UNIVERSITY OF SWAZILAND

FACULTY OF SCIENCE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

PROGRAMMING TECHNIQUES II

COURSE CODE – EE272

MAIN EXAMINATION

MAY 2012

DURATION OF THE EXAMINATION - 3 HOURS

INSTRUCTIONS TO CANDIDATES

1. There are **FIVE** questions in this paper. Answer questions **1** & **2**, and any other **TWO** 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) Explain how polymorphism promotes extensibility of software design? [4]

b) A bank requires a software system for managing customers' bank *accounts*. All customers at this bank can deposit into and withdraw from their accounts. The bank has two types of accounts: *savings* and *current*. A savings account earns interest on the money held meanwhile a current account charges a fee per transaction. All accounts have methods for *crediting*, *debiting*, and *retrieving* account balances. Savings accounts keep information on interest rates and provide a way of computing interest earned. On the other hand current accounts keep information on the *fee charged per transaction*. Current accounts refine the general functionality of crediting and debiting all accounts so that whenever a transaction is performed successfully a transaction fee is deducted from the account balance.

From the description of the proposed banking application problem above, give a detailed account of *why* and *how* the programming techniques of *inheritance* and *polymorphism* could be especially effective for solving a problem of this nature. [21]

Question 2

a)

(i) How is it that polymorphism enables programming "in the general" rather that "in the specific"? [1]

(ii) Discuss two advantages of programming "in the general". [3]

b)

(i) Discuss two problems of programming with the `switch` logic. [2]

(ii) Using an example, explain how polymorphism can be an effective alternative to switch logic. [2]

c) 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]

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

e) Describe the ways by which a derived class may inherit from a base class. [5]

f) Using an example, explain the relationship between function templates and function overloading? [4]

Question 3

Analyse the following programs and determine their outputs. Show all working.

(a)

```
#include <iostream>

using namespace std;
using namespace System;

int main(void){
	int i, j, c = 9, m, k;
	for (i = 1; i <= 5; i++) {
		for (k = 1; k <= c; k++) {
			cout << " ";
		}
		for (j = 1; j <= i; j++) {
			cout << j;
		}
		for (m = j - 2; m > 0; m--) {
			cout << m;
		}
		cout << endl;
		c = c - 2;
	}
	Console::ReadKey();
    return 0;
}
```

[9]

(b)

```
#include <iostream>

using namespace std;

int main(){

	char prnt = '*';
	int i, j, k, s, nos = -1;

	for (i = 5; i >= 1; i--) {
		for (j = 1; j <= i; j++) {
			cout << " ";
		}
		for (s = nos; s >= 1; s--) {
			cout << prnt;
		}
		for (k = 1; k <= i; k++) {
			if (i == 5 && k == 5) {
			continue;
			}
			cout << " ";
		}
		nos = nos + 2;
		cout << endl;
	}
	nos = 5;
	for (i = 2; i <= 5; i++) {
		for (j = 1; j <= i; j++) {
			cout << prnt;
		}
		for (s = nos; s >= 1; s--) {
			cout << " ";
		}
		for (k = 1; k <= i; k++) {
			if (i == 5 && k == 5) {
				break;
```

```
                }
                cout << prnt;
            }
            nos = nos - 2;
            cout << endl;
        }
    return 0;
}
```

[16]

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

Package-delivery services, such as FedEx, DHL, and UPS, offer a number of different shipping options, each with specific costs associated.

Create an inheritance hierarchy in the form of a class diagram to represent the various types of packages. Use `Package` as the base class of the hierarchy, then include classes `TwoDayPackage` and Overnight that derive from `Package`. Base class `Package` should include data members representing the *name*, *address*, *city*, and *region* for both the sender and the recipient of the package, in addition to data members that store the *weight* (in kilograms) and *cost per kilogram* to ship the package. `Package`'s constructor should initialise these data members. Ensure that the weight and cost per kilogram contain positive values.

`Package` should provide a public member function `calculateCost` that returns a double indicating the cost associated with shipping the package. `Package`'s `calculateCost` function should determine the cost by multiplying the weight by the cost per kilogram. Derived class `TwoDayPackage` should inherit the functionality of base class `Package`, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. `TwoDayPackage`'s constructor should receive a value to initialise this data member. `TwoDayPackage` should redefine member function `calculateCost` so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class `Package`'s `calculateCost` function.

Class `OverNightPackage` should inherit directly from class `Package` and contain an additional data member representing an additional fee per kilogram charged for overnight-delivery service. `OverNightPackage` should redefine member function `calculateCost` so that it adds the additional fee per kilogram to the standard cost per kilogram before calculating the shipping cost.

(i) Draw a class diagram depicting the three classes and their relationship. [3]
(ii) Write the C++ interface of each class. [6]
(iii) Write the C++ implementation of each class. [12]
(iv) Write a C++ program that creates objects of each type of package and tests their member function `calculateCost`. [4]

END OF PAPER