Experiment No. 06
Classes

OBJECTIVE:

Things that will be covered in today's lab:

- Classes
  a. Constructors
  b. Destructors
  c. Accessor/Mutator functions (Or getter/setter functions).

THEORY:

Classes are the basic building block of object oriented programming. In C++, classes have the same format as plain *structs*, except that they can also include functions and have these new things called *access specifiers*. An *access specifier* is one of the following three keywords: **private**, **public** or **protected**. These specifiers modify the access rights for the members that follow them:

- Private members of a class are accessible only from within other members of the same class (or from their *"friends"*).
- Protected members are accessible from other members of the same class (or from their *"friends"*), but also from members of their derived classes.
- Finally, public members are accessible from anywhere where the object is visible.

By default, all members of a class declared with the class keyword have private access for all its members. Therefore, any member that is declared before any other *access specifier* has private access automatically.

Class Definition: Classes are defined using either keyword *class*, with the following syntax:

```
class class_name
{
access_specifier:
//data member;
                //member functions;
access_specifier:
//data member;
                //member functions;
};
```

Access Class Members: To access any member variable or a function of the class, we use the *member access operator (.).* The member access operator is coded as a period between the

class object and member variable or function of class.

Define class member function:

In outside definition, the operator of scope (::) is used to specify that the function being defined is a member of the class and not a regular non-member function.

```
return_type class_name::function_name( arguments )
```

C++ *structs* are almost similar to classes and have the same syntax for definition. The only difference is that the reserve word *struct* is used and all functions and data member within a **struct** are *pubic* by default instead of *private* as in classes.

Constructors: A method which is called when an object is instantiated, it is used to initialize the member variables of the class. It has no return type and same name as the class..

Default constructor has no parametes. If you don't explicitly declare it within the class it gets added automatically by the system. Its general syntax is

```
class_name::class_name()
```

Exercise 1: What should be the output of this program?

```
// Example program
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
      string name;
      int rollnumber;
public:
      student ();
};
student::student()
{
      name=" ";
      rollnumber=0;
      cout<<"Default constructor called"<<end;
}

int main ()
{student A;
 student *ptrA= new Student();
 delete ptrA;
 return 0;}
```

Default constructor called
Default constructor called

A **parameterized constructor** has parametes. If you add even one parameterized constructor, the default constructor doesn't get added automatically. Its general syntax is

```
class_name::class_name( datatype1 parameter1, datatype2 parameter2,….)
```

Exercise2: What should be the output of this program?

```
// Example program
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
      string name;
      int rollnumber;
public:
//student ();
student (string, int);
};
/*student::student()
{
name=" ";
rollnumber=0;}*/
student::student(string a, int b)
{name=a;
rollnumber=b;
cout<<"parameterized constructor called"<<endl;}

int main ()
{
student A;
student B("abc", 1234);
student ptrB= new student("def", 867);

return 0;
}
```

parameterized constructor called
parameterized constructor called

Why did the above code error out? Now uncomment the default constructor and rerun the code. And state the output below.

A **copy constructor** is called when an object is intitialized at the same time as when it is

declared. It has the following general syntax:

```
class_name::class_name(class_name & parameter)
```

Exercise3: What should be the output of this program?

```
// Example program
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
      string name;
      int rollnumber;
public:
      student ();
      student (string, int);
      student (student & cp);

};
student::student()
{ cout<<"default constructor called"<<endl;

      name=" ";
      rollnumber=0;
}
student::student(string a, int b)
{
      cout<<"parameterized constructor called"<<endl;
      name=a;
      rollnumber=b;
}
student::student(student& b)
{
      cout<<"copy constructor called"<<endl;
      this->name=b.name;
      this->rollnumber=b.rollnumber;
}
int main ()
{
student A;
student B("abc", 1234);
student ptrB= new student("def", 867);

return 0;
}
```

default constructor called
parameterized constructor called
parameterized constructor called

// Create a new student object as a copy of B . to print (copy constructor)
student C = B;

Destructors: A method which is called when an object is destroyed. Its general syntax is as follows:

```
class_name::~class_name()
```

Exercise4: What should be the output of this program?

```
// Example program
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
      string name;
      int rollnumber;
public:
      student ();
      student (string, int);
      ~student ();

};
student::student()
{ cout<<"default constructor called"<<endl;

      name=" ";
      rollnumber=0;
}
student::student(string a, int b)
{
      cout<<"parameterized constructor called"<<endl;
      name=a;
      rollnumber=b;
}
student::~student()
{
      cout<<"Destructor called for
      rollnumber"<<rollnumber<<endl;
}

int main ()
{
student A;
student ptrB= new student("def", 867);
delete ptrB;
student B("abc", 1234);

return 0;
}
```

```
default constructor called
parameterized constructor called
Destructor called for rollnumber 867
parameterized constructor called
Destructor called for rollnumber 1234
Destructor called for rollnumber 0
```

Accessor/Mutator function:

Accessor/Getter function has the general syntax of :

Datatype classname::getMembervariable();

Mutator/Setter function has the general syntax of :

void classname::setMembervariable(datatype of member variable);

Exercise5: What should be the output of this program?

```
// Example program
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
      string name;
      int rollnumber;
public:
      student ();
      student (string, int);
      int getRollnumber();
      void setRollnumber(int);
      ~student ();

};
student::student()
{ cout<<"default constructor called"<<endl;

      name=" ";
      rollnumber=0;
}
student::student(string a, int b)
{
      cout<<"parameterized constructor called"<<endl;
      name=a;
      rollnumber=b;
}
student::~student()
{
      cout<<"Destructor called for
      rollnumber"<<rollnumber<<endl;
}

void student::setRollnumber(int r)
{
      rollnumber=r;
}
int student::getRollnumber()
```

default constructor called
parameterized constructor called
122
123

```
{
        return rollnumber=r;
}


int main ()
{
        student A;
        student ptrB= new student("def", 867);
        A.setRollnumber(122);
        ptrB->setRollnumber(123);

        cout<<A.getRollnumber()<<endl;
        cout<< ptrB->getRollnumber();
        return 0;
}
```

For all of the exercises below save your code on the learning management system (LMS) and give the screen shot of the output you get on the console in the space provided after every exercise.

Exercise 6: (10 points)

Using the above example as guideline please define a class named **Bike**, with four data members: *brand, model, price* and *topspeed*.

Write a program that includes the following:

a. Default constructor.
b. Parameterized constructor.
c. Copy constructor.
d. Destructor.
e. Accessor/Mutator function for every data member.
f. A general member function that displays all the data members.
g. Write a main function that:
    a. Create a Bike object and call its various **member functions** using that object.
    b. Create a Bike pointer and calls its various **member functions** using the pointer.
h. **Write a non-member function** that takes as parameter the Bike variable declared in above steps and displays its data members

Please give the screenshot of the output that you get after running your program: