VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
Jnana Sangama, Belagavi – 590018

KLS VISHWNATHRAO DESHPANDE INSTITUTE OF TECHNOLOGY,
Haliyal – 583 219, Uttara Kannada
(Accredited by NAAC with "A" Grade)

LABORATORY MANUAL

Course Title	:	**C++ Basics Laboratory**
Course Code	:	**BEC358C**
Year / Semester	:	**2nd Year / 3rd Sem**
Academic Year	:	**2025-26**
Course In-Charge	:	**Prof. Vijayalaxmi C. Kalal**

Department of Electronics and Communication Engineering

Kalal 15/07/2025
Signature of the Faculty with Date

HoD
Head of the Department
Dept. of Electronic & Communication Engg
KLS V.D.I.T. HALIYAL (U.K.)

KLS Vishwanathrao Deshpande Institute of Technology

(Accredited by NAAC with "A" Grade)
(Approved by AICTE, New Delhi, Affiliated to VTU, Belagavi)
(Recognized Under Section 2(f) by UGC, New Delhi)
Udyog Vidya Nagar, Haliyal - 581 329, Dist.: Uttara Kannada
Phone: 08284 - 220861, 220334, 221409, Fax: 08284 - 220813
www.klsvdit.edu.in | principal@klsvdit.edu.in | hodece@klsvdit.edu.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

College Vision and Mission Statements

Vision

To nurture talent & enrich society through excellence in technical education, research & innovation.

Mission

1. To augment innovative pedagogy & kindle quest for interdisciplinary learning & to enhance conceptual understanding.
2. To build competence, professional ethics & develop entrepreneurial thinking.
3. To strengthen industry institute partnership & explore global collaborations.
4. To inculcate culture of socially responsible citizenship.
5. To focus on holistic & sustainable development.

Department Vision, Mission,PEOs and PSOs Statements

Vision

To bring out talented, skilled, and sustainable Electronics and Communication Engineering Graduates through strong domain expertise to serve the Society with greater Professional Ethics.

Mission

1. To create and impart an active learning ambience to accomplish a high degree of Professional competencies
2. To inculcate innovative research and developmental thinking in effective Teaching and Learning processes for solving Societal challenges
3. To deliver the needs and requirements of the latest state of art of the Industry through quality multidisciplinary internship and training programs

PEOs

PEO 1:	To be successful in professional career in electronics, communication and allied industries by acquiring the knowledge in the fundamentals of Electronics and Communication Engineering principles and professional skills.
PEO 2:	To be in a position to analyze real life problems and design socially accepted and economically feasible solutions in the respective fields.
PEO 3:	To exhibit good communication skills in their professional career, lead a team with good leadership traits and good interpersonal relationship with the members related to other engineering streams.
PEO 4:	To involve themselves in lifelong learning and professional development by pursuing higher education and participation in research and development activities.
PEO 5:	To demonstrate professional and ethical responsibilities towards their profession, society and the environment.

PSOs

PSO 1:	An ability to use appropriate modern techniques for analysis, design and development of VLSI and Embedded Systems.
PSO 2:	Understand the architectural specifications of a communication system and determine their performance.

KLS Vishwanathrao Deshpande Institute of Technology
(Accredited by NAAC with "A" Grade)
(Approved by AICTE, New Delhi, Affiliated to VTU, Belagavi)
(Recognized Under Section 2(f) by UGC, New Delhi)
Udyog Vidya Nagar, Haliyal - 581 329, Dist.: Uttara Kannada
Phone: 08284 - 220861, 220334, 221409, Fax: 08284 - 220813
www.klsvdit.edu.in | principal@klsvdit.edu.in | hodece@klsvdit.edu.in
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Course Title	:	C++ Basics Laboratory
Course Code	:	BEC358C
Year / Semester	:	2nd Year / 3rd Semester
Academic Year	:	2025-26

Syllabus :

			members & members functions: i) Data members: employee number (an integer), Employee_ Name (a string of characters), Basic_ Salary (in integer), All_ Allowances (an integer), Net Salary (an integer). (ii) Member functions: To read the data of an employee, to calculate Net Salary & to print the values of all the data members. (All Allowances = 123% of Basic, Income Tax (IT) =30% of gross salary (=basic_ Salary_All_Allowances_IT).	
		10	Write a C++ program with different class related through multiple inheritance & demonstrate the use of different access specified by means of members variables & members functions.	26
		11	Write a C++ program to create three objects for a class named count object with data members such as roll_no & Name. Create a members function set_data () for setting the data values & display () member function to display which object has invoked it using this pointer.	28
		12	Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.	30
		13	Virtual Lab: Function	32
CLOs	:	• Understand object-oriented programming concepts and apply them in solving problems. • To create, debug and run simple C++ programs. • Introduce the concepts of functions, friend functions, inheritance, polymorphism and function overloading. • Introduce the concepts of exception handling and multithreading.		
COs	:	**CO1:**	Write C++ program to solve simple and complex problems.	
		CO2:	Apply and implement major object-oriented concepts like message passing, function overloading, operator overloading and inheritance to solve real-world problems.	
		CO3:	Use major C++ features such as Templates for data type independent designs and File I/O to deal with large data set.	
		CO4:	Analyze, design and develop solutions to real-world problems applying OOP concepts of C++.	

CO / PO Mapping	:	**COs**	**CO1**	**CO2**	**CO2**	**CO4**
		POs / PSOs	1,2,3 / 1	1,2,3 / 1	1,2,3 / 1	1,2,3 / 1
Course In-Charge	:	**Prof. Vijayalaxmi C. Kalal**				

CO-PO Mapping Matrix

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
1	1	2	1										1	
2	2	1	2										1	
3	2	2	1										1	
4	2	1	1										1	

1. Evaluation:

Students Assessment through CIE (50%) + SEE (50%)

Parameter	Particulars	Marks	Total	Scale Down Marks
CIE	Performance	05	120 (10*12 Expt)	30
	Viva-Voce	02		
	Journal	03		
LAB IA-1	Write-up + Conduction + Result	60	100	20
	Viva	40		
LAB IA-2	Write-up + Conduction + Result	60	100	
	Viva	40		
Total Marks				**50**

Note:

1. CIE for each lab session will be for 10 Marks.
2. Total CIE marks scale down to 30 Marks.
3. Two lab Internals to be conducted for 100 Marks each.
4. Lab Internal marks to be scale down to 20 Marks.

To run a C++ program on your laptop, you will need a C++ compiler installed. Here are the general steps:

1. Install a C++ Compiler:

- On Windows, you can use a compiler like MinGW, Visual C++, or Code::Blocks.
- On macOS, you can use Xcode (make sure to install the Command Line Tools).
- On Linux, you can use GCC, which is usually pre-installed on most distributions.

2. Write Your C++ Code:

Use a text editor or an integrated development environment (IDE) to write your C++ code. Save it with a .cpp extension.

3. Open a Terminal or Command Prompt:

- On Windows, open the Command Prompt or PowerShell.
- On macOS, open the Terminal.
- On Linux, open a terminal emulator.

4. Navigate to the Directory with Your C++ File:

Use the cd command to navigate to the directory where you saved your C++ file. For example:
bashCopy code

cd /path/to/your/code

5. Compile Your C++ Code:

Use the appropriate compiler command to compile your code. For example, with GCC, you can compile a program named my_program.cpp like this:
Copy code

g++ my_program.cpp -o my_program
This will create an executable file named my_program in the same directory.

6. Run Your C++ Program:

To run your program, simply type the name of the executable (without the .exe extension on Windows) and press Enter:
bashCopy code

./my_program # On macOS and Linux, my_program.exe # On Windows

Replace my_program with the actual name of your compiled executable.

7. View Output:

Your program should execute, and you will see the output in the terminal. That's it!

8. Direct Execution of Code in Code::Blocks:

After point no. 2 directly save the code with the extension .cpp and click on debug then run the code. The output window will be displayed on the screen.

Experiment: 01
Write a C++ program to find largest, smallest & second largest of three numbers using inlinefunctions MAX & Min.

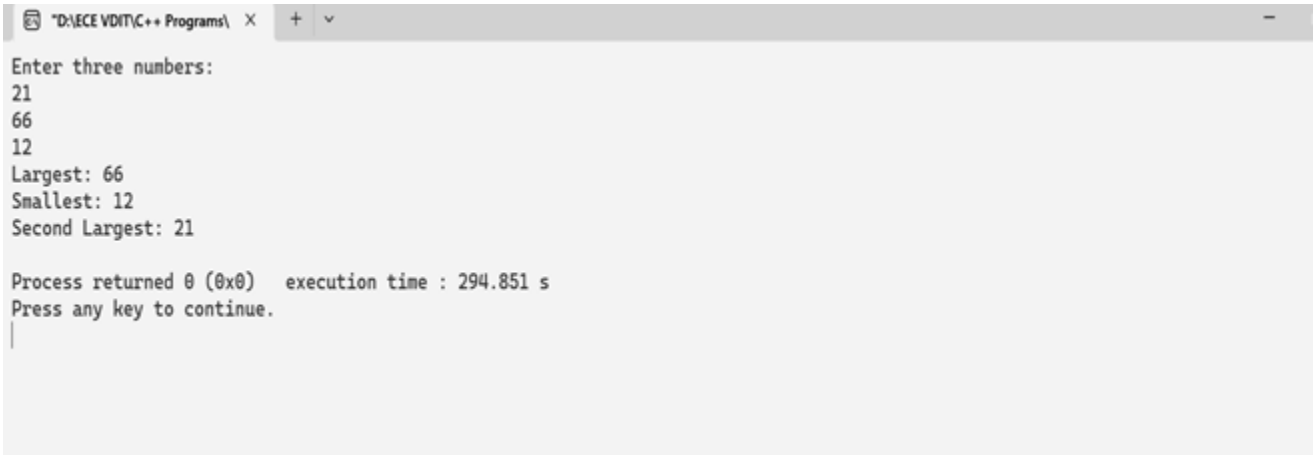
PROGRAM:

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

// Inline function to find the maximum of two numbers
inline int MAX(int a, int b) {
      return (a > b) ? a : b;
}

// Inline function to find the minimum of two numbers
inline int MIN(int a, int b) {
      return (a < b) ? a : b;
}

int main()
{
       int num1, num2, num3;

       // Input three numbers
       cout << "Enter three numbers: "<<endl;
       cin >> num1 >> num2 >> num3;
       int largest = MAX(MAX(num1, num2), num3);
       int smallest = MIN(MIN(num1, num2), num3);
       int secondLargest = num1 + num2 + num3 - largest - smallest;

       // Output the results
       cout << "Largest: " << largest << endl;
       cout << "Smallest: " << smallest << endl;
       cout << "Second Largest: " << secondLargest << endl;

      return 0;
}
```

THEORY: In this program, the MAX and MIN functions are defined as inline functions, which means that the function body is substituted directly at the call site instead of being invoked as a separate function. This can provide performance benefits for small functions like these.
The program prompts the user to enter three numbers and then uses the MAX and MIN functions to find the largest and smallest numbers, respectively. The second largest number is calculated by subtracting the largest and smallest numbers from the sum of all three numbers. Finally, the program outputs the results.

OUTPUT:

Enter three numbers:
21 66 12
Largest: 66
Smallest: 12
Second Largest: 21



Experiment: 02

Write a C++ program to calculate the volume of different geometric shapes like cube, cylinder and sphere using function overloading concept.

PROGRAM:

```
#include<iostream>
#include<cmath>
using namespace std;
const double PI = 3.14159;

// Function to calculate the volume of a cube
double volume(double side)
{
        return side * side * side;
}

// Function to calculate the volume of a cylinder
double volume(double radius, double height)
{
        return PI * radius * radius * height;
 }

// Function to calculate the volume of a sphere
double volume (int rad)
{
        return (4.0 / 3.0) * PI * pow(rad, 3);
 }

 int main()
 {
        int choice, rad;
        double side, radius, height;

        cout << "Select a shape to calculate the volume:\n";
        cout << "1. Cube\n";
        cout << "2. Cylinder\n";
        cout << "3. Sphere\n";
        cout << "Enter your choice (1-3): ";
        cin >> choice;

        switch (choice)
        {
                case 1:
                cout << "Enter the side length of the cube: ";
                cin >> side;
                cout << "Volume of the cube: " << volume(side) << endl;
```

```
            break;
            case 2:
            cout << "Enter the radius of the cylinder: ";
            cin >> radius;
            cout << "Enter the height of the cylinder: ";
            cin >> height;
            cout << "Volume of the cylinder: " << volume(radius, height) << endl;
            break;
            case 3:
            cout << "Enter the radius of the sphere: ";
            cin >> rad;
            cout << "Volume of the sphere: " << volume(rad) << endl;
            break;
            default:
            cout << "Invalid choice!" << endl;
        }
        return 0;
}
```

THEORY: In this program, three functions named volume are defined with different parameters. The function names are the same, but the parameters differ, which is known as function overloading. This allows us to use the same function name for different geometric shapes.
The program prompts the user to select a shape (cube, cylinder, or sphere) and then takes the necessary inputs (side length, radius, and height) accordingly. It then calls the appropriate overloaded volume function based on the user's choice and outputs the calculated volume.

OUTPUT:

Select a shape to calculate the volume:
1. Cube
2. Cylinder
3. Sphere

Enter your choice (1-3): 1
Enter the side length of the cube: 3
Volume of the cube: 27

Select a shape to calculate the volume:
1. Cube
2. Cylinder
3. Sphere

Enter your choice (1-3): 2
Enter the radius of the cylinder: 4
Enter the height of the cylinder: 6
Volume of the cylinder: 301.593

Select a shape to calculate the volume:
1. Cube
2. Cylinder
3. Sphere

Enter your choice (1-3): 3
Enter the radius of the sphere: 3
Volume of the sphere: 113.097

Select a shape to calculate the volume:
1. Cube

2. Cylinder
3. Sphere

Enter your choice (1-3): 4

Invalid choice!

Experiment: 03

Define a STUDENT class with USN, Name & Marks in 3 tests of a subject. Declare an array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name & the average marks of all the students.

PROGRAM:

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

const int NUM_STUDENTS = 10;
const int NUM_TESTS = 3;

class STUDENT {
  private:
        string usn;
        string name;
        int marks[NUM_TESTS];

  public:
        void setDetails(const string& usn, const string& name, const int marks[]) {
                this->usn = usn;
                this->name = name;
                for (int i = 0; i < NUM_TESTS; i++) {
                        this->marks[i] = marks[i];
                }
        }

        double calculateAverage() {
                int max1 = 0;
                int max2 = 0;

                for (int i = 0; i < NUM_TESTS; i++) {
                        if (marks[i] > marks[max1]) {
                                max2 = max1;
                                max1 = i;
                        }
                        else if (marks[i] > marks[max2]) {
                                max2 = i;
                        }
                }
               return (marks[max1] + marks[max2]) / 2.0;
        }

        void displayDetails() {
```

```
            cout << "USN: " << usn <<endl;
            cout << "Name: " << name << endl;
            cout << "Average Marks: " << calculateAverage() << endl;
            cout << endl;
        }
};
int main() {
        STUDENT students[NUM_STUDENTS];
        string usn, name;
        int marks[NUM_TESTS];

        for (int i = 0; i < NUM_STUDENTS; i++) {
            cout << "Enter details for Student " << i + 1 << ":" << endl;
            cout << "USN: ";
            cin >> usn;
            cout << "Name: ";
            cin >> name;
            cout << "Marks for 3 tests: ";
            for (int j = 0; j < NUM_TESTS; j++) {
                cin >> marks[j];
            }
            students[i].setDetails(usn, name, marks);
            cout << endl;
        }

        cout << "Details of all students:" << endl;
        for (int i = 0; i < NUM_STUDENTS; i++) {
            students[i].displayDetails();
        }

        return 0;
}
```

THEORY: In this program, the STUDENT class has private member variables usn, name, and an array marks to store the marks for each student. The class provides public member functions to set the details, calculate the average of the two highest marks, and display the details of a student.
The program initializes an array of 10 STUDENT objects. It then prompts the user to enter the details for each student, including the USN, name, and marks for 3 tests. The program calls the setDetails function to set the details for each student. Finally, it displays the details (USN, name, and average marks) of all the students using the displayDetails function.

OUTPUT:

```
Enter details for Student 1:
USN: 1BI21ET001
Name: GIRI
Marks for 3 tests: 11
12
13

Enter details for Student 2:
USN: 1BI21ET002
Name: GURU
Marks for 3 tests: 23
```

```
43
45

Enter details for Student 3:
USN: 1BI21ET003
Name: GANGA
Marks for 3 tests: 23
23
23

Enter details for Student 4:
USN: 1BI21ET004
Name: THUNGA
Marks for 3 tests: 12
13
14

Enter details for Student 5:
USN: 1BI21ET005
Name: RANGA
Marks for 3 tests: 24
24
23

Enter details for Student 6:
USN: 1BI21ET006
Name: LAKSHMI
Marks for 3 tests: 21
23
24

Enter details for Student 7:
USN: 1BI21ET007
Name: RASHMI
Marks for 3 tests: 21
23
24

Enter details for Student 8:
USN: 1BI21ET008
Name: MOHAMMAD
Marks for 3 tests: 12
23
44

Enter details for Student 9:
USN: 1BI21ET009
Name: JAMES
Marks for 3 tests: 23
24
22

Enter details for Student 10:
USN: 1BI21ET010
```

Name: ROJA
Marks for 3 tests: 11
21
23

Details of all students:
USN: 1BI21ET001
Name: GIRI
Average Marks: 12.5

USN: 1BI21ET002
Name: GURU
Average Marks: 44

USN: 1BI21ET003
Name: GANGA
Average Marks: 23

USN: 1BI21ET004
Name: THUNGA
Average Marks: 13.5

USN: 1BI21ET005
Name: RANGA
Average Marks: 24

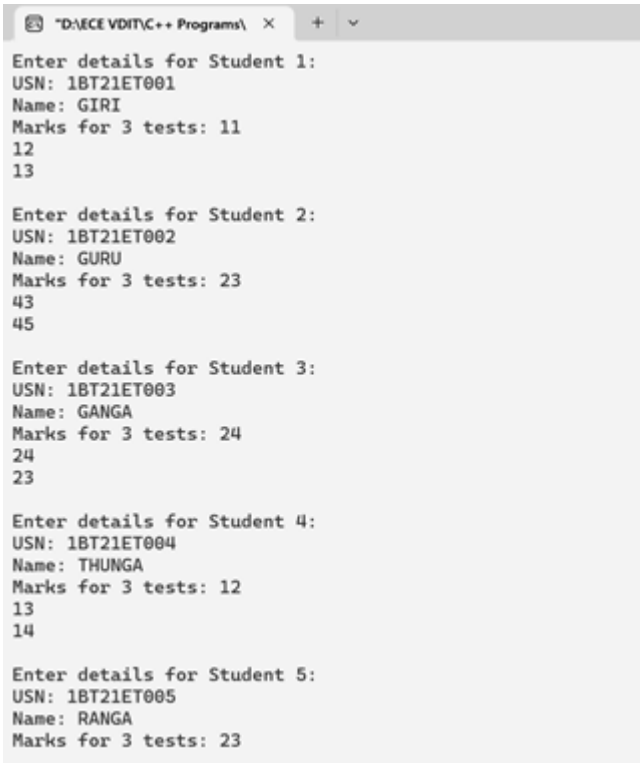
USN: 1BI21ET006
Name: LAKSHMI
Average Marks: 23.5

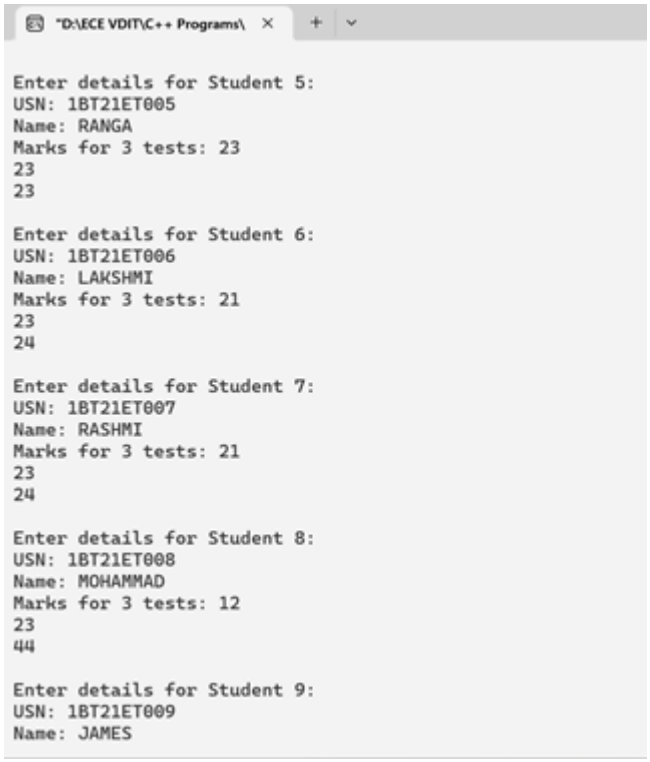
USN: 1BI21ET007
Name: RASHMI
Average Marks: 23.5

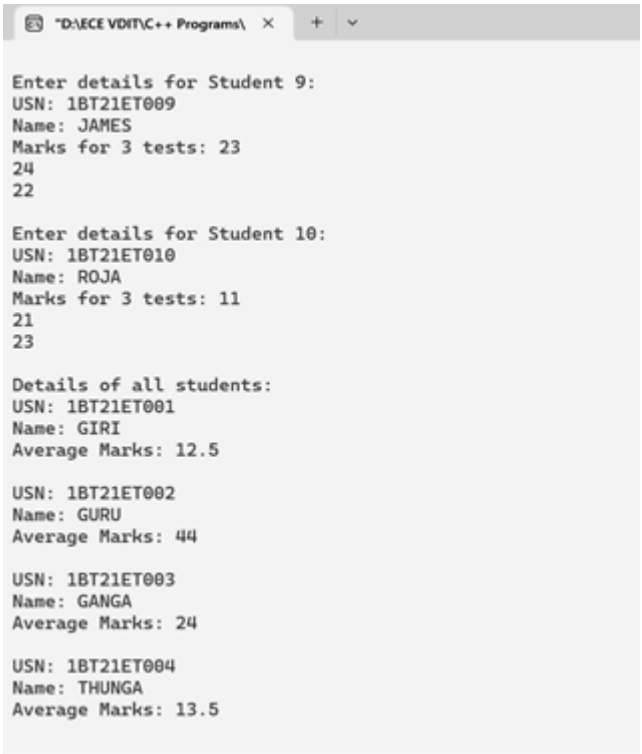
USN: 1BI21ET008
Name: MOHAMMAD
Average Marks: 33.5

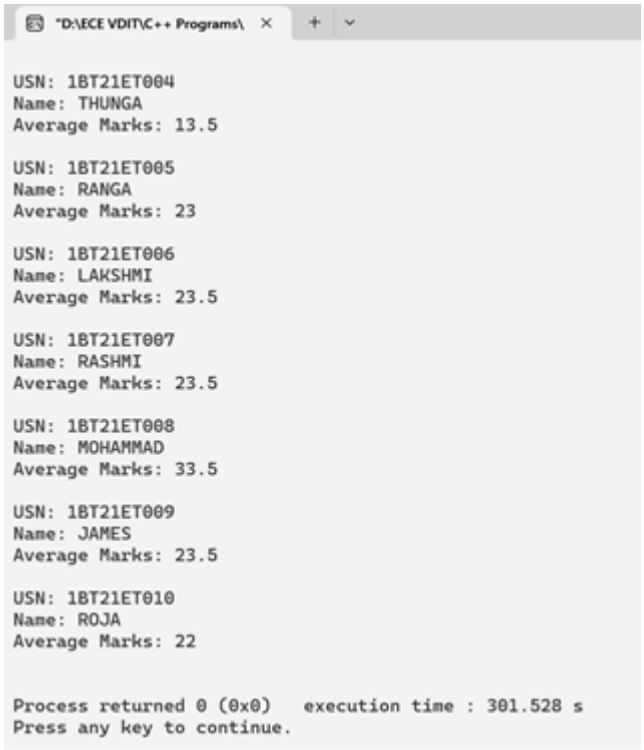
USN: 1BI21ET009
Name: JAMES
Average Marks: 23.5

USN: 1BI21ET010
Name: ROJA
Average Marks: 22


"D:\ECE VDIT\C++ Programs\
Enter details for Student 1:
USN: 1BT21ET001
Name: GIRI
Marks for 3 tests: 11
12
13
Enter details for Student 2:
USN: 1BT21ET002
Name: GURU
Marks for 3 tests: 23
43
45
Enter details for Student 3:
USN: 1BT21ET003
Name: GANGA
Marks for 3 tests: 24
24
23
Enter details for Student 4:
USN: 1BT21ET004
Name: THUNGA
Marks for 3 tests: 12
13
14
Enter details for Student 5:
USN: 1BT21ET005
Name: RANGA
Marks for 3 tests: 23


"D:\ECE VDIT\C++ Programs\
Enter details for Student 5:
USN: 1BT21ET005
Name: RANGA
Marks for 3 tests: 23
23
23
Enter details for Student 6:
USN: 1BT21ET006
Name: LAKSHMI
Marks for 3 tests: 21
23
24
Enter details for Student 7:
USN: 1BT21ET007
Name: RASHMI
Marks for 3 tests: 21
23
24
Enter details for Student 8:
USN: 1BT21ET008
Name: MOHAMMAD
Marks for 3 tests: 12
23
44
Enter details for Student 9:
USN: 1BT21ET009
Name: JAMES


"D:\ECE VDIT\C++ Programs\
Enter details for Student 9:
USN: 1BT21ET009
Name: JAMES
Marks for 3 tests: 23
24
22
Enter details for Student 10:
USN: 1BT21ET010
Name: ROJA
Marks for 3 tests: 11
21
23
Details of all students:
USN: 1BT21ET001
Name: GIRI
Average Marks: 12.5
USN: 1BT21ET002
Name: GURU
Average Marks: 44
USN: 1BT21ET003
Name: GANGA
Average Marks: 24
USN: 1BT21ET004
Name: THUNGA
Average Marks: 13.5


"D:\ECE VDIT\C++ Programs\
USN: 1BT21ET004
Name: THUNGA
Average Marks: 13.5
USN: 1BT21ET005
Name: RANGA
Average Marks: 23
USN: 1BT21ET006
Name: LAKSHMI
Average Marks: 23.5
USN: 1BT21ET007
Name: RASHMI
Average Marks: 23.5
USN: 1BT21ET008
Name: MOHAMMAD
Average Marks: 33.5
USN: 1BT21ET009
Name: JAMES
Average Marks: 23.5
USN: 1BT21ET010
Name: ROJA
Average Marks: 22
Process returned 0 (0x0) execution time : 301.528 s
Press any key to continue.

Experiment: 04

Write a C++ program to create a class called MATRIX using a two-dimensional array of integers, by overloading the operator == which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading the + and – operators respectively. Display the results by overloading the operator <<. If (ml == m2) then m3 = ml + m2 and m4 = ml – m2 else display error.

PROGRAM:

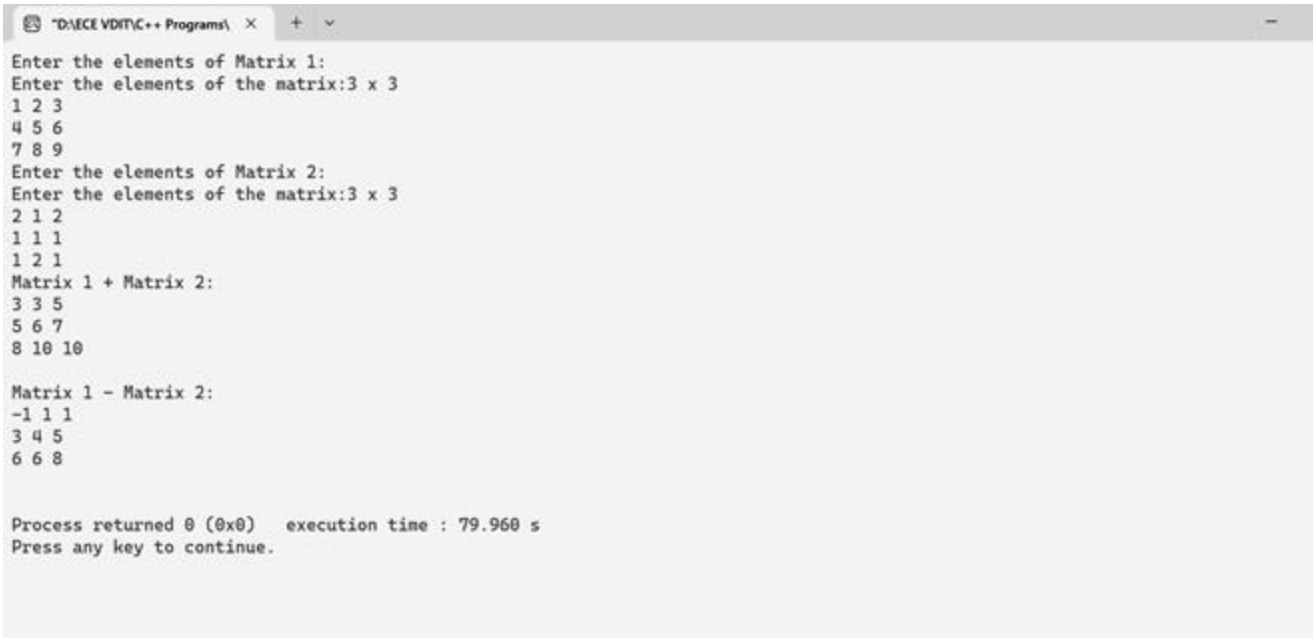
```
#include <iostream>
using namespace std;

const int MAX_ROWS = 3;
const int MAX_COLS = 3;

class MATRIX {
private:
        int mat[MAX_ROWS][MAX_COLS];
        int rows;
        int cols;

public:
        MATRIX(int rows, int cols) {
        this->rows = rows;
        this->cols = cols;
        }

        void inputMatrix() {
                cout << "Enter the elements of the matrix:" <<MAX_ROWS<<" x " <<MAX_COLS
                << endl;
                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                                cin >> mat[i][j];
                        }
                }
        }

        bool operator==(const MATRIX& other) {
                return (rows == other.rows && cols == other.cols);
        }

        MATRIX operator+(const MATRIX& other) {
                MATRIX result(rows, cols);
                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                                result.mat[i][j] = mat[i][j] + other.mat[i][j];
```

```
                }
            }
            return result;
        }
        MATRIX operator-(const MATRIX& other) {
            MATRIX result(rows, cols);
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    result.mat[i][j] = mat[i][j] - other.mat[i][j];
                }
            }
            return result;
        }

        friend ostream& operator<<(ostream& os, const MATRIX& matrix);
};

ostream& operator<<(ostream& os, const MATRIX& matrix) {
        for (int i = 0; i < matrix.rows; i++) {
            for (int j = 0; j < matrix.cols; j++){
                os << matrix.mat[i][j] << " ";
            }
            os << endl;
        }
       return os;
}

int main() {
        MATRIX m1(MAX_ROWS, MAX_COLS);
        MATRIX m2(MAX_ROWS, MAX_COLS);

        cout << "Enter the elements of Matrix 1:" << endl;
        m1.inputMatrix();

        cout << "Enter the elements of Matrix 2:" << endl;
        m2.inputMatrix();

        if (m1 == m2) {
            MATRIX m3 = m1 + m2;
            MATRIX m4 = m1 - m2;

            cout << "Matrix 1 + Matrix 2:" << endl;
            cout << m3 << endl;

            cout << "Matrix 1 - Matrix 2:" << endl;
            cout << m4 << endl;
        }
        else {
            cout << "Error: Matrices are incompatible for addition and subtraction." <<endl;
        }

        return 0;
}
```

THEORY: This program creates two MATRIX objects, m1 and m2, with the specified number of rows and columns. The user is prompted to enter the elements of each matrix using the inputMatrix() member function.
The program then checks if the matrices are compatible for addition and subtraction by overloading the == operator. If they are compatible, it performs the addition and subtraction operations using the overloaded + and - operators, respectively.
The results are displayed using the overloaded << operator, which prints the matrix elements. If the matrices are not compatible, an error message is displayed.

OUTPUT:

Enter the elements of Matrix 1:
Enter the elements of the matrix: 3 x 3
1 2 3
4 5 6
7 8 9
Enter the elements of Matrix 2:
Enter the elements of the matrix: 3 x 3
2 1 2
1 1 1
1 2 1
Matrix 1 + Matrix 2:
3 3 5
5 6 7
8 10 10
Matrix 1 - Matrix 2:
-1 1 1
3 4 5
6 6 8



Experiment: 05

Demonstrate simple inheritance concept by creating a base class FATHER with data members: First Name, Surname, DOB & bank Balance and creating a derived class SON, which inherits: Surname & Bank Balance feature from base class but provides its own feature: First Name & DOB. Create & initialize Fl & Sl objects with appropriate constructors & display the FATHER & SON details.

PROGRAM:

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

class FATHER {
protected:
        string surname;
        double bankBalance;

public:
        FATHER(const string& surname, double bankBalance) : surname(surname),
bankBalance(bankBalance) {}

        void displayFatherDetails()
        {
                cout << "Father's Surname: " << surname << endl;
                cout << "Father's Bank Balance: $" << bankBalance << endl;
        }
};

class SON : public FATHER{
private:
        string firstName;
        string dob;

public:
        SON(const string& firstName, const string& dob, const string& surname, double
        bankBalance) : FATHER(surname, bankBalance), firstName(firstName), dob(dob) {}

        void displaySonDetails()
        {
                cout << "Son's First Name: " << firstName << endl;
                cout << "Son's Date of Birth: " << dob << endl;
        }
};
```

```
int main()
{
	string fatherSurname, sonSurname, sonFirstName, sonDOB;
	double fatherBankBalance, sonBankBalance;

	cout << "Enter father's surname: ";
	cin >> fatherSurname;
	cout << "Enter father's bank balance: $";
	cin >> fatherBankBalance;

	cout << "Enter son's first name: ";
	cin >> sonFirstName;
	cout << "Enter son's date of birth: ";
	cin >> sonDOB;
	cout << "Enter son's surname: ";
	cin >> sonSurname;
	cout << "Enter son's bank balance: $";
	cin >> sonBankBalance;

	FATHER F1(fatherSurname, fatherBankBalance);
	SON S1(sonFirstName, sonDOB, sonSurname, sonBankBalance);

	cout << "Father Details:" << endl;
	F1.displayFatherDetails();
	cout << endl;

	cout << "Son Details:" << endl;
	S1.displaySonDetails();

	return 0;
}
```

THEORY: In this program, the FATHER class is defined with data members for Surname and Bank Balance. The SON class is derived from the FATHER class and adds data members for First Name and DOB. The program creates objects F1 and S1 of the FATHER and SON classes, respectively, using appropriate constructors.
The user is prompted to enter the details for the father (surname and bank balance) and the son (first name, DOB, surname, and bank balance). The program then creates the objects F1 and S1 with the provided details.
Finally, the program displays the details of the father and son by calling the respective member functions (displayFatherDetails() and displaySonDetails()).

OUTPUT:

```
Enter father's surname: raju
Enter father's bank balance: $2000
Enter son's first name: ganesh
Enter son's date of birth: 19-09-1982
Enter son's surname: raju
Enter son's bank balance: $250
Father Details:
Father's Surname: raju
Father's Bank Balance: $2000
```

Son Details:
Son's First Name: ganesh
Son's Date of Birth: 19-09-1982

Experiment: 06

Write a C++ program to define class name FATHER & SON that holds the income respectively. Calculate & display total income of a family using Friend function.

PROGRAM:

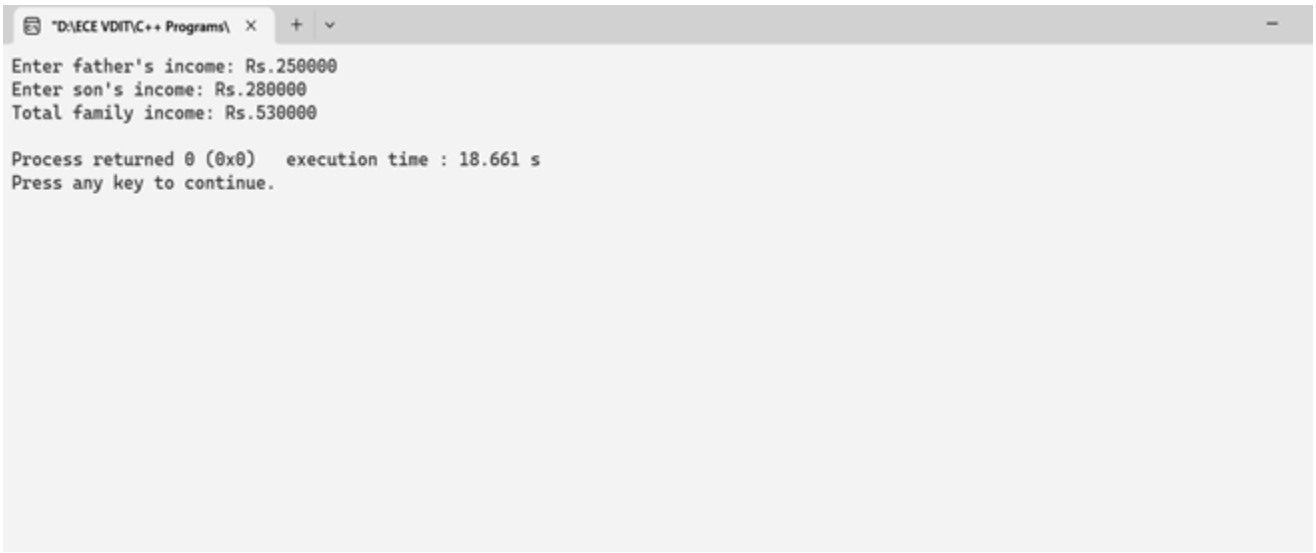
```
#include <iostream>
using namespace std;

class SON; // Forward declaration

class FATHER {
private:
        double income;

public:
        FATHER(double income) : income(income) {}

        friend double calculateTotalIncome(const FATHER& father, const SON& son);
};

class SON {
private:
        double income;

public:
        SON(double income) : income(income) {}

        friend double calculateTotalIncome(const FATHER& father, const SON& son);
};

double calculateTotalIncome(const FATHER& father, const SON& son) {
        return father.income + son.income;
}

int main() {
        double fatherIncome, sonIncome;

        cout << "Enter father's income: Rs.";
        cin >> fatherIncome;

        cout << "Enter son's income: Rs.";
        cin >> sonIncome;

        FATHER father(fatherIncome);
        SON son(sonIncome);
```

```
    double totalIncome = calculateTotalIncome(father, son);

    cout << "Total family income: Rs." << totalIncome << endl;
    return 0;
}
```

THEORY: In this program, the FATHER class is defined with a private member variable income. The SON class is also defined with a private member variable income. The friend function calculateTotalIncome is declared in both classes.
The calculateTotalIncome function takes references to the FATHER and SON objects as arguments and calculates the total income by adding their respective incomes. It has access to the private members of both classes since it is declared as a friend function in both classes.
In the main function, the user is prompted to enter the incomes of the father and son. Objects of the FATHER and SON classes are then created with the provided incomes. The calculateTotalIncome function is called with these objects to calculate the total family income. Finally, the total income is displayed to the user.*/

OUTPUT:

Enter father's income: Rs.250000
Enter son's income: Rs.280000
Total family income: Rs.530000



Experiment: 07
Write a C++ program to accept the student detail such as name & 3 different marks by get_data() method & display the name & average of marks using display() method. Define a friend function for calculating the average marks using the method mark_avg().

PROGRAM:

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

class Marks {
private:
        int mark1;
        int mark2;
        int mark3;

public:
        Marks(int mark1, int mark2, int mark3) : mark1(mark1), mark2(mark2), mark3(mark3)
{}

        friend double mark_avg(const Marks& marks);
};

class Student {
private:
        string name;
        Marks marks;

public:
        Student(const string& name, int mark1, int mark2, int mark3): name(name), marks(mark1,
        mark2, mark3) {}

        void display() {
                cout << "Student Name: " << name << endl;
                cout << "Average Marks: " << mark_avg(marks) << endl;
        }

        friend double mark_avg(const Marks& marks);
};

double mark_avg(const Marks& marks) {
        double average = (marks.mark1 + marks.mark2 + marks.mark3) / 3.0;
        return average;
}
int main() {
```

```
    string name;
    int mark1, mark2, mark3;

    cout << "Enter student name: ";
    getline(cin >> ws, name);


    cout << "Enter marks for three subjects: ";
    cin >> mark1 >> mark2 >> mark3;

    Student student(name, mark1, mark2, mark3);
    student.display();

    return 0;
}
```

THEORY: In this program, there are two classes: Marks and Student. The Marks class holds the three different marks, and the Student class holds the student's name and an object of the Marks class. The Marks class has private member variables for three marks and a constructor to initialize them.
The Student class has private member variables for the student's name and an object of the Marks class. It also has public member functions get_data() to accept the student's name and marks and display() to display the name and average marks.
The friend function mark_avg() is defined in both classes. It takes a Student object as an argument and calculates the average marks using the three marks stored in the Marks object of the Student object.
In the main() function, the user is prompted to enter the student's name and marks for three subjects. The get_data() function is called to set the student's details. Finally, the display() function is called to display the student's name and average marks using the mark_avg() friend function.

OUTPUT:

```
Enter student name: Harish
Enter marks for three subjects: 28
25
63
Student Name: Harish
Average Marks: 38.6667
```

Experiment: 08
Write a C++ program to explain virtual function (Polymorphism) by creating a base class polygon which has virtual function areas two classes rectangle & triangle derived from polygon & they have area to calculate & return the area of rectangle & triangle respectively.

PROGRAM:

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

class Polygon {
public:
        virtual double area() = 0; // Pure virtual function

        virtual ~Polygon() {} // Virtual destructor for polymorphic behavior
};

class Rectangle : public Polygon {
private:
        double length;
        double width;

public:
        Rectangle(double length, double width) : length(length), width(width) {}

        double area() override {
                return length * width;
        }
};

class Triangle : public Polygon {
private:
        double base;
        double height;

public:
        Triangle(double base, double height) : base(base), height(height) {}

        double area() override {
                return 0.5 * base * height;
        }
};

int main() {
        Polygon* shapes[2];
```

```
        Rectangle rectangle(5.0, 3.0);
        Triangle triangle(4.0, 6.0);

        shapes[0] = &rectangle;
        shapes[1] = &triangle;

        for (int i = 0; i < 2; i++) {
                cout << "Shape " << i + 1 << " Area: " << shapes[i]->area() << endl;
        }
        return 0;
}
```

THEORY: The Polygon class has a virtual area() function, making it a polymorphic base class. The Rectangle and Triangle classes inherit from Polygon and override the area() function with their own implementations.
In the main() function, an array of Polygon pointers shapes is created. Objects of Rectangle and Triangle are instantiated, and their addresses are stored in the shapes array. This allows polymorphic behavior, where the appropriate area() function is called based on the actual object type, even though the function is called through a base class pointer.
The program then iterates through the shapes array and calls the area() function for each shape, displaying the calculated areas.*/

OUTPUT:

Shape 1 Area: 15
Shape 2 Area: 12

Experiment: 09
Design, develop and execute a program in C++ based on the following requirements: An EMPLOYEE class containing data members & members functions: i) Data members: employee number (an integer), Employee_ Name (a string of characters), Basic_ Salary (in integer), All Allowances (an integer), Net Salary (an integer). (ii) Member functions: To read the data of an employee, to calculate Net_Salary& to print the values of all the data members. (All Allowances= l23% of Basic, Income Tax (IT) =3O% of gross salary (=basic_Salary_All_ Allowances_IT).

PROGRAM:

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

class EMPLOYEE {
private:
        int employeeNumber;
        string employeeName;
        int basicSalary;
        int allAllowances;
        int grossSalary;
        int incomeTax;
        int netSalary;

public:
        void readData() {
                cout << "Details of an Employee "<<endl;
                cout << "Enter employee number: ";
                cin >> employeeNumber;
                cout << "Enter employee name: ";
                cin.ignore(); // Ignore any previous newline character
                getline(cin, employeeName);
                cout << "Enter basic salary: ";
                cin >> basicSalary;
        }

        void calculateNetSalary() {
                allAllowances = static_cast<int>(1.23 * basicSalary); // 123% of Basic Salary
                grossSalary = basicSalary + allAllowances;
                incomeTax = static_cast<int>(0.3 * grossSalary); // 30% of Gross Salary
                netSalary = grossSalary - incomeTax;
        }

        void printData() {
                cout << endl;
```

```
                cout << "Salary Calculations of an Employee"<<endl;
                cout << "Employee Number: " << employeeNumber << endl;
                cout << "Employee Name: " << employeeName << endl;
                cout << "Basic Salary: " << basicSalary << endl;
                cout << "All Allowances: " << allAllowances << endl;
                cout << "gross salary: " << grossSalary << endl;
                cout << "Income Tax: " <<incomeTax << endl;
                 cout << "Net Salary: " << netSalary << endl;
        }
};

int main() {
        EMPLOYEE emp;

        emp.readData();
        emp.calculateNetSalary();
        emp.printData();

        return 0;
}
```

THEORY: In this program, the EMPLOYEE class is defined with private member variables employeeNumber, employeeName, basicSalary, allAllowances, and netSalary. The class also contains three member functions: readData() to read the employee data, calculateNetSalary() to calculate the net salary based on the given formulas, and printData() to display the values of all data members.
In the main() function, an object emp of the EMPLOYEE class is created. The readData() function is called to input the employee details. The calculateNetSalary() function is then called to calculate the net salary based on the given formulas. Finally, the printData() function is called to display all the employee data.

OUTPUT:

```
Details of an Employee
Enter employee number: 100
Enter employee name: Bhaskar
Enter basic salary: 23000
Salary Calculations of an Employee
Employee Number: 100
Employee Name: Bhaskar
Basic Salary: 23000
All Allowances: 28290
gross salary: 51290
Income Tax: 15387
Net Salary: 35903
```

Experiment: 10

Write a C++ program with different class related to multiple inheritance & demonstrate the use of different access specified by means of members variables & members functions.

PROGRAM:

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

class Base1 {
protected:
        int protectedData1;

public:
        Base1(int data) : protectedData1(data) {}

        void printBase1() {
                cout << "Base1::protectedData1 = " << protectedData1 << endl;
        }
};

class Base2 {
protected:
        int protectedData2;

public:
        Base2(int data) : protectedData2(data) {}

        void printBase2() {
                cout << "Base2::protectedData2 = " << protectedData2 << endl;
        }
};

class Derived : public Base1, protected Base2 {
private:
        int privateData;

public:
        Derived(int data1, int data2, int data3): Base1(data1), Base2(data2), privateData(data3) {}

        void printDerived() {
                printBase1();           // Accessing protected member of Base1
                Base2::printBase2();    // Accessing protected member of Base2
                cout << "Derived::privateData = " << privateData << endl;
        }
};
```

```
int main() {
        Derived derived(10, 20, 30);
        derived.printDerived();
        return 0;
}
```

THEORY: In this program, we have three classes: Base1, Base2, and Derived. The Derived class is derived from both Base1 and Base2, demonstrating multiple inheritance.
Base1 has a protected member variable protectedData1 and a member function printBase1() to print its value.
Base2 has a protected member variable protectedData2 and a member function printBase2() to print its value.
Derived inherits both Base1 and Base2. It also has a private member variable privateData of its own. The printDerived() function is defined in Derived to access and print the values of the member variables from all the classes.
In the main() function, an object derived of the Derived class is created. The printDerived() function is then called to demonstrate the use of different access specifiers and access the member variables and member functions of the respective classes.
Note that protected members are accessible within the class itself and by derived classes, while private members are only accessible within the class itself.

OUTPUT:

Base1::protectedData1 = 10
Base2::protectedData2 = 20
Derived::privateData = 30

Experiment: 11
Write a C++ program to create three objects for a class named count object with data members such as roll_no & Name. Create a members function set_data() for setting the data values & display ( ) member function to display which object has invoked it using this pointer.

PROGRAM:

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

class CountObject {
private:
        int roll_no; string name;

public:
        void set_data(int rollNo, const string& studentName) {
                roll_no = rollNo;
                name = studentName;
        }

        void display() {
                cout << "Object with roll_no " << roll_no << " and name " << name << " invoked
                display()" << endl;
                cout << "This pointer value: " << this << endl;
        }
};

int main() {
        CountObject obj1, obj2, obj3;

        obj1.set_data(1, "Alice");
        obj2.set_data(2, "Bob");
        obj3.set_data(3, "Charlie");

        obj1.display();
        obj2.display();
        obj3.display();

        return 0;
}
```

THEORY: In this program, the CountObject class is defined with private member variables roll_no and name. It has member functions set_data() to set the data values and display() to display which object invoked it using the this pointer.
In the main() function, three objects obj1, obj2, and obj3 of the CountObject class are

created. The set_data() function is called on each object to set the roll number and name. Then, the display() function is called on each object to demonstrate the use of the this pointer and display which object invoked it.
The program outputs the roll number, name, and the memory address of the object using the this pointer.

OUTPUT:

Object with roll_no 1 and name Alice invoked display()
This pointer value: 0x6ec9dffb80
Object with roll_no 2 and name Bob invoked display()
This pointer value: 0x6ec9dffb50
Object with roll_no 3 and name Charlie invoked display()
This pointer value: 0x6ec9dffb20

Experiment: 12
Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.

PROGRAM:

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

class CustomException1 : public exception {
public:
        const char* what() const noexcept override {
                return "Custom Exception 1";
        }
};

class CustomException2 : public exception {
public:
        const char* what() const noexcept override {
                return "Custom Exception 2";
        }
};

class CustomException3 : public exception {
public:
        const char* what() const noexcept override {
                return "Custom Exception 3";
        }
};

int main() {
        try {
                int choice;
                cout << "Enter an option between 1 and 5: ";
                cin >> choice;

                switch (choice) {
                case 1:
                        throw CustomException1();
                case 2:
                        throw CustomException2();
                case 3:
                        throw CustomException3();
                case 4:
```

```
                throw logic_error("Logic Error Exception");
// Built-in exception class
            case 5:
                throw out_of_range("Out of Range Exception");
// Built-in exception class
            default:
                throw runtime_error("Unknown Exception"); // Built-in exception class
            }
        } catch (const CustomException1& e) {
            cout << "Caught Custom Exception 1: " << e.what() << endl;
        } catch (const CustomException2& e) {
            cout << "Caught Custom Exception 2: " << e.what() << endl;
        } catch (const CustomException3& e) {
            cout << "Caught Custom Exception 3: " << e.what() << endl;
        } catch (const exception& e) {
            cout << "Caught Exception: " << e.what() << endl;
        }

        return 0;
}
```

THEORY: In this program, we have five exception classes: CustomException1, CustomException2, and CustomException3, which are custom exception classes derived from exception, and logic_error and out_of_range, which are built-in exception classes.

The program prompts the user to enter an option between 1 and 5. Depending on the user's choice, it throws different exceptions:

If the choice is 1, it throws a CustomException1. If the choice is 2, it throws a CustomException2. If the choice is 3, it throws a CustomException3.

If the choice is 4, it throws a logic_error exception with a custom message.

If the choice is 5, it throws a out_of_range exception with a custom message.

If the choice is any other number, it throws a runtime_error exception with a custom message.

The catch block catches the exceptions in the following order: first, the custom exceptions (CustomException1, CustomException2, and CustomException3), and then the catch block for the base class exception. It prints the appropriate error message based on the caught exception.

Feel free to modify the program as per your requirements or add more custom exception classes to showcase different exceptional cases.

OUTPUT:

Enter an option between 1 and 5: 3
Caught Custom Exception 3: Custom Exception 3

Experiment: 13
Virtual Lab Experiment: To understand the functions of programming.

AIM: To understand the functions of programming.

THEORY:

Function basically is an independent piece of code which takes some variables as input and returns a result. The function may optionally update the values of the input variables. Writing a function involves clearly specifying the characteristics of the funciton in its prototype. The prototype of a function looks like:

```
return_type Function_name(datatypes_of_input_variables);
```

For example, the prototype of a function for computing tax and returning the total payable amount may look like:

```
float compute_total(float ,float );
```

This states that the name of the function is compute_total. It accepts two float variables as input and returns a float value as output. Next, we define the function by writing the code corresponding to the computation of the return value of the function.

```
float compute_total(float subtotal,float tax_rate)
        {
          float total;
          total=subtotal*(1+(tax_rate/100));
          return total;
        }
```

The input variables in this function are named subtotal and tax_rate and both of them are of float datatype. The value which this function return will be type casted into float before returning. This function can be called from the main function as:

```
total=compute_total(1000,8);
```

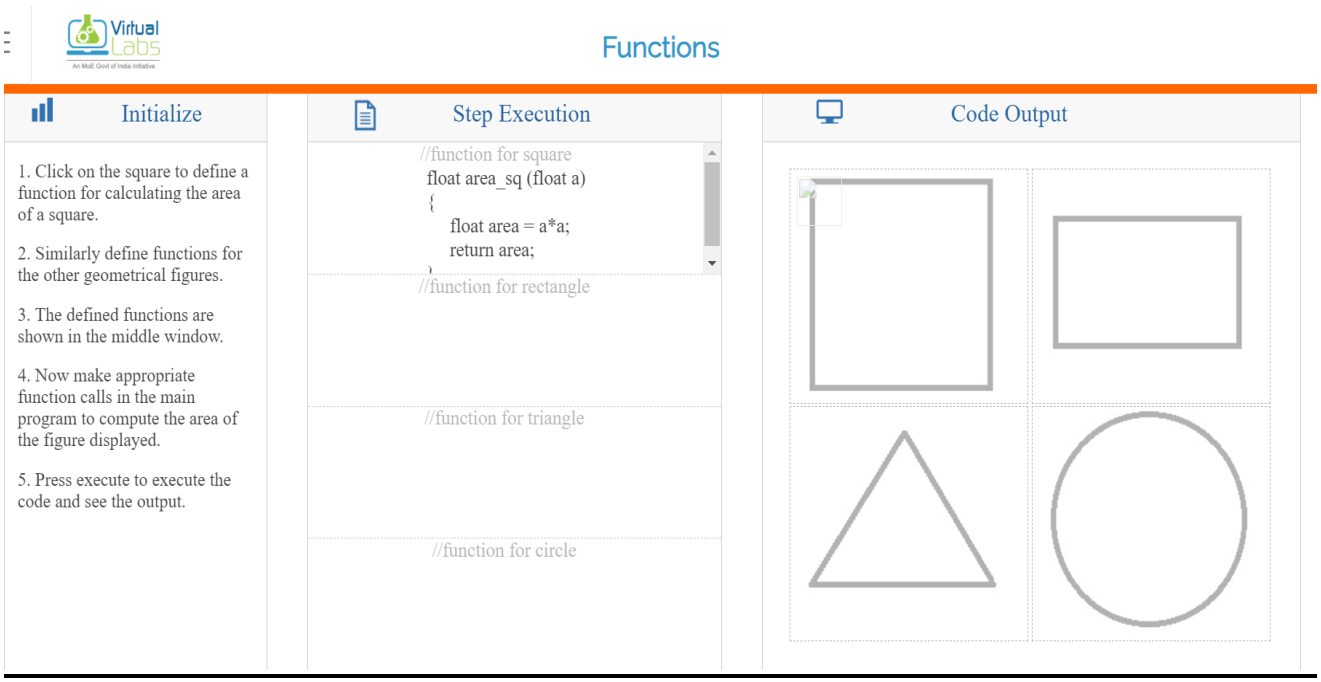
A complete code with functions may look like this:

```
function_prototypes;
      main()
      {
      function_calls;
      }
      function definitions;
```

PROCEDURE:

1. Click on the square to define a function for calculating the area of a square.
2. Similarly define functions for the other geometrical figures.
3. The defined functions are shown in the middle window.
4. Now do appropriate function calls in the main program to compute the area of the figure displayed.
5. Press execute to execute the code and see the output.

SIMULATION:



RESULT:

Link: https://cse02-iiith.vlabs.ac.in/exp/functions/