Università degli Studi di Milano

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Lezione 5 – Controllo del flusso del programma

FABIO SCOTTI

Laboratorio di programmazione per la sicurezza

Indice

1. Concetto di blocco in C

In C i blocchi sono realizzati mediante l'uso delle parentesi graffe dentro le quali vi è una sequenza di istruzioni.

E' possibile dichiarare delle variabili locali all'interno di in blocco di istruzioni C.

I blocchi possono essere annidati a piacere: le dichiarazioni delle variabili locali seguono immediatamente la parentesi sinistra e precedono la parte eseguibile del blocco.

Ecco un esempio di blocco

```
int a, b;
....
{     int temp;   /*  parte dichiarativa del blocco */
      temp = a;
      a = b;
      b = temp;
}     /*ATTENZIONE da qui in poi la variabile temp cessa di esistere */
```

Valgono le usuali regole di scopo e di visibilità: le *variabili locali* sono visibili dal punto della loro dichiarazione fino alla fine del blocco cui appartengono, e le variabili locali nascondono, rendendo non accessibili, le variabili omonime globali, cioè dichiarate in blocchi più esterni. Questi concetti saranno ulteriormente dettagliati quando nel corso verranno affrontate le *funzioni*.

Chiediamoci se ciò che è contenuto fra le graffe del main risponde alla definizione di blocco. La risposta è si. Infatti il main è un blocco.

Durante il corso non useremo i blocchi *contenenti delle dichiarazioni* se non per il main.

2. Due costrutti per la scelta: IF e SWITCH

2.1 Il costrutto IF

E' composta da UNA espressione e UNO/DUE/TRE blocchi di codice (chiamati i "rami" dell' if)

IF con una istruzione oppure un blocco

```
if (espressione)
      istruzione1
```

Esempio

```
if (x==3)
      k = 4;
```

Usando un blocco

```
if (x==3)
{
      k = 4;
      c = k;
      temp = 3;
}
```

IF con due istruzioni

```
if (espressione)
      istruzione1
else istruzione2
```

Esempio

```
if (x==3)
      k = 4;
else
      k=0;
```

Naturalmente anche in questo caso al posto di una istruzione (k=4;) è possibile inserire un blocco di codice ({k=4; c=k; …}).

IF con tre/N istruzioni

```
if (expr1)
            istr1
else if (expr2)
            istr2
else if (expr3)
            istr3
else if (expr4)
            istr4
```

Ricordando sempre che le istruzioni terminano con il punto e virgola.

2.2 Il costrutto SWITCH

Il costrutto SWITCH è solitamente impiegato quando si devono controllare molti casi. Il codice infatti risulta essere più ordinato.

E' composto da delle etichette (i CASE) ed da un caso generale (DEFAULT). Ecco un esempio autoesplicativo di come si usa un costrutto CASE. Si noti che le etichette del case DEVONO essere delle costanti.

```
// conteggio dei tipi di caratteri
char c;
int n_cifre, n_separatori, n_altri;
...
```

```
c = ....;    // sia c il carattere da esaminare
switch (c)
      {
      case '0':
            n_cifre = n_cifre + 1;
            break;

      case '1':
            n_cifre = n_cifre + 1;
            break;
      .....
      case '9':
            n_cifre = n_cifre + 1;
            break;

      case ' ':
            n_separatori = n_separatori + 1;
            break;

      case '\n':
            n_separatori = n_separatori + 1;
            break;

      case '\t':
            n_separatori = n_separatori + 1;
            break;

      default:
            n_altri = n_altri + 1;
      }
...
```

3. I costrutti WHILE e FOR

3.1 Il costrutto WHILE

E' il costrutto più generale (da cui si possono ottenere tutti gli altri costrutti per l'iterazione) ed è quello che viene consigliato per il nostro corso.

La sua struttura è la seguente:

```
while (espressione)
      istruzione
```

oppure

```
while (espressione)
      {
      istruzione1
      istruzione2
      ....
      istruzionen
      }
```

L'espressione viene valutata e, se ha valore diverso da 0 (*vero*) viene eseguita l'istruzione successiva (che può anche essere un intero blocco di istruzioni, opportunamente delimitato dalle parentesi graffe). Una volta che quest'ultima è terminata, l'espressione viene valutata nuovamente e se è nuovamente vera, si ripete l'istruzione. Ciò si ripete fino a quando l'espressione ha valore 0 (*falso*), nel qual caso il controllo si trasferisce all'istruzione successiva al `while`.

Un ciclo while può essere eseguito 0 o più volte, poiché l'espressione potrebbe essere falsa già la prima volta. Si tratta di un ciclo a *condizione iniziale*: prima di eseguire il ciclo si valuta la condizione.

Il problema principale delle strutture iterative è che possono entrare in ciclo infinito. Questo accade se per un errore di programmazione o per qualche condizione esterna la espressione del while rimane sempre vera.

Ecco un esempio nel quale vengono sommati i numeri interi che l'utente immette da tastiera fino a quando non viene immesso uno zero.

```
#include <stdio.h>
int main()
{
int i, somma;
somma = 0;
i = 1;  // valore iniziale (serve solo per entrare nel while)
while (i != 0)
      {
      scanf ("%d" , &i);
      somma = somma + i;
      }
printf("La somma e' %d", somma);

}
```

3.2 Il costrutto FOR

Il costrutto FOR è equivalente al costrutto while, ma è da impiegarsi solo nel caso nel quale il numero di iterazioni E' NOTO A PRIORI. Se dobbiamo scandire un struttura dati che ha sempre N elementi, allora useremo un ciclo FOR.

Se invece dobbiamo continuare a chiedere all'utente un numero finché non viene immesso un numero pari a zero, allora impiegheremo un costrutto WHILE in quanto il numero di iterazioni non è noto a priori.

Un programmatore esperto non confonde mai queste due situazioni e usa sempre il costrutto più appropriato.

E' costrutto FOR è composto nel seguente modo:

```
for ( expr1 ; expr2 ; expr3 )
istruzione
```

Se si vuole iterare un blocco di codice nel seguente:

```
for ( expr1 ; expr2 ; expr3 )
      {
      istruzione1
      istruzione2
      ....
      istruzionen
      }
```

Dove

expr1 è l'espressione *iniziale*,

expr2 è l'espressione *booleana (del ciclo)*,

expr3 è l'espressione *incremento*).

L'espressione *iniziale* permette di inizializzare le variabili di ciclo, e viene eseguita una volta sola, prima di qualsiasi altra operazione.
Esempio expr1: `contatore = 0`.

Successivamente ad essa, viene valutata l'espressione del ciclo e, se questa ha valore diverso da 0 viene eseguita l'istruzione che costituisce il corpo del ciclo (o il blocco di istruzioni fra parentesi graffe).
Esempio expr2: `contatore < 9`.

Al termine dell'esecuzione del corpo del ciclo, viene valutata l'espressione-incremento, di solito per poter aggiornare i valori delle variabili di ciclo.
Esempio expr3: `contatore = contatore + 1`.

Quindi, si valuta nuovamente l'espressione del ciclo e cosi via. Il ciclo si ripete finché non si valuta come falsa l'espressione del ciclo - valore 0-.

Ecco un esempio. Avendo già dichiarato la variabile i è possibile scrivere

```
for ( i=1 ; i<=10 ; i=i+1 )
      {
      printf(" Indice = %d" , i);
      }
```

3.3 Scrittura di un ciclo FOR con un ciclo WHILE

A titolo di esercizio viene mostrato ora come è possibile scrivere un ciclo for con un ciclo while. Questo è da evitare nella prassi programmativa.

```
for ( expr1 ; expr2 ; expr3 )
      {
      istruzione1
      istruzione2
      ....
      istruzionen
      }
```

e' del tutto equivalente a

```
expr1;
while (expr2)
            {
            istruzione1
            istruzione2
            ....
            istruzionen
            expr3;
```

4. Forme compatte

Il linguaggio C consente ai programmatori di usare vari stili di scrittura. Ad esempio permette di usare le cosiddette *forme compatte* ovvero delle scritture alternative dei normali istruzioni espresse con un minor numero di caratteri.

Le presentiamo di seguito con degli esempi.

x *= y	Corrisponde a	x= x * y
y -= z+1	Corrisponde a	y = y - z + 1
a /= b	Corrisponde a	a = a / b
x += y/8	Corrisponde a	x = x + y/8
y %= 3	Corrisponde a	y = y % 3

Definendo una regola generale, si può dire che le operazioni in cui la variabile risultato compare anche come operando, si possono "contrarre" anteponendo l'operatore al simbolo di uguaglianza.

Le forme compatte rendono la scrittura del codice più elegante, anche se non sempre di facile lettura. Per questo motivo nel corso ***sconsigliamo*** l'uso delle forme compatte.

5. Elenco operatori in C

Ricordiamo che esiste la seguente gerarchia in C

ISTRUZIONE

Espressione ;

Espressioni composte

A titolo di riassunto e completamento viene ora tabulata una selezione dei principali operatori che il C mette a disposizione

Gli operatori si dividono in 4 categorie:
- operatori di assegnamento;
- operatori matematici;
- operatori relazionali;
- operatori logici.

5.1 OPERATORI DI ASSEGNAMENTO

E' il carattere = . Scrivere x=y; non vuol dire realizzare un confronto, ma assegnare il valore di y alla variabile x.

5.2 OPERATORI MATEMATICI

Consentono di effettuare le operazioni aritmetiche.

Operatore	**Tipo**	**Azione**	**Esempi**
++	Unario	Incrementa l'operando di un'unità	x++; ++x;
--	Unario	Decrementa l'operando di un'unità	y--; --y;

+	Binario	Somma di due operandi	X+y;
-	Binario	Sottrazione di due operandi	y-x;
*****	Binario	Moltiplicazione di due operandi	X*y;
/	Binario	Divisione	y/x ;
%	Binario	Resto della divisione	Y%x;

La precedenza fra gli operatori è:
1) Incrementi e decrementi unari (++, --);
2) Moltiplicazioni, divisioni e resti (*, /, %);
3) Somme e sottrazioni (+ , -);

Esempio: x=5 + 4 * 3 assegna 17 alla variabile x (5+12);

Nel caso di operatori di ugual livello, il compilatore procede da sinistra verso destra: ad esempio x=5*6/3 assegna 10 a x.

Per alterare l'ordine di esecuzione, è necessario introdurre le parentesi.

Si consiglia caldamente di usare sempre le parentesi per esplicitare comunque le precedenze.

5.3 OPERATORI RELAZIONALI

Sono utilizzati per confrontare espressioni

Operatore	**Descrizione**	**Esempi**
==	Confronto di uguaglianza	X == Y
>	Maggiore di	X > Y
<	Minore di	X < Y
>=	Maggiore o uguale di	X >= Y
<=	Minore o uguale di	Y <= X
!=	Diverso (x è diverso da y?)	X != Y

Le espressioni relazionali possono essere false (0) o vere (1). Anche se il loro uso è tipico delle istruzioni condizionali come if e while, è possibile scrivere ad esempio: x= (4==4) + (2>3) che è equivalente a: x = 1 + 0.

5.4 OPERATORI LOGICI

Sono utilizzati per manipolare i valori booleani, vero o falso. Ecco i principali

Operatore	**Descrizione**	**Esempi**
!	Not logico	!X
&&	And logico	X && Y

||	Or logico	X || Y

! Operatore logico di negazione. (unario)

`!espressione` ritorna 0 se espressione ritorna un valore diverso da 0; ritorna 1 se espressione ritorna il valore uguale a 0.

&& Operatore logico and.

Esegue l'and logico fra 2 espressioni.

Valuta la prima espressione e se questa e' falsa, ritorna 0 senza valutare la seconda; altrimenti valuta la seconda espressione. Se quest'ultima e' falsa ritorna 0 altrimenti ritorna 1.

|| Operatore logico or.

Esegue l'or logico fra 2 espressioni.

Valuta la prima espressione e se questa e' vera, ritorna 1 senza valutare la seconda; altrimenti valuta la seconda espressione. Se quest'ultima e' vera ritorna 1 altrimenti ritorna 0.

5.5 OPERATORI BIT A BIT

Seguono ora degli altri operatori che lavorano sui singoli bit delle celle di memoria. Potranno essere utili nel proseguo del corso. Sono solo da leggere.

~ Operatore complemento ad 1 (unario).

`~espressione` ritorna il complemento ad 1 del valore tornato da espressione.

Il complemento ad 1 consiste nel cambio di ciascun bit con il suo complemento, ovvero ogni bit posto ad 1 viene cambiato a 0 ed ogni bit posto a 0 viene cambiato ad 1.

& Operatore and bit a bit.

Ritorna il valore dell'and effettuato bit a bit sui valori ritornati dalle 2 espressioni.

Esempio:

```
int a=10;          /* rappresentazione binaria (8 bit): 00001010 */
int b=12;          /* rappresentazione binaria (8 bit): 00001100 */
int c;

c = a&b;        /*  00001010   a
                 *  00001100   b
                 *  --------
                 *  00001000   c = a&b */
```

N.B. - Per semplicità di trattazione, sono stati considerati gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori.

| Operatore or (inclusivo) bit a bit.

Ritorna il valore dell'or effettuato bit a bit sui valori ritornati dalle 2 espressioni.

Esempio:

```
int a=10;          /* rappresentazione binaria (8 bit): 00001010 */
```

```
int b=12;          /* rappresentazione binaria (8 bit): 00001100 */
int c;

c = a|b;        /*  00001010   a
                 *  00001100   b
                 *  --------
                 *  00001110   c = a|b */
```

Per semplicità di trattazione, sono stati considerati gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori.

^ Operatore or esclusivo (ex-or) bit a bit.

Ritorna il valore dell'ex-or effettuato bit a bit sui valori ritornati dalle 2 espressioni.

Esempio:

```
int a=10;          /* rappresentazione binaria (8 bit): 00001010 */
int b=12;          /* rappresentazione binaria (8 bit): 00001100 */
int c;

c = a^b;        /*  00001010   a
                 *  00001100   b
                 *  --------
                 *  00000110   c = a^b */
```

Per semplicità di trattazione, sono stati considerati gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori.

<< Operatore shift a sinistra.

espr_1 << espr_2

Ritorna il valore della traslazione a sinistra di espr_2 bit sui bit del valore ritornato da espr_1.

I nuovi bit che entrano a destra sono posti a 0.

Esempio:

```
int a=10;          /* rappresentazione binaria (8 bit): 00001010 */
int b=2;
int c;

c = a<<b;           /*  c = 00101000  = 40 */
```

Per semplicità di trattazione, sono stati considerati gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori.

>> Operatore shift a destra.

espr_1 >> espr_2

Ritorna il valore della traslazione a destra di espr_2 bit sui bit del valore ritornato da espr_1.

I nuovi bit che entrano a sinistra possono dipendere dall'architettura dell'elaboratore e/o dalla implementazione del compilatore. Non e' garantito che siano sempre posti a 0 anche per valori negativi di espr_1; per evitare ciò e' bene assicurarsi che il valore ritornato da espr_1 sia di tipo unsigned

Esempio:

```
unsigned int a=10;    /* rappresentazione binaria (8 bit): 00001010 */
int b=2;
int c;

c = a>>b;           /*  c = 00000010  = 2 */
```

Per semplicità di trattazione, sono stati considerati gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori.