ID NO:		VENUE:		SEAT NO:	

MULTIMEDIA UNIVERSITY

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2016/2017

TCP1201 – OBJECT-ORIENTED PROGRAMMING AND DATA STRUCTURES
(All sections / Groups)

08 MARCH 2017
09:00a.m. –11:00 a.m.
(2 Hours)

Question	Mark
Total	

INSTRUCTIONS TO STUDENTS

1. This Question paper consists of 15 pages with 4 Questions only.
2. Attempt all **FOUR** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please write all your answers in **this Question Paper**.

Question 1

Using the Point class given, design and implement a Triangle class. Triangle has two private data members which are an array of three Point objects and a variable to store the perimeter of a Triangle object.

The main program declares and initializes a Triangle object and uses its member functions to get three points from the user, display them, calculate the perimeter value of the Triangle object and display the perimeter. If two points of a Triangle object overlap, an exception will be thrown and the main program will exit. Design and implement a PointOverlappedException class to support this exception. The sample runs of the main program are provided below.

Sample Run 1

```
Default constructor called. Points initialized.
Points: ( 0, 0 )( 1, 0 )( 1, 1 )
Perimeter = 3.41421

Enter 3 points in the following format:
x1 y1 x2 y2 x3 y3
0 0 3 0 3 4
Points: ( 0, 0 )( 3, 0 )( 3, 4 )
Perimeter = 12
```

Sample Run 2

```
Default constructor called. Points initialized.
Points: ( 0, 0 )( 1, 0 )( 1, 1 )
Perimeter = 3.41421

Enter 3 points in the following format:
x1 y1 x2 y2 x3 y3
1 1 2 3 1 1
Points overlapped exception. Exit program.
```

Main.cpp

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

int main()
{
  Triangle k;
  k.displayPoints();
  k.calcPerimeter();
  k.displayPerimeter();
  cout << endl;

  k.set3Points();
  k.displayPoints();
  k.calcPerimeter();
  k.displayPerimeter();

  return 0;
}
```

Continued...

Point.h

```
#ifndef POINT_H
#define POINT_H

class Point
{
  private :
    int x;
    int y;
  public :
    Point();
    void setX( int xPar );
    void setY( int yPar );
    int getX();
    int getY();
    void print();
    double distance( Point p );
};

#endif
```

Point.cpp

```
#include <iostream>
#include <cmath>
#include "Point.h"

using namespace std;

double Point::distance( Point p )
{
   double a;
   double b;
   a = x - p.x;
   b = y - p.y;
   return sqrt( a * a + b * b );
}

Point::Point()
  : x( 0 ), y( 0 )
{}

void Point::print()
{
  cout << "( " << x
       << ", " << y
       << " )";
  return;
}

void Point::setX( int xPar )
{
  x = xPar;
  return;
}
void Point::setY( int yPar )
{
  y = yPar;
  return;
}
int Point::getX()
{
  return x;
}
int Point::getY()
{
  return y;
```

Continued...

Triangle.h [3.5]

```
#ifndef TRIANGLE_H
#define TRIANGHT_H

#include "Point.h"
#include "PointsOverlappedException.h"

class Triangle
{

};

#endif
```

Triangle.cpp [5]

```
#include <iostream>
#include "Triangle.h"
#include "PointsOverlappedException.h"

using namespace std;
```

Continued...

Continued…

PointsOverlappedException.h **[0.5]**

```
#ifndef POINTS_OVERLAPPED_EXCEPTION_H
#define POINTS_OVERLAPPED_EXCEPTION_H

class PointsOverlappedException
{







};
#endif
```

PointsOverlappedException.cpp **[1]**

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

Continued...

Question 2

(a) How does **inheritance** support code reuse and make code easier to maintain? **[2]**

(b) The definition of a Patient class is given below:

```
class Patient {
  protected:
    string name;
    string NRIC;
  public:
    Patient(string name, string NRIC);
    string getName();
    string getNRIC();
    virtual void display();
};
```

(i) A panel patient is a specific type of patient where the medical fee is sponsored by the patient's employer. If you are asked to create a new class, `PanelPatient` to capture the information of a panel patient, what is the relationship between the Patient class above and the new `PanelPatient` class? **[0.5]**

Continued...

(ii) Provide the definition of the new `PanelPatient` class. The `PanelPatient` class should have all the attributes of a normal patient plus two additional attributes; (a) `employer`, the name of the employer, and (b) `maxSponsoredAmount`, the maximum amount sponsored by the employer. The `PanelPatient` class should contain a constructor method with initializer list that allows all attributes to be initialized and two query functions that retrieve the additional attributes stated above. It should also override the display function of the Patient class to display all its attributes.

Note: You do not need to provide the implementation for the class behaviors except for the constructor method. **[2.5]**

Continued...

(iii) Based on the above definition of the Patient and PanelPatient classes, the `main` program below is given but there are some errors and the program couldn't compile. Fix the errors and provide the correct `main` function.

```
int main() {
    Patient patients[] = { Patient p1("Ali", 1001),
          PanelPatient p2("Sabrina", 1002, "Intel", 200),
          Patient p3("Ken", 1003) }
    for (inti = 0; i< 3; i++) {
          patients->display();
  }
}
```

[2]

(iv) Provide an overloaded insertion (>>) operator for the `Patient` class inside the class to obtain values for the attributes of the `Patient` class. **[3]**

Continued...

Question 3

(a) Complete the class declaration below to declare a link-based implementation of the ADT (Abstract Data Type) list. The overloading of operators and exception handling is not required. The declaration must include the appropriate constructors and a destructor. Write comments to explain your code if needed. **[4]**

```
template<class ItemType>
class Node
{
private:
   ItemType        item; // A data item
   Node<ItemType>* next; // Pointer to next node

public:
   ...
   ...
}; // end Node

template<class ItemType>
class LinkedList
{
private:

   Node<ItemType>* headPtr;
   int itemCount;

   // Locates a specified node in this linked list.
   Node<ItemType>* getNodeAt(int position) const;

public:











}; // end LinkedList
```

Continued...

(b) Write a complete C++ program which uses a stack to determine whether a text string expression has balanced braces or not. You can use your own form of syntax and name of the member functions of a stack as long as they are close to the conventional names. Use comments to explain your code if the member functions cannot be easily understood. Error checking of input and exception handling is not required. **[5]**

Sample Run 1

```
Input a string: {abc{a}}
Result: The above string is balanced
```

Sample Run 2

```
Input a string: {abc{a}
Result: The above string is not balanced
```

```
#include <iostream>
#include "LinkedStack.h"

using namespace std;
```

Continued...

(c) State two uses of the ADT queue. **[1]**

Continued…

QUESTION 4

(a) Write a recursive function search() which searches a C-string for the number of occurrence of a character. The main() function will display 3 as there are three i's in the C-string "This is logical." **[5]**

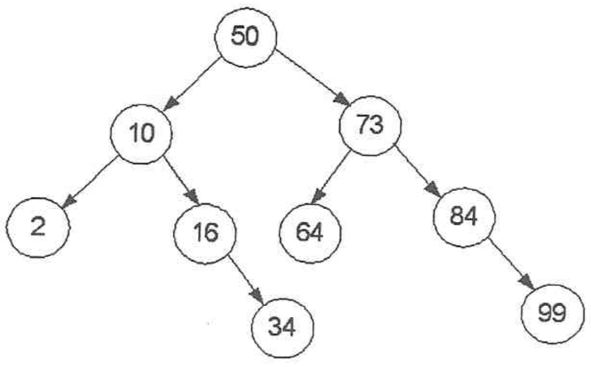
```
#include <iostream>
using namespace std;

// x is the character to search for.
// str is the c-string array.

// i is the index of array element to search from.
// i == 0 means searching from str[0].

int search(char x, char str[], int i)
{   // write your code here




}

int main()
{
     char a[ 25 ] = "This is logical.";
    cout << search('i', a, 0) << endl;
}
```

Continued...

(b) TreeNode is a pointer-baser representation for the node of a binary tree. Assuming that an integer is the item stored in each node, write C++ code to declare the data members of TreeNode. **[3]**

```
class TreeNode
{
      public:
           // ignore member functions
          ...
      private:
          // declare data members below

          // write your code here
















};
```

(c) Given a binary search tree below, show the resulting tree at each step if 99, 16, 50 and 10 are deleted one after another. You answer consists of four diagrams. Other possible solutions are to be accepted. **[2]**



End of paper