**All C++ Programs**

**//program1**

#include <iostream>

using namespace std;

class Item

{

private:

int number;

float cost;

public:

void setdata(int id, float price)/\*defining inside class\*/

{

number = id;

cost = price;

}

void showdata();//prototype or declaration only

};

void Item::showdata()

{

cout<<"Number: "<<number<<endl;

cout<<"Price: "<<cost<<endl;

}

int main()

{

Item x; //creating object x

cout<<"Object X"<<endl;

x.setdata(100, 10.5);

x.showdata();

Item y;

cout<<"Object Y"<<endl;

y.setdata(50, 5.25);

y.showdata();

return 0;

}

OUTPUT:

Object X

Number: 100

Price: 10.5

Object Y

Number: 50

Price: 5.25

**//program2**

//array as data member

#include <iostream>

using namespace std;

class Student

{

private:

char name[20];

int roll;

float score[5];

public:

void setdata();

void showdata();

};

void Student::setdata()

{

cout << "Enter Data for Student"<<endl;

cout << "Name: ";

cin >>name;

cout << "Roll: ";

cin >>roll;

for (int j = 0; j < 5; ++j)

{

cout << "Marks in subject " << j + 1 << ": ";

cin >> score[j];

}

}

void Student::showdata()

{

cout<<"Details Info:\n";

cout<<"Name: "<<name;

cout<<"\nRoll Number: "<<roll;

cout<<"\nMarks in 5 Subjects: ";

for(int j = 0; j < 5; j++)

cout<<score[j]<<" ";

}

int main()

{

Student s;

s.setdata();

s.showdata();

return 0;

}

OUTPUT:

Enter Data for Student

Name: Ashish

Roll: 3

Marks in subject 1: 45

Marks in subject 2: 67

Marks in subject 3: 89.65

Marks in subject 4: 70

Marks in subject 5: 55

Details Info:

Name: Ashish

Roll Number: 3

Marks in 5 Subjects: 45 67 89.65 70 55

**//program3**

//addition of two complex numbers

#include <iostream>

using namespace std;

class Complex

{

private:

float real;

float imag;

public:

void setdata()

{

cout << "Real Part: ";

cin >>real;

cout << "Imaginary Part: ";

cin >>imag;

}

void showdata()

{

cout<<"("<<real<<","<<imag<<")";

}

void add(Complex, Complex);

};

void Complex::add(Complex c1, Complex c2)

{

real = c1.real + c2.real;

imag = c1.imag + c2.imag;

}

int main()

{

Complex com1, com2, com3;

cout<<"Enter First Complex number:\n";

com1.setdata();

cout<<"\nEnter Second Complex number:\n";

com2.setdata();

com3.add(com1, com2);

com1.showdata();

cout<<" + ";

com2.showdata();

cout<<" = ";

com3.showdata();

return 0;

}

OUTPUT

Enter First Complex number:

Real Part: 5

Imaginary Part: -2.5

Enter Second Complex number:

Real Part: -3

Imaginary Part: 7.75

(5,-2.5) + (-3,7.75) = (2,5.25)

**//program4**

//Return object from function

#include <iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

void setdata(float r, float i)

{

real = r;

imag = i;

}

void showdata()

{

cout<<"("<<real<<" , "<<imag<<")";

}

complex add(complex);

};//end of class

complex complex::add(complex c)

{

complex temp;

temp.real = real + c.real;

temp.imag = imag + c.imag;

return temp;

}

int main()

{

complex c1, c2, c3;

c1.setdata(3, -1);

c2.setdata(-2, 8);

c3 = c1.add(c2); //c3 = c2.add(c1);

c1.showdata();

cout<<" + ";

c2.showdata();

cout<<" = ";

c3.showdata();

return 0;

}

OUTPUT:

(3 , -1) + (-2 , 8) = (1 , 7)

**//program5**

#include <iostream>

using namespace std;

class distances

{

private:

int feet;

float inch;

public:

void setdist(int, float);

void getdist();

void showdist();

void add(distances, distances);

};

void distances::setdist(int ft, float in)

{

feet = ft;

inch = in;

}

void distances::getdist()

{

cout<<"Enter feet part: ";

cin>>feet;

cout<<"\nEnter inch part: ";

cin>>inch;

}

void distances::showdist()

{

cout<<feet<<"ft "<<inch<<"'";

}

void distances::add(distances d1, distances d2)

{

feet = d1.feet + d2.feet;

inch = d1.inch + d2.inch;

while (inch >= 12)

{

++feet;

inch -= 12;

}

}

int main()

{

distances d1, d2, d3;

d1.setdist(12, 3.5);

d2.getdist();

d3.add(d1, d2);

d1.showdist();

cout<<" + ";

d2.showdist();

cout<<" = ";

d3.showdist();

return 0;

}

OUTPUT:

Enter feet part: 8

Enter inch part: 18

12ft 3.5' + 8ft 18' = 21ft 9.5'

**//program6**

//friend function for addition of two complex numbers

#include <iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

void setdata(float r, float i)

{

real = r;

imag = i;

}

void showdata()

{

cout<<"("<<real<<" , "<<imag<<")";

}

friend complex add(complex, complex);

};//end of class

complex add(complex c1, complex c2)

{

complex temp;

temp.real = c1.real + c2.real;

temp.imag = c1.imag + c2.imag;

return temp;

//cout << "(" <<temp.real<< " , " <<temp.imag<< ")";

}

int main()

{

complex c1, c2, c3;

c1.setdata(3, -1);

c2.setdata(-2, 8);

c1.showdata();

cout<<" + ";

c2.showdata();

cout<<" = ";

c3 = add(c1, c2);

c3.showdata();

return 0;

}

**OUTPUT**

(3 , -1) + (-2 , 8) = (1 , 7)

**//program7**

//addition of data members of two different classes

#include <iostream>

using namespace std;

class A; //forward declaration

class B

{

private:

int data;

public:

void setdata(int x)

{

data = x;

}

friend void add(A &, B &);

};

class A

{

private:

int data;

public:

void setdata(int x)

{

data = x;

}

friend void add(A &, B &);

};

void add(A &a, B &b)

{

cout << "Sum: " <<a.data + b.data<<endl;

//return (a.data + b.data);

}

int main()

{

Aobj\_a;

B obj\_b;

obj\_a.setdata(10);

obj\_b.setdata(20);

add(obj\_a, obj\_b);//calling friend function

return 0;

}

OUTPUT:

Sum: 30

**//program8**

//friend function: swapping private data of two classes

#include <iostream>

using namespace std;

class ABC;

class XYZ

{

private:

int value;

public:

void setdata(int);

void showdata();

friend void swap(ABC &, XYZ &);

};

class ABC

{

private:

int value;

public:

void setdata(int);

void showdata();

friend void swap(ABC &, XYZ &);

};

void XYZ::setdata(int n)

{

value = n;

}

void XYZ::showdata()

{

cout << value <<endl;

}

void ABC::setdata(int n)

{

value = n;

}

void ABC::showdata()

{

cout << value <<endl;

}

void swap(ABC &abc, XYZ &xyz)

{

int temp = abc.value;

abc.value = xyz.value;

xyz.value = temp;

}

int main()

{

ABC a;

XYZ x;

a.setdata(10);

x.setdata(20);

cout<<"\nValues before swapping\nABC value= ";

a.showdata();

cout<<"XYZ value= ";

x.showdata();

swap(a, x); //friend function

cout<<"\nValues after swapping\nABC value= ";

a.showdata();

cout<<"XYZ value= ";

x.showdata();

return 0;

}

OUTPUT:

Values before swapping

ABC value= 10

XYZ value= 20

Values after swapping

ABC value= 20

XYZ value= 10

**//program9**

//static data and function example

#include <iostream>

using namespace std;

class Part

{

private:

int pid;

static int count;/\*static data\*/

public:

void setdata()

{

pid = ++ count;

}

static int getcount()/\*static function\*/

{

return count;

}

void showid()

{

cout << "ID: " <<pid<<endl;

}

};

int Part::count; /\*int Part::count = 0;\*/

int main()

{

/\*cout<<"\nCount is: "<<Part::getcount()<<endl;/\*can be called before object creation\*/

Part p1, p2;/\*initialized to zero for the first object creation\*/

//cout<<"\nCount is: "<<Part::getcount()<<endl;

p1.setdata();

p2.setdata();

p1.showid();

p2.showid();

cout<<"\nCount is: "<<Part::getcount()<<endl;

Part p3;

p3.setdata();

p3.showid();

cout<<"\nCount is: "<<Part::getcount()<<endl;

return 0;

}

OUTPUT:

ID: 1

ID: 2

Count is: 2

ID: 3

Count is: 3

**//program10**

//inline function

#include <iostream>

using namespace std;

class Distance

{

private:

float inch;

float feet;

public:

void setFeet(float f)

{

feet = f;

}

inline void convert()

{

inch = 12 \* feet;

}

void showdata()

{

cout<<"Feet: "<<feet<<endl

<<"Inch: "<<inch<<endl;

}

};

int main()

{

Distance len;

len.setFeet(10);

len.convert();

len.showdata();

return 0;

}

OUTPUT:

Feet: 10

Inch: 120

**//program11**

/\*Using a class write a program that receives principal amount, time, and rate. Keeping rate 8% as the default argument, calculate the simple interest for three customers\*/

#include <iostream>

using namespace std;

class Account

{

private:

char name[30];

float pa;

float rate;

float time;

public:

void getdata();

float calc\_si(float r = 8);

};

void Account::getdata()

{

cout << "Name: ";

cin >>name;

cout << "Principal Amount: ";

cin >>pa;

cout << "Time in Year: ";

cin >>time;

}

float Account::calc\_si(float r)

{

rate = r;

return (pa \* rate \* time / 100);

}

int main()

{

Account a1, a2, a3;

float r;

a1.getdata();

cout << "Rate: "; cin >>r;

cout << "Simple Interest: " << a1.calc\_si(r)

<< endl;//rate = 10%

a2.getdata();

cout << "Rate: "; cin >>r;

cout << "Simple Interest: " << a2.calc\_si(r)

<< endl; //rate = 12%

a3.getdata();

cout << "Simple Interest: " << a3.calc\_si()<<endl; //rate = 8%

return 0;

}

OUTPUT:

Name: Rupesh

Principal Amount: 50000

Time in Year: 3

Rate: 14

Simple Interest: 21000

Name: Raju

Principal Amount: 100000

Time in Year: 2

Rate: 12

Simple Interest: 24000

Name: Unika

Principal Amount: 2400

Time in Year: 5

Simple Interest: 960

**//program12**

//reference variable

#include <iostream>

using namespace std;

int squareByValue(int); /\*function prototype (pass-by-value)\*/

void squareByReference(int &); /\*function prototype (pass-by-reference)\*/

int main()

{

int x = 2;

int z = 3;

cout << "x = " << x << " before squareByValue."

<<endl;

cout <<squareByValue(x) <<endl;

cout << "After squareByValue x = " << x <<endl;

cout << "z = " << z << " before squareByReference\n";

squareByReference(z);

cout << "After squareByReference z = " << z <<endl;

return 0;

}

int squareByValue(int n)

{

n \*= n;

return n;

}

void squareByReference(int &nRef)

{

nRef \*= nRef; //caller’s argument modified

}

OUTPUT:

x = 2 before squareByValue.

4

After squareByValue x = 2

z = 3 before squareByReference

After squareByReference z = 9

**//program13**

//reference variable

#include <iostream>

using namespace std;

int main()

{

int x = 3;

/\* int &y; is an error because reference must be initialized\*/

int &y = x; // y refers to x (is an alias for) x

cout << "x = " << x <<endl;

cout << "y = " << y <<endl;

y = 7; //actually modifies x

cout << "x = " << x <<endl;

cout << "y = " << y <<endl;

return 0;

}

OUTPUT:

x = 3

y = 3

x = 7

y = 7

**//program14**

#include<iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

complex() //default constructor

{

real = 0;

imag = 0;

}

void showdata()

{

cout << "(" << real << " , " <<imag

<< ")" <<endl;

}

};

int main()

{

complex c1; /\*default constructor is automatically called here\*/

c1.showdata();

return 0;

}

OUTPUT:

(0 , 0)

**//program15**

//parameterized constructors

#include <iostream>

using namespace std;

class Point

{

private:

int x;

int y;

public:

Point(int a, int b) //parameterized constructor

{

x = a;

y = b;

}

void showdata()

{

cout << "(" << x << "," << y << ")"

<<endl;

}

};

int main()

{

Point p1(3, 5); /\* called parameterized constructor implicitly\*/

p1.showdata();

Point p2 = Point(10, 7);/\* called parameterized constructor explicitly\*/

p2.showdata();

return 0;

}

OUTPUT:

(3,5)

(10,7)

**//program16**

//overloaded or multiple constructors

#include <iostream>

using namespace std;

class Time

{

private:

int hh;

int mm;

int ss;

public:

Time(){} /\*do nothing. creating object without initial value\*/

/\*Time(int h, int m, int s)

{

hh = h;

mm = m;

ss = s;

}\*/

/\*member initializer list\*/

Time(int h, int m, int s):hh(h), mm(m), ss(s){}

/\*Time(Time &t)

{

hh = t.hh;

mm = t.mm;

ss = t.ss;

}\*/

/\*member initializer list\*/

Time(Time &t):hh(t.hh), mm(t.mm), ss(t.ss){}

void showdata()

{

cout<<hh<<" : "<<mm<<" : "<<ss;

}

Time add(Time);

};

Time Time::add(Time t)

{

Time result;

result.ss = ss + t.ss;

result.mm = mm + t.mm;

result.hh = hh + t.hh;

while (result.ss>= 60)

{

++result.mm;

result.ss -= 60;

}

while (result.mm >= 60)

{

++result.hh;

result.mm -= 60;

}

return result;

}

int main()

{

Time t3; //called default constructor

Time t1(1, 50, 45); /\*called parameterized constructor\*/

Time t2(t1); /\*called copy constructor\*/

t1.showdata();

cout<<" + ";

t2.showdata();

cout<<" = ";

t3 = t2.add(t1);

t3.showdata();

return 0;

}

**//program17**

//Program to illustrate constructor and destructor

#include <iostream>

using namespace std;

class Test

{

private:

static int count;

public:

Test() //default constructor

{

++count;

cout << "Constructor is calling for object: " << count <<endl;

}

~Test() //destructor

{

cout << "Destructor is calling for Object: " << count <<endl;

--count;

}

};

int Test::count = 0;

int main()

{

Test t1;

Test t2;

Test t3;

Test t4;

cout<<"\n----------------------------------\n\n";

return 0;

}

**//program18**

#include <iostream >

using namespace std;

int main()

{

double\* pvalue = NULL; /\* Pointer initialized with null\*/

pvalue = new double; /\* Request memory for the variable\*/

\*pvalue = 29494.99; /\* Store value at allocated address\*/

cout << "Value of pvalue : " << \*pvalue<<endl;

delete pvalue; // free up the memory.

return 0;

}

**//program19**

#include <iostream>

using namespace std;

class Box

{

public:

Box() //default constructor

{

cout << "Constructor called!" <<endl;

}

~Box()//destructor

{

cout << "Destructor called!" <<endl;

}

};

int main()

{

Box\* myBoxArray = NULL;

myBoxArray = new Box[4];

cout<<"-------------------\n";

delete [] myBoxArray; // Delete array

return 0;

}

**//program20**

//distance with return object

#include <iostream>

using namespace std;

class distances

{

private:

int feet;

float inch;

public:

void setdist(int, float);

void getdist();

void showdist();

distances add(distances);

};

void distances::setdist(int ft, float in)

{

feet = ft;

inch = in;

}

void distances::getdist()

{

cout<<"Enter feet part: ";

cin>>feet;

cout<<"\nEnter inch part: ";

cin>>inch;

}

void distances::showdist()

{

cout<<feet<<"ft "<<inch<<"'";

}

distances distances::add(distances d)

{

distances temp;

temp.feet = d.feet + feet;

temp.inch = d.inch + inch;

if(temp.inch>= 12)

{

temp.feet++;

temp.inch -= 12;

}

return temp;

}

int main()

{

distances d1, d2, d3;

d1.setdist(12, 3);

d2.getdist();

d3 = d2.add(d1);

d1.showdist();

cout<<" + ";

d2.showdist();

cout<<" = ";

d3.showdist();

return 0;

}

**//program21**

//Addition of Two Times with returning object

#include <iostream>

using namespace std;

class Time

{

private:

int hh;

int mm;

int ss;

public:

void setTime();

void showTime();

Time addTime(Time);

};

void Time::setTime()

{

cout << "Hour: "; cin >>hh;

cout << "Minute: "; cin >>mm;

cout << "Second: "; cin >>ss;

}

void Time::showTime()

{

cout <<hh<< " : " << mm << " : " <<ss;

}

Time Time::addTime(Time t)

{

Time temp;

temp.ss = t.ss + ss;

temp.mm = t.mm + mm;

temp.hh = t.hh + hh;

while (temp.ss>= 60)

{

++temp.mm;

temp.ss -= 60;

}

while (temp.mm >= 60)

{

++temp.hh;

temp.mm -= 60;

}

return temp;

}

int main()

{

Time t1, t2, t3;

cout << "First Time" <<endl;

t1.setTime();

cout << "Second Time" <<endl;

t2.setTime();

t3 = t1.addTime(t2); //t3 = t2.addTime(t1);

//t3.addTime(t1, t2);

t1.showTime();

cout << " + ";

t2.showTime();

cout << " = ";

t3.showTime();

return 0;

}

**//program22**

//friend class

#include <iostream>

using namespace std;

//forward declaration

class ClassB;

class ClassA

{

private:

int numA;

// friend class declaration

friend class ClassB;

public:

/\*constructor to initialize numA to 12

ClassA() : numA(12) {}\*/

void setdata(int x)

{

numA = x;

}

};

class ClassB {

private:

int numB;

public:

void setdata(int x)

{

numB = x;

}

/\* member function to add numA from ClassA and numB from ClassB\*/

int add(ClassA A)

{

return A.numA + numB;

}

};

int main() {

ClassAA;

A.setdata(15);

ClassBB;

B.setdata(10);

cout << "Sum: " <<B.add(A) <<endl;

return 0;

}

**//program49**

//Dynamic Constructor/multiple constructor

#include <iostream>

#include <cstring>

using namespace std;

class String

{

private:

char \*name; //char name[30];

int length;

public:

String()

{

length = 0;

name = new char[length + 1];/\*one additional size for \0 \*/

}

String(char \*s) //String(char s[])

{

length = strlen(s);

name = new char[length + 1];

strcpy(name, s);

}

void display()

{

cout << name <<endl;

}

void concat(String &s1, String &s2)

{

length = s1.length + s2.length;

delete []name;

name = new char[length + 1];

strcpy(name, s1.name);/\*copies s2.name into name\*/

strcat(name, s2.name);/\*joins s1.name with name\*/

}

};

int main()

{

char str1[] = " Rajesh";

char str2[] = " Hamal";

String s1(str1); /\*parameterized constructor is called here\*/

String s2(str2);

s1.display();

s2.display();

String s3; /\*default constructor is called here\*/

s3.concat(s1, s2);

s3.display();

return 0;

}

**//program50**

//Initialization of Objects using Constructor

#include <iostream>

using namespace std;

class Deposit

{

private:

float p\_amount;

float time;

float rate;

float interest;

public:

Deposit(){} //default constructor

Deposit(float p, int t, float r = 8)

{

p\_amount = p;

time = t;

rate = r;

interest = p\_amount \* time \* rate / 100;

}

void display()

{

cout << "-------------------------" <endl;

cout<<"Principal Amount: "<<p\_amount<<endl

<<"Time in Year: "<<time<<endl

<<"Rate: "<<rate<<endl

<<"Interest: "<<interest<<endl;

}

};

int main()

{

float p, t, r;

cout << "For Customer 1" <<endl;

cout << "Principal Amount:"; cin >>p;

cout << "Time in Year:"; cin >>t;

cout << "Rate: "; cin >>r;

Deposit c1(p, t, r);

cout << "\nFor Customer 2" <<endl;

cout << "Principal Amount:"; cin >>p;

cout << "Time in Year:"; cin >>t;

cout << "Rate: "; cin >>r;

Deposit c2(p, t, r);

cout << "\nFor Customer 3" <<endl;

cout << "Principal Amount:"; cin >>p;

cout << "Time in Year:"; cin >>t;

Deposit c3(p, t);

cout<<"For Customer 1;"<<endl;

c1.display();

cout<<"For Customer 2;"<<endl;

c2.display();

cout<<"For Customer 3;"<<endl;

c3.display();

return 0;

}

**//program63**

//area of triangle with sides given using multiple constructors

#include <iostream>

#include <cmath>

using namespace std;

class TRAINGLE

{

private:

int a;

int b;

int c;

public:

//TRAINGLE(){}

Traingle(int s1, int s2, int s3)

{

a = s1;

b = s2;

c = s3;

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

float area = sqrt(s\*(s-a)\*(s-b)\*(s-c));

cout << "Area: " << area <<endl;

}

};

int main()

{

TRAINGLE T(5, 10, 20);

return 0;

}

**//program20**

//visibility labels/ access specifiers

#include <iostream>

using namespace std;

class Parent

{

private:

int three;

protected:

int two;

public:

int one;

Parent()

{

one = two = three = 42;

}

void inParent ()

{

cout << one << two << three; /\*all legal\*/

}

};

class Child : public Parent

{

public:

void inChild ()

{

cout << one; // legal

cout << two; // legal

cout << three; // error - not legal

}

};

void main ()

{

Child c;

cout << c.one; // legal

cout <<c.two; // error - not legal

cout <<c.three; // error - not legal

}

**//program23**

//class leader is derived from student

#include <iostream>

#include <cstring>

using namespace std;

class student

{

protected:

char name[20];

int id;

public:

student(const char c[], int n) : id(n)

{

strcpy(name, c);

}

void showdata()

{

cout<<"Name: "<<name<<endl

<<"ID: "<<id<<endl;

}

};

class leader : public student

{

private:

char union\_name[20];

public:

leader(const char c[], int n, const char u[]) : student(c, n)

{

strcpy(union\_name, u);

}

void showdata()

{

student::showdata(); /\*calling base class function with ::\*/

cout<<"Union Name: "<<union\_name<<endl;

}

};

int main()

{

leader l("Nishan Dhakal", 15, "FSU");

// l.student::showdata();

l.showdata();

return 0;

}

**//program24**

//single inheritance solution 2019

#include <iostream>

using namespace std;

class Base

{

public:

float a1;

float a2;

public:

void getdata()

{

cout<<"Enter two components of a vector:";

cin>>a1>>a2;

}

void show();

};

void Base::show()

{

cout<<"("<<a1<<" , "<<a2<<")";

}

class Derived : public Base

{

private:

float x;

float y;

public:

voidadd\_vector(Derived d1, Derived d2)

{

a1 = d1.a1 + d2.a1;

a2 = d1.a2 + d2.a2;

}

void display()

{

show();

}

};

int main()

{

Derived d1, d2, d3;

d1.getdata();

d2.getdata();

d1.display();

cout<<" + ";

d2.display();

cout<<" = ";

d3.add\_vector(d1, d2);

d3.display();

return 0;

}

**//program25**

//multiple inheritance with ambiguity resolution

#include <iostream>

using namespace std;

class B1

{

protected:

int a;

public:

B1(int n) : a(n){}

void displayB()

{

cout<<"a = "<<a<<endl;

}

};

class B2

{

protected:

int b;

public:

B2(int n) : b(n){}

void displayB()

{

cout<<"b = "<<b<<endl;

}

};

class D : public B1, public B2

{

private:

int prod;

public:

D(int n1, int n2) : B1(n1), B2(n2)

{

prod = a \* b;

}

void display()

{

// B1::display();

// B2::display();

cout<<"a \* b = "<<prod<<endl;

}

};

int main()

{

D d(5, 10);

d.B1::displayB();

d.B2::displayB();

d.display();

return 0;

}

**//alpha, beta, gamma old question solution**

//mulitple inheritance

#include <iostream>

using namespace std;

class alpha

{

protected:

int A;

public:

alpha(int a): A(a){}

void showa()

{

cout << A <<endl;

}

};

class beta

{

protected:

int B;

public:

beta(int b): B(b){}

void showb()

{

cout << B <<endl;

}

};

class gamma: public alpha, public beta

{

private:

int C;

public:

gamma(int a, int b, int c):alpha(a),beta(b), C(c){}

void showg()

{

cout << C <<endl;

}

};

int main()

{

gamma g(10, 20, 30);

g.showa();

g.showb();

g.showg();

return 0;

}

**//program26**

#include <iostream>

#include <cstring>

using namespace std;

class student

{

protected:

char name[30];

int roll;

public:

student(const char n[], int r)

{

strcpy(name, n);

roll = r;

}

};

class employee

{

protected:

char org\_name[50];

int eid;

public:

employee(const char on[], int e)

{

strcpy(org\_name, on);

eid = e;

}

};

class programmer : public student, public employee

{

private:

int working\_hour;

float salary;

public:

programmer(const char n[], int r, const char on[], int e, int wh, float s) : student(n, r), employee(on, e), working\_hour(wh), salary (s){}

void showdata()

{

cout<<"Name: "<<name<<endl

<<"Roll: "<<roll<<endl

<<"Organization Name: "<<org\_name<<endl

<<"ID: "<<eid<<endl

<<"Working hour: "<<working\_hour<<endl

<<"Salary: "<<salary<<endl;

}

};

int main()

{

programmer p("Nishan", 15, "UTech", 123, 8, 45000);

cout<<"---Details---"<<endl;

p.showdata();

return 0;

}

**//program27**

//Student-->Test-->Result

#include <iostream>

#include <cstring>

using namespace std;

class Student

{

protected:

int roll;

char name[20];

public:

Student(int r, const char s[]) : roll(r)

{

strcpy(name, s);

}

void showdata()

{

cout<<"Roll Number: "<<roll<<endl

<<"Name: "<<name<<endl;

}

};

class Test : public Student //First Level Derivation

{

protected:

float mar1, mar2;

public:

Test(int r, const char s[], float m1, float m2):Student(r, s), mar1(m1), mar2(m2){}

void showdata()

{

Student::showdata();

cout<<"Marks in Subject1: "<<mar1<<endl

<<"Marks in Subject2: "<<mar2<<endl;

}

};

class Result : public Test //Second level Derivataion

{

private:

float total;

public:

Result(int r, const char s[], float m1, float m2):Test(r, s, m1, m2)

{

total = mar1 + mar2;

}

void showdata()

{

Test::showdata();

cout << "Total= " << total <<endl;

}

};

int main()

{

Result R(34, "Rupesh", 85.5, 70.0);

R.showdata();

return 0;

}

**//program28**

//hierarchical inheritance

//Employee-->[Manager, Designer, Programmer]

#include <iostream>

#include <cstring>

using namespace std;

class Employee

{

protected:

int id;

char name[30];

public:

Employee(int n, const char s[]) : id(n)

{

strcpy(name, s);

}

void showdata()

{

cout<<"ID: "<<id<<endl

<<"Name: "<<name<<endl;

}

};

class Manager : public Employee

{

private:

int whour;

float salary;

public:

Manager(int n, const char s[], int h, float sal):Employee(n, s), whour(h), salary(sal){}

void showdata()

{

Employee::showdata();

cout<<"Working Hour: "<<whour<<endl

<<"Salary: "<<salary<<endl;

}

};

class Programmer : public Employee

{

private:

int whour;

float salary;

public:

Programmer(int n, const char s[], int h, float sal):Employee(n,s),whour(h),salary(sal){}

void showdata()

{

Employee::showdata();

cout<<"Working Hour: "<<whour<<endl

<<"Salary: "<<salary<<endl;

}

};

class Designer:public Employee

{

private:

int whour;

float salary;

public:

Designer(int n, const char s[], int h, float sal) : Employee(n, s), whour(h), salary(sal){}

void showdata()

{

Employee::showdata();

cout<<"Working Hour: "<<whour<<endl

<<"Salary: "<<salary<<endl;

}

};

int main()

{

Manager m(1, "Sijan", 10, 75000);

Programmer p(2, "Binit", 12, 85000);

Designer d(3, "Bimala", 6, 35000);

cout<<"\nManager Details:"<<endl;

m.showdata();

cout<<"\nProgrammer Details:"<<endl;

p.showdata();

cout<<"\nDesigner Details:"<<endl;

d.showdata();

return 0;

}

**//program29**

//Virtual Base class and multipath inheritance

//Student-->[Test,Sports]-->Result

#include <iostream>

#include <cstring>

using namespace std;

class Student

{

protected:

int roll;

char name[30];

public:

Student(int n, const char nm[]) : roll(n)

{

strcpy(name, nm);

}

void put\_std()

{

cout<<"Roll#: "<<roll<<endl

<<"Name: "<<name<<endl;

}

};

class Test : virtual public Student

{

protected:

float term1, term2;

public:

Test(int n, const char nm[], float x, float y) : Student(n, nm), term1(x), term2(y){}

void put\_num()

{

cout<<"Marks in First term: "<<term1<<endl

<<"Marks in Second term: "<<term2<<endl;

}

};

class Sports : public virtual Student

{

protected:

float score;

public:

Sports(int n, const char nm[], float s) : Student(n, nm), score(s){}

void put\_score()

{

cout<<"Sports marks: "<<score<<endl;

}

};

class Result : public Test, public Sports

{

private:

float total;

public:

Result(int n, const char nm[], float x, float y, float s) : Student(n, nm), Test(n, nm, x, y), Sports(n, nm, s)

{

total = term1 + term2 + score;

}

void display()

{

put\_std();

put\_num();

put\_score();

cout << "Total= " << total <<endl;

}

};

int main()

{

Result R(15, "Nishan", 50.5, 60.0, 25.75);

R.display();

return 0;

}

**//program30**

//differnt no of arguments

#include <iostream>

using namespace std;

void repchar(); //prototype 1

void repchar(char ch); //prototype 2

void repchar(char ch, int n); //prototype 3

int main()

{

repchar(); //1

repchar('N'); //2

repchar('R', 50); //3

return 0;

}

//prints 40 asterisks

void repchar()

{

for (int i = 0; i < 40; i++)

{

cout << '\*';

}

cout <<endl;

}

//print 40 specified character(ch)

void repchar(char ch)

{

for (int i = 0; i < 40; i++)

{

cout <<ch;

}

cout <<endl;

}

//prints n specified character(ch)

void repchar(char ch, int n)

{

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

{

cout <<ch;

}

cout <<endl;

}

**//program31**

//different type of arguments

#include <iostream>

using namespace std;

int max(int n1, int n2)

{

if (n1 > n2)

return n1;

else

return n2;

//return (n1 > n2 ? n1 : n2);

}

float max(float n1, float n2)

{

return (n1 > n2 ? n1 : n2);

/\*if (n1 > n2)

return n1;

else

return n2;\*/

}

char max(char n1, char n2)

{

return (n1 > n2 ? n1 : n2);

}

int main()

{

cout << "Greater is: " << max(10, 20) <<endl;

cout << "Greater is: " << max('A', 'a') <<endl;

cout << "Greater is: " << max(10.25, 20.75)<<endl;

return 0;

}

**//program32**

/\*Increment operator (++) overloading using non static member function\*/

#include <iostream>

using namespace std;

class counter

{

private:

int count;

public:

counter():count(0){}

int get\_count()

{

return count;

}

counter operator ++(); //prefix

counter operator ++(int); //postfix

};

counter counter::operator ++()

{

counter temp;

temp.count = ++ count;

return temp;

//return counter(++count);

}

counter counter::operator ++(int)

{

counter temp;

temp.count = count ++;

return temp;

//return counter(count++);\*/

}

int main()

{

counter c1, c2;

cout << "Initially:" <<endl;

cout << "c1 = " << c1.get\_count() <<endl;

cout << "c2 = " << c2.get\_count() <<endl;

c1 ++; //c1.operator ++(int);

++ c2; //c2.operator ++();

cout << "\nAfter increment: \n";

cout << "c1 = " << c1.get\_count() <<endl;

cout << "c2 = " << c2.get\_count() <<endl;

/\*

cout << "\nPre-Increment:\n";

counter c3;

c3 = ++c2;

cout<<"c2= "<<c2.ret\_count()<<endl;

cout<<"c3= "<<c3.ret\_count()<<endl;

cout<<"\nPost-Increment:\n";

c3 = c2++;

cout<<"c2= "<<c2.ret\_count()<<endl;

cout<<"c3= "<<c3.ret\_count()<<endl; \*/

return 0;

}

**//program33**

/\*overloading + operator to add two Strings using non-static member function\*/

#include <iostream>

#include <cstring>

using namespace std;

class String

{

private:

char str[30];

public:

String(){}

String(const char s[])

{

strcpy(str, s);

}

void display()

{

cout <<str;

}

String operator +(String &s)

{

String temp;

strcpy(temp.str, str);

strcat(temp.str, s.str);

return temp;

}

};

int main()

{

String s1("Nishan "), s2("Dhakal"), s3;

s3 = s1 + s2; //s1.operator +(s2);

s1.display();

cout<<" + ";

s2.display();

cout<<" = ";

s3.display();

return 0;

}

**//program34**

/\*Increment operator overloading using non-member function, i.e. Friend Function\*/

#include <iostream>

using namespace std;

class counter

{

private:

int count;

public:

counter():count(0){}

int get\_count()

{

return count;

}

friend counter operator ++(counter &c);//prefix

friend counter operator ++(counter &c, int);//postfix

};

counter operator ++(counter &c)

{

counter temp;

temp.count = ++ c.count;

return temp;

//return (++c.count);

}

counter operator ++(counter &c, int)

{

counter temp;

temp.count = c.count ++;

return temp;

//return (c.count++);

}

int main()

{

counter c1, c2;

cout << "Initially:" <<endl;

cout << "c1 = " << c1.get\_count() <<endl;

cout << "c2 = " << c2.get\_count() <<endl;

c1 ++; //operator ++(c1, int);

++ c2; //operator ++(c2);

cout << "\nAfter increment: \n";

cout << "c1 = " << c1.get\_count() <<endl;

cout << "c2 = " << c2.get\_count() <<endl;

/\*

cout<<"\nPre-Increment:\n";

c3 = ++c2;

cout<<"c2= "<<c2.ret\_count()<<endl;

cout<<"c3= "<<c3.ret\_count()<<endl;

cout<<"\nPost-Increment:\n";

c3 = c2++;

cout<<"c2= "<<c2.ret\_count()<<endl;

cout<<"c3= "<<c3.ret\_count()<<endl;\*/

return 0;

}

**//program35**

//unary minus(-) overloading

#include <iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

complex(){}

complex(float r, float i):real(r), imag(i){}

void display()

{

cout<<"("<<real<<" , "<<imag<<")"<<endl;

}

void operator -();

};

void complex::operator -()

{

real = - real;

imag = - imag;

}

int main()

{

complex c1(2, -3);

cout<<"Before Negation, c1: ";

c1.display();

- c1; /\*c1.operator -(); calling operator overloading\*/

cout<<"After negation, c1: ";

c1.display();

return 0;

}

**//program36**

//overloading + operator using member function

#include <iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

complex(){}

complex(float r, float i):real(r), imag(i){}

void display()

{

cout<<"("<<real<<" , "<<imag<<")";

}

complex operator +(complex c);

};

complex complex::operator +(complex c)

{

complex temp;

temp.real = real + c.real;

temp.imag = imag + c.imag;

return temp;

}

int main()

{

complex c1(4.4, 3.6), c2(1.8, 1.06), c3;

c3 = c1 + c2; //c3 = c1.operator+(c2)

c1.display();

cout<<" + ";

c2.display();

cout<<" = ";

c3.display();

return 0;

}

**//program37**

/\*overloading + operator using non-member function (friend function)\*/

#include <iostream>

using namespace std;

class complex

{

private:

float real;

float imag;

public:

complex(){}

complex(float r, float i):real(r), imag(i){}

void display()

{

cout<<"("<<real<<" , "<<imag<<")";

}

friend complex operator +(complex c1,complex c2);

};

complex operator +(complex c1, complex c2)

{

complex temp;

temp.real = c1.real + c2.real;

temp.imag = c1.imag + c2.imag;

return temp;

}

int main()

{

complex c1(4.4, 3.6), c2(1.8, 1.06), c3;

c3 = c1 + c2; //c3 = c1.operator+(c2)

c1.display();

cout<<" + ";

c2.display();

cout<<" = ";

c3.display();

return 0;

}

**//program38**

/\*overloading assignment operator(>=) using friend function\*/

#include <iostream>

using namespace std;

class Distance

{

private:

int feet;

int inch;

public:

Distance(){}

Distance(int f, int i):feet(f), inch(i){}

void display()

{

cout << "(" << feet << " , " << inch <<")";

}

friend bool operator >=(Distance &d1, Distance &d2)

{

if((d1.feet \* 12 + d1.inch) >= (d2.feet \* 12 + d2.inch))

return true;

else

return false;

}

/\*int operator >=(Distance &d)

{

if((feet\*12+inch)>=(d.feet\*12+d.inch))

return 1;

else

return 0;

/\*(feet\*12+inch>=(d.feet\*12+d.inch)?1:0;

}\*/

};

int main()

{

Distance d1(3, 11), d2(4, 2);

cout<<"Larger distance among ";

d1.display();

cout<<" and ";

d2.display();

cout<<" is: ";

if(d1 >= d2) /\*operator >= (d1, d2); >= operator overloading\*/

{

d1.display();

}

else

{

d2.display();

}

return 0;

}

**//program39**

/\*overloading > operator to check for greater string using member function\*/

#include <iostream>

#include <cstring>

using namespace std;

class String

{

private:

char str[20];

public:

String(){}

String(const char s[])

{

strcpy(str, s);

}

void display()

{

cout <<str;

}

bool operator >(String &s)

{

if(strcmp(str, s.str) > 0)

return true;

else

return false;

}

};

int main()

{

String s1("Ram"), s2("ram");

if (s1 > s2) //if(s1.operator >(s2))

{

s1.display();

cout<<" is greater than ";

s2.display();

}

else

{

s2.display();

cout << " is greater than ";

s1.display();

}

return 0;

}

**//program40**

//shows mixed expressions

//automatic conversion

#include <iostream>

using namespace std;

int main()

{

int count = 7;

float avgWeight = 155.5;

double totalWeight = count \* avgWeight;

cout << "totalWeight= " <<totalWeight<<endl;

return 0;

}

**//program41**

// cast.cpp

// tests signed and unsigned integers

#include <iostream>

using namespace std;

int main()

{

int intVar = 1500000000; //1,500,000,000

intVar = (intVar \* 10) / 10; //result too large

cout << "intVar = " <<intVar<< endl; /\*wrong answer\*/

intVar = 1500000000; //cast to double

intVar = (static\_cast<double>(intVar) \* 10) / 10;

cout << "intVar = " <<intVar<< endl; /\*right answer\*/

return 0;

}

**//program42**

//Data Conversion between user defined types

//From Polar to Cartesian

//x = rcos(theta)

//y = rsin(theta)

//conversion routine in Destination Class

#include <iostream>

#include <cmath>

using namespace std;

class Polar

{

private:

float rad;

float ang;

public:

Polar(){}

Polar(float r, float a):rad(r), ang(a){}

float put\_radius()

{

return rad;

}

float put\_angle()

{

return ang;

}

void display()

{

cout << "(" << rad << " , " << ang << ")" <<endl;

}

};

class Cartesian

{

private:

float x;

float y;

public:

Cartesian(){}

Cartesian(float xco, float yco):x(xco), y(yco){}

Cartesian(Polar p)

{

x = p.put\_radius() \* cos(p.put\_angle());

y = p.put\_radius() \* sin(p.put\_angle());

}

void display()

{

cout << "(" << x << " , " << y << ")" <<endl;

}

};

int main()

{

Polar pol(10.25, 0.5);

Cartesian cart;

//conversion from Polar into Cartesian

cart = pol; //cart=Cartesian(pol);

cout<<"Polar Form: ";

pol.display();

cout<<"Eqivalent Cartesian Form: ";

cart.display();

return 0;

}

**//program43**

// example of this pointer

#include <iostream>

#include <cstring>

using namespace std;

class Person

{

private:

char name[20];

int age;

public:

Person(const char \*p, int n) : age(n)

{

strcpy(name, p);

}

Person greater(Person &p) //pass by reference

{

if(p.age> age)

return p;

else

return \*this;

}

void display()

{

cout<<"Name: "<<name<<endl

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

}

};

int main()

{

Person p1("Surya", 22), p2("Unika", 21);

Person p = p1.greater(p2);

cout<<"Older Person is: ";

p.display();

return 0;

}

**//program44**

//with virtual function

#include <iostream>

using namespace std;

class Point

{

public:

virtual void draw()

{

cout<<"Drawing Point."<<endl;

}

};

class Line:public Point

{

public:

void draw()

{

cout<<"Drawing Line."<<endl;

}

};

class Circle : public Point

{

public:

void draw()

{

cout<<"Drawing Circle."<<endl;

}

};

class Box : public Point

{

public:

void draw()

{

cout<<"Drawing Box."<<endl;

}

};

int main()

{

Point s;

/\*Point \*sp[] = {&s, &l, &c, &b};

for(int i = 0; i < 4; i++)

sp[i]->draw();\*/

Point \*sp;

sp = &s;

sp->draw();

Line l;

sp=&l;

sp->draw();

Circle c;

sp=&c;

sp->draw();

Box b;

sp=&b;

sp->draw();

return 0;

}

**//program45**

//Array of pointers in Virtual Function

#include <iostream>

using namespace std;

class Animal

{

public:

virtual void display() //virtual function

{

cout<<"Base class Animal."<<endl;

}

};

class Dog : public Animal

{

public:

void display()

{

cout<<"Derived class Dog."<<endl;

}

};

class Tiger : public Animal

{

public:

void display()

{

cout<<"Derived class Tiger."<<endl;

}

};

class Cow : public Animal

{

public:

void display()

{

cout<<"Derived class Cow."<<endl;

}

};

int main()

{

Animal a;

Dog d;

Tiger t;

Cow c;

Animal \*ptr[] = {&a, &d, &t, &c};

for(int i = 0; i < 4; i++)

ptr[i]->display();

return 0;

}

**//program46**

//Runtime Polymorphism (Virtual Function)

#include <iostream>

#include <cstring>

using namespace std;

class Media

{

protected:

char title[20];

float price;

public:

Media(const char \*s, float x)

{

strcpy(title, s);

price = x;

}

virtual void display(){}

};

class Book : public Media

{

private:

int pages;

public:

Book(const char \*s, float x, int n):Media(s,x), pages(n){}

void display();

};

class Tape : public Media

{

private:

float time;

public:

Tape(const char \*s, float x, float y):Media(s,x), time(y){}

void display();

};

void Book::display()

{

cout<<"Title: "<<title<<endl

<<"Pages: "<<pages<<endl

<<"Price: "<<price<<endl;

}

void Tape::display()

{

cout<<"Title: "<<title<<endl

<<"Play Time: "<<time<<endl

<<"Price: "<<price<<endl;

}

int main()

{

char Title[50];

float time, cost;

int pages;

cout<<"Enter Book Details"<<endl;

cout<<"Book Title: ";

cin.getline(Title,50);

cout<<"Pages: ";

cin>>pages;

cout<<"Price: ";

cin>>cost;

Book b(Title, cost, pages);

cout<<"Enter Tape Details"<<endl;

const char Title1[]="OOP tutorials";

cout<<"Play Time in Minutes: ";

cin>>time;

cout<<"Price: ";

cin>>cost;

Tape t(Title1, cost, time);

Media \*m[2] = {&b, &t};

cout<<"\nMedia details: "<<endl;

cout<<"------BOOK------"<<endl;

m[0]->display(); //displays Book details.

cout<<"------TAPE------"<<endl;

m[1]->display(); //displays Tape details.

return 0;

}

**//program47**

//pure virtual function

#include <iostream>

using namespace std;

class parallelogram //abstract base class

{

protected:

float length;

float breadth;

public:

parallelogram(){}

parallelogram(float l, float b):length(l), breadth(b){}

virtual float perimeter() = 0;/\*pure virtual function\*/

virtual float area() = 0;/\*pure virtual function\*/

};

class rectangle : public parallelogram

{

public:

rectangle(){}

rectangle(float l, float b) : parallelogram(l,b){}

float area()

{

return (length \* breadth);

}

float perimeter()

{

return 2 \*(length + breadth);

}

};

class square : public parallelogram

{

public:

square(){}

square(float s) : parallelogram(s, s){}

float area()

{

return length \* length;

}

float perimeter()

{

return 4 \* length;

}

};

int main()

{

parallelogram \*ptr[2];

rectangle rect(4, 5);

square sqr(4);

ptr[0] = &rect;

ptr[1] = &sqr;

cout<<"For Rectangle:\n";

cout<<"perimeter= "<<ptr[0]->perimeter()<<endl;

cout<<"area= "<<ptr[0]->area()<<endl;

cout<<"For Square:\n";

cout<<"perimeter= "<<ptr[1]->perimeter()<<endl;

cout<<"area= "<<ptr[1]->area()<<endl;

return 0;

}

**//program48**

//virtual destructor

#include <iostream>

using namespace std;

class Base

{

public:

//~Base() //non-virtual destructor

virtual ~Base() //virtual destructor

{

cout<<"Base Destroyed."<<endl;

}

};

class Derived : public Base

{

public:

~Derived()

{

cout<<"Derived Destroyed."<<endl;

}

};

int main()

{

Base \*ptr;

ptr = new Derived;

delete ptr;

return 0;

}

**//program51**

/\*finding maximum value among two parameters without template\*/

#include <iostream>

using namespace std;

int find\_max(int a, int b)

{

if (a > b)

return a;

else

return b;

//return(a>b?a:b);

}

floatfind\_max(float a, float b)

{

if (a > b)

return a;

else

return b;

//return(a>b?a:b);

}

char find\_max(char a, char b)

{

if (a > b)

return a;

else

return b;

//return(a>b?a:b);

}

int main()

{

int p = 10, q = 7;

float x = 3.35, y = 5.35;

char c1 = 'A', c2 = 'a';

cout<<"Maximum Value: "<<find\_max(p, q)<<endl;

cout<<"Maximum Value: "<<find\_max(x, y)<<endl;

cout<<"Maximum Value: "<<find\_max(c1, c2)<<endl;

return 0;

}

**//program52**

/\*finding maximum value among two parameters with template\*/

#include <iostream>

#include <cstring>

using namespace std;

template <class T>

T find\_max(T a, T b)

{

if (a > b)

return a;

else

return b;

//return(a > b ? a : b);

}

int main()

{

int p = 10, q = 7;

float x = 3.35, y = 5.35;

char c1 = 'A', c2 = 'B';

cout<<"Maximum Value: "<<find\_max(p, q)<<endl;

cout<<"Maximum Value: "<<find\_max(x, y)<<endl;

cout<<"Maximum Value: "<<find\_max(c1, c2)<<endl;

return 0;

}

**//program53**

//finding minimum among array of data

#include <iostream>

using namespace std;

const int size = 5;

template <class T>

T Minimum(T a[])

{

T m = a[0];

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

{

if (a[i] < m)

{

m = a[i];

}

}

return m;

}

int main()

{

int I[] = {10, 20, 30, 5, 7};

float F[] = {10.25, 20.4, 30.75, 7.09, 5.321};

char C[] = {'a', '/', '1', '~', '0'};

int int\_m = Minimum(I);

cout<<"Minimum among integer number:"<<int\_m<<endl;

float float\_m = Minimum(F);

cout<<"Minimum among float number:"<<float\_m<<endl;

char char\_m = Minimum(C);

cout<<"Minimum among characters: "<<char\_m<<endl;

return 0;

}

**//program54**

//multiple generic parameters

#include <iostream>

using namespace std;

template <class T1, class T2>

void display(T1 a, T2 b)

{

cout<<a<<endl

<<b<<endl;

}

int main()

{

int a = 5;

float x = 9.8;

char ch = 'R';

char name[20] = "United Technical";

display(a, x);

display(ch, name);

display(a, name);

return 0;

}

**//program55**

//Template Vector class using DMA

#include <iostream>

using namespace std;

const int size = 3;

template<class T>

class Vector

{

private:

T \*v;

public:

Vector()

{

v = new T[size];

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

v[i] = 0;

}

Vector(T \*a)

{

v = new T[size];

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

{

v[i] = a[i];

}

}

T operator\*(Vector &x)

{

T sum = 0;

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

sum += this->v[i] \* x.v[i];

delete[] v;

return sum;

}

void display()

{

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

cout << v[i] << " ";

cout<<endl;

}

};

int main()

{

int x[size] = {10, 20, 30};

int y[size] = {1, 2, 3};

Vector<int>v1;

Vector<int>v2;

v1 = x;

v2 = y;

cout << "v1: ";

v1.display();

cout << "v2: ";

v2.display();

cout << "v1 \* v2 = " << v1 \* v2 <<endl;

float p[size] = {10.5, 20.5, 30.5};

float q[size] = {1.0, 2.0, 3.0};

Vector<float>v3;

Vector<float>v4;

v3 = p;

v4 = q;

cout << "v3: ";

v3.display();

cout << "v4: ";

v4.display();

cout << "v3 \* v4 = " << v3 \* v4 <<endl;

return 0;

}

**//program56**

//Template Vector class without using DMA

#include <iostream>

using namespace std;

const int size = 3;

template<class T>

class Vector

{

private:

T v[size];

public:

Vector()

{

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

v[i] = 0;

}

Vector(T a[])

{

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

{

v[i] = a[i];

}

}

T operator\*(Vector &x)

{

T sum = 0;

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

sum += this->v[i] \* x.v[i];

return sum;

}

void display()

{

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

cout<<v[i]<<" ";

cout<<endl;

}

};

int main()

{

int x[size] = {10, 20, 30};

int y[size] = {5, 10, 15};

Vector<int>v1;

Vector<int>v2;

v1 = x;

v2 = y;

cout<<"v1: ";

v1.display();

cout<<"v2: ";

v2.display();

cout<<"v1\*v2= "<<v1\*v2<<endl;

float p[size] = {10.0, 20.5, 30.0};

float q[size] = {1.0, 2.0, 3.0};

Vector<float>v3;

Vector<float>v4;

v3 = p;

v4 = q;

cout<<"v3: ";

v3.display();

cout<<"v4: ";

v4.display();

cout<<"v3\*v4= "<<v3\*v4<<endl;

return 0;

}

**//program57**

//Class Template Stack Implementation

#include <iostream>

using namespace std;

const int size = 5;

template<class T>

class Stack

{

private:

T data[size];

int top;

public:

Stack()//default constructor

{

top = -1;

}

void push(T a)

{

top++;

data[top] = a;//data[++top]=a

}

T pop()

{

return data[top--];

}

int totaldata()

{

return (top + 1);

}

};

int main()

{

Stack<int>s1;

cout<<"Integer Stack.\n";

cout<<"-----------------\n";

cout<<"Size of Stack: "<<s1.totaldata()<<endl;

s1.push(6);

s1.push(7);

s1.push(10);

cout<<"Size of Stack: "<<s1.totaldata()<<endl;

cout<<"Popped item: "<<s1.pop()<<endl;

cout<<"Popped item: "<<s1.pop()<<endl;

cout<<"Size of Stack: "<<s1.totaldata()<<endl;

Stack<char>s2;

cout<<"\nCharacter Stack.\n";

cout<<"-----------------\n";

cout<<"\nSize of Stack: "<<s2.totaldata()<<endl;

s2.push('A');

s2.push('B');

s2.push('C');

cout<<"Size of Stack: "<<s2.totaldata()<<endl;

cout<<"Popped item: "<<s2.pop()<<endl;

cout<<"Popped item: "<<s2.pop()<<endl;

cout<<"Size of Stack: "<<s2.totaldata()<<endl;

Stack<float>s3;

cout<<"\nFloating Point Stack.\n";

cout<<"-----------------\n";

cout<<"\nSize of Stack: "<<s3.totaldata()<<endl;

s3.push(1.5);

s3.push(2.5);

s3.push(3.5);

cout<<"Size of Stack: "<<s3.totaldata()<<endl;

cout<<"Popped item: "<<s3.pop()<<endl;

cout<<"Popped item: "<<s3.pop()<<endl;

cout<<"Size of Stack: "<<s3.totaldata()<<endl;

return 0;

}

**//program58**

//Class Template With Multiple Parameters

#include <iostream>

using namespace std;

template<class T1, class T2>

class Test

{

private:

T1 a;

T2 b;

public:

Test(T1 x, T2 y):a(x), b(y){}

void display()

{

cout<<a<<" and "<<b<<endl;

}

};

int main()

{

Test <float,int> t1(5.5, 55);

cout<<"t1: ";

t1.display();

Test <int, char> t2(10, 'R');

cout<<"t2: ";

t2.display();

return 0;

}

**//program59**

//vector container- an example of STL

#include <iostream>

#include <vector>

using namespace std;

void display(vector<int>&v)

{

for(int i = 0; i <v.size(); i++)

cout<<v[i]<<" ";

}

int main()

{

vector<int> v; //integer vector

cout<<"Initial size: "<<v.size()<<endl;

//putting values into vector

int x;

cout<<"Enter three integer values: ";

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

{

cin>>x;

v.push\_back(x);

}

cout<<"\nSize of insertion: "<<v.size()<<endl;

//display contents of vector

cout<<"Current elements: ";

display(v);

//inserting at 2nd position

vector<int> :: iterator itr = v.begin();//iterator

itr = itr + 1;//itr points to the 2nd element

v.insert(itr, 5);

cout<<"\nContents after inserting: ";

display(v);

itr = v.begin();

itr += 2;

//removing 3rd element

v.erase(itr);

cout<<"\nContents after deletion: ";

display(v);

return 0;

}

**//program60**

//Divide by zero handling

#include <iostream>

using namespace std;

int main()

{

int a, b;

cout<<"Enter values of a and b: ";

cin>>a>>b;

try

{

if(b == 0)

/\*throw "Divide by zero is invalid operation";\*/

throw 0;

int result = a / b;

cout<<"The result: "<<result;

}

//catch(const char\* ex)

catch(int a)

{

cout<<"Exception:Divide by "<<a <<"case."<<endl;

}

return 0;

}

**//program61**

//invoking function that generates exception

#include <iostream>

using namespace std;

void divide(int x, int y, int z)

{

if((x-y) != 0)

{

int R = z / (x - y);

cout<<"Result: "<<R<<endl;

}

else

{

throw(x - y); //throw point

}

}

int main()

{

try

{

divide(10, 20, 30);

divide(10, 10, 30);

}

catch(int i)

{

cout<<"Divide by zero exception is caught"<<endl;

}

return 0;

}

**//program62**

//multiple catch statements

#include <iostream>

using namespace std;

void test(int x)

{

try

{

if(x == 1)

throw x; //int

else if(x == 0)

throw 'x'; //char

else if(x == -1)

throw 1.0; //double

cout<<"End of try\_block.\n";

}

catch(char c)

{

cout<<"Caught a character\n";

}

catch(int m)

{

cout<<"Caught a integer\n";

}

catch(double p)

{

cout<<"Caught a double\n";

}

cout<<"End of try\_catch mechanism.\n";

}

int main()

{

test(1);

test(0);

test(-1);

test(2);

return 0;

}