Assignment 2 CMPT135 Due Date: April 12, 11:55 pm

Any plagiarism/cheating results in zero marks for all the parties involved.

Question 1:

Background and Review

Encapsulation, inheritance and polymorphism are the main concepts of object-oriented programming and you are specifically being assessed on

1) understanding of encapsulation in the design of your classes;
2) understanding of inheritance;
3) understanding of polymorphism;
4) writing useful comments

Encapsulation allows:

1. The bundling of data and functions together
2. Access restrictions to certain components as defined by the programmer (you!)

Therefore, when you write a class, you place data members / variables (of type double, int, string, etc.) at the same place as you would place functions, constructors and destructors. You must think carefully what members should have what types of access levels and place them appropriately within the respective areas.

Inheritance is demonstrated through several related classes. In C++ (and OOP in general) the parent class is a more GENERAL version of a child class. This means, the child class is a more SPECIFIC version of the parent class. Always check the validity of a child class by using the "… is a …" test. Polymorphism is demonstrated through dynamic (or late) binding of child objects not yet known during compile time.

Your Tasks:

Write the following five classes:

Animal, Dog, Cat, PetDog, StrayDog.

These are the criteria for the above classes that you should attempt to fulfill:

1. They all have a name and an age each even though they need not be declared in every class
2. Hide the name and age members but provide getter and setter functions to access them.
3. They can all speak: The Dog barks and the Cat meows. (i.e., write a function called speak). However, the StrayDog's speak function overrides the speak function of Dog, and it should say "woof woof".
4. The PetDog performs an action: "fetch a stick"; while the StrayDog performs another action: "chase cars" (i.e., write a function called action)

Using inheritance, sensibly construct the family tree of the above five classes. You may add more class members (including variables, functions, constructors, destructors) as you see fit. For example:

birthday: The animal's age increases and "Happy Birthday!"

introduce: "Hello my name is … and I am … years old!"

Note the speak and action functions merely display some words on the console. Demonstrate encapsulation by writing appropriate functions and data members for the above classes with appropriate access restrictions. In your main() function:

i. declare an array of seven Animal objects;
ii. instantiate (create) at least one object from each of the five classes;
iii. assign the objects in (ii) as elements of the Animal array
iv. using a suitable loop, call the speak function of all Animal elements (but what happens?)
v. by way of polymorphism, what can you do differently So, that the speak task in question
iv. above yields the correct result corresponding to the actual Animal objects?

Assignment 2 CMPT135 Due Date: April 12, 11:55 pm
Any plagiarism/cheating results in zero marks for all the parties involved.

Question 2:

To answer this question, see the Question2_Program.txt file.

Question 3:

void findAndReplace(char *phrase, char find, char replace, int startPos)

Write a recursive function (**findAndReplace**) to find and replace characters in a character array. Recall that each recursive function call should be working on a sub-problem of the overall task. Thus, we need a way of accessing a sub-section of an array. There are multiple ways of doing this. Here's the recommended approach:

Previously when working with an array, we used the index 0 (array[0]) to access the first element of the array. To work with a sub-array, we will need a "starting position" parameter to use in place of 0 to indicate where the sub-array starts. In the provided function header, this is the startPos parameter.

As always, we need a way of knowing where the array ends. When we worked with arrays of integers, we used a size/length parameter to denote the number of elements in the array. When working with cstrings (as we are here) we can use the strlen function to determine the number of characters in the cstring.

Following this approach, your recursive function call should have the following parameters**: The character array**, **the find and replace values**, and **the starting index** of the subproblem.

Question 4.

C++ FILE I/O program implementation

You may program the below as individual programs or in OOP style using a class and methods.

Assume that you have a text file with 10 lines as shown below:

Line1
Line2
Line3
….
Line9
Line10

1. Write a program which will read the input file and write output to another file. Remember to manage opening file i/o errors.

2. Given the input text file, write a program which will double space the file. (an extra blank line between two lines)

Line1

Line2

Line3
….

Line9

Line10

3. Given the input file with text, write a program to read from the file, alternate the lines on the same file as shown below.

Line2
Line1
Line4
Line3
….
Line10
Line9

Question 5.

To answer this question, see the Question5_Program.txt file.

Question 6.

Mike, a student of CMPT135, wishes to implement the game of rock, paper and scissors on C++. However, he wants to modify the rules of the game slightly. He assigns each weapon (rock, paper or scissor) a numerical value of strength, which is decided by the user during runtime. Now, when facing each other, Rock's strength is temporarily doubled when fighting scissors, but halved when fighting paper. In the same way, paper has the advantage against rock, and scissors against paper. For example, if the strengths of rock, paper and scissor are 5, 7, and 24 respectively, then:
(i) When rock fights against paper, the rock's strength is halved, and the paper's strength is doubled. So, the rock's strength becomes 2 and the paper's strength becomes 14. Hence, paper wins the battle between the two in this case.
(ii) When paper fights against scissor, paper's strength is halved, and the scissor's strength is doubled. So, the paper's strength becomes 3 and scissor's strength becomes 48. Thus, scissor wins the battle between paper and scissor.
(iii) When scissor fights against rock, scissors' strength is halved, and rock's strength is doubled. So, the scissor's strength becomes 12 and rock's strength becomes 10.
Thus, scissor wins the battle between rock and scissor. After the battle, the original strength is restored.
To implement this, he makes use of a **class Weapon**, which has an int field called **strength**, and a char field called **type**. The Weapon class also contains the function **void setPower(int)**, which sets the strength for the 'Weapon'. Apart from this, there are two other functions **int return_strength()** and **char return_type()**, which return the strength and type of the object. Character type can be 'r' or 'R', 's' or 'S', 'p' or 'P' for each weapon. There is an additional int field called **modified**, whose usage will not be revealed for obvious reasons. (You are required to figure it out)
Further, he needs 3 more classes called **Rock**, **Paper** and **Scissors**, which inherit from Weapon. These classes will also need a public function **bool battle(Weapon)** that compares their strengths in the following way:
● Rock's strength is doubled (temporarily) when fighting scissors, but halved (temporarily) when fighting against paper

● In the same way, paper has the advantage against rock, and scissors against paper.

The strength field shouldn't change in the function, which returns true if the original class wins in strength and false otherwise.

Note:

You may use your own choice of visibility modes for the definition of the derived classes. The program outputs the results of the battles of rock with paper, paper with scissors and scissors with rock. Please note that the results of the battles are nothing but the return values of the function bool battle(Weapon). For example, if you want the result of the battle of rock with paper, you need to call the function as: r.battle(p) , where r is an object of class Rock and p is an object of class Paper

Here are some sample inputs along with their outputs for your reference. You may also use them to check the correctness of your program:

(i) Input: 5 7 24
Output: 001

(ii) Input:
3 4 5
Output: 000

Question 7.

A rational number is represented as *numerator/denominator*. We need to identify whether two rational numbers are equal or not. For example, consider two rational numbers '1/4' and '4/16'. Both are equal, as, '4/16' if reduced to lowest terms is '1/4'.

Consider the following structure 'rational' that has member variables 'numerator' and 'denominator'.

```
struct rational {
  int numerator;
  int denominator;
} ;
```

The main function is coded as shown below:

```
int main() {
  int result; rational num1, num2; cout << "Enter Details of Number 1 " << endl;
  cout << "Enter Numerator :" << endl; cin >> num1.numerator;
  cout << "Enter Denominator :" << endl; cin >> num1.denominator;
  cout << "Enter Details of Number 2 " << endl;
  cout << "Enter Numerator :" << endl; cin >> num2.numerator;
  cout << "Enter Denominator :" << endl; cin >> num2.denominator;
  result = isEqual(num1, num2);     //to check rational 'num1' is equal to rational 'num2'
  if(result == true)
    cout << "Both rational numbers are equal" << endl;
  else
    cout << "Both rational numbers are not equal" << endl;
```

```
  return 0;
}
```

You are required to write 2 functions:
To find out whether the two rational numbers are equal or not, we need to write two functions **'reduce'** and **'equal'**, as described below:

Function 1: reduce
This function has two parameters:

1. **rational *inputrational**: *inputrational is a pointer to the structure rational, which is the actual rational number to be reduced
2. **rational *outputrational**: *outputrational is a pointer to the structure rational, which will store the rational number in its lowest form

Function 2: isEqual
This function has two parameters:

1. **rational num1**:
2. **rational num2**:

The function should do the following

1. This function should call the function '**reduce**' twice.
2. Thereafter, the function should check whether both the rational numbers obtained in the lowest form are equal or not. If yes, then it should return boolean value 'true', else, it should a boolean value 'false'.

In the 'main' program, the function '**isEqual**' is called using two arguments, num1 and num2 which are passed by reference, both of type 'rational'

Question 8. (Credit for this question goes to Dr. Yonas)

In this question you are given the following classes fully implemented.
1. Shape: A C++ interface to represent Shape objects.

2. Triangle: A C++ class derived from the Shape interface to represent Triangle objects.

3. Rectangle: A C++ class derived from the Shape interface to represent Rectangle objects.

4. Circle: A C++ class derived from the shape interface to represent Circle objects.

5. Node: A C++ class with two member variables namely a pointer to Shape and a pointer to Node. Please note carefully that Node does not have Shape object member variable. Instead it has a pointer to a Shape as its member variable. The pointer to Shape member variable of any given Node may point to a Triangle object, a Rectangle object or a Circle object. This will help us achieve polymorphism as we traverse through the Nodes of a Linked List. Therefore, as we traverse through the Nodes of a Linked List, we can call the virtual member functions in the Shape interface in order to work with the actual Shape objects pointed to by the pointer to Shape member variables of the Nodes in the Linked List. In addition to the above classes, you are also given
6. LinkedList: A partially implemented C++ class with one member variable; namely a pointer to Node.
The Linked List class is simply a C++ class that represents a Linked List of Nodes. In order to get started, I have already implemented for the Linked List class the following member functions:
6.1. Default Constructor: Sets the head member variable to nullptr.

6.2. Destructor: Destructs the Linked List.

6.3. Printing Linked List: A member function to print the Shape objects in the Linked List.

6.4. Check Empty Linked List: A member function to test if a Linked List is Empty List.

6.5. Head Insert: A member function that inserts a new Node as a head node in the Linked List.

The aim of the question is for you to implement the following member functions of the Linked List class:

a. insert_grouped: This function takes a pointer to Shape as an argument. It must create a new Node, point the pointer to Shape member variable to the argument, and then insert the new Node in the Linked List such that Nodes of the same type of shapes are always grouped together in the Linked List. This means, you can insert the new Node either

i. At the head of the Linked List if the Linked List is empty, or

ii. Right before the first Node of the same type of Shape in the Linked List, or

iii. After any Node of the same type of shape in the Linked List, or

iv. At the tail of the Linked List if there is no Node of the same type of Shape in the Linked List.

Hint: The easiest approach is to insert the new Node right before the first Node of the same type in the Linked List. This approach will take care of all the four possible scenarios explained above.

b. insert_grouped_sorted: This function takes a pointer to Shape as an argument. It must create a new Node, point the pointer to Shape member variable to the argument, and finally insert the new Node in the Linked List such that Nodes of the same type of shapes are always grouped together in the Linked List. Moreover, within the same group of Nodes, the Nodes must be sorted in increasing order of the areas of the shapes.

How to Test Your Work

A test main program and the class files are provided for you in Question8_Program.txt. Copy and Paste the code on to your project and then implement the missing functions as described above.

Sample Output

A sample output shown below is provided to help you understand the problem statement. Please note that your output might be in different order; what is important is that the insert_grouped member function must insert objects such that same objects are grouped together; and the insert_grouped_sorted should insert objects such that same objects are grouped and within a group objects are sorted in increasing area.

Assignment 2 **CMPT135** **Due Date: April 12, 11:55 pm**

Any plagiarism/cheating results in zero marks for all the parties involved.

Sample Output

```
Inserting Triangle with head_insert member function.
Inserting Triangle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Circle with head_insert member function.
Inserting Circle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Triangle with head_insert member function.
Inserting Triangle with head_insert member function.
Inserting Circle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Circle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Triangle with head_insert member function.
Inserting Circle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Triangle with head_insert member function.
Inserting Rectangle with head_insert member function.
Inserting Circle with head_insert member function.

After inserting twenty nodes with head_insert member function, the Linked List is
Circle:      Perimeter = 6.40496   Area = 3.2662
Rectangle:   Perimeter = 15.1214   Area = 14.0831
Triangle:    Perimeter = 8.46098   Area = 1.70801
Rectangle:   Perimeter = 12.5496   Area = 9.75379
Circle:      Perimeter = 10.2036   Area = 8.28927
Triangle:    Perimeter = 9.73885   Area = 4.49202
Rectangle:   Perimeter = 13.7991   Area = 10.5926
Circle:      Perimeter = 19.3732   Area = 29.8822
Rectangle:   Perimeter = 10.1674   Area = 5.28554
Circle:      Perimeter = 7.548     Area = 4.53601
Triangle:    Perimeter = 11.351    Area = 5.90003
Triangle:    Perimeter = 11.0459   Area = 5.49519
Rectangle:   Perimeter = 10.3684   Area = 6.20457
Rectangle:   Perimeter = 13.7659   Area = 10.6472
Circle:      Perimeter = 29.5601   Area = 69.57
Circle:      Perimeter = 19.9482   Area = 31.6822
Rectangle:   Perimeter = 7.23692   Area = 2.73271
Rectangle:   Perimeter = 9.82855   Area = 5.43104
Triangle:    Perimeter = 10.4064   Area = 4.71268
Triangle:    Perimeter = 11.6928   Area = 6.15353

Deleting the Linked List
```

After deleting the Linked List, the Linked List is
Empty Linked List

Press any key to continue . . .
Inserting Triangle with insert_grouped member function.
Inserting Triangle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Circle with insert_grouped member function.
Inserting Circle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Triangle with insert_grouped member function.
Inserting Triangle with insert_grouped member function.
Inserting Circle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Circle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Triangle with insert_grouped member function.
Inserting Circle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Triangle with insert_grouped member function.
Inserting Rectangle with insert_grouped member function.
Inserting Circle with insert_grouped member function.

After inserting twenty nodes with insert_grouped member function, the Linked List is

Triangle:	Perimeter = 8.46098	Area = 1.70801
Triangle:	Perimeter = 9.73885	Area = 4.49202
Triangle:	Perimeter = 11.351	Area = 5.90003
Triangle:	Perimeter = 11.0459	Area = 5.49519
Triangle:	Perimeter = 10.4064	Area = 4.71268
Triangle:	Perimeter = 11.6928	Area = 6.15353
Rectangle:	Perimeter = 15.1214	Area = 14.0831
Rectangle:	Perimeter = 12.5496	Area = 9.75379
Rectangle:	Perimeter = 13.7991	Area = 10.5926
Rectangle:	Perimeter = 10.1674	Area = 5.28554
Rectangle:	Perimeter = 10.3684	Area = 6.20457
Rectangle:	Perimeter = 13.7659	Area = 10.6472
Rectangle:	Perimeter = 7.23692	Area = 2.73271
Rectangle:	Perimeter = 9.82855	Area = 5.43104
Circle:	Perimeter = 6.40496	Area = 3.2662
Circle:	Perimeter = 10.2036	Area = 8.28927
Circle:	Perimeter = 19.3732	Area = 29.8822
Circle:	Perimeter = 7.548	Area = 4.53601
Circle:	Perimeter = 29.5601	Area = 69.57
Circle:	Perimeter = 19.9482	Area = 31.6822

Assignment 2 CMPT135 Due Date: April 12, 11:55 pm

Any plagiarism/cheating results in zero marks for all the parties involved.

```
Deleting the Linked List

After deleting the Linked List, the Linked List is
Empty Linked List

Press any key to continue . . .
Inserting Triangle with insert_grouped_sorted member function.
Inserting Triangle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Triangle with insert_grouped_sorted member function.
Inserting Triangle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Triangle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Triangle with insert_grouped_sorted member function.
Inserting Rectangle with insert_grouped_sorted member function.
Inserting Circle with insert_grouped_sorted member function.

After inserting twenty nodes with insert_grouped_sorted member function, the Linked List is
Triangle:    Perimeter = 8.46098    Area = 1.70801
Triangle:    Perimeter = 9.73885    Area = 4.49202
Triangle:    Perimeter = 10.4064    Area = 4.71268
Triangle:    Perimeter = 11.0459    Area = 5.49519
Triangle:    Perimeter = 11.351     Area = 5.90003
Triangle:    Perimeter = 11.6928    Area = 6.15353
Rectangle:   Perimeter = 7.23692    Area = 2.73271
Rectangle:   Perimeter = 10.1674    Area = 5.28554
Rectangle:   Perimeter = 9.82855    Area = 5.43104
Rectangle:   Perimeter = 10.3684    Area = 6.20457
Rectangle:   Perimeter = 12.5496    Area = 9.75379
Rectangle:   Perimeter = 13.7991    Area = 10.5926
Rectangle:   Perimeter = 13.7659    Area = 10.6472
Rectangle:   Perimeter = 15.1214    Area = 14.0831
Circle:      Perimeter = 6.40496    Area = 3.2662
Circle:      Perimeter = 7.548      Area = 4.53601
Circle:      Perimeter = 10.2036    Area = 8.28927
```

Assignment 2 CMPT135 Due Date: April 12, 11:55 pm

Any plagiarism/cheating results in zero marks for all the parties involved.

```
Circle:     Perimeter = 19.3732   Area = 29.8822
Circle:     Perimeter = 19.9482   Area = 31.6822
Circle:     Perimeter = 29.5601   Area = 69.57

Deleting the Linked List

After deleting the Linked List, the Linked List is
Empty Linked List

Press any key to continue . . .
```