Istruzioni e strutture di controllo

Nicola Bicocchi

DIEF - UNIMORE

Strutture di controllo

- Il paradigma di programmazione strutturata consente di dirigere il flusso delle istruzioni a tempo di esecuzione, in base allo stato dinamico del programma.
- Il programmatore può predisporre molteplici blocchi di istruzioni da eseguire alternativamente o ripetutamente in base al soddisfacimento di determinate condizioni.
- A parte alcune minime differenze sintattiche, queste istruzioni sono simili a quelle che si possono trovare negli altri linguaggi.
- E' possibile realizzare due forme principali di controllo:
  - Esecuzioni condizionali
  - Esecuzioni iterative

Strutture di controllo

- Costrutti condizionali
  - if
  - switch-case
- Costrutti Iterativi
  - while
  - do-while
  - for
- Istruzioni aggiuntive
  - break
  - continue

Costrutto if

```
if (expression)
    statement
```

```
  if (expression)
    statement1
  else
    statement2
```

```
  if (expression1)
    statement1
  else if (expression2)
    statement2
  else
    statement3
```

Costrutto if

- Principali operatori di verifica delle condizioni: < , > , == , != , >= , <=
- Le condizioni valutate dal costrutto *if* sono valori interi interpretati come “booleani” (non esiste il tipo boolean in C)

```c
include <stdio.h>

int main() {
    int a = 5;
    if (a < 0) {
        printf("a < 0");
    } else {
        printf("a >= 0");
    }
}
```

Costrutto if (condizioni multiple)

```c
include <stdio.h>
int main() {
    int a = 5;
    if (a < 0) {
        printf("a < 0");
    } else if (a < 10) {
        printf("0 <= a < 10");
    } else {
        printf("a >= 10");
    }
}
```

Costrutto if (condizioni annidate)

- Le istruzioni condizionali if possono essere annidate per creare strutture di controllo più raffinate. Un esempio è mostrato nel listato seguente, dove la valutazione ed esecuzione delle istruzioni contenute all'interno del blocco *if* annidato è condizionato dalla valutazione della clausola del blocco *if* più esterno.

```
1 include <stdio.h>
2 int main() {
3     int a = 5, b = -2;
4     if (a > 0) {
5         if (b > 0) {
6             printf("a>0, b>0\n");
7         }
8     }
9 }
```

Operatore ?

L' operatore *?* (ternary condition) e' la forma piu' efficente per esprimere semplici costrutti if.

```
espressione1 ? espressione2 : espressione3
```

che equivale a:

```
if espressione1 then espressione2 else espressione3
```

Ad esempio:

```
z=(a>b) ? a : b
```

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

Costrutto switch-case

- Lo switch case consente di implementare decisioni multiple e si basa sul confronto tra il risultato di un'espressione e un insieme di valori costanti. L'espressione deve essere di tipo *int* o *char*.
- Il costrutto è composto da una espressione di controllo e da diversi blocchi di istruzione *case*, ognuno dei quali è associato a un particolare valore dell'espressione di controllo iniziale. Ogni blocco di istruzioni termine con l'istruzione *break*
- L'ultimo blocco di istruzioni è detto *default* e' facoltativo e raggruppa tutti gli altri casi.

```
1 switch ( espressione ) {
2     case costante1: istruzioni1; break;
3     case costante2: istruzioni2; break;
4     case costante3: istruzioni3; break;
5     ....
6     default: istruzioni; break;
7 }
```

Costrutto switch-case

```c
int a = 7;
switch(a) {
    case 0:printf("Eseguo il case 0\n");
        break;
    case 3:printf("Eseguo il case 3\n");
        break;
    case 7: printf("Eseguo il case 7\n");
        break;
    default: printf("Caso default\n");
        break;
}
```

Costrutto switch-case (senza break)

```c
int a = 7;
switch(a) {
    case 0:printf("Eseguo il case 0\n");
    case 3:printf("Eseguo il case 3\n");
    case 7: printf("Eseguo il case 7\n");
        break;
    default: printf("Caso default\n");
        break;
}
```

- Se a è uguale a 0, eseguo case 0, case 3 e case 7
- Se a è uguale a 3, eseguo case 3 e case 7
- Se a è uguale a 7, eseguo case 7

Costrutto while

- Il costrutto *while* è un costrutto condizionale dove un blocco di istruzioni viene eseguito fino a quando (while) l'espressione di controllo risulta vera
- *Ogni esecuzione del blocco di istruzioni è detta ciclo o iterazione*. In ogni ciclo sono eseguite le stesse istruzioni del blocco

```
while ( espressione di controllo ) {
    // blocco di istruzioni
}
```

```
int i = 0;
while(i < 10) {
    printf("Il valore di i é %d\n", i);
    i++;
}
```

Costrutto do-while

- Il costrutto *do-while* è un costrutto post-condizionale dove prima sono eseguite le istruzioni che formano il blocco dell'iterazione e successivamente è verificata la la condizione. Se la condizione è vera allora si ripete il ciclo, altrimenti si passa all'istruzione successiva
- Il costrutto *do-while* può servire in tutti i casi in cui la prima iterazione deve essere eseguita a prescindere dal successo dell'istruzione di controllo. *Casi d'uso limitati*

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

```
int i = 0;
do {
    printf("Il valore di i é %d\n", i);
    i++;
} while(i < 10);
```

Costrutto for

- Il costrutto *for* viene utilizzato per codificare la ripetizione di istruzioni per un numero prestabilito di volte
- Le variabili che controllano il flusso del for sono completamente ed esclusivamente gestite all'interno della direttiva for: **assegnazione, controllo, modifica post-ciclo**

```
for ([<espressione assegnazione>]; [<espressione controllo>];[<espressione
    post-ciclo>]) {
    // blocco di istruzioni
}
```

```
int i;
for(i = 0; i < 10; i++) {
    printf("Il valore di i é %d\n", i);
}
```

Costrutto for (assegnazione, controllo, modifica post-ciclo)

```c
int i;
for(i = 0; i < 10; i++) {
    printf("Il valore di i é %d\n", i);
}
```

- **i = 0**: comando di assegnazione eseguito come prima operazione
- **i < 10**: condizione controllata prima di ogni ciclo (anche prima del primo ciclo, come per while)
- **i++**: comando/azione eseguito ad ogni ciclo dopo il controllo della condizione

Istruzione break

- L'istruzione *break* interrompe le iterazioni di costrutto iterativo (*for, while, do-while*)
- In genere è associata a una struttura condizionale *if* per associarla al verificarsi di un evento
- L'istruzione *break* è utile per evitare cicli infiniti, spostando l'espressione di controllo all'interno del ciclo

```
1  int i=0;
2  for(i=0; i<5; i++){
3      if (i==2) {
4          break;
5      }
6      printf("%d ", i);
7  }
8
9  //Output: 0 1
```

Istruzione continue

- L'istruzione *continue* interrompe l'iterazione di un ciclo per saltare a quella successiva
- Quando il compilatore incontra l'istruzione *continue*, interrompe l'iterazione corrente tornando all'espressione di controllo per iniziare l'interazione successiva, senza uscire dal ciclo

```c
int i=0;
for(i=0; i<5; i++){
    if (i==2) {
        continue;
    }
    printf("%d ", i);
}

//Output: 0 1 3 4
```

Cicli infiniti

- Sia *for* che *while* possono essere utilizzati anche per creare cicli infiniti
- Spesso i cicli infiniti sono interrotti attraverso un'istruzione *break* che verifica una condizione all'interno del ciclo

```c
for (;;) {
  // istruzioni
}
```

```c
#define TRUE 1

while (TRUE) {
  // istruzioni
}
```

Errori comuni (assegnamento e verifica di uguaglianza)

- Confondere gli operatori di assegnamento (=) e di verifica di uguaglianza (==) può produrre innumerevoli errori
- Considerata la facilità dell'errore (typo), fino a Python 3.8 l'assegnamento all'interno di una condizione *if* non era supportato

```
1 int main(){
2     int a = 10;
3     if (a == 0) { }
4 }
```

```
1 int main(){
2     int a = 10;
3     if (a = 0) { }
4 }
```

Errori comuni (overflow e underflow)

```c
// esempio di overflow char[-128, 127]
// dovrebbe terminare non NON termina!
int main() {
  char a;
  for(a=0; a<200; a++) { }
  return 0;
}
```

```c
// esempio di underflow
// unsigned int[0, 4.294.967.295], int[-2.147.483.648, 2.147.483.647]
// non dovrebbe terminare MA termina!
int main() {
    int a;
    for(a=9; a>=0; a++) { }
    return 0;
}
```

Istruzione goto

- *goto* è un costrutto di salto incondizionato che rappresenta l'istruzione *jump* presente nei linguaggi di basso livello
- A volte si utilizza nella gestione degli errori, ma in generale è sconsigliato utilizzarlo se non si ha piena coscienza di ciò che si sta facendo
- *Edsger W. Dijkstra - Go To statement considered harmful*

```c
int main(){
    goto end;
    printf("Non eseguito\n");

end:
    printf("Fine\n");
return 0;
}
```