COS1512 ASSIGNMENT 3 2023

DUE DATE:	**21 August 2023**
CUT-OFF DATE:	**24 August 2023**
TUTORIAL MATTER:	**Chapters 10, 11, 12 and 15 of the Study Guide (Appendix D)** **Chapters 10, 11, 12 (excluding "Creating a Namespace")** **Appendices 7 and 8 in Savitch**
EXTENTION:	**None**
WEIGHT:	**40%**

Question 1

Consider the following structure used to keep employee records:

```
struct Employee
{
    string firstName;
    string lastName;
    float salary;
}
```

Turn the employee record into a class type rather than a structure type. The employee record class should have private member variables for all the data. Include public member functions for each of the following:

- a default constructor that sets the employee's first name and last name to a blank string, and his annual salary to 0;
- an overloaded constructor that sets the member variables to specified values;
- member functions to set each of the member variables to a value given as an argument to the function (i.e. mutators);
- member functions to retrieve the data from each of the member variables (i.e accessors);

Embed your class definition in a test program. The test program should create two Employee objects and display each object's annual salary. Use the overloaded constructor to initialise one of the Employee records as Joe Soap with an annual salary of R145600.00. Obtain the following input values for the other Employee record from the keyboard:
Joanne Soape
R154460.66

Now give each employee a 10% raise and display each `Employee` object's annual salary again.

Quesion 2

Design and implement a C++ class called `Module` that handles information regarding your assignments for a specific module. Think of all the things you would want to do with such a class and write corresponding member functions for your Module class. Your class declaration should be well-documented so that users will know how to use it.

At a minimum your class `Module` should contain information on the module code, the module name, the marks for assignments, the year mark and the final mark.

Test your class in a small program where you instantiate an object of the class, initialize the member variables that represents the assignment marks, calculate a year mark and display the year mark.

Quesion 3

Implement your class `Module` from question 2, as an abstract data datatype (ADT) using separate compilation, so that separate files are used for the interface and implementation.

Write a main program that does the following:

- Declare an array of all your modules. The elements of the array must be of objects of the class `Module`.
- Initialise the array with the modules you are registered for. Obtain the module codes, names of modules and assignment marks from the user. Initialise each module with the assignment marks you have obtained for the module (or would like to obtain for the module).
- Determine your semester mark for each module: the first assignment contributes 30% and the second assignment 70%. Display the semester marks.
- Adjust the marks for Assignment 2 for COS1512 with +5%.
- Determine your semester mark for COS1512 again to see what effect the update had.
- Display an updated list of your semester marks for all the modules you are registered for.

Enrichment exercise:

Adapt the application program to use a vector instead of an array. It should not be necessary to change the class interface or implementation file in any way.

Question 4

Consider the following class declaration:

```
class ExamType
{
     public:
         ExamType();
         ExamType(string m, string v, int t, string d);
     private:
         string module;
         string venue;
         int time;
         string date;
};
```

Explain what is wrong with the following code fragment and include code to correct it:

```
int main()
{
    ExamType exams[12];
    for (int i = 0; i < 12; i++)
        if (exams.module[i] == "COS1512")
            cout << "COS1512 will be written on "
                 << exams.date << " at " << exams.time;
    return 0;
}
```

Question 5

Consider the following class declaration:

```
class Date{
   public:
      //constructors
      Date();
      Date(int day, int month, int year);
      //accessors
      int getDay() const;
      int getMonth() const;
      int getYear() const;

      //mutators
      void setDay(int day);
      void setMonth(int month);
      void setYear(int year);
      //operators to calculate next and previous days
      Date &operator++();
      Date &operator--();
```

```
         bool operator<(const Date &d);
      private:
         //the current day month and year
         int theday;
         int themonth;
         int theyear;
         //return the length of current month, taking into
         //account leap years
         int monthLength();
   };
```

Implement and test the `Date` class, taking the following guidelines into account:

a) The default constructor should initialise the date to 14 September 1752. (1)
b) The overloaded constructor should initialise the date with the given day, month and year. (1)
c) The functions `getDay()`, `getMonth()` and `getYear()` should return the current day, month and year respectively. (1)
d) The functions `setDay()`, `setMonth()` and `setYear()` should change the current day, month or year to the given value. (1)
e) The operator ++ should advance the date by one, and return the new date. (2)
f) The operator -- should set the date back by one day, and return the new date.(2)
g) The operator < should calculate whether the receiving date object (left argument) precedes the parameter date (right argument).
For example, Date(1,1,2002) < Date(1,3,2002). (3)
h) The private member function `monthLength()` should return the number of days in the current month, taking into account leap years. A year is a leap year if it is either (i) divisible by 4, but not by 100, or (ii) divisible by 400. In a leap year, February has 29 days, otherwise it has 28 days. (3)
i) Also overload the insertion operator << to output a date to the screen. For example, the date in (a) above should be written as: 14 September 1752. (3)

Test the `Date` class in a C++ program that will do the following:

- Declare a new `Date` object called `d1` using the default constructor.(1)
- Display the day, month and year of `d1` on the screen. (1)
- Change the date to 28 February 2000. (1)
- Advance this date by one and display the new date on the screen. (1)
- Now change the date to 1 January 2002. (1)
- Set this date back by one and display the new date on the screen. (1)
- Finally change the date to 31 December 2002. (1)
- Advance this date by one and display the new date on the screen. (1)
- Declare a second date object `d2(1,1,2003)`. (1)
- Determine if `d1` is earlier than `d2` and write the result on the screen.(1)
- Operators ++, -- and < are declared as member functions in the class declaration above. Implement these operators as friend functions of class `Date` also. Run your program twice (each time with a different version of the overloaded operator ++, -- and <; comment the other versions out during each run). (6)
- 2 marks for correct output (2)

Enrichment exercise:

Turn the `Date` class into an ADT, so that separate files are used for the interface and implementation. Use separate compilation to compile the implementation separate from the application program that tests the ADT.

Question 6

Define a class `PhoneCall` as an ADT that uses separate files for the interface and the implementation. This class represents a phone call and has three member variables:

- `number`, a string that holds the phone number (consisting of 10 digits) to which a call is placed
- `length`, an `int` representing the length of the call in minutes
- `rate`, a `float` representing the rate charged per minute.

In addition, the class should contain a default constructor that initializes `number` to an empty string, `length` to 0 and `rate` to 0. It should also contain an overloaded constructor that accepts a new phone number and sets `length` and `rate` both to 0, as well as a destructor that does not perform any action.

Include accessor functions that returns the values stored in each of an object of class `PhoneCall`'s member variables respectively.

Class `PhoneCall` also contains a member function `calcCharge()` to determine the amount charged for the phone call. Use the following prototype:

```
float calcCharge();
```

Overload the equality operator== as a `friend` function to compare two phone calls. Use the following prototype:

```
bool operator==(const PhoneCall & call1, const PhoneCall & call2)
```

This function returns `true` if both `call1` and `call2` have been placed to the same number and `false` otherwise.

Overload the stream extraction operator >> (implemented as a `friend` function) so that it can be used to input values of type `PhoneCall`, and the stream insertion << (implemented as a `friend` function) so that it can be used to output values of type `PhoneCall`.

Demonstrate the class in an application program (`main()`) that is used to determine the total amount spend on phone calls to a specific phone number in one month. Allow the user to enter the phone number for which the total amount spent should be determined. Use the overloaded constructor to initialise the `PhoneCall` object theCall to the number the user specified. The PhoneCall objects representing the calls made during one month is stored in a file MyCalls.dat. Use a while loop to read the phone calls from MyCalls.dat, use the overloaded equality operator== to compare the phone numbers read from MyCalls.dat one by one with theCall, and determine the total amount spend on phone calls to theCall, as well as the number of calls made to this number. Also determine the longest call made to theCall and display this call together with the total amount spent on calls to theCall, and the number of calls to theCall.

Test your program with the following data:

Phone calls in file MyCalls.dat:

```
0123452347     12   3.50
0337698210     9    3.15
0214672341     2    1.75
0337698210     15   3.15
0442389132     8    1.75
0232189726     5    3.50
0124395623     6    3.50
0337698210     2    3.15
0337698210     5    3.15
```

Phone number to test:
```
0337698210
```

Question 7

Define a class Student with member variables for a student's name, student number, address and degree. All of these member variables are strings. Add appropriate constructors and accessors for class Student and include the following member functions:

- a member function display_info() that overloads the stream insertion operator << to display the values of all the member variables of a student.
- a member function calcFee() to calculate the initial registration fee for a student. For undergraduate students the initial registration fee is R500 and for postgraduate students the initial registration fee is R600. All undergraduate student degrees begin with a 'B' which will allow you to determine whether a student is an undergraduate or postgraduate student.

(a) Implement class Student.

(b) Test class Student in a driver program that does the following:

- instantiates an object of class Student, with the following details:

    name: Mary Mbeli

student number: 12345678

address: Po Box 16, Pretoria, 0818

degree: BSc

- use the accessor functions to display the specifications of the instantiated object on the console
- display the specifications of the instantiated object on the console with the member function `display_info()`.
- calculate and display the fee for the student.

(c) Derive and implement a class `PostgradStd` from class `Student`. This class has an additional member variable, `dissertation` (the title of the Masters of doctorate the student is doing). Class `PostgradStd` also has an overloaded constructor and an accessor member to return the member variable `dissertation`. The class `PostgradStd` should override function `display_info()` in order to display the values of all the member variables of `PostgradStd`. The class `PostgradStd` should also override function `calcFee()` to determine the additional fee for a postgraduate student which is R12000.

Implement the overloaded constructor for the class `PostgradStd` by invoking the base class constructor.

(d) Test class `PostgradStd` in a driver program that does the following:

- instantiates an object of class `PostgradStd`, with the following details:

  name: Mary Mbeli

  student number: 12345678

  address: Po Box 16, Pretoria, 0818

  degree: PhD

  dissertation: How to get a PhD

- use the accessor functions to display the specifications of the instantiated object on the console
- display the specifications of the instantiated object on the console with the member function `display_info()`.
- calculate and display the outstanding fee for the student.