Computer science – personal data manipulation

Abstract

The program ftp account editor is used to keep in an external file the domain name of the ftp account, the user name and the password. Three strings of characters that do not contain spaces since the structure of all these is “ spaceless” by default. The program uses a single linked list which’s nodes is having as attribute a class. Various methods are implemented giving the basic functionality to the user. The main aspects in which the software is produced and designed are modularity and reusability. The target goal is achieved not in the desirable level but in a satisfactory one if we consider the low level programming skills of the programmer.

Table of Contents

main. cpp 21

Classes, methods and attributes

Three classes consist the whole program. With an inside-out approach this report explains the functionality and the use of their methods and attributes. The main target goal was to achieve the productions of software as modular and reusable as possible. But due to the low level programming skills of the programmer the above requirements are met not in the desirable but in a quite satisfactory level. The inside-out approach begins with a graphical representation of the classes which is not a UML diagram although it also represents the connections between the classes.

Class ftp

We start as well with the description of the class ftp. By the name we can understand that the objects of this class are responsible to keep the data of each ftp account. Each one object of them is " included" as an attribute to the corresponding Node. Of course we could have the same attributes in each Node as well with the methods but then we lose the true meaning of modularity. Static character arrays are used for all attributes since the domain names as long with the usernames and the passwords do not have spaces by default. The sizes cannot be changed dynamically because some method is not implemented although it's feasible. The constructor of this class does not do anything because of when a new object is instantiated the new values for its attributes are received as parameters in the responsible method of the liked list through a temporary object created in the main function which's values are assigned using the setData methods (setDomain, setUserName, setPassword). The assignment operator is overloaded thus to copy the attributes of an object to another one like using integers (a= b;). The equality and the inequality operators are overloaded so as to compare two objects of class ftp again like using integers (if (a= = b)) and is used by the list’s methods whenever the list is traversing the nodes until it finds an ftp instance that its attribute’s values match the requirements.

The printData method uses the standard output (cout) to print the attributes. Of course we could overload the output operator but this is further discussed in the extra methods section. Also all the setData methods receive as parameter a pointer to an array of characters and then copy the string in the corresponding attribute (strcpy(attribute_, _tmp_attribute)). Finally the getData methods do not receive any parameters and do not carry any operations than to return the corresponding attribute. (getDomain returns domain).

Class Node

The Class Node is the smallest of the three in lines of code although it’s the key idea of the link list data structure. Though it’s important to be as simple as possible to achieve modularity. With the way it’s implemented it is the same as long as the most methods of the linked list. No matter what kind of data we use (integers, characters, structs, objects etc). Simply we have to overload the operators of the data class with another implementation. The specific class has as attributes the data instance and a pointer to an object of data type Node. The public part Node has it’s constructor who initializes it’s next pointer to NULL and a the friend class UrlList so as enforce the list to alter it’s next pointer values easily just like if it was public.

Class UrlList

The last class used for the implementation of this software is responsible for the usability of the afore mentioned classes. Although the Node is the key idea of the linked list this one is actually the implementation of the linked list. It’s only attributes are the head of the tail pointing to the first Node and the tail which point to the last Node of the list. The length can be used either to access a specific Node by indexing (i. e. Node [3]) my simply overloading the[ ] operator or to check if the list is empty (if(length= 0)). Though in this software it is not used to anything but it’s there and it’s increased/decreased every time a Node is inserted or deleted. If somebody wants to use it thus in a new method the length already exists.

The public part of the UrlList class contains only methods as usual. Starting with the constructor we see it initializing the head and the tail of the list to NULL since when the list is instantiated it’s empty. This is used to the isEmpty method which checks if the head points where the tail does to NULL and returns true. After in the implementation we meet the print and the checkIfExists methods. Actually the are quite similar since the first traverses the list and in each loop it calls the print method of the corresponding data object while the second one receives as parameter an object and traverses the list until the ends and calls the print method of every data object that matches the parametrically passed one (if (Node-> data= = temp)).

As well as the above two methods are similar the insert and append methods are identically the same except two lines of code where the insert puts the newNode as head of the list and the append as the tail of the list. It receives as a parameter the same object as all the above that receive object parametrically and assigns to the newNode created dynamically the values of the temporary (newNode-> Data= temp;). After that it checks if the list is Empty and if true is returned the newNode is both the head and the tail of the list. Else depending on if method is the insert or the append it follows the afore mentioned procedure.

Coming to the file manipulation there are two methods that make it feasible. The intro(_) used when the program starts running and it has to load the data line by line from the file to the Nodes of the list. As well the exodus(_) method outputs the data of all the Nodes in the external file before the user terminates the program. In order to explain the way that the methods operate it’s better to start with the exodus method because it actually operates before the intro method (to load something there must be something saved in the file). First in the exodus then we create an output file stream instance and we connect it to the external file. If this already exists it’s opened for output and connected to the output steam out_file. If it does not exist a new one is created. Then we traverse the list and output the returns of each data instance getData methods with the end line operator. So every attribute is saved alone in a single line thus in the file we may have a structure like the following

The Intro method is quite more complicated. First of all it creates three variables same type as the attributes if the ftp class to hold the values we will from the file this is why their names are quite the same adding ‘ _file’ to the end. Also we need a character variable to check the eof character of the file. We create an input file stream instance and we connect it to the external file. After that we check if the file exists (some compilers like MS visual C++ create a file with the specified name). For that reason we check then if the first character is the eof. If yes then there is nothing to read and exit. Then while the eof is not the next character we create a new node which we initialize it’s attributes by assigning the values we read from each line to the corresponding afore mentioned variable and then call the node’s setData methods passing the ‘ _file’ variables as parameters. Then we append the newNode to the list.

Known bugs & possible solutions

In part of the report some known bugs are reported along with some possible solutions whenever they are known but not implemented for some reasons. The first known bug is founded in the display of the menu and generally in the main program. The final output messages should be displayed before the screen clears and the menu is displayed again. For unknown reasons even if the system(“ cls”) command is before the display_menu function the program insists on clearing the screen then finish with it’s operation although it’s in a loop and after display the menu. These are basic programming skills that are mastered by the programmer but no matter what he tried (the clear screen was in the end of case ‘ x’ but it still was doing the same thing, or in the appropriate list method) this was not working. If though it was working then we should place the system(“ pause”) command before the clear screen so as the user could see the result for as long as he wants and then continue.

Next one in the intro( ) method of the linked list. It appends an extra node in the linked list that its data values are nothing (probably the newline character). This is solved if we remove the ‘ endl’ from the third out_file operation of the exodus method. But when we removed it there was another bug in the data as shown bellow

Also we could use a garbage array of characters where we could assign the problematic newline character but this as well was producing the same bug as above even if the logic seems correct. Finally a more off-beat solution could be to delete the last Node exactly after the execution of the intro method. Although the method is implemented as shown in the extra methods section it never managed to work with the rest of the software and was crashing immediately so we suppose that a logical error is hidden somewhere but it could not be debugged.

The last known bug is in the checkIfExists method which asks from the user to create a temporary ftp instance and compare it with all the others in the list and print all those who match the criteria (if(strcmp(domain, temp. domain)&&! strcmp(usrname, temp. usrname)&&! strcmp(password, temp. password))). There should be a way though to print a message of there in no node matching even if the list is not empty. Thus we have the check if(tail->data!= temp) cout <<“ Such record does not existn”;) This says that if the whole list is traversed and the last node does not match then there is no such record. It was working perfect with two nodes, but with more it outputs the message every time, since there could be a node in the list matching and the tail would not much.

Extra methods

Before presenting the source codes of the software some extra methods are reported where most of them were implemented but they were crashing for unknown reasons thus they were never debugged. Those methods would ‘ increase’ the modularity or the functionality of the software while some could solve some bugs. First is the input-output operator overloading. This would increase definitely the software’s modularity and reusability since we could use them is many other methods as explained bellow after the source codes of those semi-implemented methods.

We could use the output operator very simply by calling the object as variable (i. e. cout < data;) This could be used in all cases where we want to print the attributes or to output them in a file (i. e. out_file < data <

After the input/output operators we could also solve the bug in the ifstream operation by deleting the last node as afore mentioned. The source code of this is given bellow. This as well never worked normally for unknown reasons and was crashing the whole program.

Two more methods are those that would be used in the encryption/decryption of the data during the exodus/intro methods execution correspondingly. Those methods very simple and identically the same but they both never worked correctly as they were crashing the whole program even if there were no compiling errors or warnings.

Finally a method that was partial implemented is the insertInPlace method which would insert a new Node is a place according to specific criteria. The problem in the implementation of this method is that the programmer could not find any criteria (i. e. alphabetical order does not work since all domain names start with ‘ www’). As well imagine that we have found something and we have located the correct position in the list by traversing it from the head to the desired node. So we could simply change the next pointers by those two steps (the tmp pointer points to the node that will be before the newNode after the insertion) newNode-> next= tmp-> next;