OOP
Lecture 20

21/12/2021

Quiz

names[1000][30] = { “............”, “.........”,.............................}; // 1000 names

Question 1 : Create a class Student with data members Name(select randomly from above given global array), Roll number (1 to 50), class number (1 to 10), section (A to D).

- Write a parameterized constructor with three parameters(roll number, class, section) to initialize its data members.
- Write any other function required in below questions.

Solution :

```
    #include<iostream>

using namespace std;

class Student{

    protected:
        char name[30];
        int rollNo;
        int classNo;
        char section;

    public:
        Student( int rollNumber, int classNumber, char sec )
        {
            strcpy(name, names[rand()%1000]);
```

```
                if( rollNumber >= 1 && rollNumber <= 50 )
                        rollNo = rollNumber;
                else
                        rollNo = 1;

                if( classNumber >= 1 && classNumber <= 10 )
                        classNo = classNumber;
                else
                        classNo = 1;

                if( sec >= 'A' && sec <= 'D' )
                        section = sec;
                else
                        section = 'A';
        }

        void show() const
        {
                cout << "Name: " << name << '\n';
                cout << "Roll Number: " << rollNo << '\n';
                cout << "Section: " << section << '\n';
                cout << "Class: " << class << '\n';
        }
};
```

Question 2 : Create a class **Class** with upto 50 students with attributes Name(select randomly from above given global array), Roll number (1 to 50), class number (1 to 10), section (A to D). Write following member functions:

- Write a Parameterized constructor with 3 parameters, Number of Students(default is 10), class number, and section. Assign Roll numbers from 1 to onwards.
- addStudent() to add a student if the count is less than 50.
- Write a show function to display output like this.

```
Class : 7         Section : A
Roll numbers      Names
—-----------------------------------
1                 Mohsin
2                 Haider
```

Solution :

```
class Class{
        Student *student[50];
        int numOfStudents;

public:
        Class(int num, int classNo, char section)
        {
                if ( num <= 0 || num > 50 )
                        num = 10;
                numOfStudents = num;

                for ( int i = 0; i < numOfStudents; i++ )
                        student[i] = new Student( i+1, classNo, section );
        }

        void addStudent()
        {
                if ( numOfStudents == 50 )
                        cout << "50 Students are Max. \n";

                else
                {
                        numOfStudents++;
        student[numOfStudent - 1] = new
Student(numOfStudent,student[0].getClassNo(),student[0].getSection());
//Passing Parameters, Roll Number, Class number, section
                }
        }
```

```
	void show() const
	{
		cout << "Class: " << student[0].getClassNo();
		cout << "\tSection: " << student[0].getSection() << '\n';
		cout << "Roll Numbers \t Names \n -------------------------------- \n";

		for ( int i = 0; i < numOfStudents; i++ )
			cout << i+1 << '\t' << student[i].getName() << '\n';
	}
};
```

Question 3 : Write a class MFStudent inherited from the Student class. Add one more data member gender to store 'M' or 'F' in it. Write following member functions:

- Write a Parameterized constructor with 4 parameters
- Write Show function to show the details of Student including its gender.

Solution :

```
class MFStudent : public Student{
	char gender;

public:
	MFStudent( int rollNo, int classNo, char section, char gender )
	: Student( rollNo, classNo, section )
	{
		if ( gender == 'F' )
			this->gender = 'F';
		else
			this->gender = 'M';
	}
```

```
        void show() const
        {
                Student :: show();
                cout << "Gender: ";
                if ( gender == 'M' )
                        cout << "Male";
                else
                        cout << "Female";
                cout << '\n';
        }
};
```

QUIZ ENDED

Writing Code in Multiple Files :

As programs get larger, it is common to split them into multiple files for organizational or reusability purposes. One advantage of working with an IDE is that they make working with multiple files much easier. We already know how to create and compile single-file projects. Adding new files to existing projects is very easy.

We just need to add the cpp file in our main project file with #include directive, Like:

# include “Point.cpp”
# include “RationalNumbers.cpp”

Like this we can add as many files as we want in our major file and these are all works exactly like these are written in the same file. The pre-processor directive replaces the #include line with the contents of the whole file.

There is a problem that arises if we add the same file multiple times in a project. So, there will be the redefinition of the codes which will generate certain errors. To avoid this problem, we add macos in the file which is to be included in the project.

Macro :

Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The ‘#define’ directive is used to define a macro.

Conditional Macro :

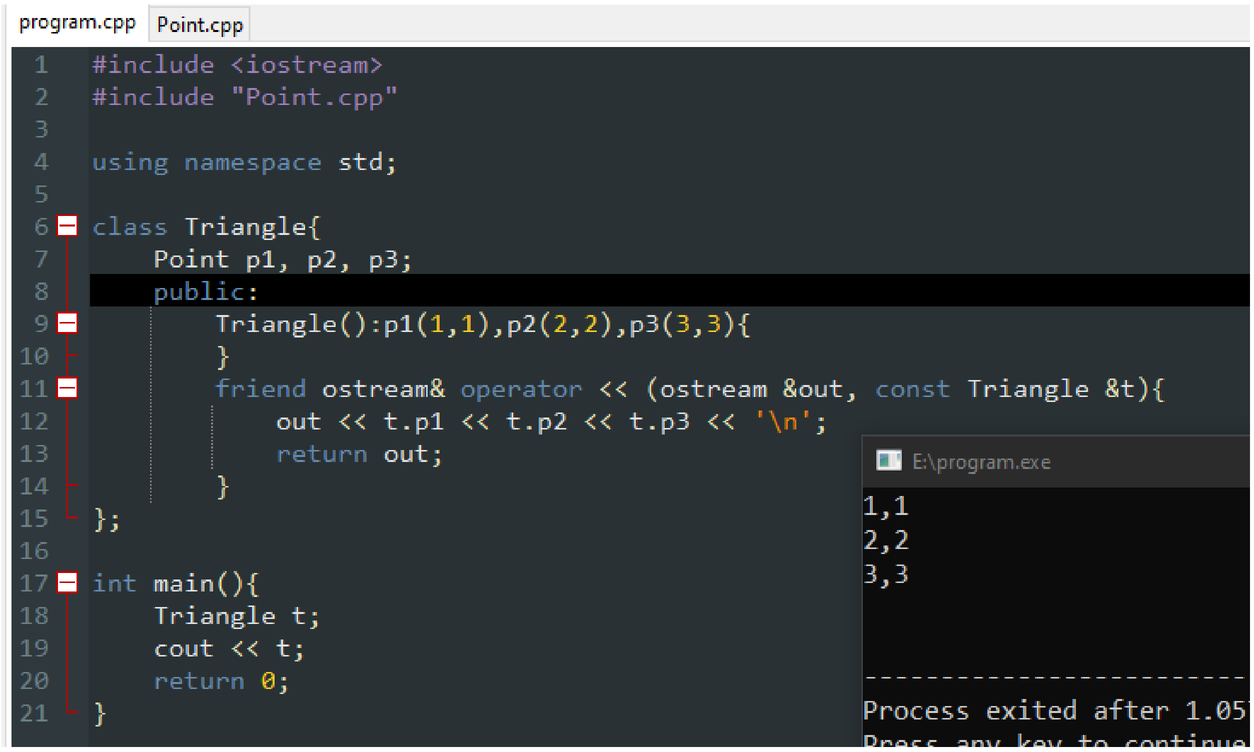
Conditional Compilation directives are a type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions. This can be done with the help of two preprocessing commands **‘ifndef‘** and **‘endif‘**.

Syntax :

```
    #ifndef macro_name
    #define macro_name

    —----------------------------
    —----------------------------
        (Statements)
    —----------------------------
    —----------------------------

    #endif
```

It will not add the same code again in the file even if we write the #include statements multiple times.

Sample Program :

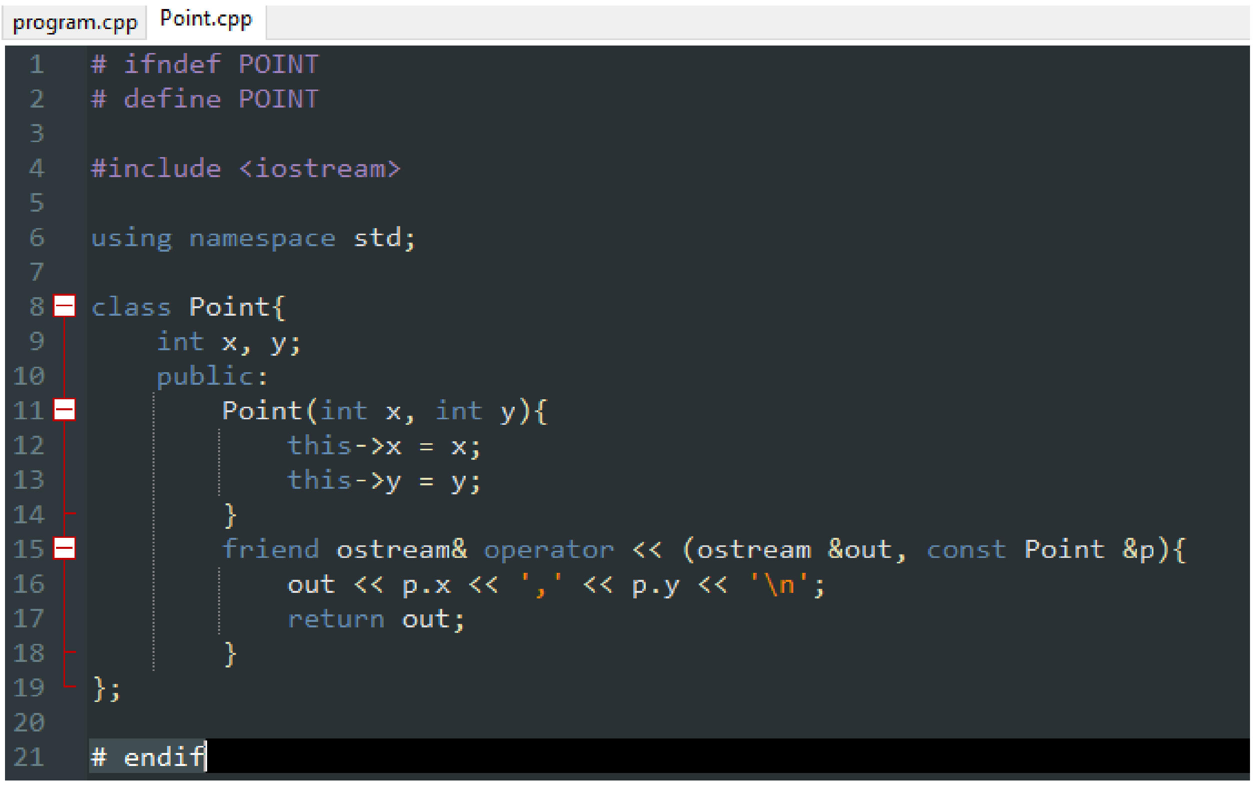


program.cpp Point.cpp

```cpp
#include <iostream>
#include "Point.cpp"

using namespace std;

class Triangle{
    Point p1, p2, p3;
    public:
        Triangle():p1(1,1),p2(2,2),p3(3,3){
        }
        friend ostream& operator << (ostream &out, const Triangle &t){
            out << t.p1 << t.p2 << t.p3 << '\n';
            return out;
        }
};

int main(){
    Triangle t;
    cout << t;
    return 0;
}
```

E:\program.exe

```
1,1
2,2
3,3

--------------------------
Process exited after 1.05
Press any key to continue
```



program.cpp Point.cpp

```cpp
# ifndef POINT
# define POINT

#include <iostream>

using namespace std;

class Point{
    int x, y;
    public:
        Point(int x, int y){
            this->x = x;
            this->y = y;
        }
        friend ostream& operator << (ostream &out, const Point &p){
            out << p.x << ',' << p.y << '\n';
            return out;
        }
};

# endif
```

Note : All files to be included should be in the same directory.

Child Class Modes :
In C++, we can derive a child class from the parent class in
different access modes. For example,

```
class Parent{
.... ... ....
};

class Child : public Parent {
.... ... ....
};
```

Notice the keyword public in the code : class Child : **public** Parent

This means that we have created a child class from the parent class in public mode. Alternatively, we can also derive classes in protected or private modes.

These 3 keywords (public, protected, and private) are known as access specifiers in C++ inheritance.

public, protected and private inheritance in C++:

public, protected, and private inheritance have the following features:

- public inheritance makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.

- protected inheritance makes the public and protected members of the base class protected in the derived class.
- private inheritance makes the public and protected members of the base class private in the derived class.

Note : private members of the base class are inaccessible to the derived class.

```
class Base {
   public:
      int x;
   protected:
      int y;
   private:
      int z;
};

class PublicDerived: public Base {
   // x is public
   // y is protected
   // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
   // x is protected
   // y is protected
   // z is not accessible from ProtectedDerived
};
```

```
class PrivateDerived: private Base {
   // x is private
   // y is private
   // z is not accessible from PrivateDerived
}

class grandDerieved : public PrivateDerived{
        // x is not accessible from grandDerieved
        // y is not accessible from grandDerieved
        // z is not accessible from grandDerieved
        // as all the members of grandDerieved  are private
};
```

-THE END-