Tutorial letter 202/2/2017

Introduction to Programming II
COS1512

Semester 2

School of Computing

This tutorial letter contains the solution to Assignment 2

IMPORTANT INFORMATION:

Please activate your *myUnisa* and *my*Life email addresses and ensure you have regular access to the *my*Unisa module site COS1512-2017-S1 as well as your e-tutor group site.

Due to regulatory requirements imposed by the Department of National Education the following apply:

To be considered for examination admission in COS1512, a student must meet the following requirement:
Submit assignment 1 or assignment 2 BEFORE 8 September 2017.

Note: This is a blended online module, and therefore your module is available on myUnisa. However, in order to support you in your learning process, you will also receive some study materials in printed format. Please visit the COS1512 course website on myUnisa at least twice a week.

Content

1. Introduction

The purpose of this tutorial letter is to supply the solution for Assignment 2, and to indicate how you should interpret the results you obtained for this assignment. This assignment covers the work discussed in Chapters 10, 11 and 12 of the Study Guide (Tutorial Letter 102), as well as the relevant sections in Chapters 10, 11 and 12, and Appendices 7 and 8 of Savitch. Note that you should have included the input and output of all programs you were required to write.

The assessment and feedback given on your assignment, serve as an indication of your mastering of the study material.
If you received a good mark, you can be confident that you are on the right track. If you did not receive a good mark, it is an indication that you need to revise your study method for COS1512.

2. Tutorial matter distributed to date

DISK 2017	Prescribed software
COS1512/**101**/3/2017	First tutorial letter: General information, study programme, exam admission and assignments
COS1512/**102**/2/2017	Study Guide
COS1512/**103**/2/2017	How to create an assignment as a PDF file
COS1512/**201**/2/2017	Solution to Assignment 1
COS1512/**202**/2/2017	This letter: Solution to Assignment 2

If you have not received all of the above-mentioned tutorial matter, please download it from myUnisa at https://my.unisa.ac.za.

3. Allocation of marks

When we mark assignments, we comment on your answers. Many students make the same mistakes and consequently we discuss general problems in the tutorial letters. It is, therefore, important to work through the tutorial letters and to make sure you understand our solutions and where you went wrong.

The maximum number of marks you could obtain for Assignment 2 is 80. This is converted to a percentage. If you for instance obtained 40 marks for Assignment 2, you received 50% for Assignment 2. This percentage in turn contributes a weight of 80% to the year mark, as can be seen in the summary of the weights allocated to the assignments for COS1512 below.

Assignment number	Weight
1	20
2	80
3	0

Questions 2 and 4 have not been marked. However, 5 marks are awarded for question 2 and 10 marks for question 4 if you *attempted* questions 2 and 4 (note that this is not the way in which questions will be marked in the examination!). We include complete solutions for *all* questions.

The marks you received for question 1 were determined on the following basis:

Question not done	0/5
Question attempted, but the program does not work at all	1/5
A good attempt, but there are a few problems with your answer	3/5
The program works correctly and produces the correct output	5/5

For question 3(a), a maximum of 2 marks were awarded and for question 3(b), a maximum of 3 marks, i.e. 5 marks in total.

The marks you received for questions 5 and 6 was determined on the following basis:

Question not done	0/20
Question attempted, but the program does not work at all	7/20
A good attempt, but there are a few problems with your answer	15/20
The program works correctly and produces the correct output	20/20

The marks you received for question 7 was determined on the following basis:

Question not done	0/15
Question attempted, but the program does not work at all	5/15
A good attempt, but there are a few problems with your answer	10/15
The program works correctly and produces the correct output	15/15

In other words, you can obtain a maximum of 5 marks for questions 1, 2 and 3; a maximum of 10 marks for question 4; a maximum of 20 marks for questions 5 and 6; and a maximum of 15 marks for question 7.

Note that not all mistakes are *corrected* – but we will provide informative comments.

If you did not include the program output for questions 1, it means there are “a few problems with your answer” and the maximum you can get is then 3/5. If you did not include the program output for questions 5 and 6, it means there are “a few problems with your answer” and the maximum you can get is then 15/20. If you did not include the program output for question 7, it means there are “a few problems with your answer” and the maximum you can get is then 10/15.

We did not award any marks to assignments submitted more than four weeks after the due date. However, we still provided informative comments.

4. Solution to Assignment

Question 1 [max of 5 marks]

Discussion:

For this question, you had to convert the `struct Employee` into a class. There is essentially only a slight difference between a `struct` and a `class`. A `class` is the same as a `struct` (i.e. a `struct` may also contain both member variables and member functions just like a `class`, even though the examples in Savitch do not show that) except that, by default, all members are inaccessible to the general user of the class. This means that all members of a `struct` are by default `public`, and all members of a `class` are by default `private`. Therefore, we have to specify that the member functions are **`public`**. (As an exercise, omit the `public` keyword from the class and recompile it.) While there are situations where a `struct` is more suitable, as a rule, you should use classes rather than structs. In general, it is preferable to use classes, as classes offer better protection.

An *object* encapsulates or combines data and operations on that data into a single unit. In C++, the mechanism that allows you to combine data and the operations on that data in a single unit is called a **`class`**.

A **`class`** can be seen as a user-defined data type. We use a class to declare or *instantiate* an object. When an object of a specific class is instantiated, the constructor for the class is called by default. The purpose of the constructor is to initialise the member variables of the object automatically. This means that a constructor should not include instructions to obtain values for the member variables (`cin` statements). It also means that the member variables should not be re-declared inside the constructor, since these variables will then be local to the function, and will prevent the constructor function from accessing the member variables to initialise them. Study the constructor for class `Employee` as shown below to see how these concepts are implemented:

```
Employee:: Employee ()           //constructor
//Note: no cin statements and no variable declrations
{
    firstName = " ";
    lastName = " ";
    salary = 0;
}
```

As `firstName`, `lastName`, and `salary` are **`private`** member variables (see page 549 of Savitch 6th edition/ page 581 of Savitch 7th edition/ pages 573-582 of Savitch, 8th edition/pages 589-595 of Savitch, 9th edition) of class `Employee`, they cannot be accessed directly in the `main()` function. As a result, `public` member functions `getFirstName()`, `getLastName()` and `getSalary()` are used to access these

member variables in order to determine their values. These functions are known as *accessor* functions, while `setFirstName()`, `setLastName()` and `setSalary()` are *mutator* functions. (Study pages 553 – 554 of Savitch, 6th edition/ pages 585 -586 of Savitch 7th edition/ pages 581-582 of Savitch, 8th edition/ pages 597-598 of Savitch, 9th edition).

Note the prototypes for member functions:

```
string getFirstName()const;
string getLastName()const;
float getSalary()const;
```

These member function are accessors - hence the `const` keyword at the end of the function definition. Note that a member function that does not have the `const` keyword at the end of it could mutate (change or modify) the state of the object.

Note that the purpose of an **accessor** function (a function whose name starts with *get*, and sometimes called a get function) is not to obtain an input value for the member variable from a user, BUT to **return the current value of the member variable**. Accessor functions need to indicate a return type corresponding to the member variable for which it returns a value. An accessor function has no parameters. Since an accessor function is a member function of the class, it refers directly to the member variable which value it should return in the body of the function, i.e. there is no need to declare (and return) a local variable inside an accessor function. In fact, if you do declare a local variable inside an accessor function, and return the value of the local variable, it will NOT return the current value of the member variable!

Study the accessor function `getFirstName()` to see how these concepts are implemented:

```
//Accessors to retrieve (get) data from each of member variables
//Note: no parameters; return type of member function is the same as the
type
//of the member variable whose value is returned; and no variable
declarations
string Employee::getFirstName()const
{
    return firstName;
}
```

Mutator functions are used to change or modify member variables of an object. The parameter of the mutator function typically indicates the value according to which the member variable should be changed. For example, the mutator function `setFirstName()` below modifies member variable `firstName` to the string in variable `d`:

```
void Invoice::setFirstName(string d)
{
    description = d;
}
```

Program listing:

```
#include <iostream>
#include <string>
#include <iomanip>
```

```
using namespace std;

//declare class type
class Employee
{
public:
   Employee();
   Employee(const string fName, const string lName, const float sal);
   void setFirstName(string fName);
   void setLastName(string lName);
   void setSalary(float sal);
   string getFirstName()const;
   string getLastName()const;
   float getSalary()const;
private:
    string firstName;
    string lastName;
    float salary;
};

//Implement class

Employee::Employee()
{
    firstName = " ";
    lastName = " ";
    salary = 0;
}

Employee::Employee(const string fName, const string lName,
          const float sal)
{
    firstName = fName;
    lastName = lName;
    salary = sal;
}

void Employee::setLastName(string lName)
{
    lastName = lName;
}

void Employee::setFirstName(string fName)
{
    firstName = fName;
}

void Employee::setSalary(float sal)
{
    salary = sal;
```

```
}

string Employee::getFirstName()const
{
    return firstName;
}

string Employee::getLastName()const
{
    return lastName;
}

float Employee::getSalary()const
{
    return salary;
}

int main()
{
    Employee anotherEmployee;
    string fName, lName;
    float sal;

//Instantiate an employee
    Employee anEmployee("Joe", "Soap", 1456.00);

//Obtain employee detail
    cout << "Please enter first name of employee: ";
    cin >> fName;
    anotherEmployee.setFirstName(fName);
    cout << "Enter last name of employee: ";
    cin >> lName;
    anotherEmployee.setLastName(lName);
    cout << "Enter salary for employee: ";
    cin >> sal;
    anotherEmployee.setSalary(sal);

//Display employee's salaries
    cout.setf(ios::showpoint);
    cout.setf(ios::fixed);

    cout << endl << setprecision(2) << anEmployee.getFirstName() << ' '
         << anEmployee.getLastName() << "'s salary is R"
         << anEmployee.getSalary() << endl;
    cout << anotherEmployee.getFirstName() << ' '
         << anotherEmployee.getLastName() << "'s salary is R"
         << anotherEmployee.getSalary() << endl << endl;

//Give employees a 10% raise
     float raise =  anEmployee.getSalary() * 1.1;
```

```
     anEmployee.setSalary(raise);
     raise =  anotherEmployee.getSalary() * 1.1;
     anotherEmployee.setSalary(raise);

//Display employee's salaries again
    cout << "Having received a 10% raise employees salaries are now:"
         << endl;
    cout << anEmployee.getFirstName() << ' ' << anEmployee.getLastName()
         << "'s salary is R"
        << anEmployee.getSalary() << endl;
    cout << anotherEmployee.getFirstName() << ' '
         << anotherEmployee.getLastName() << "'s salary is R"
         << anotherEmployee.getSalary() << endl;
    return 0;
}
```

Input and output:

```
Please enter first name of employee: Joanne
Enter last name of employee: Soape
Enter salary for employee: 15446.66

Joe Soap's salary is R1456.00
Joanne Soape's salary is R15446.66

Having received a 10% raise employees salaries are now:
Joe Soap's salary is R1601.60
Joanne Soape's salary is R16991.33
Press any key to continue . . .
```

Question 2 [This question was not marked, and *5 marks* were allocated for attempting it]

(a) What is the purpose of the keywords **public** and **private** in the class declaration?
Member variables and member functions declared following a `private` label are accessible only to other member functions of the class.√ Data members and member functions following a `public` label are accessible to functions that are not members of the class (client functions).

(b) What is the difference between a **class** and an **object**?
A class can be seen as a user-defined data type. A class is a type whose variables are objects. We use a class to declare or instantiate an object.
An object is a variable that has functions associated with it. These functions are called member functions. The object's class determines which member functions the object has.

(c) What does it mean to '**instantiate'** an object?
To '**instantiate'** an object means to declare or create an object. This will allocate memory to the object where the values of the object's member variables can be stored. It will also initialize those member variable to the values specified in the object's constructor.

(d) What is the purpose of a **constructor**?

A constructor is automatically invoked when an object is instantiated (declared). The constructor can initialise some or all of the object's data members to appropriate values.

(e) What is the difference between the **default constructor** and the **overloaded constructor**?
The default constructor takes no arguments. The default constructor is automatically invoked when an object is instantiated. An overloaded constructor is a constructor with arguments so that the user can specify the values with which the object should be initialized.

(f) What is the purpose of a **destructor**?
Like constructors, destructors are also functions and do not have a return type. However, a class may have only one destructor and the destructor has no parameters. The name of a destructor is the tilde character (~) followed by the name of the class. The destructor automatically executes when the class object goes out of scope and releases the memory used by the object.

(g) What is the purpose of an **accessor**?
An accessor is a member function that accesses the contents of an object in order to return the value of a specific member variable. An accessor for an object does not modify that object. This is why it is good practice to use the keyword const when accessors are defined.

(h) What is the purpose of a **mutator**?
A mutator is a member function that can modify an object. Every non-const member function of a class is potentially a mutator. In the simplest case, a mutator just assigns a new value to one of the data members of the class. In general, a mutator performs some computation and modifies any number of data members

(i) What is the purpose of the **scope resolution operator**?
The **scope resolution operator ::** is used to specify the class name when giving the function definition for a member function.

(j) What is the difference between the **scope resolution operator** and the **dot** operator?
The scope resolution operator is used to indicate the class name to which an object belongs, whereas the dot operator is used to indicate to which object a member variable belongs.

(k) What is the difference between a **member function** and an **ordinary function**?
A member function is a function that is associated with an object. An ordinary function belongs to the main program.

(l) What is an **abstract data type (ADT)?**
An ADT is a data type that separates the logical properties from the implementation details, i.e. the interface and the implementation of the class are separated.

(m) How do we create an **ADT**?
We define an ADT as a class and place the definition of the class and implementation of its member functions in separate files. √ The ADT class is then compiled separately from any program that uses it.

(n) What are the advantages of using **ADT**s?
ADTs prevent programmers who use the ADT from having access to the details of how the values and operations are implemented. The user of the ADT only knows about the interface of the ADT and need not know anything about the implementation. This protects the ADT against inadvertent changes. The same ADT class can be used in any number of different programs.

(o) What is **separate compilation**?
Separate compilation means that a program is divided into parts that are kept in separate files, compiled separately and then linked together when the program is run.

(p) What are the **advantages** of **separate compilation**?
Separate compilation allows programmers to build up a library of classes so that many programs can use the same class.

(q) What is a **derived class?**
A **derived class** is a class that was obtained from another class by adding features through inheritance to create a specialized class, which also includes the features of the base class.

(r) What is the purpose of inheritance?
Inheritance allows one to define a general class and then define more specialized classes by adding some new details to the existing general class.

Question 3 [max of 5 marks]

(a) For this question, you were given the following class declaration, and code fragment. You had to explain what is wrong and in the code and write code to correct it:

```
class Person
{
public:
   Person();
   string getFirstName();
private:
   string ID;
   string lastName;
   string firstName;
   string phoneNumber;
};

int main()
{
      Person p;
      ……
                  string pNumber = p. class Person
{
public:
   Person();
   string getFirstName();
private:
   string ID;
   string lastName;
   string firstName;
   string phoneNumber;
};

int main()
{
      Person p;
      ……
      string pNumber = p.phoneNumber;
      return 0;
}
```

From the class declaration, it can be seen that the member variable `phoneNumber` is **`private`** and cannot be accessed directly. Only member functions of class `Person` have direct access to member variable `phoneNumber` of object `p`. To allow access from outside to the private member variables of object `e`, we need to define an accessor function for `getPhoneNumber()`, i.e. change the class definition as shown in bold:

```
class Person
{
public:
   Person();
   string getName();
   string getPhoneNumber();√
private:
   string ID;
   string lastName;
   string firstName;
   string phoneNumber;
};
      :
      :
int main()
{
      Person p;
      ……
      string pNumber = p.getPhoneNumber();        √
      return 0;
}
```

(b) For this part of the question, you had to explain what is wrong with the following code fragment, and then write code to correct it:

```
Person me, friends[10];
for (int i = 0; i <10; i++)
{
      if (me.firstName == friends.firstname[i])
            cout << "My friend no " << i
            << " has the same first name as me!";
}
```

This code fragment contains two errors. As in 3(a), member variable `firstname` is private to class `Person` and an accessor function is required to return its value. Also, the index to array `friends` should appear directly after the name of the array, instead of after the name of the member variable. The member variable `firstName` is not an array.

```
class Person
{
public:
   Person();
   string getName();
   string getPhoneNumber();
   string getFirstname();√
private:
   string ID;
```

```
    string lastName;
    string firstName;
    string phoneNumber;
};

    Person me, friends[10];
    for (int i = 0; i <10; i++)
    {
        if (me.getFirstName() == friends[i].getFirstname())√ √
            cout << "My friend no " << i
            << " has the same first name as me!";
    }
```

Question 4 [This question has not been marked. A max of 10 marks if you *attempted* this question]

There are many acceptable solutions to this program. The purpose of this question was just to allow you to get your hands dirty with designing and implementing your own class – not a trivial exercise, but very time-consuming to mark each individual version, which is why we only show our own (very simple) version.

We designed our `Book` class to contain information such as the title of the book, the author of the book, the price, and the number of copies in stock. We did not use separate compilation.

Our class should be able to do the following:

- Retrieve the title of the book
- Retrieve the author of the book
- Retrieve the price
- Retrieve the number of copies in stock
- Set the title of the book
- Set the author of the book
- Set the price
- Set the number of copies in stock
- Apply the discount percentage to adapt the price of the book
- Display the information available for one book

In the application we use a loop to obtain information for 3 books. Then we use two loops to first count and list the books where are more than 20 copies in stock, adapt the price according to the discount, and then to display the updated list of info for the books.

Program listing:

```
#include <iostream>

using namespace std;

//class defintion
class Book
{
```

```
public:
     Book();
    ~Book();
      Book(string titleP, string authorP, double priceP, int nrInStockP);
      string getTitle() const;
      string getAuthor() const;
    double getPrice() const;
    int getNrInStock() const;
    void setTitle(string titleP);
    void setAuthor(string authorP);
    void setPrice(double priceP);
    void setNrInStock(int nrP);
      void CalcSalePrice(double discountPercentage);
      void DisplayInfo() const;

private:
      string title;
      string author;
      double price;
    int nrInStock;
      };


//class implementation
Book::Book()
{
    title = "";
    author = "";
    price = 0.00;
    nrInStock = 0;
}

//overloaded constructor
Book::Book(string titleP, string authorP, double priceP, int nrInStockP)
{
    title = titleP;
    author = authorP;
    price = priceP;
    nrInStock = nrInStockP;
}
Book::~Book()
{
    cout << "Goodbye!" << endl;
}

string Book::getTitle() const
{
   return title;
}
string Book::getAuthor() const
{
   return author;
}
double Book::getPrice() const
{
   return price;
}
int Book::getNrInStock() const
```

```
{
   return nrInStock;
}

void Book::setTitle(string titleP)
{
    title = titleP;
}

void Book::setAuthor(string authorP)
{
    author  = authorP;
}

void Book::setPrice(double priceP)
{
    price = priceP;
}

void Book::setNrInStock(int nrP)
{
    nrInStock = nrP;
}


void Book:: CalcSalePrice(double discountPercentage)
{
   price = price - (discountPercentage/100 * price);
}

void Book::DisplayInfo() const
{
     cout << "Book title: " << title << endl;
     cout << "Author: " << author << endl;
     cout.setf(ios::fixed);
    cout.precision(2);
    cout.setf(ios::showpoint);
     cout << "Price: R" << price << endl;
     cout << "Nr of books in stock: " << nrInStock << endl;
}


int main()
{
    double percentage;
    Book bookShop[3];
    string sTitle, sAuthor;
    double dPrice;
    int iNr, count = 0;


    //Initialise array
    cout << "Enter details for books in bookshop" << endl;
    for (int i = 0; i < 3; i++)
    {
        cout << "Please enter the book/'s title: ";
        getline(cin, sTitle, '\n');
        bookShop[i].setTitle(sTitle);
```

```
        cout << "Please enter the author of the book: ";
        getline(cin, sAuthor, '\n');
        bookShop[i].setAuthor(sAuthor);
        cout << "Please enter the price in R: ";
        cin >> dPrice;
        bookShop[i].setPrice(dPrice);
        cout << "Please enter the number of copies in stock: ";
        cin >> iNr;
        bookShop[i].setNrInStock(iNr);
        cin.get();
        cout << endl;
    }

//Determine number of books with more than 2 copies in stock and display
info
    cout << "List of books where there are more than 20 copies in stock:"
         << endl;
    for (int i = 0; i < 3; i++)
    {
        if (bookShop[i].getNrInStock() > 20)
        {
            ++count;     //count number of books with more than 20 in stock
            bookShop[i].DisplayInfo();     //display books with more than
                                           //20 in stock
            bookShop[i].CalcSalePrice(15);  //discount price with 15%
            cout << endl;
        }
    }
    cout << "There are " << count << " books where there are more than 20 "
         << "copies in stock." << endl << endl;

    //Display info for all books
    cout << " List of all books in stock with updated info:" << endl;
    for (int i = 0; i < 3; i++)
    {
        bookShop[i].DisplayInfo();
        cout << endl;
    }

    cout << endl;

    return 0;
}
```

Output:

```
Enter details for books in bookshop
Please enter the book/'s title: Learning C++
Please enter the author of the book: P WIlliams
Please enter the price in R: 340.59
Please enter the number of copies in stock: 23

Please enter the book/'s title: Growing Aloes
Please enter the author of the book: P Brain
Please enter the price in R: 230.31
Please enter the number of copies in stock: 3

Please enter the book/'s title: Growing Old Gracefully
```

```
Please enter the author of the book: B Young
Please enter the price in R: 500.99
Please enter the number of copies in stock: 35

List of books where there are more than 20 copies in stock:
Book title: Learning C++
Author: P WIlliams
Price: R340.59
Nr of books in stock: 23

Book title: Growing Old Gracefully
Author: B Young
Price: R500.99
Nr of books in stock: 35

There are 2 books where there are more than 20 copies in stock.

 List of all books in stock with updated info:
Book title: Learning C++
Author: P WIlliams
Price: R289.50
Nr of books in stock: 23

Book title: Growing Aloes
Author: P Brain
Price: R230.31
Nr of books in stock: 3

Book title: Growing Old Gracefully
Author: B Young
Price: R425.84
Nr of books in stock: 35


Goodbye!
Goodbye!
Goodbye!
Process returned 0 (0x0)   execution time : 54.190 s
Press any key to continue.
```

Discussion:
There are various solutions to this question. The concepts of declaring a class of objects may be new to you, and you may have struggled to implement it. We hope that after studying our solution you will understand it, and be able to apply it.

- If a class has constructors and you declare an array of class objects, the class must have the default constructor. The default constructor is used to initialise each (`array`) class object.
- The declaration statement
  `Book bookShop[3];`
  declares an array named `bookShop` that contains `3` elements that each is of type `Book`.
  Note the three `Goodbye!`s in the output indicating that the destructor was called three times.
- We refer to a specific member function of a specific element in the array by using the dot operator ".".

Question 5 [max of 20 marks]

For this question, you had to define a class `Team` as an Abstract Data Type (ADT), so that separate files are used for the interface and implementation. See section 10.3, page 573 – 582 of Savitch 6th edition / page 605-614 of Savitch 7th / page 601-610 of Savitch 8th edition/ page 618 - 627 of Savitch 9th edition for more on ADT's and section 12.1, page 683-693 of Savitch 6th edition / page 726-741 of Savitch 7th / page 717-727 of Savitch 8th edition/ page 734-744 of Savitch 9th edition for more on separate compilation.

C++ has facilities for dividing a program into parts that are kept in separate files, compiled separately, and then linked together when the program is run. The header (`.h`) files are effectively the interfaces of the different classes, and the `.cpp` files contain the implementations.
For this exercise, you should have created three files:

```
Team.h
Team.cpp
main.cpp
```

Discussion:
You had to define a class `Team` as an ADT with member variables `country`, `round`, `points`, `goalsFor` and `goalsAgainst`. The class contains

- a default constructor,
- an overloaded constructor,
- a destructor,
- accessor and mutator functions for each of the member variables,
- a void member function `reset();`
- two member functions `calcGoalDifference()` and `update()`
- and overloaded friend functions for the operators ==, >, ++ as well as for the stream extraction operator >> and the stream insertion operator <<.

Consider operators ==, > and ++ overloaded as friend functions of the class `Team`:

```
friend bool operator>(const Team &Team1, const Team &Team2);
friend bool operator==(const Team &Team1, const Team &Team2);
```

Note, the **`const`** keyword is used to guarantee that these functions will not modify the `Team` objects. When we compare two `Team` objects with == or >, we do not want to change or modify the objects we compare. The same situation holds if we would define + and − operators for class `Team`. A **`friend`** function may manipulate the underlying structure of the class but the **`const`** keyword ensures that these manipulations do not modify the object in any possible way. We cannot place a **`const`** at the end of each prototype as the function is not a member function.

The relational friend functions == and > return boolean values. `Operator==` returns `true` when the `points` member variables and the goal difference of the two teams are the same. `Operator>` returns `true` when the `points` member variable of `Team1` is bigger than that of `Team2`; or if the `points` member variable of `Team1` is equal to that of `Team2` and the goal difference of `Team1` is bigger than that of `Team2`.

The prefix `operator++` is a unary operator, which means that it takes only one operand, in contrast to binary operators such as + or – which use two operands. Therefore, when overloading the prefix `operator++` as a friend function, only one parameter is specified, which represents the object that we want to increment:

```
friend Team operator++(Team &T);
```

Since the object will be modified in the `friend` function, it must be a reference parameter. In this implementation of the prefix operator++, we increment the `round` member variable of the object:

```
Team operator++(Team &T)
{
    ++T.round;
    return T;
}
```

See also the discussion on Question 2(h) in this tutorial letter.

Note the difference between the overloaded friend functions for the stream extraction operator >> and the stream insertion operator <<:

```
friend ostream & operator<<(ostream & out, const Team & T);
friend istream & operator>>(istream & ins,  Team & T);
```

The overloaded stream insertion operator << uses the `const` keyword to ensure that the object which is sent to the output stream will not be modified. In the case of the overloaded stream extraction operator >>, the object retrieved from the input stream, must change. The overloaded stream insertion operator << modifies the output stream and the overloaded stream extraction operator >> modifies the input stream, therefore both the `ostream` and the `istream` must be reference parameters.

Also, note that in the implementation of the overloaded operator >> we do not use `cout` statements to tell the user to type in input, or in what format the input should be. The purpose of overloading the operator >> is to allow us to use the operator >> to read in (extract) an object in the same way we would use it to read in or extract an ordinary variable. Consider the implementation of the overloaded >>:

```
istream & operator>>(istream & ins,  Team & T)
{
      ins >> T.country;
      ins >> T.round;
      ins >> T.points;
      ins >> T.goalsFor;
      ins >> T.goalsAgainst;
      return ins;
}
```

and the way it is used in the application:

```
Team opponent, newOpponent;
cout << "Enter data for new opponent (country, round, points, goals "
     << "for and goals against): ";
cin >> newOpponent;
```

In the same way, we overload the stream insertion operator << in order to be able to use it as follows:

```
Team opponent, newOpponent;
Team home("South-Africa", 1, 4, 6, 4);
opponent.reset("Germany", 1, 4, 6, 4);

cout << "The home team is: " << endl << home << endl;
cout << "The opponent team is " << endl << opponent << endl;
```

The complete listing for the `Team` ADT and the application program you had to use to test it, is shown below.

Program listing:
Team.h class definition / interface for class Team:

```
#ifndef TEAM_H
#define TEAM_H
#include <iostream>
#include <string>
```

```
using namespace std;
class Team
{
    public:
        Team();
        Team(string c, int r, int p, int gFor, int gAgainst);
        ~Team();
        string get_country() const;
        int get_round() const;
        int get_points()const;
        int get_goals_for()const;
        int get_goals_against()const;
        void set_country(string c);
        void set_round(int r);
        void set_points(int p);
        void set_goals_for(int gFor);
        void set_goals_against(int gAgainst);
        void reset(string c, int r, int p, int gFor, int gAgainst);
        int goalDifference()const;
        void update(int p, int gFor, int gAgainst);
        friend Team operator++(Team &T);
        friend bool operator>(const Team &Team1, const Team &Team2);
        friend bool operator==(const Team &Team1, const Team &Team2);
        friend istream & operator>>(istream & ins, Team & T);
        friend ostream & operator<<(ostream & out, const Team & T);

    private:
        string country;// the name of the country for which this team plays
        int round;          // the round in which the team currently plays
        int points;   // the points the team has accumulated
        int goalsFor; // the goals the team has scored
        int goalsAgainst;  // the goals scored against the team

};
#endif
```

Implementation file: Team.cpp

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

Team::Team()
{
    country = "Country 0";
    round = 0;
    points = 0;
    goalsFor = 0;
    goalsAgainst = 0;
}

Team::Team(string c, int r, int p, int gFor, int gAgainst)
{
    country = c;
    round = r;
    points = p;
    goalsFor = gFor;
```

```
    goalsAgainst = gAgainst;
}
Team::~Team()
{
    cout<<"Game Over!" << endl;
}

string Team::get_country() const
{
    return country;
}

int Team::get_round() const
{
    return round;
}

int Team::get_points()const
{
    return points;
}

int Team::get_goals_for()const
{
    return goalsFor;
}

int Team::get_goals_against()const
{
    return goalsAgainst;
}
void Team::set_country(string c)
{
    country = c;
}

void Team::set_round(int r)
{
    round = r;
}

void Team::set_points(int p)
{
    points = p;
}

void Team::set_goals_for(int gFor)
{
    goalsFor = gFor;
}

void Team::set_goals_against(int gAgainst)
{
    goalsAgainst = gAgainst;
}
void Team::reset(string c, int r, int p, int gFor, int gAgainst)
{
    country = c;
```

```
    round = r;
    points = p;
    goalsFor = gFor;
    goalsAgainst = gAgainst;
}

int Team::goalDifference()const
{
    return (goalsFor - goalsAgainst);
}

void Team::update(int p, int gFor, int gAgainst)
{
    points = points + p;
    goalsFor = goalsFor + gFor;
    goalsAgainst = goalsAgainst + gAgainst;
}

Team operator++(Team &T)
{
    ++T.round;
    return T;
}

bool operator>(const Team &Team1, const Team &Team2)
{
    if ((Team1.points > Team2.points) ||
        ((Team1.points == Team2.points) &&
         (Team1.goalDifference() > Team2.goalDifference())))
        return true;
    return false;
}

bool operator==(const Team &Team1, const Team &Team2)
{
    if ((Team1.points == Team2.points) &&
        (Team1.goalDifference() == Team2.goalDifference()))
        return true;
    return false;
}

istream & operator>>(istream & ins,  Team & T)
{
      ins >> T.country;
      ins >> T.round;
      ins >> T.points;
      ins >> T.goalsFor;
      ins >> T.goalsAgainst;
      return ins;
}

ostream & operator<<(ostream & out, const Team & T)
{
      cout << "Country : " << T.country << endl;
      cout << "round : " << T.round << endl;
      cout << "points : " << T.points << endl;
      cout << "goals for: " << T.goalsFor << endl;
      cout << "goals against: " << T.goalsAgainst << endl;
```

```
      return out;
}
```

Application file - TestTeam.cpp:

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

int main()
{
    Team opponent, newOpponent;
    Team home("South-Africa", 1, 4, 6, 4);
    opponent.reset("Germany", 1, 4, 6, 4);

    cout << "The home team is: " << endl << home << endl;
    cout << "The opponent team is " << endl << opponent << endl;

    if (home == opponent)
        cout << "This is a tie!" << endl;
    else if (home > opponent)
           ++home;
         else ++opponent;

    cout << home.get_country() << " has " << home.get_points()
         << " points and " << opponent.get_country() << " has "
         << opponent.get_points() << " points" << endl;
    cout << home.get_country() << " is in round " << home.get_round()
         << " and " << opponent.get_country() << " is in round "
         << opponent.get_round() << endl << endl;

    cout << "Enter data for new opponent (country, round, points, goals "
         << ”for and goals against): ";
    cin >> newOpponent;
    cout << endl;

    int SApoints, Spoints, goalsForSA, goalsForS,goalsAgainstSA;
    int goalsAgainstS;
    cout << "Enter points earned for this match by South-Africa: ";
    cin >> SApoints;
    cout << "Enter goals for SA: ";
    cin >> goalsForSA;
    cout << "Enter goals against SA: ";
    cin >> goalsAgainstSA;
    cout << "Enter points earned for this match by Spain: ";
    cin >> Spoints;
    cout << "Enter goals for Spain: ";
    cin >> goalsForS;
    cout << "Enter goals against Spain: ";
    cin >> goalsAgainstS;
    home.update(SApoints,goalsForSA,goalsAgainstSA);
    newOpponent.update(Spoints,goalsForS,goalsAgainstS);

    if (home == newOpponent)
        cout << "This is a tie!" << endl;
    else if (home > newOpponent)
```

```
            ++home;
         else ++newOpponent;

    cout << home.get_country() << " has " << home.get_points()
         << " points and " << newOpponent.get_country() << " has "
         << newOpponent.get_points() << " points" << endl;
    cout << home.get_country() << " is in round " << home.get_round()
         << " and " << newOpponent.get_country() << " is in round "
         << newOpponent.get_round() << endl << endl;

  return 0;
}
```

Output:

```
The home team is:
Country : South-Africa
round : 1
points : 4
goals for: 6
goals against: 4

The opponent team is
Country : Germany
round : 1
points : 4
goals for: 6
goals against: 4

This is a tie!
South-Africa has 4 points and Germany has 4 points
South-Africa is in round 1 and Germany is in round 1

Enter data for new opponent (country, round, points, goals for and goals
agains
): Spain 1 7 8 2

Enter points earned for this match by South-Africa: 3
Enter goals for SA: 2
Enter goals against SA: 0
Enter points earned for this match by Spain: 0
Enter goals for Spain: 0
Enter goals against Spain: 2
This is a tie!
South-Africa has 7 points and Spain has 7 points
South-Africa is in round 1 and Spain is in round 1

Game Over!
Game Over!
Game Over!

Process returned 0 (0x0)   execution time : 47.780 s
Press any key to continue.
```

Tip:

When coding a large abstract data type (ADT) it is best to write the member functions in a stepwise manner. The idea here is to “Code in small increments and then test.” For instance, code the constructors and the `friend` function > initially. Then write a small test program to check if these member functions work properly. You could probably write the code for the `friend` function `++` next. Then include

statements in your test program to test the `friend` function ++. In other words, you should not write the entire implementation (`Team.cpp`) and then commence with testing. By coding stepwise, it is easier to isolate errors.

Question 6 [max of 20 marks]

Discussion:

For this question, you had to define a class `Student` as an Abstract Data Type (ADT), so that separate files are used for the interface and implementation. See section 10.3, page 573 – 582 of Savitch 6th edition / page 605-614 of Savitch 7th / page 601-610 of Savitch 8th edition/ page 618 - 627 of Savitch 9th edition for more on ADT's and section 12.1, page 683-693 of Savitch 6th edition / page 726-741 of Savitch 7th / page 717-727 of Savitch 8th edition/ page 734-744 of Savitch 9th edition for more on separate compilation.

For this exercise, you should have created three files:

Student.h
Studdent.cpp
main.cpp

The ADT class `Student` has member variables `name, ID, stdtNr, diploma, modules[5], results[5]` and `average`. The class contains

- a default constructor,
- a destructor,
- accessor and mutator functions where needed to get or set the values of member variables,
- an `bool` member function `pass()` that converts to determine whether or not a student has passed a module,
- a `void` member function `calcAverage()` that calculates the average of the five results for the five modules,
- a `void` member function `displayResults ()` that displays a `Student`'s results after the examinations,
- and an overloaded friend function for the stream insertion operator <<.

Note the `const` in the header for the overloaded friend functions for the stream insertion operator <<:

```
friend ostream& operator << (ostream& outs, const Student& stud);
```

The overloaded stream insertion operator << uses the `const` keyword to ensure that the object which is sent to the output stream will not be modified. In the case of an overloaded stream extraction operator >>, the object retrieved from the input stream, must change, and should therefore not have a `const` in the header, e.g.

```
friend istream& operator >> (istream& ins, Student& stud);
```

The overloaded stream insertion operator << modifies the output stream and the overloaded stream extraction operator >> modifies the input stream, therefore both the `ostream` and the `istream` must be reference parameters.

We overload the stream insertion operator << as follows:

```
ostream& operator << (ostream& outs, const Student& stud)
{
    outs << stud.name << stud.ID << stud.stdtNr << stud.diploma;
    for (int i = 0; i < 5; i++)
        outs << stud.modules[i];
    for (int i = 0; i < 5; i++)
        outs << stud.results[i];
    outs << stud.average;
```

```
    return outs;
}
```
in order to be able to use it as follows:

```
        outfile << stud;
```
or
```
       outfile << registeredStudents[i];
```

Tips for separate compilation with multiple files:

- Only the `.cpp` files should be 'added' to the project.
- All files must be in the same directory or folder as the project file.
- The .cpp files must contain the preprocessor directive to include the `.h` files.
  e.g. `#include "Student.h"`

Note that all the files (`Student.h`, `Student.cpp` and `main.cpp`) must be in the same folder.

The complete listing for the `Student` ADT and the application program you had to use to test it, is shown below

Program code:

//Student.h
```
#ifndef STUDENT_H
#define STUDENT_H
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
using namespace std;

class Student
{
    public:
    friend ostream& operator << (ostream& outs, const Student& stud);
    Student();                                      //default constructor
    ~Student();                                     //destructor
    void calcAverage();
    bool pass();
    void displayResults();
    string getName() const;
    string getID() const;
    string getStdtNr() const;
    string getDiploma() const;
    double getAverage() const;
    string getModule(int i);
    void setResult(int i, int r);
    void setName(string n);
    void setID(string id);
    void setStdtNr(string stNr);
    void setDiploma(string dip);
    void setAverage(double avg);


    private:
        string name;
        string ID;
```

```
        string stdtNr;
        string diploma;
        string modules[5];
        int results[5];
        int average;
    };

#endif
```

//Student.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include "Student.h"
using namespace std;

Student::Student()                  //default constructor
{
    name = "";
    ID = "";
    stdtNr = "";
    diploma = "";
    for (int i = 0; i < 5; i++) //
        modules[i] = "";
    for (int i = 0; i < 5; i++)
        results[i] = 0;
    average = 0;//
}

Student::~Student() //destructor
{
cout<< "bye";
};

void Student::calcAverage()
{
    int sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum = sum + results[i];
    }

    average = sum / 5;
}

bool Student::pass()
{
    bool pass = true;
    for (int i = 0; i < 5; i++)
        if (results[i] <= 50)
            pass = false;
    return pass;
}

void Student::displayResults()
{
```

```
    cout << "Name: " << name << endl;
    cout << "Student number: " << stdtNr << endl;
    cout << "Diploma: " << diploma << endl;
    cout << "Average: " << average << endl;
    if (pass())
        cout << "Congratulations! You have passed" << endl;
        else cout << "Unfortunately you do not pass as you have not "
                  << "obtained 50% in all modules" << endl;
    cout << endl;
}

string Student::getName() const
{
    return name;
}

string Student::getID() const
{
    return ID;
}

string Student::getStdtNr() const
{
    return stdtNr;
}

string Student::getDiploma() const
{
    return diploma;
}

double Student::getAverage() const
{
    return average;
}

 string Student::getModule(int i)
 {
     return modules[i];
 }

void Student::setName(string n)
{
    name = n;
}

void Student::setID(string id)
{
    ID = id;
}

void Student::setStdtNr(string stNr)
{
    stdtNr = stNr;
}

void Student::setDiploma(string dip)
{
```

```
    diploma = dip;
    if (diploma == "Garden Design")
    {
        modules[0] = "G1";
        modules[1] = "G2";
        modules[2] = "G3";
        modules[3] = "G4";
        modules[4] = "G5";
    }
    else if (diploma == "Gourmet Cooking")
    {
        modules[0] = "C1";
        modules[1] = "C2";
        modules[2] = "C3";
        modules[3] = "C4";
        modules[4] = "C5";
    }
}

void Student::setAverage(double avg)
{
    average = avg;
}

void Student::setResult(int i, int result)
 {
     results[i] = result;
 }

ostream& operator << (ostream& outs, const Student& stud)
{
    outs << stud.name << stud.ID << stud.stdtNr << stud.diploma;
    for (int i = 0; i < 5; i++)
        outs << stud.modules[i];
    for (int i = 0; i < 5; i++)
        outs << stud.results[i];
    outs << stud.average;
    return outs;
}
```

//main.cpp

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

void captureResults(Student &s)// Note the reference parameter
{
    int r;
    cout << "Please enter exam results for " << s.getName()
         << " with ID " << s.getID() << " and student number "
         << s.getStdtNr() << " for the following modules:" << endl;
    for (int i = 0; i < 5; i++)
    {
        cout << "Please enter results for " << s.getModule(i)<< ": " ;
```

```
        cin >> r;
        s.setResult(i, r);
    }
    cout << endl;
}

int main()
{
    Student registeredStudents[3];

    ofstream outfile;
    outfile.open ("RegisteredStudentsResults.dat");
    if (outfile.fail())
    {
       cout<<"Error opening file RegisteredStudentsResults.dat";
       exit(1); // for opening file"
    }

    registeredStudents[0].setName("John Martin");
    registeredStudents[0].setID("78120189");
    registeredStudents[0].setStdtNr("12345");
    registeredStudents[0].setDiploma("Garden Design");
    registeredStudents[1].setName("Busi Molefe");
    registeredStudents[1].setID("81011201");
    registeredStudents[1].setStdtNr("23456");
    registeredStudents[1].setDiploma("Gourmet Cooking");
    registeredStudents[2].setName("Sean Naidoo");
    registeredStudents[2].setID("69812018");
    registeredStudents[2].setStdtNr("34567");
    registeredStudents[2].setDiploma("Garden Design");

    for (int i =0; i < 3; i++)
    {
        captureResults(registeredStudents[i]);
        registeredStudents[i].calcAverage();
        outfile << registeredStudents[i];
    }

    cout << "Results for this exam:" << endl;
    for (int i = 0; i < 3; i++)
        registeredStudents[i].displayResults();

    return 0;
}
```

RegisteredStudentsResults.dat:

```
John Martin7812018912345Garden DesignG1G2G3G4G5556559686061Busi
Molefe8101120123456Gourmet CookingC1C2C3C4C5697166596365Sean
Naidoo6981201834567Garden DesignG1G2G3G4G5694566575057
```

Output for main():

```
Please enter exam results for John Martin with ID 78120189
and student number 12345 for the following modules:
Please enter results for G1: 55
Please enter results for G2: 65
Please enter results for G3: 59
```

```
Please enter results for G4: 68
Please enter results for G5: 60

Please enter exam results for Busi Molefe with ID 81011201
and student number 23456 for the following modules:
Please enter results for C1: 69
Please enter results for C2: 71
Please enter results for C3: 66
Please enter results for C4: 59
Please enter results for C5: 63

Please enter exam results for Sean Naidoo with ID 69812018
and student number 34567 for the following modules:
Please enter results for G1: 69
Please enter results for G2: 45
Please enter results for G3: 66
Please enter results for G4: 57
Please enter results for G5: 50

Results for this exam:
Name: John Martin
Student number: 12345
Diploma: Garden Design
Average: 61
Congratulations! You have passed

Name: Busi Molefe
Student number: 23456
Diploma: Gourmet Cooking
Average: 65
Congratulations! You have passed

Name: Sean Naidoo
Student number: 34567
Diploma: Garden Design
Average: 57
Unfortunately you do not pass as you have not obtained 50% in all modules

byebyebye
Process returned 0 (0x0)   execution time : 56.760 s
Press any key to continue.
```

Tip:
When coding a large abstract data type (ADT) it is best to write the member functions in a stepwise manner. The idea here is to "Code in small increments and then test." For instance, code the constructors and the `friend` function << initially. Then write a small test program to check if these member functions work properly. You could probably write the code for the `friend` function > next. Then include statements in your test program to test the `friend` function >. In other words, you should not write the entire implementation (`Student.cpp`) and then commence with testing. By coding stepwise, it is easier to isolate errors.

Question 7 [max of 15 marks]

For this question you had to implement the class `Prescription`. You should have used separate

compilation. The interface or header file `Prescription.h` and implementation `Prescription.cpp` of class `Prescription` are shown below.

7 (a)

Prescription.h

```
#ifndef PRESCRIPTION_H
#define PRESCRIPTION_H
#include <iostream>

using namespace std;
class Prescription
{
public:
     Prescription();
     ~Prescription();
      Prescription(string medicineP, string patientP, string medAidNameP,
                  int medAidNrP, double priceP);
      string getMedicine() const;
      string getPatient() const;
      string getMedAidFund()const;
     int getMedAidNr() const;
     double getPrice() const;
      void Discount(double discountPercentage);
      void DisplayInfo() const;

protected:
      string medicine;
      string patientName;
      string medAidFund;
      int medAidNr;
      double price;

      };
#endif
```

Prescription.cpp

```
//Prescription.cpp
#include "Prescription.h"
#include <iostream>
using namespace std;

Prescription::Prescription()
{
    medicine = "";
    patientName = "";
    medAidFund = "";
    medAidNr = 0;
    price = 0.00;
}

//overloaded constructor
Prescription::Prescription(string medicineP, string patientP, string
medAidFundP, int medAidNrP, double priceP)
{
    medicine = medicineP;
    patientName = patientP;
    medAidFund = medAidFundP;
```

```
    medAidNr = medAidNrP;
    price = priceP;
}

Prescription::~Prescription()
{
    //destructor – do nothing
}

string Prescription::getMedicine() const
{
   return medicine;
}

string Prescription::getPatient() const
{
   return patientName;
}

string Prescription::getMedAidFund() const
{
   return medAidFund;
}

double Prescription::getPrice() const
{
   return price;
}

int Prescription::getMedAidNr() const
{
   return medAidNr;
}

void Prescription:: Discount(double discountPercentage)
{
   price = price - (discountPercentage/100 * price);
}

void Prescription::DisplayInfo() const
{
      cout << "Medicine prescribed: " << medicine << endl;
      cout << "Patient: " << patientName << endl;
      cout << "Medical Aid Fund: " << medAidFund<< endl;
      cout << "Medical Aid Number: " << medAidNr<< endl;
      cout.setf(ios::fixed);
     cout.precision(2);
     cout.setf(ios::showpoint);
      cout << "Price: R" << price << endl;

}
```

7 (b)

You had to test your implementation of class `Prescription` in a driver program which use the overloaded constructor to instantiate an object of class `Prescription`, use the accessor functions to display the medicine prescribed, the patient, the medical aid fund, the medical aid number and the cost. Then a new price based on the discount discount had to be calculated using member function

`Discount()` and member function `DisplayInfo()` had to be used to display the updated information about the prescription.

Main.cpp

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

using namespace std;

int main()
{
    Prescription myPrescription ("Panado", "John Apfelbaum",
                                 "THEAID", 12345, 39.59);
    int percentage;
    cout << "The medicine prescribed is "
         << myPrescription.getMedicine() << endl;
    cout << "The patient for whom it is prescribed is "
         << myPrescription.getPatient() << endl;
    cout << "This patient belongs to "
         << myPrescription.getMedAidFund()
         << " and the medical aid number is "
         << myPrescription.getMedAidNr() << endl;
    cout.setf(ios::fixed);
    cout.precision(2);
    cout.setf(ios::showpoint);
    cout << "The price for the medicine is R"
         << myPrescription.getPrice() << endl << endl;
    cout << "What is the discount percentage? ";
    cin >> percentage;
    myPrescription.Discount(percentage); //recalculate cost based on
                                         //discount
    cout << endl << "The discounted price is R"
         << myPrescription.getPrice() << endl;
    cout << endl << "Updated info for prescription:" << endl;
    myPrescription.DisplayInfo();
    cout << endl;
    return 0;
}
```

Output:

```
The medicine prescribed is Panado
The patient for whom it is prescribed is John Apfelbaum
This patient belongs to THEAID and the medical aid number is 12345
The price for the medicine is R39.59

What is the discount percentage? 10

The discounted price is R35.63

Updated info for prescription:
Medicine prescribed: Panado
Patient: John Apfelbaum
Medical Aid Fund: THEAID
Medical Aid Number: 12345
Price: R35.63
```

```
Process returned 0 (0x0)   execution time : 5.507 s
Press any key to continue.
.
```

7 (c)

In this question you had to derive a class `RepeatPrescription` from class `Prescription` and implement it.

RepeatPrescription.h:

```
#ifndef REPEATPRESCRIPTION_H
#define REPEATPRESCRIPTION_H
#include "Prescription.h"
#include <iostream>

using namespace std;
class RepeatPrescription: public Prescription
{
public:
     RepeatPrescription();
     ~RepeatPrescription();
      RepeatPrescription(string medicineP, string patientP, string
medAidNameP, int medAidNrP, double priceP, int numberOfRepeatsP, string
lastDateIssuedP);
    int getNumberOfRepeats() const;
    string getLastDateIssued() const;
    void DisplayInfo() const;
    void issuePrescription(string currentDate);

private:
      int numberOfRepeats;
      string lastDateIssued;

      };
#endif
```

RepeatPrescription.cpp:

```
//RepeatPrescription.cpp
#include "Prescription.h"
#include "RepeatPrescription.h"
#include <iostream>
using namespace std;

RepeatPrescription::RepeatPrescription(): Prescription(),
numberOfRepeats(0), lastDateIssued("")
{
}

//overloaded constructor
RepeatPrescription::RepeatPrescription(string medicineP, string patientP,
string medAidFundP, int medAidNrP, double priceP, int numberOfRepeatsP,
string lastDateIssuedP):
                      Prescription(medicineP, patientP, medAidFundP,
                    medAidNrP, priceP),numberOfRepeats(numberOfRepeatsP),
                    lastDateIssued(lastDateIssuedP)
{
}
```

```
RepeatPrescription::~RepeatPrescription()
{
    //destructor – do nothing
}

string RepeatPrescription::getLastDateIssued() const
{
   return lastDateIssued;
}

int RepeatPrescription::getNumberOfRepeats() const
{
   return numberOfRepeats;
}

void RepeatPrescription::issuePrescription(string currentDate)
{
    --numberOfRepeats;
    if (numberOfRepeats == 0)
    cout << endl << "Last issue of current prescription" << endl << endl;
    lastDateIssued = currentDate;
}

void RepeatPrescription::DisplayInfo() const
{
     cout << "Medicine prescribed: " << medicine << endl;
     cout << "Patient: " << patientName << endl;
     cout << "Medical Aid Fund: " << medAidFund<< endl;
     cout << "Medical Aid Number: " << medAidNr<< endl;
     cout.setf(ios::fixed);
     cout.precision(2);
     cout.setf(ios::showpoint);
     cout << "Price: R" << price << endl;
     cout << "Number of repeat prescriptions left: " << numberOfRepeats
          << endl;
     cout << "Date of last issue: " << lastDateIssued << endl;

}
```

7 (d)

In this question you had to test the class `RepeatPrescription` derived from class `Prescription` in a driver program. Note that the implementation of the parent class (`Prescription.cpp`) must be included as part of the project files, and the header file `Prescription.h` must be included in the application file with the `#include "Prescription.h"` directive. Both `Prescription.h` and `Prescription.cpp` also need to be in the same folder as the other files for class `RepeatPrescription`.

Application file:

```
#include <iostream>
#include "Prescription.h"
#include "RepeatPrescription.h"

using namespace std;

int main()
{
    RepeatPrescription myPrescription ("Myprodol", "Annie Apfelbaum",
                                       "THEAID", 43215, 89.59, 1, "20170620");
```

```
    string today; // for today's date
    cout << "The medicine prescribed is " << myPrescription.getMedicine()
         << endl;
    cout << "The patient for whom it is prescribed is "
         << myPrescription.getPatient() << endl;
    cout << "This patient belongs to " << myPrescription.getMedAidFund()
         << " and the medical aid number is "
         << myPrescription.getMedAidNr() << endl;
    cout.setf(ios::fixed);
    cout.precision(2);
    cout.setf(ios::showpoint);
    cout << "The price for the medicine is R"
         << myPrescription.getPrice() << endl;
    cout << "The number of repeats allowed for this prescription is "
         << myPrescription.getNumberOfRepeats() << endl;
    cout << "The prescription was last issued on "
         <<  myPrescription.getLastDateIssued() << endl << endl;

    cout << "Please enter today's date in the format yyyymmdd: ";
    cin >> today;

    myPrescription.issuePrescription(today); //recalculate cost based on
                                             // discount

    cout << endl << "Updated info for prescription:" << endl;
    myPrescription.DisplayInfo();
    cout << endl;
    return 0;
}
```

Output:

```
The medicine prescribed is Myprodol
The patient for whom it is prescribed is Annie Apfelbaum
This patient belongs to THEAID and the medical aid number is 43215
The price for the medicine is R89.59
The number of repeats allowed for this prescription is 1
The prescription was last issued on 20170620

Please enter today's date in the format yyyymmdd: 20170821

Last issue of current prescription


Updated info for prescription:
Medicine prescribed: Myprodol
Patient: Annie Apfelbaum
Medical Aid Fund: THEAID
Medical Aid Number: 43215
Price: R89.59
Number of repeat prescriptions left: 0
Date of last issue: 20170821


Process returned 0 (0x0)   execution time : 6.240 s
Press any key to continue.
```

2017