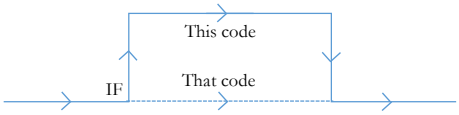
Programming Notes 5: Control Flow

As usual, more examples of the notes are in the example program.

1. Branching

The basic concept of control flow is branching, or making your code follow different paths - depending on what happens within the program. This is an essential part of programming, and you'll be using it for more or less everything in the future. For example, in a simple if statement:



2. If statements

The basic form of branching is an "if" statement, which simply evaluates an expression, and if that expression turns out to be true, it allows a branch of code to run. They're very simple to use: you type the keyword "if," followed by a Boolean expression in parenthesis, and finally the brackets defining the branch (or scope) of code that will be run. However, if there will only be one line of code under the "if," the brackets can be omitted. A Boolean expression, or statement, is anything that will evaluate to true or false. This can include simply "true" or "false," Boolean variables, functions that return Boolean values, and most commonly comparisons between values. The most basic comparison is equals to, which is performed by the comparison operator, ==. As you may have guessed, there are several other operators for other types of comparison:

== -equal to	> -greater than	>= -greater than or equal to
!= -not equal to	< -less than	=< -less than or equal to

To use the comparison operators, simply put the first variable or value on the left, and the second on the right. Examples:

```
if(selection == 'c') {
  do stuff
  do more stuff
}

if(bool)
  do one thing
```

3. AND, OR, NOT

Of course, you're not limited to one test per if statement. You can chain Boolean statements together us the "and" (&&) and "or" (| |) operators. You've probably heard of this before—the "and" operator will evaluate to true if both sides of the operator evaluate to true, and the "or" operator will evaluate to true if any of the sides evaluate to true. To use them, you must have a complete Boolean statement on each side, meaning you can't just add another comparison operator

to test if your variable is between a value (i.e. you must do var >= 5 && var <=10 rather than var >= 5 && <= 10). The 'not' (!) operator, on the other hand, is added to a single Boolean statement, and it simply, once the statement has been evaluated, flips the value from true to false or vice versa. Furthermore, while there is not a technically defined order of operators for these statements, you can remember that functionally, the not operator is evaluated first, then and, and finally or. Finally, you can add more Boolean expressions to test by simply adding more 'and's or 'or's. Examples:

```
5 == 5 && 0 <= 12     -This will evaluate to true, as both 5 == 5 and 0 <= 12 are true

(10 / 2) <= 5 || 3 % 2 == 0     -This will also evaluate to true, even though 3 % 2 does not equal 0

8 != 8 && 7 * 2 == 14     -This will evaluate to false, as one of the expressions is not true

5 == 5 && 2 – 4 <= 0 || 1 > 5   -This will evaluate to true: first, the AND is evaluated, and both statements are true, so it evaluates to true. Then, the OR is evaluated, and since the left expression was true, the statement evaluates to true even though 1 is not greater than 5.

if(variable == 5 && age >= 18 || name != "steve")
```

4. If and else

When you use an "if" statement, it will test the Boolean expression you give it, and if it is true, it will execute the code in its branch. However, if the expression is false, nothing will happen at all—your program will simply skip the if statement. Now, most of the time you're probably going to want something to happen on true, and something else to happen on false. This can be achieved through the use of if and else statements. An else statement simply runs its block of code if the "if" statement above it did not trigger. To use it, simply add the keyword "else" after an "if" statement and curly brackets (again, if there is only one statement in the block, these can be omitted). Additionally, you can add another "if" statement right onto the else block by adding the keyword "if" and another Boolean expression. This is called an else/if statement, and it will do the following: the first if statement will test, and if it fails, it will go to the second statement, and test that one instead. If the second statement also does not trigger, it will go on to the third, and so on, until it either runs out of statements or gets to an else block. Examples:

```
if(5 == 9) {
  this will not trigger
} else {
  this will trigger
}

if(variable == thing)
  this may or may not trigger
else if(variable == otherThing)
  this may trigger if the first one does not trigger
else
  this will trigger if neither of the first two do
```

5. Nesting Conditionals

You will often want to test multiple conditions, and execute some things for everything, or some things from other things, or a variety of other relatively complicated logic paths. You can do this by nesting conditional statements, meaning adding more if statements within other ones. This is very self-explanatory—you can just add another statement in another's code block. Example:

```
if(thisValue) {
  do something

  if(otherThing == 5) {
    do another thing
  }
}
```

6. Switch

With your current knowledge, to choose a code path from several different options, you'd have to write a long chain of if/else statements—but wait! There's a better way: the switch statement. The switch statement will take a character or integer value, and quite easily run a block of code corresponding with the value of the variable given to it. The syntax is a bit complicated, so here's an example:

```
switch(charVar) {
  case 'a':
    do things
  break;
  case 'b':
    do other things
  break;
}
```

So, to start, type the keyword "switch," and the variable you want to switch on in parenthesis. Then define a block of code, which is where you will put your cases. Each "case" statement defines a value that the switch statement will test for and the block of code that will be run if that case is chosen. To add a case, simply type the keyword "case," the integer or character value you want to test for, and a colon (note you don't need brackets here). Then, add the code you want to process, and end it with the keyword "break." "Break" tells the program that you are done with that case, and to be finished with the switch statement.

However, you don't technically have to include a break statement—if you don't, your code will simply continue down the switch statement, which you may or may not want. Here's an example that uses this feature to run the same code if a few different things are true (of course, if you actually use this you would have more than one possible path):

```
switch(charVar) {
  case '1':
  case 'a':
```

```
  case 'A':
    do something if character holds the characters 1, a, or A
  break;
}
```

This can also be used to do something like this, which will increment "count" different numbers of times based on which case is chosen. Basically, if case 3 is chosen, if will run all three blocks of code, whereas if one is chosen, it will only run one:

```
switch(intVar) {
  case 3:
    count++;
  case 2:
    count++;
  case 1:
    count++;
  break;
}
```

The last thing you need to know about switch statements is how to have a section run if no cases are chosen, like how an else statement would run if no if/else statements are chosen. To do this, simply add another case, but instead of using the keyword "case," use the keyword "default." Example:

```
switch(intVar) {
  case 1:
    do stuff
  break;
  default:
    this will run if none of the other cases are chosen
  break;
}
```

7. Boolean Functions

Finally, the last little subtopic is Boolean functions. Boolean functions are just like any other functions you've seen, except they have the return type "bool." This makes them very useful, as they can be used in Boolean expressions (such as in "if" statements) and will act as the value that is returned from it. A simple example (next page):

```
bool testEqual(int a, int b) {
  return a == b;
}

int main() {
  int x = 7;
  int y = 13;

  if(testEqual(x,y) && x < y) {
    cout << x << " equals " << y << endl;
  } else {
    Cout << x << " does not equal " << y << endl;
  }
}
```