Strutture di controllo in C

Prof. Orazio Mirabella

Sommario

- Il teorema di Bohm-Jacopini
- La struttura Alternativa
  - IF-ELSE
  - ELSE-IF
  - SWITCH
- La struttura Ripetitiva
  - WHILE
  - FOR
  - DO-WHILE
- Istruzioni BREAK e CONTINUE

La Programmazione Strutturata

- Esempio: dobbiamo effettuare una chiamata telefonica. Qual è l'algoritmo appropriato?

- Possiamo fare riferimento a tre tipi di istruzioni:

  ➔Istruzioni organizzate in sequenza. Ad es:
  - ✓Leggi il numero telefonico;
  - ✓Alza la cornetta;
  - ✓Componi il numero.

La Programmazione Strutturata

➔Istruzioni eseguite in alternativa con altre. Ad es:
- ✓Se è libero
  attendi che qualcuno risponda
- ✓Se è occupato
  riaggancia

➔Istruzioni che vengono ripetute più di una volta. Ad es:
- ✓Finchè è occupato
  riprova a chiamare

Il Teorema di Bohm-Jacopini

- Un qualsiasi algoritmo può essere espresso usando esclusivamente le strutture di sequenza, di selezione e di iterazione.

SEQUENZA

Istruzione 1

Istruzione 2

Istruzione 3

…

ALTERNATIVA

SE condizione

ALLORA

istruzione1

ALTRIMENTI

istruzione2

RIPETITIVA

RIPETI

istruzioni

FINCHE'

condizione

Istruzione IF

- Quando si desidera eseguire un'istruzione al verificarsi di una certa condizione, si utilizza l'istruzione if

  if(*espressione*) *istruzione*

- Per visualizzare minore di 100 se i è minore di 100:

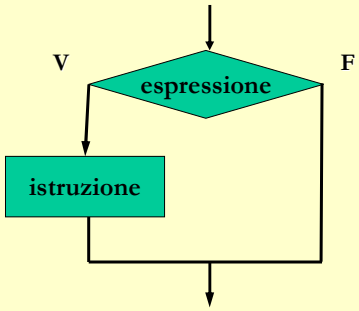
  ```
  if(i<100) printf("minore di 100");
  ```

- se ***espressione*** è vera si esegue ***istruzione***; ***espressione*** restituisce ***vero*** o ***falso***; in C *uno* e *zero*, o più esattamente ***diverso da zero*** e ***zero***.

IF

- L'if semplice permette codifiche compatte quando non è necessaria una ramificazione delle istruzioni.

```
Es1:    if ( k = 0)
        printf("k vale zero\n");

Es2:    if  ( y != 0 ) {
        z = x/y;
        printf ("z vale %f\n", z);
}
```



All'interno di un *if* si può anche eseguire un blocco di istruzioni, mettendole tra parentesi graffe

Esempio uso if

```
/* Utilizzo if */
#include <stdio.h>
main()
{
int i;
printf("Dammi un intero: ");
scanf("%d", &i);
if(i<100)
printf("minore di 100\n");
}
```

IF

- Se a è una variabile intera, l'esempio precedente è funzionalmente identico a:

  a = i<100;
- if(a!=0) printf("minore di 100");
- != significa *diverso da*,
- **if(a!=0)** significa *a è diverso da zero?* o anche:
- *a è vero?*
- che è un controllo eseguito per default, che equivale a: **if(a)**

Esempio uso if

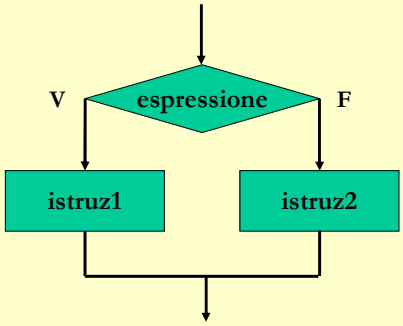
```
/* Utilizzo if */

#include <stdio.h>
main()

{
int i, a;
printf("Dammi un intero: ");
scanf("%d", &i);
A=(i<100);

if(a!=0)
printf("minore di 100\n");

}
```

```
/* Utilizzo if */

#include <stdio.h>
main()

{
int i, a;
printf("Dammi un intero: ");
scanf("%d", &i);
A=(i<100);

if(a)
printf("minore di 100\n");

}
```

IF-ELSE

- L'istruzione condizionale ***if-else*** viene usata per esprimere una decisione
- La sintassi generale è:

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



- If valuta l'espressione e se risulta vera viene eseguita l'istruzione1 altrimenti verrà eseguita l'istruzione2

Esempio uso IF-ELSE

```
#include<stdio.h>
main()
{
int a, i;
double x;
a = 6;
i = 5;

if (i!=0){
        x = a/i;
        printf ( "x vale: %f ", x);
        }
else
        printf ( "La divisione è impossibile!" );
}
```

Istruzioni IF annidate

- In assenza di *else* all'interno di una sequenza di *if* annidati, ogni *else* viene associato all'*if* più vicino

```
if  ( n > 0 )
        if  ( a > b )
                z = a;
else
        z = b;
```

equivale a:

```
if  ( n > 0 ){
        if  ( a > b )
                z = a;
        else
                z = b;
}
```

L'indentazione non è discriminante per il compilatore. Ogni ambiguità va risolta con l'uso delle parentesi.

Istruzioni IF annidate

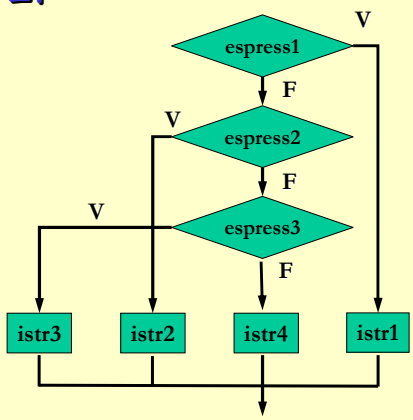
- Per fare in modo che il costrutto precedente sia interpretato correttamente dal compilatore occorre usare le parentesi graffe:

```
if  ( n > 0 ){
        if  ( a > b )
                z = a;
}
else
        z = b;
```

Simili ambiguità possono essere molto pericolose!

ELSE IF

```
if (espressione1)
        istruzione1;
else if (espressione2)
        istruzione2;
else if (espressione3)
        istruzione3;
else istruzione4;
```



- **È il modo più generale per realizzare una scelta fra più opzioni;**
- **Le espressioni vengono analizzate nell'ordine in cui si presentano. Se una di esse è vera viene eseguita l'istruzione corrispondente e l'intera catena termina.**

ELSE IF

- Il codice corrispondente ad ogni istruzione può essere anche un blocco di istruzioni fra parentesi graffe;
- L'ultimo *else* è il caso di default: viene eseguito quando nessuna delle precedenti espressioni è vera e a volte può essere omesso;
- Simili costrutti possono risultare complicati se nella scelta si deve valutare un numero molto grande di espressioni.

IF- ELSE IF- ELSE

- Uno statement IF segue la seguente struttura:

```
if (condition)
 { // eseguito solo se la “condizione”y if  è vera
}
else if (altra condizione)
{ // eseguita solo se "condizione" era false e “altra condizione " è vera
 }
else
 { // eseguita solo se sia”condizione” che “altra condizione” sono false
}
```

- La porzione **if** è l’unico blocco di istruzioni che è obbligatorio.
- **else if** ci permette di dire "ok, IF la condizione precedente non era vera, allora se questa condizione è vera...".
- L’**else** ci dice “se nessuna delle condizioni precedenti era vera...“
- Possiamo avere diversi blocchi else if, ma solo un blocco if e solo uno (o nessuno) blocco else.

Costrutto switch-case

Le decisioni a più vie possono essere risolte utilizzando più if-else in cascata oppure col costrutto switch-case, che consente di implementare decisioni multiple basandosi sul confronto fra il risultato di un’espressione (int o char) e un insieme di valori costanti.

```
if(espressione1)
  istruzione1
else
    if(espressione2)
      istruzione2
    else
       if(espressione3)
         istruzione3
...
                else
                  istruzioneN
```

```
switch(espressione) {
   case costante1:
     istruzione
     ...
   case costante2:
     istruzione
     ...
   case costante3:
     istruzione
     ...
  [default:
     istruzione
     ... ]
}
```

SWITCH

- Ogni caso possibile è dato da un'espressione o da un valore intero costanti;
- Le espressioni e le costanti nei diversi casi devono essere tra loro differenti;
- Il caso *default*, opzionale, viene eseguito solo se nessuno dei precedenti si è verificato;
- L'istruzione break, quando è aggiunta in fondo ad ogni blocco di istruzioni, provoca l'uscita immediata dallo ***switch*** *e fa eseguire solamente le istruzioni associate a un singolo case.*

Esempio utilizzo CASE

```
/* Esempio utilizzo case */
#include <stdio.h>
int x;
main()
{
printf("Digita una cifra: ");
scanf("%d", &x);
switch(x) {
   case 0:
     printf("zero\n");
     break;
   case 1:
     printf("uno\n");
     break;
  case 2:
    printf("due\n");
    break;
   case 3:
     printf("tre\n");
     break;
   case 4:
     printf("quattro\n");
     break;
   case 5:
     printf("cinque\n");
     break;
   default:
     printf("non compreso\n");
     break;
}
}
```

Costrutti Iterativi (cicli)

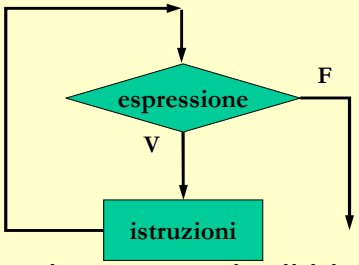
- L'iterazione serve a fare in modo che un'istruzione o un blocco di istruzioni vengano eseguite ripetutamente per un "certo numero di volte";
- Tale numero può essere:
  - ➔ noto a priori, essendo uguale al valore di una variabile o di una costante che non cambia durante l'iterazione;
  - ➔ determinato da condizioni dipendenti dall'iterazione;
- In C esistono tre costrutti iterativi:

WHILE
FOR
DO … WHILE

Ciclo WHILE

- La sintassi:

```
while (espressione)
{
   blocco di istruzioni;
}
```



- Se l'espressione è vera, viene eseguito il blocco di istruzioni al termine del quale l'espressione viene rivalutata;
- Non appena l'espressione diventa falsa, l'iterazione si interrompe e viene eseguita la prima istruzione successiva al ciclo *while*.

Ciclo WHILE

- Es:

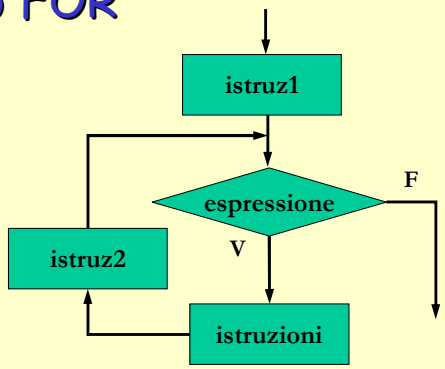
```
i=0;
while (i<100){
      printf("il messaggio verrà ripetuto\n");
      i=i+1;
      }
```

- Verifica la condizione **i**<100 e, trovandola vera per **i**=0, esegue il ciclo *while*, stampa il messaggio, incrementa di 1 la variabile **i** e ricontrolla la condizione **i**<100. Quando **i** raggiunge il valore 100 l'iterazione si interrompe.

Ciclo FOR

- La sintassi:

```
for (istruz1 ; espress ;
istruz2)
{
   blocco di istruzioni;
}
```



- istruz1 rappresenta una o più istruzioni di inizializzazione;
- espressione è la condizione da verificare prima di ogni iterazione;
- istruz2 rappresenta una o più istruzioni di aggiornamento, da eseguire ad ogni ciclo, dopo che è stato ultimato il blocco istruzioni.

Ciclo FOR

- Es:
  - ➔Col ciclo *while*:

```
i=0;
while (i<100){
      printf("il messaggio verrà ripetuto\n");
      i=i+1;
      }
```

  - ➔Col ciclo *for*:

```
for (i=0; i<100; i++){
      printf("il messaggio verrà ripetuto\n");
      }
```

L'operatore "virgola"

- Usando l'operatore , (virgola) il costrutto *for* consente di avere più assegnazioni in un unico statement per le istruzioni di inizializzazione e di aggiornamento:

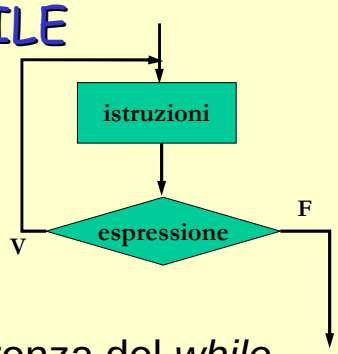
```
for (i=0, j=5; i<j; i++, j--){
      printf("il messaggio verra' ripetuto\n");
      }
```

- **i** è inizializzato a 0 e **j** a 5; all'inizio viene valutata la condizione **i<j** che risulta vera; quindi viene stampato il messaggio, **i** viene incrementato di 1 e **j** decrementato di 1. L'espressione viene testata nuovamente. Non appena diventa falsa si esce dal ciclo.

Ciclo DO…WHILE

- La sintassi:

```
do {
    blocco di istruzioni;
} while (espressione);
```



- Il blocco di istruzioni, a differenza del *while*, viene sempre eseguito almeno una volta, in quanto l'espressione viene valutata alla fine. Si tratta di un costrutto iterativo post-condizionale

Ciclo DO…WHILE

- Es:

```
int i=0;
do {
      printf("il messaggio verra' ripetuto\n");
      i++;
      } while (i<100);
```

- Finché è vera l'espressione (**i**<100), viene stampato il messaggio sull'output e l'iterazione continua.

Salti condizionati e incondizionati

- L'istruzione break consente di interrompere l'esecuzione di un case, provocando un salto del flusso di esecuzione alla prima istruzione successiva.
- Una seconda possibilità di uso di break è quella di forzare la terminazione di un'iterazione for, while o do-while, provocando un salto alla prima istruzione successiva al ciclo.
- L'istruzione *break* provoca l'uscita incondizionata da un ciclo senza eseguire le istruzioni successive né valutare la condizione di controllo;

```
for(i=1; numero!=0; i++) {
printf("Inser. intero positivo: \t");
scanf("%d", &numero);
if(numero>max) max=numero;
somma+=numero;
if(i==10) break;
```

Salti condizionati e incondizionati

- L'istruzione continue invece, fa in modo che le istruzioni che la seguono vengano ignorate ma non impedisce l'incremento della variabile di controllo o il controllo della condizione di test del ciclo. In altri termini, se la variabile di controllo soddisfa ancora la condizione di test, si continuerà con l'esecuzione del ciclo.

```
for(i=1; numero!=0 && i<=10; i++) {
printf("Inser. intero positivo: \t");
scanf("%d", &numero);
if(numero<0) continue;
if(numero>max) max=numero;
somma+=numero;
}
```

Esempio "continue"

```
#include <stdio.h>
main()
{
int i; int j = 10;
for( i = 0; i <= j; i ++ )
   {
   if( i == 5 )
        { continue;
        }
 printf("Hello %d\n", i );
   }
}
```

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 6
Hello 7
Hello 8
Hello 9
Hello 10
```

Continue fa saltare una iterazione del loop

Salti condizionati e incondizionati

- La funzione exit, che fa parte della libreria standard stdlib.h provoca l'immediata terminazione del programma e il ritorno al sistema operativo.
- Normalmente la funzione viene chiamata non passandole nessun argomento, il che significa *terminazione normale*.
- Altri argomenti consentono di indicare che si è verificato un particolare tipo di errore e il destinatario di questa comunicazione dovrebbe essere un processo di livello superiore in grado di gestire la condizione anomala.
- Es: in caso di immissione da parte dell'utente del valore zero, il programma sarebbe terminato:

```
if(numero<0) exit();
```