UNIVERSITY OF SWAZILAND

FACULTY OF SCIENCE & ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

SUPPLEMENTARY EXAMINATION

JULY 2013

PROGRAMMING TECHNIQUES II

COURSE CODE – EE272

DURATION - 3 HOURS

INSTRUCTIONS TO CANDIDATES

(a) There are **FIVE** questions in this paper. Answer questions **1** & **2**, and any other **TWO** questions.

(b) Each question carries equal marks.

(c) Show all your steps clearly in any calculations.

(d) State clearly any assumptions made.

(e) Start each new question on a fresh page.

Question 1

(a) What is polymorphism? [4]

(b) What is inheritance? [2]

(c) How does polymorphism support inheritance? [3]

(d) How is overriding related to polymorphism? [4]

(e) Discuss how polymorphism makes software systems extensible and maintainable? [5]

(f) What is the difference between an object and a class? [3]

(g) What are constructors and how are they defined? [4]

Question 2

(a) Using an example, explain where you would use a unary scope resolution operator. [2]

(b) In object-oriented programming it is recommended that we should separate interface from implementation. Explain the reason for this. [3]

(c) Explain the difference between the use of the dot selection operator(.) and the arrow member selection operator (->). [3]

(d) What is a friend function of a class? [2]

(e) What is a static class member? [4]

(f) Why is it that static class members do not have the *this* pointer? [2]

(g) Discuss four restrictions on operator overloading in C++? [4]

(h) Explain the following object-oriented terms: abstract class, base class, and a derived class. [5]

Question 3

Analyse the following THREE programs and determine their outputs.

(a) Program 1 [3]

Class Interface

```
#pragma once

class DemoProg1 {
      public:
            DemoProg1(void);
            ~DemoProg1(void);
};
```

Class Implementation

```
#include "DemoProg1.h"
#include <iostream>

using namespace std;

DemoProg1::DemoProg1(void){
      int k, num=30;
      k = (num>5 ? (num <=10 ? 100 : 200): 500);
      cout << num << endl;
}

DemoProg1::~DemoProg1(void){
}

int main(void) {
      DemoProg1 dp1;
      return(0);
}
```

(b) Program 2 [12]

Class Interface

```
#pragma once

class DemoProg2 {
      public:
            DemoProg2(void);
            ~DemoProg2(void);
};
```

Class Implementation

```
#include "DemoProg2.h"
#include <iostream>

using namespace std;

DemoProg2::DemoProg2(void){
      char c=48;
```

```
        int i, mask=01,value;

        for(i=1; i<=4; i++){
                value = c|mask;
                cout << value << endl;
                mask = mask<<1;
        }
}

DemoProg2::~DemoProg2(void){
}

int main(void) {
        DemoProg2 dp2;
        return(0);
}
```

(c) **Program 3** [10]

Class Interface

```
#pragma once

class DemoProg3 {
        public:
                DemoProg3(void);
                ~DemoProg3(void);
};
```

Class Implementation

```
#include "DemoProg3.h"
#include <iostream>

using namespace std;

DemoProg3::DemoProg3(void){
        int i=4, j=8, value1, value2, value3;
        value1 = i|j&j|i;
        value2 = (i<<1)&j|j&i;
        value3 = i^j;
        cout << value1 << ", " << value2 << ", " << value3;
        cout << endl;
}

DemoProg3::~DemoProg3(void){
}

int main(void) {
        DemoProg3 dp3;
        return(0);
}
```

Question 4

A college administrator requires a program that reads in test scores and applies two different curves to them. The program should contain a base class *ScoreBank* with two private data members: an integer array for the scores and a float for the average. The maximum number of scores is 10. The class should contain a method *EnterScores*

which asks the user how many test scores are needed and reads in the scores. The class should also contain a method *CalcAverage* which stores the average of the entered scores in the private float data member. Scorebank should also have an *Output* function that prints a sorted list of test scores to the screen as well as the average.

Derive from *ScoreBank* a class called *Curve1* which contains a method *Curve*. This curve sets the average score to 75, finds out how far away from 75 the actual average is, and then add this value to each test score. Overload the *Output* method to print, sorted, the original scores and the curved scores as well as the original and new average.

Derive from ScoreBank a class called *Curve2* which contains a method *Curve*. This curve sets the highest score to 100. The method then finds out how is the highest score from 100 and then adds the difference to each score. Overload the *Output* function to print the original scores, the new scores, and the averages for both sets.

(i) Write the interfaces of each of the three classes. [6]

(ii) Write the implementations of the classes. [19]

Question 5

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]

END OF PAPER