***Software for Basic Banking***

-Lab Batch 75

Keeping in mind the large amount of data that the banks have to deal with, we decided to work on software which would help modify, update and store this data in a readable text format.

Modus Operandi:

We divided our batch of 13 into 2 teams of 4 and one of 5(which was required). Skeleton for the programme was laid out where the services to be offered by the software were decided upon.

Members from all the three teams got together to code the “main ()” function, so that each team would be familiar with the variables and functions for further coding.

1. A structure named “acc\_details” was globally declared. This included details like the name and address of an account holder (as a character array), the age (as an integer) and the balance (as a float type variable).

Account Holder

1. The various facilities offered by the bank would be done so with the help of functions, which were again, globally declared. These included
   1. Creation of account
   2. Deposition of funds
   3. Withdrawal of funds
   4. Balance inquiry
   5. Deletion of account

3. The file containing the customer information was named “records.txt”.

To facilitate coordination, a thread on Gmail was started where everyone was kept informed of the additions and editions in the program.

Process and Algorithm

We started out by accessing the data directly from the text file to perform various functions. This approach proved to be cumbersome for extraction of a particular record. A lot of conversions from string to another data type were required at every step. For example to assign a unique account number to a new customer, the account number of the last record in the existing file would be incremented by one. For this, the account number read from the data file was to be converted to an integer data type using the “atoi” function, in the “cstdlib” header file.

To avoid this, another method was devised. According to this, data from “records.txt” would be read, in a particular format, into the structure array “acc\_holder[]” at the start of the program(using “fscanf”). This made extraction and modification of a record a lot easier. Following the same format as that of reading, the modified data would be written to “records.txt” at the end of the program, using “fprintf”.

In between the reading and writing the various operations would actually be performed on the structure array “acc\_holder[]”. Certain regulations were put in some functions, like in creation and withdrawal the balance at the end of the process was to be more than Rs.500/-.

For the first submission we had a smoothly running software providing all the aforementioned facilities and a password check for the person logging in. The program would be terminated if the username and password were entered incorrectly for the third consecutive time.

Start

Password

Check

No

Try=Try+1

Yes

Yes

Transfer of data from “records.txt” to the structure “acc\_holder”

Try<3

Options

1. Create account 2.deposit 3.withdraw 4.delete account 5.balance inquiry 6.exit

No 6

Transfer of data form structure “acc\_holder” to the data file “records.txt”.

End

1 2 3 4

5. Balance inquiry

4. Delete account

3. Withdraw

2. Deposit

1. Create Account

The addition made to the above facilities in the second half of our project were that of transfer of funds from one account to another in the same bank (the regulation of having a minimum of Rs.500/- after every transaction was enforced here, too)

Another data file, named “employee.txt”, containing the usernames and passwords of all the bank employees was created. Responsibilities of creating and deleting an account were to be handled by the manager himself. To practice caution, the option of changing username and password were generated for the employees. The “finish” function, which wrote the updated data back into “records.txt” was renamed “refresh” and could be called anytime while on the main menu.

Interests were awarded on the sum deposited. This interest would be compounded per second for the time it was invested with the bank. To ensure this the “cinterest” function was called at the beginning of every function, when the entered account number was found to be valid. The calculation of the length of time for which the money was kept in the bank was done using the “ctime” header file. Another variable, named “domodi” (a character array) was added to “acc\_holder” array structure. This character array was converted to a time structure using the inbuilt “strptime” function which was in turn converted to a time length (in seconds) using the inbuilt “mktime” function. The date and time of modification would be updated and written in character string format using the “strftime” function.

Considerable amount of time was spent on better presenting the software and making it more user-friendly. Even more of time and effort went into documentation and description of all the functions. A short introduction on the project together with the names of all batch members was well documented in the program itself. This made the program much easier to read and understand the first time on.

What we weren’t able to do was to come up with was a better algorithm for interest calculation. This required recording all the transactions for a month together with the date and time when they took place. This in turn asked for different data files for each account holder. And accessing them was the difficulty we failed to overcome.

This project is the outcome of the joint efforts of Labbatch 75. It includes about 1100 lines of coded language with about 200 lines of documentation all of which took us about 434.5hrs to complete.