1. C++ Program to print half pyramid as using \*.

#include <iostream>

using namespace std;

int main()

{

int i,j,rows;

cout<<"\n\*\*\*\*\*\* Program by Himanshu Beniwal \*\*\*\*\*\*"<<endl;

cout<<"\nEnter the number of rows: ";

cin>>rows;

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

{

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

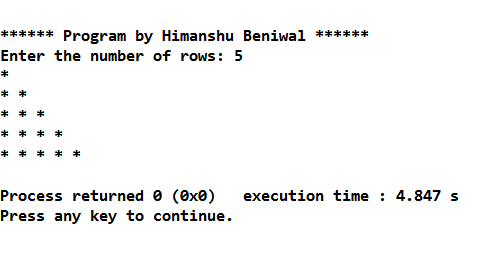
{ cout<<"\* "; }

cout<<"\n";

}

}

Output –



1. C++ Program to print triangle of characters.

#include <iostream>

using namespace std;

int main()

{ int i,j;

char input,temp='A';

cout<<"\n\*\*\*\*\*\* Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"Enter uppercase character you want in triange at last row: ";

cin>>input;

for(i=1;i<=(input-'A'+1);++i)

{

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

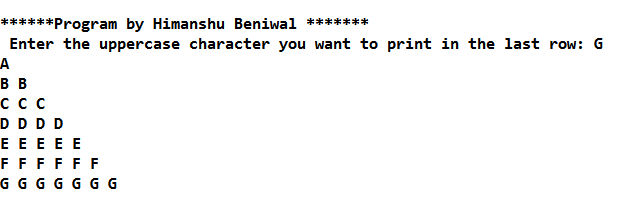
cout<<temp<<" "<<endl;

++temp;

}

}

Output –



1. C++ program to print pyramid using \*.

#include <iostream>

using namespace std;

int main()

{

int i,space,rows,k=0;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"\nEnter the number of rows: ";

cin>>rows;

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

{

for(space=1;space<=rows-i;++space)

{ cout<<" "; }

while(k!=2\*i-1)

{

cout<<"\* ";

++k;

}

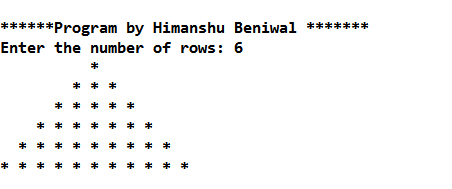
k=0;

cout<<"\n";

}

}

Output –



1. C++ program to print the pyramid of digits in pattern.

#include <iostream>

using namespace std;

int main()

{

int rows, count = 0, count1 = 0, k = 0;

cout<<"\n\*\*\*\*\*\* Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout << "\nEnter number of rows: ";

cin >> rows;

for(int i = 1; i <= rows; ++i)

{

for(int space = 1; space <= rows-i; ++space)

{ cout << " ";

++count;

}

while(k != 2\*i-1)

{ if (count <= rows-1)

{ cout << i+k << " ";

++count;

}

else

{ ++count1;

cout << i+k-2\*count1 << " ";

}

++k;

}

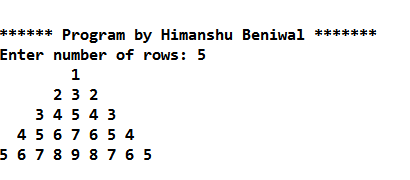
count1 = count = k = 0;

cout << endl;

}

}

Output –



1. C++ Program to Draw Pascal's triangle.

#include <iostream>

int main()

{ int rows, coef = 1;

cout<<"\n\*\*\*\*\*\* Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout << "\nEnter number of rows: ";

cin >> rows;

for(int i = 0; i < rows; i++)

{

for(int space = 1; space <= rows-i; space++)

cout <<" ";

for(int j = 0; j <= i; j++)

{ if (j == 0 || i == 0)

coef = 1;

else

coef = coef\*(i-j+1)/j;

cout << coef << " ";

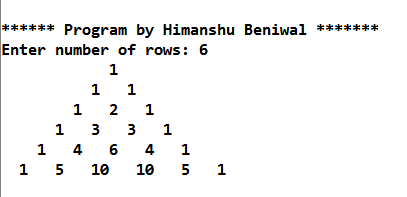
}

cout << endl;

}

}

Output –



1. C++ Program to display Floyd's Triangle.

#include <iostream>

int main()

{ int rows, number = 1;

cout<<"\n\*\*\*\*\*\*\*Project by Himanshu beniwal \*\*\*\*\*\*";

cout << "\nEnter number of rows: ";

cin >> rows;

for(int i = 1; i <= rows; i++)

{ for(int j = 1; j <= i; ++j)

{ cout << number << " ";

++number;

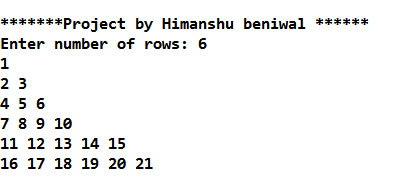
}

cout << endl;

}

}

Output –



1. C++ Program to assign members of a STRUCTURE variable and display it.

#include <iostream>

struct Person

{ char name[50];

int age;

float salary;

};

int main()

{

Person p1;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout << "Enter Full name: ";

cin.get(p1.name, 50);

cout << "Enter age: ";

cin >> p1.age;

cout << "Enter salary: ";

cin >> p1.salary;

cout << "\nDisplaying Information." << endl;

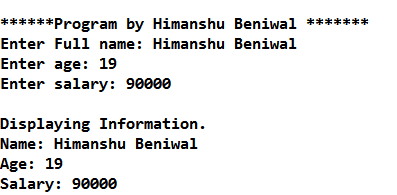
cout << "Name: " << p1.name << endl;

cout <<"Age: " << p1.age << endl;

cout << "Salary: " << p1.salary;

}

Output –



1. Enumeration Type –

#include <iostream>

enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main()

{ week today;

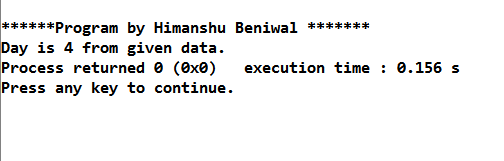
today = Wednesday;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout << "\nDay is " << today+1<<" from given data.";

}

Output –



1. Simple Class Example Program In C++.

#include <iostream>

using namespace std;

class person

{ public:

string name;

int number;

};

int main()

{ person obj;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"Enter the Name :";

cin>>obj.name;

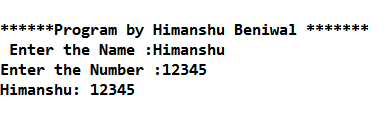
cout<<"Enter the Number :";

cin>>obj.number;

cout << obj.name << ": " << obj.number << endl;

}

Output –



1. Union example in c++.

#include<iostream>

using namespace std;

typedef union myunion

{ double PI;

int B;

}MYUNION;

int main()

{ MYUNION numbers;

numbers.PI = 3.14;

numbers.B = 50;

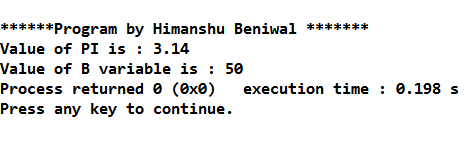
cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"\nValue of PI is : "<<numbers.PI;

cout<<"\nValue of B variable is : "<<numbers.B;

}

Output –



1. Scope resolution operator in c++.

#include <iostream>

using namespace std;

char c = 'a'; // global variable

int main()

{ char c = 'b'; //local variable

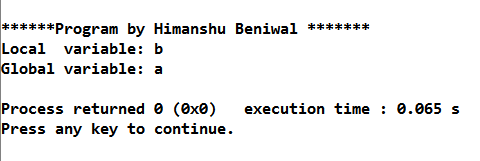
cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*\n";

cout << "Local variable: " << c << "\n";

cout << "Global variable: " << ::c << "\n"; //using scope resolution operator

}

Output –



1. Bitwise Operator in c++.

#include <iostream>

using namespace std;

int main()

{ unsigned int num1 = 12; // 12 = 1100

int num2 = 0;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"\n\n\n-----Now Right (<<) Shift Operator-----";

num2 = num1 >> 2; // 3 = 0011

cout << "Value of num2 is: " << num2 << endl ;

cout<<"\n-----Now Left (>>) Shift Operator-----";

unsigned int num3 = 12; // 12 = 0000 1100

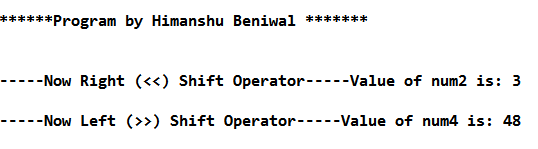
int num4 = 0;

num4 = num3 << 2; // 48 = 0011 0010

cout << "Value of num4 is: " << num4 << endl ;

}

Output –



1. New and Delete Operator in c++.

#include <iostream>

using namespace std;

int main()

{ int n, \*pointer, c;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout << "\nInput an integer\n";

cin >> n;

pointer = new int[n];

cout << "Input " << n << " integers\n";

for (c = 0; c < n; c++)

cin >> pointer[c];

cout << "Elements entered by you are\n";

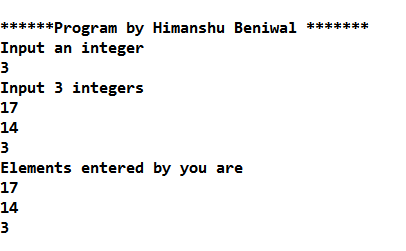
for (c = 0; c < n; c++)

cout << pointer[c] << endl;

delete[] pointer;

}

Output –



1. Setw() function in c++.

#include <iostream>

using namespace std;

#include <iomanip>

int main()

{ int a,b,c;

a=1234;

b=34;

c=133;

cout<<"\n\*\*\*\*\*\*Program by Himanshu Beniwal \*\*\*\*\*\*\*";

cout<<"\n......This is Right Hand Justified.....";

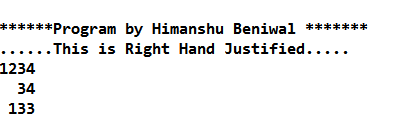
cout<<endl<<setw(4)<<a;

cout<<endl<<setw(4)<<b;

cout<<endl<<setw(4)<<c;

}

Output –



1. Class (public) program in C++.

#include<conio.h>

#include<string.h>

#include<stdio.h>

#include <iostream>

using namespace std;

class student

{ public:

int rollno,er;

char name[20];

void enrol()

{ er=rollno+1000;

// cout<<"the enrollment is"<<er;

}

}s1,s2,s3;

int main()

{ s1.rollno=45;

cout<<"Enter name of student : ";

gets(s1.name);

s1.enrol();

s2.rollno=46;

strcpy(s2.name,"Manish");

s2.enrol();

s3.rollno=47;

strcpy(s3.name,"Ajay");

s3.enrol();

cout<<"\n Roll no of First Student : "<<s1.rollno;

cout<<"\n Name of First Student : " <<s1.name;

cout<<"\n Enrollment number : "<<s1.er;

cout<<"\n Roll no of Second Student : "<<s2.rollno;

cout<<"\n Name of Second Student : " <<s2.name;

cout<<"\n Enrollment number : "<<s2.er;

cout<<"\n Roll no of Third Student : "<<s3.rollno;

cout<<"\n Name of Third Student : " <<s3.name;

cout<<"\n Enrollment number : "<<s3.er;

getch();

return 0;

}Output –

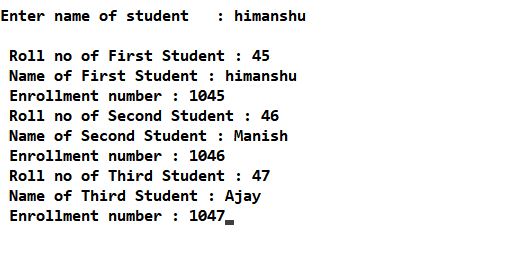


Figure 15: Screenshot of Class (Public) example.

1. Class (private) example in C++

#include <iostream>

using namespace std;

class Box { public:

double length;

void setWidth( double wid );

double getWidth( void );

private:

double width;

};

double Box::getWidth(void) {

return width ;

}

void Box::setWidth( double wid ) {

width = wid;

}

int main( ) {

Box box;

cout<<"\nProgram by Himanshu Beniwal \n";

box.length = 10.0; // OK: because length is public

cout << "Length of box : " << box.length <<endl;

box.setWidth(10.0); // Use member function to set it.

cout << "Width of box : " << box.getWidth() <<endl;

return 0;

}

Output –

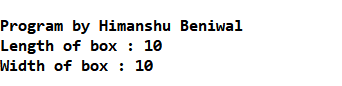


Figure 16: Screenshot of Class (Private) example.

1. Class (protected) example in C++

#include <iostream>

using namespace std;

class Box {

protected:

double width;

};

class SmallBox:Box // SmallBox is the derived class.

{ public:

void setSmallWidth( double wid );

double getSmallWidth( void );

};

double SmallBox::getSmallWidth(void) {

return width ;

}

void SmallBox::setSmallWidth( double wid ) {

width = wid;

}

int main( ) {

SmallBox box;

cout<<"\nProgram by Himanshu beniwal \n";

// set box width using member function

box.setSmallWidth(5.0);

cout << "Width of box : "<< box.getSmallWidth() << endl;

}

Output –

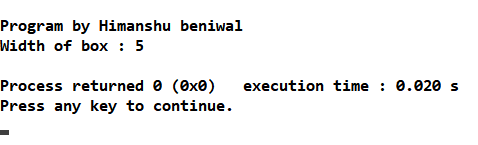


Figure 17: Screenshot of Class (protected) example.

1. Program of Mutli-level inheritance in C++.

#include <iostream>

using namespace std;

class A

{ public:

void display()

{cout<<"Base class content."; }

};class B : public A

{

};

class C : public B

{

};

int main()

{ cout<<"\nProgram by Himanshu beniwal \n";

C obj;

obj.display();

return 0;

}

Output –

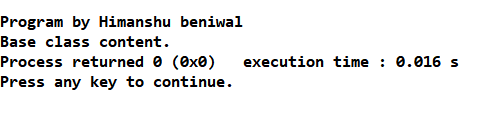


Figure 18: Screenshot of Multi-level inheritance.

1. Program of Multiple Inheritance in C++.

#include <iostream>

using namespace std;

class Mammal {

public:

Mammal()

{cout << "Mammals can give direct birth." << endl; }

};

class WingedAnimal { public:

WingedAnimal()

{ cout << "Winged animal can flap." << endl; }

};

class Bat: public Mammal, public WingedAnimal

{

};

int main()

{ cout<<"\nProgram by Himanshu beniwal \n";

Bat b1;

return 0;

}

Output –

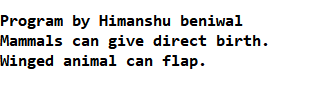


Figure 19: Screenshot of Multiple inheritance.

1. Program for Hierarchical Inheritance in C++.

#include <iostream>

#include <string.h>

using namespace std;

class member {

char gender[10];

int age;

public:

void get()

{

cout << "Age: "; cin >> age;

cout << "Gender: "; cin >> gender;

}

void disp()

{

cout << "Age: " << age << endl;

cout << "Gender: " << gender << endl;

}

};

class stud : public member {

char level[20];

public:

void getdata()

{

member::get();

cout << "Class: "; cin >> level;

}

void disp2()

{

member::disp();

cout << "Level: " << level << endl;

}

};

class staff : public member {

float salary;

public:

void getdata()

{

member::get();

cout << "Salary: Rs."; cin >> salary;

}

void disp3()

{

member::disp();

cout << "Salary: Rs." << salary << endl;

}

};

int main()

{ cout<<"\nProgram by Himanshu beniwal \n";

member M;

staff S;

stud s;

cout << "Student" << endl;

cout << "Enter data" << endl;

s.getdata();

cout << endl

<< "Displaying data" << endl;

s.disp();

cout << endl

<< "Staff Data" << endl;

cout << "Enter data" << endl;

S.getdata();

cout << endl

<< "Displaying data" << endl;

S.disp();}

Output –

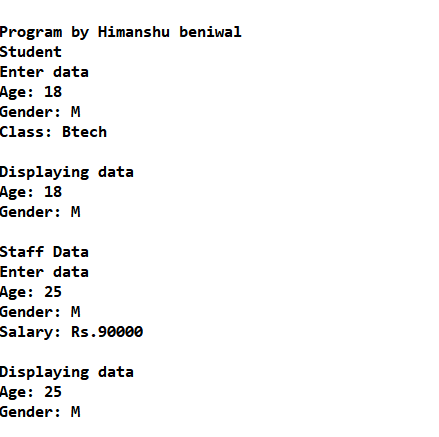


Figure 20: Screenshot of Hierarchical inheritance.

1. Program for Single-inheritance in C++.

#include <iostream>

#include<conio.h>

#include <string.h>

#include<stdio.h>

using namespace std;

class person /\*Parent class\*/

{

private:

char fname[100],lname[100],gender[10];

protected:

int age;

public:

void input\_person();

void display\_person();

};

class student: public person /\*Child class\*/

{

private:

char college\_name[100];

char level[20];

public:

void input\_student();

void display\_student();

};

void person::input\_person()

{

cout<<"First Name: ";

cin>>fname;

cout<<"Last Name: ";

cin>>lname;

cout<<"Gender: ";

cin>>gender;

cout<<"Age: ";

cin>>age;

}

void person::display\_person()

{

cout<<"First Name : "<<fname<<endl;

cout<<"Last Name : "<<lname<<endl;

cout<<"Gender : "<<gender<<endl;

cout<<"Age : "<<age<<endl;

}

void student::input\_student()

{

person::input\_person();

cout<<"College: ";

fflush(stdin);

gets(college\_name);

cout<<"Level: ";

cin>>level;

}

void student::display\_student()

{

person::display\_person();

cout<<"College : "<<college\_name<<endl;

cout<<"Level : "<<level<<endl;

}

int main()

{ cout<<"\nProgram by Himanshu Beniwal.\n";

student s;

cout<<"Input data"<<endl;

s.input\_student();

cout<<endl<<"Display data"<<endl;

s.display\_student();

getch();

return 0;

}

Output –

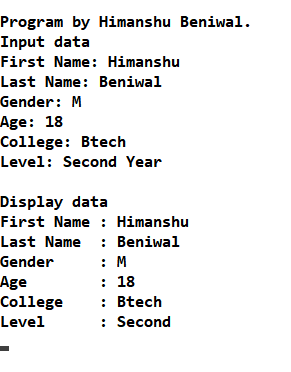


Figure 21: Screenshot of Single inheritance.

1. Hybrid Inheritance in C++

#include <iostream>

#include<conio.h>

int a,b,c,d,e;

using namespace std;

class A {

protected:

public:

void getab() {

cout<<"\n ENter a and b value:";

cin>>a>>b; }};

class B:public A {

protected:

public:

void getc() {

cout<<"Enter c value:";

cin>>c; }};

class C {

protected:

public:

void getd() {

cout<<"Enter d value:";

cin>>d; }};

class D:public B,public C {

protected:

public:

void result() {

getab(); getc();

getd(); e=a+b+c+d;

cout<<"\n Addition is :"<<e; }};

int main() {

cout<<"\nProgram by Himanshu Beniwal\n";

D d1;

d1.result();

return 0; }

Output –

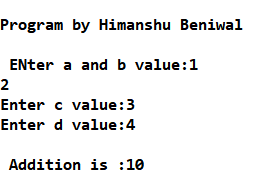


Figure 22: Screenshot of Hybrid inheritance.

1. Virtual Function

#include <iostream>

using namespace std;

class b

{

public:

virtual void show()

{

cout<<"\n Showing base class....";

}

void display()

{

cout<<"\n Displaying base class...." ;

}

};

class d:public b

{

public:

void display()

{

cout<<"\n Displaying derived class....";

}

void show()

{

cout<<"\n Showing derived class....";

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal\n";

b B;

b \*ptr;

cout<<"\n\t P points to base:\n" ;

ptr=&B;

ptr->display();

ptr->show();

cout<<"\n\n\t P points to drive:\n";

d D;

ptr=&D;

ptr->display();

ptr->show();

}

Output –

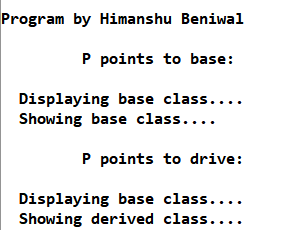


Figure 23: Screenshot of Virtual Function.

1. Friend Function

#include <iostream>

using namespace std;

#include<windows.h>

class Distance {

private:

int meter;

public:

Distance(): meter(0){ }

friend int func(Distance); //friend function

};

int func(Distance d){

//function definition

d.meter=10; //accessing private data from non-member function

return d.meter;

}

int main(){

Distance D;

cout<<"\nProgram by Himanshu beniwal \n";

cout<<"Distace: "<<func(D)<<endl;

system("pause");

return 0;

}

Output –

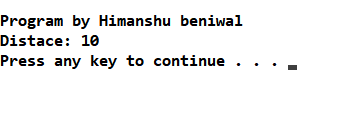


Figure 24: Screenshot of Friend Function.

1. Constructor

#include <iostream>

using namespace std;

class Calc {

public:

int val;

Calc()

{

val = 20;

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal\n";

Calc c1;

cout << c1.val;

}

Output –

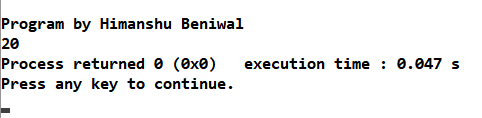


Figure 25: Screenshot of Constructor.

1. Parameterized Constructor

#include <iostream>

using namespace std;

class Calc

{

public:

int val2;

Calc(int x)

{

val2=x;

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

Calc c1(10);

Calc c2(20);

Calc c3(30);

cout << c1.val2<<endl;

cout << c2.val2<<endl;

cout << c3.val2<<endl;

}

Output –

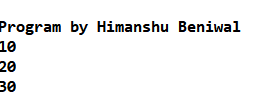


Figure 26: Screenshot of Parametrized Constructor.

1. Copy Constructor

#include <iostream>

using namespace std;

class CopyCon {

int a, b;

public:

CopyCon(int x, int y)

{

a = x;

b = y;

cout << "\nHere is the initialization of Constructor";

}

void Display()

{

cout << "\nValues : \t" << a << "\t" << b;

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

CopyCon Object(30, 40);

//Copy Constructor

CopyCon Object2 = Object;

Object.Display();

Object2.Display();

return 0;

}

Output –

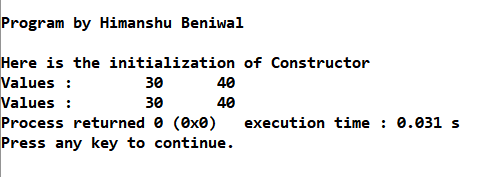


Figure 27: Screenshot of Copy- Constructor.

1. Operator Overloading

#include <iostream>

using namespace std;

class MinusOverload {

private:

int a;

int b;

public:

void Distance()

{

a = 0;

b = 0;

}

MinusOverload(int f, int i)

{

int c;

a = f;

b = i;

c = a - b;

cout << "\nC:" << c;

}

void display()

{

cout << "A: " << a << " B:" << b << endl;

}

MinusOverload operator-()

{

a = -a;

b = -b;

return MinusOverload(a, b);

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

MinusOverload M1(6, 8), M2(-3, -4);

-M1;

M1.display();

cout<<endl;

-M2;

M2.display();

return 0;

}

Output –

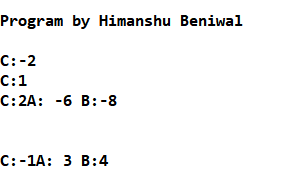


Figure 28: Screenshot of Operator Overloading.

1. Function Overloading

#include <iostream>

using namespace std;

long add(long, long);

float add(float, float);

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

long a, b, c;

float e, f, g;

cout << "Enter two integers\n";

cin >> a >> b;

c = add(a, b);

cout << "Sum of integers: " << c << endl;

cout << "Enter two floating point numbers\n";

cin >> e >> f;

g = add(e, f);

cout << "Sum of floats: " << g << endl;

}

long add(long c, long g)

{

long sum;

sum = c + g;

return sum;

}

float add(float c, float g)

{

float sum;

sum = c + g;

return sum;

}

Output –

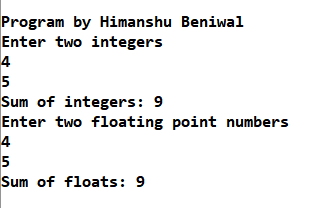


Figure 29: Screenshot of Function Overloading.

1. Pointers

#include <iostream>

using namespace std;

int main() {

int \*pc, c;

cout<<"\nProgram by Himanshu Beniwal \n";

c = 5;

cout << "Address of c (&c): " << &c << endl;

cout << "Value of c (c): " << c << endl << endl;

pc = &c; // Pointer pc holds the memory address of variable c

cout << "Address that pointer pc holds (pc): "<< pc << endl;

cout << "Content of the address pointer pc holds (\*pc): " << \*pc << endl << endl;

c = 11; // The content inside memory address &c is changed from 5 to 11.

cout << "Address pointer pc holds (pc): " << pc << endl;

cout << "Content of the address pointer pc holds (\*pc): " << \*pc << endl << endl;

\*pc = 2;

cout << "Address of c (&c): " << &c << endl;

cout << "Value of c (c): " << c << endl << endl;

return 0;

}

Output –

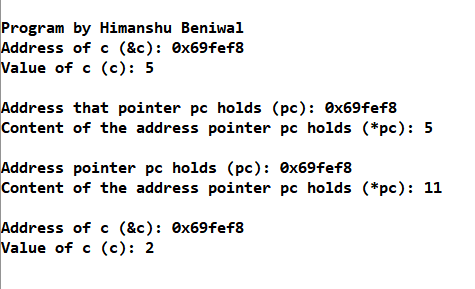


Figure 30: Screenshot of Pointers.

1. Virtual Base Class.

#include <iostream>

using namespace std;

#include<conio.h>

class A

{

public:

int i;

};

class B : virtual public A

{

public:

int j;

};

class C: virtual public A

{

public:

int k;

};

class D: public B, public C

{

public:

int sum;

};

int main()

{ cout<<"\nProgram by himanshu Beniwal \n";

D ob;

ob.i = 10; //unambiguous since only one copy of i is inherited.

ob.j = 20;

ob.k = 30;

ob.sum = ob.i + ob.j + ob.k;

cout << "Value of i is : "<< ob.i<<"\n";

cout << "Value of j is : "<< ob.j<<"\n";

cout << "Value of k is :"<< ob.k<<"\n";

cout << "Sum is : "<< ob.sum <<"\n";

return 0;

}

Output –

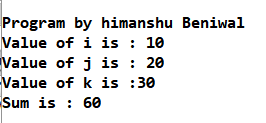


Figure 31: Screenshot of Virtual Base Class.

1. Constructor Overloading

#include <iostream>

using namespace std;

class Student

{

int rollno;

string name;

public:

Student(int x)

{

rollno=x;

name="None";

}

Student(int x, string str)

{

rollno=x ;

name=str ;

}

void display()

{

cout<<"\n Roll no : "<<rollno;

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

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

Student A(10);

A.display();

Student B(11,"Ram");

B.display();

}

Output –

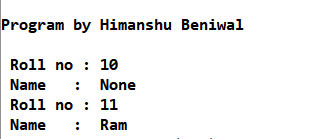


Figure 32: Screenshot of Constructor Overloading.

1. Destructor

#include <iostream>

using namespace std;

class A

{

public:

A()

{

cout << "Constructor called\n";

}

~A()

{

cout << "Destructor called\n";

}

};

int main()

{ cout<<"\nPRogram by Himanshu Beniwal \n";

A obj1; // Constructor Called

int x=1;

if(x)

{

A obj2; // Constructor Called

} // Destructor Called for obj2

} // Destructor called for obj1

Output-

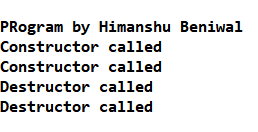


Figure 31: Screenshot of Destructor

1. Inline Function

#include <iostream>

using namespace std;

inline int cube(int s)

{

return s\*s\*s;

}

int main()

{ cout<<"\nPRogram by Himanshu Beniwal \n";

cout << "The cube of 3 is: " << cube(3) << "\n";

return 0;

}

Output –

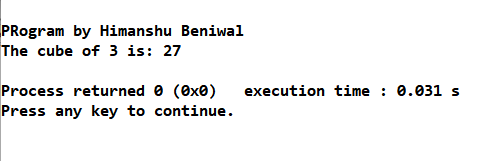


Figure 34: Screenshot of Inline Function.

1. ‘This’ Pointer

#include<iostream>

using namespace std;

class Test

{

private:

int x;

int y;

public:

Test(int x = 0, int y = 0) { this->x = x; this->y = y; }

Test &setX(int a)

{ x = a;

return \*this;

}

Test &setY(int b)

{ y = b;

return \*this;

}

void print() { cout << "x = " << x << " y = " << y << endl; }

};

int main()

{ cout<<"\n Program by Himanshu Beniwal \n";

Test obj1(5, 5);

obj1.setX(10).setY(20);

obj1.print();

return 0;

}

Output –

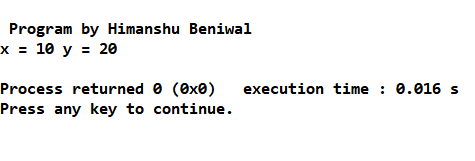


Figure 35: Screenshot of ‘This’ Pointer.

1. Polymorphism in function

#include <iostream>

#include<conio.h>

using namespace std;

class Base

{

public:

void show()

{

cout<<"Base class\n";

}

};

class Derived:public Base

{

public:

void show()

{

cout<<"Derived Class\n";

}

};

int main()

{ cout<<"\nPRogram by Himanshu Beniwal \n";

Base b; //Base class object

Derived d; //Derived class object

b.show(); //Early Binding Ocuurs

d.show();

}

Output –

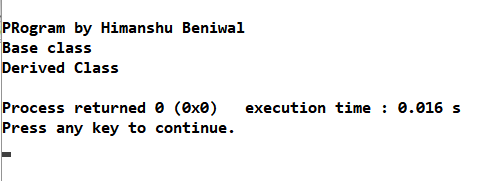


Figure 36: Screenshot of Polymorphism.

1. Abstract Class and Virtual Function.

#include <iostream>

using namespace std;

// Abstract class

class Shape

{

protected:

float l;

public:

void getData()

{

cin >> l;

}

// virtual Function

virtual float calculateArea() = 0;

};

class Square : public Shape

{

public:

float calculateArea()

{ return l\*l; }

};

class Circle : public Shape

{

public:

float calculateArea()

{ return 3.14\*l\*l; }

};

int main()

{

Square s;

Circle c;

cout<<"\nPRogram by Himanshu Beniwal \n";

cout << "Enter length to calculate the area of a square: \n";

s.getData();

cout<<"Area of square: " << s.calculateArea()<<endl;

cout<<"\nEnter radius to calculate the area of a circle: \n";

c.getData();

cout << "Area of circle: \n" << c.calculateArea()<<endl;

return 0;

}

Output –

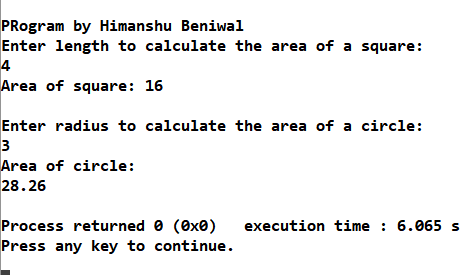


Figure 37: Screenshot of Abstract Class and Virtual Function.

1. Array of Objects.

#include <iostream>

#include<conio.h>

using namespace std;

class rec

{

private:

int I;

int b;

public:

rec(int a,int c)

{

I=a;

b=c;

}

void put()

{

cout<<"Area is : "<<I\*b <<endl;

}

};

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

rec obj[3]={rec(3,6),rec(2,5),rec(5,5)};

cout<<"Displaying Areas of Rectangles : \n";

for(int i=0;i<3;i++)

{

obj[i].put();

}

return 0;

}

Output –

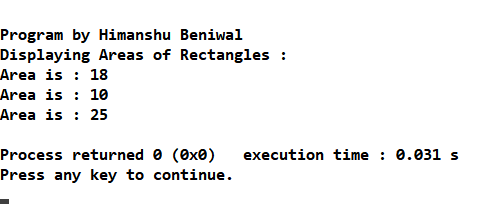


Figure 38: Screenshot of Array of Objects.

1. Constructor Defined Outside Of Class

#include <iostream>

#include <iomanip>

#include <string.h>

using namespace std;

class Book

{ public:

char title[256];

char author[64];

float price;

Book(char \*btitle, char \*bauthor, char \*bpublisher, float bprice);

void show\_title(void) { cout << title << '\n'; };

float get\_price(void) { return(price); };

void show\_book(void)

{

show\_title();

show\_publisher();

};

void assign\_publisher(char \*name) { strcpy(publisher, name); };

private:

char publisher[256];

void show\_publisher(void) { cout << publisher << '\n'; };

};

Book::Book(char \*btitle, char \*bauthor, char \*bpublisher, float bprice)

{

strcpy(title, btitle);

strcpy(author, bauthor);

strcpy(publisher, bpublisher);

price = bprice;

}

int main(void)

{ cout<<"\nProgram by Himanshu Beniwal \n";

Book tips("A", "B", "C", 49.95);

Book diary("D", "E", "F", 9.95);

tips.show\_book();

diary.show\_book();

}

Output –

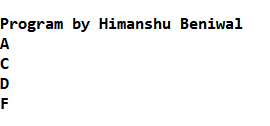


Figure 39: Screenshot of Constructor Defined Outside The Class.

1. Exception Handling in C++.

#include <iostream>

using namespace std;

int main()

{ cout<<"\n\*\*\*\* Program by Himanshu Beniwal\*\*\*\* \n";

int x = -1;

// Some code

cout << "Before try \n";

try {

cout << "Inside try \n";

if (x < 0)

{

throw x;

cout << "After throw (Never executed) \n";

}

}

catch (int x ) {

cout << "Exception Caught \n";

}

cout << "After catch (Will be executed) \n";

return 0;

}

Output –

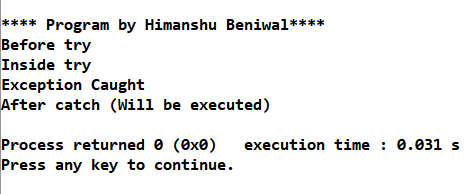


Figure 40: Screenshot of Exception Handling.

1. Read-Write to any File.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{ cout<<"\nProgram by Himanshu Beniwal \n";

fstream file; //object of fstream class

//opening file "sample.txt" in out(write) mode

file.open("sample.txt",ios::out);

if(!file)

{

cout<<"Error in creating file!!!"<<endl;

return 0;

}

cout<<"File created successfully."<<endl;

//write text into file

file<<"ABCD.";

//closing the file

file.close();

//again open file in read mode

file.open("sample.txt",ios::in);

if(!file)

{

cout<<"Error in opening file!!!"<<endl;

return 0;

}

//read untill end of file is not found.

char ch; //to read single character

cout<<"File content: ";

while(!file.eof())

{

file>>ch; //read single character from file

cout<<ch;

}

file.close(); //close file

return 0;

}

Output –

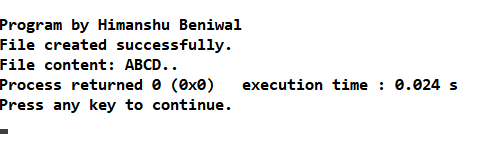


Figure 41: Screenshot of Read-Writing to Any file.

1. Read Text From any File.

#include<iostream>

#include<fstream>

using namespace std;

int main()

{ cout<<"\nProgram by himanshu beniwal \n";

char ch;

const char \*fileName="sample.txt";

//declare object

ifstream file;

//open file

file.open(fileName,ios::in);

if(!file)

{

cout<<"Error in opening file!!!"<<endl;

return -1; //return from main

}

//read and print file content

while (!file.eof())

{

file >> noskipws >> ch; //reading from file

cout << ch; //printing

}

//close the file

file.close();

return 0;

}

Output –

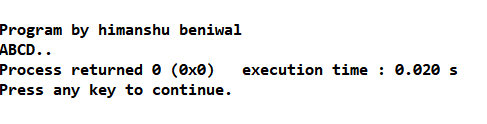


Figure 42: Screenshot of Reading to Any file.