Classe III[a] A Inf – Informatica – Recupero Gennaio 2013

Riepilogo basi di programmazione

Variabili

Iniziando a parlare della programmazione in C (e C++) abbiamo considerato diversi tipi di variabili. Le variabili sono contenitori atte a contenere i dati di una elaborazione, ossia ad implementare la parte di memoria di un programma. In modo complementare abbiamo il cosiddetto codice (per noi realizzato tramite istruzioni C / C++), che realizza la parte logica e di elaborazione sui dati contenuti appunto nelle variabili. Proprio per lo stesso significato della parola si può comprendere che i contenuti delle variabili non restano usualmente immutati, tranne in alcuni casi, ma piuttosto evolvono opportunamente a seconda delle necessità e dell'elaborazione del programma.

Abbiamo già fatto notare che sul libro di testo di testo a pagina 216 e seguenti (paragrafo 3) c'è un buon riepilogo dei tipi di variabili, diversi dei quali abbiamo già utilizzato in alcune occasioni.

In particolare sono da noi state utilizzate le variabili di tipo char[1] (carattere, contenente un singolo carattere - ampiezza di 1 byte), int (contenenti un intero a 32 bit – ampiezza 4 byte), bool (contenenti un booleano – ampiezza 1 byte), float (numero con la virgola – ampiezza 4 byte) e double (numeri con la virgola in alta precisione – ampiezza 8 byte). Si ricorda che le differenze di ampiezza sono importanti soprattutto nel caso nel programma esista un alto numero di variabili di un certo tipo, di modo da minimizzare opportunamente l'occupazione di memoria.

In C / C++ qualunque variabile deve essere predichiarata nel programma prima di essere utilizzata, pena errori di compilazione. Le dichiarazioni vengono poste nella parte iniziale del programma ed ogni variabile avrà un suo opportuno indicatore di tipo, essendo il C / C++ un linguaggio tipizzato[2].

Le variabili sono nella metafora comune contenitori, come quelli che troviamo in cucina, ognuno adeguato per contenere un certo tipo di cibo (pasta, zucchero, caffè), ma nella realtà essi sono zone di memoria centrale allocate appositamente che risultano protette da eventuali operazioni di altre programmi o degli utenti. Proprio per questo ad ogni variabile è associabile un indirizzo che rappresenta la sua posizione in memoria.

Costanti

Siccome abbiamo parlato di variabili viene naturale chiedersi come si possano definire dei valori che siano sicuramente invariabili. Indichiamo qui il classico modo di definire una costante in C:

```
#define PIGRECO 3.14
```

Questo modo di scrivere comporta la definizione di una costante di nome PIGRECO (usualmente segnata tutta in lettere maiuscole), che sarà fissata al valore numerico opportuno. Tale valore non sarà successivamente modificabile (pena errori di compilazione) con nuove ulteriori assegnazioni, e quindi risulterà una costante.

1 E' possibile vedere questa variabile anche come intero corto con segno di 8 bit (1 byte)

2 Si ricorda che nei linguaggi non tipizzati, come i linguaggi di script (JavaScript ad esempio), le variabili possono contenere qualunque tipo di dato indifferentemente ed anzi anche cambiare tipo.

Istruzioni condizionali

In generale schematizzando esistono in un programma tre categorie di codice possibile: la sequenza (di istruzioni), le istruzioni condizionali e i cicli[3].

Il concetto di sequenza è il più intuitivo: si tratta di una serie di comandi di vario tipo dati uno di seguito all'altro, ed infatti eseguiti appunto in sequenza. Qui ci occuperemo del secondo tipo di istruzioni, ossia di quelle istruzioni che permettono di decidere se una certa istruzione, o meglio un certo gruppo di istruzioni sia da eseguire oppure no. In pratica quindi le istruzioni condizionali realizzano una esecuzione condizionata del codice, in base a certe condizioni. A seconda che tali condizioni si verifichino, il codice sottoposto alla relativa condizione viene eseguito oppure no.

La più semplice forma di codice condizionato è quello che condiziona una sola istruzione tramite l' istruzione if; ad esempio:

```
if (r == 0)  printf("E' divisibile.");
```

in realtà l'istruzione if non opera sempre in modo così limitato, ma possono condizionare gruppi di istruzioni; ad esempio:

```
if (n2 != 0)
{
   q = n1 / n2;
   r =  n1 % n2;
}
```

In questo caso la condizione "se n2 è diverso da 0" condiziona l'esecuzione di due istruzioni ed esse vengono eseguite solo se essa è vera. Abbiamo visto che l'istruzione if viene seguita sempre da una condizione che definisce se il codice sotto condizione verrà eseguito o meno, vale a dire verrà eseguito se la condizione è vera e non eseguito se essa è falsa. Ovviamente data la stessa condizione, in due momenti diversi dell' esecuzione del programma, in un momento si potrà avere che la condizione risulta verificata, in un altro passaggio che non risulta tale. In definitiva ciò per dire che come le variabili cambiano il loro valore durante l'esecuzione di un programma, anche il risultato ultimo di una condizione logica cambia usualmente durante l'esecuzione del programma stesso (e questo è dovuto proprio alla variazione di valori nelle variabili coinvolte nella condizione in questione[4]).

In alcuni casi si desidera però non solo condizionare l'esecuzione di una parte di codice, ma eseguire del codice alternativo nel caso un blocco di codice non sia eseguito. Programmare ciò è del tutto analogo all'affermazione "se ho i soldi vado al luna park, altrimenti faccio un giro in centro". Le due azioni qui prospettate infatti non sono contemporaneamente effettuabili e quindi i due "segmenti di codice" sono alternativi l'uno all'altro in quanto logica vuole che non sia possibile andare contemporaneamente al luna park ed in centro[5].

Un esempio di una alternativa di questo tipo potrebbe essere:

```
if (r == 0)
    printf("Il numero è divisibile.");
else
```

3 In realtà nei primi linguaggi di programmazione alle istruzioni condizionali (di tipo if...) venivano accostate più semplicemente le istruzioni di salto

4 Infatti una condizione non contenente valori variabili dà sempre un medesimo risultato

5 Ovviamente supponendo che il luna park non si trovi in centro

```
printf("Il numero non è divisibile.");
```

Il caso prospettato è molto semplice, ma comunque è chiaro che le due scritte sono ovviamente alternative l'una all'altra e non devono apparire assieme sullo schermo.

Strutture di condizione più complesse possono essere formate accodando opportunamente le condizioni in cascata con indicazioni else if.... ; si osservi l'esempio che segue:

```
if (n == 0)
      printf("E' zero.");
else if (n > 0)
      printf("E' maggiore di zero.");
else
      printf("E' minore di zero.");
```

in cui si vuole determinare se un numero sia uguale, maggiore o minore di 0.

Selezione

Una variante dell'istruzione if, creata per comodità, nel caso di numerose selezioni multiple, è l'istruzione switch. Essa opera sul tipo intero o su tipi che siano direttamente riconducibili agli interi (come il char).

La struttura dello switch è data dall'istruzione switch al seguito della quale viene messa tra parentesi la variabile in base alla quale si determina la selezione. Sotto all'interno di parentesi graffe vengono inseriti tutti gli opportuni casi di selezione, indicati con la sintassi case x:.... ove x sarà un numero od un carattere. A seguire ogni caso seguono le varie istruzioni non riquadrate da parentesi. Usualmente l'ultima istruzione sotto un certo caso è il break, che porta il controllo di esecuzione nello switch ad uscire dallo stesso. In realtà in alcuni casi è possibile non inserire tale terminatore. In questo caso ogni caso scritto in questo modo (ossia privo di break finale) esegue il suo codice e prosegue al caso successivo, eseguendo anche le istruzioni del caso che segue.

Se il valore della variabile di selezione non coincide con alcun valore di selezione (case), allora nessun codice viene eseguito. Altrimenti viene eseguito il codice seguente al caso corrispondente ed eventualmente anche quello dei casi successivi fino al primo break.

Consideriamo un esempio pratico:

```
int n, x;
n = 5;
switch(n)
{
      case 1: x = x + 1;  break;
      case 2: x = x + 2;  break;
      case 3: x = x + 3;  break;
      case 4: x = x + 4;  break;
}
```

in questo codice la selezione su n determina l'entità dell'incremento che avviene su x.

Con il valore di n inizializzato, però, nessuna delle alternative viene selezionata e quindi nessun incremento di x viene fatto. In realtà per coprire eventuali casi non contemplati dai valori di selezione, basta aggiungere un caso indicato come default: nel quale convergono tutti i casi alternativi alle selezioni indicate.

Ovviamente nel caso n ricadesse in uno dei casi indicati verrebbe eseguito semplicemente il codice (che sia

costituito da una o più righe di codice) relativo al caso medesimo.

Operatori logici e condizione composte

Seppur negli esempi precedenti e seguenti abbiamo proposto condizioni particolarmente semplici, non sempre è possibile controllare condizionamenti del codice o cicli con espressioni condizionali così elementari. Spesso è vero il contrario, ossia che le condizioni da inserire sono combinazione di più elementi e quindi risultano condizioni composite piuttosto complesse. Tali condizioni composite in realtà, a ben vedere risultano composte da codizioni logiche più semplici, messe assieme tramite operatori logici booleani, And Or Not. Si ricorda che tali operatori operano su valori booleani e proprio per questo sono in gradio di combinare più espressioni condizionali fra loro, in quanto qualunque condizione esprime in ultima analisi un valore booleano. Sarà quindi possibile scrivere espressioni condizionali composite quali:

```
if ((x == 0) && (y > 6))
...
while ((n != 2) || (n > 6))
...
```

ecc. Gli operatori booleani sono quindi molto importanti nella programmazione, e ciò è testimoniato dal fatto che tutti i linguaggi di programmazione posseggono almeno gli operatori logici fondamentali And (&& in C) , Or (|| in C), Not (! in C).

Cicli

Dopo il codice opportunamente condizionato, è opportuno considerare le strutture a ciclo che permettono sostanzialmente la ripetizione di parti di codice e sono direttamente legate alle vecchie istruzioni di salto ormai in disuso nei moderni linguaggi strutturati. Si ricorda che abbiamo considerato due tipi di ciclo: il ciclo for ed il ciclo while. La differenza più netta è che il ciclo while è più generico, mentre il ciclo for viene utilizzato in genere in situazioni in cui il numero di cicli da effettuare è noto e ben prefissato.

Iniziamo a considerare il ciclo for. Esso è una istruzione suddivisa in tre sottosezioni:

- inizializzazione
- condizione di permanenza nel ciclo
- incremento della variabile di ciclo

si consideri che un ciclo for ha sempre almeno una variabile di ciclo il cui valore è bene non venga modificato[6] dal programma. Un ciclo di questo tipo ripete se stesso per il numero di volte che la condizione di fine ciclo rende possibile, dopodichè esce. Si tenga presente che:

- il numero di cicli effettuati dipende dalla condizione di uscita e dal passo sulla variabile di ciclo
- è possibile uscire dal ciclo con una opportuna istruzione (condizionata) break
- è possibile avere cicli for infiniti dato un incremento nullo o condizioni di uscita evidentemente irraggiungibili

In un ciclo while non sono integrati nella istruzione di ciclo stessa l'inizializzazione della variabile di ciclo e la

6 Come ben dovreste sapere è possibile leggere in qualunque momento il valore della variabile di ciclo

variazione della stessa. Anche i cicli while possono avere situazioni che li portano a durare in modo indefinito. I cicli while hanno spesso meccanismi di uscita che non determinano univocamente il numero di passi del ciclo.

Esempi di cicli sono:

```
for (x = 0; x <= 100; x++)
{
     printf("%i \t", x);
}
```

che stampa una tabella di numeri da 1 a 100

e ad esempio:

```
n = 10;
while (n > 0)
{
    printf("%i \n",n);
    n--;
}
```

che contempla anch'esso una inizializzazione (n = 10), una condizione di permenenza nel ciclo (n > 0) e una variazione (n--) della variabile di ciclo.