**Assignment – InterBanking Pty**

**Introduction**

A new Internet Bank called InterBanking Pty has given you the opportunity to develop their next generation customer and account management system. We will be guided through the design, coding and testing of the required application.

At each stage of this project, instructions will be given as to what is required and what should be implemented and submitted for review.

The main task of this assignment is to produce 09 small programs (or components) their iterative integration to build the required customer and account management system.

Further specifications will be released as and when required, and will be clearly marked.

**Stage 01**

**Task:**

1. You are required to produce a model of the customer account creation component of your

system. For this you can use any modelling notation you are familiar with including ERD,

OOD, etc.

2. You need to implement a C++ console-based application that performs the following

functionality including:

* Create an application that enables registered/authorized users to log in and create a

new customer account and **BankAccount**. Use four-digit account numbers, but the

program does not have to enforce this rule.

* Develop a **struct** so that it contains public data fields that hold the **int** account

number and **double** account balance and customer name and password.

* Write a **main()** function that declares a **BankAccount** and allows you to enter data

values for the **Bank Account** fields.

* Echo the input – displays on screen the input details.
* Use control constructs studied in lecture to develop a logging in part of the system,

which will prompt users for their account numbers and name to enable them access to

the program functionality such as viewing their bank balance.

**Assumptions:**

* Creating a BankAccount would simultaneously create a Customer Account.

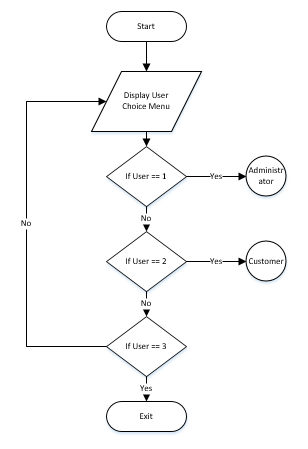
**Flowchart:**

Figure 1: Main Flow Chart

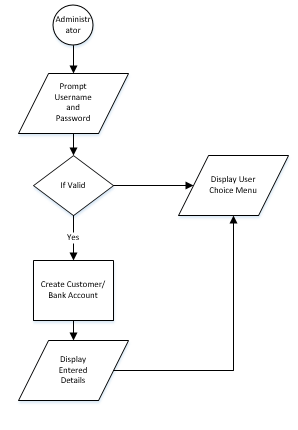


Figure 2: Administrator

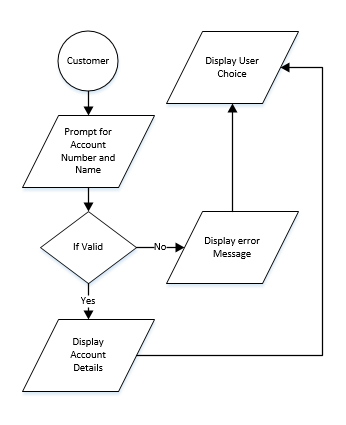
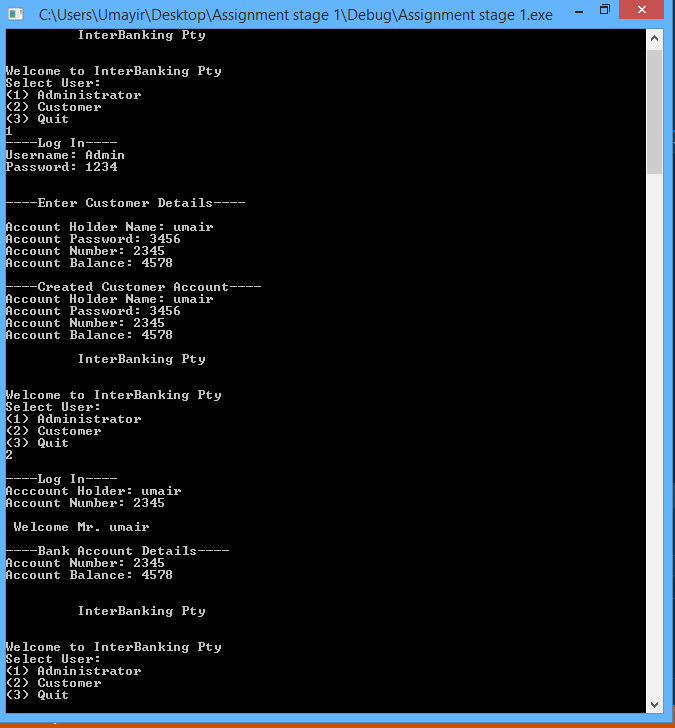


Figure 3: Customer

**Screenshots:**



**Stage 02**

**Task:**

Using the same structure created in **stage 1**, write a new (or amended) **main()** function,

which you create two or more **BankAccount** objects for a given customer. Start with only

two bank accounts per user. Add required statements to the **main()** function to do the

following:

* Prompt the user for account numbers and beginning balances for his/her

BankAccounts.

* For data entry validation, the bank account number should be between 1000 and

9999; issue an error message if the user does not enter a valid account number.

The balance should not be negative; issue an error message if it is. Continue with

the program only if the account numbers and balances are valid for both.

* Display the full account numbers and starting balances for the two accounts.
* Prompt the user for a dollar amount to be transferred from the first account to the

second account. Issue an error message if the two accounts have the same

account number, and do not carry out the transfer of funds.

* Issue an error message if the requested transfer causes the first account to fall

below $0.00, and do not make the transfer.

* Issue a warning message if the transfer causes the balance in the first account to

drop below $10.00, but make the transfer.

* Issue a warning message if the transfer causes the balance in the second account

to be greater than $100,000.00, which is the highest amount that is federally

insured.

* Display the ending balances after the transfer amount has been deducted from the

first account and added to the second.

* For now, when an error message is issued, the program should terminate

gracefully. For warning messages, the program continues to execute.

In this section you should whenever possible demonstrate in your code the use of loops,

array and string processing, etc.

**Assumptions:**

* Creating a BankAccount would simultaneously create a Customer Account.
* Customer does not have a password, only the Administrator does.
* One Customer can have only one account of each type (savings or current).
* Transactions occur only between the savings and current accounts of one Customer.

**Flowchart:**

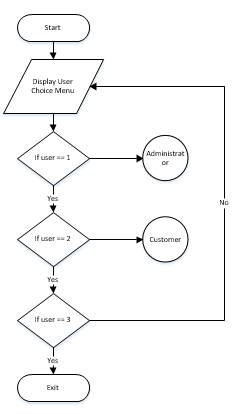
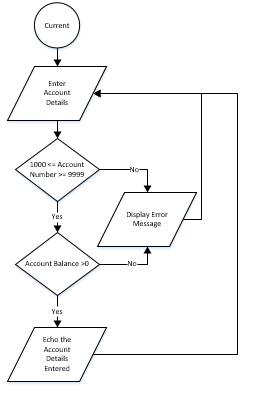
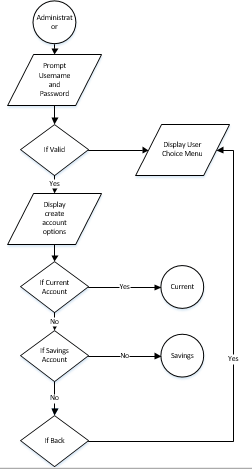
****

Figure 4: Main Flow Chart





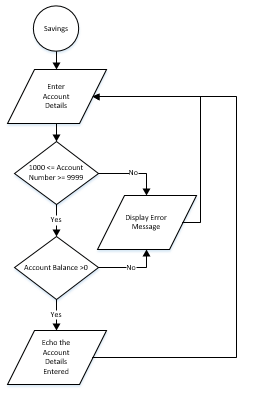


Figure 5: Administrator creates Current Account

Figure 6: Administrator

Figure 7: Administrator creates Savings Account

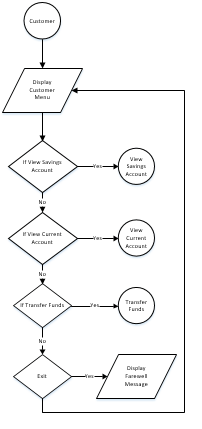
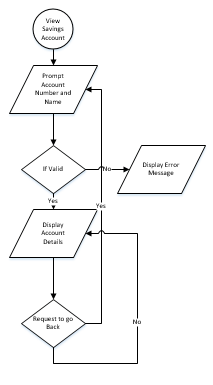
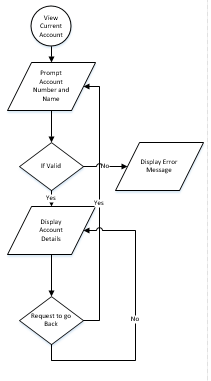


Figure 8: View Current Account

Figure 9: View Savings Account

Figure 10:Customer

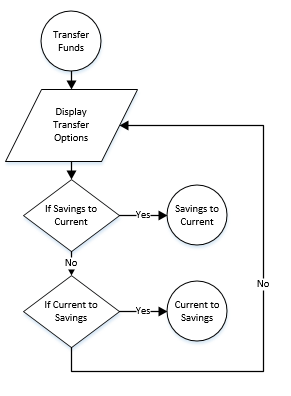
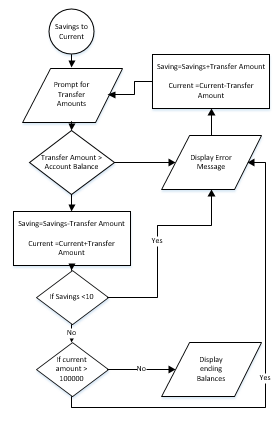
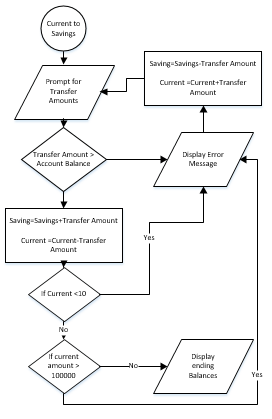


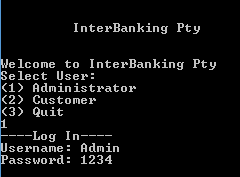
Figure 11: Current to Savings Fund Transfer

Figure 12: Savings to Current Fund Transfer

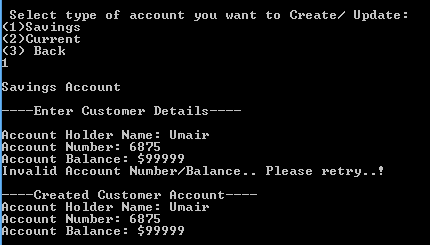
Figure 13: Transfer Funds

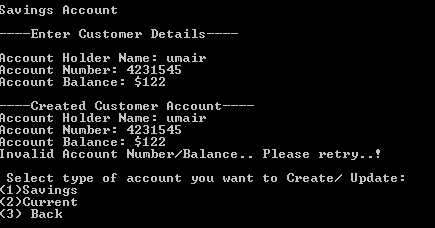
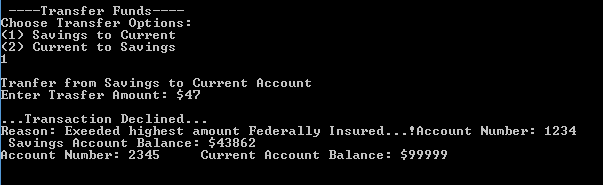
**Screenshots:**

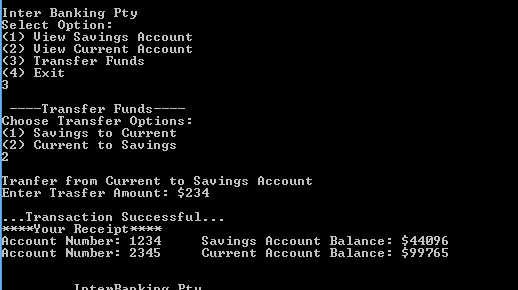
* **Prompt user for username and password**

****

* **Successful Creation of Savings Account**

****

* **Unsuccessful Account Data Entry (Invalid Account Number or Balance)**
* **Unsuccessful Transaction**
* Successful Transaction



**Stage 03**

**Task:**

Next you should add a third field that holds an annual interest rate earned on the account.

Adding statements to the **main()** function developed previously to do the following:

* Prompt the user for account numbers, beginning balances, and interest rates for

the two **BankAccount**s. We assume here that the interest rate will be different for

different type of bank account. For instance, the interest rate will be higher for

saving account compared to current accounts.

* The usual data validation developed in Stage 2 should now include the validation

of the interest rate. For instance, the interest rate is between 0.01% and 15.0%.

Also, the balance of each account should not be negative or over $100,000.00.

* Prompt the user for an automatic deposit account per month (in dollars and cents)

and an automatic withdrawal amount per month (also in dollars and cents).

* The display a table that forecasts the balance every month for the term of each

account. Calculate each month’s balance as follows:

* Calculate interest earned on the starting balance; interest earned is 1/12 of

the annual interest rate times the starting balance.

* Add the monthly automatic deposit.
* Subtract the monthly automatic withdrawal.
* Display the year number, month number, month-starting balance, and

month-ending balance.

* Write a C++ function to simply read and write from a text file. This will be used

to store each customer’s bank accounts details.

* At this stage you don’t need to provide any support for file search function, or

other.

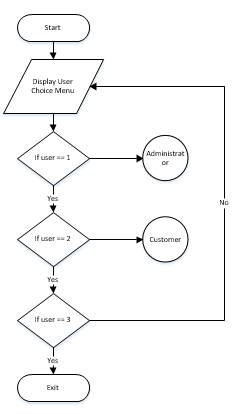
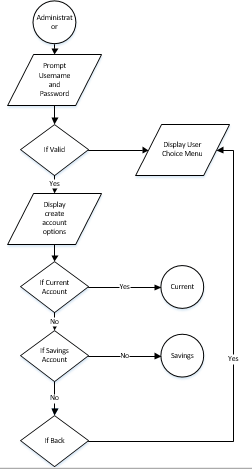
As this stage you should be able to demonstrate good use of all aspects studied so far,

including use of functions to modularize your code, arrays, string processing for instance for

passwords validation, and loops and control structures for data validation.

**Assumptions:**

* Creating a BankAccount would simultaneously create a Customer Account.
* Customers do not have a username/password, only the Administrator does.
* A customer can only create an account in January.
* Customer has to renew account at the beginning of every year.
* One customer can have only one account of each type (Savings/Current).
* Transactions may only occur between the Savings and Current Accounts of one Customer.
* Automatic deposit would be the interest rate
* Automatic Withdrawal would be the bank charge of $20.00 per month.

**Flowchart:**

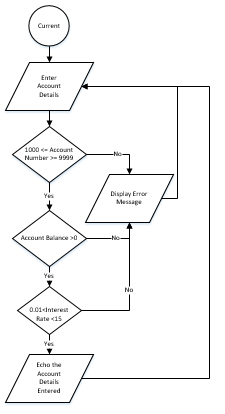


Figure : Create Current Account

Figure : Administrator

Figure : Backbone

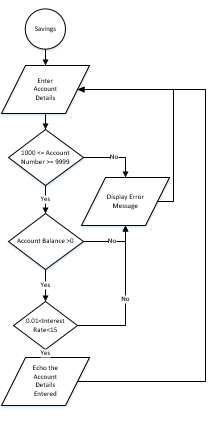


Figure : Create Savings Account

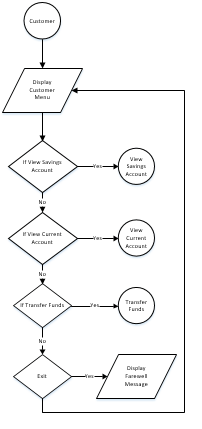
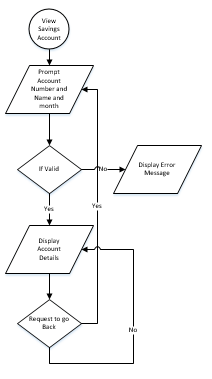
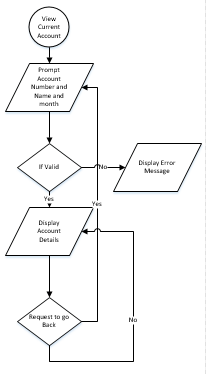
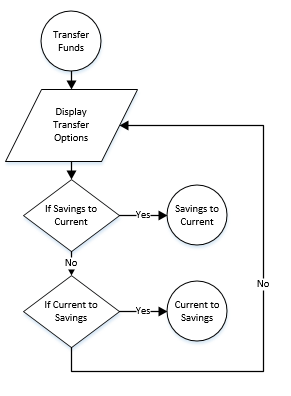
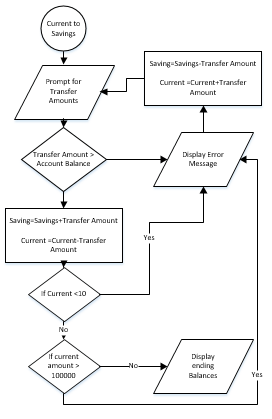
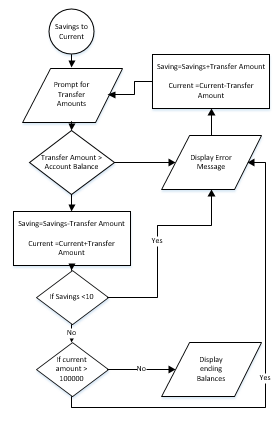


Figure :Current to Savings

Figure :Savings to Current

Figure : Transfer Funds

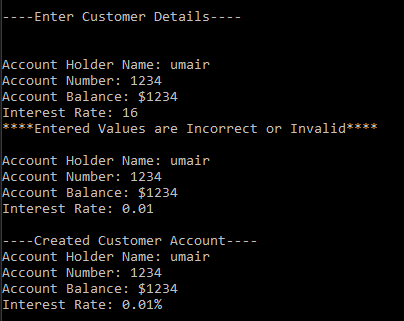
Figure : View Current Account

Figure : View Savings Account

Figure : Customer

**Screenshots:**

* Valid and Invalid “Interest Rate” data entry.



* Monthly Statement

