

GOVERNMENT POLYTECHNIC, AMRAVATI

(An Autonomous Institute of Government of Maharashtra)

NBA Accredited Institute

Certificate



Name of Department: Computer Science and Engineering.

This is to certify that **Mr. Ayush Shashikant Bulbule** Identity Code **19CM007** has completed the practical work of the course **FC3405 Object Oriented Programming using C++** during the Academic year.

Signature of the Teacher

Date:

who taught the examinee

Head of Department

Index

S.No	Name of Experiment	Date	Page	Remarks
1	Develop minimum 2 programs using constants, variables, arithmetic expression operators, exhibiting data type conversion.	20-9-2020	3	
2	Develop a program to implement decision making statements (if-else, switch).	24-9-2020	6	
3	Develop a program to demonstrate control structures (for, while, do-while).	26-9-2020	8	
4	Develop a program to implement 1-dimension array	28-9-2020	10	
5	Develop a program to perform matrix operation using multi-dimensional array	30-9-2020	13	
6	Develop a program that implements a class and use it with objects	2-10-2020	16	
7	Develop a program that implements a class and create array of objects.	4-10-2020	19	
8	Write a program to implement friend function.	7-10-2020	22	
9	Write a program to implement inline function.	8-10-2020	25	
10	Write a program to implement Constructor for creation of Object.	9-10-2020	28	
11	Write a program to implement Multiple constructors.	15-10-2020	30	
12	Write a program to implement Copy Constructors.	17-10-2020	32	
13	3 13. Write a program to implement constructor overloading with destructor.	19-10-2020	34	
14	Write a program to demonstrate operator overloading for Unary operator	22-10-2020	37	
15	Write a program to demonstrate operator overloading for Binary operator.	28-10-2020	41	
16	Write a program to demonstrate operator overloading for Binary operator	19-11-2020	44	
17	Write a program for implementing multi level inheritance.	20-11-2020	46	
18	Write a program for implementing multiple inheritances	22-11-2020	49	
19	. Write a program for implementing Constructor in Derived class	2-12-2020	51	
20	Develop minimum 1 program to demonstrate Pointer to object	3-12-2020	53	

21	Develop minimum 1 program to demonstrate Pointer to derived class.	8-12-2020	55	
22	Develop minimum 1 program to Implement Compile time polymorphism.	20-12-2020	57	
23	Develop minimum 1 program to Implement run time polymorphism using virtual functions.	24-12-2020	59	
24	Write a program to read and write data to and from a file.	08-01-2021	61	

Practical no 1.

Aim: Develop minimum 2 programs using constants, variables, arithmetic expression operators, exhibiting data type conversion.

Requirements: Computer system with minimal specifications and c++ development IDE (VS Code used).

Theory:

Variables in C++:

A variable is a name given to a memory location. It is the basic unit of storage in a program.

Constants in C++:

Constants refer to fixed values that the program may not alter and they are called literals. **Constants** can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

Arithmetic operators in C++:

Operators which perform some arithmetic operations are called as Arithmetic operators. For example, Addition operator +, Subtraction Operator -, Multiplication Operator * and Division Operator / etc.

Datatype conversion in C++:

Data type conversion is also called as Type casting in C plus plus. It means to convert data type of one type into another datatype.

There are two types of data conversions in C++:

1. Implicit Type Conversion: Done by Compiler on it's own.
2. Explicit Type Conversion: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

Program:

Frist C++ program using variables, constants and arithmetic expressions:

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

int main()
{
    const double pi = 3.14; //Constant
    int r;                  //variables
    double a;

    cout << "Develop minimum 2 programs using constants variables
    arithmetic expression operators,exhibiting data type conversio
n "<<endl;
    cout << "Enter the Radius of circle: ";
    cin >> r;
    a = pi * r * r; //Using operators

    cout << "Araea of circle is : " << a << endl;

    return 0;
}
```

Second Program exhibiting datatype conversion:

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

int main()
{
    int x = 10;
    char y = 'A';
    int z;

    z = x + y;
    cout << "Practical 1.2: Develop minimum 2 programs using constan
ts, variables,"
        << " arithmetic expression operators,exhibiting data type c
onversion " << endl;
    cout << "The value of x: " << x << endl;
}
```

```
cout << "The value of y: " << y << endl;
cout << "The value of z: " << z << endl;

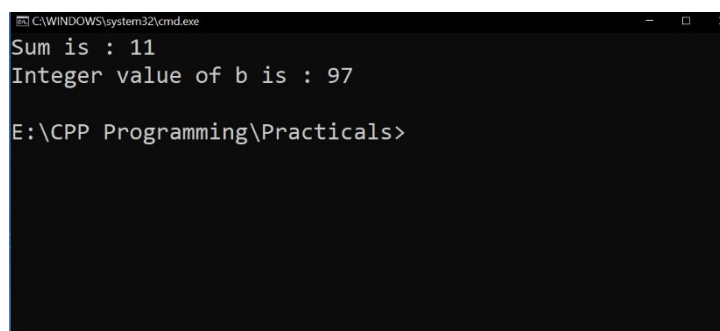
return 0;
}
```

Third program implementing explicit datatype conversion:

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

int main()
{
    float a = 1.2;
    int c;
    char b = 'a';
    cout << "Practical 1.3 :" << endl;
    c = (int)a + 10; //float a is Explicitly converted to int
    cout << "Sum is : " << c << endl;
    //Explicitly converting char to integer
    cout << "Integer value of b is : " << (int)b << endl;
    return 0;
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
Sum is : 11
Integer value of b is : 97

E:\CPP Programming\Practicals>
```

Result: In the above practical we have successfully used concepts such as variables, constants and arithmetic expression operators, exhibiting data type conversion.

Practical no 2.

Aim: Develop a program to implement decision making statements (if-else, switch).

Requirements: Computer system with minimal specifications and c++ development IDE (VS Code used).

Theory:

If-else statements in C++:

If-else statements in C++ are called as decision making statements in C++. They are used when we need to make some decisions and based on these decisions, we decide what should we do next.

Program:

Implementing if-else in C++.

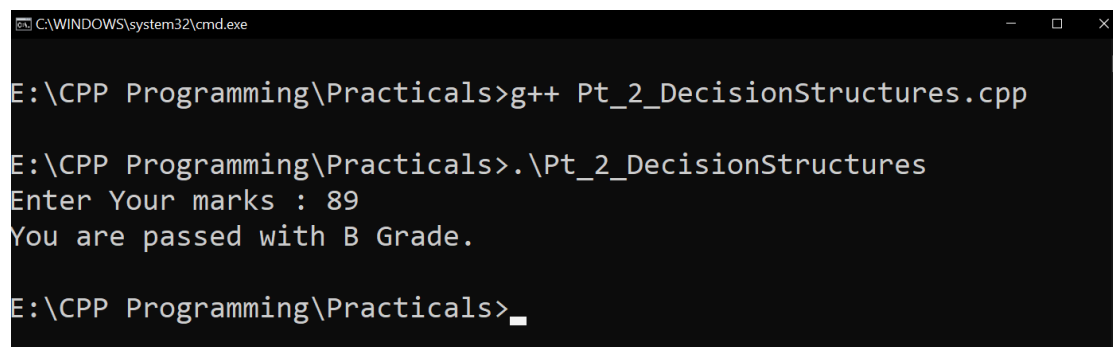
```
#include <iostream>

using namespace std;

int main()
{
    float marks;
    cout << "Practical 2:Develop a program to implement decision making statements(if-else, switch)" << endl;
    cout << "Enter Your marks : ";
    cin >> marks;
    //Demonstrating if-else in C++
    if (marks <= 100 && marks >= 90)
        cout << "You are passed with A Grade." << endl;
    else if (marks < 90 && marks >= 80)
        cout << "You are passed with B Grade." << endl;
    else if (marks < 80 && marks >= 70)
        cout << "You are passed with C Grade." << endl;
    else if (marks < 70 && marks >= 60)
        cout << "You are passed with D Grade." << endl;
```

```
else if (marks < 60 && marks >= 45)
    cout << "You are passed." << endl;
else if (marks < 45 && marks > 0)
    cout << "You are Fail." << endl;
else
    cout << "Invalid marks!!" << endl;
return 0;
}
```

Output :



```
C:\WINDOWS\system32\cmd.exe
E:\CPP Programming\Practicals>g++ Pt_2_DecisionStructures.cpp
E:\CPP Programming\Practicals>.\Pt_2_DecisionStructures
Enter Your marks : 89
You are passed with B Grade.
E:\CPP Programming\Practicals>_
```

Result: In the above practical we have successfully used concept of decision making statements such as if-else statement.

Practical no 3.

Aim: Develop a program to demonstrate control structures (for, while, do-while).

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Control structures statements in C++:

Control Structures are also called as loops in C++. Control Structure are used to execute a program several number of times. Loop executes the following single line or multiple line of code until the specified condition becomes False. There are three types of loop:

1) While loop 2) For loop 3) Do While loop.

Program:

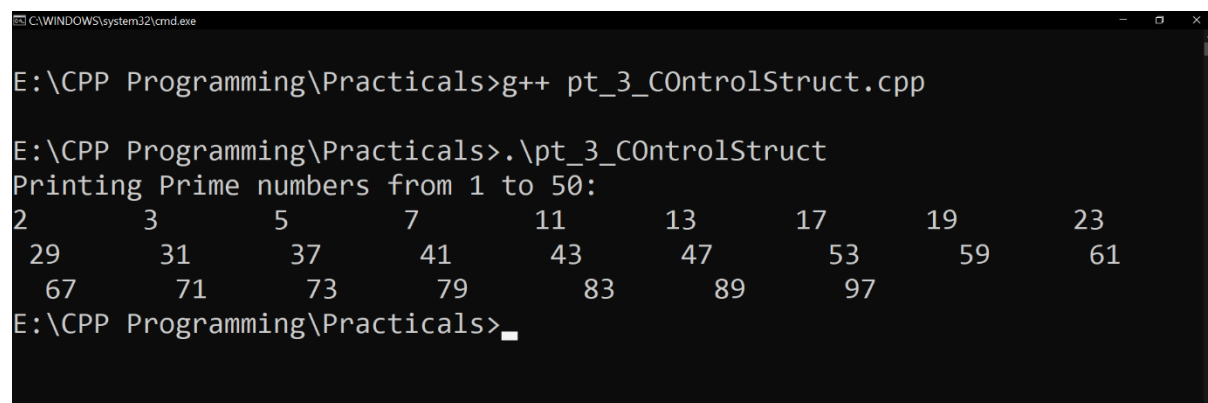
Implementing control structures in C++.

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

int main()
{
    cout << "Practical 3: Develop a program to demonstrate control s
tructures (for, while, do-while)." << endl;
    cout << "Printing Prime numbers from 1 to 50: " << endl;
    int i;
    for (i = 1; i <= 100; i++)
    {
        int count = 0;
        for (int j = 1; j <= i; j++)
        {
            if (i % j == 0)
                count++;
        }
    }
```

```
        if (count == 2)
        {
            cout << i << "\t";
        }
    }
    return 0;
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
E:\CPP Programming\Practicals>g++ pt_3_ControlStruct.cpp
E:\CPP Programming\Practicals>.\pt_3_ControlStruct
Printing Prime numbers from 1 to 50:
2      3      5      7      11     13     17     19     23
29     31     37     41     43     47     53     59     61
67     71     73     79     83     89     97
E:\CPP Programming\Practicals>
```

Result: In the above practical we have successfully used concept of control structure to print all prime nubers from 1 to 100 such for loop.

Practical no 4.

Aim: To develop a program to implement 1-dimension array.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Arrays in C++:

An array in C or C++ is a collection of items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They are used to store similar type of elements as in the data type must be the same for all elements. They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type. To add to it, an array in C or C++ can store derived data types such as the structures, pointers etc. Given below is the picturesque representation of an array.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

Program:

Implementing single dimensional Array .

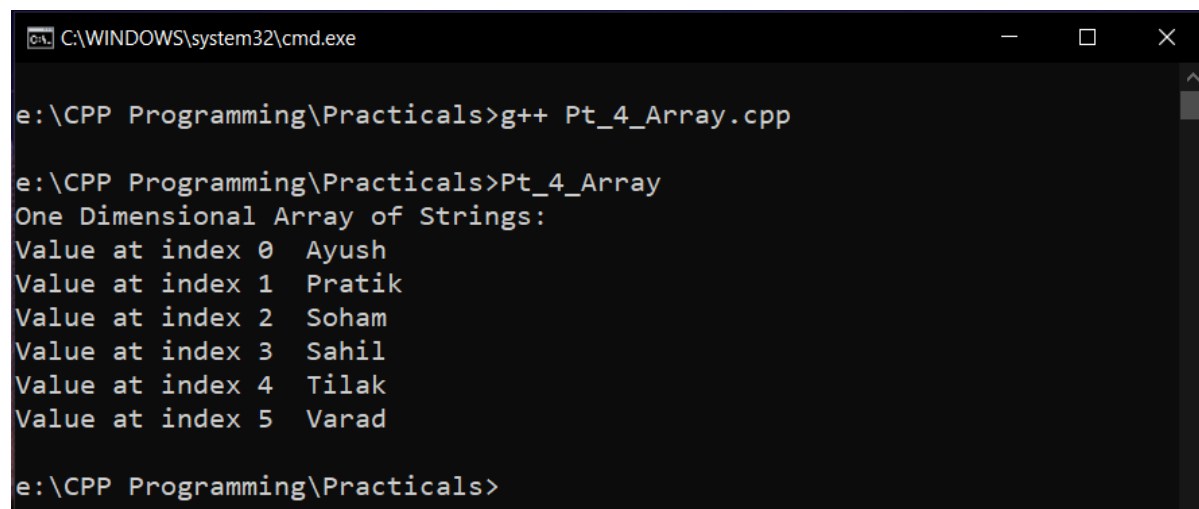
```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Practical No 4:Develop a program to implement 1-
dimension array\n"<<endl;

    //Array Decleration By specifizing size and initilizing Elements
    string arr[6] = {"Ayush", "Pratik", "Soham", "Sahil", "Tilak", "
Varad"};

    cout << "One Dimensional Array of Strings: " << endl;
    for (int i = 0; i <= 5; i++)
    {
        cout << "Value at index " << i << " " << arr[i] << endl;
    }
    return 0;
}
```

Output :



```
C:\WINDOWS\system32\cmd.exe

e:\CPP Programming\Practicals>g++ Pt_4_Array.cpp

e:\CPP Programming\Practicals>Pt_4_Array
One Dimensional Array of Strings:
Value at index 0  Ayush
Value at index 1  Pratik
Value at index 2  Soham
Value at index 3  Sahil
Value at index 4  Tilak
Value at index 5  Varad

e:\CPP Programming\Practicals>
```

Result: In the above practical we have successfully implemented the concept of single(one) dimensional array.

Practical no 5.

Aim: To develop a program to perform matrix operation using multi-dimensional array..

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Multi-dimensional arrays in C++:

In C/C++, we can define multidimensional arrays in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

General form of declaring N-dimensional arrays:

```
data_type array_name[size1][size2]....[sizeN];
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Program:

Implementing matrix operation using multi-dimensional Array.

```

#include <iostream>
using namespace std;

int main()
{
    int a[3][3];
    int b[3][3];
    int c[3][3];

    cout<<"Practical No 5: Develop a program to perform matrix operation using multi-dimensional array.\n"<<endl;

    //Taking first input
    cout << "Enter First matrix : " << endl;
    for (int i = 0; i <= 2; i++)
        for (int j = 0; j <= 2; j++)
            cin >> a[i][j];

    //taking second matrix input
    cout << "Enter Second matrix : " << endl;

    for (int i = 0; i <= 2; i++)
        for (int j = 0; j <= 2; j++)
            cin >> b[i][j];

    cout << "Entered 1st Matrix: " << endl;
    for (int i = 0; i <= 2; i++)
    {
        for (int j = 0; j <= 2; j++)
        {
            cout << a[i][j] << "\t";
        }
        cout << "\n";
    }
    cout << "Entered 2nd Matrix: " << endl;
    for (int i = 0; i <= 2; i++)
    {
        for (int j = 0; j <= 2; j++)
        {
            cout << b[i][j] << "\t";
        }
        cout << "\n";
    }
}

```

```

    }

    cout << "Addition of matrix: " << endl;
    for (int i = 0; i <= 2; i++)
    {
        for (int j = 0; j <= 2; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
            cout << c[i][j] << "\t";
        }
        cout << "\n";
    }

    cout << "Subtraction of matrix: " << endl;
    for (int i = 0; i <= 2; i++)
    {
        for (int j = 0; j <= 2; j++)
        {
            c[i][j] = a[i][j] - b[i][j];
            cout << c[i][j] << "\t";
        }
        cout << "\n";
    }

    return 0;
}

```

Output:

The screenshot shows a Windows command prompt window titled "E:\C++ Programming\Practicals\Pr_5_MatrixOperation.exe". The program is "Practical No 5: Develop a program to perform matrix operation using multi-dimensional array." The output is as follows:

```

Enter Frist matrix :
1 2 3
2 3 4
1 2 3
Enter Second matrix :
0 1 2
1 2 3
1 2 3
Entered 1st Matrix:
1      2      3
2      3      4
1      2      3
Entered 2nd Matrix:
0      1      2
1      2      3
1      2      3
Addition of matrix:
1      3      5
3      5      7
2      4      6
Subtraction of matrix:
1      1      1
1      1      1
0      0      0

```

The output is displayed in a black window with white text. The title bar of the window is highlighted in yellow and reads "Output Practical No 5".

Result: In the above practical we have successfully developed a program for matrix operation using multi-dimensional array.

Practical no 6.

Aim: To develop a program that implements a class and use it with objects.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Class in C++:

A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit, mileage* etc and member functions can be *apply brakes, increase speed* etc.

Objects:

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Program:

Implementing matrix operation using multi-dimensional Array .

```

#include <iostream>
using namespace std;

class Box //Class named Box
{
private:
    int H, B, L;

public:
    void setdata(int l, int b, int h)
    {
        H = h;
        B = b;
        L = l;
    }

    void getdata()
    {
        cout << "Length of Box is " << L << "cm" << endl;
        cout << "Breadth of Box is " << B << "cm" << endl;
        cout << "Height of Box is " << H << "cm" << endl;
    }

    int getVolume()
    {
        return (L * B * H);
    }
};

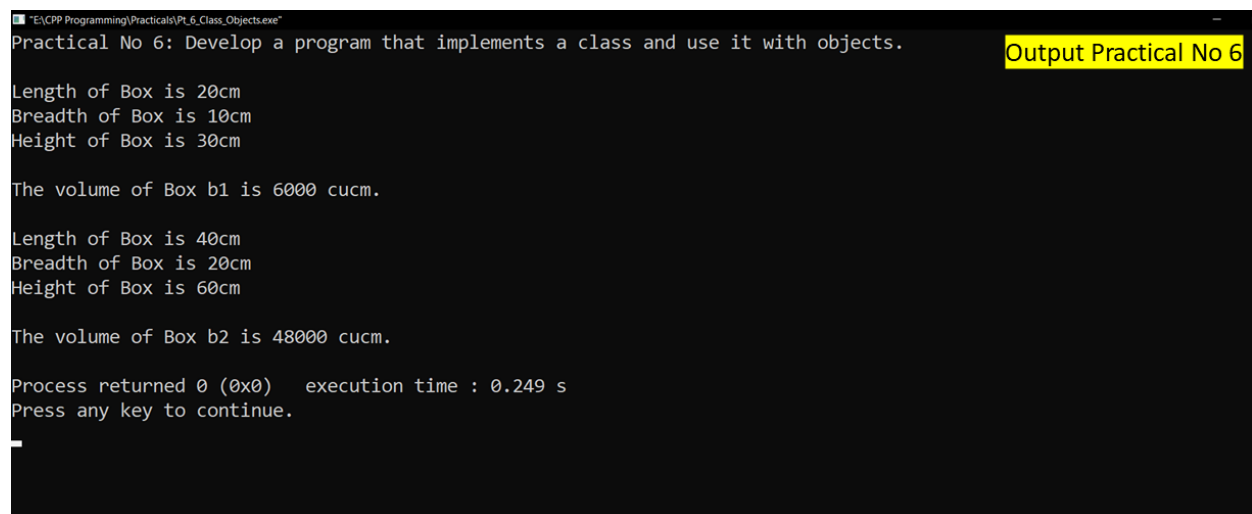
int main()
{
    cout<<"Practical No 6: Develop a program that implements a class
and use it with objects.\n"<<endl;
    Box b1; //Creating objects of class Box
    b1.setdata(20, 10, 30);
    b1.getdata();
    cout << "\nThe volume of Box b1 is " << b1.getVolume() << " cu cm
.\n"<< endl;

    Box b2;
    b2.setdata(40, 20, 60);
    b2.getdata();
}

```

```
        cout << "\nThe volume of Box b2 is " << b2.getVolume() << " cucm\n";  
    }  
    return 0;  
}
```

Output:



The screenshot shows the output of a C++ program titled "E:\CPP Programming\Practicals\PL_6_Class_Objects.exe". The program's purpose is stated as "Practical No 6: Develop a program that implements a class and use it with objects." The output displays the dimensions and volume for two box objects, b1 and b2. For b1, the dimensions are Length: 20cm, Breadth: 10cm, Height: 30cm, and Volume: 6000 cucm. For b2, the dimensions are Length: 40cm, Breadth: 20cm, Height: 60cm, and Volume: 48000 cucm. The program ends with "Process returned 0 (0x0) execution time : 0.249 s" and "Press any key to continue." A yellow label "Output Practical No 6" is visible in the top right corner of the screenshot.

```
E:\CPP Programming\Practicals\PL_6_Class_Objects.exe  
Practical No 6: Develop a program that implements a class and use it with objects.  
Length of Box is 20cm  
Breadth of Box is 10cm  
Height of Box is 30cm  
The volume of Box b1 is 6000 cucm.  
Length of Box is 40cm  
Breadth of Box is 20cm  
Height of Box is 60cm  
The volume of Box b2 is 48000 cucm.  
Process returned 0 (0x0) execution time : 0.249 s  
Press any key to continue.  
-
```

Result: In the above practical we have successfully developed a program which uses class and create objects of it.

Practical no 7.

Aim: To develop a program that implements a class and create array of objects.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Class in C++:

Like array of other user-defined data types, an array of type class can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

Program:

```
#include <iostream>

using namespace std;

class employee
{
private:
    char name[30];
    float age;
    char job[20];

public:
    void getdata();
    void putdata();
};

void employee::getdata()
{
    cout << "Enter name of Employee: ";
    cin >> name;
```

```

    cout << "Enter Job: ";
    cin >> job;
    cout << "Enter of age: ";
    cin >> age;
}
void employee::putdata()
{
    cout << "Name of Employee: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Job: " << job << endl;
}
int main()
{
    cout<<"Practical 7: Develop a program that implements a class and create array of objects\n"<<endl;
    const int size = 3;

    employee manager[size]; //declaring array of Obj

    for (int i = 0; i < size; i++)
    {
        cout << "\nEnter details of Employee no : " << i + 1 << endl;
        manager[i].getdata();
    }

    cout << "\nSaved details of employees:" << endl;
    for (int i = 0; i <= size; i++)
    {
        cout << "Details of manager: " << i + 1 << endl;
        manager[i].putdata();
    }

    return 0;
}

```

Output:

```
"E:\CPP Programming\Practicals\Pt_7_Arrayofobj.exe"
Practical 7: Develop a program that implements a class and create array of objects

Enter details of Employee no : 1
Enter name of Employee: Ayush
Enter Job: Manager
Enter of age: 34

Enter details of Employee no : 2
Enter name of Employee: HArdik
Enter Job: Assistant
Enter of age: 30

Enter details of Employee no : 3
Enter name of Employee: Vinay
Enter Job: Cleark
Enter of age: 28

Saved details of employees:
Details of manager: 1
Name of Employee: Ayush
Age: 34
Job: Manager
Details of manager: 2
Name of Employee: HArdik
Age: 30
Job: Assistant
Details of manager: 3
Name of Employee: Vinay
Age: 28
Job: Cleark
```

Result: In the above practical we have successfully developed a program which uses concept of array of objects.

Practical no 8.

Aim: To write a program to implement friend function.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Friend function in C++:

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

Program:

```
#include <iostream>

using namespace std;

class Complex
{
private:
    int a, b;

public:
    void setdata(int n1, int n2)
    {
        a = n1;
        b = n2;
    }
    //creating friend
    friend Complex sumComplex(Complex o1, Complex o2);
    void printNumber()
    {
```



```

        cout << a << " + " << b << "i" << endl;
    }
};
Complex sumComplex(Complex o1, Complex o2)
{
    Complex o3;
    o3.setdata((o1.a + o2.a), (o1.b + o2.b));
    return o3;
}
main()
{
    cout<<"Practical 8:Write a program to implement friend function.
\n"<<endl;
    Complex c1, c2, sum;

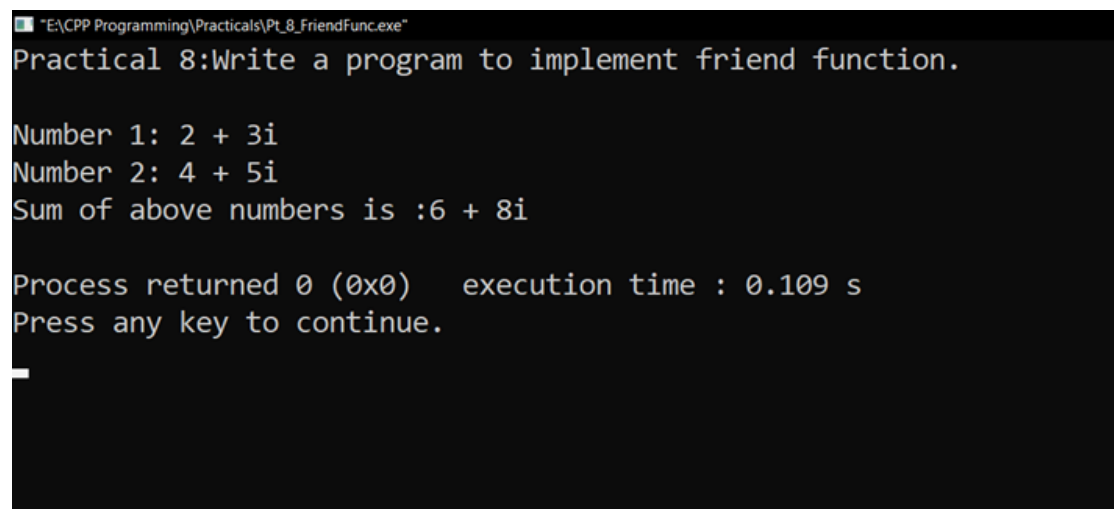
    c1.setdata(2, 3);
    cout << "Number 1: ";
    c1.printNumber();
    c2.setdata(4, 5);
    cout << "Number 2: ";
    c2.printNumber();

    sum = sumComplex(c1, c2);

    cout << "Sum of above numbers is :";
    sum.printNumber();
}

```

Output:



```

E:\CPP Programming\Practicals\Pt_8_FriendFunc.exe
Practical 8:Write a program to implement friend function.

Number 1: 2 + 3i
Number 2: 4 + 5i
Sum of above numbers is :6 + 8i

Process returned 0 (0x0)   execution time : 0.109 s
Press any key to continue.

```

Result: We implemented friend function to perform complex number addition with two different objects – c1 and c2.

Practical no 9.

Aim: To write a program to implement inline function.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Inline function in C++:

C++ **inline** function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time. Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality. To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line

Program:

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

class Operation
{
private:
    int a, b;

public:
    Operation(int x, int y)
    {
        a = x;
        b = y;
    }
}
```

```

    void sum();
    void diff();
    void mult();
    void div();
};
//creating inline functions
inline void Operation::sum()
{
    cout << "Sum of entered numbers is " << a + b << endl;
}
inline void Operation::diff()
{
    cout << "Difference of entered numbers is " << a - b << endl;
}
inline void Operation::mult()
{
    cout << "Multiplication of entered numbers is " << a * b << endl;
;
}
inline void Operation::div()
{
    cout << "Division of entered numbers is " << a / b << endl;
}
main()
{
    cout<<"Practical No 9: Write a program to implement inline funct
ion.\n"<<endl;
    int n1, n2;
    cout << "Enter num 1:";
    cin >> n1;
    cout << "Enter num 2:";
    cin >> n2;

    Operation op1(n1, n2);
    op1.sum();
    op1.diff();
    op1.mult();
    op1.div();
}

```

Output:

```
"E:\CPP Programming\Practicals\Pt_9_InlineFunctions.exe"  
Practical No 9: Write a program to implement inline function.  
  
Enter num 1:80  
Enter num 2:50  
Sum of entered numbers is 130  
Difference of entered numbers is 30  
Multiplication of entered numbers is 4000  
Division of entered numbers is 1  
  
Process returned 0 (0x0)   execution time : 12.119 s  
Press any key to continue.
```

Result: We created simple function which implements inline functions.

Practical no 10.

Aim: To write a program to implement Constructor for creation of Object.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

Constructor in C++:

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class. Constructor has same name as of class.

Program:

```
#include <iostream>
#define PI 3.141592653589793238
using namespace std;
class Circle
{
private:
    double R;

public:
    Circle(double r)
    {
        R = r;
        cout<<"Radius of Circle: "<<R<<endl;
    }

    void area()
    {
        double area = PI * R * R;
        cout << "Area of Circle is " << area << endl;
    }
    void circumference()
    {
```

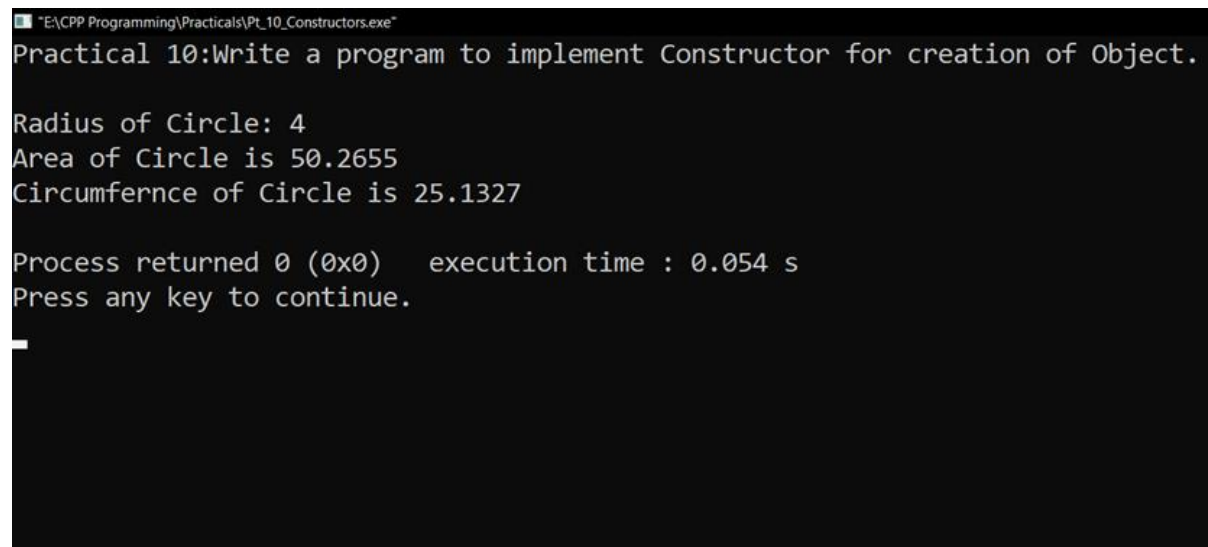
```

        double circumference = 2 * PI * R;
        cout << "Circumfernce of Circle is " << circumference << endl;
    }
};
int main()
{
    cout<<"Practical 10:Write a program to implement Constructor for
creation of Object.\n"<<endl;
    Circle c1(4);
    c1.area();
    c1.circumference();

    return 0;
}

```

Output:



The screenshot shows a Windows command prompt window titled "E:\CPP Programming\Practicals\Pt_10_Constructors.exe". The output of the program is as follows:

```

Practical 10:Write a program to implement Constructor for creation of Object.

Radius of Circle: 4
Area of Circle is 50.2655
Circumfernce of Circle is 25.1327

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.

```

Result: We a program in which a constructor created to assign value to objects.

Practical no 11.

Aim: To write a program to implement Multiple Constructors.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: We can create multiple constructors of same class with same name unless and until we have different arguments in the constructors. It may occur a condition that we have to initialize different no of variables at different time at that time we can use concept of Multiple Constructors.

Program:

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

class shape
{
private:
    int l, b;

public:
    shape(int L, int B)
    {
        l = L;
        b = B;
    }
    shape(int S)
    {
        l = S;
        b = S;
    }
    int area()
    {
        int a = l * b;
        return a;
    }
    int perimeter()
    {
```



```

        int p = 2*(l + b);
        return p;

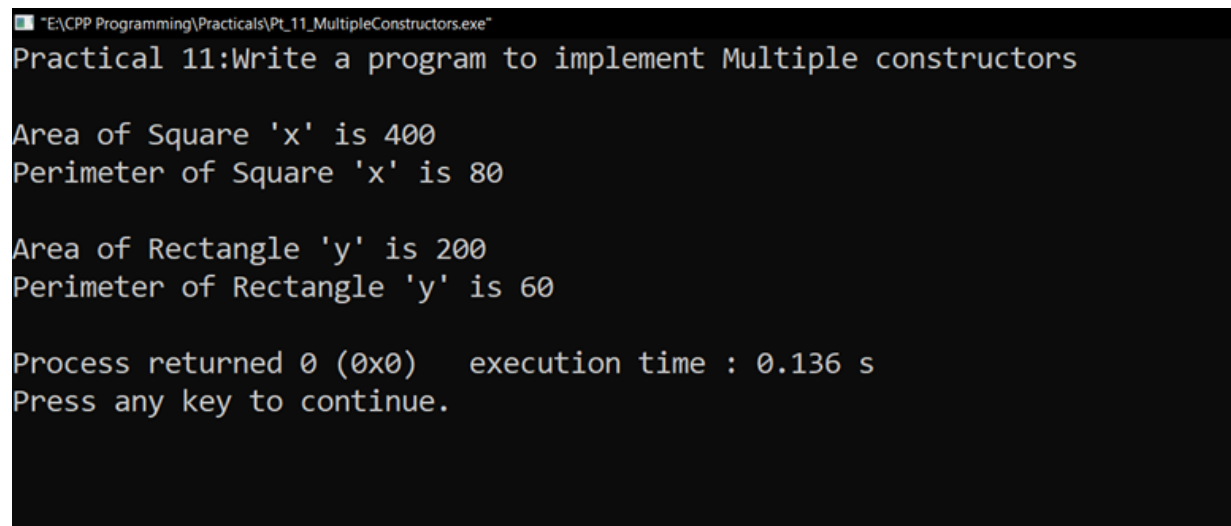
    }
};

int main()
{
    cout<<"Practical 11:Write a program to implement Multiple constructors\n"<<endl;
    shape x(20);
    cout << "Area of Square 'x' is " << x.area() << endl;
    cout << "Perimeter of Square 'x' is " << x.perimeter() << endl;

    shape y(10, 20);
    cout << "\nArea of Rectangle 'y' is " << y.area() << endl;
    cout << "Perimeter of Rectangle 'y' is " << y.perimeter() << endl;
    return 0;
}

```

Output:



```

E:\CPP Programming\Practicals\Pt_11_MultipleConstructors.exe
Practical 11:Write a program to implement Multiple constructors

Area of Square 'x' is 400
Perimeter of Square 'x' is 80

Area of Rectangle 'y' is 200
Perimeter of Rectangle 'y' is 60

Process returned 0 (0x0)   execution time : 0.136 s
Press any key to continue.

```

Practical no 12.

Aim: To write a program to implement Copy Constructors.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: A copy constructor is used to declare and initialize an object from another object. For example, the statement → `integer l2(l1);` would define the object l2 and at the same time initialize it to the values of l1. Another form of this statement is `integer l2 = l1;`

Program:

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

class vehical
{
private:
    int wheels;
    int doors;
    int seats;

public:
    vehical() {}
    vehical(int w, int d, int s);
    vehical(vehical &obj, int s); //Copy Constructor
    void display();
};

vehical::vehical(int w, int d, int s)
{
    wheels = w;
    doors = d;
    seats = s;
}

vehical::vehical(vehical &obj, int s)
{
    wheels = obj.wheels;
    doors = obj.doors;
```

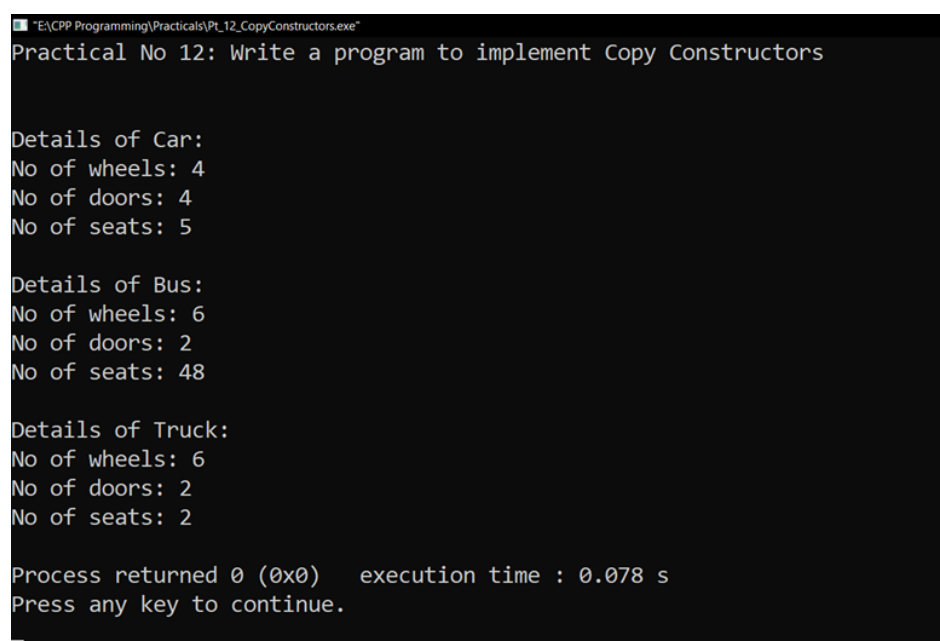
```

        seats = s;
    }
    void vehical::display()
    {
        cout << "No of wheels: " << wheels << endl;
        cout << "No of doors: " << doors << endl;
        cout << "No of seats: " << seats << endl;
    }
    main()
    {
        cout << "Practical No 12: Write a program to implement Copy Constructors\n" << endl;
        vehical car, truck, bus;
        car = vehical(4, 4, 5);
        bus = vehical(6, 2, 48);
        truck = vehical(bus, 2);

        cout << "\nDetails of Car: " << endl;
        car.display();
        cout << "\nDetails of Bus: " << endl;
        bus.display();
        cout << "\nDetails of Truck: " << endl;
        truck.display();
    }
}

```

Output:



```

E:\CPP Programming\Practicals\Pr12_CopyConstructors.exe
Practical No 12: Write a program to implement Copy Constructors

Details of Car:
No of wheels: 4
No of doors: 4
No of seats: 5

Details of Bus:
No of wheels: 6
No of doors: 2
No of seats: 48

Details of Truck:
No of wheels: 6
No of doors: 2
No of seats: 2

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.

```

Result: We used the copy constructor concepts to create the object of class vehicle.

Practical no 13.

Aim: Write a program to implement constructor overloading with destructor.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: A copy constructor is used to declare and initialize an object from another object. For example, the statement \rightarrow `integer l2(l1);` would define the object l2 and at the same time initialize it to the values of l1. Another form of this statement is `integer l2 = l1;`

Program:

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

class Cube
{
private:
    int a;

public:
    Cube()
    {
        a = 1;
        cout << "\nCube created with side : " << a << endl;
    }
    Cube(int s)
    {
        a = s;
        cout << "\nCube created with side : " << a << endl;
    }
    void getArea()
    {
        cout << "The area of cube is: " << 6 * a * a << " squareunit
s." << endl;
    }
    void getVolume()
    {
        cout << "The volume of cube is: " << a * a * a << " cubicuni
ts." << endl;
    }
}
```

```

    }
    ~Cube()
    {
        cout << "Cube of side " << a << " is destroyed!" << endl;
    }
};
int main()
{
    int s;
    cout << "Practical No 13: Write a program to implement construct
or overloading with destructor" << endl;

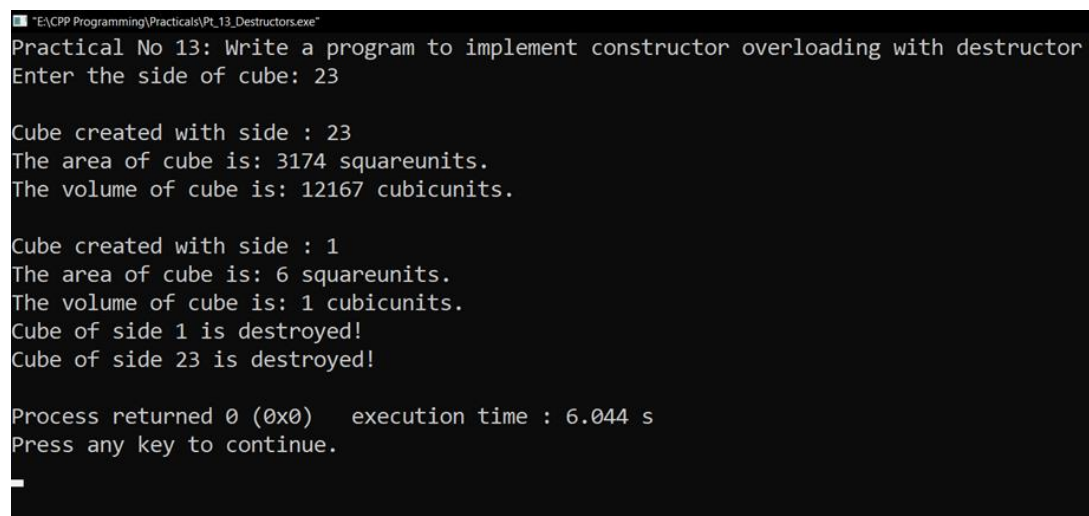
    cout << "Enter the side of cube: ";
    cin >> s;
    Cube c1(s);
    c1.getArea();
    c1.getVolume();

    Cube c2;
    c2.getArea();
    c2.getVolume();

    return 0;
}

```

Output:



```

E:\C++ Programming\Practicals\Plt_13_Destructors.exe
Practical No 13: Write a program to implement constructor overloading with destructor
Enter the side of cube: 23

Cube created with side : 23
The area of cube is: 3174 squareunits.
The volume of cube is: 12167 cubicunits.

Cube created with side : 1
The area of cube is: 6 squareunits.
The volume of cube is: 1 cubicunits.
Cube of side 1 is destroyed!
Cube of side 23 is destroyed!

Process returned 0 (0x0)   execution time : 6.044 s
Press any key to continue.

```

Practical no 14.

Aim: Write a program to demonstrate operator overloading for Unary operator.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: We can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

Program:

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

class Team
{
private:
    int points;
    int stat;

public:
    Team(int p, int s)
    {
        points = p;
        stat = s;
    }
    void getdata()
    {
        cout << "Points are: " << points << endl;
        cout << "Status is: " << stat << endl;
    }
    void operator++()
    {
        ++points;
    }

    void operator--()
    {
        --points;
    }
}
```

```

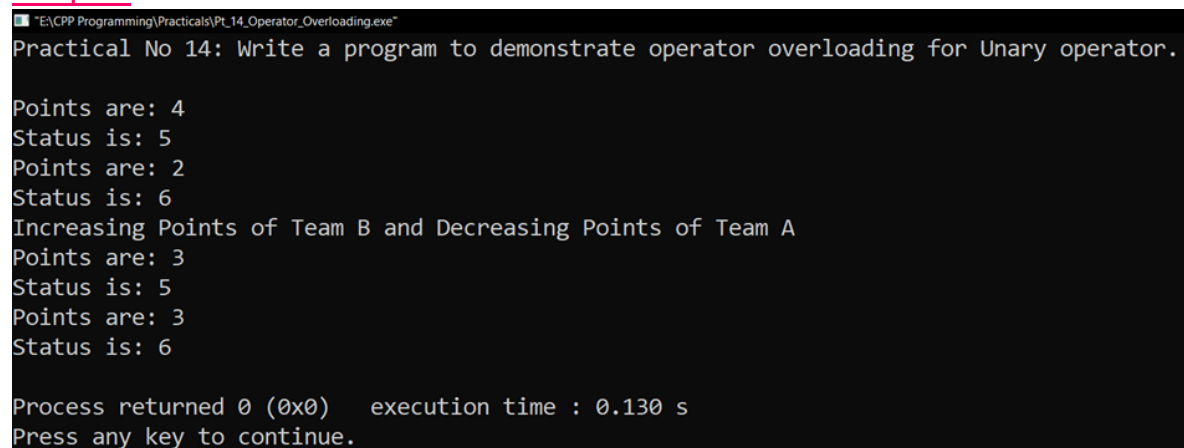
    }
};
int main()
{
    cout << "Practical No 14: Write a program to demonstrate operator overloading for Unary operator.\n" << endl;

    Team A(4, 5), B(2, 6);
    A.getdata();
    B.getdata();
    --A;
    ++B;
    cout << "Increasing Points of Team B and Decreasing Points of Team A" << endl;
    A.getdata();
    B.getdata();

    return 0;
}

```

Output:



```

E:\C++ Programming\Practicals\Pt_14_Operator_Overloading.exe
Practical No 14: Write a program to demonstrate operator overloading for Unary operator.

Points are: 4
Status is: 5
Points are: 2
Status is: 6
Increasing Points of Team B and Decreasing Points of Team A
Points are: 3
Status is: 5
Points are: 3
Status is: 6

Process returned 0 (0x0)   execution time : 0.130 s
Press any key to continue.

```

Result:

We successfully overloaded the unary operators ++ and – in the above program.

Practical no 15.

Aim: Write a program to demonstrate operator overloading for Binary operator.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Binary operators work on two operands. For example. $C = A + 10$

Here, + is a binary operator that works on the operands A and 10.

When we overload the binary operator for user-defined types by using the code.

Object C = Object A + Object B The operator function is called using the Object A object and Object B is passed as an argument to the function.

Program:

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

class Circle
{
private:
    float r, area;

public:
    Circle(float R = 0)
    {
        r = R;
    }
    void display()
    {
        cout << "The Radius of Circle is: " << r << endl;
    }
    float getarea()
    {
        area = 3.14 * r * r;
        return area;
    }
    Circle operator+(Circle b)
    {
        Circle t;
```

```

        t.r = r + b.r;
        return t;
    }
    Circle operator-(Circle b)
    {
        Circle t;
        t.r = r - b.r;
        return t;
    }
};
int main()
{
    cout << "Practical 15: Write a program to demonstrate operator o
verloading for Binary operator.\n"
        << endl;

    Circle a(10),
        b(2), c;
    cout << "The value of Circle 'a' " << endl;
    a.display();
    cout << "\nThe value of Circle 'b' " << endl;
    b.display();

    c = a + b;
    cout << "\nThe Valuesm of a+b is: " << endl;
    c.display();
    cout << "The area of circle c is: " << c.getarea() << endl;
    return 0;
}

```

Output:

```
"E:\CPP Programming\Practicals\Pr_15_BinaryOperatorOverloading.exe"
Practical 15: Write a program to demonstrate operator overloading for Binary operator.

The value of Circle 'a'
The Radius of Circle is: 10

The value of Circle 'b'
The Radius of Circle is: 2

Thev Valuesm of a+b is:
The Radius of Circle is: 12
The area of circle c is: 452.16

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
_
```

Result: We successfully overloaded the binary operator+ in the above program.

Practical no 16.

Aim: Write a program for implementing single inheritance.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: **Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class. For E.g. Class A is Derived from Class B. $B \rightarrow A$.

Program:

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

class Shape
{
protected:
    int breadth;
    int length;
    int side;

public:
    void setdata(int S)
    {
        side = S;
    }
    void setdata(int L, int B)
    {
        breadth = B;
        length = L;
    }
};

class Rectangle : public Shape
{
public:
    void getdata()
    {
        cout<<"Length of Rectangle: "<<length<<endl;
        cout<<"Breadth of Rectangle: "<<breadth<<endl;
    }
}
```

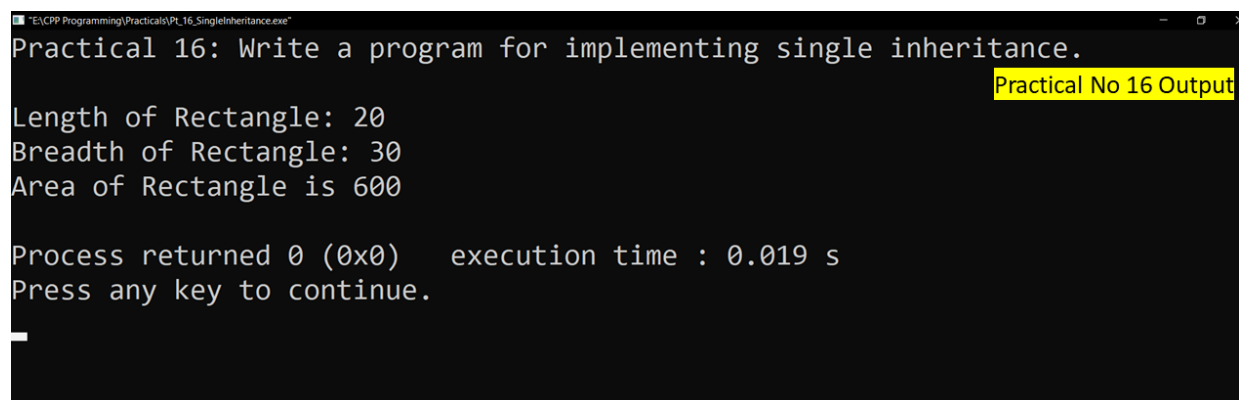
```

void rectarea()
{
    int area;
    area = length * breadth;
    cout << "Area of Rectangle is " << area << endl;
}
};
main()
{
    cout << "Practical 16: Write a program for implementing single i
nheritance.\n"
        << endl;
    Rectangle a;
    a.setdata(20, 30);
    a.getdata();
    a.rectarea();

    return 0;
}

```

Output:



The screenshot shows a terminal window titled "E:\CPP Programming\Practicals\Pl_16_SingleInheritance.exe". The output text is as follows:

```

Practical 16: Write a program for implementing single inheritance.
Length of Rectangle: 20
Breadth of Rectangle: 30
Area of Rectangle is 600
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.

```

A yellow highlight box with the text "Practical No 16 Output" is positioned over the output text.

Result: Class Rectangle successfully Inherited From class shape.

Practical no 17.

Aim: Write a program for implementing multi-level inheritance.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Multilevel inheritance refers to a mechanism in OOP where one can inherit from a derived class, thereby making this derived class the base class for the new class. For e.g. C is subclass or child class of B and B is a child class of A.

Program:

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

class Employee
{
protected:
    int id;

public:
    void setId(int a)
    {
        id = a;
    }
    void putId()
    {
        cout << "Id of Employee is: " << id << endl;
    }
};

class Salary : public Employee
{
protected:
    int sal;
    int bonus;

public:
    void setsalary(int s, int b)
    {
        sal = s;
```

```

        bonus = b;
    }
    void getsalary()
    {
        cout << "Salary is :" << sal << endl;
        cout << "Bonus is: " << bonus << endl;
    }
};
class Package : public Salary
{
private:
    float package;

public:
    void getPackage()
    {
        package = (sal * 12) + (2 * bonus);
        cout << "The Yearly Package is : " << package << endl;
    }
};
int main()
{
    cout << "Practical 17: Write a program for implementing multi level inheritance.\n"
        << endl;
    Package Employee1;
    Employee1.setId(7);
    Employee1.setsalary(50000, 4000);
    Employee1.putId();
    Employee1.getsalary();
    Employee1.getPackage();

    return 0;
}

```

Output:

```
"E:\CPP Programming\Practicals\Pt_17_MultilevelInheritance.exe"
Practical 17: Write a program for implementing multi-level inheritance.

Id of Employee is: 7
Salary is :50000
Bonus is: 4000
The Yearly Package is : 608000

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Result: In Above we used Multilevel Inheritance to Inherit Class Package from class Salary Class which was Inherited from class Employee.

Practical no 18.

Aim: Write a program for implementing multiple inheritances.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can inherit characteristics and features from more than one parent object or parent class.

Program:

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

class Shape
{
protected:
    int breadth;
    int length;

public:
    void setdata(int L, int B)
    {
        breadth = B;
        length = L;
    }
};

class cuttingCost
{
public:
    float getCuttingCost(float area)
    {
        return area * 90;
    }
};

class Field : public Shape, public cuttingCost
{
public:
    float fieldarea()
```

```

    {
        return (length * breadth);
    }
};

main()
{
    cout << "Practical 18: Write a program for implementing multiple inheritance.\n"
        << endl;

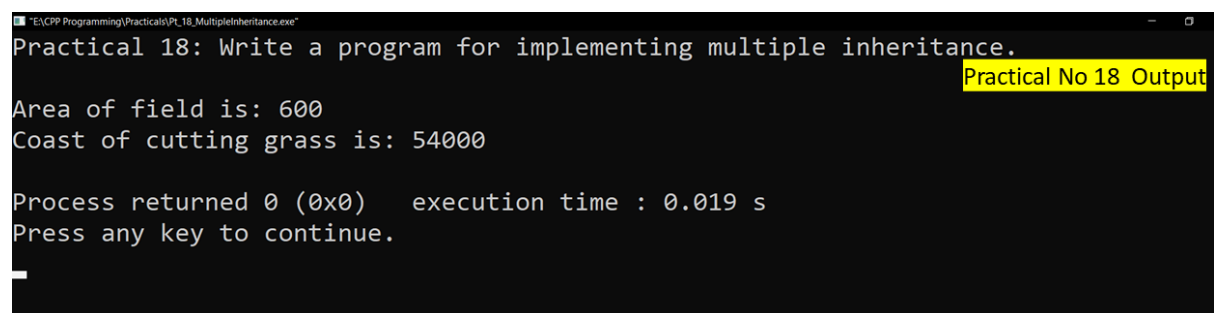
    float area, coast;
    Field a;
    a.setdata(20, 30);
    area = a.fieldarea();
    coast = a.getCuttingCost(area);

    cout << "Area of field is: " << area << endl;
    cout << "Coast of cutting grass is: " << coast << endl;

    return 0;
}

```

Output:



```

E:\C++ Programming\Practicals\Pl_18_MultipleInheritance.exe
Practical 18: Write a program for implementing multiple inheritance.
Area of field is: 600
Coast of cutting grass is: 54000
Process returned 0 (0x0) execution time : 0.019 s
Press any key to continue.

```

Result: We used the concept of Multilevel inheritance in inheriting class Field from class Shape and Class Cutting cost.

Practical no 19.

Aim: Write a program for implementing Constructor in Derived class.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: A Derived class constructor has access only to its own class members, but a Derived class object also have inherited property of Base class, and only base class constructor can properly initialize base class members. Hence all the constructors are called, else object wouldn't be constructed properly.

Program:

```
#include <iostream>
#include<string.h>

using namespace std;

class Vehicle
{
protected:
    int id;

public:
    Vehicle(int n)
    {
        id = n;
    }
};

class Bike : public Vehicle
{
private:
    char company_name[10];

public:
    Bike(char c[],int n) : Vehicle(n)
    {
        strcpy(company_name,c);
    }
    void getDetails()
```

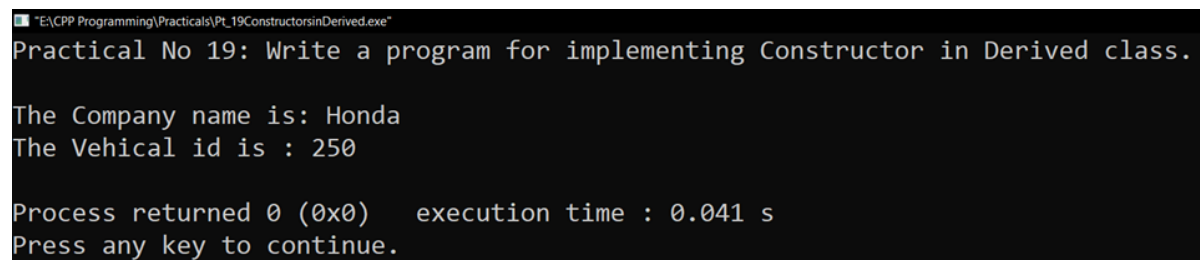
```

    {
        cout << "The Company name is: " << company_name << endl;
        cout << "The Vehical id is : " << id << endl;
    }
};
int main()
{
    cout<<"Practical No 19: Write a program for implementing Constru
ctor in Derived class.\n"<<endl;
    Bike a("Honda", 250);
    a.getDetails();

    return 0;
}

```

Output:



The screenshot shows a terminal window with the following output:

```

Practical No 19: Write a program for implementing Constructor in Derived class.

The Company name is: Honda
The Vehical id is : 250

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.

```

Result:

We implemented constructor in the derived class Bike.

Practical no 20.

Aim: Develop minimum 1 program to demonstrate Pointer to object.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory:

A Pointer is a variable which points to another variable (stores the address of pointing variable). A pointer holds the address of an object stored in memory. The pointer then simply “points” to the object. The type of the object must correspond with the type of the pointer. The & character specifies that we are storing the address of the variable succeeding it.

Program:

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

class Field
{
private:
    int l, b;

public:
    Field(int L, int B)
    {
        l = L;
        b = B;
    }
    int getCoast()
    {
        return (200 * (l * b));
    }
};

int main()
{

    cout<<"Practical No 20: Develop minimum 1 program to demonstrate
    Pointer to object.\n"<<endl;
```

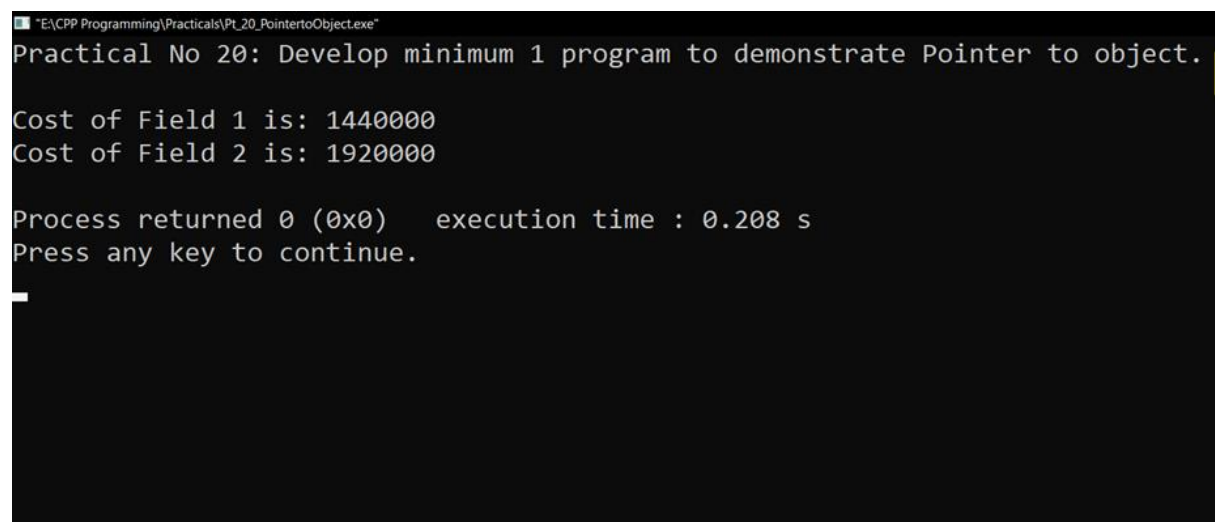
```
Field f1(90, 80);
Field f2(120, 80);

Field *ptr = &f1;
cout << "Cost of Field 1 is: " << ptr->getCoast() << endl;

ptr = &f2;
cout << "Cost of Field 2 is: " << ptr->getCoast() << endl;

return 0;
}
```

Output:



The screenshot shows a terminal window titled "E:\CPP Programming\Practicals\Pt_20_PointertoObject.exe". The output text is as follows:

```
Practical No 20: Develop minimum 1 program to demonstrate Pointer to object.

Cost of Field 1 is: 1440000
Cost of Field 2 is: 1920000

Process returned 0 (0x0)   execution time : 0.208 s
Press any key to continue.
```

Result:

We created a pointer variable *ptr which points to object f1.

Practical no 21.

Aim: Develop minimum 1 program to demonstrate Pointer to derived class.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: A base class pointer can point to a derived class object, but we can only access base class member or virtual functions using the base class pointer because object slicing happens when a derived class object is assigned to a base class object.

Program:

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

class Employee
{
protected:
    char N[25];
    int id;
    int sal;
};

class Manager : public Employee
{
private:
    int branch_code;

public:
    void setdata(char n[], int i, int s, int b)
    {
        strcpy(N, n);
        id = i;
        sal = s;
        branch_code = b;
    }
    void display()
    {
        cout << "Manager name: " << N << endl;
        cout << "Id: " << id << endl;
    }
}
```

```

        cout << "Brance Code: " << branch_code << endl;
        cout << "Salary: " << sal << endl;
    }
};

int main()
{
    cout<<"Practical No 21: Develop minimum 1 program to demonstrate
    Pointer to derived class."<<endl;

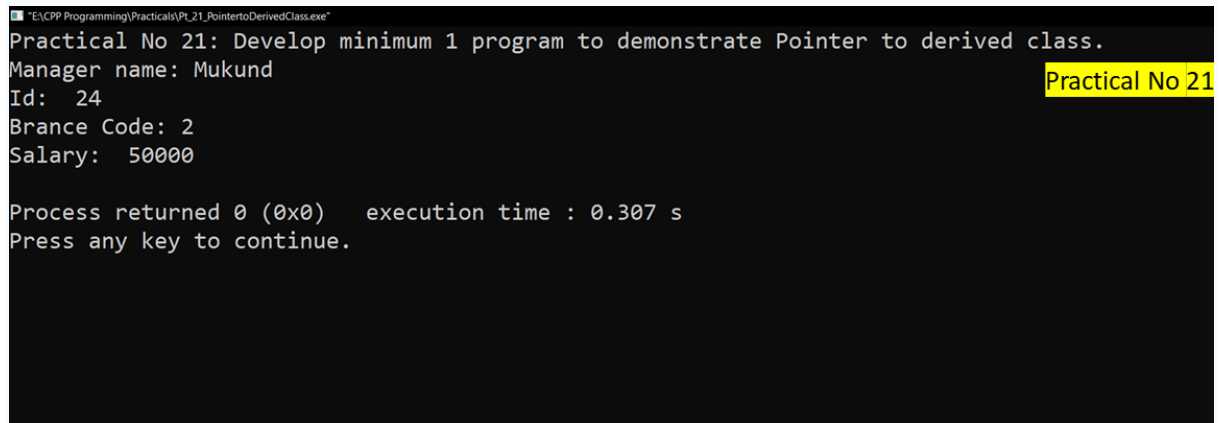
    Manager *manager_pointer;

    Manager manager1;

    manager_pointer = &manager1;
    manager_pointer->setdata("Mukund", 24, 50000, 2);
    manager_pointer->display();
    return 0;
}

```

Output:



```

E:\CPP Programming\Practicals\Pl_21_PointertoDerivedClass.exe
Practical No 21: Develop minimum 1 program to demonstrate Pointer to derived class.
Manager name: Mukund
Id: 24
Brance Code: 2
Salary: 50000

Process returned 0 (0x0)   execution time : 0.307 s
Press any key to continue.

```

Result: We created a pointer which points to derived class object manager1.

Practical no 22.

Aim: Develop minimum 1 program to Implement Compile time polymorphism.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Whenever an object is bound with their functionality at the compile-time, this is known as the compile-time polymorphism. At compile-time, compiler knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding.

Program:

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

class Circle
{
private:
    float R;

public:
    Circle(int r)
    {
        R = r;
        cout << "R = " << r << endl;
    }
    void area()
    {
        double a = R * 3.114 * R;
        cout << "R = " << R << endl;
        cout << "Area of Circle is " << a << endl;
    }

    void operator++()
    {
        R++;
    }
    void operator--()
    {
        --R;
    }
}
```

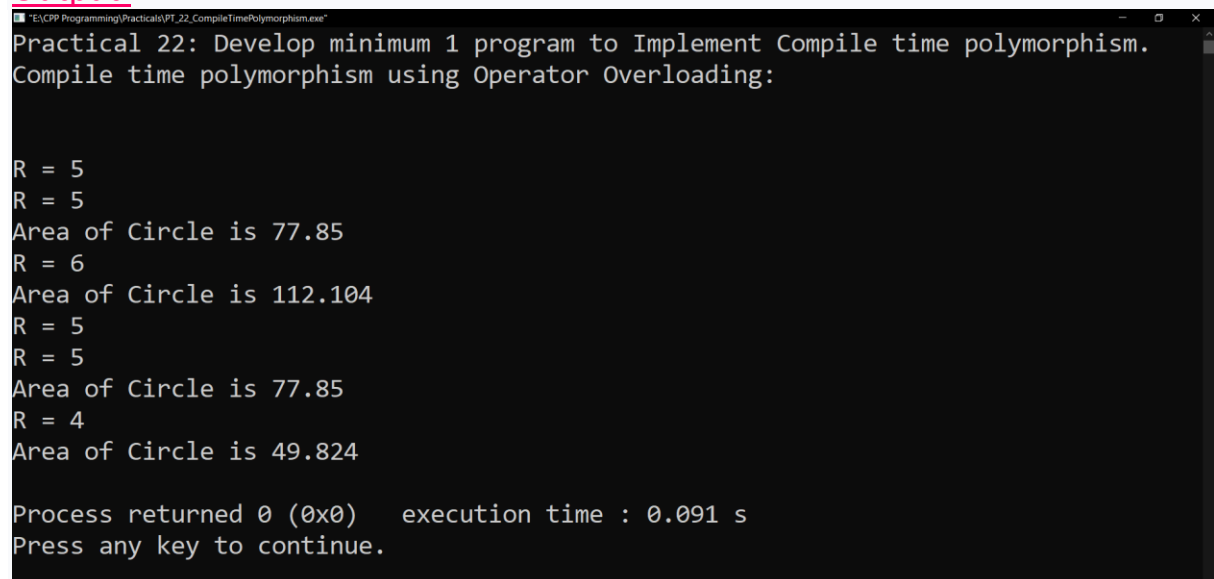
```

    }
};
main()
{
    cout << "Practical 22: Develop minimum 1 program to Implement Co
mpile time polymorphism." << endl;
    cout << "Compile time polymorphism using Operator Overloading:\n
\n"<< endl;
    Circle a(5);
    a.area();
    ++a;
    a.area();

    Circle b(5);
    b.area();
    --b;
    b.area();
}

```

Output:



```

Practical 22: Develop minimum 1 program to Implement Compile time polymorphism.
Compile time polymorphism using Operator Overloading:

R = 5
R = 5
Area of Circle is 77.85
R = 6
Area of Circle is 112.104
R = 5
R = 5
Area of Circle is 77.85
R = 4
Area of Circle is 49.824

Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.

```

Result:

We achieved compile time polymorphism using the operator overloading.

Practical no 23.

Aim: Develop minimum 1 program to Implement run time polymorphism using virtual functions.

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Runtime polymorphism: This type of polymorphism is achieved by Function Overriding. Function overriding on the other hand occurs when a derived class has a definition for one of the member functions of the base class.

Program:

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

class BaseClass
{
protected:
    int var_base = 10;

public:
    virtual void display()
    {
        cout << "Value of Base class variable: " << var_base << endl
;
    }
};

class DerivedClass : public BaseClass
{
    int var_derived = 20;

public:
    void display()
    {
        cout << "Value of derived class variable: " << var_derived <
< endl;
    }
};

int main()
```

```

{
    BaseClass *base_class_pointer;
    BaseClass base_class_obj;

    DerivedClass *derived_class_pointer;
    DerivedClass derived_class_obj;

    base_class_pointer = &derived_class_obj;

    base_class_pointer->display();
    return 0;
}

```

Output:

```

E:\CPP Programming\Practicals\Pr_23_RunTimePoly.exe
Practical 22: Develop minimum 1 program to Implement run time polymorphism using virtual functions.
Value of derived class variable: 20
Value of derived class variable: 20

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.

```

Result: Used the virtual base class to call the function of derived class by the base class pointer.

Practical no 24.

Aim: Write a program to read and write data to and from a file

Requirements: Computer system with minimal specifications and C++ development IDE (VS Code used).

Theory: Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it. ... In C++ we have a set of file handling methods. These include ifstream, ofstream, and fstream

Program:

```
#include <iostream>
#include <string.h>
#include <fstream>

using namespace std;

class Student
{
private:
    int StudentId;
    char StudentName[20];

public:
    void setData()
    {
        cout << "Enter StudentId: ";
        cin >> StudentId;
        cout << "Enter Student Name: ";
        cin.ignore();
        cin.getline(StudentName, 19);
        cout << "Student Initilized!!"<<endl;
    }
    void readData()
    {
        cout << "Student Id: " << StudentId << endl;
        cout << "Student Name: " << StudentName << endl;
    }
};
```

```

int main()
{
    cout<<"Practical 24: Write a program to read and write data to a
nd from a file\n" << endl;
    Student student1;
    student1.setData();

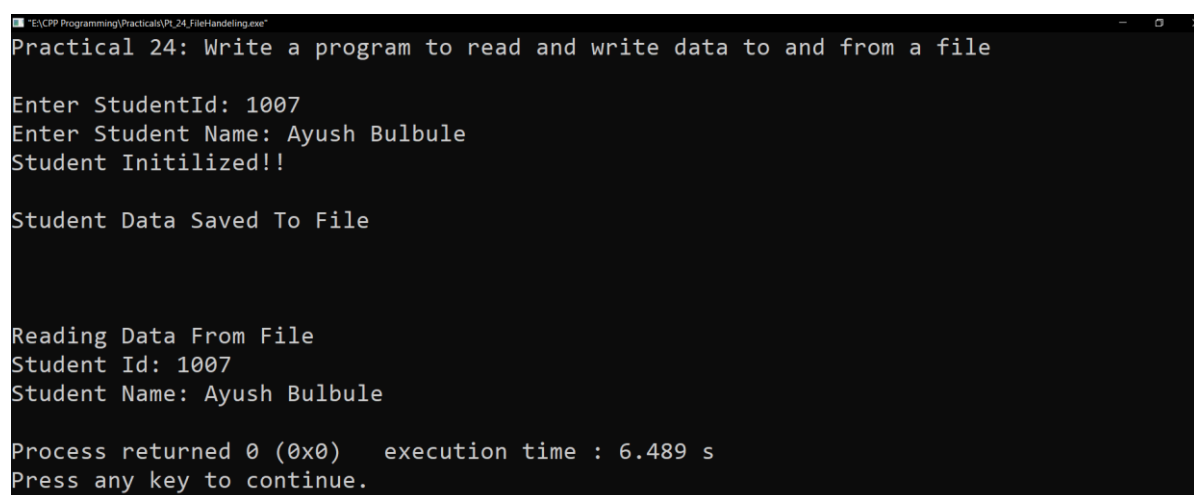
    ofstream fout;
    fout.open("studentData.txt", ios::app);
    fout.write((char *)&student1, sizeof(student1));
    cout<<"\nStudent Data Saved To File\n"<<endl;
    fout.close();

    cout<<"\n\nReading Data From File"<<endl;
    ifstream fin;
    fin.open("studentData.txt", ios::in);
    fin.read((char *)&student1, sizeof(student1));
    fin.seekg(0);
    student1.readData();

    return 0;
}

```

Output:



```

Practical 24: Write a program to read and write data to and from a file

Enter StudentId: 1007
Enter Student Name: Ayush Bulbule
Student Initilized!!

Student Data Saved To File

Reading Data From File
Student Id: 1007
Student Name: Ayush Bulbule

Process returned 0 (0x0) execution time : 6.489 s
Press any key to continue.

```

Result: We Initialized data to a object and then saved it to file.
