## cis1.5 spring2009 lecture II.3

today we are going to talk about...

- switch control structures
- random numbers
- for loops



## `switch` control structure

- a `switch` statement is useful if you are making a choice between a number of options all concerning the value of a single, simple-typed variable
- it can replace multiple `if-else-if-else...` statements and tends to look neater
- but it can only replace multiple `if-else-if-else...` statements if the variable being compared in each statement is the same variable and it is of a simple data type (e.g., `int`, `char`, `bool`, etc.)



for example:

```
char c;
...
if ( c=='F' ) {
  y = y + 1;
}
else if ( c=='B' ) {
  y = y - 1;
}
else if ( c=='Q' ) {
  q = true;
}
else {
  cout << "oops!\n";
}
```

which can be replaced with:

```
char c;
...
switch ( c ) {
  case 'F':
    y = y + 1;
    break;
  case 'B':
    y = y - 1;
    break;
  case 'Q':
    q = true;
    break;
  default:
    cout << "oops!\n";
    break;
} // end of switch
```



- note the new keywords:
  - `switch` which begins the statement and indicates the name of the variable you want to compare
  - `case` which indicates the value that you want to compare the variable to
  - `break` which ends the clause that gets executed for each matching "case", i.e., when the value of the variable matches that specified in the enclosing `case`
  - `default`, which specifies the "default" case, when the value of the variable does not match any of those in the specified cases
- note that if you don't use a `break` command, then the program control will keep going at the end of one case and go into the code for the next case
  There are times when you want this behavior, but most of the time you don't.
  Here's an example of a case when you would want this behavior:

```
switch ( c ) {
  case 'Q':
  case 'q':
    q = true;
    break;
} // end of switch
```

## random numbers

- computers can generate "random" numbers, which is like picking a number by rolling dice
- there are two steps necessary for generating random numbers:
  1. *seeding* the random number generator
  2. picking the random number
- the **seed** is used to create a sequence of "pseudo random numbers"
  *you can always get the same sequence again if you use the same seed!*
- the random numbers generated are integers (of type `int`)
- `void srand( long seed )`
  is used to seed (initialize) the random number sequence
- `int rand()`
  returns a random integer between 0 and `RAND_MAX`, where `RAND_MAX` is a constant defined by the C language
- you can *scale* the result returned from `rand()` to get a number in the range you want (e.g., $0..10$)



## random numbers: example

```
// initialize random seed
srand( time( NULL ));

// find random initial location for robot
x = rand() % 10;
y = rand() % 10;

// instead of
// x = 0;
// y = 0;
```



## loops

- last class, we introduced *while* 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:
  - `while` : generally facilitates *condition-controlled* looping
  - `for` : generally facilitates *counter-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

## 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!



## 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 ( <initialize-stmt> ; <condition> ; <continue-stmt> ) {
  <body-of-for-loop>
} // end of for loop
```

- for example:

```
int i;
for ( 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 `<initialize-statement>` 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 `<condition>` 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 `<continue-statement>` 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!



## 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 common when writing programs for robots to write infinite loops—programs that run as long as the robot is turned on
- however, it is *not* common and typically *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`