|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **National University of Computer and Emerging Sciences, Lahore Campus** | | | | |
| C:\Users\saif\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\final design.jpg | **Course:** | **Object Oriented Programming** | **Course Code:** | **CS-1004** |
| **Program:** | **BS (Computer Science)** | **Semester:** | **Spring 2022** |
| **Deadline:** | **30-July-22** | **Total Marks:** | **30** |
| **Section:** | **9A** | **Weightage:** | **3** |
|  | **HomeWork\_1** |  |  |
|  |  |  |  |
| **Instruction/Notes:** |  | | | |

**In case of plagiarism/cheating, this activity will be marked as zero.**

## **Task#1:**

Create a class fraction with the following data members and supporting member functions.

**Data Members**

* static int fractionsCount;
* int numerator;
* int denominator;

**Member functions:**

1. **fraction(int n = 1, int d = 1);**

default parameterized constructor assigns the value of ‘n’ to **numerator** and ‘d’ to **denominator**. The value of ‘d’ should not be zero. Increment the value of static data member.

1. **void setNum(int n);**

Setter function to assign the value of ‘n’ to numerator.

1. **void setDen(int d);**

Setter function to assign the value of ‘d’ to denominator. Value should not be zero.

1. **int getNum() const;**

const getter function to return numerator

1. **int getDen() const;**

const getter function to return denominator

1. **fraction operator + (const fraction & f);**

overload addition operator (+). This function should store the sum of calling object and the parameter **f** in a newly created object and return the simplest form of object to the calling function. e.g.,

**Remember! You can not add the fractions until their denominators are same.**

1. **fraction operator \* (const fraction & f);**

overload multiplication operator (\*). This function should store the product of calling object and the parameter **f** in a newly created object and return the simplest form of object to the calling function. e.g.,

1. void operator ++(); **//overload pre and post fix increment/decrement**
2. bool operator == (const fraction & f);

convert the fractions into simplified form before comparison because the fractions and are equal since in its simplified form is equals to .

1. bool operator > (const fraction & f);

overload relational operator (>). This function should return true if the calling fraction is greater than the parameter **f.** if the denominators of both the fractions are same then simply check the numerator.

1. Overload stream insertion (<< ) and extraction operators (>>)

1. bool **isProper()** const;

This function should return true if the fraction is proper, false otherwise. A proper fraction is one in which numerator is greater than denominator.

1. static int getCount();

This function should return the count of total number of fractions.

**Initialize the static data member by 0 outside the class.**

1. void show() const
2. Create a destructor which decrements the value of fractionCount.

**Write a main function to test the functionalities of all the member functions of fraction ADT. Provide the implementation of all the methods outside the class. Your class should only be used for the declaration of methods.**

# **Task#2:**

**Create** a class **ATM** with following data members and methods:

* int **totalCash**
* default parameterized constructor
* void setCash(int)
* int getCash()
* void showBalance() //do not change the name of function.

**Derive** a class **Transaction** **from ATM** with the following data members:

A dynamic array of AccountHoler class

* AccountHolder \* arr
* int size
* **default parameterized constructor**, it should receive the size of array otherwise assign the value 0 to size. Declare a dynamic array inside the constructor and initialize the values with 0.
* **void withdraw Amount function**. This function should receive the **account number** and **withdrawal amount** from main function.
  1. Check that whether the account number is valid or invalid
  2. Check that whether the balance in the user's account is sufficient enough for this transaction.
  3. Check that whether the balance in the ATM is sufficient enough for this transaction.
  4. update the totalCash of ATM and balance of the account holder after this transaction
* **void setData function**. This should receive the **index value, account number, and balance**. If the index is in the given range and account number is valid than update balance in the account of that particular account holder. if the index value is out of bound than report the message “**Invalid index**”. If account number doesn’t exist than display the message “**invalid account number**”.
* void showBalance function. This function should display the balance of all the account holders on console
* void **resize function.** This function should receive the size as a parameter.
* Update the previous value of size with the current value.
* Create a new array with the updated size.
* Copy the contents of previous array in this new array.
* If the size of newly created array is greater than previous array than initialize the remaining indices with 0.
* Delete the previous array and store the address of newly created array in arr.

**Create** a class **AccountHolder** with the following data members and methods:

* int Account Number
* int balance
* default parameterized constructor.
* setAccountNumber
* setBalance function
* getAccountNumber
* getBalance

**Inside main function**

transaction t1(3, 5000); **//3 is the size of array and 5000 is ATM cash**

int choice;

do{

showMenu();

cout << “Enter your choice”;

cin >> choice:

switch(choice){

case 1:

case 2:}

}while(choice!=0);

Display the following menu inside showMenu function. You can change the numbering against the operations, but it is mandatory to handle all the cases.

Enter 1 to insert a record// for set data function

Enter 2 to update ATM cash//

Enter 3 to update account holders balance// this should handle the case of cash deposit.

Enter 4 to view ATM cash//

Enter 5 to view account holder’s balance

Enter 6 to resize the array

Enter 7 to withdraw amount from the account

Enter 0 to terminate the program

**There should not be any memory leak in your program. Your program should only accept valid input and in case of invalid input, prompt the user repeatedly to enter the valid input.**