

# 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

```
1  if (expression)
2      statement
```

```
1  if (expression)
2      statement1
3  else
4      statement2
```

```
1  if (expression1)
2      statement1
3  else if (expression2)
4      statement2
5  else
6      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)

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

# 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' efficiente per esprimere semplici costrutti if.

```
1 expression1 ? expression2 : expression3
```

che equivale a:

```
1 if expression1 then expression2 else expression3
```

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  }
```

- 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

```
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 do-while può servire in tutti i casi in cui la prima iterazione deve essere eseguita a prescindere dal successo dell'istruzione di controllo. *Pochi casi d'uso*

```
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 un'istruzione che viene utilizzata per codificare la ripetizione di istruzioni che devono essere ripetute 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**

```
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("Il valore di i é %d\n", i);  
4  }
```

- **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'iterazione successiva, senza uscire dal ciclo

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

# Cicli infiniti

- *for* e *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

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

```
1  #define TRUE 1  
2  
3  while (TRUE) {  
4    // istruzioni  
5  }
```

# 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)

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

# 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*

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