Control Structures - Intro, Selection

Flow of Control:

Flow of control through any given function is implemented with three basic types of control structures:

- **Sequential**: default mode. Sequential execution of code statements (one line after another) -- like following a recipe
- **Selection**: used for decisions, branching -- choosing between 2 or more alternative paths. In C++, these are the types of selection statements:
  - `if`
  - `if/else`
  - `switch`
- **Repetition**: used for looping, i.e. repeating a piece of code multiple times in a row. In C++, there are three types of loops:
  - `while`
  - `do/while`
  - `for`

The *function* construct, itself, forms another way to affect flow of control through a whole program. This will be discussed later in the course.

Some useful tools for building programs or program segments

- pseudocode - helps "think" out a problem or algorithm before trying to code it
- flowcharting - graphical way to formulate an algorithm or a program's flow
- stepwise refinement (top-down design) of algorithms

True and False

- Selection and repetition statements typically involve decision steps. These steps rely on conditions that are evaluated as **true** or **false**
- C++ has a boolean data type (called `bool`) that has values `true` and `false`. Improves readability.
- Most functions that answer a yes/no question (or a true/false situation) will return a boolean answer (or in the case of user-defined functions, they *should* be coded that way)
- **Important**: ANY C++ expression that evaluates to a value (i.e. any R-value) can be interpreted as a true/false condition. The rule is:
  - If an expression evaluates to 0, its truth value is **false**
  - If an expression evaluates to non-zero, its truth value is **true**

Logical Operators:

The arithmetic comparison operators in C++ work much like the symbols we use in mathematics. Each of these operators returns a **true** or a **false**.

```
x == y        // x is equal to y
x != y        // x is not equal to y
x < y         // x is less than y
x <= y        // x is less than or equal to y
x > y         // x is greater than y
x >= y        // x is greater than or equal to y
```

We also have Boolean operators for combining expressions. Again, these operators return **true** or **false**

```
x && y        // the AND operator -- true if both x and y are true
x || y        // the OR operator -- true if either x or y (or both) are true
!x            // the NOT operator (negation) -- true if x is false
```

These operators will be commonly used as test expressions in selection statements or repetition statements (loops).

Examples of expressions

```
(x > 0 && y > 0 && z > 0)     // all three of (x, y, z) are positive
(x < 0 || y < 0 || z < 0)     // at least one of the three variables is negative

( numStudents >= 20 && !(classAvg < 70))
      // there are at least 20 students and the class average is at least 70

( numStudents >= 20 && classAvg >= 70)
      // means the same thing as the previous expression
```

Short Circuit Evaluation:

- The `&&` and `||` operators also have a feature known as **short-circuit evaluation**.
- In the Boolean AND expression `(X && Y)`, if `X` is false, there is no need to evaluate `Y` (so the evaluation stops). Example:

```
(d != 0 &&  n / d > 0)

// notice that the short circuit is crucial in this one.  If d is 0,
//  then evaluating (n / d)  would result in division by 0 (illegal). But
//  the "short-circuit" prevents it in this case.  If d is 0, the first
//  operand (d != 0) is false.  So the whole && is false.
```

- Similarly, for the Boolean OR operation `(X || Y)`, if the first part is true, the whole thing is true, so there is no need to continue the evaluation. The computer only evaluates as much of the expression as it needs. This can allow the programmer to write faster executing code.

Selection Statements

The `if/else` Selection Statement

- The most common selection statement is the **`if/else`** statement. Basic syntax:

```
if (expression)
   statement
else
   statement
```

- The else clause is optional, so this format is also legal:

```
if (expression)
   statement
```

- The *expression* part can be any expression that evaluates a value (an R-value), and it **must** be enclosed in parintheses ( ).
  - The best use is to make the expression a **Boolean expression**, which is an operation that evaluates to **true** or **false**
  - For other expressions (like `(x + y)`, for instance):
    - an expression that evaluates to 0 is considered **false**
    - an expression that evalautes to anything else (non-zero) is considered **true**
- The *statement* parts are the "bodies" of the if-clause and the else-clause. The *statement* after the if or else clause **must** be either:
  - an empty statement

```
;
```

  - a single statement

```
expression;
```

  - a **compound statement** (i.e. a block). Can enclose multiple code statements. Remember, a compound statement is enclosed in set braces { }

- Appropriate indentation of the bodies of the if-clause and else-clause is a very good idea (for human readability), but irrelevant to the compiler

Examples

```
if (grade >= 68)
   cout << "Passing";
```

// Notice that there is no else clause. If the grade is below 68, we move on.

```
if (x == 0)
   cout << "Nothing here";
else
   cout << "There is a value";
```

// This example contains an else clause. The bodies are single statements.

```
if (y != 4)
{
   cout << "Wrong number";
   y = y * 2;
   counter++;
}
else
{
   cout << "That's it!";
   success = 1;
}
```

Multiple statements are to be executed as a result of the condition being true or false. In this case, notice the compound statement to delineate the bodies of the if and else clauses.

Be careful with **if**s and **else**s. Here's an example of an easy mistake to make. If you don't use { }, you may think that you've included more under an `if` condition than you really have.

// What output will it produce if val = 2? Does the "too bad" statement really go with the "else" here?

```
if (val < 5)
   cout << "True\n";
else
   cout << "False\n";
   cout << "Too bad!\n";
```

* Indentation is only for people! It improves readability, but means nothing to the compiler.

Example links

- Miscellaneous if/else examples
- Example program: Figuring a letter grade
- Example program: Computing overtime pay

Some common errors

What's wrong with these if-statements? Which ones are syntax errors and which ones are logic errors?

- ```
  if (x == 1 || 2 || 3)
     cout << "x is a number in the range 1-3";
  ```

- ```
  if (x > 5) && (y < 10)
     cout << "Yahoo!";
  ```

- ```
  if (response != 'Y' || response != 'N')
     cout << "You must type Y or N (for yes or no)";
  ```

The `switch` statement

- A **switch** statement is often *convenient* for occasions in which there are multiple cases to choose from. The syntax format is:

  ```
  switch (expression)
  {
      case constant:
          statements
      case constant:
          statements

      ...  (as many case labels as needed)

      default:            // optional label
          statements
  }
  ```

- The switch statement evaluates the *expression*, and then compares it to the values in the case labels. If it finds a match, execution of code jumps to that case label.
- The values in case labels must be *constants*, and may only be integer types, which means that you
  - This means only integer types, type `char`, or enumerations (not yet discussed)
  - This also means the case label must be a *literal* or a variable declared to be `const`
  - **Note:** You may *not* have case labels with regular variables, strings, floating point literals, operations, or function calls
- If you want to execute code only in the case that you jump to, end the case with a `break` statement, otherwise execution of code will "fall through" to the next case
- **Examples**:
  - Recall this example -- an if/else structure to determine a letter grade
  - Switch Example 1 -- An attempt to convert the letter grade example into a switch. Syntactically correct, but has a logic flaw. What is it?
  - Switch Example 2 -- corrected version of Example 1
  - Switch Example 3 -- Uses a switch statement to process a menu selection, using both upper and lower case options

---

The Conditional Operator
  ```

There is a special operator known as the **conditional operator** that can be used to create short expressions that work like `if/else` statements.

- **Format:**

```
test_expression ? true_expression : false_expression
```

- **How it works:**
  - The *test_expression* is evaluated for true/false value. This is much like the test expression of an if-statement
  - If the test expression is true, the operator returns the *true_expression*
  - If the test expression is false, the operator returns the *false_expression*
- Note that this operator takes **three** operands. It is the one *ternary* operator in the C++ language
- **Example 1:**

```
cout << (x > y ? "x is greater than y" : "x is less than or equal to y");

// Note that this expression gives the same result as the following
if (x > y)
   cout << "x is greater than y";
else
   cout << "x is less than or equal to y";
```

- **Example 2:**

```
(x < 0 ? value = 10 : value = 20);

// this gives the same result as:
value = (x < 0 ? 10 : 20);

// and also gives the same result as:
if (x < 0)
   value = 10;
else
   value = 20;
```