***Basic oop***

* **Structure :**

# Class //keyword class name{

Private :// access modifier

Public: //access modifier

};//semi colon\*

Int main()

{

Name of class // + declaration

Fuction call

}

Example

Class keyword class name

{

Private:

int x;>> data member

public:

test()>>>member function

{

}

};

Int main()

{

Class name + objects created (variables);

Return 0;

}

Example 1:

Class test {

Private :

Int x=20;

Public:

Void show()

{

Cout<<” x is “<<x<<endl;

};//semi colon\*

Int main()

{

Test t;

t.show();

}

* **Function outside the class:**

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test{

private :

string name ="obaid";

float age =20;

public:

void show();

};

void test::show()

{

cout<<"name of user "<<name<<endl;

cout<<"age of user "<<age<<endl;

}

int main()

{

test t;

t.show();

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

public :

void data(int age, string name);

};

void test::data(int age, string name)

{

cout<<"age of user is : "<<age<<endl;

cout<<"name of user is : "<<name<<endl;

}

int main()

{

int age;

string name;

test t;

cout<<" enter age and name of user \n";

cin>>age;

cout<<"\n";

**getline(cin,name);//string input**

t.data( age, name);

}

* **Use of set and get**

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int age;

string name;

public:

void setage(int ages)

{

age=ages;

}

void setname(string names)

{

name=names;

}

void show()

{

cout<<"age is "<<age<<endl;

cout<<"name is "<<name;

}

};

int main()

{

test t;

t.setage(88);

t.setname("ob");

t.show();

}

* **Access specifiers**

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

void inputdata()

{

cout<<"enter n "<<endl;

cin>>x;

}

void showdata()

{

cout<<"n is "<<x<<endl;

}

};

int main()

{

test t;

t.inputdata();

t.showdata();

return 0;

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

void inputdata()

{

cout<<"enter n "<<endl;

cin>>x;

}

void showdata()

{

cout<<"n is "<<x<<endl;

}

};

int main()

{

test t;

t.x=10; **//error x can’t be accessed in main bcs its private**

return 0;

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

int x;

public:

void inputdata()

{

cout<<"enter n "<<endl;

cin>>x;

}

void showdata()

{

cout<<"n is "<<x<<endl;

}

};

int main()

{

test t;

t.x=10;// if it is not specified the type it will be by default private

return 0;

}

Example 3:

#include<iostream>

#include <string>

using namespace std;

class test{

public:

int x;

void inputdata()

{

cout<<"enter n "<<endl;

cin>>x;

}

void showdata()

{

cout<<"n is "<<x<<endl;

}

};

int main()

{

test t;

t.x=10;

t.showdata();

return 0;

}

* **Constructors**

1. Default constructors
2. Parameterized constructor

Default constructors

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

test() **// same name as class name**

{

cout<<"constructor is being called......"<<endl;

x=8;

}

void showdata()

{

cout<<"value of x : "<<x<<"\n";

}

};

int main()

{

test t;

t.showdata();

return 0;

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x,y;

public:

test():x(5),y(8) **//initializing at parameters**

{

cout<<"constructor is being called......"<<endl;

}

void showdata()

{

cout<<"value of x : "<<x<<"\n";

cout<<"value of y : "<<y<<"\n";

}

};

int main()

{

test t;

t.showdata();

return 0;

}

**Parameterized constructor**

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

test()

{

cout<<"constructor is being called......"<<endl;

}

test(int n)

{

x=n;

}

void showdata()

{

cout<<"value of x : "<<x<<"\n";

}

};

int main()

{

test t(7);

t.showdata();

return 0;

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

test():x(8) **//initialized value as a parameter**

{

cout<<"constructor is being called.......\n";

}

test(int n)

{

x=n;

}

void showdata()

{

cout<<"value of x "<<x<<endl;

}

};

int main()

{

test x(9),y;

x.showdata();

y.showdata();

}

* **Destructors**

1. //same as class name
2. ~ key before any destructor
3. It is called when object is destroyed

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

test()

{

cout<<"constructor is being called.......\n";

}

~test()

{

cout<<"destructor is called............\n";

}

};

void objcreate()

{

test t1;

}

int main()

{

objcreate();

system ("pause");

test t2;

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x,y;

int sum;

public:

test():x(3),y(4)

{

cout<<"constructor is being called.......\n";

}

~test ()

{

sum=0;

sum=x+y;

cout<<"sum is "<<sum<<"\n";

}

};

void objcreate()

{

test t1 ;

}

int main()

{

objcreate ();

}

* **Placing a class in a separate file**

#include<iostream>

#include <string>

using namespace std;

class test{

private:

int x;

public:

test():x(0)

{

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

}

void set(int n)

{

x=n;

}

int get()

{

return x;

}

~test ()

{

cout<<"destructor is called............\n";

}

};

void createobj()

{

test t1,t2;

t1.set(4);

cout<<t1.get()

}

* **Player information short project #1**

*#include<iostream>*

*#include <string>*

*using namespace std;*

*class player{*

*private:*

*int age;*

*char player\_name[40],team\_name[40];*

*public:*

*player ()*

*{*

*cout<<"enter name of player.....\n";*

*cin.getline(player\_name,39);*

*if(!player\_name=='a' <='z' || player\_name=='A' >='Z')*

*{*

*cout<<".............enter name in charachters only..........\n";*

*cin.getline(player\_name,39);*

*}*

*cout<<"enter team name.....\n";*

*cin.getline(team\_name,39);*

*if(!team\_name=='a' <='z' || team\_name=='A' >='Z')*

*{*

*cout<<".............enter team name in charachters only..........\n";*

*cin.getline(team\_name,39);*

*}*

*cout<<"enter player age.....\n";*

*cin>>age;*

*if(!player\_name==48 >=57)*

*{*

*cout<<".............enter age in numbers only..........\n";*

*cin>>age;*

*}*

*system("cls");*

*}*

*void input\_data()*

*{*

*cout<<"enter name of player.....\n";*

*cin>>player\_name;*

*cout<<"enter team name.....\n";*

*cin>>team\_name;*

*cout<<"enter player age.....\n";*

*cin>>age;*

*}*

*void show()*

*{*

*cout<<"name of player.....\n"<<player\_name;*

*cout<<"\n team name.....\n"<<team\_name;*

*cout<<"\n player age.....\n"<<age;*

*system("pause");*

*}*

*void setdata(int new\_age,char new\_name[40]='\0',char new\_tname[40]='\0')*

*{*

*input\_data();*

*age=new\_age;*

*int j=0;*

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

*{*

*player\_name[j]=new\_name[i];*

*team\_name[j]=new\_tname[i];*

*j++;*

*}*

*system("pause");*

*}*

*};*

*int main()*

*{*

*player p;*

*int choice,i;*

*char j[40],k[40];*

*do*

*{*

*system("cls");*

*cout<<"..........Main menu.............\n";*

*cout<<"enter 1 : to enter data \n";*

*cout<<"enter 2 : to display data \n";*

*cout<<"enter 3 : to re - enter data \n";*

*cout<<"enter 4 : to exit \n";*

*cout<<"enter any choice \n ";*

*cin>>choice;*

*if(choice==1)*

*{*

*p.input\_data();*

*}*

*else if(choice==2)*

*{*

*p.show();*

*}*

*else if(choice==3)*

*{*

*p.setdata(i,j,k);*

*}*

*else if(choice==4)*

*{*

*break;*

*}*

*else*

*{*

*cout<<"....you had entered wrong choice........\n";*

*}*

*}while(1);*

*}*

* Count number of objects

#include<iostream>

#include <string>

using namespace std;

class player{

private:

static int counter;

public:

player ()

{

counter++;

show();

}

void show()

{

cout<<"show \n "<<counter;

}

};

int player :: counter=0;

int main()

{

player p,p1;

}

* **Bank account project #2**

#include<iostream>

#include <string>

using namespace std;

class bank{

private:

int balance,acc\_num;

char acc\_name[40],depositor\_name[40];

public:

bank ()

{

cout<<"enter depositor name.....\n";

cin.getline(depositor\_name,39);

cout<<"enter account name.....\n";

cin.getline(acc\_name,39);

cout<<"enter account number.....\n";

cin>>acc\_num;

cout<<"enter openning balance amount in the account.....\n";

cin>>balance;

system("cls");

}

void deposit(int amount)

{

cout<<"enter amount to deposit.....\n";

cin>>amount;

amount=amount+balance;

balance=amount;

cout<<"balance after deposit.......\n"<<amount;

system("pause");

}

void withdraw(int amount)

{

cout<<"enter amount to withdraw........\n";

cin>>amount;

amount=balance-amount;

balance=amount;

cout<<"balance after withdraw.......\n"<<amount;

system("pause");

}

void display()

{

cout<<"depositor name is.....\n"<<depositor\_name;

cout<<"\naccount number is.....\n"<<acc\_num;

cout<<"\naccount type is.....\n"<<acc\_name;

cout<<"\ncurrent balance is.....\n"<<balance;

system("pause");

}

};

int main()

{

bank b;

int choice,i,j;

do{

system("cls");

cout<<"press 1: to deposit money....\n";

cout<<"press 2: to withdraw money....\n";

cout<<"press 3: to diplay account details....\n";

cout<<"press 4: to exit....\n";

cin>>choice;

if(choice==1)

{

b.deposit(i);

}

else if(choice==2)

{

b.withdraw(j);

}

else if(choice==3)

{

b.display();

}

else if(choice==4)

{

break;

}

else

{

cout<<"invalid input.........\n";

}

}while(1);

system("pause");

return 0;

}

* **Passing object as an function argument**

1. **Class name will be the data type**

**Void func(bank b)**

**{**

**Bank bb;// in the function the class will used again initialized to a new variable;**

1. **To add**

**bb.km=km+b.km;// fucnt declared = actual var + var declar in parameter**

**}**

#include<iostream>

#include <string>

using namespace std;

class bank{

private:

int km;

int hr;

public:

bank ():km(0),hr(0)

{

}

void get()

{

cout<<"enter distance in km only......\n";

cin>>km;

cout<<"enter hours it took......\n";

cin>>hr;

}

void show()

{

cout<<"you travelled distance "<<km<<" kilometers "<<" in "<<hr<<" hours \n";

}

void total(bank b)

{

bank bb;

bb.km=km+b.km;

bb.hr=hr+b.hr;

cout<<"total distance in km is "<<bb.km<<" in "<<bb.hr<<" hours \n";

}

};

int main()

{

bank b,b1,i;

cout<<"...........input data..........\n";

b.get();

b.show();

cout<<"........your data............\n";

b1.get();

b1.show();

cout<<"...........total is ............\n";

b.total(b1);

return 0;

}

* **Returning object as an function argument**

#include<iostream>

#include <string>

using namespace std;

class bank{

private:

int km;

int hr;

public:

bank ():km(0),hr(0)

{

}

void get()

{

cout<<"enter distance in km only......\n";

cin>>km;

cout<<"enter hours it took......\n";

cin>>hr;

}

void show()

{

cout<<"you travelled distance "<<km<<" kilometers "<<" in "<<hr<<" hours \n";

}

bank total(bank b) ***//return type will be same as class name,***

{

bank bb***;// variable will be init with class name***

1. ***Class name + var***
2. ***Class name + new var***
3. ***Return this var***

bb.km=km+b.km;

bb.hr=hr+b.hr;

//cout<<"total distance in km is "<<bb.km<<" in "<<bb.hr<<" hours \n";

return bb;

}

};

int main()

{

bank b,b1,i,k;

cout<<"...........input data..........\n";

b.get();

b.show();

cout<<"........your data............\n";

b1.get();

b1.show();

cout<<"...........total is ............\n";

k=b.total(b1);

cout<<"total is : ";

k.show();

return 0;

}

* **Constructor overloading**

#include<iostream>

#include <string>

using namespace std;

class test

{

public:

test()

{

cout<<"constructor is called.........\n";

}

test(int n,float k)

{

num=n;

j=k;

cout<<"num is..."<<num<<"\n";

cout<<"num is..."<<j<<"\n";

}

test(float k,int n)

{

num=n;

j=k;

cout<<"num is..."<<num<<"\n";

cout<<"num is..."<<j<<"\n";

}

private:

int num;

float j;

};

int main()

{

test t(4.5,2),t3(4.5,2);

return 0;

}

* **Const variables**

#include<iostream>

#include <string>

using namespace std;

int main()

{

const int x=3;

const int y=4;

y=y+3;//compile time error

cout<<" x is : "<<x<<endl;

cout<<" y is : "<<y<<endl;

x++;//compile time error

return 0;

}

* ***Const class data members***

#include<iostream>

#include <string>

using namespace std;

class test

{

private:

const int x=2;

public:

void setvalue(int n)

{

x=n;// error cant initilize it bcs of constant...

cout<<"x is "<<x<<endl;

}

};

int main()

{

test t;

int n=3;

t.setvalue(n);

return 0;

}

EXAMPLE 2:

#include<iostream>

#include <string>

using namespace std;

class test

{

private:

const int x=2;

public:

test():x(4)// no error in initilization on the constructor

{

}

void setvalue(int n)

{

cout<<"x is "<<x<<endl;

}

};

int main()

{

test t;

int n=3;

t.setvalue(n);

return 0;

}

* ***Const objects***

#include<iostream>

#include <string>

using namespace std;

class test

{

private:

const int x=2;

public:

void setvalue(int n)

{

cout<<"x is "<<x<<endl;

}

};

int main()

{

const test t;// constant object error

int n=3;

t.setvalue(n);

return 0;

}

* ***Const functions***

#include<iostream>

#include <string>

using namespace std;

class test

{

private:

int x=2;

public:

void setvalue const(int n) //error bcs function is constant

{

x=n;

cout<<"x is "<<x<<endl;

}

};

int main()

{

test t;

int n=3;

t.setvalue(n);

return 0;

}

* ***Static data members***

#include<iostream>

#include <string>

using namespace std;

class counter

{

private:

static int count;

public:

counter()

{

count++;

}

void show()

{

cout<<" you hae declared "<<count<<" objects \n";

}

};

int counter::count=0;

int main()

{

counter obj1,obj2;

obj1.show();

return 0;

}

#include<iostream>

#include <string>

using namespace std;

class counter

{

private:

static int count;

public:

counter()

{

count++;

}

void show()

{

cout<<" you hae declared "<<count<<" objects \n";

}

};

// keyword (data type) (class name) (::) (= 0)int counter::count=0;

int main()

{

return }

***static member function***

#include<iostream>

#include <string>

using namespace std;

class counter

{

public:

static void show()

{

cout<<" xyz ";

}

};

int main()

{

Syntax : (class name ) (::) (function name () ) counter::show();

return 0;

}

***Friend function***

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test

{

private:

int num;

public:

test():num(0)

{

cout<<"constructor is called....\n";

}

friend void show(test t1);

};

void show(test t1)

{

t1.num=11;

cout<<"nubmer is "<<t1.num;

}

int main()

{

test t1;

show(t1);

}

Example 2:

#include<iostream>

#include <string>

using namespace std;

class test1{

private:

int num;

public:

test1():num(0)

{

num=33;

}

friend void show(test1 num);

};

void show(test1 num)

{

cout<<"value of n is "<<num.num<<"\n";

}

int main()

{

test1 t1;

show(t1);

}

Example 3:

#include<iostream>

#include <string>

using namespace std;

class test2;

class test1

{

private:

int num1;

public:

void input();

friend void add(test1 ,test2);

};

void test1::input()

{

cout<<"enter value of 1n \n";

cin>>num1;

}

class test2

{

public:

int num2;

public:

void input();

friend void add(test1 ,test2);

};

void test2::input()

{

cout<<"enter value of 2n \n";

cin>>num2;

system("cls");

}

void add(test1 t1,test2 t2)

{

cout<<"total is "<<t1.num1+t2.num2<<"\n";

}

int main()

{

test1 t1;

t1.input();

test2 t2;

t2.input();

add(t1,t2);

}

* ***Friend class:***

Example 1:

#include<iostream>

#include <string>

using namespace std;

class test1{

protected:

int a;

private:

int b;

public:

friend class test2;

};

class test2{

public:

void input(test1 a)

{

cout<<"enter two numbers \n";

cin>>a.a;

cin>>a.b;

sum(a);

}

void sum(test1 a)

{

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

}

};

int main()

{

test1 t1;

test2 t2;

t2.input(t1);

}

***RANDOM PROGRAM :***

#include<iostream>

#include <string>

#include <fstream>

using namespace std;

class test1{

private:

string pass,old\_pass;

int key,mob\_num;

ifstream f;

f.open(file.txt);

string username,old\_username;

public:

void screen()

{

cout<<".....................WELCOME TO XYZ PAGE...................\n";

cout<<"..............LOGIN OR SIGN UP TO CREATE NEW ACCOUNT...................\n";

cout<<"\n\n 1.SIGNUP PAGE ";

cout<<" 2.LOGIN PAGE \n";

}

void login()

{

cout<<"ENTER YOUR USER-NAME\n";

cin>>username;

cout<<"ENTER YOUR PHONE NUMBER\n";

cin>>mob\_num;

cout<<"ENTER KEY\n";

cin>>key;

cout<<"ENTER A STRONG PASSWORD\n";

cin>>pass;

}

void signup()

{

cout<<"ENTER YOUR USER-NAME\n";

cin>>old\_username;

fin>>old\_username;

cout<<"ENTER YOUR PASSWORD\n";

cout<<" FORGOT PASSWORD\n";

cin>>old\_pass;

system("cls");

forgot();

}

void forgot()

{

cout<<"ENTER YOUR PHONE NUMBBER\n";

cin>>mob\_num;

cout<<"ENTER KEY\n";

cin>>key;

cout<<"ENTER A STRONG PASSWORD\n";

cin>>pass;

}

};

int main()

{

test1 t1;

int key;

t1.screen();

cin>>key;

if(key==1)

{

system("cls");

t1.login();

}

else if(key==2)

{

system("cls");

t1.signup();

}

else if(key==3)

{

system("cls");

t1.forgot();

}

}

* ***OPERATOR OVERLOADING***

#include<iostream>

#include <string>

#include <fstream>

using namespace std;

class test1{

private:

int num;

public:

void getvalue()

{

cout<<"enter number : ";

cin>>num;

}

void operator +(test1 t)

{

test1 t1;

t1.num=num+t.num;

cout<<"the addition is "<<t1.num;

}

void operator -(test1 t)

{

test1 t1;

t1.num=num-t.num;

cout<<"the sub is "<<t1.num;

}

};

int main()

{

test1 t1,t2;

cout<<"enter 1st number \n";

t1.getvalue();

cout<<"enter 2nd number \n";

t2.getvalue();

cout<<"press 1: to add\nor \npress 2: to sub\n";

int choice;

cin>>choice;

if(choice==1)

{

t1+t2;// operator overloading

}

else if(choice==2)

{

t1-t2;

}

}

***Increment / decrement operators***

#include<iostream>

#include <string>

#include <fstream>

using namespace std;

class test1{

private:

int num;

public:

void getvalue()

{

cout<<"enter number : ";

cin>>num;

}

void operator ++()

{

num++;

}

void operator ++(int)

{

num++;

}

void operator--()

{

num--;

}

void operator--(int)

{

num--;

}

void show()

{

cout<<"n is "<<num<<"\n";

}

};

int main()

{

test1 t1;

t1.getvalue();

++t1;

t1++;

t1.show();

--t1;

t1--;

t1.show();

}

***Operator overloading example program:***

#include<iostream>

#include <string>

#include <fstream>

using namespace std;

class distances{

private:

int feets ;

float inches;

public:

distances():feets(0),inches(0)

{

}

distances(int f,float i):feets(f),inches(i)

{

}

void show()

{

cout<<"feeets =: "<<feets<<" inches =: "<<inches<<"\n";

}

distances operator ++()

{

return distances(++feets,++inches);

}

distances operator ++(int)

{

return distances(++feets,++inches);

}

void operator --()

{

--feets;

--inches;

}

void operator --(int)

{

--feets;

--inches;

}

};

int main()

{

distances d1(2,3.3),d2(4,5.5),d3,d4;

d1.show();

d2.show();

cout<<"after increments \n";

d3=++d1;

d4=d2++;

cout<<"d1 is := \n";d1.show();

cout<<"\nd2 is := \n";d2.show();

cout<<"\nafter decrement \n";

--d1;

d2--;

cout<<"d1 is := \n";d1.show();

cout<<"\nd2 is := \n";d2.show();

cout<<"\nd3 is := \n";d1.show();

cout<<"\nd4 is := \n";d2.show();

}

* ***Returning in operator overloading***

#include<iostream>

#include <string>

#include <fstream>

using namespace std;

class test{

private:

int n;

public:

test ():n(0)

{

}

void setvalue()

{

cout<<"enter n \n";

cin>>n;

}

test operator +(test t)

{

test t1;

t1.n=n+t.n;

return t1;

}

//mandatory class name when return any fuction test operator -(test t)

{

test t1;

t1.n=n-t.n;

return t1;

}

int show()

{

//return the initilzed var return n;

}

};

int main()

{

test t1,t2;

t1.setvalue();

t2.setvalue();

test sum,sub;

sum=t1+t2;

sub=t1-t2;

cout<<"sum is "<<sum.show();//key to show in return function

cout<<"\nsub is "<<sub.show();

}

# **Concatenate Two String Using Overloading Operator**

#include<iostream>

#include <string.h>

using namespace std;

class strin{

private:

char str[1000];

public:

strin()

{

strcpy(str," ");

}

strin(char s[])

{

strcpy(str,s);

}

void getvalue()

{

cin.getline(str,1000);

}

strin operator+(strin s)

{

strin temp;

strcpy(temp.str,str);

strcat(temp.str,s.str);

return temp;

}

void show()

{

cout<<str<<"\n";

}

};

int main()

{

strin s1,s2,s3;

cout<<"enter your first name \n";

s1.getvalue();

cout<<"enter your last name \n";

s2.getvalue();

cout<<"your full name \n";

s3=s1+s2;

s3.show();

}

# **Relational Operator Overloading**

#include<iostream>

using namespace std;

class strin{

private:

int feets;

float inches;

public:

strin():feets(0),inches(0.0)

{

}

strin(int f , float i ):feets(f),inches(i)

{

}

int operator<(strin d)

{

if(feets<d.feets && inches<d.inches)

{

return 1;

}

else

{

return 0;

}

}

void show()

{

cout<<"feets : "<<feets<<"inches :"<<inches<<"\n";

}

};

int main()

{

strin s1(3,3.9),s2(11,13.3),s3;

s1.show();

s2.show();

if(s1<s2)

{

cout<<"s1 is less \n";

}

else

{

cout<<"s2 is less \n";

}

}

# ***Subscript Operator Overloading Array Subscript***

#include<iostream>

using namespace std;

const int size=10;

class array{

private:

int arr[size];

public:

array()

{

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

{

arr[i]=i;

}

}

int &operator[](int i)

{

return arr[i];

}

};

int main()

{

array a1;

cout<<a1[5];

}

# ***inheritance***

#include<iostream>

using namespace std;

const int size=10;

class rectangle{

public:

int l=5;

};

class square:public rectangle{

public:

int b=4;

};

int main()

{

square s;

cout<<s.l<<endl;

cout<<s.b;

}

# ***Visibility Modes***

* ***With Protected object***

#include<iostream>

using namespace std;

const int size=10;

class test{

protected:

int n1;

public:

int n2;

void show()

{

cout<<"the value of n1 : "<<n1<<"\n";

cout<<"the value of n2 : "<<n2<<"\n";

}

};

class test1:protected test{

public:

void input()

{

cout<<"enter value of n1: ";

cin>>n1;

cout<<"enter value of n2: ";

cin>>n2;

show();

}

};

int main()

{

test1 t1;

t1.input();

}

* ***With Public object***

#include<iostream>

using namespace std;

const int size=10;

class test{

protected:

int n1;

public:

int n2;

void show()

{

cout<<"the value of n1 : "<<n1<<"\n";

cout<<"the value of n2 : "<<n2<<"\n";

}

};

class test1:public test{

public:

void input()

{

cout<<"enter value of n1: ";

cin>>n1;

cout<<"enter value of n2: ";

cin>>n2;

}

};

int main()

{

test1 t1;

t1.input();

t1.show();

}

# **Constructor and Destructor in Inheritance**

#include<iostream>

using namespace std;

const int size=10;

class test{

public:

test()

{

cout<<"constructor of parent class \n";

}

~test()

{

cout<<"destructor of parent class \n";

}

};

class test1:public test{

public:

test1()

{

cout<<"constructor of child class \n";

}

~test1()

{

cout<<"destructor of child class \n";

}

};

class test2:public test1{

public:

test2()

{

cout<<"constructor of child 2 class \n";

}

~test2()

{

cout<<"destructor of child 2 class \n";

}

};

int main()

{

test2 t1;

}

# ***Multilevel Inheritance***

#include<iostream>

using namespace std;

const int size=10;

class data{

private:

int age;

char name[100];

public:

void set()

{

cout<<"enter your name ";

gets(name);

cout<<"enter your age ";

cin>>age;

}

void show()

{

cout<<"your name "<<name<<endl;

cout<<"your age is "<<age;

}

};

class salary:public data{

private:

int amount;

public:

void get()

{

data::set();

cout<<"enter amount of salary ";

cin>>amount;

}

void show()

{

data::show();

cout<<" of salary is "<<amount<<endl;

}

};

class programming:public salary{

private:

int n;

public:

void get()

{

salary::set();

cout<<"enter n numbers of lang uh know ";

cin>>n;

}

void show()

{

salary::show();

cout<<"numbers of lang uh know "<<n;

}

};

int main()

{

programming p;

p.get();

p.show();

}

# ***Access parent class methods***

Method 1:

#include<iostream>

using namespace std;

const int size=10;

class parent{

public:

void show()

{

cout<<"parent class \n";

}

};

class child:public parent{

public:

void show()

{

cout<<"child class \n";

}

};

main()

{

child c;

c.show();

**//c.parent::show();**

}

Method 2:

#include<iostream>

using namespace std;

const int size=10;

class parent{

public:

void show()

{

cout<<"parent class \n";

}

};

class child:public parent{

public:

void show()

{

parent::show();

cout<<"child class \n";

}

};

main()

{

child c;

c.show();

}

# ***Declare Pointer Object***

#include<iostream>

using namespace std;

const int size=10;

class test{

public:

void show()

{

cout<<"hello ";

}

};

main()

{

test \*ptr;

ptr=new test;

ptr->show();

}

# ***Array of Pointer to Objects***

#include<iostream>

using namespace std;

const int size=10;

class test{

public:

void show()

{

cout<<"hello world\n";

}

};

main()

{

test \*t[5];

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

{

t[i]=new test;

t[i]->show();

}

}

# ***Virtual Function***

#include<iostream>

using namespace std;

const int size=10;

class test{

public:

virtual void show()

{

cout<<"hello world\n";

}

};

class tests:public test{

public:

void show()

{

cout<<"hello kitty\n";

}

};

main()

{

test t;

tests t1;

test \*ptr;

ptr=&t***; pointers name = and operator and object name***

ptr->show();

ptr=&t1;

ptr->show();

}

# **Early and Late Binding**

#include<iostream>

using namespace std;

class parent {

public:

virtual void show ()

{

cout<<"hello word \n";

}

};

class child:public parent {

public:

void show()

{

cout<<"hello world \n";

}

};

class subchild:public child{

public:

void show()

{

cout<<"sub child \n";

}

};

int main()

{

parent \*ptr;

int op;

cout<<"enter choice\n1 to active parent class\n2 to active child class\n3 to active subchild class\n";

cin>>op;

if(op==1)

{

ptr=new parent;

ptr->show();

}

else if(op==2)

{

ptr=new child;

ptr->show();

}

else if(op==3)

{

ptr=new subchild;

ptr->show();

}s

}

# **Pure Virtual Function and Abstract Classes**

Pure virtual func:

Esa func jo 0 sy initialized hu is func ka object nahi bnta lkn inka pointer bnta hy ese func srf declare kye jaty hy; is k pointer sy isky inherited classes ko access kia jaskhta hy

Abstract class:

Esa classes jisme ik virtual fuct hu

Code:

#include<iostream>

using namespace std;

class parent {

public:

virtual void show()=0; // pure virtual function

};

class child:public parent{

public:

void show()

{

cout<<"xyz\n";

}

};

class subchild:public child{

public:

void show()

{

cout<<"obaid\n";

}

};

int main()

{

parent \*ptr[2];

ptr[0]=new child;

ptr[1]=new subchild;

for(int i=0;i<2;i++)

{

ptr[i]->show();

}

}

# **Generic Programming**

#include<iostream>

using namespace std;

template <class t> template\_keyword + < class keyword + class name >

t show(t n , t e) class name + func name + ( parameters with keyword name of class )

{

int x=n+e;

cout<<"addition is "<<x<<endl;

}

template <class r>

r print(r name[50])

{

cout<<"name is "<<name<<endl;

}

int main()

{

show(4,4);

show(5,5);

print("obaid");

}

# **Class Templates**

#include<iostream>

using namespace std;

template <class t> template\_keyword <class\_keyword + class name>

class test{

private:

t n;

public:

void get()

{

cout<<"enter data \n";

cin>>n;

}

void show()

{

cout<<n<<"\n";

}

};

int main()

{

test<int> objint; // class name + < data type > + obj\_name+datatype

objint.get();

objint.show();

test<char> objchar;

objchar.get();

objchar.show();

}

# **Example Book and Writer Class**

#include<iostream>

#include<string>

using namespace std;

class book{

private:

int bookid;

string bookname;

int price;

public:

void input()

{ static int count=0;

cout<<"enter data of book "<<count+1<<endl;

count++;

cout<<"enter book id ";

cin>>bookid;

cin.ignore();

cout<<"enter book name ";

getline(cin,bookname);

cout<<"enter book price";

cin>>price;

}

void show()

{

cout<<"book id is "<<bookid<<"\n";

cout<<"book name is "<<bookname<<"\n";

cout<<"book price is "<<price<<"\n";

}

};

class writer{

private:

string writer\_name;

string address;

int n\_books;

book books[5];

public:

void get()

{

cout<<"enter writers name ";

getline(cin,writer\_name);

cout<<"enter writers address ";

getline(cin,address);

cout<<"enter number of books written by the Writer ";

cin>>n\_books;

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

{

books[i].input();

}

}

void show()

{

cout<<"writers name is "<<writer\_name;

cout<<"writers address is "<<address;

cout<<"number of books writen by the writer "<<n\_books;

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

{

books[i].show();

}

}

};

main()

{

writer b;

b.get();

b.show();

}

# **Example Teacher Class Writer Class Inherit Scholar Class**

#include<iostream>

#include<string>

using namespace std;

class teacher{

private:

int age;

string name,address;

public:

void input()

{ cout<<"enter teachers name ";

getline(cin,name);

cout<<"enter teachers age ";

cin>>age;

cin.ignore();

cout<<"enter teachers address ";

getline(cin,address);

}

void show()

{

cout<<"teachers name is "<<name<<"\n";

cout<<"teachers age is "<<age<<"\n";

cout<<"teachers address is "<<address<<"\n";

}

};

class writer{

private:

string writer\_name;

string address;

int n\_books;

public:

void get()

{

cout<<"enter writers name ";

getline(cin,writer\_name);

cout<<"enter writers address ";

getline(cin,address);

cout<<"enter number of books written by the Writer ";

cin>>n\_books;

}

void show()

{

cout<<"writers name is "<<writer\_name<<"\n";

cout<<"writers address is "<<address<<"\n";

cout<<"number of books writen by the writer "<<n\_books<<"\n ";

}

};

class scholar:public teacher, public writer

{

public:

void get()

{

teacher::input();

writer::get();

}

void show()

{

teacher::show();

writer::show();

}

};

main()

{

scholar b;

b.get();

b.show();

}

# **Create a Class TV That Contain Attribute**

#include<iostream>

#include<string>

#include<string.h>

using namespace std;

class tv{

private:

char name[40],model[40];

int price;

public:

tv(char n[], char m[] , int p)

{

strcpy(name,n);

strcpy(model,m);

price=p;

}

void show()

{

cout<<"brand name "<<name<<"\n";

cout<<"model "<<model<<"\n";

cout<<"price "<<price<<"\n";

}

void edit(char nam[],char mod[], int pr)

{

strcpy(name,nam);

strcpy(model,mod);

price=pr;

}

};

main()

{

tv t("sony","trx",40000);

t.show();

t.edit("pj","trdx",405);

t.show();

}

# **Create a Class Animal**

#include<iostream>

#include<string>

#include<string.h>

using namespace std;

class animal{

protected:

char name[40];

int age;

public:

void set(int a, char n[])

{

age=a;

strcpy(name,n);

}

};

class zebra:public animal{

public:

void show()

{

cout<<"name of zebra is "<<name<<"its age is "<<age<<" its colour is black and white \n";

}

};

class dolphin:public animal{

public:

void show()

{

cout<<"name of dolphin is "<<name<<"its age is "<<age<<" its colour is yellow and white \n";

}

};

main()

{

zebra z;

char n1[200]="zebrio",n2[200]="dolio";

z.set(9,n1);

z.show();

dolphin d;

d.set(40,n2);

d.show();

}

# **Create two classes A and B | Drive Third Class C from Two Class A and B using Multiple Inheritance**

#include<iostream>

#include<string>

#include<string.h>

using namespace std;

class animal{

public:

animal()

{

cout<<"hello \n";

}

};

class zebra{

public:

zebra()

{

cout<<"hello 123 \n";

}

};

class dolphin:public animal,public zebra{

public:

dolphin()

{

cout<<"eee \n ";

}

};

main()

{

dolphin d;

}

# **Calculate Rectangle Area using Classes and Object**

#include<iostream>

#include<string>

using namespace std;

class rectangle{

protected:

double r;

public:

virtual void get()=0;

virtual void show()=0;

};

class area:public rectangle{

public:

void get()

{

cout<<"enter radius of circle....\n";

cin>>r;

}

void show()

{

float a;

a=3.14\*r\*r;

cout<<"raduis of circle is "<<a<<"\n";

}

};

main()

{

rectangle \*r;

area a;

r=&a;

r->get();

r->show();

}

# **Calculate Rectangle Area using Classes and Object**

#include<iostream>

#include<string>

using namespace std;

class shape{

protected:

int l,w;

public:

virtual void get()=0;

virtual void show()=0;

};

class area:public shape{

public:

void get()

{

cout<<"enter lenght and widht of rectangle....\n";

cin>>l>>w;

}

void show()

{

long int a;

a=l\*w;

cout<<"area is "<<a<<"\n";

}

};

main()

{

shape \*r;

area a;

r=&a;

r->get();

r->show();

}

# **Area of Cone Shape**

#include<iostream>

#include<string>

#include<cmath>

using namespace std;

class shape{

protected:

double r,h;

public:

virtual void get()=0;

virtual void show()=0;

};

class cone:public shape{

public:

void get()

{

cout<<"Enter height and raduis of the cone \n";

cin>>r>>h;

}

void show()

{

double area=3.14\*r\*(r+sqrt(h\*h+r\*r));

cout<<"Area of cone is :"<<area<<"\n";

}

};

main()

{

shape \*s;

cone c;

s=&c;

s->get();

s->show();

}

Definitions:

Object -> perform any function , attribute , unique identity

Class -> collection of object which have same property

Function over loading : ik hi nam ky lkn diffrentr parameters waly functions nam b same return type b same lkn parameters dfrnt hu

Function over riding : jisme in heritance used hu nam b same return type b same

# **Calculate Area of Different Shapes**

#include<iostream>

#include<string>

using namespace std;

class shape{

protected:

float r;

int l,w,b,h,a;

public:

virtual void get()=0;

virtual void show()=0;

};

class rectangle:public shape{

public:

void get()

{

cout<<"enter lenght and widht of rectangle....\n";

cin>>l>>w;

}

void show()

{

long int a;

a=l\*w;

cout<<"area is "<<a<<"\n";

}

};

class circle:public shape{

public:

void get()

{

cout<<"enter radius of circle \n";

cin>>r;

}

void show()

{

float ar;

ar=3.14\*r\*r;

cout<<"raduis is \n"<<ar;

}

};

class triangle:public shape{

void get()

{

cout<<"enter base of triangle \n";

cin>>b;

cout<<"enter height of triangle \n";

cin>>h;

}

void show()

{

float area;

area=b\*h/2;

cout<<"raduis is \n"<<area;

}

};

class square:public shape{

void get()

{

cout<<"enter area of square \n";

cin>>a;

}

void show()

{

float are;

are=a\*a;

cout<<"raduis is \n"<<are;

}

};

main()

{

shape \*s[4];

circle c;

triangle t;

rectangle r;

square sq;

s[0]=& r;

s[1]=& c;

s[2]=& t;

s[3]=& sq;

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

{

s[i]->get();

s[i]->show();

}

}