**CSC 2110**

**Computer Science I**

**Fall Term 2015**

**Programming Project**

**80 points**

**Due 11/16/2015 (12:00 P.M.)**

**The goal of this project is seven-folds:**

1. Understand the principle of **abstraction** and **encapsulation** and use them to implement classes.

2. Extend classes via **inheritance**.

3. Learn about class **polymorphism**

**4.** Learn about **overloading.**

5. Learn about class **templates**

6. Use **dynamic array** in classes

7. Analyze, design, implement, and test a practical real-world application.

**Requirements:**

• Analyze the problem; outline the problem and its solution requirements.

• Design an algorithm to solve the problem.

• Implement the algorithm in C++, and verify that the algorithm works.

• Each class must contain a header file (.h) and an implementation file (.cpp).

**Restrictions:**

You must work individually. Use only material from class or from the text book (chapters 1-13). All code must be the work of the individual. Do not share your code or copy from external resources.

**Grading:**

The grade of each program will be based on the creation of a program that works correctly, up to some details (40%), clear problem analysis and algorithm design (10%), the appropriate use of classes (20%), the production of clear output, with readable formatting and without unnecessary repetition (15%), composition of informative comments (10%), and testing the program with different inputs (05%). Programs must compile.

**Submission**

* Create the application program from scratch using visual studio C++ 2013.
* Type your analysis and algorithm for each problem in this file.
* Solve each problem and include the source file of each problem and this file in a folder. Name the folder (CSC2110\_Project\_01). Compressed the folder and upload it to the blackboard using the appropriate folder by the due date. No email or hard copy is accepted.

**Problem 01 (30 points)**

Define a class named Document that contains a member variable of type string named text that stores any textual content for the document. Create a function named getText that returns the text field, a way to set this value, and an overloaded assignment operator.

Next, define a class for Email that is derived from Document and that includes member variables for the sender , recipient , and title of an e-mail message.

Implement appropriate accessor and mutator functions. The body of the e-mail message should be stored in the inherited variable text . Also overload the assignment operator for this class.

Similarly, define a class for File that is derived from Document and that includes a member variable for the pathname. Implement appropriate accessor and mutator functions for the pathname and overload the assignment operator.

Finally, create several sample objects of type Email and File in your main function.

Test your objects by passing them to the following subroutine, which will return true if the object contains the specified keyword in the text property.

bool ContainsKeyword( const Document& docObject, string keyword)

{

if (docObject.getText().find(keyword) != string::npos)

return true;

return false;

}

For example, you might test to see whether an e-mail message contains the text "c++" with the call ContainsKeyword(emailObj, "c++"); .

**Analysis:**

In this problem we will have to create the document class as a base class for the derived classes of email and file. In each class we will have to make default and parameter constructors for the classes and then accessor function that will return the individual variables of the class objects. We will also have to write mutator functions to change the variable values of each class object. Finally we will create overloading operator function that will change the values of the class object in regard to another object of the same class(i.e change a file class type with another file class type).

**Algorithm:**

1. Create the document class

Class Document

{

Public:

String getText();

Void setText(string text);

Document& operator=(const document& doc)

Private:

String text //private variable

}

1. Write implementation of functions
2. Create file class

Class File: public Document

{

Public:

String getPathName() cosnt;

Void setPathName(string pName);

File& operator =(const File& obj);

Private:

String pathName

}

1. Write implementation of functions
2. Create email class

Class Emal:public document

{

Public:

String getSender();

String getRecipient();

String getTitle();

Email& operator =(const Email& obj);

Void setSender(string send);

Void setRecipient (string recieve);

Void setTitle(string tittle);

Private:

String sender;

String reviever;

String title;

String text;

}

1. Write implementation of functions
2. Create the subroutine in the main function
3. Create class object of the file and email type and then test the functions of each class on the objects.

**Problem 02 (50 points)**

One problem with dynamic arrays is that once the array is created using the new operator the size cannot be changed. For example, you might want to add or delete entries from the array similar to the behavior of a vector. This project asks you to create a class called DynamicStringArray that includes member functions that allow it to emulate the behavior of a vector of strings.

The class should have the following:

* A private member variable called dynamicArray that references a dynamic array of type string.
* A private member variable called size that holds the number of entries in the array.
* A default constructor that sets the dynamic array to NULL and sets size to 0.
* A function that returns size .
* A function named addEntry that takes a string as input. The function should create a new dynamic array one element larger than dynamicArray , copy all elements from dynamicArray into the new array, add the new string onto the end of the new array, increment size, delete the old dynamicArray , and then set dynamicArray to the new array.
* A function named deleteEntry that takes a string as input. The function should search dynamicArray for the string. If not found, it returns false . If found, it creates a new dynamic array one element smaller than dynamicArray .It should copy all elements except the input string into the new array, delete dynamicArray , decrement size, and return true .
* Big three (a destructor, a copy constructor, and overload an assignment operator)
* Use a template so the implementation is not limited to strings. Test the class with dynamic arrays of integers in addition to strings.

**Analysis:**

For this problem we will create a dynamic string array class that will have two private variables of a dynamic array and int size. In the header we will create a default constructor that will initialize the array to null and the size to 0, and add a function that will return the size of the array. An add and delete function that will not only add or remove elements of the array but at the same modify the size to represent only the number of elements in the array will also be created. After that we will add a destructor, copy constructor and an overloaded assignment operator. After we create all the function we will turn the class into a template class and use it to test dynamic arrays of type string and int.

**Algorithm:**

1. Create the class header:

Class DynamicStringArray

{

Public:

DynamicStringArray();

Int getSize();

Void addEntry(string element);

Bool deleteEntry(string element);

String getEntry(int i);

dynamicStringArray(const DynamicStringArray& other);

DynamicStringArray operator = (const DynamicStringArray& other);

~DynamicStringArray;

Private:

String\* dynamicArray;

Int size

}

1. Create the add entry function:

Void dynamicStringArray::addEntry(string element)

{

String\* new\_array = new string[size +1];

Int I ;

For(i=0; i<size; i++)

{

new\_array[i]=dynamicArray[i];

}

New\_array[i]=element;

Size++;

Delete[] dynamicArray;

dynamicArray=new\_array;

}

1. Create the delete entry function:

Bool DynamicStringArray::deleteEntry(string element)

{

Int I;

For(i=0; i<size; i++)

{

If (dynamicArray[i]==element;

{

Break;

}

}

If (i==size)

{

Return false;

}

String\*new\_array=new string[size-1];

Int index = 0;

For (int j=0; j<size; j++)

{

If (dynamicArray[i] != element)

{

New\_array[index]=dynamicArray[i];

Index++;

}

}

Delete[] dynamicArray;

Size--;

dynamicArray=new\_array;

return true;

}

1. Create the copy constructor and destructor
2. Create the overloaded operator function.
3. Turn the DynamicStringArray class into a template class and test it with type int as well as type string.

**Extra Credit (40 points)**

**Problem 01** **(15 points)**

The following lists a Dice class that simulates rolling a die with a different number of sides. The default is a standard die with six sides. The rollTwoDice function simulates rolling two dice objects and returns the sum of their values. The srand function requires including

class Dice

{

public:

Dice();

Dice( int numSides);

virtual int rollDice() const;

protected:

int numSides;

};

Dice::Dice()

{

numSides = 6;

srand(time(NULL)); // *Seeds random number generator*

}

Dice::Dice(int numSides)

{

this->numSides = numSides;

srand(time(NULL)); *// Seeds random number generator*

}

int Dice::rollDice() const

{

return (rand() % numSides) + 1;

}

*// Take two dice objects, roll them, and return the sum*

int rollTwoDice(const Dice& die1, const Dice& die2)

{

return die1.rollDice() + die2.rollDice();

}

Write a main function that creates two Dice objects with a number of sides of your choosing. Invoke the rollTwoDice function in a loop that iterates ten times and verify that the functions are working as expected.

Next create your own class, LoadedDice , that is derived from Dice . Add a default constructor and a constructor that takes the number of sides as input. Override the rollDice function in LoadedDice so that with a 50% chance the function returns the largest number possible (i.e., numSides ), otherwise it returns what Dice’s rollDice function returns.

Test your class by replacing the Dice objects in main with LoadedDice objects.

You should not need to change anything else. There should be many more dice rolls with the highest possible value. Polymorphism results in LoadedDice’s rollDice function to be invoked instead of Dice’s rollDice function inside rollTwoDice .

**Analysis:**

In this problem we will first test the functions of the Dice class provided. After that we will create a loaded dice class that is derived from the dice class, in the new class the constructor will take the number of sides as a parameter. We will create the roll dice function in this class to return the largest number possible 50% of the time. Afterward we will replace the dice objects with loaded dice object and test the code.

**Algorithm:**

1. Create the dice header and implementation classes
2. Put the roll two dice function into the source file.
3. Create two dice objects using the default constructor
4. Implement the rolltwodice function with the two dice objects.
5. Create the loaded dice class as a derived class of dice:

Class LoadedDice:public Dice

{

Public:

LoadedDice();

LoadedDice(int numsides);

Virtual int rollDice() const;

Private:

Int numSides;

}

1. Implement the functions in the implementation file
2. Create two loaded dice objects in the main file and use the rollTwoDice function again but with the loaded dice.

**Problem 02** **(25 points)**

Create a class named Student that has three member variables:

name – A string that stores the name of the student

numClasses – An integer that tracks how many courses the student is currently enrolled in

classList – A dynamic array of strings used to store the names of the classes that the student is enrolled in

Write appropriate constructor(s), mutator, and accessor functions for the class along with the following:

* A function that inputs all values from the user, including the list of class names. This function will have to support input for an arbitrary number of classes.
* A function that outputs the name and list of all courses.
* A function that resets the number of classes to 0 and the classList to an empty list.
* An overloaded assignment operator that correctly makes a new copy of the list of courses.
* A destructor that releases all memory that has been allocated.
* A copy constructor that correctly makes a new copy of the list of courses.

Write a main function that tests all of your functions.

**Analysis:**

In this problem first we are going to have to create header class called student with three private variables: string name, int numClasses, and string\* ClassList. For public member function we will have the default, and copy constructor, destrucor, input and output functions, a function that will clear the array of the student object, an overloaded assignment operator. And then test this all in the main class.

**Algorithm:**

1. Create the student class:

Class student

{

Public:

Student();

~student();

Student(const student& list);

Void zeros();// this function will clear the memory of the classList array

Void input();

Void output();

Student& operator =(const student& list);

Private:

String name;

Int numclasses;

String \*classList;

}

1. Implement in the implementation file
2. Test in the main function making sure that you destruct objects when necessary.