In this assignment, you will write ***a hangman game*** that:

1. Display top 10 highest scores and user statistics and will keep track of how much time they are in the top 10.
2. Provides three levels of difficulty: easy, medium, and high.
3. Provide tips for the gamers.
4. Provides two different sets of users: a) the player: those who will play the game. B) the managers: those who can adapt the levels of difficulty in the game and security mechanism.
5. The information stored in files will be encoded using the Atbash Cypher.

At this time, I will set the output format flexible. However, the gamer must have the following options:

1. Play game
2. Your statistics
3. Display top 10
4. Exit game

For managers, the following options should appear:

1. Reset game
2. Change game levels
3. Add another manager
4. Set the word file
5. Exit game

# Exercise 1 (40%)

Within the folder ex1 in the zip file, you will find files called Person.h , Person.cpp, Gamer.h and Manager.h. These files provide a class to hold details of a person, and declarations for a derived class to hold details of a gamer and a manager. Such classes would generally be expected to have more data members but only those that are relevant to this assignment have been declared.

The Person class is complete, but you may, if you wish, add extra member functions to the class, e.g. an operator string method; *BUT you should not change any of the existing material, except where noted in this brief.*

The Gamer class and Manager class declaration simply provide declarations for member functions that must be implemented; *you must provide complete definitions for these functions in a file called Gamer.cpp and Manager.cpp, respectively.* The behaviour of the functions should be as specified in the comments supplied in the header file. Again you may, if you wish, add extra member functions to the class (e.g. a method to calculate the average scores); if you do so the function bodies should be written in the .cpp file.

In a separate file, you should create your main function. You should then ask the username and define what type of user he is, and direct to their respective options. Manager and users should be identifiable by a mechanism of your choice (here you can modify the existing one). It can be, as an example, name, or a given registration number. For gamers, if the name does not match to a previous name, then a new gamer is created, and a message should be output to the screen.

In the main function you will attempt to open and read from a text file, whose name should be defined in the program. Each line of that file will contain the registration number and name and scores of a Gamer (in the order of easy, medium and high); these should be passed as arguments to the Gamer constructor to create a new gamer object, which should be added to a collection of gamers. This collection may be a vector, a list or a set (from the standard template library).

The main function should next attempt to open and read from another text file, whose name should again be established in the program. Each line of this file will contain a registration number and a manager name. You should create an object of the class Manager that holds who will be the Manager this time. Only an existing manager should add another manager, and the first manager should be you.

For the hangman game itself, you are shown a set of blank letters that match a word, and you must guess what these letters are to reveal the hidden word. You guess by typing letters are inputs – as you should check if the user attempts to input a word. If you pick a letter that is in the word, a letter is revealed from the blank letters; however, if you pick a letter that is not in the word, then a stickman is slowly drawn. With each wrong letter guess, the man is drawn more and more. When the man is finished, he is hung, and the game is lost. Therefore, the game is called 'Hangman'. If you can reveal all the letters in the word before the man is hung then you are successful, and the full word is revealed along with an image showing the meaning of the word.

The following example game illustrates a gamer trying to guess the word *hangman* using a strategy based solely on letter frequency. As the player continues, a part of the stick figure on the noose is added. Once a full body is drawn, the game is over, and the player lost. Diagram

Description automatically generated with medium confidence

|  |
| --- |
| The guessing player has lost this game as the diagram had been completed before all the letters were guessed. *You are not required to display the stickman as shown; you may choose how to display the word guess and misses but you must inform the user the state of the stickman somehow.* |

For the levels of difficulty, the following rules apply

*Easy -> words with size between 1-5*

*Medium -> words with size between 6-10*

*Hard -> words with size 10 and above*

These levels must be asked to the gamer right before the game itself starts (whether they want to play on the Easy, Medium or Hard modes).

*Obtaining words will be explained in exercise 2.*

The scores of the game should be defined by you. However, you will have two scores: one that is per game, but also an accumulated score. The accumulated score is the sum of all scores a gamer has obtained.

The manager can then select to change the range of each level of difficulty. They can change either the minimum or/and the maximum values. The minimum and maximum should be displayed to the user when asking for a guess.

For the top 10 options, you must create an additional function in the main file. The function should have as arguments a registration number (of type string) and should print on the screen the highest top 10 scores for each difficulty level, but the output must highlight the score from that registration number. How you highlight the score is up to you. This will make sure we know where the user is in the top 10 if it really is in there.

The main function must be controlled by a loop that reads the options from the user and guides the actions in the program itself. The loop must end when the user selects the “exit” option.

# Exercise 2 (20%)

The game must provide tips to the user so that the word is guesses correctly. This tip consists of showing the word in a phrase, but of course, omitting the word. For example if a word to be guesses is LOVE the tip should look like

*Tip: “And Maria knew why her family helped her so much, it was because they XXXX her”*

If a gamer has already guessed a letter the tip should display such letters in the word. Example, if the gamer has guessed L and E, the tip should look like

*Tip: “And Maria knew why her family helped her so much, it was because they LXXE her”*

If there are more than one letter in the tip, it also should be treated in the same manner as explained above.

The program should read the name of the text file to be used that contains words and phrases. The program should terminate cleanly with an appropriate error message if any of the files cannot be opened. A file containing a book is found in the zip file. You may use this book to get words randomly but also to get the phrases containing the words. All you need to do is search in the book file. This will make your life way easier. However, you may produce your own file if you wish.

For this assignment a “word” should be regarded as being a sequence of non-white-space characters containing more than one letter. Any punctuation at the beginning or end of a word should be removed (note that there may be more than once such punctuation symbol); punctuation symbols in the middle of a word and any digits are to be regarded as part of the word. You must use the function ispunct, defined in the header file cctype, to check if a character is punctuation. All input should be converted to lower case, so ”IT”, ”It”, and ”it.” should all be converted to ”it”.

Your program must contain a class called ReadWords, each with a .cpp and a .h file. The header file for the ReadWords class and a partial implementation are provided in the ex2 folder in the zip file; you must use these and must not make any changes to the header file other than the required changes to the comments. Changes to the provided code in the the cpp file are permitted.

You must then randomly select a word based on their size values that define the game level, in the sample file. Store words in the array named words. Once you know that word, you must search for phrases containing those words and store them into the phrases array of the ReadWords class. You must select a total of 10 words and 5 phrases.

It is recommended that you use the array words when searching for phrases; a simple input loop like that used in lab 3 (but inputting strings instead of numbers) will be sufficient for this – but you must extend it so that phrases can be also obtained.

# Exercise 3 (20%)

The top 10 should show the date when the user play and how many days they are in the top 10. In this way we are not also tracking efficiency gamers but consistent players as well. In the top 10 you should only show accumulated scores. For example, the output of the top 10 option should look something similar to:

*Top 10*

*1 – Rick Morris | Easy score: 10 | Medium score: 30 | Hard score: 30 – 145 days since the first appearance in the top 10. First time: 20/05/2019*

The Date.h file in the reassessment folder of the zip file is a header file for a Date class similar to the Time class from week 3 lecture videos and slides. You are required to implement this class in a Date.cpp file.

The isLeapYear function should return a Boolean value indicating whether the year held in the object to which it is applied is a leap year. (A year is normally a leap year if it is a multiple of 4, but years that are multiples of 100 are leap years only if they are divisible by 400, so 2000 was a leap year but 2100 will not be.) Use that function to provide the required application output.

You must implement one function overriding the operator <<. The << operator should output the date using the format 02/11/2019, always using 2, 2 and 4 digits, with leading zeroes where necessary.

The operator string function should return a string object (not a C-string or character array) holding a string representation of the date using the format “Month, Day and Year”, always using a string, 2 and 4 digits, with leading zeroes where necessary. The comments in the header file indicate how string objects can be built by converting numbers to strings and using concatenation. (The to\_string function is declared in the <string> header file; it is not a member of any class.)

This operator is invoked whenever a declaration of the form string s(d); is used when d is a Date object, and also if string(d) is used anywhere in an expression, so to test it you could use a statement such as cout << string(bday);.

# Exercise 4 (20%)

We will now improve the game of the exercise 1 and add security mechanisms to secure the identity and scores of the gamers and managers. This will ensure that no one can just alter the text files and damage the functioning of the game. For that you must implement a new class derived from an abstract class “encryption” containing functions such as encrypt and decrypt. Then, you must use the encryption when storing objects in the file and decryption when you read the objects. You must implement the Atbash Cypher, details will be provided bellow.

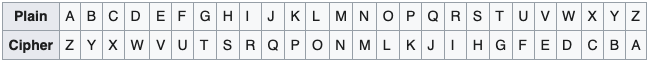
There is a header file for an abstract class called Encryption, to be used a base class for developing a new derived class. It contains a pure virtual function called isEncrypted which is intended to be used to check if a message is encrypted or not. Implementations of this in the new derived class should take a gamer name from the file as an argument and return true if the word satisfies the criterion. The derived classes should be written to override the virtual function.

The header also contains the constructor, where you inform the key to the encryption. Also, it contains the encryption and decryption functions, that must be implemented. You must implement this function in a new Encryption.cpp file.

The new derived class must be written in separate .cpp files with an associated .h file. Implement the Atbash Cypher as a new derived class.

In the main function, you must add precisely the encryption and decryption functions in the appropriate places to make all text files encrypted. You may also create a function in the encryption library, where it checks if the file is encrypted, and if not, it encrypts all the information.

The Atbash Cypher works very simply. The letter of a word must be changed by another one from the inverse alphabetic order. For example, you should consider this alphabet



Number should follow the same method.

# Deliverables

You should submit a single zip file containing all exercises. The only file formats acceptable are .zip , .7z or a linux gzipped tar file. If you submit a file in any other format you will lose 10% of the marks for this assignment.

Both folders should contain a MAKE file, necessary for the compilation of the application. Also, you need to include all files necessary to make your program executable.

***Files generated by the compiler, such as .o and .exe files, should not be submitted.***

# Marking Scheme

10% of the marks for this assignment will be awarded for programming style, 10% for efficiency and 10% for structure/comments.

Issues that will may lose marks for efficiency could include unnecessary copying of objects, unnecessary repetition of code. Your comments should say precisely what each function does and what each non-local variable represents; within function bodies you should use only brief comments to say what groups of lines do – you must not say what every statement does.

The total mark available for style and comments will be proportional to the amount of the assignment that you have attempted, e.g. if you attempt only about one exercise the mark available for style and comments would be about 10%.

Of the remaining 70%, 40% of those goes to exercise 1, and 20% for each of the other exercises.

I must compile the code using the MAKE file and the execute the application for testing. Marks will be deducted if:

1. The MAKE file does not compile the application properly. (5 marks – 2.5 each exercise)
2. The executable file does not pass the functioning test. (15 marks – 7.5 each exercise)
3. The implementation of the application does not follow specified instructions. (remainder of the marks)

*Needless to say, but: use the STL as much as you like but wisely. You should use as much as you like!*