## cisc1110 fall 2010 lecture V.2

- *control structures*
- looping statements



## looping statements

- you have already been using *for* loops
- today we'll talk more generally about loops
- *looping*, or *iteration*, means doing something more than once, perhaps doing something over and over and over and ... and over again
- there are times when you want your program to do something once, and there are other times when you want your program to do something more than once—without having to repeat the code again
- when you write a loop, you need to decide several things:
  - how will the program know when to stop looping?
  - will anything change about the behavior of the program each time the loop runs?
- in C++, there are two "control structures" that facilitate looping:
  - `for` : generally facilitates *counter-controlled* looping
  - `while` : generally facilitates *condition-controlled* looping



## types of loops

- controlled, non-infinite loops have an end
- loops end in two ways:
  - because they have run for a certain number of times; these are called *counter*-controlled loops
  - because a condition has changed that causes them to stop running; these are called *condition*-controlled loops



## counter-controlled loops: "for"

- a *for* loop is used when you know how many times you want something to run
- the syntax for a *for* loop is:

```
for ( <initialization> ; <termination> ; <continuation> ) {
  <body-of-for-loop>
} // end of for loop
```

- for example:

```
for ( int i=0; i < 10; i++ ) {
  cout << "hello\n";
}
```

this example will print the word `hello` on the screen ten times, each word on its own line

- the `<initialization>` is something like `i=0;`
  typically, it initializes a variable referred to as the *loop counter*;
  this variable keeps track of how many times the loop executes
- the `<termination>` is something like `i<10;`
  typically, it evaluates the loop counter to make sure it has not exceeded its maximum (i.e., the number of times the loop should run)
- the `<continuation>` is something like `i++`
  typically, it increments (or decrements) the loop counter
- it is important that something happens in the continuation statement (or the body of the loop) to change the value of the condition, eventually; otherwise you will have an infinite loop
- note that the condition can be false before the loop begins, in which case the loop will never execute!



## condition-controlled loops: "while"

- we have already used condition-controlled loops:

```
boolean q;
  q = false;
  while ( q==false ) {
    ...
  } // end of while loop
```

- the syntax for a *while* loop is:

```
while ( <condition> ) {
  <body-of-while-loop>
} // end of while loop
```

- `<condition>` is something like `q==false` or any boolean value or clause
- it is important that something happens in the body of the loop to change the value of the condition, eventually; otherwise you will have an infinite loop
- note that the condition can be false before the loop begins, in which case the loop will never execute!



## looping statements: `while`

- a `while` loop allows us to repeat things:
- example:

```
while ( y <= 10 ) {
  y = y + 1;
}
```

- as long as the condition:
  `( y <= 10 )`
  is `true`,
  the code inside the loop (in between the curly brackets ...) will execute
- you need to make sure that the condition will become `false` at some point, otherwise the code will run forever
  this is called an *infinite loop*
  generally infinite loops are BAD
- *note:* don't confuse `while` with `if` statements!
  `if` is used for *branching*; `while` is used for *repeating*



## infinite loops

- a loop that never ends is called an *infinite* loop
- an infinite loop will run as long as the program is running
- it is *unadvisable* to write infinite loops for programs that run on a computer
- in case, by mistake (!), you create an infinite loop on your computer, you can usually get the program to stop by pressing `Ctrl-C` (the control "Ctrl" key and the "C" key at the same time)
- if that doesn't work, try closing the window where the program is running
- if that doesn't work, you may have to kill the program using the TaskManager, which is invoked as follows:
  — on Windows, by pressing `Ctrl-Alt-Del`
  — on Mac, by pressing `option-apple-esc`