Control Structures

Fundamentals of Programming

Lecture 2

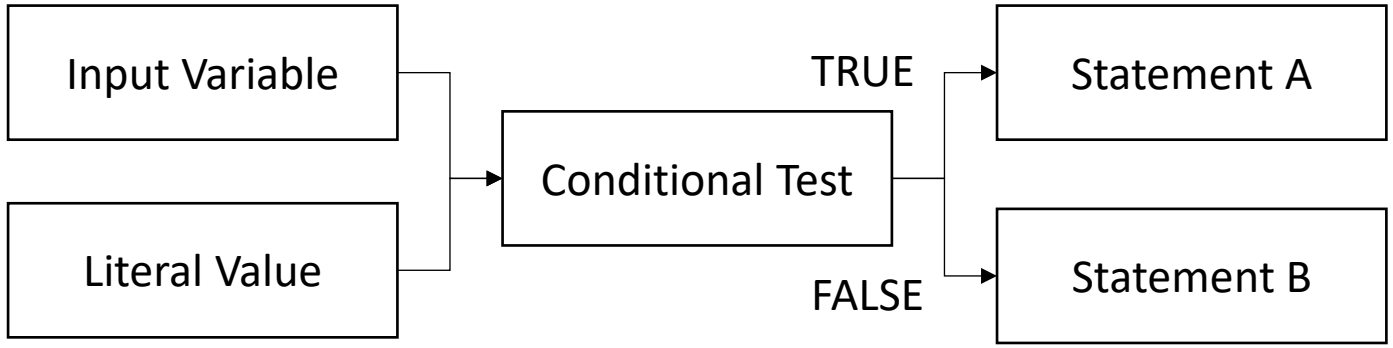
Contents

- Operators
  - Relational
  - Logical
- Control structure
  - IF, IF-ELSE, IF-ELSEIF
  - SWITCH
  - WHILE
  - DO-WHILE
  - FOR
- Nested control structure

Introduction

- Conditional test is used to decide on which statement to be executed.

- Conditional test is done using relational and/or logical operators



Compound statement

- Compound statement is a group of statements in a control structure
- Compound statement is enclosed within a pair of braces
- Also referred as a ‘body of code’ or ‘block of code’

```
{
   statement1
   statement2
   …
   statementN
}
```

Operators

Relational operators

- It is used to test a condition in a flow control statement
- Result is either TRUE (1) or FALSE (0)

Operator	**Description**
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational operators

Conditional test	Result
(7 > 2)	TRUE
(2 > 7)	FALSE
(7 < 2)	FALSE
(2 < 7)	TRUE
(7 >= 2)	TRUE
(7 >= 7)	TRUE
(7 <= 2)	FALSE
(2 <= 7)	TRUE

Conditional test	Result
7 == 2	FALSE
2 == 2	TRUE
7 != 2	TRUE
7 != 7	FALSE

Logical operators

- It is used to connect multiple conditional tests in a control flow statement
- Result is either TRUE (1) or FALSE (0)

Operator	**Description**
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical operators

Logical AND Truth table (A && B)

Inputs		Output
A	**B**	**Result**
0	0	0 (FALSE)
0	1	0 (FALSE)
1	0	0 (FALSE)
1	1	1 (TRUE)

Logical operators

Logical OR Truth table (A || B)

Inputs		Output
A	**B**	**Result**
0	0	0 (FALSE)
0	1	1 (TRUE)
1	0	1 (TRUE)
1	1	1 (TRUE)

Logical operators

Logical NOT Truth table (! A)

Input	Output
A	**Result**
0	1 (TRUE)
1	0 (FALSE)

Logical operators: examples

Conditional test	Result
(7>2) && (8>3)	
TRUE and TRUE	TRUE
(0>5) && (7>-2)	
FALSE and TRUE	FALSE
(7<5) && (1>=9)	
FALSE and FALSE	FALSE
!0 && !3	
TRUE and FALSE	FALSE

Conditional test	Result
(3>2) || (6>1)	
TRUE or TRUE	TRUE
(6>10) || (2>=2)	
FALSE or TRUE	TRUE
(7==2) || (-6>=1)	
FALSE or FALSE	FALSE
!0 || !5	
TRUE or FALSE	TRUE

Bitwise operators

- It is used to manipulate the bit value in data

Operator	Description
&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR (XOR)
<<	left shift
>>	right shift
~	one’s complement (unary)

Bitwise operators: XOR

Logical XOR Truth table (A ^ B)

Inputs		Output
A	**B**	**Result**
0	0	0 (FALSE)
0	1	1 (TRUE)
1	0	1 (TRUE)
1	1	0 (FALSE)

Bitwise example 1

- Variable x = 50, its binary data is 00110010
- Variable y = 23, its binary data is 00010111

Bit Address	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
x = 50	0	0	1	1	0	0	1	0
y = 23	0	0	0	1	0	1	1	1
x & y = 18	0	0	0	1	0	0	1	0

Bitwise example 2

- Variable x = 50, its binary data is 00110010
- Variable y = 23, its binary data is 00010111

Bit Address	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
x = 50	0	0	1	1	0	0	1	0
y = 23	0	0	0	1	0	1	1	1
x | y = 55	0	0	1	1	0	1	1	1

Bitwise example 3

- Variable x = 50, its binary data is 00110010
- Variable y = 23, its binary data is 00010111

Bit Address	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
x = 50	0	0	1	1	0	0	1	0
y = 23	0	0	0	1	0	1	1	1
x ^ y = 37	0	0	1	0	0	1	0	1

Bitwise example 4

- Variable x = 50, its binary data is 00110010

Bit Address	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
x = 50	0	0	1	1	0	0	1	0
~x = –51	1	1	0	0	1	1	0	1

Bit 7: 0 for positive, 1 for negative

Bitwise example 5

- Variable y = 23, its binary data is 00010111

Bit Address	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
y = 23	0	0	0	1	0	1	1	1
~y = –24	1	1	1	0	1	0	0	0

Bit 7: 0 for positive, 1 for negative

Bitwise test program

```
#include <stdio.h>
int x, y;
int main()
{
  x = 50;
  y = 23;
  printf("x&y = %d\n", x&y);
  printf("x|y = %d\n", x|y);
  printf("x^y = %d\n", x^y);
  printf("~x = %d\n", ~x);
  return 0;
}
```

Order of precedence (1)

No.	Operator	Associativity
1	`() [] -> .`	Left to right
2	`! ~ ++ -- + - * (type) sizeof`	Right to left
3	`* / %`	Left to right
4	`+ -`	Left to right
5	`<< >>`	Left to right
6	`< <= > =>`	Left to right
7	`== !=`	Left to right

Order of precedence (2)

No.	Operator	Associativity
8	&	Left to right
9	^	Left to right
10	|	Left to right
11	&&	Left to right
12	||	Left to right
13	? :	Right to left
14	= += -= *= /= %= &= ^= |= <<= >>=	Right to left
15	,	Left to right

Prefix and postfix notations

Increment/Decrement variables

- Variables are used to keep track of:
  - Values
  - Counters
  - Index in an arrays
  - Error codes
- Increment and decrement are frequently used in arithmetic operations
- Contracted notation (++) or (--) makes the code more readable
  - Used as prefix or postfix to any variable

Increment/Decrement syntax

- Increment of variable x

```
x = x + 1
```

- Alternative

```
x++
```

- Decrement of variable x

```
x = x - 1
```

- Alternative

```
x--
```

- Illegal operations

```
(x + y)++
(x + y)--
```

Prefix and postfix

Pre/Post	Operator	Description	Value of n	Before	After
Postfix	n++	Use current value of n, then increment n	5	5	6
Prefix	++n	Increment n, then use new value of n	5	6	6
Postfix	n--	Use current value of n, then decrement n	5	5	4
Prefix	--n	Decrement n, then use new value of n	5	4	4

Prefix/postfix test program

```
#include <stdio.h>
int n;
int main()
{
  n = 5;
  printf("before: n++ = %d\n", n++);
  printf("after: n++ = %d\n", n);

  n = 5;
  printf("before: ++n = %d\n", ++n);
  printf("after: ++n = %d\n", n);

  n = 5;
  printf("before: n-- = %d\n", n--);
  printf("after: n-- = %d\n", n);

  n = 5;
  printf("before: --n = %d\n", --n);
  printf("after: --n = %d\n", n);
  return 0;
}
```

Expressions

Expression vs Statement

- An expression: assigning a value to a variable or constant, etc.
- Expressions are used in the conditional test
- An expression does not have a semicolon at the end of the statement
- Expression: `x = 0`
- Statement: `x = 0;`

Assignment expression

- Variable is assigned a value using equal sign (=) followed by a value

```
x = 1
```

- To increment variable x by 2 every time the statement is executed

```
x = x + 2
```

- Simplified expression to perform similar action

```
x += 2
```

Assignment expression list

Operator	Description	Regular expression	Simplified expression
+=	Increment	n = n + 3	n += 3
-=	Decrement	n = n - 3	n -= 3
*=	Multiply	n = n * 3	n *= 3
/=	Divide	n = n / 3	n /= 3
%=	Modulus	n = n % 3	n %= 3
&=	Bitwise AND	n = n & 3	n &= 3
|=	Bitwise OR	n = n | 3	n |= 3
^=	Bitwise XOR	n = n ^ 3	n ^= 3
<<=	Left shift	n = n << 3	n <<= 3
>>=	Right shift	n = n >> 3	n >>= 3

Conditional expression

- Conditional expression is a compact form of the IF statement
- Expressions are separated by the ternary operator (?:)

```
expression1 ? expression2 : expression3
```

- Expression1 is the conditional test
- Expression2 is executed if test was TRUE
- Expression3 is executed if test was FALSE

Conditional expression with assignment

- The ternary operator (?:) will assign value from either expression2 or expression3 based on the Boolean result of expression1

```
variable = expression1 ? expression2 : expression3
```

- Expression1 is the conditional test
- Variable = Result of Expression2 if test was TRUE
- Variable = Result of Expression3 if test was FALSE

Conditional expression example 1

```
#include <stdio.h>
int a, b, z;

int main()
{
   a = 10;
   b = 3;
   z = (a > b) ? a : b;
   printf("z = %d\n", z);
   return 0;
}
```

Conditional expression example 2

```
#include <stdio.h>
int a, b;

int main()
{
  a = 10;
  b = 3;
  (a > b) ? printf("a is larger\n") : printf("b is larger\n");
  return 0;
}
```

Control structure

Control structure

- Sequential structure
  - Statements are executed in sequence
  - Default method of control
- Selection structure
  - Statement is executed depending on a logical condition
- Repetition structure
  - Statements are repeated as long as a logical condition is met

Sequential structure

- A sequential control structure is a series of statements executed one after another

```
statement1
statement2
…
statementN
```

Sequential statement example

```
int x = 2;
int y = 5;
printf("Value x = %d\n", x);
printf("Value y = %d\n", y);
```

Selection structure

- A selection control structure will have a conditional test and two possible outcomes (true or false)
- A statement may be associated with each logical outcome
- Flow control statements that have selection structure:
  - IF
  - IF-ELSE
  - IF-ELSEIF
  - IF-ELSEIF-ELSE
  - SWITCH

IF statement

- IF statement has only a single selection
- The outcome of the logical expression will result in either executing the associated statement or not

```
if (expression)
   statement
```

IF statement example 1

```
int x = 1;
if (x > 0)
  printf("x is positive\n");
```

IF statement example 2

```
int x = 1;
int y = 5;
if (x != y)
  printf("x is not equal to y\n");
```

IF statement example 3

```
int x = 5;
int y = 5;
if (x > 0 && x == y)
  printf("x is positive and equal to y\n");
```

IF statement example 4

```
int x = 0;
int y = 12; //non-zero is true
if (x && y)
  printf("x and y are true\n");
```

IF-ELSE statement

- IF statement has two selections
- The outcome of the logical expression will result in either executing the statement1 or statement2

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

IF-ELSE statement example 1

```
int x = -1;
if (x > 0)
  printf("x is positive\n");
else
  printf("x is negative\n");
```

IF-ELSE statement example 2

```
int x = 5;
int y = 5;
if (x != y)
  printf("x is not equal to y\n");
else
  printf("x is equal to y\n");
```

IF-ELSE statement example 3

```
int x = 1;
int min = 0;
int max = 5;
if (x > min && x <= max)
  printf("x is within range\n");
else
  printf("x is out of range\n");
```

IF-ELSE statement example 4

```
int x = -1;
int min = 0;
int max = 5;
if (x < min || x > max)
  printf("x is outside tolerance levels\n");
else
  printf("x is within tolerance levels\n");
```

IF-ELSEIF statement

- IF-ELSEIF statement has multiple selections
- Each logical expression is evaluated in the order of priority
  - Highest priority is first IF statement
  - Lowest priority is the last IF statement
- Any expression that is true will have the associated statement executed

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

IF-ELSEIF-ELSE statement

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

IF-ELSEIF statement example 1

```
int x = 2;
if (x == 1)
  printf("x is one");
else if (x == 2)
  printf("x is two");
else
  printf("x is unknown");
```

IF-ELSEIF statement example 2

```
int x = 1;
if (x > 0)
  printf("x is positive\n");
else if (x < 0)
  printf("x is negative\n");
else
  printf("x is zero\n");
```

SWITCH statement

- SWITCH statement will evaluate an expression and find a matching label, then execute the associated statement
- Label must be a single value
- The `break` keyword must be included after the statements
- Default case is optional

```
switch (expression) {
case <label1>:
  statements;
  break;
case <label2>:
  statements;
  break;
default:
  statements;
  break;
}
```

SWITCH statement example

```
int x = 2;
switch (x) {
case 1:
  printf("x is one");
  break;
case 2:
  printf("x is two");
  break;
default:
  printf("x is unknown");
  break;
}
```

Repetition structure

- A repetition control structure has a conditional test used to determine if the loop will continue or not
- Flow control statements with a repetition structure:
  - WHILE
  - DO-WHILE
  - FOR

WHILE statement

- WHILE statement is a loop that execute the statement when the expression is true
- A loop usually requires a counter variable to keep track of the number of rounds in a loop

```
while (expression)
   statement
```

WHILE statement example 1

```
int n = 0;
while (n < 5) {
   printf("Hello\n");
   n++;
}
```

WHILE statement example 2

```
int n = 1;
while (n < 5) {
   printf("%d ", n);
   n++;
}
```

WHILE statement example 3

```
int x;
int n = 1;
while (n < 4) {
   printf("Enter integer %d: ", n);
   scanf("%d", &x);
   printf("%d + %d = %d\n", n, x, n+x);
   n++;
}
```

DO-WHILE statement

- DO-WHILE statement is a loop similar to WHILE statement
- However, the statement is executed once before the expression is evaluated
- Remember to add a semicolon at the end of the expression

```
do
   statement
while (expression);
```

DO-WHILE statement example 1

```
int n = 0;
do {
   printf("Hello\n");
   n++;
} while (n < 5);
```

DO-WHILE statement example 2

```
int n = 1;
do {
   printf("%d ", n);
   n++;
} while (n < 5);
```

FOR statement

- FOR statement is a loop that runs for a fixed number of rounds
  - Use a counter variable to keep track of the loop
  - Expression1 is the initialisation of counter variable
  - Expression2 is the conditional test to check if loop needs to continue
  - Expression3 is to increase/decrease the counter variable

```
for (expression1; expression2; expression3)
  statement
```

FOR statement example 1

```
int n;
for (n = 0; n < 5; n++)
  printf("Hello\n");
```

FOR statement example 2

```
int n;
for (n = 1; n < 5; n++)
  printf("%d ", n);
```

FOR statement example 3

```
int x, n;
for (n = 1; n < 5; n++) {
   printf("Enter x value %d: ", n);
   scanf("%d", &x);
   printf("%d + %d = %d\n", n, x, n+x);
}
```

FOR statement example 4

```
// output 1 3 5 7 9
int x, y;
for (x = 0, y = 1; x+y < 10; x++, y++)
  printf("%d ", x+y);
```

Infinite loops

- An infinite loop is used to run statements forever
- This can be achieved using the WHILE or FOR statements
- Be cautious to use the infinite loop
  - Suggest adding conditions to get out of the loop if necessary

```
if(n>=10)
  break;
```

```
int n = 0;
while (1) {
  printf("%d\n", n);
  n++;
}
```

```
int n = 0;
for (;;) {
  printf("%d\n", n);
  n++;
}
```

Nested control structure

Nested statements

- Control structures can be nested to provide additional controls
- In the case of repetition control structure, it creates an inner loop structure

Nested selection structure

- An IF statement may be nested within another IF statement
- There are no limits to the number of levels of nested IF statements
- Braces should be used to avoid errors

```
if (expression1) {
   if (expression2) {
      statement
   }
}
```

Nested selection structure example 1

```
int x = 2;
int y = 5;
if (x > 0) {
  if (y > 0) {
    printf("x and y are positive\n");
  }
}
```

Nested selection structure example 2

```
int x = 2;
if (x != 0)
{
  if (x > 0)
  {
    printf("x is positive\n");
  }
  else if (x < 0)
  {
    printf("x is negative\n");
  }
}
else
{
  printf("x is zero\n");
}
```

Nested repetition structure

- Any loop structure may be nested within another loop structure
- There are no restrictions to the number of levels in a loop, it is best to set limits
- Braces should be used to avoid errors
- It can be used to generate patterns or process multi-dimensional arrays

Nested WHILE structure

```
while (expression1) {
  while (expression2) {
    statement
  }
}
```

Nested WHILE example

- The code is to generate the pattern:

```
*
**
***
```

- Outer loop runs for 3 rounds
- Inner loop depends on variables m and n
  - Round 1: n = 1, m = 0
  - Round 2: n = 2, m = 0, 1
  - Round 3: n = 3, m = 0, 1, 2

```
#include <stdio.h>
int main()
{
   int n = 1;
   int m;
   while (n < 4) {
      m = 0;
      while (m < n) {
         printf("*");
         m++;
      }
      printf("\n");
      n++;
   }
   return 0;
}
```

Nested FOR structure

```
for (expr1; expr2; expr3) {
  for (expr1; expr2; expr3) {
    statement
  }
}
```

Nested FOR example

- The code generates the pattern:

```
1
2 1
3 2 1
4 3 2 1
```

- Outer loop runs for 4 rounds
- Inner loop depends on both variables

```
#include <stdio.h>
int main()
{
  int n, m;
  for (n = 1; n < 5; n++) {
    for (m = n; m > 0; m--) {
      printf("%d ", m);
    }
    printf("\n");
  }
  return 0;
}
```

Summary

- Compound statements are used to create complex operations
- Operators are used to evaluate variables and values in conditional tests
- Three types of control structure exists: sequential, selection, repetition
- Nested loops are used to generate patterns or process multi-dimensional arrays

Further readings

- C: How to Program, 8th Edition, Paul Deitel & Harvey Deitel
  - Selections: pp. 105–109
  - Iterations: pp. 146–162
  - Operators: pp. 124–126, 164–168
- The C Programming Language, 2nd Edition, Kernighan & Ritchie
  - Selections: pp. 52–56
  - Iterations: pp. 56–60
  - Operators: pp. 40–51