|  | **BSc Computing Level 3 Semester - 2**  **Object Oriented Programming in C++**  **Assignment 2 – Final Project**  **Total marks: 40% of total Assessment**  **Submission Due date : 19th August 2022 – [11:00 PM]** |
| --- | --- |

At the end of the session as a part of the Assessment 2 (Project work), students are required to work on the case scenario given below and

* Come up with a working program in C++ following appropriate coding conventions
* Perform a live demo on the prescribed date in front of the instructor for evaluation purpose
* Submit a report in proper format – It should contain minimum requirements analysis according to OOP [That is UML diagram etc…] followed by explanations of work flow of the project and Code of the project.
* **Student must have to submit their work in a zip FinalProject.rar containing the report and the project folder containing all the .cpp files or any extra library files used** in the project.

**All students are required to make use of appropriate coding conventions in C++, do proper commenting and make use of the things covered and learnt in the classes for solving the problem. The project work should be done by students on an individual basis.**

**Bank Account**

Each student is required to come up with a **menu driven** software system for a financial institution that allows bank officials to serve its customers in performing financial transactions including creating accounts, making deposits, withdrawing money, searching for customer information etc.

Perform a **thorough requirements analysis of the system** that you are designing and developing so that you become familiar with the domain and the existing requirements.

**Design** the system by making use of visual representations that show architecture of the system comprising the existing classes in the system to be developed.

**Identify potential classes**, their attributes (state), member functions (behavior) for the given scenario then write a complete program that demonstrates these classes by asking the user to enter the amounts of deposits and withdrawals for a savings account and checking accounts.

The program **should display statistics** for the month, including beginning balance, total amount of deposits, total amount of withdrawals, service charges, and ending balance.

All these activities carried out by the students for doing the project work must be specified in a **report** in proper format and submit it to the module leader in Orbund.

**Guidelines:**

Design a generic class to hold the following information about a bank account:

* Balance
* Number of deposits this month
* Number of withdrawals
* Annual interest rate
* Monthly service charges

The class should have the following member functions:

**Constructor**: Accepts arguments for the balance and annual interest rate.

**makeDeposit**: A virtual function that accepts an argument for the amount of the deposit. The function should add the argument to the account balance. It should also increment the variable holding the number of deposits.

**makeWithdrawal**: A virtual function that accepts an argument for the amount of the withdrawal. The function should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals.

**calcInt**: A virtual function that updates the balance by calculating the monthly interest earned by the account, and adding this interest to the balance. This is performed by the following formulas:

Monthly Interest Rate = (Annual Interest Rate / 12)

Monthly Interest = Balance \* Monthly Interest Rate

Balance = Balance + Monthly Interest

**monthlyProc**: A virtual function that subtracts the monthly service charges from the balance, calls the calcInt function, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

Next, design a savings account class, derived from the generic account class. The savings account class should have the following additional member:

**status** (to represent an active or inactive account)

If the balance of a savings account falls below $25, it becomes inactive. (The status member could be a flag variable.) No more withdrawals may be made until the balance is raised above $25, at which time the account becomes active again.

The savings account class should have the following member functions:

**makeWithdrawal**: A function that checks to see if the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account is not active.) A withdrawal is then made by calling the base class version of the function.

**makeDeposit**: A function that checks to see if the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. The deposit is then made by calling the base class version of the function.

**monthlyProc**: Before the base class function is called, this function checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4 is added to the base class variable that holds the monthly service charges.(Don’t forget to check the account balance after the service charge is taken. If the balance falls below $25, the account becomes inactive.)

Next, design a checking account class, also derived from the generic account class. It should have the following member functions:

**makeWithdrawal**: Before the base class function is called, this function will determine if a withdrawal (a check written) will cause the balance to go below $0. If the balance goes below $0, a service charge of $15 will be taken from the account. (The withdrawal will not be made.) If there isn’t enough in the account to pay the service charge, the balance will become negative and the customer will owe the negative amount to the bank.

**monthlyProc**: Before the base class function is called, this function adds the monthly fee of $5 plus $0.10 per withdrawal (check written) to the base class variable that holds the monthly service charges.

**Note:**

- Remember proper input validation is a MUST

- Proper commenting and documentation should be done

- Proper indentation in coding is a MUST

- Thorough testing should have been done before doing the demo!!

**Assessment Criteria**

Students MUST make use of file handling features of C++ for depositing data permanently.

Students MUST have to use proper exception handling techniques.

Students MUST assimilate all of the features of C++ that they have studied in the course in the Project work like Encapsulation, Inheritance, Polymorphism, Type conversion, Operator Overloading etc.

***Also note that 30% of marks for the creativity shown in the software developed*** which you must have to explain during the demo.

Note: ***The submission of a working program that satisfies the specification does not in itself guarantee marks***. A student must have the ability to demonstrate, discuss and explain their submitted work. The awarded mark may be lowered if the student is unable to satisfactorily explain their work.

All the students go through the following marking schema to understand the minimum requirements of the ass

| **Element** | **First (70+ %)** | **2:1 (60-69%)** | **2:2 (50-59%)** | **3rd (40-49%)** | **Fail (0-39%)** | **Marks** |
| --- | --- | --- | --- | --- | --- | --- |
|
|  | | | | | | |
| **Working Program (10%)** | Creativity shown in the software developed. A complete program written using proper indentation, documentation, all input validation done, all functionalities achieved, all conditions specified in the question fulfilled accurately. Used all features studied in C++ | A complete program written using proper indentation, documentation, all input validation done, all functionalities achieved, all conditions specified in the question fulfilled accurately with minor errors, most features studied in C++ are used | Main structure of the program is identified. Most functionalities achieved and input validation done, some conditions specified in the question not fulfilled, few features studied in C++ are incorporated | Recognisable as a C++ program following Object Oriented conventions, but major omissions or errors seen in the program written. Not many features studied in C++ are incorporated | Doesn’t resemble a C++ program , Object Oriented conventions not followed. Major omissions and or errors.Features studied in C++ are not incorporated |  |
|  | **10-7.0** | **6.9-6.0** | **5.9-5.0** | **4.9-4.0** | **3.9-0.0** |  |
| **Demonstration (20%)** | Excellent demonstration of the coded program, came up with a thorough explanation with professionalism, fully able to handle questions asked every effectively, fulfilled the requirement of the course | Excellent demonstration of the coded program, came up with a very good explanation, fully able to handle questions asked every effectively, fulfilled the requirement of the course | Good demonstration of the coded program explained well, answered most of the questions asked every effectively, fulfilled the requirement of the course | Weak demonstration of the coded program, answered few questions asked, showed some amount of effort being put in for completing the assignment | Showed little or no evidence of preparation, ,failed to answer important questions asked, failed to fulfill requirement of the course |  |
|  | **15-10.5** | **10.4-9.0** | **8.9-7.5** | **7.4-6.0** | **5.9-0.0** |  |
| **Report (10%)** | Excellent report, very well written, consistency maintained fully, proper documentation and explanation provided. The report provides an exhaustive amount of information with clarity. | Very well written report, with consistency well maintained, good documentation and explanation provided. The provides a good amount of idea of the student's outcome | Good report, tried to maintain consistency, good amount of details provided, some amount of touch up is desirable | Report tries to give a hint on what the student has achieved but failed to maintain consistency in many places, contents seem hazy, documentation needs improvement | The report is in poor format, no consistency is maintained, contents in the report are placed haphazardly. |  |
|  | **20-14.0** | **13.9-12.0** | **11.9-10.0** | **9.9-8.0** | **7.9-0.0** |  |
| **Feedback:** |  |  |  |  | **Total** |  |