**Sneaky Hangman Part 1**

For the first part of the assignment I coded the hangman game in 3 files.

It reads in a text file “words.txt”

I had a Hangman.cpp which held the Hangman class and all functions needed to run the game. This also required a header file which is included.

My third file was main.cpp, this is simply used to create an instance of Hangman and call its function playGame() to start the game.

I included 5 functions in Hangman.cpp:

1. string getWord()

This function accesses a .txt file and reads all the words in the file into a vector. It then chooses one of these words at random and returns it.

1. string coverWord(string s);

This function takes a string as a parameter and returns a string of the same length that only contains “\_”

i.e. if “hello” is passed “\*\*\*\*\*” is returned.

3. bool wordCompare(const string& x, const string& y);

This function compares two strings and returns 0 if they are equal. It ignores case sensitivity.

3.

bool checkLetter(char c, string word);

This function checks to see if a letter was already read in to the program

4.

Void playGame()

This function plays the game calling appropriate functions when needed

**Sneaky Algorithm**

I will code this section in part 2 of the assignment.

The code checks the users input and for instance if the user typed the letter ‘f’ it looks to see if the family containing ’f’ is larger or smaller than the family containing words that don’t have ‘f’ . The larger family is always picked so the computer has a greater chance of winning.

If it happens that the family with “f” is picked then it compares the sub families containing “f” and choses the larger family. Then using a vector the the code splits the vector into multiple vectors where the letter ‘f’ according to where “f” is in the word.

If the family with “f” is not picked, then the code will move to a family that does not contain the letter “f”. The previous method is then used again.

By the time the user is down to 1 letter left to guess the code will have choosen the largest possible family and the user will more than likely lose.

In my sneaky.cpp file I will have two methods to enable me to implement the idea above.

One method will use the algorithm while the other will simply pick the word at the end.