SOP

&

Lab Manual

for

C ++ PROGRAMMING

DEPARTMENT OF COMPUTER SCIENCE (AIDED)

Supported by

DBT Star College Scheme

Published by

KONGUNADU ARTS AND SCIENCE COLLEGE (Autonomous)

Re-accredited by NAAC with A^+ Grade – 4^{th} Cycle
52^{nd} Rank among Colleges in NIRF 2024
College of Excellence (UGC)
Coimbatore -641 029.
2024-2025

STANDARD OPERATING

PROCEDURE

FOR

C ++ PROGRAMMING

PREFACE

Welcome to the C++ Programming Lab Manual! This manual is designed to accompany your journey into the world of C++ programming, providing practical guidance, exercises, and insights to reinforce your learning experience.

C++ programming is a fundamental skill for aspiring programmers, offering a strong foundation in computer science and software development. This lab manual is designed to provide students with hands-on experience in C ++ programming through structured exercises, real-world problem-solving, and practical applications.

The manual covers key concepts such as variables, data types, control structures, functions, arrays, pointers, and file handling, ensuring a comprehensive understanding of the language. Each lab session is structured to reinforce theoretical knowledge through practical implementation, enabling students to develop logical thinking and problem-solving skills.

By following this lab manual, students will gain confidence in writing efficient and optimized C++ programs. The exercises progressively enhance their coding abilities, preparing them for more advanced topics in software development, embedded systems, and algorithm design.

We hope this manual serves as a valuable resource for students, fostering a deeper understanding of C ++programming and its applications.

Standard Operating Procedure

Standard Operating Procedures (SOPs) in java programming refer to a set of guidelines and documented processes that outline the best practices, workflows, and steps involved in various aspects of developing java applications and some common components typically covered in SOPs for java programming

1. Requirements Gathering:

- Define the process for gathering and documenting client requirements for the mobile application.
- Specify methods for conducting interviews, surveys, or workshops to identify and prioritize features and functionalities.
- Outline the documentation format for capturing and validating requirements.

2. Design and Wireframing:

- Describe the process for creating the visual design and user interface (UI) of the mobile application.
- Explain how wireframing and prototyping tools are utilized to develop mockups and obtain feedback .
- Define guidelines for ensuring consistency in UI elements, navigation, and usability across different screens and devices.

3. Development Environment Setup:

- Specify the required development tools, software, and platforms for building the mobile application.
- Provide instructions for configuring integrated development environments (IDEs), emulators, and simulators for testing and debugging.
- Outline the process for setting up version control systems and collaborative tools for code management and team communication.

4. Coding Standards and Guidelines:

- Establish coding standards, naming conventions, and best practices for writing clean, maintainable, and efficient code.
- Define guidelines for organizing project files, folders, and code repositories.
- Specify documentation requirements, such as code comments or inline documentation.

5. Development Process:

- Outline the overall development process, such as Agile or waterfall methodologies, and how it applies to mobile app development.
- Define the roles and responsibilities of team members involved in the development process, including developers, testers, and project managers.

6. Testing and Quality Assurance:

- Specify the testing approaches and methodologies to be followed, including unit testing, integration testing, and user acceptance testing.
- Describe the process for identifying and reporting bugs or issues, including bug tracking tools and workflows.
- Outline the criteria for ensuring the quality and performance of the mobile application before release.

7. Deployment and Release:

- Define the process for packaging and deploying the mobile application to app stores (e.g., Apple App Store, Google Play Store).
- Specify the requirements and guidelines for creating app store listings, including app descriptions, screenshots, and promotional materials.
- Explain the process for obtaining necessary certificates, signing the app, and handling updates or version releases.

8. Maintenance and Support:

- Provide guidelines for post-release maintenance and support, including bug fixes, performance optimizations, and feature enhancements.
- Outline the process for gathering user feedback, conducting user surveys, and incorporating user suggestions into future updates.
- Describe the mechanisms for monitoring and analyzing the app's performance, crash reports, and analytics data.

These guidelines can serve as a starting point, but it's recommended to adapt and customize them to align with your organization's unique requirements and practices in C programming.

INDEX

C ++ PROGRAMMING LAB EXPERIMENTS

1. STACK OPERATION

AIM:

To write a C++ program to develop stack operation.

ALGORITHM:

STEP 1: Include all header file.

STEP 2: Create a class stack consist of a function push(), pop().

STEP 3: Get the data from the user.

STEP 4: Check for the condition underflow overflow.

STEP 5: Terminate the program

CODING

```
#include<iostream>

using namespace std;

class stack

{

private:

int top,n;

int s[10];

public:

stack()

{top=0;}

void init();

void push();
```

```
void pop();

void disp();

};

void stack::init()

{int i;

cout<<"\n Maximum number of elements";

cin>>n;

for(i=0;i<n;i=i+1)

s[i]=0;

}

void stack::push()

{

if (top<n)
```

```
{

cout<<"Enter number";

cin>>s[top];

top=top+1;
}
else

cout<<"\nOVERFLOW.";

}

void stack::pop()

{

top=top-1;

if(top<0)

{

cout<<"\nUNDERFLOW..";
```

```
top=0;

}

}

void stack::disp()

{

int i;

cout<<"\n The elements are...";

for(i=0;i<top;i=i+1)

cout<<"\n"<<s[i];

}

int main()

{

int ch;
```

```
stackt

t.init();

menu:

cout<<"\nMENU...";

cout<<"\n 1-> PUSH ";
cout<<"\n 2-> POP ";

cout<<"\n 3-> DISPLAY ";

cout<<"\n 4-> EXIT ";

cout<<"\n Enter your choice (1-4) ";

cin>>ch;

switch(ch)

{

case 1:

t.push();
```

```
goto menu;

case 2:

t.pop();

goto menu;

case 3:

t.disp();

goto menu;

case 4:

cout<<"\n EXITED ..... ";

}

}
```

OUTPUT:

Maximum number of elements 30

MENU...

1 PUSH

2 POP

3 DISPLAY

4 EXIT

Enter your choice(1-4) 1

Enter number 28

MENU...

1 PUSH

2 POP

3 DISPLAY

4 EXIT

Enter your choice(1-4) 1

Enter number 29

MENU...

1 PUSH

2 POP

3 DISPLAY

4 EXIT

Enter your choice(1-4) 3

The elements are 28

29

د

MENU...

1 PUSH

2 POP

3 DISPLAY

4 EXIT

2.ARITHMETIC OPERATION

AIM:

Create a class arithmetic which contains a float and an integer number, write member function ADD(), SUB(), MUL(), DIV(), MOD() to perform addition, subtraction, multiplication, division, module. Respectively write a member function to get and display values.

ALGORITHM:

STEP 1: Include the header file.

STEP 2: Create a class Arith which consist of one integer and one float variable.

STEP 3: Write a member function ADD(). SUB(), MUL(), DIV(), MOD().

STEP 4: To perform addition, subtraction, multiplication, division, module.

STEP 5: Get the values from the user and calculate display the output.

CODING

```
#include<iostream>

using namespace std;

class Arith

{

private:

int a;

float b,c;

public:

void getdata();

void putdata();

void ADD();

void SUB();
```

```
void MUL();

void DIV();

void MOD();

};

void Arith::getdata()

{

cout<<"\n Enter Int number.

cin>>a;

cout<<"\n Enter in floating number..... ";

cin>>b;

}

void Arith::putdata()

{
```

```
cout<<"\n a = "<<a;

cout<<"\n b = "<<b;

}
```

[12:01 pm, 20/3/2025] Balanethraa Kasc: void Arith::ADD()

```
{

c=a+b;

cout<<"\n Sum ="<<c;

}

void Arith::SUB()

{

c=a-b;

cout<<"\n Subtraction ="<<c;

}

void Arith::MUL()
```

```
{
  c=a*b;
  cout<<"\n Multiplication ="<<c;
  }
  void Arith::DIV()
  {
  c=a/b;
  cout<<"\n Division ="<<c;
  }
  void Arith::MOD()
  {
  c=a%(int)b;
  cout<<"\n Remainder ="<<c;
```

```
}
  int main()
  {
  Arith t;
  t.getdata();
  t.putdata();
  t.ADD();
  [12:02 pm, 20/3/2025] Balanethraa Kasc: t.SUB();
  t.MUL();
  t.DIV();
  t.MOD();
  return 0;
  }
```

OUTPUT:

Enter int number 23

Enter float number 27

a=23

b=27

sum=50

subtraction=-4

multiplication=621

division=0.851852

remainder=2

3. OPERATOR OVERLOADING WITH MATRIX

AIM:

Create a matrix using a class MAT and variables R and C to represent rows and columns. Overload the operators +, -, and * to add, subtract, and multiply two matrices. Include member functions get() and display() for input and output operations.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Declare a class MAT to represent rows and columns.

STEP 3: Get the values of the matrix from the user using the get() function.

STEP 4: Overload the operators +, -, and * to perform addition, subtraction, and multiplication of matrices.

STEP 5: Display the output using the display() function

CODING

```
#include<iostream>

using namespace std;

class MAT

{

private:

int r1,c1,a[10][10],i,k,j;

public:

void getdata();

void putdata();

MAT operator +(MAT);

MAT operator -(MAT);
```

```
MAT operator *(MAT);

};

void MAT::getdata()

{

cout<<"\n Enter no of rows ";

cin>>r1;

cout<<"\n Enter no of columns";

cin>>c1;

cout<<"\n Enter values :";

for(i=0;i<r1;i=i+1)

{

for(j=0;j<c1;j<j+1)

{
```

```
cin>>a[i][j];

}

}

}

void MAT::putdata()

{
cout<<"\n";

for(i=0;i<r1;i=i+1)

for(j=0;j<c1;j=j+1)

{

cout<<a[i][j]<<"\t";
}

cout<<"\n";

}
```

```
}

MAT MAT::operator + (MAT b)

{

MAT c;

for(i=0;i<r1;i=i+1)

{

for(j=0;j<c1;j=j+1)

{

c.a[i][j] = a[i][j] + b.a[i][j];

}

cout<<"\n";

}

c.r1=r1;
```

```
c.c1=c1;

return (c)

}

MAT MAT::operator - (MAT b)

{

MAT C;

for(i=0;i<r1;i=i+1
[12:07 pm, 20/3/2025] Balanethraa Kasc: {

for(j=0;j<c1;j=j+1)

{

c.a[i][j]=a[i][j]-b.a[i][j];

}

cout<<"\n";

}
```

```
c.r1=r1;

c.c1=c1;

return (c);

ע

}

MAT MAT::operator (MAT b)

{

MAT c;

for(i=0;i<r1;i=i+1)

{

for(j=0;j<c1;j=j+1)

{

c.a[i][j]=0;
```

```
for(k=0;k<c1;k=k+1) { c.a[i][j] = ca[i][j]+a[i][k]*b.a[k][j];

}

}

cout<<"\n";

}

c.r1=r1;

c.c1=b.c1;

return (c);

}
 int main()

{

MAT a,b,c;

a.getdata();
```

```
b.getdata();

cout<<"\n given matrices are.....";

a.putdata();

b.putdata();

c=a+b;

cout<<"\n a+b is ";

c.putdata();

c=a-b;

cout<<"\n a-b is ";

c.putdata();

c=a*b;

cout<<"\n a*b is ";

c.putdata();
```

```
return 0;
  }
```

OUTPUT:

Enter no of rows 3

Enter no of columns 3

Enter value: 1

2

3

4

5

6

7

8

9

Enter no of rows 3

Enter no of rows 3

Enter value: 1

2

3

4

5

6

7

8

9

Given matrix are

1 2 3
4 5 6
7 8 9

1 2 3

4 5 6

7 8 9

a+b is

2 4 6

8 10 12

14 16 18

a-b is

2 4 6

8 10 12

14 16 18

a*b is

4096 0 0

1 0 1

4096 0 1

4.VIRTUAL FUNCTION

AIM:

Write a program to define classes A, B, and C. Class C is derived from A and B. All member functions are virtual. Count the number of objects created and display the output.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create classes A, B, and C. Class C is derived from A and B.

STEP 3: Define the member functions of all the classes as virtual.

STEP 4: Count the number of objects created.

STEP 5: Display the output.

CODING

```
#include<iostream>

using namespace std;

int nobj=0;

class A

{

public:

virtual void count()

{

cout<<"\n classs A ";

nobj=nobj +1;

}
```

```
void putcount()

{

cout<<"\n no of objects \t"<<nobj;

}

};

class B

{

public:

virtual void count()

{

cout<<"\n class = B";

nobj=nobj+1;

}
```

```
};

class C:public A, public B

{

public:
virtual void count()

{

cout<<"\n class = C";

nobj=nobj+1;

}

};

int main()

{

A a1,a2;

Bb;
```

```
C c;

a1.count();

a2.count();

b.count();

c.count();

c.putcount();

return 0;

}
```

OUTPUT:

Class a

Class b

Class = B

Class = C

No of objects

5. BANK TRANSACTION

AIM:

Define a class to represent a bank account, including the following members:

1. Name of the depositor , 2. Account number , 3. Type of account , 4. Balance amount.

Member functions:1. To assign initial values , 2. To deposit an amount , 3. To withdraw an amount , 4. To display the account holder's name and balance.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a class Bank and include details like the depositor's name, account number, account type, and balance.

STEP 3: Write member functions to:

Deposit an amount

Withdraw an amount

Display the account holder's name and balance

STEP 4: Display the output

CODING

```
#include<iostream>

using namespace std;

float amt=0;

class Bank

{

private:

char name[25];

int accno;

char type[10];

float bal;

public:
```

```
void getdata()

{

cout<<"\n Enter name";

cin>>name;

cout<<"\n Account number";

cin>>accno;

cout<<"\n Enter types of account";

cin>>type;

cout<<"\n Enter Balance";

cin>>bal;

}

void deposit()

{
```

```
cout<<"\n Enter amount to deposit: ";

cin>>amt;

bal=bal+amt;

cout<<"\n Balance = "<<bal;

}

void withdraw()
 {
cout<<"\n Enter amount to withdraw";

cin>>amt;
{

if (amt>bal)

cout<<"\n No sufficient balance....";
}

}

else
```

```
cout<<"\n amount available ="<<bal;

bal-bal-amt;

cout<<"\n Balance: "<<bal;

}

}

void display()

{

cout<<"\n Name =" << name;

cout<<"\n Account no =" << accno;

cout<<"\n Type of account" << type;

cout<<"\n Balance =" << bal;

}

};
```

```
int main()

{

int ch;

Bank b1;

b1.getdata();

start:

}
 cout<<"\n 1-> Deposit";

cout<<"\n 2-> Withdraw";

cout<<"\n 3-> Display ";

cout<<"\n 4-> Exit";

cout<<"\n choice (1-4): ";

cin>>ch;

switch(ch)
```

```
{
case 1:
د
b1.deposit();
goto start;
د
case 2:
b1.withdraw();
goto start;
case 3:
b1.display();
goto start;
case 4:
```

```
cout<<"\n Exiting";

}

}
}
```

OUTPUT

Enter name: Harini

Account number: 2327

Enter type of account savings

Enter balance 20,000

1-deposit

2->withdraw

3->display

4->exit

Choice(1-4): 1

Enter amount to deposit: 10000

Balance 30000

1->deposit

2->withdraw

3->display

4->exit

Choice(1-4): 2

Enter amount withdraw 500

Amount available = 30000

Balance: 29500

1->deposit

2->withdraw

3->display

4->exit

Choice(1-4): 3

Name: harini

Account number = 2327

Type of account = savings

Balance 29500

1->deposit

2->withdraw

3->display

4->exit

Choice(1-4): 4

Exitin

6. DESTRUCTOR

AIM:

Write a program to implement a destructor.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a class that includes a constructor and a destructor.

STEP 3: Use the constructor to create objects.

STEP 4: Use the destructor to destroy the objects.

STEP 5: Display the output showing how many objects are destructor.

CODING:

```
#include<iostream>

using namespace std;

int c=0;

class A

{public:

A()

{c=c+1;

cout<<"\n One object created " << c;

}

~A()

{
```

```
c=c-1;

cout<<"\n One object destroyed = "<< c;

}

};

int main()

{

A a1,a2,a3;

{

cout<<"\n Entering block 1";

A a4;

cout<<"\n Exiting block 1";

}

A a5;
```

```
{

cout<<"\n Entering block 2 ";

A a6,a7;

cout<<"\n Exiting block 2";

}
return 0;

}
```

OUTPUT:

One object created 1

One object created 2

One object created 3

Entering block 1

One object destroyed 3

One object created 4

Entering block 2

One object created 5

One object created 6

Exiting block 2

One object destroyed = 5

One object destroyed = 4

One object destroyed = 3

One object destroyed = 2

One object destroyed = 1

One object destroyed = 0

7. MULTILEVEL INHERITANCE

AIM:

Write a program to implement multilevel inheritance.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a class Student and derive another class Marks from it.

STEP 3: Create a class Result that is inherited from Marks.

STEP 4: Enter the roll number in the Student class, get marks in the Marks class, and calculate the total in the Result class.

STEP 5: Display the output.

CODING

```
#include<iostream>

using namespace std;

class stud

{

protected:

char rno[10];

public:

void getrno()

{

cout<<"\n Enter rno :";

cin>>rno;
```

```
}
void putrno()
{
cout<<"\n Register no = "<<rno;
}
};
class test: public stud
{
protected:
int m1, m2;
public:
void getmarks()
{
```

```
cout<<"\n Enter marks: ";

cin>>m1>>m2;

void putmarks()
 {

cout<<"\n mark1 =" <<m1;

cout<<"\n mark2 =" <<m2;

}

};

class result: public test

{

private:

int tot;

public:
```

```
void calcdisp()

{

tot=m1+m2;

putrno();

putmarks();

cout<<"\n Total = " << tot;

}

};

int main()

{

result r1;

r1.getrno();

r1.getmarks();
```

```
r1.calcdisp();

}
```

OUTPUT:

Enter rno: 12

Enter marks: 50

90

Register no: 12

Mark 1 = 50

Mark 2 = 90

Total = 140

8. FUNCTION OVERLOADING

AIM:

Write a program to overload a member function in both base and derived classes.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create two classes (base and derived) and define a method with the same name in both classes.

STEP 3: Create an object of the derived class.

STEP 4: Call the method using the derived class object.

STEP 5: Display the output

CODING

```
#include<iostream>

usingnamespace std;

class parent

{

public:

void display()

{

cout<<"Parent class";

}

};
```

```
classchild:public parent
{
public:
void display()
{
cout<<"child class";
}
};
int main()
{
child oc;
oc.display();
}
```

OUTPUT:

Child class parent class

9. STRING OPERATION

AIM:

Create a class String. Write member functions to initialize, get, and display strings. Overload the operators + to concatenate two strings and == to compare two strings. Also, write a member function to find the length of a string.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a class String.

STEP 3: Overload the operator + to concatenate two strings and the operator == to compare two strings.

STEP 4: Write a member function to find the length of a string.

STEP 5: Display the output

CODING

```
#include<iostream>

#include<string>

#include<cstring>

using namespace std;

class Strings

{

private:

char str[80];

public:

Strings()

{
```

```
strcpy(str, "");

}

Strings(const char s[])

{

strcpy(str, s);

}

void display()

{

cout << "\n" << str << "\n";

}

void length()

{

cout << "\n Length of the string = " << strlen(str) << "\n";
```

```
}

Strings operator + (Strings s2)

{

Strings 53;

strcpy(s3.str, str);
 strcat(s3.str, s2.str);

return $3;

}

bool operator == (Strings s2)

{

cout << "\n" << str << " " << s2.str;

if (strcmp(str, s2.str) == 0)

return true;

else
```

```
return false;

}

};

int main()

{

Strings s1 = "kongu";

Strings s2 = "nadu";

Strings s3;

s1.display();

s1.length();

s2.display();

s2.length();

s3 = s1 + s2;
```

```
s3.display();

if (s1 == s2)

cout << "\n \n Equal";

else
 cout << "\n \n Not equal";

return 0;

}
```

OUTPUT:

Kongu

Length of the string = 5

Nadu

Length of the string = 4

Kongunadu

Kongu nadu

Not equal

10. PAYSLIP PREPARATION

AIM:

Create a class to manage details of an employee, including employee ID, name, department, basic salary, and grade. Write functions to get and display the details. Derive a class from the above class to include member functions for calculating PF, HRA, and DA based on the grade. Display the output using I/O operations.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Get the employee details from the user.

STEP 3: Write member functions to calculate PF, HRA, and DA based on the grade.

STEP 4: Display the calculated results along with the employee details.

CODING

```
#include<iostream>

#include<iomanip>

using namespace std;

class Emp

{

protected:

int eno;

char ename[25], dept[5],gr;

float bp;

public:

void getdata()
```

```
{
cout<<"\n emp number ";
cin>>eno;
cout<<"\n emp name ";
cin>>ename;
cout<<"\n department ";
cin>>dept;
cout<<"\n grade A/B/C ";
cin>>gr;
cout<<"\n basic pay ";
cin>>bp;
}
};
```

```
class Pay: public Emp

{

private:

float da, hra, pf,gross, dedn, nett;

public:

void calc()
[20/03, 12:31] Balanethraa K: {

if (gr==^ prime a^ prime )

{

da = bp * 0.3

hra=1000;

pf =bp^ * 0.0833;

}
```

```
if (gr==^ prime b^ prime )

{

da = bp * 0.4

hra=2000;

pf = bp * 0.0833

}

if (gr==^ prime c^ prime )

{

da = bp * 0.5

hra=3000;

pf = bp * 0.0833

}

gross=bp+da+hra;
```

```
dedn=pf;

nett-gross-dedn;

void putdata()

{

cout.setf(ios::fixed, ios::floatfield);

cout.precision(2);

cout<<"\n Emp no = " << eno;

cout<<"\n Emp name = "<<ename;

cout<<"\n department = " << dept;
[20/03, 12:31] Balanethraa K: cout<<"\n grade = " << gr;

cout<<"\n basic pay = " << bp;

cout<<"\n DA = " << da;

cout<<" \n HRA = " << hra;

cout<<"\n PF = " << pf;
```

```
cout<<"\n gross pay = " << gross;

cout<<"\n deduction =" << dedn;

cout<<"\n net pay = " << nett;

}

};

int main()

{

const int n=2;

int i;

Pay p;

p.getdata();

p.calc();

cout<<"\n pay slip \n";
```

```
p.putdata();

cout<<"\n press key";

return 0;

}
```

OUTPUT:

Emp number: 2330

Emp name: Dipeeka

Department: cs

Grade A/B/C: a

Basic pay: 5000

Pay slip

Emp no: 2330

Emp name: Dipeeka

Department: cs

Grade: a

Basic pay: 5000.00

DA: 1500.00

HRA: 100.00

PF: 416.50

Gross.pay = 7500.00

Deduction = 416.50188543622170 83.50

11. CALCULATION OF AREA AND PERIMETER USING VIRTUAL FUNCTION

AIM:

Create two classes that consist of private members and virtual functions calc_area() and calc_perimeter() to calculate the area and perimeter of various figures like squares, rectangles, and triangles. Display the calculated values.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a base class that consists of method declarations for calc_area() and calc_perimeter().

STEP 3: Define these functions in derived classes for different shapes.

STEP 4: Get data from the user and calculate the area and perimeter in the derived classes.

STEP 5: Display the calculated area and perimeter values.

CODING

```
#include<iostream>

#include<cmath>

using namespace std;

class Shape

{

protected:

float a,b,c,l,ar, pr,s;

public:

virtual void cal_area()

{

}
```

```
virtual void cal_peri()

{

}

};

class Square:public Shape

{

public:

void getdata()

{

cout<<"\n Enter length of a side: ";

cin>>a;

}

void cal_area()
```

```
{
ar=a*a;
}
void cal_peri()
 {
pr=4*a;
}
void display()
{
cout<<"\n Area of square: "<<ar;
cout<<"\n Perimeter of square: "<<pr;
}
};
class Rectangle: public Shape
```

```
{
public:
void getdata()
{
cout<<"\n Enter length and breadth:";
cin>>l>>b;
}
void cal_area()
{
ar=l*b;
}
void cal_peri()
{
```

```
pr=2*(1+b);

}

void display()

{

cout<<"\n Area of rectangle: "<<ar;

cout<<"\n Perimeter of rectangle : "<<pr;

}
};
class Triangle: public Shape

public:

{

void getdata()

{

cout<<"\n Enter a,b,c :";
```

```
cin>>a>>b>>c;

}

void cal_area()

{

s=(a+b+c)/2.0;

ar=sqrt(s*(s-a) (s-b)* (s-c));

}

void cal_peri()

{

pr=a+b+c;

}

void display()

{
```

```
cout<<"\n Area of triangle: "<<ar;

cout<<"\n Perimeter of triangle: "<<pr;

}

};

int main()

{

Shape "bptr;

Square s;

Rectangle r;

Triangle t;

bptr=&s;

s.getdata();
bptr->cal_area();
```

```
bptr->cal_peri();

s.display();

bptr=&r;

r.getdata();

bptr->cal_area();

bptr->cal_peri();

r.display();

bptr=&t;

t.getdata();

bptr->cal_area();

bptr->cal_peri();

t.display();

return 0;}
```

OUTPUT:

Enter length of a scale: 20

Area of square: 400

Perimeter of square: 80

Enter length and breath: 15

34

Area of rectangle: 34

Perimeter of rectangle: 70

Enter a,b,c: 2

3

4

Area of triangle: 2.90474

Perimeter or triangle: 9

12. FRIEND FUNCTION

AIM:

Create two classes with private variables in each. Write a program to add two integers and one float variable using a friend function. The friend function will access and combine the values from both classes.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create two classes and declare a friend function in both classes.

STEP 3: Declare one integer and one float variable in each class.

STEP 4: Pass the objects of both classes to the friend function and calculate the sum of the variables.

STEP 5: Display the output.

CODING

```
#include<iostream>

using namespace std;

class two;

class one

{

private:

int a;

float b;

public:

void getdata()

{

cout<<"\n Enter the integer value :";
```

```
cin>>a;

cout<<"\n Enter the floating value :";

cin>>b;

}

void putdata()

{

cout<<"\n A =" << a;

cout<<"\n B =" << b;

}

friend void add(one,two);

class two
```

```
{
  private:
  int m;
  float n;
   public:
  void getdata()
  {
  cout<<"\n Enter the integer value :";
  cin>>m;
  cout<<"\n Enter the floating value :";
  cin>>n;
  }
  void putdata()
  {
```

```
cout<<"\n M =" << m;

cout<<"\n N =" << n;

}

friend void add(one, two);

};

void add(one p,two q)

{

cout<<"\n sum of int values: "<<p.a + q.m;

cout<<"\n sum of floating values: "<<p.b + q.n;

}

int main()

{

one p;
```

```
two q;

p.getdata();

q.getdata();

p.putdata();

q.putdata();
add(p, q);

return 0;

}
```

OUTPUT:

Enter the integer value: 23

Enter the floating value: 12.23

Enter the interger value: 56

Enter the floating value: 90.7

A = 23

B = 12.23

M = 56

N = 90.7

Sum of int values: 79

Sum of floating values: 102.93

13. MULTIPLICATION TABLES

AIM:

Create a program to define user functions that use formatting to display multiplication tables. Use formatting commands like setprecision(), and other manipulators to display the tables properly.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Create a user-defined function to apply formatting for displaying tables.

STEP 3: Accept a number from the user and generate its multiplication table.

STEP 4: Use the formatting function to display the multiplication table neatly.

STEP 5: Display the output.

CODING

```
#include<iostream>

#include <iomanip>

using namespace std;

ostream & userfun(ostream &out)

{

setw(8);

cout.setf(ios::showpoint);

cout.setf(ios::showpos);

cout.precision(2);

return (out);

}
```

```
int main()

{

float n,i;

cout<<"Enter a number";

cin>>n;

cout<<"\n Multiplication table of " << n;

for(i=1;i<=10;i=i+1)

{

cout<<"\n" << userfun <<n <<"*" << i <<" = " << n*i;

}

return 0;

}
```

OUTPUT:

Enter a number 5

Multiplication table of 5

+5.0^ * + 1 = 5

+5.0^ * + 2 = 10 .

+5.0^ * + 3 = 15 .

+5.0^ * + 4 = 20 .

+5.0^ * + 5 = 25 .

+5.0^ * + 6 = 30

+5.0^ * + 7 = 35 .

+5.0^ * + 8 = 40 .

+5.0^ * + 9 = 45 .

+5.0^ * + 10 = 50 .

14. FUNCTIONS WITH DEFAULT ARGUMENTS

AIM:

Write a program to implement the concept of default arguments in functions.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Define a function with default arguments that take default values when arguments are not provided.

STEP 3: Get the input data from the user (if required) and call the function.

STEP 4: Calculate the result using the function.

STEP 5: Display the output.

CODING

```
#include<iostream>

using namespace std;

int main()

{

float add(float, float, float=50);

float a,b,c,sum;

cout<<"\n enter 3 values : ";

cin>>a>>b>>c;

sum=add(a,b,c);

cout<<"\n sum = "<<sum <<"\n";

cout<<"\n Enter two numbers ";
```

```
cin>>a>>b;

sum=add(a,b);

cout<<"\n sum = " << sum;

return 0;

}

float add(float x, float y, float z)

{

return (x+y+z);

}
```

OUTPUT:

Enter 3 values: 23

30

16

Sum=69

Enter two numbers 53

12

Sum= 11

15. FILE CONCEPTS

AIM:

Write a program to use two files. The program should take file names as command-line arguments, copy the contents of the first file to the second file line by line, and display the output.

ALGORITHM:

STEP 1: Include the necessary header files

STEP 2: Declare the arguments in the main() function to handle command-line arguments.

STEP 3: Open the first file, read its contents, and display them.

STEP 4: Copy the contents of the first file line by line into the second file.

STEP 5: Display the output to confirm successful copying.

CODING

```
#include <iostream>

#include <fstream>

#include <string>

int main()

{

std::ifstream sourceFile("z:\\j.txt");

std::ofstream destFile("z:\\j1.txt");

if (!sourceFile) {

std::cerr << "Error opening source file!" << std::endl;

return 1;

}
```

```
if (!destFile)

{

std::cerr << "Error opening destination file!" << std::endl;

return 1;

}

std::string line;

while (getline(sourceFile, line)) {

destFile << line << std::enell,

}

std::cout << "Text has been copied successfully!" << std::endl
}
sourceFile.close();

destFile.close();

return 0;
}
```

OUTPUT:

If both files z: j.txt and z:\j1.txt

Are successfully opened:

Text has been copied successfully!

If the source file z:\j.txt cannot be

Opened, the output will be:

Error opening source file!

It the destination file z:\j1. Txt cannot be

Opened, the output will be:

Error opening destination file!

16. STUDENT STRUCTURE

AIM:

Write a program to create a structure for a student to store and display student details.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Define a Student structure with fields for roll number, name, age, and marks.

STEP 3: Declare a variable of type Student to hold the student's information.

STEP 4: Store values in the fields of the Student structure.

STEP 5: Display the student's information and terminate the program.

CODING

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

Struct student {

Char name[50];

int roll; };

float marks;

Int main() {

student s;

cout<<"Enter information,"<<endl;

cout<<"enter name:";

cin>>s.name;

cout<<"enter roll number:";
```

```
cin>>s.roll;

cout<<"enter marks:";

cin>>s.marks

cout<<"\ndisplaying information,"<<endl;

cout<<"Name:"<<s.name<<endl;

cout<<"roll no:"<<s.rno<<endl;

cout<<"Marks:"<<s.marks<<endl;

cout<<"Name:"<<s.name<<endl;

cout<<"roll:"<<s.roll<<endl;

cout<<"marks:"<<s.marks<<endl;

return 0;

}
```

OUTPUT:

Enter information,

Enter name: john

Enter roll number: 101

Enter marks: 85.5

Displaying information,

Name: john

Roll: 101

Marks: 85.5

17.PYRAMID PATTERN

AIM:

Create a C++ program to print an inverted pyramid pattern.

ALGORITHM:

STEP 1: Include the necessary header files.

STEP 2: Iterate through the rows starting from the first row (top) and proceed to the last row (bottom).

STEP 3: Before printing stars in each row, print the appropriate number of spaces for alignment.

STEP 4: Adjust the number of spaces and stars for the next row and repeat the process.

STEP 5: Terminate the program.

CODING

```
Using namespace std;

#include<bits/stdc++.h>

#include<iostream>

Int main()

Int n = 4;

For(int I = n; i>=1;--i)

{ { { } }

For(int j=1;j<=1;++i)

Cout<<" * ";

Cout<<endl;

Return 0;
}
```

OUTPUT:

****

***

**

*

18. LEAP YEAR

AIM:

Write a C++ program that checks whether a given year is a leap year using conditional statements.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Input the year.

STEP 3: If the year is divisible by 400, print "Leap year."

STEP 4: Else, if the year is divisible by 100, print "Not a leap year."

STEP 5: Else, if the year is divisible by 4, print "Leap year."

STEP 6: Otherwise, print "Not a leap year."

STEP 7: End the program.

CODING

```
#include <iostream>

using namespace std;

int main()

{

int year;

cout << "Enter a year: ";

cin >> year;

if (year % 400 == 0) {

cout << year << " is a leap year.";

}

else if (year % 100 == 0) {

cout << year << " is not a leap year.";
```

```
}

else if (year % 4 == 0) {

cout << year << " is a leap year.";

}

else {

cout << year << " is not a leap year.";

}

return 0;
}
```

OUTPUT:

Enter a year: 2024

2024 is a leap year.

Enter a year: 1900

1900 is not a leap year.

Enter a year: 2000

2000 is a leap year.

Enter a year: 2023

2023 is not a leap