AST21105 Object-Oriented Programming & Design

Lab 8 – Classes and Objects II

A. Submission Details

In this lab, you are required to submit **ONE** C++ program with **TWO** files to solve the given problem shown in the section "Date Class". To make the program implementation easier, you are suggested to write your program using Visual Studio .NET instead of doing it directly using paper & pencil. After you have completed the implementation, submit your program files (i.e. WordData.h and WordData.cpp for this lab) using the electronic "drop-box" in Canvas, **One week** after your lab section is conducted. For details, please refer to the following:

Tuesday sections	*by*	*19:00 on 15 March 2016 (Tuesday)*
Wednesday sections	*by*	*19:00 on 16 March 2016 (Wednesday)*

You are reminded to double-check your solution to verify everything is correct before submission. You are also required to put your name and student ID at the beginning of your source file.

Important: You are only able to submit your work once.

B. Objective

Constructors are special member functions used to initialize data members. The objective of this lab is to help you get familiar with the procedure on creating user-defined type that you have learned in class and perform initialization of data members using constructors. The first part of this lab is meant to be a simple review of constructors and some important member functions. While the second part of this lab is a practical task that allows you to use all the techniques introduced.

C. Review

1. What are constructors?

   C++ supports a more general mechanism for user-defined initialization of class object through special member function called constructor.

   Constructor should have:
   (1) The **same name** as the class.
   (2) **No return type** (not even void type)

   Note: Although constructors are class member functions, they are normally not being called explicitly. They will be called implicitly whenever an object of class is created.

2. Several types of constructor:

   a) Default constructor
   b) Type-conversion constructor
   c) Copy constructor
   d) Other constructors

   In this lab, we will only focus on the first two types.

3. Default constructor

   A default constructor is a constructor that is called with **NO** argument. The general form of default constructor is as follows:

   ```
   <class name>::<class name>( )
   ```

   where <class name> is the name of the **new type**.

   It is used to initialize an object with user-defined default values.

```
#ifndef POINT3D_H
#define POINT3D_H

class Point3D
{
    private:
        double _x;
        double _y;
        double _z;
    public:
        Point3D();
        // ...
};
#endif
```

Point3D.h

```
#include <iostream>
#include "Point3D.h"
using namespace std;

Point3D::Point3D() {
    _x = 1.2;
    _y = 2.4;
    _z = 3.6;
    cout << "Default cont is called"
    << endl;
}
// ...
```

Point3D.cpp

```
#include <iostream>
#include "Point3D.h"
using namespace std;

int main() {
    Point3D p;
    return 0;
}
```

main.cpp

Output:

```
Default const is called.
```

If there are NO user-defined constructors at all (i.e. without any constructor), the compiler will automatically generate the default constructor for you. (Note: But it does nothing for you.)

4. Default constructor should carry no argument. **Any exception**?

Yes, an exception will be one that can take arguments, provided they are given default values.

```
#ifndef POINT3D_H
#define POINT3D_H

class Point3D
{
    private:
        double _x;
        double _y;
        double _z;
    public:
        Point3D(double x = 1.2,
                double y = 2.4,
                double z = 3.6);
        // ...
};
```

Point3D.h

```
#include <iostream>
#include "Point3D.h"
using namespace std;

Point3D::Point3D(double x = 1.2,
                 double y = 2.4,
                 double z = 3.6) {
    _x = x;
    _y = y;
    _z = z;
    cout << "Default cont is called"
    << endl;
}
// ...
```

Point3D.cpp

5. Type conversion constructor

A type conversion is a constructor that is called with a **SINGLE ARGUMENT** specifies a conversion from its argument type to the type of its class.

```
<class name>::<class name>(<type> <variable name>)
```

where <class name> is the name of the **new type**. <type> is a data type other than the class type.

It is used to initialize an object with user-defined value.

```
#ifndef WORD_H
#define WORD_H

class Word
{
    private:
        string _str;
        int _frequency;
    public:
        Word(string s);
        // ...
};
#endif
```

Word.h

```
#include <iostream>
#include "Word.h"
using namespace std;

Word::Word(string s) {
    _str = s;
    _frequency = 1;
    cout << "Type-conversion const is
    called" << endl;
}
// ...
```

Word.cpp

c

```
#include <iostream>
#include "Word.h"
using namespace std;

int main()
{
    Word* w1 = new Word("Dave");
    Word w2("Peter");
    Word w3 = "Tom";
    return 0;
}
```

```
Type-conversion const is called
Type-conversion const is called
Type-conversion const is called
```

main.cpp

6. We know that a type-conversion constructor should take only single argument, which specifies a conversion from its argument type to the type of its class. Any exception?

   Yes, another example of a “type-conversion constructor” is one that can take more than one argument, provided there are given **default values except the first one**.

```
#ifndef WORD_H
#define WORD_H

class Word
{
    private:
        string _str;
        int _frequency;
    public:
        Word(string s, int k = 1);
        // ...
};
#endif
```

Word.h

```
#include <iostream>
#include "Word.h"
using namespace std;

Word::Word(string s, int k) {
    _str = s;
    _frequency = k;
    cout << "Type-conversion const is
    called" << endl;
}
// ...
```

Word.cpp

```
#include <iostream>
#include "Word.h"
using namespace std;

int main()
{
    Word w1("Hello");
    Word w2 = "Hello again";
    return 0;
}
```

```
Type-conversion const is called
Type-conversion const is called
```

main.cpp

D. Word Data Class

In this part, you are required to:

i) Create a New Project and give your project the name Lab8.

ii) In this lab, you are required to create a class **WordData** that has the following features.

A **WordData** object should be able to store a character string and a number of integers representing: the number of vowels, the number of consonants, the number of digits and the number of special characters in the character string. Also, it consists of a number of functions to manipulate the data.

The UML class diagram is given below:

WordData
- word : char* - vowels : int - consonants : int - digits : int - specialChars : int
+ WordData() + ~WordData() + setWord(const string& inWord) : void + getWord() const : string + displayData() const : void

where

- word: a data member to store a string represented in character string
- vowels: a data member to store the number of vowels in "word"
- consonants: a data member to store the number of consonants in "word"
- digits: a data member to store the number of digits in "word"
- specialChars: a data member to store the number of special characters in "word"

- WordData(): a default **constructor** that sets the word to **new char[1];** with **word[0] = '\0';** Also it sets all the remaining data members to 0.

  Note: '\0' is a character, which represents the end of a character string.

- ~WordData(): a **destructor**, which releases the memory allocated for "word"
  (More details about destructor will be given later in lectures!)
- setWord(const string& inWord): a mutator function that performs the following:
  a) Allocate memory space to store the string associated with inWord
     (You can use string member function "size()" to determine the length of inWord. Also, you are reminded to allocate memory space for the character '\0', which should be put at the end of the string.)
  b) Count the number of vowels in inWord (a, e, i, o, or u)
  c) Count the number of consonants in inWord (letters that are not vowels)
  d) Count the number of digits in inWord (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9)
  e) Count the number of "special characters" in inWord (characters that do not fall in any of the other categories)

  Note: The ASCII range of 'A' to 'Z' is 65 to 90, 'a' to 'z' is 97 to 122, range of '0' to '9' is 48 – 57.
- getWord(): an inspector function (a CONSTANT member function as well) that returns the character string as a C++ string.

  Example to convert a char* to string:

  ```
  string str(charString);
  // where str is in string type, charString in char* type
  ```

- displayData(): an output function (a CONSTANT member function as well) that outputs
  a) Character string "word"
  b) All the other data members

  in one line.

  Note: **TWO TAB** must be added after displaying the character array, **ONE TAB** in between each of the remaining data.

iii) Amain function (in main.cpp) has been provided for you to test out your class.

The following is the sample output.

```
Word           Vowels  Const.  Digits  Special
----           ------  ------  ------  -------
This           1       3       0       0
string         1       5       0       0
has            1       2       0       0
15             0       0       2       0
words.         1       4       0       1
It             1       1       0       0
has            1       2       0       0
lots           1       3       0       0
of             1       1       0       0
letters        2       5       0       0
and            1       2       0       0
very           1       3       0       0
few            1       2       0       0
special        3       4       0       0
chars!         1       4       0       1
```

Marking Scheme:

Graded items	Weighting
1. Correctness of program (i.e. whether your code is implemented in a way according to the requirements as specified.)	60%
2. Indentation	30%
3. Documentation (with reasonable amount of comments embedded in the code to enhance the readability.)	10%
	100%

Program Submission Checklist

Before submitting your work, please check the following items to see you have done a decent job.

Items to be checked	☑ / ☒
1. Did I put my name and student ID at the beginning of all the source files?	☐
2. Did I put reasonable amount of comments to describe my program?	☐
3. Are they all in .cpp extension and named according to the specification?	☐
4. Have I checked that all the submitted code are compliable and run without any errors?	☐
5. Did I zip my source files using Winzip / zip provided by Microsoft Windows? Also, did I check the zip file and see if it could be opened? ***(Only applicable if the work has to be submitted in zip format.)***	☐
6. Did I submit my lab assignment to Canvas?	☐

-End-