

K.L.E.Society's

K.L.E. COLLEGE OF ENGINEERING & TECHNOLOGY, CHIKODI - 591 201.

DEPT.OF ELECTRONICS & COMMUNICATION ENGG.

Academic Year 2023-24



Semester: III

C++ BASICS (BEC358C)

LAB IN CHARGE

Prof. Prashant Hebbale M.Tech.

LAB INSTRUCTOR

Mr. Sunil Akkole Diploma

Lab In-Charge

HOD

Expt. No.	Title of Experiments	Page No.
1	Write a C++ program to find largest, smallest & second largest of three numbers using inline functions MAX & Min.	3-4
2	Write a C++ program to calculate the volume of different geometric shapes like cube, cylinder and sphere using function overloading concept.	5-7
3	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.	8-10
4	Write a C++ program to create class called MATRIX using 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 + and - operators respectively. Display the results by overloading the operator <<. If (m1 == m2) then m3 = m1 + m2 and m4 = m1 - m2 else display error	11-13
5	Demonstrate simple inheritance concept by creating a base class FATHER with data members: <i>First Name, Surname, DOB & bank Balance</i> 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 F1 & S1 objects with appropriate constructors & display the FATHER & SON details.	14-15
6	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.	16-17
7	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().	18-19
8	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.	20-21
9	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 = 123% of Basic, Income Tax (IT) =30% of gross salary (=basic_ Salary_All_Allowances_IT).	22-23
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.	24-25
11	Write a C++ program to create three objects for a class named count object with data members 1133..0099..22002233 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.	26-27
12	Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.	28-29

How to Download Turbo C++ and Install

There are various compilers with advanced tools and features for C++. In this blog, we will discuss and learn how to download and use Turbo, a popular integrated development environment (IDE), which is a compiler for C and C++ programming languages.

Turbo compilers offer various advanced features in one place, such as an editor, syntax highlighting, auto code completion, debugging tools, etc. Check out the article to learn how to download Turbo C++ on your system and use it.

Table of Contents

What is Turbo C++?

The Turbo C++ compiler was developed by Borland in 1990. This compiler is known for its fast execution speed and quick compilation. Turbo C++ was a popular integrated development environment for C and C++ in the late 1980s and early 1990s. It includes various advanced tools that offer fast execution and better development environments for developers. You can also trace your source code using the tracing property in Turbo C++. It is very easy to download in any environment. It is accessible on Google and will support the current version of Windows.

Advanced Features of Turbo C++

Some of the advanced features of Turbo C++ are given below.

- It provides a full developer package. Developers can write, compile and debug their code within the same interface.
- Turbo C++ offers very fast compilation, which helps developers execute and compile their programs effectively.
- There are debugging tools and syntax highlighting available in Turbo C++.
- It helps to enhance scalability.
- It offers a DOS based environment as it was developed in the 1990s. However, various modifications occurred on the interface and now

Download Turbo C++

Turbo C++ is an open source compiler available for free online. You can download it from various websites.

Step 1: Search for Turbo C++ on Google; you can download Turbo C++ from various websites for free. It is an open source compiler.

Step 2: The download will start, and after downloading, open the zip folder. You can find the zip folder in your download file in the file manager.

Step 3: Click on the setup.exe file inside the zip folder. It will start installing Turbo C++ on your system.

Step 4: Now, the setup configuration window will open and the window installer will prepare for installation. It will verify that your system is compatible for the compiler.

Step 5: Click on the **Next** button to start the Turbo C++ installation. Do not close the window before installation completes.

Step 6: Accept the license agreement by checking the checkbox in the dialog box.

Step 7: Installation for Turbo C++ will start.

Step 8: Click on the Finish button once the installation gets completed.

--

Step 9: A new window of Turbo C++ will appear on your screen. You can start a new project.

Steps for Program Execution:

1. **Editing**
2. **Compiling**
3. **Linking Library files**
4. **Loading**
5. **Execution**

Editing:

Editing refers the typing or writing the program in any text editor. But we want all the things in one place like writing the program, compiling, and executing it. This is achieved with the help of software that is known as IDE (Integrated Development Environment). IDE integrated all the tasks that are required to run a program.

Examples of IDEs: Turbo C++, Devcpp, Xcode, Visual Studio Code, CodeBlocks, Eclipse, etc.

Compiling:

Consider a program **first.cpp** which is saved in Hard Disc. To compile the first.cpp file, we need an IDE that contains a compiler. The compiler converts the high-level code into machine-level language code and a new executable file with the name first.exe is generated and get stored in the hard disc. If the compiler finds any error in the code, it throws the error to the programmer else the code is successfully compiled.

Example: When first.cpp is compiled, the executable files are generated like max.exe and main.exe and get stored in the hard disc to get executed later.

Linking Libraries:

Every language has some built-in objects and functions that can be reused in any program. The built-in objects and functions are grouped inside libraries that can be included in programs as header files. These libraries and header files are linked with the code during compilation where the library code is also converted to an executable file along with the entire program.

Example: We included iostream which is a header file for cout and cin objects. The iostream is attached to the code during compilation where the header file code is also converted to executable code with .exe extension. This is called the linking of the library.

Loading:

To execute the program code, the code must be brought to the main memory from the secondary memory.

Execution:

As soon as the program gets loaded in the main memory in different sections as given below, the program execution starts. The execution of the program starts from the first line of the main function.

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

PROGRAM:

```
#include<iostream.h>

// 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

Process returned 0 (0x0)   execution time : 294.851 s
Press any key to continue.
|
```

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

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

PROGRAM:

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

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
```

```
Process returned 0 (0x0)   execution time : 16.948 s
Press any key to continue.
|
```

```
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
```

```
Process returned 0 (0x0)   execution time : 42.401 s
Press any key to continue.
|
```

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!

3. 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.h>
#include <string.h>

class STUDENT {
private:
    char USN[15];
    char Name[30];
    float Marks[3];

public:
    void setData() {
        cout << "Enter USN: ";
        cin >> USN;
        cout << "Enter Name: ";
        cin.ignore(); // Ignore newline character
        cin.getline(Name, 30);
        cout << "Enter Marks in 3 tests: ";
        for (int i = 0; i < 3; i++) {
                                cin >> Marks[i];
                                }
        }

    float calculateAverage() {
        // Sort the marks in descending order
        for (int i = 0; i < 2; ++i) {
            for (int j = i + 1; j < 3; j++) {
                if (Marks[i] < Marks[j]) {
                    // Swap
                    float temp = Marks[i];
                    Marks[i] = Marks[j];
                    Marks[j] = temp;
                }
            }
        }
        // Calculate average of the two better marks
        return (Marks[0] + Marks[1]) / 2.0;
    }

    void displayData() {
        cout << "USN: " << USN << "\nName: " << Name << "\nAverage Marks: " <<
        calculateAverage() << "\n\n";
    }
};
```

```
int main() {
    STUDENT students[10];
    int i;
    // Input data for each student
    for (i=0; i < 10; i++) {
        cout << "Enter details for Student " << i + 1 << ":\n";
        students[i].setData();
    }

    // Display USN, Name, and Average Marks for all students
    cout << "\nDetails of all students:\n";
    for (i = 0; i < 10; i++) {
        students[i].displayData();
    }

    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 setData 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 USN: 2KD22EC009
Enter Name: ANISHA
Enter Marks in 3 tests: 13
17
15
Enter details for Student 10:
Enter USN: 2KD22EC010
Enter Name: ANITA
Enter Marks in 3 tests: 13
14
15

Details of all students:
USN: 2KD22EC001   Name: ABHISHEK C   Average Marks: 13.5
USN: 2KD22EC002   Name: ABHISHEK N   Average Marks: 16
USN: 2KD22EC003   Name: ADITYA SHINGE   Average Marks: 16
USN: 2KD22EC004   Name: ADITYA S   Average Marks: 15
USN: 2KD22EC005   Name: AISHWARYA   Average Marks: 16
USN: 2KD22EC006   Name: AKHIL R   Average Marks: 17.5
USN: 2KD22EC007   Name: AMOGH   Average Marks: 16.5
USN: 2KD22EC008   Name: AMRUTA   Average Marks: 17.5
USN: 2KD22EC009   Name: ANISHA   Average Marks: 16
USN: 2KD22EC010   Name: ANITA   Average Marks: 14.5
```

Enter details for Student 1: USN: 2KD22EC001

Name: ABHISHEK C

Marks for 3 tests: 12

13

14

Details of all students: USN: 2KD22EC001 Name: ABHISHEK C Average Marks: 12.5

4. 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 (m1 == m2) then m3 = m1 + m2 and m4 = m1 - m2 else display error.

PROGRAM:

```
#include <iostream.h>

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];

}
}
}

int 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:

```
C:\TURBOC3\BIN>TC
Enter the elements of Matrix 1:
Enter the elements of the matrix:3 x 3
1 2 3
3 4 5
1 2 3
Enter the elements of Matrix 2:
Enter the elements of the matrix:3 x 3
1 2 3
1 2 3
1 2 3
Matrix 1 + Matrix 2:
2 4 6
4 6 8
2 4 6

Matrix 1 - Matrix 2:
0 0 0
2 2 2
0 0 0
```

```
Enter the elements of Matrix 1:
Enter the elements of the matrix: 3 x 3
1 2 3
3 4 5
1 2 3
Enter the elements of Matrix 2: 3 x 3 Enter the elements of the matrix:
1 2 3
1 2 3
1 2 3
Matrix 1 + Matrix 2:
2 4 6
4 6 8
2 4 6
Matrix 1 - Matrix 2:
0 0 0
2 2 2
0 0 0
```

5. 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 F1 & S1 objects with appropriate constructors & display the FATHER & SON details.

PROGRAM:

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

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

Process returned 0 (0x0)   execution time : 49.466 s
Press any key to continue.
|

```

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

6. 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.h>

class Son; // Forward declaration for Son class

class Father {
private:
    float income;

public:
    Father(float inc) : income(inc) {}

    // Friend function declaration
    friend float calculateTotalIncome(const Father& father, const Son& son);
};

class Son {
private:
    float income;

public:
    Son(float inc) : income(inc) {}

    // Friend function declaration
    friend float calculateTotalIncome(const Father& father, const Son& son);
};

// Friend function definition
float calculateTotalIncome(const Father& father, const Son& son) {
    return father.income + son.income;
}

void main() {
    // Creating objects of Father and Son classes
    int fatherincome,sonincome;
    cout<<"enter father income\n";
    cin>> fatherincome;
    cout<< "enter son income\n";
    cin>>sonincome;
    Father father(fatherincome);
    Son son(sonincome);

    // Calculating and displaying total income using the friend function
    float totalIncome = calculateTotalIncome(father, son);

    // Displaying the result
```

```
    cout << "Total Family Income: $" << totalIncome << endl;  
}
```

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  
  
Process returned 0 (0x0)   execution time : 18.661 s  
Press any key to continue.
```

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

7. 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.h>
#include<conio.h>

class Student {
private:
    char name[30];
    float marks[3];

public:
    void get_data();
    void display();
    friend float mark_avg(Student s);
};

void Student::get_data() {
    cout << "Enter student name: ";
    cin.getline(name, 30);

    cout << "Enter marks for 3 subjects:\n";
    for (int i = 0; i < 3; i++) {
        cout << "Subject " << (i + 1) << ": ";
        cin >> marks[i];
    }
}

void Student::display() {
    cout << "\nStudent Name: " << name << endl;
    cout << "Average Marks: " << mark_avg(*this) << endl;
}

float mark_avg(Student s) {
    float sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += s.marks[i];
    }
    return sum / 3;
}

int main() {
    clrscr();

    Student student1;
    student1.get_data();
```

```
student1.display();

getch();
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

Process returned 0 (0x0)   execution time : 18.189 s
Press any key to continue.
|
```

```
Enter student name: Harish
Enter marks for three subjects: 2825 63
Student Name: Harish Average Marks: 38.6667
```

8. 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.h>
#include<conio.h>

class Polygon {
public:
    virtual float area() {
        return 0;
    }
};

class Rectangle : public Polygon {
private:
    float length, breadth;

public:
    void get_data() {
        cout << "Enter length of the rectangle: ";
        cin >> length;
        cout << "Enter breadth of the rectangle: ";
        cin >> breadth;
    }

    float area() {
        return length * breadth;
    }
};

class Triangle : public Polygon {
private:
    float base, height;

public:
    void get_data() {
        cout << "Enter base of the triangle: ";
        cin >> base;
        cout << "Enter height of the triangle: ";
        cin >> height;
    }

    float area() {
        return 0.5 * base * height;
    }
};
```

```
int main() {
    clrscr();

    Polygon *poly;

    Rectangle rect;
    rect.get_data();
    poly = &rect;
    cout << "\nArea of Rectangle: " << poly->area() << endl;

    Triangle tri;
    tri.get_data();
    poly = &tri;
    cout << "\nArea of Triangle: " << poly->area() << endl;

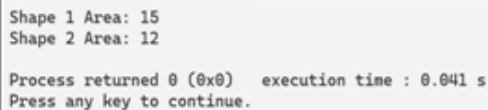
    getch();
    return 0;
}
```

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

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

Shape 1 Area: 15
Shape 2 Area: 12

9. 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 = 123% of Basic, Income Tax (IT) = 30% of gross salary (=basic_Salary_All_Allowances_IT)).

PROGRAM:

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

class Employee
{
    char emp_name[30];
    int emp_number;
    float basic, da, it, gross_salary, net_salary;
public:
    void read_emp_details(){

        cout<<"\nEmployee Number: ";
        cin>>emp_number;
        cout<<"Employee Name: ";
        cin>>emp_name;
        cout<<"Basic Salary: ";
        cin>>basic;
        cout<<"\n---- Employee Daitails are saved ----\n\n";
    }
    float find_net_salary(){
        da = basic * 1.23;
        gross_salary = basic + da;
        it = gross_salary * 0.30;
        net_salary = (basic + da) - it;
        return net_salary;
    }
    void display_emp_details(){
        cout<<"\n\n*** Employee Details ***\n\n";
        cout<<"\nEmployee Number    : "<<emp_number;
        cout<<"\nEmployee Name        : "<<emp_name;
        cout<<"\nBasic Salary      : "<<basic;
        cout<<"\nAll allowance    : "<<da;
        cout<<"\nGross Salary     : "<<gross_salary;
        cout<<"\nIncome Tax       : "<<it;
        cout<<"\nNet Salary      : "<<net_salary;

    }
};
int main(){
    Employee emp;
```



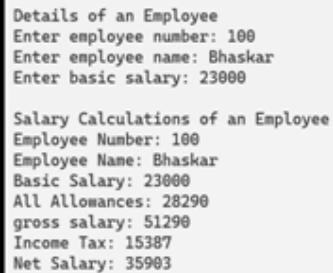
```
int number_of_emp;
clrscr();
emp.read_emp_details();
emp.find_net_salary();
emp.display_emp_details();

cout<<"\nPress any key to close!!!";
getch();
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
```

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: 359

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.h>
#include <conio.h>

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();
    getch();
    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

Process returned 0 (0x0)   execution time : 2.582 s
Press any key to continue.
|
```

Base1::protectedData1 = 10

Base2::protectedData2 = 20

Derived::privateData = 30

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.h>
#include <conio.h>

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();
    getch();
    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: 0x6ec9dfffb80
Object with roll_no 2 and name Bob invoked display()
This pointer value: 0x6ec9dfffb50
Object with roll_no 3 and name Charlie invoked display()
This pointer value: 0x6ec9dfffb20

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

Object with roll_no 1 and name Alice invoked display() This pointer value: 0x6ec9dfffb80
Object with roll_no 2 and name Bob invoked display() This pointer value: 0x6ec9dfffb50
Object with roll_no 3 and name Charlie invoked display() This pointer value: 0x6ec9dfffb20

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

PROGRAM:

```
#include <iostream.h>
#include <exception.h>
#include <stdexcept.h>

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:
case 2:
case 3:
case 4:
throw CustomException1(); throw CustomException2(); throw CustomException3();
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.

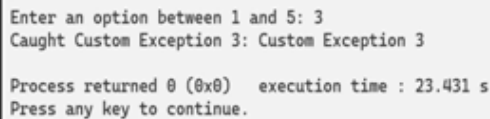
If the choice is 5, it throws a out_of_range exception with a custom message.

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

Process returned 0 (0x0)   execution time : 23.431 s
Press any key to continue.
```

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