ICARUS SYSTEMS

SECURE. ELEGENT. EFFICIENT.

Project report. 7 January 2019

Sardar Hassan - 223488
Syed Usama Minhaj - 227760
Hamza Atta - 221587

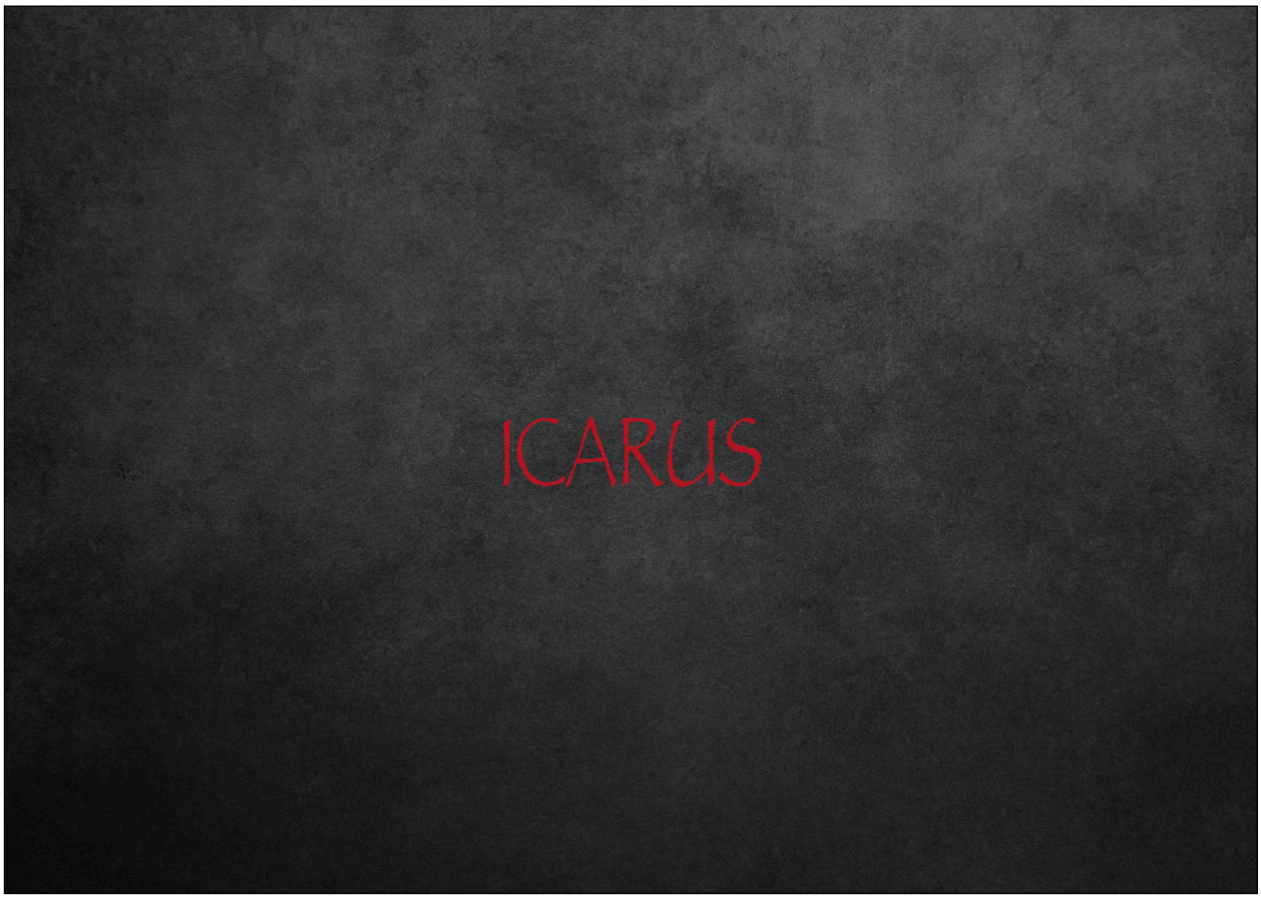


Table of contents

Acknowledgments

We would like to thank Allah (SWT) for the ability and opportunity to attend NUST and for the patience required to create this project. All of this would not have been possible without His will.

We would also like to thank SFML for creating such an exquisite library and for providing information on all classes and their member functions.

Special thanks is also due to Khan Moeen Danish for his advice on the colour scheme of the GUI.

ABSTRACT:

This report presents the implementation and working of a complex and efficient management system of a bank using advanced data structures along with advanced queue management. The management system also supports secure permanent storage of data. The implementation is done on C++ but can be extended to any other language. This report also presents the Graphical User Interface implementation of the project using **SFML** as graphics library.

LANGUAGE USED:

The whole project is made using C++ as core programming language. Graphic User Interface is made on **SFML.**

Part 1: Back End Processing

By: Usama Minhaj

IDE USED: Visual Studio

DATA STRUCTURES USED:

- Heaps:

  Used in the implementation of priority queue. Heaps are the best data structures for the implementation priority queue, so its choice was natural. They have insertion and Extract Min time complexity of only **O(logn).** They are also memory efficient as they are implemented using arrays.

- Queues:

  They are used along with heap for implementation of priority queue.

- Hashing:

  Hashing is used for the storage of data of an account. It is one of the fastest and secure data management technique. Insertion, deletion and searching can be done only in **O(1)** in average case while **O(n)** in worst case. It is very efficient in terms of time and as well as in terms of memory. Data is stored in arrays and very small arrays can be used to hold large amount of data using collision.

- Binary Search Tree:

  Hashing become very inefficient if load factor is very great or a lot of collision has happened. In order to resolve this issue, we used Binary Search Tree which could insert and delete in only **O(logn).** So even if look at the worst-case scenario then its complexity become the time complexity of binary search tree. So, in this way our program becomes very efficient even in case of worst case.

MAIN COMPONENTS OF THE PROJECT:

PRIORITY QUEUE:

For most banks, queue management is one of the important tasks. Normal queues are very simple, but they are also unjust in some ways. So we employed a priority queue with following options with decreasing priority order:

1. VIP

2. Old
3. Woman
4. Transgender
5. Man

So when a person enters the bank, he is assigned a code and asked his appropriate priority. Based on the combination of priority and order of code assigned to him he is called using his code.

DATA MANAGEMENT:

Main algorithm for storage of data is hashing. The size of hash array is **100.** Hashing function used is **folding.** So in this way we can guarantee both fast and secure storage of data as hashing is one of fastest algorithm with average case of main operation being only **O(1).**

The main disadvantage of hashing is in case of collisions. When the collision becomes very frequent then hashing becomes very inefficient. So, to counter this problem we used Binary Search Tree as an algorithm for **separate chaining.** So even in case of frequent collisions, Binary Search Trees would make sure that efficiency is not compromised. We have also used an advanced algorithm for **Account Number Generation** so make sure that we have a **Balanced Binary Search Tree**.

STORAGE AND SECURITY:

As we have created a bank management system, so secure management of data become very crucial. For security, first we have a **master password** which ensures that only an admin can log in to this management system and have access to all its features. The password is entered in password case i.e it is not visible on screen. **Folding** is used as method of hashing which ensures secure storage of data. When this data is stored in external file, even then it is not stored insecurely. Name and password are secured through **ciphering** while account number is encrypted through **modular exponentiation**. In this way our program becomes highly secure.

Part 2: Front End Processing

By: Sardar Hassan Arfan Khan

IDE USED: XCODE

Description:

We wanted to develop a GUI using C++ that would be fast, intuitive, and easy to use. Thus we decided to go with the Simple and Fast Media Library (SFML). This library enables us to display anything we want on an output window the opens along with usual IDE output window. It allowed us to customise everything from the text to the buttons. SFML not only enabled us to link pictures to our application, and use them as our start up, and background pictures, but it also allowed us to link different fonts, and music. All of the files linked were stored on the final application bundle.

Implementation:

I used an object orientated approach in our project. I created one class called graphics. Every variable and object (of other classes) created in this our GUI is used as public in our class to allow for different interactions between header files. All our member function are also declared as public.

I have created a default constructor in my program inside which every variable's attribute's (such as text size, input box colour etc) are set. Thus, every attribute is set right after program running.

This OOP approach also improves security of the program, and allows for data encapsulation though no such option is implemented yet. A text file has also been created to store the username and password of every employee who wishes to access ICARUS systems. The data and password is read from the file sequentially until they are matched with the ones entered. If no such match is made, the system denies access, informs the user, and prompts him/her for the correct username a password.

At the start of the program, the start up page is displayed for a fixed time of 5 seconds, and that time is measured using the clock feature in C++ 17.

Furthermore, when ever the back to home pages button or back to main page button is clicked, all the variables are set to their original values. This entire program is run and managed using about 2 dozen boolean variables, each corresponding to a specific event. When ever a button is pressed or a value is enter or a new a page is loaded, the corresponding boolean variable(s) is set to 1.

Below are the list of member functions of class graphics:

```
void startup_verification(sf::RenderWindow &window, sf::Event
event);
int add_to_queue(sf::RenderWindow &window, sf:: Event event);
void make_transaction(sf::RenderWindow &window,  sf:: Event event);
void show_balance(sf::RenderWindow &window,  sf:: Event event);
void credit(sf::RenderWindow &window,  sf:: Event event);
void debit(sf::RenderWindow &window,  sf:: Event event);
void process_queue(sf::RenderWindow &window,  sf:: Event event);
void add_account(sf::RenderWindow &window,  sf:: Event event);
void delete_account(sf::RenderWindow &window,  sf:: Event event);
void draw(sf::RenderWindow &window,  sf:: Event event);
void loadstartup(sf::RenderWindow &window,  sf:: Event event);
void size_of_queue(sf::RenderWindow &window,  sf:: Event event);
void Save();
```

Note: The constructor is not shown.
Note: The member attributes are not shown as they are arbitrary and many.
Note: The names of member functions are self explanatory.

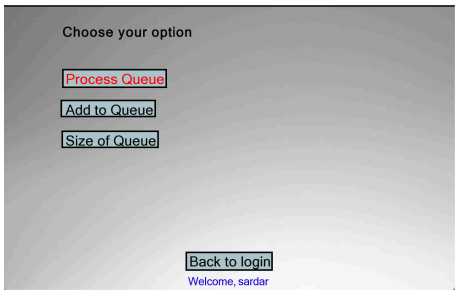
LIBRARIES USED:

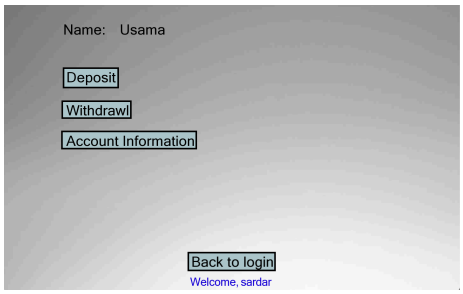
- stdio.h
- iostream
- SFML/Graphics.hpp
- SFML/Audio.hpp
- cmath
- fstream
- String
- sstream

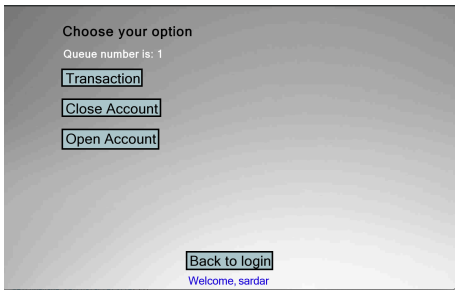
Salient features:

- Light blue colour of data input box when data is not entered.
- Dark grey colour of data input box once data has been entered.
- Buttons light up to red colour once the mouse in hovered over them.
- Welcome message with user's name at the bottom of every page.

- Bank account holder's name displayed when ever account is accessed.
- Multiple pages, each designated for a special purpose.
- Different backgrounds to create aesthetic environment.
- Data hiding and encapsulation.
- Back to home page, and back to main page buttons.
- Displays customer's queue number when processing queue.







Problems faced:

The core problem faced in the development of the GUI was that everything that to be developed from the ground up. There was no template of any sort nor was there any preset attributes of the different classes used (text class, image class, sprite class etc). I had to set and calculate every attribute of every object of every class we used. For example: when we wanted to display a text, I had to initialise the object of the TEXT class, then I had to set the font, the colour, the character size, the location of the beginning of the text, and so on. What made things even more difficult, and cumbersome was that we wanted to make everything dynamic (and we did.) This complicated things as formulas had to be derived and functions of the TEXT class like 'transaction.getGlobalBounds().top' had to be used.

Another problem that I faced was that there is no function or capability in SFML to halt all operations until the next page is loaded. For example, if we press a button on a page that loads another page, then if there is a button in the same place as in the previous page, then that button is also pressed.

In SFML, there is no capability to recognise, and allow input of integer, floats, double or boolean data type. Very input to the program had to be taken as a string. It was then converted to the required data type. Moreover, SFML does not allow the use of back space. So usage of the key will not remove the most recently entered key, rather it will store it (back space) in the variable.

Part 3: Linking Part 1 and Part 2

By: Sardar Hassan Arfan Khan

Team members involved:

1. **Sardar Hassan Arfan Khan**
2. **Usama Minhaj**
3. **Hamza Atta**

IDE used: XCODE

Implementation:

This task was potentially one the most difficult and time demanding. We created multiple header files each containing source code for a specific function. The header files of the back end working were included in the header file of the front end working.

We then declared objects of different classes contained in the multiple header files in the default constructor of the graphics class in the gui header file. We then included the gui header file to our main function which allows all back end and front end header files to be linked to our main file. An object of the graphics class was then declared in our main, and then a while loop is run, containing member function calls of our graphics class.

Note: gui header file only contains graphics class.

List of header files:

- headergui.h
- Headertextfile.h
- Database.h
- Priority_Queue.h
- Account.h

List of cpp files:

- Account.cpp
- Database.cpp

Result

After combining all the above components, the result was a very secure and efficient banking management system with a beautiful and user-friendly graphical user interface. It has all the components required for a great banking management system. The back-end processing works flawlessly, and what's even more amazing is the efficiency of our application. It has advanced queue management with priority given to each customer as it enters the bank. Our application is highly efficient due to the combination of hashing and binary search tree. The time complexity table with comparison is:

Type	Average Case	Worst Case
Hash Table	**O(1)**	**O(n)**
BST	**O(log n)**	**O(n)**
Our Project	**O(1)**	**O(log n)**

So the worst case is nearly **O(log n)** due to usage of BST in the hash table. This make our program workable even under heavy conditions.

We also managed to make our program extremely secure. It has master password which makes sure that only admin can login. Account details are stored in stored in external memory after encryption. Folding is used to make the hashing even more secure. So, these techniques make sure that data is saved from anyone who wishes to cause harm.

Conclusion

During the making of this project, a lot of things were learned. We learned the importance of teamwork and how it could make or break a project. Such an extensive program (4000 lines) could not have been written by one programmer, certainly not with the given deadline. This project exposed us to real life conditions faced in corporations and exposed us to only certainty when working on a big project; teamwork will get you through.

Also, we are of the opinion that the position of team leader is a significant one that needs to be filled at all cost. Without one, there is a certain air of undue subordination in the group. Fights and indecisiveness amongst the team members is an inevitability in the absence of one. Thus, it is vital for a good project that a team leader is chosen.

Even after trying our best, however, there are still some improvements which can be made. C++ is very unpopular for its graphics. We can use other languages like JAVA to make a better GUI in less lines of code. Another improvement may be made on the security of the application. Basic Encryption is used in this application which may be replaced by advanced encryption algorithms coming nowadays.