*Notes 10/13*

3 types of loops:

*while (condition)  
 statement*

*do*  
 *statement*  
*while (condition);*

*for (initialization; stay-in-loop condition; prepare-for-next-iteration)  
 statement*

*Do/While* loops always run the statement at least once because it does the statement before checking the condition. *While* and *for* statements check before running the statement, so they can be skipped over if the condition is not met. Most of the time you don’t use *do/while* because *while* works just as well and is less confusing, but there are certain conditions where you might need it.

If what you are going to do for the next iteration is really short, like the increment operator, and if it affects the state of the condition, using *for* is easier. Counting loops are easier in *for* than *while.* *While* is better if what is being changed in the loop is not related to the conditions.

You can declare and initialize a variable within the *for* loop, while in the *while* loop you must do it beforehand. In standard C++ you can’t use an initialized variable in the following statement in *for* loops, but in Visual C++ you can (Microsoft being Microsoft.)

Just declare variables above the loop conditions. It’s easier and more portable. The only benefit to limiting the scope using the Microsoft style is ease of reading for the programmer – just toss a comment on it.

To terminate an infinite loop hit Ctrl-C or just close the window. You can tell if an infinite loop is a problem if the program outputs a ton of shit, or if the cpu fan gets really loud and the program stops. If you can’t find the loop, make the program output the key variables in each iteration of the loop until a ton of stuff comes out.

The *continue* command, used within a loop, tells the program to abandon the rest of the current iteration and move on to the next. It works best when you have a nested *if* statement and no *else*, but you want the iteration to stop if the *if* statement triggers.

“N-and-a-half-times loops” have a test not at the top or the bottom, but in the middle. Example:

*for (;;)  
{  
 getline(cin, response);  
 if (response == “yes” || response == “no”)  
 break;  
 cout << “Please respond yes or no: “;  
}*

Make sure when using this type of loop you make sure the conditions are always true – to do this, you use the code *for (;;).* It’s a well-known phrase among coders to let the reader know that this loop will always run. *Break* is a piece of code that always ends the nearest loop – you use it in this case to prevent an infinite loop. *Continue* abandons the current iteration of the loop; *break* ends the entire loop.

If you want to output a certain character in a string use the command *s.at(k)*, where *s* is the name of the string and k is the # of the character (in “Hello,” H = 0, e = 1, l = 2, l = 3, o = 4). If you try to access a k value that does not exist for the string, your program will throw an exception and terminate.

*S[k]* does the same thing. You read it as “S sub K”. The difference is that using this syntax, the program will not throw an exception – it will enter undefined behavior.

The new type that these commands create is called *char* (character.) It’s a variable type for a single character -- simple enough.

C++ is weird because if you use a char when an int is expected, it will work – the computer will just convert the char to its corresponding ASCII number. It works both ways – you can initialize a char as an integer (without single quotes) and it will initialize as the corresponding ASCII character, or you can initialize an int as a quoted character and it will initialize to the corresponding ASCII number.

Don’t write code that does this though, it’s **fucking dumb as shit**.

*Functions*

Functions and subroutines break up the code and make it easier for humans to read. You use them when you repeat certain parts of the program instead of typing the same code out over and over.

When you just want your function to do something and move on, you declare it as *void*. You would declare it as *int* if you wanted to use a number it generated later on in the main routine.

*Void greet()  
{  
 for (int n=1; n <= 3; n++)  
 cout << “Hello” << endl;  
}  
  
int main()  
{  
…  
…  
greet();  
…  
greet();  
…  
}*

You have to put the functions above the main or main won’t know what it is. If you don’t want to, you can declare a “prototype” at the top ( in this case, nothing but *void greet();* ) and the compiler will be happy if you put the function at the bottom, because it will know where to look for the function in main.

You can change the parameters in the function so when you call the function in main, different outputs are possible