If Statements

Conditions, Logical Expressions, and Selection Control Structures

Flow of Control

- **Flow of Control** - the order in which the computer executes statements in a program
- Flow of control is normally sequential - one statement executed after the next
- **Control structures** (statements used to alter the normally sequential flow of control) allow for non-sequential flow of control

- **Selection control structure** -
  - Make an *assertion* (claim) that is either true or false
  - If the claim is true, execute one set of statements
  - If the claim is false, execute a different set of statements
Conditions and Logical Expressions

- **Asking questions in C++** means making an assertion that is either True or False
- The `bool` DataType is used for Boolean data which consist of just two values: `true` or `false`
  - `true` has a value of 1
  - `false` has a value of 0
  - **Any non zero number will evaluate as `true`**
- `true` and `false` are special constants in C++ and are reserved words. They are not variable names or strings
- Variables of DataType `bool` are declared the same way as other DataTypes:
  ```
  bool finished;    // Is the process completed?
  bool verified;    // Has the data been verified?
  ```
Logical Expressions

- **Logical expressions**
  - are made up of logical values and operations;
  - every logical expression evaluates true or false

- Examples of logical expressions: (later we see how these are used)
  1) A single Boolean variable or constant
  2) **Relational expression**: An expression followed by a relational operator followed by an expression (these can be any expression). The result of a relational expression is of DataType bool
  3) A logical expression followed by a logical operator followed by a logical expression

Relational Operators

- **Relational operators** test a relationship between two expressions and result in a Boolean value.

```cpp
bool test; int j,k;
 cin >> j >> k; // input two integer numbers
 test = (j < k); // if j is less than k, then test is assigned // the value of true else it is assigned the value // of false. If j = k the value assigned is false
```

- **Relational Operators**
  ```cpp
   == equal to**(see next slide)
   != Not equal to
   > Greater than
   < Less than
   >= Greater than or equal to
   <= Less than or equal to
  ```

See If_01.cpp for C++ examples of Boolean expressions
Relational Operators (continued)

- Expressions used with relational operators can consist of integers, floats, strings or chars. In the case of strings and chars, **uppercase letters are less than lowercase letters.**

- When making comparisons, always compare like DataTypes

- **NOTE THE RELATIONAL OPERATOR == IS COMPLETELY DIFFERENT FROM THE ASSIGNMENT OPERATOR =**

  \[
  A == B \text{ means: is } A \text{ equal to } B, \text{ and} \]

  \[
  A = B \text{ means: assign the value of } B \text{ to } A
  \]

  See If_02.cpp for C++ examples of relational operators

---

Relational Operator Examples

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Relational Expression</th>
<th>Result is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>Result = (X*2 &lt;= Y);</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>Result = (X == Y);</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Result = (X<em>5 != Y</em>2)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>Result = (Y &gt;= X)</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>Result = (X &gt;= Y)</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>Result = (X == Y)</td>
<td></td>
</tr>
</tbody>
</table>

- Comparing strings: what we have been calling string variables are more correctly called string objects. Literal string values - such as "letters" - are also called C strings.

  - Comparisons can be made between 2 string objects or a string object and a C string (literal string), but not two C strings
  - The strings are compared on a character by character basis until either a mismatch occurs or the final character of one of the strings has been reached
Relational Operator Examples

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Relational Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>Result = (X*2 &lt;= Y);</td>
<td>False</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>Result = (X == Y);</td>
<td>False</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Result = (X<em>5 != Y</em>2)</td>
<td>False</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>Result = (Y &gt;= X)</td>
<td>True</td>
</tr>
<tr>
<td>‘M’</td>
<td>‘m’</td>
<td>Result = (X &gt;= Y)</td>
<td>False</td>
</tr>
<tr>
<td>‘A’</td>
<td>‘A’</td>
<td>Result = (X == Y)</td>
<td>True</td>
</tr>
</tbody>
</table>

Logical Operators

- The logical operators **AND **&&, **OR **|| and **NOT **! take logical expressions as operands.

- The **&& **operator requires both operands (logical expressions) to be **true **in order for the result to be **true.**
  - \( (\text{true } \&\& \text{ true}) \) evaluates as **true**
  - \( (\text{false } \&\& \text{ true}) \)
  - \( (\text{false } \&\& \text{ false}) \)
  - \( (\text{true } \&\& \text{ false}) \)

- The **|| **operator requires at least one operand (logical expression) to be **true **in order for the result to be **true.**
  - \( (\text{false } \mid\mid \text{ false}) \) evaluates as **false**
  - \( (\text{false } \mid\mid \text{ true}) \)
  - \( (\text{true } \mid\mid \text{ true}) \)
  - \( (\text{true } \mid\mid \text{ false}) \)

CPE 112: If - Relational Operator Examples
Logical Operators

- The ! Operator precedes a single logical expression and gives its opposite as the result
  - !true evaluates as false;
  - !false evaluates as true.
  - !(logical expression) evaluates as the opposite of what the logical expression evaluates

- Evaluation of logical expressions proceeds from left to right, and the computer stops the evaluation as soon as possible - when it knows the truth of the entire expression (short-circuit evaluation)

Logical Operators (continued)

- If an operand of a logical operator is not a logical expression, the value of the operand is temporarily coerced to type bool as follows:
  - A 0 value is coerced to false, and all others to true
  - literal strings and literal chars do compile and do evaluate to true
  - string variable and const string do not work - compile error
  - char variable and const char do compile and do evaluate to true

- Read the May we introduce George Boole - pg. 212
Precedence of Operators Revisited

- Arithmetic, relational and logical operators:
  - !, Unary +, Unary - (Highest precedence)
  - *, /, % (multiply, divide, modulo)
  - +, - (add, subtract)
  - Relational Operators: <, <=, >, >= (less than, less than equal to, greater than, greater than equal to)
  - Relational Operators: ==, != (is equal to, is not equal to)
  - Logical Operator: && (AND)
  - Logical Operator: || (OR)
  - = (assignment, “set equal to”) (Lowest precedence)
- Parentheses can over-ride the precedence order. Evaluation is from left to right with highest operations performed first

Logical Operator Examples

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>!(A == B)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>-3</td>
<td>!(A &gt; B &amp;&amp; C &lt; D)</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>2</td>
<td>A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>-2</td>
<td>(A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-2</td>
<td>(B</td>
<td></td>
</tr>
</tbody>
</table>

See If_03.cpp for C++ examples of Logical Operators
Logical Operator Examples

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>EXPRESSION</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>!(A == B)</td>
<td>TRUE</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>-3</td>
<td>!(A &gt; B &amp;&amp; C &lt; D)</td>
<td>TRUE</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>2</td>
<td>A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>-2</td>
<td>(A &gt; 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-2</td>
<td>(B</td>
<td></td>
</tr>
</tbody>
</table>

CPE 112: If - Logical Operator Example

The if Statement

• The if statement is the fundamental control structure that allows branches in the flow of control.
• The if statement allows the program to decide which action is to be taken given some input or calculation result.

• SYNTAX TEMPLATE: IfStatement (if-then-else form)
  
  ```
  if (Logical Expression)
  Statement1A  //executed if the expression evaluates as true
  else
  Statement1B  //executed if the expression evaluates as false
  ```

• if and else are reserved words.
• observe the indentation of the statement
**if statement (continued)**

- Note that the `if` statement line itself does not have `;`
- **SYNTAX TEMPLATE:** IfStatement (if-then form - no else is used)
  
  ```
  if (Logical Expression)
  
  Statement 1A // execute if expression is true
  ```

- **Unless block statements are used, the if or else statement can have ONE line after it only** - Regardless of how you indent the program (more on Blocks shortly)

  ```
  string myString = "hgfdAijgk";
  string::size_type pos;
  pos = myString.find("A");
  if (pos == string::npos)
      cout << "no 'A' found in the string\n";
  else
      cout<<"an 'A' found in pos. "<<pos<< endl;
  ```

---

**if statement Example, Block Statements**

- If more than one statement is required, block statements can be utilized. These statements are enclosed in braces `{}`
- Never use a semicolon after the right brace of a block

  ```
  if (a > b)
  {
      statement1A;
      statement2A;
      statement3A;
  }
  else
  {
      statement1B;
      statement2B;
      statement3B;
  }
  ```
More Examples

```cpp
if ( x > 40.0) // if-then statement
    cout << "X is too big";

if ( y != 30) // if-then-else statement
    cout << "y is not 30
";
else
    cout << "y is 30
";

if ( a == 30 && b != 10) // if-then using a block
{
    cout << "Conditions are almost met for a";
    cout << " perfect score
";
    cout << "Just need a score of 75 on c
";
}

See If_05.cpp for C++ examples of if statements
```

if-then-else-if Statements

- **Unnested if statements** - Note: here each if statement is executed
  ```
  if (a > 40)
      cout << "a is > 40";
  if (a > 30)
      cout << "a is > 30";
  if (a > 20)
      cout << "a is > 20";
  ```

- **SYNTAX TEMPLATE:** IfStatement (if-then-else-if form)
  ```
  if (Logical Expression #1)
      Statement 1A    // execute if expression #1 is true
  else if (Logical Expression #2)
      Statement 2A    // execute if expression #2 is true and #1 is false
  else
      Statement 3A    // execute if expressions #1 and #2 are false
  ```
More on if Statements

- Logic is important with nested if statements. Programmers need to make sure all conditions are tested, no tests are skipped and that the testing order is correct.

- Beware of the dangling else - remember that indentation does not affect the execution of the code.
  - The else statement is always paired with the closest preceding if that doesn’t already have an else paired with it.
  - can use {} to make a block statement to avoid the problem.

Dangling else Example

- Incorrect logic for desired output
  ```cpp
  if (first_name == "Ron")
    if (last_name == "Bowman")
      cout << "Instructor\n";
  else
    cout << "Wrong first name for instructor\n";
  ```

- Corrected logic – using block statements
  ```cpp
  if (first_name == "Ron")
  {
    if (last_name == "Bowman")
      cout << "Instructor\n";
  }
  else
    cout << "Wrong first name for instructor\n";
  ```

See If_07.cpp for C++ example of if-then-else-if and dangling else.
A VERY Common Mistake

- A mistake that can cause confusion in operation of a program is using '=' instead of '=='

```c++
int n, m;
cout << "enter 2 integers\n";
cin >> n >> m;
if (n = m)  //assigns the value of m to n  
// then tests the value of n
    cout << "The numbers are the same\n";
else
    cout << "the numbers are not the same\n";
```

As long as m != 0, this always outputs - The numbers are the same

See If_06.cpp for C++ example of this mistake.
Also contains if-then-else examples

Testing the State of an I/O Stream

- Test the name of a stream object as if it were a Boolean variable
  - Examples: if (cin) if (!outFile) if (inFile)
  - Result is either true if last stream operation was successful or
  - Result is false if the last stream operation failed

```c++
ifstream inFile; ofstream outFile;
inFile.open("mydata.dat");
if (!inFile) // if file not opened, terminate program
    { cout << "open of mydata.dat failed\n"; return 1; }
outFile.open("output.dat");
if (outFile)
    {cout << "output.dat was successfully opened\n"; ...}
else
    {cout << "open of the output file failed\n"; return 1;}
```

See If_08.cpp for C++ example of testing I/O streams
Testing the State of an I/O Stream

• An input stream in the fail state mode can be reset and then reused
  • For the standard input stream – cin
    if (!cin) // is input stream in the fail state mode?
    
    cin.clear(); // reset the input stream for use
    // remove the fail state cause from the input stream
    cin.ignore(2000,'\n');
    
    } // end if
  • For user defined input file streams – i.e. ifstream inFile – once it is in the fail state mode, it can be reset.
    if (!inFile) // input file stream inFile in the fail state?
    
    inFile.clear();
    inFile.ignore(2000,'\n'); // depends on cause of the fail state
    
    } // end if

File Stream Status Functions

• good() – returns true if the file stream has no error flags set – file stream can be used
• bad() – returns true if previous I/O operation results in a loss of integrity of the stream (This is not the opposite of good)
• fail() – checks if either the fail status bit or bad status bit has been set. This function returns true when some error occurred with a stream operation other than reaching the end of the file
• eof() – returns true if the end of the file is reached with the previous I/O operation.
  • This function returns false until there is no more data in the file (typically the last character in a file will be the newline character).
  • This function will return true if the last newline character is read, or a read beyond the end of the file is attempted.
• Examples: In.good(), In.bad(), In.fail(), In.eof()
Testing and Debugging

- Read problem-solving case study pg. 217-222
- Testing in the problem-solving phase (pg. 223-225)
- Testing in the Implementation Phase (pg. 225-229)
- Testing and debugging hints - pages 231 - 232. READ THIS SECTION
- Read through the quick check, exam preparation exercises and programming warm-up exercises. Pages 233-235