

Istruzioni e strutture di controllo

Corso di Programmazione di Sistema

Dr. Nicola Bicocchi

DIEF/UNIMORE

Marzo 2022

- 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

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

Costrutto if

```
1  if (espressione)
2      istruzione1
```

```
1  if (espressione)
2      istruzione1
3  else
4      istruzione2
```

```
1  if (espressione1)
2      istruzione1
3  else if (espressione2)
4      istruzione2
5  else
6      istruzione3
```

Costrutto if

- Principali operatori di verifica delle condizioni: $<$, $>$, $==$, $!=$, $>=$, $<=$
- Le condizioni valutate dal costrutto *if* sono valori *interi* interpretati come *booleani* (convenzionalmente, $0 == \text{falso}$, $1 == \text{vero}$)
- Il tipo *bool* è stato introdotto a partire dallo standard C99

```
1  include <stdio.h>
2
3  int main() {
4      int a = 5;
5      if (a < 0) {
6          printf("a < 0\n");
7      } else {
8          printf("a >= 0\n");
9      }
10 }
```

Costrutto if (condizioni multiple)

```
1  include <stdio.h>
2  int main() {
3      int a = 5;
4      if (a < 0) {
5          printf("a < 0");
6      } else if (a < 10) {
7          printf("0 <= a < 10");
8      } else if (a < 20) {
9          printf("10 <= a < 20");
10     } else {
11         printf("a >= 20");
12     }
13 }
```

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.
- La valutazione ed esecuzione delle istruzioni contenute all'interno del blocco *if* annidato ($b > 0$) è condizionato dalla valutazione della clausola del blocco *if* più esterno ($a > 0$).

```
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' efficiente per esprimere semplici costrutti if.

```
1 espressione1 ? espressione2 : espressione3
```

che equivale a:

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

Ad esempio:

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

```
1 if (a>b)
2     z=a;
3 else
4     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

```
1  int a = 7;
2  switch(a) {
3      case 0: printf("Eseguo il case 0\n");
4          break;
5      case 3: printf("Eseguo il case 3\n");
6          break;
7      case 7: printf("Eseguo il case 7\n");
8          break;
9      default: printf("Caso default\n");
10         break;
11 }
```

Costrutto switch-case (senza break)

```
1  int a = 7;
2  switch (a) {
3      case 0: printf("Eseguo il case 0\n");
4      case 3: printf("Eseguo il case 3\n");
5      case 7: printf("Eseguo il case 7\n");
6          break;
7      default: printf("Caso default\n");
8          break;
9  }
```

- Eliminando le istruzioni *break* per separare i casi, cambia la semantica
- se $a == 0$, vengono eseguiti i casi 0, 3, 7
- se $a == 3$, vengono eseguiti i casi 3, 7
- se $a == 7$, viene eseguito il caso 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

```
1 while ( espressione di controllo ) {  
2     // blocco di istruzioni  
3 }
```

```
1 int i = 0;  
2 while(i < 10) {  
3     printf("Il valore di i é %d\n", i);  
4     i++;  
5 }
```

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 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 relativamente limitati*

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

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

Costrutto for

- Il costrutto *for* è utilizzato per la ripetizione di istruzioni per un numero prestabilito di volte
- Le variabili che controllano il flusso del ciclo *for* sono gestite all'interno della direttiva *for*
- (1) assegnazione, (2) controllo, (3) modifica post-ciclo

```
1  for ([<espressione assegnazione>]; [<espressione controllo>]; [<espressione  
    post-ciclo>]) {  
2      // blocco di istruzioni  
3  }
```

```
1  int i;  
2  for(i = 0; i < 10; i++) {  
3      printf("Il valore di i é %d\n", i);  
4  }
```

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

```
1  int i;  
2  for(i = 0; i < 10; i++) {  
3      printf("i=%d\n", i);  
4  }
```

- **i = 0**: assegnazione (eseguita una sola volta, come prima operazione)
- **i < 10**: condizione di controllo (verificata prima di ogni ciclo)
- **i++**: modifica post-ciclo (eseguita al termine di ogni ciclo)

```
1  int i;  
2  for(i = 0, j = 0; i < 10; i++, j++) {  
3      printf("i=%d, j=%d\n", i, j);  
4  }
```

Cicli infiniti

- I costrutti iterativi possono essere utilizzati anche per creare cicli infiniti
- Spesso i cicli infiniti sono interrotti attraverso un'istruzione dedicata (*break*) posizionata all'interno del blocco di istruzioni

```
1  for (;;) {  
2      /* istruzioni */  
3  }
```

```
1  #define TRUE 1  
2  while (TRUE) {  
3      /* istruzioni */  
4  }
```

```
1  do {  
2      /* istruzioni */  
3  } while (TRUE);
```


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* sposta l'espressione di controllo all'interno del ciclo

```
1  int i=0;
2
3  for(i=0; i<5; i++) {
4      if (i==2) {
5          break;
6      }
7      printf("%d ", i);
8  }
9
10 //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'iterazione successiva
- Il ciclo non viene interrotto

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

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. Ora è supportato attraverso un operatore dedicato (:=)

```
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)

```
1 // esempio di overflow char[-128, 127]
2 // dovrebbe terminare non NON termina!
3 int main() {
4     char a;
5     for(a=0; a<200; a++) { }
6     return 0;
7 }
```

```
1 // esempio di underflow
2 // unsigned int[0, 4.294.967.295], int[-2.147.483.648, 2.147.483.647]
3 // non dovrebbe terminare MA termina!
4 int main() {
5     int a;
6     for(a=9; a>=0; a++) { }
7     return 0;
8 }
```

Istruzione goto

- *goto* è un costrutto di *salto incondizionato* che rappresenta l'istruzione assembly *jump*
- 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*

```
1  int main(){
2      goto end;
3      printf("Non eseguito\n");
4
5  end:
6      printf("Fine\n");
7  return 0;
8  }
```