Laboratorio di Programmazione 1

Docente: dr. Damiano Macedonio

Lezione 5 – 31/10/2013

Le Variabili

Una variabile identifica una *porzione di memoria* destinata a contenere dei dati, *che possono essere modificati* durante l'esecuzione del programma.

- Le variabili *devono* essere dichiarate prima di essere utilizzate.
- La dichiarazione di una variabile ha il seguente formato:
  - `tipo nome;`
- La dichiarazione di una variabile può anche essere combinata con la sua inizializzazione:
  - `tipo nome = valore_iniziale;`

Nome di una Variabile

- I nome assegnati alle variabili devono rispettare alcune regole:
  - Deve iniziare con una lettera o un underscore _
  - Può essere seguito da una qualsiasi combinazione di lettere (maiuscole e minuscole), underscore _, e cifre da 0 a 9.
  - Non deve essere una parola riservata (es. `int`).

- Le lettere maiuscole e minuscole sono diverse:
  - `name` ≠ `Name` ≠ `NAME`, ecc...

Tipo di una Variabile

- Il linguaggio C fornisce questi tipi fondamentali:
  - `int`: numeri interi.
  - `float`: numeri in virgola mobile (precisione singola).
  - `double`: numeri in virgola mobile (precisione doppia).
  - `char`: singolo carattere.

- Utilizzando i tipi fondamentali è possibile costruire dei tipi derivati (prossime lezioni).

- Ad ogni tipo fondamentale è associato un intervallo di valori che dipende dallo *spazio allocato in memoria* per contenere quel particolare tipo di dato.
  - La quantità di memoria può dipendere dalla macchina (es. 32 o 64 bit) e dal compilatore, ma ci sono delle garanzie di quantità minima.
  - Per sapere lo spazio (in byte) effettivamente allocato in memoria si usa la funzione sizeof( ).

Il Tipo Int

- Il tipo `int` permettere di memorizzare numeri interi positivi e negativi.
  - Sequenza di una o più cifre eventualmente precedute dal simbolo meno.
  - Non usare il punto per separare le migliaia!
  - Es. 10, -30, 0.

- Base diversa da quella decimale:
  - Base ottale:
    - Si ottiene anteponendo uno `0` prima del numero.
    - Le cifre valide sono 0-7.
    - Es. 050 è il valore 50 in base 8 che corrisponde a 40 in base 10.
  - Base esadecimale:
    - Si ottiene anteponendo `0x` prima del numero.
    - Le cifre valide sono 0-9 A-F.
    - Es. 0xA93 è il valore A93 in base 16 che corrisponde a 2707 in base 10.

Il Tipo Int

- Formattazione (`printf` e `scanf`):
  - `%i`: per gli interi in generale.
  - `%d`: per gli interi in base 10.
  - `%o` oppure `%#o`: per gli interi in base 8.
  - `%x` oppure `%#x`: per gli interi base 16.

- Memoria allocata:
  - Da 16 a 32 bit (tipicamente 32 bit).
  - Corrisponde alla dimensione di una parola (*word*) sul computer.

Il Tipo Float

- Il tipo `float` permette di memorizzare numeri contenenti cifre decimali (in virgola mobile).
  - La parte intera e la parte decimale sono separate da un punto.
  - Il valore deve essere seguito dalla costante `f` o `F`.
  - Si può omettere la parte intera e decimale, ma non entrambe o il punto:
    - 3. = 3.0
    - .3 = 0.3

- I valori in virgola mobile possono essere rappresentati anche usando la *notazione scientifica*.
  - `1.7e4` *corrisponde a* $1.7 * 10^4$
  - 1.7 è chiamato mantissa
  - 4 è chiamato esponente (positivo o negativo)

Il Tipo Float

- Formattazione (`printf` e `scanf`):
  - `%f`: per la formattazione normale.
  - `%e`: per la formattazione in notazione scientifica.
  - `%g`: per lasciar decidere a `printf` quale notazione utilizzare
    - Esponente compreso tra -4 e 5: formattazione normale, altrimenti notazione scientifica.

- Memoria allocata:
  - Tipicamente 32 bit.
  - Garantisce la rappresentazione di almeno 6 cifre di precisione (i.e. cifre per l'esponente).

Il Tipo Double

- Il tipo `double` permette di rappresentare numeri che contengono cifre decimali (in virgola mobile) con una precisione circa doppia rispetto al tipo float.

- Per default le costanti in virgola mobile sono considerate double.

- Formattazione (`printf` e `scanf`): uguale a quella del tipo `float`.
  - In alcuni sistemi si deve utilizzare `%lf` con la funzione `scanf` per evitare perdite di precisione.

- Memoria allocata:
  - Tipicamente 64 bit.
  - Garantisce la rappresentazione di almeno 10 cifre di precisione (i.e. cifre per l'esponente).

Il Tipo Char

- Il tipo `char` permette di memorizzare un singolo carattere.

- Un valore di tipo carattere si forma racchiudendo un carattere tra una coppia di apici *singoli.*
  - `'a' ≠ "a"`
  - Le sequenze formate da una backslash `\` e un carattere sono considerate come un singolo carattere (es. `\n`).

- Formattazione (`printf` e `scanf`): `%c`.

- Memoria allocata: 8 bit.

Specificatori

- Gli specificatori permettono di modificare l'intervallo di valori rappresentabili con un certo tipo.
  - Vanno indicati nella dichiarazione di una variabile prima del tipo:
    `spec tipo nome;`

- `long int / long double`
  - Permette di dichiarare un intero (double) con una precisione estesa (aumentare l'intervallo di valori rappresentabili):
    - es. `long int x;`
  - Memoria allocata: almeno 32 bit (int) / almeno 64 bit (double).
  - I valori sono specificati aggiungendo una `L` (o `l`) alla fine del valore:
    - es. `x = 2343432342332L;`
  - Per la formattazione si antepone la lettera `l` ai formati `i`, `o`, `x` e la lettera `L` ai formati `f`, `e`, `g`:
    - `Per int: %li, %lo, %lx`
    - Per double: `%Lf` , `%Le` , `%Lg`.

- `long long int`
  - Memoria allocata: almeno 64 bit.
  - Per la formattazione si antepone `ll` ai formati `i`, `o`, `x`.

Specificatori

- `short int`
  - Permette di dichiarare una variabile utilizzata per memorizzare interi piccoli (diminuire l'intervallo di valori rappresentabili).
    - es. `short int x;`
  - Per la formattazione si antepone la lettera `h` ai formati `i`, `o`, `x`:
    - `%hi`, `%ho`, `%hx`.
  - Memoria allocata: almeno 16 bit.

- `unsigned int`
  - Permette di dichiarare una variabile utilizzata per memorizzare soltanto numeri positivi (raddoppia l'intervallo di valori positivi rappresentabili).
    - es. `unsigned int x;`
  - I valori sono specificati aggiungendo la lettera `U` (o u) alla fine del valore.
    - es. `x = 2342332U;`

Operatori Aritmetici Binari

- Operatori aritmetici binari:
  - `+` addizione
  - `-` sottrazione
  - `*` moltiplicazione
  - `/` divisione
  - `%` modulo
- Le regole di *precedenza* o *priorità* tra gli operatori sono quelle classiche dell'aritmetica.
  - Per modificare l'ordine di valutazione si usano le parentesi tonde.
- È possibile indicare un'espressione come argomento della `printf` senza assegnarne il valore ad una variable.
  - Per stampare il simbolo % all'interno di una stringa si deve usare il doppio simbolo %.
    - `printf( "a %% b = %i\n", a % b );`

Operatori Aritmetici

- Il risultato prodotto dagli operatori aritmetici dipende dal tipo degli argomenti:
  - Se gli argomenti sono entrambi interi viene utilizzata l'aritmetica degli interi.
    - La divisione viene intesa come divisione tra interi e la parte decimale viene persa (si ottiene sempre un intero!).
  - Per usare l'aritmetica in virgola mobile almeno uno dei due argomenti deve essere un float o un double.

```
int a = 13;
int b = 2;
int d = a/b; → d=6
```

```
double a = 13.;
double b = 2.;
double d = a/b; → d=6.5
```

- Il simbolo – può essere utilizzato anche come operatore unario per cambiare il segno di un valore.

Conversione tra Tipi

- `double/float → int`
  - Se un valore in virgola mobile viene assegnato ad un intero, la parte decimale viene persa o troncata.

```
float x = 10.5;
int y = x; → y = 10
```

- `int → double/float`
  - Se un valore intero viene assegnato ad uno in virgola mobile non si ha alcuna modifica sul valore.
  - Conversione automatica.

```
int x = 10;
float y = x; → y = 10
```

Conversione tra Tipi

- Quando si esegue un'operazione aritmetica tra due interi, viene *sempre* utilizzata l'aritmetica degli interi *indipendentemente* dal tipo della variabile a cui assegnamo il risultato!

```
int x = 11;
double y = x / 2; → y = 5

int x = 11;
double y = x / 2.0; → y = 5.5

int x = 11;
double y = (double) x / 2; → y = 5.5
```

Conversione tra Tipi

- L'operatore di *cast* permette di convertire il valore di una variabile in un altro tipo (quando possibile):
  - Es.`(double) x`
- Ha la priorità più alta rispetto agli altri operatori aritmetici, ad eccezione dell'operatore unario -.
  - `(double) x / 2`
  - Prima converte il valore di x in un double e poi esegue la divisione.
- Non ha un effetto permante sulla variabile a cui viene appalicato.

Conversione tra Tipi

- Nelle espressioni, le conversioni di tipo avvengono operatore per operatore

  - ```
    float x = 28.0;
    int y = 7;
    int z = 2;
    float w = x / ( y / z ); → y = 9.333...
    ```

  - ```
    int x = 28;
    int y = 7;
    float z = 2.0;
    float w = x / ( y / z ); → y = 8.0
    ```

Operatori di Assegnamento

- Gli operatori aritmetici +, -, *, /, % possono essere combinati con l'operatore di assegnamento = per scrivere espressioni più compatte.

  - `count = count + 10;` *equivale a*
  - `count += 10;`

  - `a = a / (b+c);` *equivale a*
  - `a /= b+c;`

Esercizio 1

- Scrivere un programma C che prima chiede all'utente di inserire un intero, un float, un double e un carattere. Poi risponde stampando a video i valori inseriti formattati nella stringa:
  - "Il valore intero inserito è: ..."
  - "Il valore float inserito è: ..."
- Per i valori float e double stampare sia il formato normale che la notazione scientifica.

Esercizio 1 (Precisazioni)

- La funzione `scanf` con formato `%c` legge il primo carattere disponibile sullo standard input.

- Quando la `scanf` sta leggendo un carattere con il formato `%c`, il successivo carattere disponibile è considerato quello di input, *anche se è uno spazio vuoto*.
  - Se prima di eseguire la funzione `scanf` di un carattere sono stati inseriti altri valori da tastiera seguiti da un *enter*, allora tale carattere è il primo disponibile e la funzione `scanf` non aspetta l'inserimento di un altro carattere.
  - La funzione `scanf` ignora sempre gli spazi iniziali quando legge un numero intero oppure in virgola mobile.

- Per far funzionare la `scanf` con `%c` ignorando la presenza di spazi iniziali è necessario usare il formato "` %c`".
  - "` %c`" indica che l'input può contenere un numero arbitrario di spazi iniziali che saranno ignorati.

Esercizio 1 (Precisazioni)

- L’acquisizione tramite `scanf` di un valore double può essere eseguita con `%f`, ma questo può produrre delle perdite di precisione.
  - Il compilatore avverte attraverso dei warning.
  - Per eliminare i warning è necessario usare il formato `%lf`.

Esercizio 2

- Scrivere un programma C che richiede all’utente due valori e stampa il risultato dell’operazione di addizione, sottrazione, moltiplicazione e divisione considerandoli prima come interi e poi come valori in virgola mobile (usare la conversione tra tipi).