Chapter 3: Control Statements

The `bool` Type and Operators

Often in a program you need to compare two values, such as whether i is greater than j.

C++ provides six *relational operators* (also known as *comparison operators*) that can be used to compare two values.

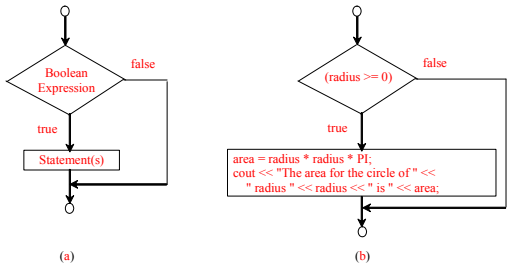
Comparison Operators

Operator	Name	Example	Result
<	less than	1 < 2	true
<=	less than or equal to	1 <= 2	true
>	greater than	1 > 2	false
>=	greater than or equal to	1 >= 2	false
==	equal to	1 == 2	false
!=	not equal to	1 != 2	true

Simple `if` Statements

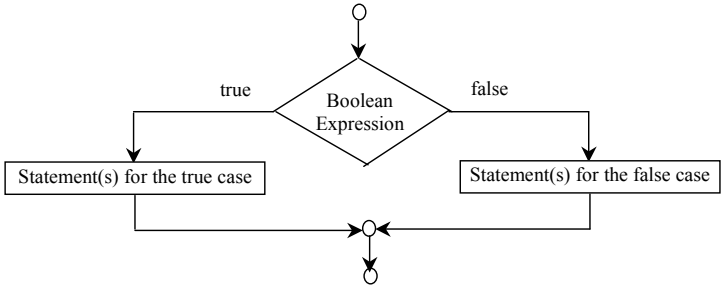
```
if (booleanExpression)
{
  statement(s);
}
```

```
if (radius >= 0)
{
  area = radius * radius * PI;
  cout << "The area for the circle of " <<
    " radius " << radius << " is " << area;
}
```



The `if...else` Statement

```
if (booleanExpression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



TIP

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

This is better

```
bool even
  = number % 2 == 0;
```

(b)

CAUTION

```
if (even == true)
  cout <<"It is even.";
```

(a)

Equivalent

This is better

```
if (even)
  cout << "It is even.";
```

(b)

```
if (even == false )
  cout <<"It is odd   ";
```

(a)

Equivalent

This is better

```
if (even)
  cout << "It is odd   ";
```

(b)

Example 1

```
#include <iostream>
using namespace std;

int main()
{
   bool y;
  // Prompt the user to enter two numbers integer
  int number1,number2;
   cout << "Enter  first integer number: ";
   cin >> number1;
   cout << "Enter  second integer number: ";
   cin >> number2;
```

```
if (number1>=number2)
  {
 cout<< number1 <<endl;
  }
  else
  {
cout<< number2 <<endl;
  }
return 0;
}
```

Example 1+

Run

```
#include <iostream>
using namespace std;

int main()
{
   bool y;
  // Prompt the user to enter two numbers integer
  int number1,number2;
   cout << "Enter  first integer number: ";
   cin >> number1;
   cout << "Enter  second integer number: ";
   cin >> number2;
```

```
if (number1>=number2)
  {
          y=true;

   }
  else
  {
          y=false;
    }
    cout<<y<<endl;
return 0;
}
```

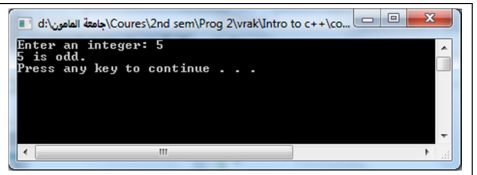

Example 2

Give a program that checks whether a number is even or odd.

The program prompts the user to enter an integer and displays "number is even" if it is even and "number is odd" if it is odd

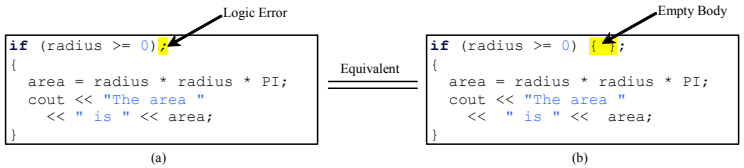
Example 2

```
#include <iostream>
using namespace std;

int main()
{
int number;
  cout << "Enter an integer: ";
  cin >> number;

  if (number % 2 == 0)
            cout << number << " is even."<<endl;
  else
  //if (number % 2 != 0)
    cout << number << " is odd."<<endl;
  system("pause");
  return 0;
}
```



Caution

Adding a semicolon at the end of an if clause is a common mistake.



This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

Boolean Operators

Operator	*Name*
!	not
&&	and
||	or

Truth Table for Operator !

p	!p
true	false
false	true

Example

!(1 > 2) is true, because (1 > 2) is false.

!(1 > 0) is false, because (1 > 0) is true.

Truth Table for Operator &&

p1	p2	p1 && p2
false	false	false
false	true	false
true	false	false
true	true	true

Example

(3 > 2) && (5 >= 5) is true, because (3 > 2) and (5 >= 5) are both true.

(3 > 2) && (5 > 5) is false, because (5 > 5) is false.

Truth Table for Operator ||

p1	p2	p1 || p2
false	false	false
false	true	true
true	false	true
true	true	true

Example

(2 > 3) || (5 > 5) is false, because (2 > 3) and (5 > 5) are both false.

(3 > 2) || (5 > 5) is true, because (3 > 2) is true.

Note

Outer parentheses required

Braces can be omitted if the block contains a single statement

```
if ((i > 0) && (i < 10))
{
  cout << "i is an " <<
    "integer between 0 and 10";
}
```

(a)

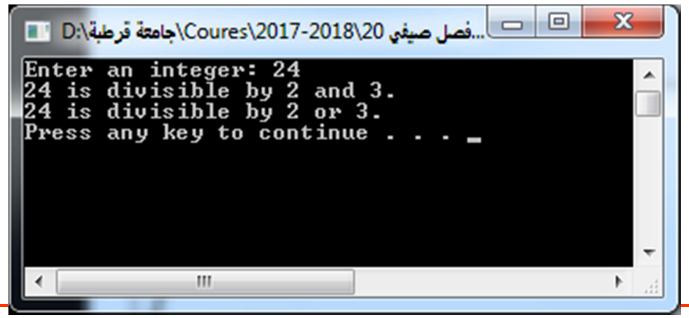
Equivalent

```
if ((i > 0) && (i < 10))
  cout << "i is an " <<
    "integer between 0 and 10";
```

(b)

Example 3 : Complex Conditions

Give a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:



Examples 3

Run

```
#include <iostream>
using namespace std;
int main()
{
  int number;
  cout << "Enter an integer: ";
  cin >> number;

  if (number % 2 == 0 && number % 3 == 0)
    cout << number << " is divisible by 2 and
3." << endl;

  if (number % 2 == 0 || number % 3 == 0)
    cout << number << " is divisible by 2 or
3." << endl;

  if ((number % 2 == 0 || number % 3 == 0)
&&
      !(number % 2 == 0 && number % 3 ==
0))
    cout << number << " divisible by 2 or 3,
but not both." << endl;
  system("pause");
  return(0);
}
```


Exercise

Write a program that lets the user enter a year and checks whether it is a leap year.

A year is a *leap year* if it is divisible by 4 but not by 100 or if it is divisible by 400. So you can use the following Boolean expression to check whether a year is a leap year:

(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)

Leap Year

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Enter a year: ";
  int year;
  cin >> year;
  // Check if the year is a leap year
  bool isLeapYear =    (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

  // Display the result in a message dialog box
  if (isLeapYear)
    cout << year << " is a leap year.";
  else
    cout << year << " is a not leap year.";

  return 0;
}
```


Nested if Statements

```
if (i > k)
{
  if (j > k)
    cout << "i and j are greater than k";
}
else
  cout << "i is less than or equal to k";
```


Multiple Alternative if Statements

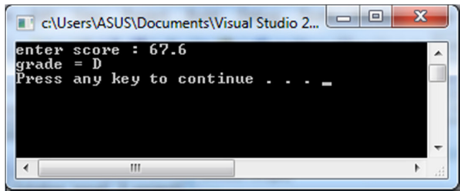
```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Nested if Statements example

```
#include <iostream>
using namespace std;
int main ()
{ double score;
 char grade;
 cout << "enter score : " ;
cin >> score ;
  if (score >= 90.0)
  grade = 'A';
  else if (score >= 80.0)
    grade = 'B';
    else if (score >= 70.0)
      grade = 'C';
      else if (score >= 60.0)
        grade = 'D';
        else
          grade = 'F';
cout<< "grade = "<<grade << endl;
return 0 ;  }
```



Note

The else clause matches the most recent if clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    cout << "A";
else
    cout << "B";
```

(a)

Output ?

Nothing is printed from the preceding statement. !!??!!

Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1; int j = 2; int k = 3;
if (i > j)
{
  if (i > k)
    cout << "A";
}
else
  cout << "B";
```

This statement prints B.

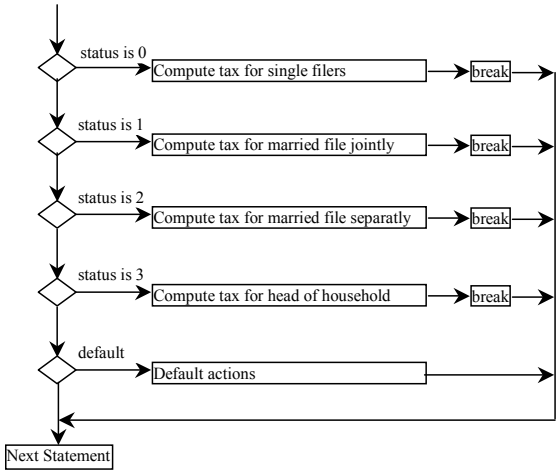
Example: Computing Taxes, cont.

```
if (status == 0)
{
  // Compute tax for single filers }
else if (status == 1)
{
  // Compute tax for married file jointly }
else if (status == 2)
{
  // Compute tax for married file separately }
else if (status == 3)
{
  // Compute tax for head of household }
else
{
  // Display wrong status }
```

switch Statements

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(0);
}
```


switch Statement Flow Chart



animation

Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
  case 'a': cout << ch;
  case 'b': cout << ch;
  case 'c': cout << ch;
}
```


animation

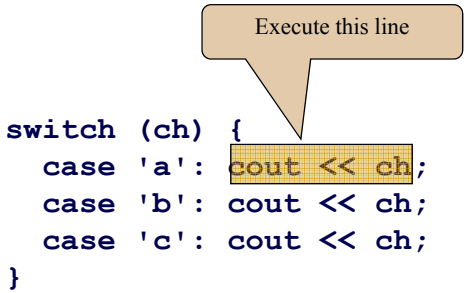
Trace switch statement

ch is 'a':

```
switch (ch) {
  case 'a': cout << ch;
  case 'b': cout << ch;
  case 'c': cout << ch;
}
```

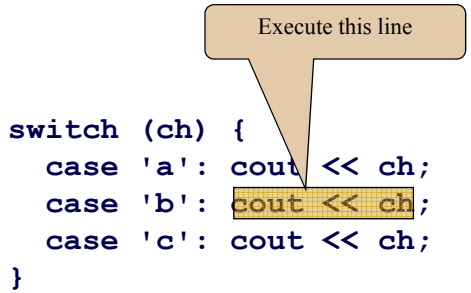
animation

Trace switch statement



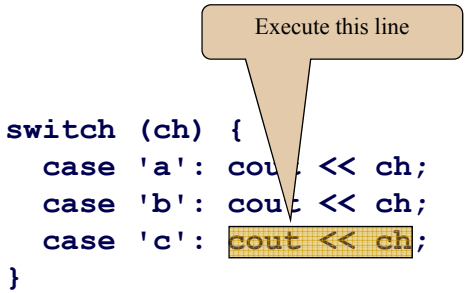
animation

Trace switch statement



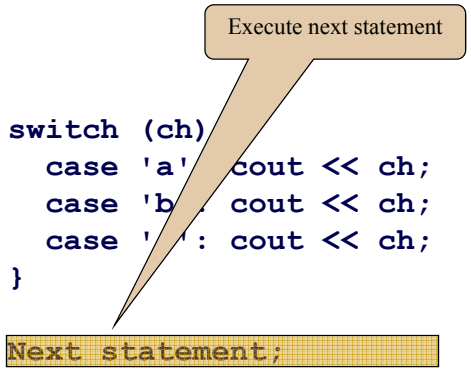
animation

Trace switch statement



animation

Trace switch statement



animation

Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
  case 'a': cout << ch;
            break;
  case 'b': cout << ch;
            break;
  case 'c': cout << ch;
}
```


animation

Trace switch statement

ch is 'a':

```
switch (ch) {
  case 'a': cout << ch;
            break;
  case 'b': cout << ch;
            break;
  case 'c': cout << ch;
}
```


animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': cout << ch;
            break;
  case 'b': cout << ch;
            break;
  case 'c': cout << ch;
}
```


animation

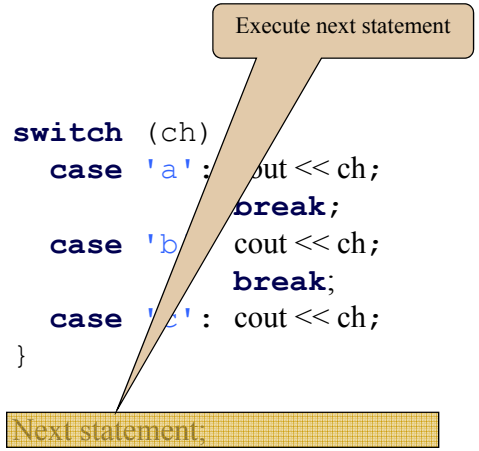
Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': cout << ch;
            break;
  case 'b': cout << ch;
            break;
  case 'c': cout << ch;
}
```

animation

Trace switch statement



switch – case exercise

أكتب برنامج لإعطاء اسم اليوم من أيام الأسبوع عند إعطاء رقمه .

```
#include <iostream>
using namespace std;
int main ()
{int c ;
cout << "enter number : " ;
cin >> c ;
switch (c )
  {        case 1 : { cout << " saturday " ; break ; }
           case 2 : { cout << " sunday " ; break ; }
           case 3 : { cout << " monday " ; break ; }
           case 4 : { cout << " tuesday " ; break ; }
           case 5 : { cout << " wednesday " ; break ; }
           case 6 : { cout << " thursday " ; break ; }
           case 7 : { cout << " friday " ; break ; }
           default : { cout << " that number is out of range " ; }
}
return 0 ; }
```

Conditional Operator

```
if (x > 0)
  y = 1
else
  y = -1;
```

is equivalent to

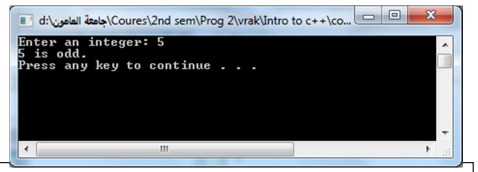
```
y = (x > 0) ? 1 : -1;
(booleanExpression) ? expression1 : expression2
```

Example:

```
cout << ((num % 2 == 0) ? "num is even" : "num is odd");
```


Example

Give a program that checks whether a number is even or odd.




```
#include <iostream>
using namespace std;

int main()
{
int number;
  cout << "Enter an integer: ";
  cin >> number;
  cout << ((number % 2 == 0) ? "num is even" : "num is odd")<<endl;

return 0;
}
```

Enumerated Types

enum Day {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};

Once a type is defined, you can declare a variable of that type:

Day day;

The variable day can hold one of the values defined in the enumerated type. For example, the following statement assigns enumerated value MONDAY to variable day:

day = MONDAY;

Enumerated Types

As with any other type, you can declare and initialize a variable in one statement:

Day day = MONDAY;

Furthermore, C++ allows you to declare an enumerated type and variable in one statement. For example,

enum Day {MONDAY' TUESDAY, WEDNESDAY, THURSDAY, FRIDAY} day = MONDAY;
MONDAY=1 or ='a',

Enumerated Examples

Run

```
#include <iostream>
using namespace std;

int main()
{
  enum Day {MONDAY = 1, TUESDAY,
WEDNESDAY, THURSDAY, FRIDAY}
day;

  cout << "Enter a day (1 for Monday, 2
for Tuesday, etc): ";
  int dayNumber;
    cin >> dayNumber;

  switch (dayNumber) {
    case MONDAY:
      cout << "Play soccer" << endl;
      break;
    case TUESDAY:
      cout << "Piano lesson" << endl;
      break;
    case WEDNESDAY:
      cout << "Math team" << endl;
      break;
    default:
      cout << "Go home" << endl;
  }
  system("pause");
  return 0;
}
```


Example: A Simple Math Learning Tool

This example creates a program for a first grader to practice subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 > number2 and displays a question such as "What is 9 – 2?" to the student, as shown in the sample output. After the student types the answer, the program displays a message to indicate whether the answer is correct.

```
v1 = rand() % 100;
// v1 in the range 0 to 99
v2 = rand() % 100 + 1;
// v2 in the range 1 to 100
v3 = rand() % 30 + 1985;
// v3 in the range 1985-2014
```