Mini Project Report

Linux OS and programming

|  |  |  |
| --- | --- | --- |
| **Name** | **PS Number** | **Email ID** |
| R PAVAN RAJ | 99002460 | rpavan.raj@ltts.com |

**Details**

# 

# **Contents**

[**Mini project** 4](#_Toc52747092)

# **Table of Figures**

[Fig 4. 1input.csv file 8](#_Toc52744839)

[Fig 4. 2 results.csv file 9](#_Toc52744840)

[Fig 6. 1 Use Case Diagram for Savings Account 17](#_Toc52744710)

[Fig 6. 2 Activity Diagram for Account 17](#_Toc52744711)

[Fig 6. 3 Object Diagram for Account 18](#_Toc52744712)

[Fig 6. 4 Use Case Diagram for Bank Account 19](#_Toc52744713)

[Fig 6. 5 Activity Diagram for Savings Account Database using Mutex and Semaphore 20](#_Toc52744714)

[Fig 6. 6 Class Diagram for Bank Account 21](#_Toc52744715)

## **Mini project**

**Project Theme: Banking Management System**

**1.1 Research:**

**Link for the Miniproject Folder:** [**Activity-6.**](https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EkSjrwsdOP1Jo4q9MCItHR4BOpVat78RRP-dzczDIbyMVA?e=2CuCVD)

**1.1.1 Ageing:**

|  |  |  |
| --- | --- | --- |
| **Types of Banks** | **Period Of Existence** | **Description** |
| 1. Monetary | 9000 B.C | The history of banking is intertwined with the history of money. Ancient types of money known as grain-money and food cattle-money were used from a time of around at least 9000 BC, as two of the earliest things that could be used for the purposes of barter. |
| 2.Record-keeping | 8000 BCE | Items utilized for record keeping, "bulla" and tokens, have been recouped from inside Near East unearthing’s, dated to a period starting 8000 BCE and finishing 1500 BCE, as records of the checking of agrarian produce. Starting in the late fourth centuries mental helper images were being used by individuals from sanctuaries and castles to serve to record loads of produce. |
| 3. Paper Banking (Passbook) | 18th century | A passbook or bankbook is a paper book used to record bank or building society exchanges on a store account.  A page with a pre-printed table. It has transcribed sections demonstrating measures of stores and withdrawals, and the equalization. Every passage has a mailing station date stamp. |
| 4.Web Banking | 1998 | In 1998, ICICI Bank introduced internet banking to its customers is an electronic installment framework that empowers clients of a bank or other budgetary organization to direct a scope of money related exchanges through the monetary foundation's site. |
| 5. Mobile Banking | 2017 | State Bank of India launched an integrated banking platform in India called YONO offering conventional banking functions but also payment services for things such as online shopping, travel planning, taxi booking or online education. |

**1.2 Description of Solution:**

Banking Management System consists of 2 types account namely savings and credit account.

Each account has different functionalities like debit, credit and display balance.

Database in terms of list: adding account, removing account and other critical functionalities are used in the database classes.

C++ Concepts used:

* List for the database for the two different account and List operations for the adding, removing and various evaluation of the Customer.
* Multilevel Inheritance used for the base class account and derived classes savings account and credit account class and level-2 derived classes savings account database and credit account database.
* Dynamic cast for the down casting of the inheritance and the methods for the derived classes.
* Virtual functions in the base/abstract class.

Linux Concepts used:

* Make files for the execution of the project.
* Static and Dynamic libraries for the faster execution and quicker run time.
* Threads and Semaphore used for the process synchronization in the test file.
* IPC concepts for communication between the files.

**1.3 SWOT Analysis:**

* Strengths
* Less Errors.
* Better Data Protection.
* Simple Functionality
* Abstraction of critical data.
* Weakness
* Optimization of the code for faster run time.
* Opportunities
* The code can be developed for other accounts like Demat, fixed deposit account.
* It can be used as base for integrating other concepts like file management and any database like SQL.
* Threats
* Better Solution can be made
* Corner Test Case can be tested and code could fail.
* The quality of the code could be improved.
* Usage of pointers and memory management for effective solution of the project and saving of memory

**1.4 Requirements:**

**1.4.1 High Level Requirements:**

|  |  |
| --- | --- |
| ID | Description |
| H1 | Create an account of savings account. |
| H2 | Create an account of credit account. |
| H3 | Remove an account with account no. |
| H4 | Display the balance of any account type. |
| H5 | Display the correct balance after debit and credit operations correctly. |
| H6 | Display the details of a customer with balance. |

**1.4.2 Low Level Requirements:**

|  |  |
| --- | --- |
| ID | Description |
| L1 | Create a database of account with given records using the list concept in C++ |
| L2 | Remove a customer from the create list by list operations and erasing the customer and displaying NULL whenever prompted to display the deleted account. |
| L3 | Get the correct balance of the savings account type. Credit means adding money into the savings account and debit means removing money from the account. After this operation display the correct balance. |
| L4 | Get the correct balance of the credit account type. Debit means adding money into the savings account and credit means removing money from the account.. |
| L5 | Counting all the accounts in the database of the list using the list operations and displaying it. |
| L6 | Computing the maximum balance from the list of accounts and displaying it. |
| L7 | Display the account details from the database of list. |
| L8 | Display the balance whenever prompted in the code. |

**1.5 Design:**

**1.5.1 High Level Design:**

**Behavioral Diagram:**

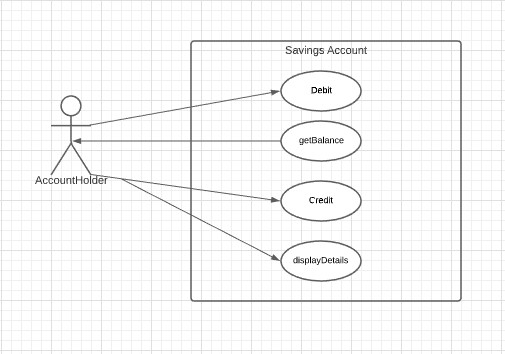


Fig 6. 1 Use Case Diagram for Savings Account

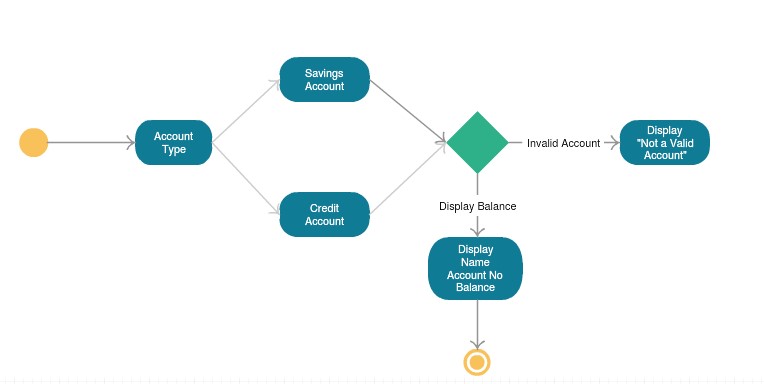
****

Fig 6. 2 Activity Diagram for Account

**Structural Diagram:**

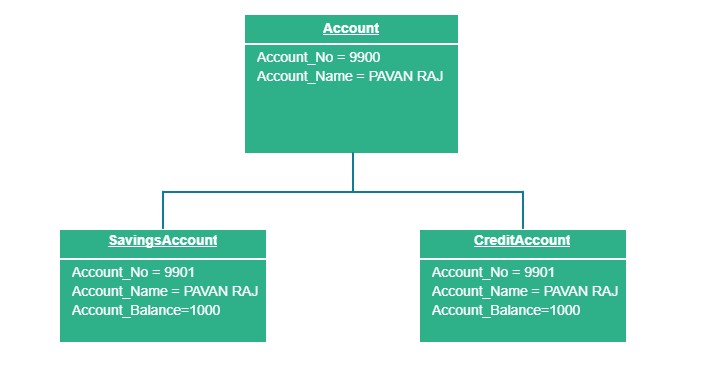
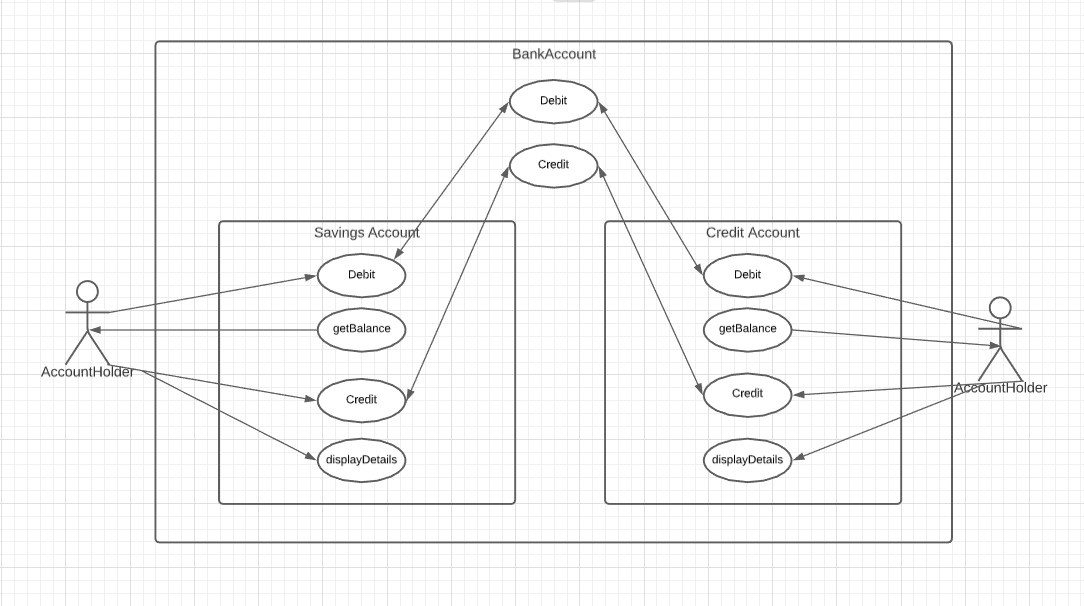
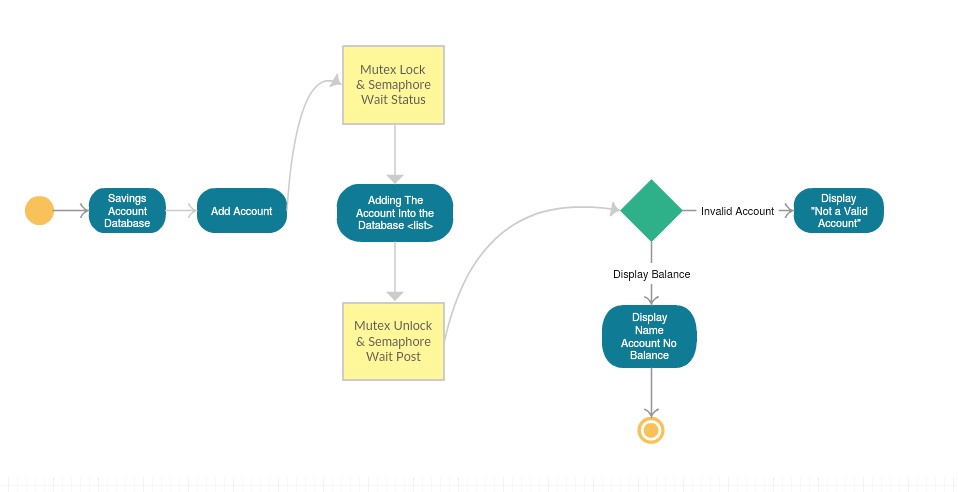
****

Fig 6. 3 Object Diagram for Account

**1.5.2 Low Level Design:**

**Behavioral Diagram:**

Fig 6. 4 Use Case Diagram for Bank Account

Fig 6. 5 Activity Diagram for Savings Account Database using Mutex and Semaphore

**Structural Diagram:**

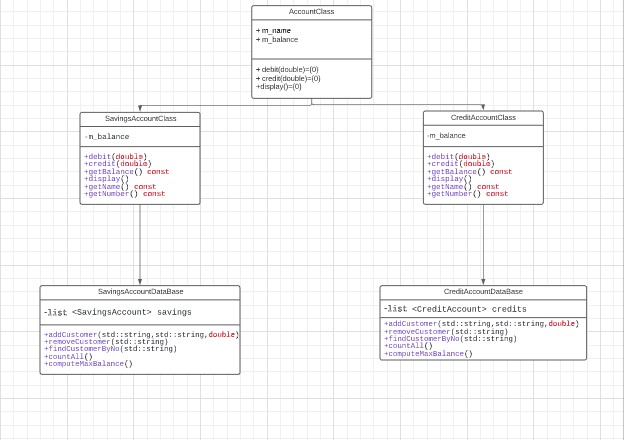
****

Fig 6. 6 Class Diagram for Bank Account

**1.6 Test Plan:**

**1.6.1 High Level Test Plan (Integration Test Plan)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Precondition** | **Expected IO** | **Expected OP** | **Actual OP** |
| TL1 | Displaying the correct name, account no and the balance. | Account Number, Name and Balance set using memory allocation | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TL2 | Debit any amount in the account number | Existing Bank account customer | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TL3 | Credit any amount in the account number | Existing Bank account customer | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TL4 | Count the number of customers in list. | Add the customers at the start of the code | Set using the GTest Framework | No of Customers in list | No of Customers in list |
| TL5 | Get the Balance of a customer | Existing Bank account customer | Set using the GTest Framework | Balance of the customer | Balance of the customer |

**1.6.2 Low Level Test Plan (Unit Test Plan)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Precondition** | **Expected IO** | **Expected OP** | **Actual OP** |
| TH1 | Default Constructor Check | Set the name, account and balance to zero. | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH2 | Parameterized Constructor Check | Set a name, account and balance. | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH3 | Computation Test  Compute credit and debit and check balance | Set a name, account and balance. | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH4 | With Dynamic Cast Test | Set a name, account and balance. | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH5 | Add customers to a list of savings account and credit account | Create a list of accounts and use Mutex and semaphore for synchronization | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH6 | Remove a customer from a list of savings account and credit account | Preexisting list of accounts and use Mutex and semaphore for synchronization | Set using the GTest Framework | Build Success  Output Passing. | Build Success  Output Passing. |
| TH7 | Compute Maximum Balance from the list of customers | Preexisting list of accounts and use Mutex and semaphore for synchronization | Set using the GTest Framework | Build Success  Output Passing.  Display the maximum balance | Build Success  Output Passing. Display the maximum balance |
| TH8 | Count the number of customers in list. | Preexisting list of accounts and use Mutex and semaphore for synchronization | Set using the GTest Framework | Build Success  Output Passing.  Display the number of customers. | Build Success  Output Passing. Display the number of customers |

**1.6 Implementation:**

**Repository Link:** [**Mini Project Repo.**](https://github.com/99002460/MiniProject_CPP_and_LINUX.git)

CI framework:

* GIT: Used git for storing and continuous evaluation of the code. Very important tool for Projects
* Build and Make –

Used GitHub Workflow files for GitHub actions namely,

1. C/C++ CI workflow
2. Cppcheck
3. Valgrind
4. Build

* Unit Test Framework: Used the GTest framework for the implementation of the test plan and successfully passing 16 test cases.
* Code Quality: Used CODACY for code quality. “A” Grade for the code implemented in the repository.