Capitolo 4 – Controllo dei Programmi

Outline

Introduzione

- Questo capitolo introduce
  - Altre strutture di controllo iterative
    - `for`
    - `do…while`
  - Statement di selezione multipla `switch`
  - Statement `break`
    - Usato per uscire immediatamente da una certa struttura di controllo
  - Statement `continue`
    - Usato per saltare le rimanenti istruzioni di un blocco di una struttura iterativa e procedere con la successiva iterazione dle ciclo

Ripetizione Counter-Controlled

- Esempio:

```
int counter = 1;          // initialization
while ( counter <= 10 ) { // repetition condition
   printf( "%d\n", counter );
   ++counter;             // increment
}
```

  - Lo statement

```
int counter = 1;
```

    - Definisce una variabile `counter`
    - Definisce la variabile di tipo intero
    - Riserva uno spazio in memoria
    - Imposta il valore iniziale a `1`


```
1  /* Fig. 4.1: fig04_01.c
2     Counter-controlled repetition */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1;              /* initialization */
9
10    while ( counter <= 10 ) {     /* repetition condition */
11       printf ( "%d\n", counter ); /* display counter */
12       ++counter;                  /* increment */
13    } /* end while */
14
15    return 0; /* indicate program ended successfully */
16
17 } /* end function main */
```

Outline

fig04_01.c

Program Output

```
1
2
3
4
5
6
7
8
9
10
```

Ripetizione Counter-Controlled

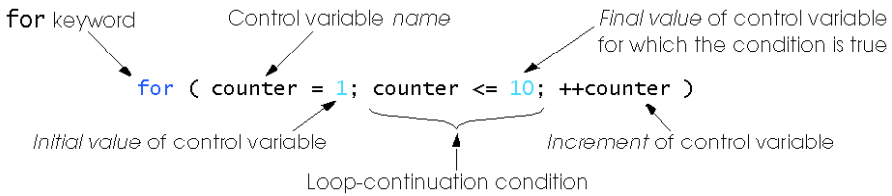
- Codice più compatto
  - Inizializza `counter` a `0`
    - `while ( ++counter <= 10 )`
      `printf( "%d\n", counter );`

Outline

fig04_02.c

```
1  /* Fig. 4.2: fig04_02.c
2     Counter-controlled repetition with the for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter; /* define counter */
9
10    /* initialization, repetition condition, and increment
11       are all included in the for statement header. */
12    for ( counter = 1; counter <= 10; counter++ ) {
13       printf( "%d\n", counter );
14    } /* end for */
15
16    return 0; /* indicate program ended successfully */
17
18 } /* end function main */
```


`for` Repetition Statement



`for` Repetition Statement

- Formato per cicli di tipo `for`

  `for` (*initialization*; *loopContinuationTest*; *increment*)
  statement

- Esempio:

  ```
  for(counter = 1; counter <= 10; counter++ )
     printf( "%d\n", counter );
  ```

  – Stampa gli interi da 1 a 10

No `;` dopo questa espressione

`for` Repetition Statement

- I cicli *for* possono essere riscritti semplicemente come cicli di tipo *while*:

```
initialization;
while ( loopContinuationTest ) {
  statement;
  increment;
}
```

- Inizializzazione ed incremento
  - Possono essere delle liste con valori separati da virgole
  - Esempio:

```
for (i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```


`for` Statement : Note e osservazioni

- Espressioni aritmetiche
  - Inizializzazione, loop-continuation, e incremento possono contenere espressioni aritmetiche.
    Se `x` uguale a `2` e `y` uguale a `10`

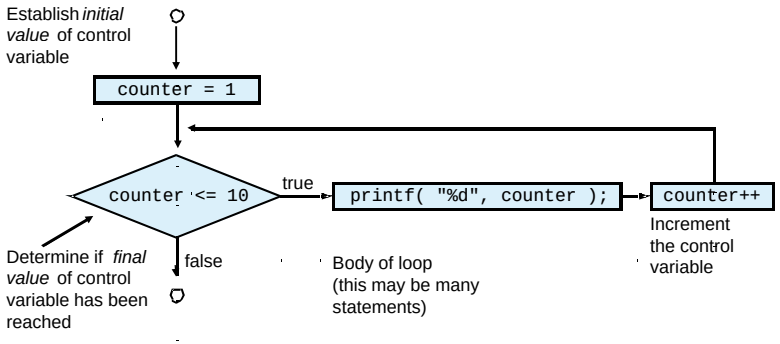
```
for ( j = x; j <= 4 * x * y; j += y / x )
```

    equivalente a

```
for ( j = 2; j <= 80; j += 5 )
```

- Note sullo statement `for` :
  - "Incremento" può essere negativo (decremento)
  - Se la condizione loop-continuation è inizialmente `false`
    - Il corpo dello statement `for` non viene eseguito
    - Il controllo procede con il successivo statement dopo lo statement `for`
  - Variabile di controllo
    - Spesso stampata o usata all'interno del corpo, ma non necessaria

`for` Statement : Note e osservazioni



Outline

fig04_05.c

```
1  /* Fig. 4.5: fig04_05.c
2     Summation with for */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int sum = 0; /* initialize sum */
9     int number;  /* number to be added to sum */
10
11    for ( number = 2; number <= 100; number += 2 ) {
12       sum += number; /* add number to sum */
13    } /* end for */
14
15    printf( "Sum is %d\n", sum ); /* output sum */
16
17    return 0; /* indicate program ended successfully */
18
19 } /* end function main */
```

Program Output

```
Sum is 2550
```

Outline

fig04_06.c (Part 1 of 2)

```
1  /* Fig. 4.6: fig04_06.c
2     Calculating compound interest */
3  #include <stdio.h>
4  #include <math.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     double amount;            /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;         /* interest rate */
12    int year;                  /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20       /* calculate new amount for specified year */
21       amount = principal * pow( 1.0 + rate, year );
22
23       /* output one table row */
24       printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
```


Outline

fig04_06.c (Part 2 of 2)

Program Output

```
27    return 0; /* indicate program ended successfully */
28
29 } /* end function main */
Year    Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```


`switch` Multiple-Selection Statement

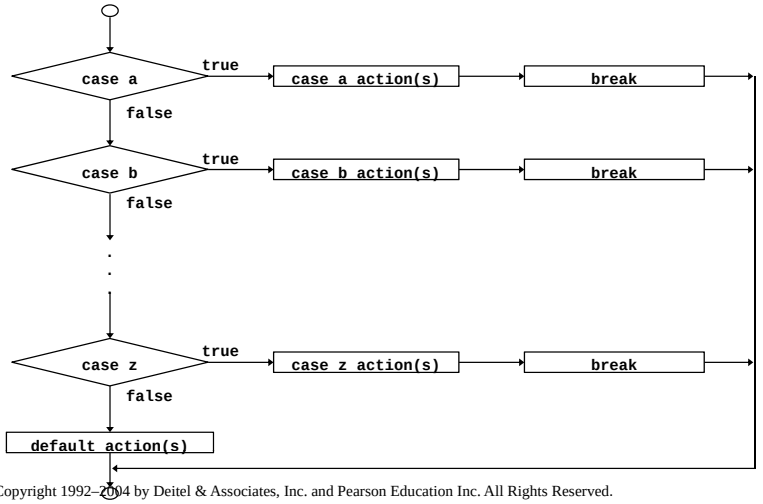
- `switch`
  - Utile quando una variable o un'espressione è testata per tutti i valori che può assumere e sono intraprese azioni differenti
- Formato
  - Serie di `case` una clausola opzionale di caso di `default`

```
switch ( value ){
   case '1':
      actions
   case '2':
      actions
   default:
      actions
   }
```

  - `break;` uscita dallo statement

`switch` Multiple-Selection Statement

- Flowchart dello statement `switch`



Outline

fig04_07.c (Part 1 of 3)

```
1  /* Fig. 4.7: fig04_07.c
2     Counting letter grades */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int grade;      /* one grade */
9     int aCount = 0; /* number of As */
10    int bCount = 0; /* number of Bs */
11    int cCount = 0; /* number of Cs */
12    int dCount = 0; /* number of Ds */
13    int fCount = 0; /* number of Fs */
14
15    printf(  "Enter the letter grades.\n"  );
16    printf(  "Enter the EOF character to end input.\n"  );
17
18    /* loop until user types end-of-file key sequence */
19    while ( ( grade = getchar() ) != EOF ) {
20
21       /* determine which grade was input */
22       switch ( grade ) { /* switch nested in while */
23
24          case 'A':    /* grade was uppercase A */
25          case 'a':    /* or lowercase a */
26             ++aCount; /* increment aCount */
27             break;    /* necessary to exit switch */
28
```


Outline

fig04_07.c (Part 2 of 3)

```
29          case 'B':    /* grade was uppercase B */
30          case 'b':    /* or lowercase b */
31             ++bCount; /* increment bCount */
32             break;    /* exit switch */
33
34          case 'C':    /* grade was uppercase C */
35          case 'c':    /* or lowercase c */
36             ++cCount; /* increment cCount */
37             break;    /* exit switch */
38
39          case 'D':    /* grade was uppercase D */
40          case 'd':    /* or lowercase d */
41             ++dCount; /* increment dCount */
42             break;    /* exit switch */
43
44          case 'F':    /* grade was uppercase F */
45          case 'f':    /* or lowercase f */
46             ++fCount; /* increment fCount */
47             break;    /* exit switch */
48
49          case '\n':   /* ignore newlines, */
50          case '\t':   /* tabs, */
51          case ' ':    /* and spaces in input */
52             break;    /* exit switch */
53
```


Outline

fig04_07.c (Part 3 of 3)

```
54          default:     /* catch all other characters */
55             printf( "Incorrect letter grade entered." );
56             printf( " Enter a new grade.\n" );
57             break;    /* optional; will exit switch anyway */
58       } /* end switch */
59
60    } /* end while */
61
62    /* output summary of results */
63    printf( "\nTotals for each letter grade are:\n" );
64    printf( "A: %d\n", aCount ); /* display number of A grades */
65    printf( "B: %d\n", bCount ); /* display number of B grades */
66    printf( "C: %d\n", cCount ); /* display number of C grades */
67    printf( "D: %d\n", dCount ); /* display number of D grades */
68    printf( "F: %d\n", fCount ); /* display number of F grades */
69
70    return 0; /* indicate program ended successfully */
71
72 } /* end function main */
```


Outline

Program Output

```
Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

do…while Repetition Statement

- do…while repetition statement
  - Simile a una struttura while
  - La condizione per la ripetizione è testata dopo che il corpo del ciclo è eseguito
    - Tutte le azioni sono eseguite almeno una volta
  - Formato:

```
do {
   statement;
} while ( condition );
```


do…while Repetition Statement

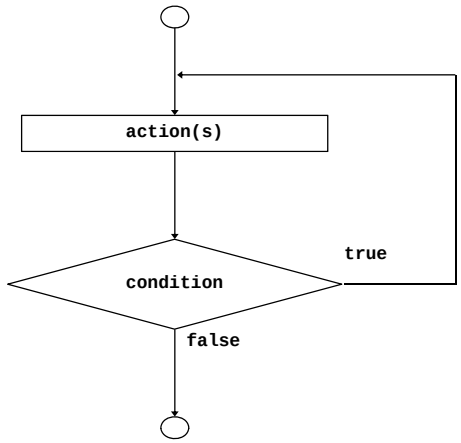
- Esempio (sia counter = 1):

```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

  - Stampa gli interi da 1 a 10

do…while Repetition Statement

- Flowchart dello statement do…while



Outline

fig04_09.c

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int counter = 1; /* initialize counter */
9
10    do {
11       printf( "%d  ", counter ); /* display counter */
12    } while ( ++counter <= 10 ); /* end do...while */
13
14    return 0; /* indicate program ended successfully */
15
16 } /* end function main */
```

Program Output

```
1  2  3  4  5  6  7  8  9  10
```

Statement break e continue

- break
  - Causa l'immediata uscita da uno statement while, for, do…while o switch
  - Il programma continua eseguendo il primo statement dopo la struttura
  - Usato comunemente per:
    - Terminare l'esecuzione di un ciclo
    - Saltare il resto dei controlli di uno statement switch

Outline

fig04_11.c

```
1  /* Fig. 4.11: fig04_11.c
2     Using the break statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, terminate loop */
14       if ( x == 5 ) {
15          break; /* break loop only if x is 5 */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

Program Output

```
1 2 3 4
Broke out of loop at x == 5
```


Statement break e continue

- continue
  - Salta le istruzioni rimanenti nel corpo di un while, for o do…while
    - Procede con la successiva iterazione del ciclo
  - while e do…while
    - Il test per la continuazione del ciclo è valutato immediatamente dopo l'esecuzione dello statement continue
  - for
    - L'espressione di incremento viene eseguita, dunque il test per la continuazione del ciclo è valutato

Outline

fig04_12.c

```
1  /* Fig. 4.12: fig04_12.c
2     Using the continue statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, continue with next iteration of loop */
14       if ( x == 5 ) {
15          continue; /* skip remaining code in loop body */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

Program Output

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Operatori logici

- && (AND logico)
  - Ritorna `true` se entrambe le condizioni sono `true`
- || (OR logico)
  - Ritorna `true` se una delle condizioni è `true`
- ! (NOT logico, negazione logica)
  - Inverte la verità o falsità di una condizione
  - Operatore unario
- Utili per le condizioni nei cicli

Espressione	Risultato
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

Operatori logici

expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Fig. 4.13 Tavola di verità per l'operatore && (AND).

expression1	expression2	expression1 || expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Fig. 4.14 Tavola di verità per l'operatore || (OR).

expression	! expression
0	1
nonzero	0

Fig. 4.15 Tavola di verità per l'operatore ! (negazione).

Operatori logici

Operatori						Associatività	Tipo
++	--	+	-	!	(type)	right to left	unary
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
&&						left to right	logical AND
||						left to right	logical OR
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment
,						left to right	comma

Fig. 4.16 Precedenza tra operatori e associatività.

Confondere operatori di uguaglianza (==) e assegnamento (=)

- Errore pericoloso
  - Non è causa di errori sintattici
  - Ogni espressione che produce un valore può essere usato in una struttura di controllo
  - Valori diversi da zero indicano `true`, mentre zero `false`
  - Esempio con ==:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```


Confondere operatori di uguaglianza (==) e assegnamento (=)

– Esempio, rimpiazzando == con =

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

  - `payCode` viene inizializzato a `4`
  - `4` è un valore diverso da zero, dunque l'espressione è `true`, e il bonus è ricevuto indipendentemente dal valore di `payCode`

– Errore logico, non di sintassi

Confondere operatori di uguaglianza (==) e assegnamento (=)

- lvalues
  – Espressioni che possono comparire alla sinistra di una equazione
  – I loro valori possono essere modificati
  – Possono esserci nomi di variabili
    - `x = 4;`
- rvalues
  – Espressioni che possono comparire solo nella parte destra di una equazione
  – Costanti, come numeri
    - Non possiamo scrivere `4 = x;`
    - Dobbiamo scrivere `x = 4;`
  – lvalues possono essere usati come rvalues, ma non viceversa
    - `y = x;`