

**COS1512
RCO1512**

May/June 2017

INTRODUCTION TO PROGRAMMING II

Duration 2 Hours

75 Marks

EXAMINERS :

FIRST

SECOND

DR MA SCHOEMAN

MS A THOMAS

Closed book examination

This examination question paper remains the property of the University of South Africa and may not be removed from the examination venue

This paper consists of 8 pages and 6 questions
Please make sure that you have all 8 pages with the 6 questions

Instructions:

- Answer all the questions
- Do all rough work in the answer book
- The mark for each question is given in brackets next to the question
- Please answer the questions in the correct order. If you want to do a question later, leave enough space
- Number your answers and label your rough work clearly
- Marks are awarded for part of an answer, so do whatever you are able to in each question

GOOD LUCK!

TURN OVER

QUESTION 1**[5]**

- 1 1 What is the difference between the following two declarations (2)
 char myName[] = "Peter",
 char yourName[] = {'P', 'e', 't', 'e', 'r'},
- 1 2 Consider the following incomplete code fragment and answer the questions below
- ```

1 vector<int> number(4),
2 number[0] = 3,
3 for (int k = 1, k < number.size(), k++)
4 {
5 number[k] = 2 + number[k - 1],
6 cout << number[k] << " ",
7 }
8
9 cout << number[number.size()-1],
```
- 1 2 1 Explain in one sentence what happens in line 1 (1)
- 1 2 2 Give the C++ statement to add another element to `number` with the value of 11 in line 8 (1)
- 1 2 3 What will the output of the code fragment be after including the code in line 8 as indicated in 1 2 2? (1)

**QUESTION 2****[4]**

Refer to the code below to answer the questions that follow

- 2 1 Give the output of the program below using the following input (2)  
       Hello
- 2 1 What does function `r()` do? (1)
- 2 3 What will happen if the `' '` after the word `Hello` is omitted? (1)

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

```
void r() //recursive function
{
 char c,
 cin.get(c),
 if (c != ' ')
 r(),
 cout << c,
 return,
```

TURN OVER

```

}
int main()
{
 cout << "Please enter a sentence that ends with a full stop. "
 << endl,
 r(),
 cout << endl,
 return 0;
}

```

**QUESTION 3****[7]**

3 1 What is a dangling pointer? (1)

3 2 Which of the following statement(s) is/are problematic and why? (1)

```

1. int *p1, p2,
2 p1 = new int,
3 *p1 = 25,
4 p2 = p1,

```

3 3 Write a statement or statements to correct the problem(s) in question 3 2 (1)

3 4 Consider the following code which is assumed to be embedded in a correct and complete C++ program

```

1 int array_size = 10,
2 int *a,
3 a = new int [array_size],
4 cout << &a,

```

3 4 1 Describe the action of the new operator. What does the new operator return? (2)

3 4 2 Write code to return the memory allocated to a, back to freestore (1)

3 4 3 What will the statement in line 4 display? (1)

**QUESTION 4****[34]**

Note Read through the whole of Question 4 below before you attempt to answer the questions that follow

The main program in this question is used to manage an user's shareholdings. The program inputs a new purchase of shares, checks a file of shares `Portfolio.dat` to see whether the user already owns shares in the same company, and if so, updates the shareholding for that company. This produces an updated version of the file of shares `NewPortfolio.dat`.

TURN OVER

At the same time, the total value of the shares is determined. For each share, the last five purchase values per share are also kept in an array as part of a `Shares` object.

The program uses a class `Shares` that represents shares for a shareholder in a specific company on the JSE. This class has four member variables:

- `company`, a string that holds the name of the company
- `nrShares`, an integer value that indicates the number of shares this shareholder owns
- `unitValue`, a float value that indicates the current value of one share and
- `pastUnitValues`, an array to keep five double values

In addition, the class has the following member functions:

- A default constructor that initializes `company` to an empty string. `nrShares` and `unitValue` should each be initialized to 0. All elements in `pastUnitValues` should be initialized to 0.
- An overloaded constructor that represents shares bought in a new company and sets `company`, `nrShares` and `unitValue` to specified values, while all elements in `pastUnitValues` are initialized to 0.
- A destructor that does not perform any action.
- Accessor functions for member variables `nrShares` and `unitValues`.
- An overloaded equality operator `==` to compare two `Shares` objects. The `==` operator is implemented as a **friend** function with the following prototype:  

```
bool operator==(const Shares & s1, const Shares & s2)
```

 This function returns `true` if `shares1` and `shares2` have the same company name, and `false` if not.
- A member function `updateShares()` that adds the number of shares bought to `nrShares`, update the value of `unitValue` and also update `pastUnitValues` to include the unit value of the current purchase as the last element in array `pastUnitValues`. Hint: move all elements in `pastUnitValues` one position to the left to add the current `unitValue` as the last element. Use the following prototype:  

```
void updateShares(const Shares & s1),
```
- An overloaded extraction operator `>>` (implemented as a **friend** function) so that it can be used to input values of type `Shares`.
- An overloaded insertion operator `<<` (implemented as a **friend** function) that outputs all the member variables of a `Shares` object.

4.1 Create the header file `Shares.h` that contains the `Shares` class specification (9)

4.2 Create the implementation of the class `Shares` including all the **friend** functions **except** the overloaded insertion operator `<<` (16)

4.3 Complete the application program (`main()`) below by citing the number and writing down the missing statement(s). This program obtains the detail for a new purchase of shares from the user and creates an object to represent the new purchase of shares. It then extracts all existing shares in the user's portfolio one by one from a file `Portfolio.dat` and checks whether the shareholder already has shares in the

company If the user already holds shares in the company, the member function `updateShares()` is used to update the shares in the company to include the new shares purchased. All shares (updated as well as unchanged existing shares) are output to a new portfolio file called `NewPortfolio.dat`. If the user does not already own shares in the company, the new purchase of shares are simply output to `NewPortfolio.dat`. Before outputting a shareholding to `NewPortfolio.dat`, the program calculates the value of the shareholding (number of shares \* unit value per share) and adds it to a running total. The running total is used to calculate the total value of the shares, which is displayed on the console window at the end of the program. Be sure to use appropriate member functions in the required statements (9)

```
#include <iostream>
#include <cstdlib>
#include <iomanip>

_____1_____ //1.Include files needed
using namespace std;

int main()
{
_____2_____ //2.Declare input and output files

_____3_____ //3.Open the file Portfolio.dat and check
// that the file exists NB ½ mark

_____4_____ //4.Open the file NewPortfolio.dat and
// check that the file exists NB ½ mark

//Add a new purchase of shares
string companyBought,
int nrBought,
double shareValue,

cout << "Enter name of company in which shares are bought: ",
cin >> companyBought,
cout << "Enter number of shares bought ",
cin >> nrBought,
cout << "Enter unit value of one share. ",
cin >> shareValue,

_____5_____ //5.Instantiate object newShare for the
// new share bought
Shares aShare, //instantiate object aShare to extract an
//existing share from file Portfolio.dat
double totalSharesValue = 0,
```

```

while (_____ 6 _____) //6.Extract an existing share from file
 // Portfolio.dat
{
 if (_____ 7 _____) //7.Compare newShare with aShare to
 //check whether shareholder already
 //owns shares in the company
 {
 _____ 8 _____ //8.Update shares to include newly
 //bought shares
 }

 cout setf(10s showpoint),
 cout.setf(10s fixed),
 cout << setprecision(2),

 totalSharesValue += _____ 9 _____ //9. Calculate and add
 // current value of shares for this company
 outfile << aShare, //Output share to file NewPortfolio.dat
}

cout << "\nTotal value of shares R " << totalSharesValue << endl
 << endl,
 _____ 10 _____ //10.close files
return 0;
}

```

**QUESTION 5****[14]**

Consider the class specification (interface) for the class BankAccount below

```

class BankAccount
{
public
 BankAccount(),
 BankAccount(int accNr, double initBalance),
 void setAccNr(int nr),
 int getAccNr() const,
 double getBalance() const,
 void deposit(double amnt),
 void withdraw(double amnt);
 string obtainAccInfo() const, //return a string with accNr and
 // balance
private
 int accNr,
 double balance,
}

```

TURN OVER

- 5 1 Derive a class `SavingsAccount` from the class `BankAccount`. This class has an additional member variable `interestRate` representing the interest rate on the savings account. The class also has additional member functions `setInterestRate()`, `getInterestRate()` and `postInterestRate()`. Member function `postInterestRate()` adds earned interest to the balance. The class should override member function `displayAccInfo()` to return a string with the account number, current balance and interest rate.

Provide only the interface of the class `SavingsAccount` in terms of a header file. The header file should contain compiler directives to prevent multiple definitions. Assume that the interface of class `BankAccount` is contained in an interface file called `BankAccount.h`. (9)

- 5 2 Implement the overloaded constructor for the class `SavingsAccount` by invoking the base class constructor. (3)
- 5 3 Member function `postInterestRate()` adds earned interest to the balance. Show how to adapt class `BankAccount` to allow `postInterestRate()` to access the member variable `balance` by name and explain why the adaptation is done in this way. (2)

## QUESTION 6

[11]

Consider the following class `Data` that keeps statistics for rainfall figures.

```
class Data
{
public:
 Data(),
 double calcAverage(),
 void captureData(double theData[]);
 double findMin(),
 double findMax(),
private:
 vector< double > theData,
},
```

The class `Data` has the following operations:

`captureData` – reads the rainfall figures into `theData`  
`calcAverage` – calculates and returns the average of the rainfall figures in `theData`  
`findMin` – finds and returns the minimum rainfall figure  
`findMax` – finds and returns the maximum rainfall figure

- 6 1 Write a template version of the `Data` class. In other words, redesign the `Data` interface so that it may be used to hold data of any numerical type. For example, it must be able to hold objects of type `int` or `long`. Provide only the interface. (5)

TURN OVER

- 6.2 Implement the `calcAverage()` function of the template class `Data`. (5)
- 6.3 Provide a declaration for a `Data` object intended to contain objects of class `Prices`. (1)