UNIVERSITY OF SWAZILAND

FACULTY OF SCIENCE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

PROGRAMMING TECHNIQUES II

COURSE CODE – EE272

SUPPLIMENTARY EXAMINATION

JULY 2012

DURATION OF THE EXAMINATION - 3 HOURS

INSTRUCTIONS TO CANDIDATES

1. There are **FIVE** questions in this paper. Answer question **1**, and any other **THREE** questions.
2. Each question carries equal marks.
3. Show all your steps clearly in any calculations.
4. State clearly any assumptions made.
5. Start each new question on a fresh page.

Question 1

(a) Information hiding is one of the key features that distinguish object-oriented programming from structured programming. Using an example, explain the rationale of information hiding and how it relates to the following object-oriented programming concepts: *abstraction*, *coupling*, and *cohesion*.

[4]

(b) Explain the following Object-Oriented programming terms:

a. Polymorphism [2]
b. Class Constructor [2]
c. Interface [2]

(c) Discuss the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors. [4]

(d) Describe the ways by which a derived class may inherit from a base class. [6]

(e) Explain the advantage of separating interface from implementation of a class. [2]

(f) Explain the difference between local variable and a data member? [1]

(g) What is the difference between *overriding* and *redefining* a function. [2]

Question 2

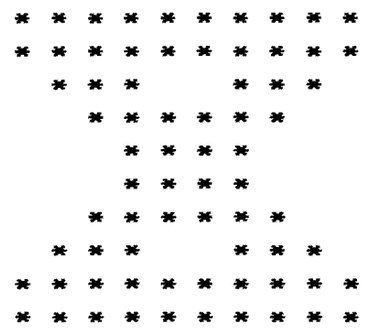
Declare a class named *Triple* with three private data members (floats) x, y, and z. Provide public functions for setting and getting values of all the private data members. Define a constructor that initializes the values to user-specified values or, by default, sets the values all equal to 0. Also overload the following operators:

i. *Addition* (+) so that corresponding elements are added together
ii. *Output* (<<) so that it displays the Triple in the form "The triple is (x,y,z)"
iii. *Assignment* (=) that copies x to z, y to x, and z to y.
iv. *Post-increment* (++) so that x and z are increased by one each.
v. *Input* (>>) such that it is possible to input a Triple in the form:
(x, y, z) e.g. (3, 2, 5)

[25]

Question 3

(a) Using **only** *recursive* functions (NO repetition statements), write a C++ program that displays the following checkerboard pattern: [15]



Your program must use **only** three output statements, one (or more) of each of the following forms:

```
cout << "* ";
cout << " ";
cout << endln;
```

(b) Carefully analyse the program shown in Figure 5 and determine its output. Show all working. [10]

```
#include <iostream>

using namespace std;

int main(void) {

int m, n;
        int x = 1, y = 10;

        for(m=x; m<=y; m++) {
                for(n=x; n<=y; n++) {
                        if((m<=x+1)||(m>=y-1)) {
                              printf("* ");
                        } else {
                          ((y-n+1>=m-1)&&(y-n+1<=(m+1))) ? cout<<"* ":cout<< "  ";
                        }
                }
                cout << endln;
        }

        return 0;
}
```

Question 4

Create a class *HugeInteger* that uses a 40-element array of digits to store integers as large as 40 digits each. Provide the following members functions for the class.

(a) Input and Output member functions:

(i) *Input*: reads the digits of a HugeInteger object. [3]

(ii) *Output*: writes out the digits of a HugeInteger object. [1]

(b) Arithmetic member functions:

(i) *Add*: to calculate the sum of two HugeInteger objects. [5]

(ii) *Subtract*: to calculate the difference between two HugeInteger objects.

[5]

(c) Member functions for comparing *HugeInteger* objects:

(i) *isEqualTo:* returns TRUE if a *HugeInteger* object is greater than or equal to another *HugeInteger* object. Returns FALSE otherwise. [2]

(ii) *isNotEqualTo:* returns TRUE if a *HugeInteger* object is NOT equal to another HugeInteger object. Returns FALSE otherwise. [1]

(iii) *isGreaterThan:* returns TRUE if a *HugeInteger* object is greater than another *HugeInteger* object. Returns FALSE otherwise. [2]

(iv) *isLessThan*: returns TRUE if a *HugeInteger* object is less than another *HugeInteger* object. Returns FALSE otherwise. [2]

(v) *isGreaterThanOrEqualTo:* returns TRUE if a *HugeInteger* object is greater than or equal to another *HugeInteger* object. Returns FALSE otherwise. [1]

(vi) *isLessThanOrEqualTo:* returns TRUE if a *HugeInteger* object is less than or equal to another *HugeInteger* object. Returns FALSE otherwise. [1]

(vii) *isZero:* returns TRUE if a *HugeInteger* is equal to 0. Returns FALSE otherwise . [2]

Question 5

Create a class called `Invoice` that a hardware store might use to represent an invoice for an item sold at the store. An `Invoice` should include four pieces of information as data members – a part number (type string), a part name (type string), a quantity of the item being purchased (type double), a price per item (type double), and a value added tax (VAT) in percent (type double). Your class should have a constructor that initialises the four data members. Provide set and get functions for each data member. In addition, provide a member function named *InvoiceAmount* that calculates the invoice amount (i.e. multiplies the quantity by the price per item and then add the VAT), then returns the amount as a double value. If the quantity is not positive it should be set to 0.0. If the price per item is not positive it should be set to 0.0.

(i) Write an interface and implementation for this class. [20]

(ii) Write a test program that creates two Invoice objects. The first object represent the purchase of two shovels (part # SH00101), each costing E250. The second object represent the purchase of nine bags of cement (part # CM01012), each costing E65. Assume that the current VAT set by the government is 14%. Your test program should print the values of each invoice. [5]

END OF PAPER