جامعة
القاهرة الجديدة
التكنولوجية

NEW CAIRO
TECHNOLOGICAL
UNIVERSITY

TUE – The Technological Universities in Egypt

NCTU – New Cairo Technological University

Faculty of Industry and Energy Technology

Information Technology Department

Second-Year

Course: Programming Essentials in C++

Lecture 4

Presented by

Dr. Ghada Maher

Contents:

- C++ Decision Making Statements (Selection control statements)
  - The if Statement
  - The if..else Statement
  - Keywords
  - Comparison Operators
  - Statement Blocks
  - Compound Conditions
  - Short-circuiting
  - Boolean Expressions
  - Nested Selection Statements
  - The else if Construct
  - The switch Statement
  - The Conditional Expression Operator

Compound Conditions

p && q	evaluates to true if and only if both p and q evaluate to true
p || q	evaluates to false if and only if both p and q evaluate to false
!p	evaluates to true if and olny if p evaluates to false

For example, `(n % d || x >= y)` will be false if and only if `n % d` is zero and `x` is less than `y`.

The definitions of the three logical operators are usually given by the *truth tables* below.

p	q	p && q
T	T	T
T	F	F
F	T	F
F	F	F

p	q	p || q
T	T	T
T	F	T
F	T	T
F	F	F

p	!p
T	F
F	T

These show, for example, that if `p` is true and `q` is false, then the expression `p && q` will be false and the expression `p || q` will be true.

The next example solves the same problem that Example 3.5 on page 39 solved, except that it uses compound conditions.

Example 8: Using Compound Conditions


```
int main()
{ int n1,n2,n3;
cout << "Enter three integers: ";
cin >> n1 >> n2 >> n3;
if (n1 <= n2 && n1 <= n3) cout << "Their minimum is " << n1 <<endl;
if (n2 <= n1 && n2 <= n3) cout << "Their minimum is " << n2 <<endl;
if (n3 <= n1 && n3 <= n2) cout << "Their minimum is " << n3 <<endl;
}
```

Run:

Enter two integers: **77 33 55**
Their minimum is 33

Example 9: User-Friendly Input

This program allows the user to input either a “Y” or a “y” for “yes”:

```
int main()
{ char ans;
cout << "Are you enrolled (y/n): ";
cin >> ans;
if (ans == 'Y' || ans == 'y') cout << "You are enrolled.\n";
else cout << "You are not enrolled.\n";
}
```

Run:

```
Are you enrolled (y|n): N
You are not enrolled.
```

Short-circuiting

Compound conditions that use **&&** and **||** will not even evaluate the second operand of the condition unless necessary. This is called *short-circuiting*. As the truth tables show, the condition p && q will be false if p is false. In that case there is no need to evaluate q. Similarly if p is true then there is no need to evaluate q to determine that p || q is true. In both cases the value of the condition is known as soon as the first operand is evaluated.

Example 10: Short-Circuiting


```
int main()      //This program tests integer divisibility
{ int n,d;
cout << "Enter two positive integers: ";
cin >> n >> d;
if (d != 0 && n%d == 0) cout << d << " divides " << n << endl;
else cout << d << " does not divide " << n << endl;
}
```

In this run,d is positive and n%d is zero, so the compound condition is true:

Enter two positive integers: **300 6**
6 divides 300

In this run, d is positive but n%d is not zero, so the compound condition is false:

Enter two positive integers: **300 7**
7 does not divide 300

In this run,d is zero, so the compound condition is immediately determined to be false without evaluating the second expression "n%d == 0":

Enter two positive integers: **300 0**
0 does not divide 300

Example 11: Another Logical Error

This program is erroneous:

```
int main()
{ int n1,n2,n3;
cout << "Enter three integers: ";
cin >> n1 >> n2 >> n3;
if (n1 >= n2 >= n3) cout << "max = "<<n1; // LOGICAL ERROR!
}
```

Run:

```
Enter an integer: 0 0 1
max = 0
```

The source of this error is the fact that boolean expressions have numeric values. Since the expression (n1 >= n2 >= n3) is evaluated from left to right, the first part n1 >= n2 evaluates to "true" since 0 >= 0. But "true" is stored as the numeric value 1. That value is then compared to the value of n3 which is also 1, so the complete expression evaluates to "true" even though it is really false! (0 is not the maximum of 0, 0, and 1.) The problem here is that the erroneous line is syntactically correct, so the compiler cannot catch the error.

Nested Selection Statements

Like compound statements, selection statements can be used wherever any other statement can be used. So a selection statement can be used within another selection statement. This is called *nesting* statements.

Example 12: Using Nested Selection Statements


```
int main()
{ int n1,n2,n3;
cout << "Enter three integers: ";
cin >> n1 >> n2 >> n3;
if (n1 < n2)
        if (n1 < n3) cout << "Their minimum is " << n1 << endl;
                else cout << "Their minimum is " << n3 << endl;
else // n1 >= n2
if (n2 < n3) cout << "Their minimum is " << n2 << endl;
else cout << "Their minimum is " << n3 << endl;
}
Run:
```

Enter three integers: **77 33 55**
Their minimum is 33

The else if Construct

Nested **if..else** statements are often used to test a sequence of parallel alternatives, where only the **else** clauses contain further nesting. In that case, the resulting compound statement is usually formatted by lining up the **else if** phrases to emphasize the parallel nature of the logic.

Example 13: Using the else if Construct for Parallel Alternatives

This program requests the user's language and then prints a greeting in that language:

```
int main()
{ char language;
cout << "Engl.,Fre n.,Ger.,Ital.,or Rus.? (e|f|g|i|r): ";
cin >> language;
if (language == 'e') cout << "Welcome to ProjectEuclid.";
else if (language == 'f') cout << "Bon jour,ProjectEuclid.";
else if (language == 'g') cout << "Guten tag,ProjectEuclid.";
else if (language == 'i') cout << "Bon giorno,ProjectEuclid.";
else if (language == 'r') cout << "Dobre utre,ProjectEuclid.";
else cout << "Sorry; we don't speak your language.";
}
```

Run:

```
Engl., Fren., Ger., Ital., or Rus.? (e|f|g|i|r): i
Bon giorno, ProjectEuclid.
```

This program uses nested `if..else` statements to select from the five given alternatives.

As ordinary nested `if..else` statements, the code could also be formatted as

```
if (language == 'e') cout << "Welcome to ProjectEuclid.";
else
  if (language == 'f') cout << "Bon jour, ProjectEuclid.";
  else
    if (language == 'g') cout << "Guten tag, ProjectEuclid.";
    else
      if (language == 'i') cout << "Bon giorno, ProjectEuclid.";
       else
         if (language == 'r') cout << "Dobre utre, ProjectEuclid.";
         else cout << "Sorry; we don't speak your language.";
```

But the given format is preferred because it displays the parallel nature of the logic more clearly. It also requires less indenting.

Example 14: Using the else if Construct to Select a Range of Scores

This program converts a test score into its equivalent letter grade:

```
int main()
{ int score;
cout << "Enter your test score: "; cin >> score;
if (score > 100) cout << "Error: that score is out of range.";
else if (score >= 90) cout << "Your grade is an A." << endl;
else if (score >= 80) cout << "Your grade is a B." << endl;
else if (score >= 70) cout << "Your grade is a C." << endl;
else if (score >= 60) cout << "Your grade is a D." << endl;
else if (score >= 0) cout << "Your grade is an F." << endl;
else cout << "Error: that score is out of range.";
}
```

Run:
Enter your test score: **83**
Your grade is a B.

the switch statement

The switch statement can be used instead of the else if construct to implement a sequence of parallel alternatives. Its syntax is

switch (*expression*)
{
case *constant1*: *statementList1*;
case *constant2*: *statementList2*;
case *constant3*: *statementList3*;
:
:
case *constantN*: *statementListN*;
default: *statementList0*;
}

Example 15: Using a switch Statement to Select a Range Scores


```
int main()
{ int score;
cout << "Enter your test score: "; cin >> score;
switch (score/10)
{ case 10:
case 9: cout << "Your grade is an A." << endl; break;
case 8: cout << "Your grade is a B." << endl; break;
case 7: cout << "Your grade is a C." << endl; break;
case 6: cout << "Your grade is a D." << endl; break;
case 5:
case 4:
case 3:
case 2:
case 1:
case 0: cout << "Your grade is an F." << endl; break;
default: cout << "Error: score is out of range.\n";
}
cout << "Goodbye." << endl;
}
```

Run:
Enter your test score: **83**
Your grade is a B.
Goodbye.

An Erroneous Fall-through in a switch Statement

It is normal to put a break statement at the end of each case clause in a switch statement. Without it, the program execution will not branch directly out of the switch block after it finishes executing its case statement sequence. Instead, it will continue within the switch block, executing the statements in the next case sequence. This (usually) unintended consequence is called a *fall through*.

Example 16: An Erroneous Fall-through in a switch Statement


```
int main()
{ int score;
cout << "Enter your test score: "; cin >> score;
switch (score/10)
{ case 10:
case 9: cout << "Your grade is an A." << endl; // LOGICAL ERROR
case 8: cout << "Your grade is a B." << endl; // LOGICAL ERROR
case 7: cout << "Your grade is a C." << endl; // LOGICAL ERROR
case 6: cout << "Your grade is a D." << endl; // LOGICAL ERROR
case 5:
case 4:
case 3:
case 2:
case 1:
case 0: cout << "Your grade is an F." << endl; // LOGICAL ERROR
default: cout << "Error: score is out of range.\n";
}
cout << "Goodbye." << endl;
}
```

Run:
Enter your test score: **83**
Your grade is a B.
Your grade is a C.
Your grade is a D.
Your grade is an F.
Error: score is out of range.
Goodbye.

The Conditional Expression Operator

- C++ provides a special operator that often can be used in place of the **if...else** statement. It is called the *conditional expression operator*. It uses the **?** and the **:** symbols in this syntax:

  condition **?** *expression1* **:** *expression2*

- It is a *ternary operator*; *i.e.*, it combines three operands to produce a value. That resulting value is either the value of *expression1* or the value of *expression2*, depending upon the Boolean value of the *condition*. For example, the assignment

  min = **(x<y ? x : y)**;

- would assign the minimum of x and y to min, because if the condition **x<y** is true, the expression **(x<y ? x : y)** evaluates to x; otherwise it evaluates to y.

- Conditional expression statements should be used sparingly: only when the condition and both expressions are very simple.

Example 17: Finding the Minimum using The Conditiona Expression Operator


```
int main()
{ int m,n;
cout << "Enter two integers: ";
cin >> m >> n;
cout << ( m<n ? m : n ) << " is the minimum." << endl;
}
```

Run:
Enter two integers: **77 55**
55 is the minimum.

The conditional expression **(m<n ? m : n)** evaluates to m if m<n, and to n otherwise.

References:

John R. Hubbard, "*Schaum's outline of* theory and problems of programming with C++", Second Edition, Schaum's Outline Series, McGRAW-HILL, *New York San Francisco Washington.*