Project Specification- Major Practical

Introduction:

My project implements a bank app for customers. It allows customers to

- Login (name and password)
- View profile
- Track transactions (date, location, amount)
- Set up savings plan with 2 options in terms of interest and calculate their loan period, monthly payment (all data such as interest, etc will be customers' expectations)

***All information is stored on a database, which is a file called data.txt**

***Use data.txt in case you want to run the app as it will ask for name and password.**

Design description

Assessment concepts

1) **Memory allocation from stack and the heap:**
   - **Array**: I will use dynamic arrays to create customers and transactions, so that in the future, I can add in more (adding feature will not be included this time).
   - **Strings**: customers' names, location, address, DOB, etc. input/ output
   - **Object**s: Bank, Customers, Transactions, plan savings and Loan
2) **User Input and Output:**
   - **I/O of different data types:** Users enter names, passwords, select options by numbers.
3) **Object- oriented programming and design:**
   - **Inheritance:** There are different savings and loan options. For example, for the savings plan, each type of saving has a different formula.
   - **Polymorphism:** Although they are all savings plan, they have different ways to calculate the total money, depending on their types.
   - **Abstract classes:** Plan_savings has an abstract class with child classes returning the total money.
4) **Testing:**
   - **Test inputs/ Expected outputs:** Read in customers' info and then check if they print out accurately. Do the same for customers' transactions. To make it efficient, diff is applied.
   - **Automated Testing:** Inform errors when reading in invalid inputs (like options with negatives, etc.). If there is no error, nothing will be printed out.
   - **Boudary Testing:** Use for plan savings and loan classes as they include mathematical formulas. In particular, test for values near the limit to see if they return the correct answers
   - **Regression Testing:** run makefile everytime a change is made.

Class Descriptions

Customer

Customer has basic information (name, address, dob, job, password) and transactions. Methods allow setting and name those pieces of information. In addition, printInfo will print out information of the customer.

Bank

Bank has its own name, address, a list of customers and necessary functionalities (like calculating savings and loan). Methods allow adding customers and getting the customers list. They also calculate savings and loan which help customers setting up their plans.

Plan savings

It is an abstract class with the initial amount of money a, interest r, period n. If customers want to put money into savings, they will use it to calculate interest, total money, … Set_values(a,r,n) allows users to input needed value to calculate their inquiries.

- Double_interest: The interest from previous periods will be added to the initial amount of money and double the interest.
- Annuity: Every month, users will put an amount of money into their savings account. Based on given values, it will calculate the total money after a period.

Loan

It is a parent class using virtual with interest r, payment p and the amount of money a. If customers want to borrow some money from the bank, they will use it to calculate how long they will fully repay the loan or how much the monthly payment is. Setting functions allow users to input their expected values and then functions will return the results.

- Payment: calculate the monthly payment based on given values and the period
- Period: calculate how long it takes to fully repay the money (including interests)

Code style

- Codes will be properly indented.
- Functions and variables will be named after their functionalities, so that readers can immediately know what they are going to perform.
- In main, comments will be used to separate clearly functions from each other.
- "////////" is applied to make it easier to understand each part of the program.

Testing Plan

Unit testing

- Our unit tests will cover public functions in our classes.
- Each class will have a test file like ClassNameTester.cpp. They include a main method.
- For class customer and transaction, each function with a set of inputs and test it matches an expected output, using diff
- For plan savings and loan, use "if- statement" to make comparison between real outputs and expected ones.

Input- Output testing

- Create input files and expected outputs for each function
- Using diff to check if the functions print out correct results

Automated Testing

- Create a makefile that will compile each functions and test files. When typing make, it will compile everything. If it does not print out anything, the program works.

Boundary Testing

- Use for two classes that require some calculations
- Try values near the limits (for example, r>0, try 1, 2 and big numbers)
- Increase the values everytime

Integration Testing

- Make use of makefile to run 2 or 3 functions to see if they work correctly
- If there is any associations between those, check carefully to avoid any conflicts.

Regression Testing

- Whenever there is a change or implementation, re-run everything from the beginnings to see how it behaves.
- When there is an error, check each file seperately and check if there is any connections between those.

Schedule Plan

WEEK 8:

- Set up a plan
- Make a pseudo code for 4 functions of the Bank application
- Create the first class – Transaction
- Create Customer class with some basic methods (constructor, destructor, set-get info)

WEEK Break 1:

- Implement Customer class by adding get_trans and add_transactions
- Create a database
- Search for a way to read in data from an Excel file
- Create Bank class with basic methods
- Implement Bank class to store data of 8 customers from database
- Solve financial problems to get 2 important formulas about double interest and annuity.
- Create Plan_savings class with an abstract class and 2 child classes

WEEK Break 2:

- Develop 2 formulas to calculate monthly payments and period of a loan.

- Create Loan class with 1 parent class and 2 child classes.
- Implement the class with 2 found equations
- Test 2 classes (plan_savings and loan) to check the accuracy of formulas

WEEK 9:

- Implement the Bank class with 2 more functions which calculate savings and loan
- Create a main program, each time add one function of the app to see if it works
- First, make sure that data are fully read in
- When there is no error, implement the main with conditions for variable that users input
- Design instructions so that the users can easily use the app
- In the main, set up a security mechanism which asks for passwords.

WEEK 10:

- Develop a way to test data from the text file
- Test everything, including small functions
- Check the concept “Stack- Heap”
- Check carefully pointers using allocation and delete them in the destructor.

WEEK 11:

- Implement more tests on functions to make sure they work.
- Correct mistakes when printing out and instructions which make the customers confused
- Try random inputs to see how the program works
- Run the whole program and fixed remained errors.

CLASS DIAGRAM (hand-drawn)