Variabili e Funzioni

Luca Abeni

March 18, 2014

Programmi C

- Un programma C e composto da funzioni e variabili
  - Variabile: memorizza valori appartenenti ad un insieme di definizione (dipendente dal tipo della variabile)
  - Funzione: istruzioni che specificano le operazioni da compiere sulle variabili
- Programma: descritto da funzione `int main()`
  - Algoritmo scomponibile in sotto-algoritmi $\rightarrow$ una funzione C può invocare altre funzioni
  - Funzioni "fornite" dal compilatore (`printf()`, `sqrt()`, ...) o implementate nel programma
- Variabili "visibili" da tutte le funzioni, o solo da una
  - Variabili globali vs Variabili locali

Identificatori

- Ogni "entità" ha un nome...
- ...Che la identifica $\Rightarrow$ Identificatore!
  - Case sensitive (`int variabile` è diverso da `int Variabile`)
  - Composto da lettere, cifrie e "_"
  - Non può cominciare con una cifra
- Inoltre, gli identificatori devono essere diversi da parole chiave riservate del linguaggio:

`auto`, `break`, `case`, `char`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extern`, `float`, `for`, `goto`, `if`, `int`, `long`, `register`, `return`, `short`, `signed`, `sizeof`, `static`, `struct`, `switch`, `typedef`, `union`, `unsigned`, `void`, `volatile`, `while`

Variabili

- Variabile
  - Coppia (n, v) con n nome e $v \in \mathcal{I}$ ($\mathcal{I}$: insieme di definizione)
  - Identificatore (nome) associato ad una o piu' locazioni di memoria
- Tipo di una variabile: ne specifica l'insieme di definizione
  - Numero di locazioni di memoria usate
  - Come interpretare la sequenza di bit memorizzata in tale memoria
- Classe di Memorizzazione: specifica tempo di vita e visibilità
  - Tempo di vita: quando è creata (e distrutta) l'associazione fra identificatore e locazioni di memoria?
  - Visibilità quali funzioni del programma possono usare la variabile?
  - Noi qui semplificheremo un po' le cose...

Dichiarazione e Definizione di Variabili

- Per poter usare una variabile, devo averla prima dichiarata
  - Ricordi? Differenza fra *dichiarazione* e *definizione*
  - Per le variabili, spesso coincidono
    - Eccezione: variabili `extern`
- Definizione di variabilie: *tipo* nome;
  - Notare il ; finale
  - Nota: due variabili dello stesso tipo possono essere definite assieme, separandole tramite virgola
  - `int n1; int n2` è equivalente a `int n1, n2;`
- Variabile locale: definita all'inizio di un blocco
- Variabile globale: definita fuori da ogni blocco

Tipi di Variabili in C

- Pochi tipi scalari fondamentali:
  - `char` (carattere - 8 bit)
  - `int` (numero intero)
  - `float`, `double` (numeri in virgola mobile)
- Il C99 aggiunge `_Bool` ed altri tipi (`int`*n*`_t` ed `uint`*n*`_t`)
  - Usabili solo dopo aver incluso appositi header file (`stdbool.h`, `stdint.h`, etc...)
- Attributi che modificano i tipi:
  - `short`, `long`
  - `unsigned`, `signed`

Tipi in C: Esempi

- `unsigned int` → numeri naturali
- `long int` → numeri interi espressi su più bit
  - Quanti? Dipende dal compilatore...
- `long long int` → numeri interi espressi su ancora più bit
- `char c`: caratteri rappresentati tramite (extended) ASCII
  - Numero di 8 bit → generalmente contiene valori $[0, 255]$
  - Si possono indicare direttamente i caratteri racchiudendoli fra ’ (virgoletta singola): `c = 'a';`
  - Attenzione!!! Non confondere con " (virgolette doppie) usato per le stringhe!

Visibilità delle Variabili

- Quali parti di codice/funzioni possono usare una determinata variabile?
  - Visibilità per blocchi di codice
- Variabili locali: usabili solo nel blocco di codice dove sono dichiarate

```
...
if (delta == 0) {
    double x;

    x = -b / (2 * a);
}
...
```

  - `x` è usabile (visibile) solo dentro a questo blocco
- Variabili globali: usabili in qualsiasi blocco di codice (contenuto nel file dove sono dichiarate)

Esempio

```
int main()
{
  int x;

  x = 5;
  {
    int x;

    x = 7;
    printf("Blocco interno: x=%d\n", x);
  }
  printf("Blocco esterno: x=%d\n", x);

  return 0;
}
```

- La definizione di `x` nel blocco interno "maschera" la variabile `x` definita nel blocco esterno

Tempo di Vita delle Variabili

- Generalmente dipende da dove sono definite
  - Eccezione: variabili locali `static`
- Variabili locali: create quando si entra nella funzione dove sono definite
- Variabili globali: create quando il programma inizia ad eseguire
- Se esiste un'unica funzione (`main()`), non c'e' grossa differenza...
- Ma in caso ci siano altre funzioni definite nel programma, la differenza è notevole!
  - Se una funzione `f()` è invocata più volte, le variabili locali in essa definite sono create e distrutte ogni volta!!!
  - Variabili locali: non inizializzate; variabili globali: generalmente inizializzate a 0

Inizializzazione delle Variabili

- Descrizione generica di un algoritmo
  - Una variabile non ha alcun valore associato fino a che non viene fatto un assegnamento
- Programma C
  - Le variabili locali sono non inizializzate: hanno un valore casuale fino al primo assegnamento
  - Le variabili globali sono inizializzate a 0
- Le variabili locali vanno sempre inizializzate prima di usarle!!!
  - Sintassi semplificata per dichiarazione seguita da inizializzazione (`int i; i = 10;` $\rightarrow$ `int i = 10;`)

Esempio

```
int main()
{
  double a = 1, b = 5, c = 4;
  int x = 10, y = 20;
  char ch = 'a';
...
```

→

```
int main()
{
  double a;
  double b;
  double c;
  int x;
  int y;
  char ch;

  a  = 1;
  b  = 5;
  c  = 4;
  x  = 10;
  y  = 20;
  ch = 'a';
...
```

Funzioni

- Funzione
  - Blocco di istruzioni che agiscono su variabili (globali o locali)
  - Nome (identificatore) associato, permette di invocarne l'esecuzione
  - Può ricevere in ingresso dei parametri
  - Può ritornare un valore
- Per alcuni versi simile a funzione matematica
  - Mappa valori dei parametri in valori di ritorno
- Per altri versi differente
  - Può non avere parametri in ingresso
  - Può non avere valori di ritorno
  - Può avere effetti collaterali

Dichiarazioni di Funzione

- Come per variabili: per invocare una funzione devo averla prima dichiarata
  - In realtà è un po' più complesso di così...
  - ...Ma semplifichiamo applicando questa semplice regola
- Dichiarazione di funzione: *tipo di ritorno* identificatore(*parametri*) ;
  - *tipo di ritorno*: identifica l'insieme di definizione del valore di ritorno (vedi tipi delle variabili)
  - *parametri*: lista di dichiarazioni *tipo* nome separate da virgole
    - Eventualmente lista vuota se parametri non specificati
    - `void` se non ci sono parametri

Dichiarazioni di Funzione - Prototipo

- Dichiarazione di funzione: prototipo
  - Specifica che una funzione esiste
  - Specifica i suoi evuentuali parametri
  - Specifica un eventuale valore di ritorno
- Non specifica le istruzioni che compongono la funzione (corpo della funzione) ne' le sue variabili locali
- Equivalente matematico: specifico nome, dominio e codominio di una funzione
  - $f() : \mathcal{R} \rightarrow \mathcal{R}$
  - `double f(double x);`

Definizioni di Funzione

- Specifica le istruzioni che compongono una funzione
- Definizionedi funzione: *tipo di ritorno* identificatore(*parametri*) (nota: no ;) seguita dal blocco contenente le istruzioni (corpo)
  - In pratica, contiene anche il prototipo della funzione
- Il corpo di una funzione è un blocco di istruzioni: può contenere definizioni di variabili private
- Inoltre, può usare i parametri come fossero variabili (inizializzate dal chiamante!)
- Termina con `return`
  - Utilizzabile per specificare il valore di ritorno

Parametri Formali ed Attuali

- Parametri formali: specificati nel prototipo
  - Si specificano tipo e nome
  - Equivalenti a variabili pre-inizializzate...
  - Esempio: `double discriminante(double a, double b, double c);`
- Parametri attuali: valori usati per invocare la funzione
  - Possono variare da invocazione ad invocazione
  - Risultato di valutazione di espressioni
  - Esempio: `delta = discriminante(1, 5, 4);`
- Informalmente, un parametro formale è un nome che viene associato al valore del corrispondente parametro attuale

Invocazione di Funzioni

- Una funzione è invocabile specificandone il nome, con fra parentesi i valori da assegnare ai parametri attuali
  - Esempio: `f = discriminante(a, b, c);`
- I valori fra parentesi (parametri attuali) possono essere il risultato di espressioni
  - Esempio: `printf("somma = %d", a + b);`
- Passaggio di parametri per valore:
  - Invocazione di funzione $\rightarrow$ valori dei parametri attuali copiati nello spazio di memoria allocato ai parametri formali
  - Se la funzione invocata modifica il valore dei parametri formali, no conseguenze sul chiamante

Esempio - Funzione Sbagliata

```c
#include <stdio.h>

void non_scambia(int a, int b)
{
  int tmp;

  tmp = a;
  a = b;
  b = tmp;
}

int main()
{
  int v1 = 1, v2 = 2;

  printf("V1 = %d\tV2=%d\n", v1, v2);
  non_scambia(v1, v2);
  printf("V1 = %d\tV2=%d\n", v1, v2);

  return 0;
}
```

- La funzione `non_scambia()` vorrebbe scambiare i valori di 2 variabili...
- Ma non funziona!!! Perchè?