dimensione della locazione di memoria.

Ogni varibile x usata in un programma va preceduta dalla dichiarazione del suo tipo. In base a tale dichiarazione, viene stabilita la dimensione della locazione associata a x.

Esempio

```
int  n;
```

La variabile `n` è il nome di una locazione di memoria che può contenere la rappresentazione binaria di un intero di tipo `int`.

- Valore
  Il valore di una variabile x in un determinato istante t dell'esecuzione del programma è dato dal contenuto della locazione associata a x al tempo t

  L'ANSI C non stabilisce quale debba essere il valore iniziale di una variabile. È possibile assegnare a una variabile un valore iniziale nella sua dichiarazione.

Esempio

```
int a, b = 5, c = 10;
...
a = -1; // (A)
b = -2; // (B)
```

Vengono dichiarate tre variabili `a`, `b` e `c` di tipo `int`
Il valore iniziale di `b` è 5, il valore iniziale di `c` è 10, il valore iniziale di `a` non è specificato e dipende dal sistema
Dopo l'istruzione (A), `a` vale -1, `b` vale 5, `c` vale 10.

Dopo l'istruzione (B), `a` vale -1, `b` vale -2, `c` vale 10.

Funzioni

- La risoluzione efficace di problemi si basa sulla *decomposizione* in sottoproblemi, ognuno dei quali può essere risolto in maniera indipendente (programmazione top-down).

  Questa metodologia di programmazione è attuabile in C mediante l'uso di funzioni.

- Una funzione può essere vista come un sottoprogramma che svolge un ben preciso compito. La comunicazione di informazioni fra una funzione e l'ambiente chiamante avviene attraverso il meccanismo del passaggio dei parametri.

- È buona regola di programmazione documentare con un commento il comportamento della funzione:
  - Significato dei parametri e del valore restituito.
  - Eventuali condizioni sui parametri.

```
/* Dati x > 0 e y>= 0, restituisce x^y */

int power(int x, int y)
  ...
```


Definizione di una funzione

```
<type> nomeFunzione (<lista parametri> ) {
     <dichiarazione variabili locali>
     <istruzioni>
  }
```

- nomeFunzione è il nome della funzione.
- `type` è il tipo restituito dalla funzione.
  - Il tipo `void` indica che la funzione non restituisce alcun valore.
  - In caso non venga specificato alcun tipo, il compilatore assume che il tipo della funzione sia `int`.
- `lista parametri` è una lista di dichiarazioni di variabili distinte separate da virgole e denota i parametri formali della funzione. La lista può essere vuota.

```
double f(int m, int n){
  ...
  }
```

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

Variabili locali

Le variabili locali di una funzione sono:

- I parametri della funzione.
  Il valore iniziale dei parametri formali è stabilito quando la funzione è chiamata attraverso il meccanismo del passaggio dei parametri.
- Le variabili dichiarate all'inizio della funzione, prima di ogni altra istruzione.

Le variabili locali a una funzione `f()` sono visibili esclusivamente all'interno di `f()`.

Esempio

```
int f(){
 int n;     // n e' visibile solo in f()
 ...
}

int main(){
  int n;     // n e' visibile solo in main()
  ...
}
```


L'istruzione `return`

L'istruzione `return` permette a una funzione di restituire un valore al chiamante. La sintassi è:

```
return   <espressione>;
```

(1). Viene valutata `<espressione>` (ossia, ne viene calcolato il valore). Se necessario, tale valore viene convertito al tipo della funzione.

Esempio

```
double f(){
  int n;
  ...
  return n;
}
```

Il compilatore inserisce automaticamente un cast a `double` del valore di `n` prima che venga restituito. È come se fosse scritto

```
double f(){
  int n;
  ...
  return (double) n;
}
```


(2). Il valore determinato in (1) viene restituito al chiamante e l'esecuzione della funzione termina.

Sia `return` che `<espressione>` sono opzionali. In tal caso:

- Se `return` non è seguito da alcuna espressione, non viene restituito alcun valore.
- Se manca l'istruzione `return`, la funzione termina quando vengono eseguite tutte le istruzioni nel corpo della funzione e non viene restituito alcun valore.

Nota

L'ANSI C è ammette la definizione di funzioni in cui manca l'istruzione `return`, anche se la funzione ha tipo diverso da `void`. Se si compila con `gcc` usando opzione `-Wall` viene rilevato un warning.
Provare ad esempio a eliminare dal programma `cerchio.c` della scorsa l'istruzione

```
return area;
```


Prototipo di una funzione

Il prototipo di una funzione definisce:

- Il tipo restituito dalla funzione.
- Numero e tipo dei parametri della funzione

Esempio
Il prototipo della funzione `area()` definita in `cerchio.c` è

```
double area(int r)
```

oppure

```
double area(int)
```

Nota

Quando una funzione `f()` è chiamata, deve essere visibile al compilatore il prototipo di `f()`.

Chiamata di una funzione

Per chiamare una funzione occorre scrivere il suo nome seguito da un opportuno elenco di espressioni (detti argomenti o parametri attuali) fra parentesi tonde.

Dal punto di vista sintattico, la chiamata di una funzione `f()` è una espressione E_f (infatti, la coppia di parentesi '`( )`' che segue il nome della funzione è un *operatore* avente priorità massima e associatività "left-to-right").

- Il tipo di E_f è quello di `f()`.
- Il valore di E_f è il valore restituito dalla chiamata a `f()`.

Passaggio dei paramerti

Suppoiamo che la funzione `f()` abbia prototipo

```
T f(T1 x1, T2  x2, ... ,Tn xn)
```

dove `T1` è il tipo di `x1`, ... e `T` è il tipo di `f()`.

Una chiamata a `f()` è una espressione della forma

```
f(<espr1>, ... , <esprn>)
```

che è valutata nel modo seguente:

(1). Passaggio dei parametri
Per ogni $k = 1, \ldots, n$ viene calcolato il valore di `<esprk>`, che viene poi assegnato alla variabile `xk` di `f()` (passaggio parametri per per valore).

Cast impliciti

- Il valore di `<esprk>` viene convertito al tipo `Tk` del parametro `xk`.
- Il valore restituito dalla funzione viene convertito al tipo `T`.

Il compilatore effettua i cast impliciti di questo tipo basandosi sul prototipo di `f()`.

Esempio
La funzione della libreria matematica `power()` (elevamento a potenza) ha il seguente prototipo (scritto nel file `math.h`):

```
double power(double a, double b) // calcola a elevato a b
```

Date le linee di codice

```
double p;
int m, n;
p = power(m,n)
```

Il compilatore inserisce automaticamente i seguenti cast:

```
p = power( (double) m, (double) n);
```

Il valore m^n è calcolato correttamente.

Prototipi di default

Se nessun prototipo è stato definito, il compilatore aggiunge automaticamente un prototipo di default, che potrebbe non essere corretto per `f()`, causando così errori in esecuzione.

Esempio
Consideriamo il programma `main.c` della scorsa lezione e supponiamo di cancellare il prototipo

```
double area(int r);
```

Il programma viene compilato, ma il comportamento è scorretto (come quando se si scrive un prototipo sbagliato).

- Se si compila con comando
  `gcc main.c calc.c`
  il compilatore non dà alcun messagio (neppure se si usa opzione `-ansi`).
- Se si compila con comando
  `gcc -Wall main.c calc.c`
  il compilatore segnala il warning
  `main.c:16: warning: implicit declaration of function 'area'`
  che è utile per capire la fonte dell'errore

Ordine di valutazione dei parametri

Non è definito l'ordine in cui i parametri vengono valutati. Questo può provocare problemi quando la valutazione di una espressione ha *effetti collaterali*.

Esempio

Data una funzione `h()` con prototipo

```
int h(int, int)
```

la chiamata

```
int x,y;
x = 5;
y = h( x++,  x++);
```

può produrre due effetti diversi.

- Se gli argomenti di `f()` sono valutati *da sinistra a destra*, il primo argomento passato a `f()` è 5, il secondo è 6 (`x` dopo il passaggio dei parametri vale 7). Equivale a:
  ```
  y = h(x, x+1);
  x = x+2;
  ```
- Se invece gli argomenti di `f()` sono valutati *da destra a sinistra*, il secondo argomento è 5, il primo è 6 (`x` dopo il passaggio dei parametri vale 7).

Esecuzione della funzione

(2). Dopo il passagio dei parametri, il controllo passa alla funzione chiamata.
Vengono eseguite le istruzioni nel corpo della funzione `f()`, dove i parametri `x1`, ..., `xn` hanno i valori iniziali determinati al punto (1).

(3). L'esecuzione termina quando viene eseguita l'istruzione `return` oppure quando sono state eseguite tutte le istruzioni nel corpo della funzione.
Se viene eseguita l'istruzione

```
return <espr>;
```

la funzione restituisce al chiamante il valore di `<espr>`.

Record di attivazione di una funzione

Area di memoria che contiene tutte le informazioni necessarie per l'esecuzione di una funzione (variabili locali della procedura, informazioni necessari per riprendere l'istruzione interrotta dalla chiamata, ...).

Il record di attivazione di una funzione viene allocato e deallocato dalla memoria in modalità LIFO (last-in-first-out), quindi l'area di memoria dedicata ai record di attivazione è gestita come uno stack (pila).

- Quando una funzione è chiamata viene posto in cima allo stack il suo record di attivazione (*push*);
- Quando la funzione termina, il record di attivazione è tolto (*pop*) dallo stack.

Il modo in cui lo stack evolve è impredicibile a compile time, ma può essere determinato solo a run time (durante l'esecuzione del programma).

Variabili globali

Una variabile dichiarata fuori da tutte le funzioni è detta variabile globale.

È visibile in tutte le funzioni definite dopo la sua dichiarazione ed esiste per tutta l'esecuzione del programma. Anche le variabili globali possono essere inizializzate esplicitamente.

Nota

Un programma può essere visto come un insieme di *definizioni* di variabili globali e di funzioni (eventualmente suddivise su più file), fra cui deve esserci una e una sola funzione `main()`.
Fuori dalle funzioni possono esserci solo dichiarazioni globali (variabili globali, prototipi,...), direttive al preprocessore (linee che iniziano con #) o commenti, ma non istruzioni.

Regole di visibilità (Scope rule)

Una variabile si dice *visibile* in un certo punto del programma se in tale punto il suo nome può essere usato (per leggere o modificare il suo valore).

- Una variabile globale è visibile ovunque a partire dal punto in cui è dichiarata.
- Una variabile locale è visibile solo nella funzione in cui è dichiarata.
- Shadow effect (effetto di *ombreggiatura*):
  Il nome interno fa ombra al nome esterno.

Esempio

```
int x, a, b;  /* variabili globali */

void f(int x){
 int a = 7;
 b = x + a;
}
...
```

Le variabili a e x locali a `f()` nascondono le variabili globali con lo stesso nome. Quando `f()` viene eseguita, alla variabile b globale viene assegnato la somma del valore di x locale a `f()` (il cui valore è noto solo quando `f()` è chiamata) e 7 (valore della variabile a locale a `f()`).

La visibilità di un nome è determinabile a compile time (ossia, in fase di compilazione), in quanto dipende solamante dalla struttura del programma e non dall'esecuzione.

Tempo di vita di una variabile

- Le variabili globali sono allocate in memoria per tutta la durata del programma. Sono allocate in un'area dati particolare, chiamata area dati globali.
- Una variabile locale definita nella funzione `f()` è allocata in memoria nel record di attivazione di `f()` esclusivamente durante l'esecuzione di `f()`.

Nota
Le variabili globali e le variabili dichiarate in `main()` hanno lo stesso tempo di vita (la durata del programma). Tuttavia:

- Le variabili dichiarate in `main()` sono variabili locali e sono visibili solo in `main()`.
- Una variabile globale è visibile in qualunque funzione posta dopo la sua definizione.

Nota sull'uso variabili globali

Le variabili globali vanno usate con molta cautela.
Devono essere usate esclusivamente per contenere dati globali, che hanno un significato rilevante all'interno del programma.

- NON vanno utilizzate per passare valori fra le funzioni, in quanto le funzioni devono comunicare con il chiamante in modo trasparente tramite i parametri. Infatti, una funzione deve essere il più possibile indipendente dal contesto in cui è definita, in modo da poter essere riutilizzabile in ambiti diversi.
- L'abuso di variabili globali diminuisce la leggibilità del codice (risulta difficile controllare chi modifica la variabile globale); il programma risulta poco modulare (le funzioni non sono "autocontenute", ma dipendono da valori esterni) e quindi difficoltoso da modificare.