**Inheritance**

1. Animal Hierarchy:

Problem Statement: Create a hierarchy of animal classes. Start with a base class Animal and then create derived classes like Mammal, Bird, and Fish. Each of these derived classes should have specific properties and methods related to their respective categories of animals.

Code:

#include<iostream>

using namespace std;

class Animal

{

private:

string eat;

bool breath;

public:

Animal()

{

cout << "Inside Default Animal Constructor" << endl;

eat = "food";

breath = true;

}

Animal(string eat, bool breath)

{

cout << "Inside Parameterise Animal Constructor" << endl;

this->eat = eat;

this->breath = breath;

}

void DisplayBase()

{

cout << "Eat :: " << eat << endl;

cout << "breath :: " << breath << endl;

}

};

class Bird : public Animal

{

private:

string name;

string sound;

public:

Bird()

{

cout << "Inside Default Bird Constructor" << endl;

name = "NA";

sound = "NA";

}

Bird(string name, string sound)

{

cout << "Inside Parameterise Bird Constructor" << endl;

this->name = name;

this->sound = sound;

}

void DisplayBird()

{

cout << "name :: " << name << endl;

cout << "sound :: " << sound << endl;

}

};

class Fish : public Animal

{

private:

string name;

string sound;

public:

Fish()

{

cout << "Inside Default Fish Constructor" << endl;

name = "NA";

sound = "NA";

}

Fish(string eat,bool breath,string name, string sound):Animal(eat,breath)

{

cout << "Inside Parameterise Fish Constructor" << endl;

this->name = name;

this->sound = sound;

}

void DisplayFish()

{

DisplayBase();

cout << "name :: " << name << endl;

cout << "sound :: " << sound << endl;

}

};

class Mammal : public Animal

{

private:

string name;

string sound;

public:

Mammal()

{

cout << "Inside Default Mammal Constructor" << endl;

name = "NA";

sound = "NA";

}

Mammal(string name, string sound)

{

cout << "Inside Parameterise Mammal Constructor" << endl;

this->name = name;

this->sound = sound;

}

void DisplayMammal()

{

cout << "name :: " << name << endl;

cout << "sound :: " << sound << endl;

}

};

int main()

{

Animal aobj;

aobj.DisplayBase();

Animal aobj1("Vegetable",true);

aobj1.DisplayBase();

Bird b1("sparrow","chew-chew");

Fish f1("verms",true,"Nimo","hums");

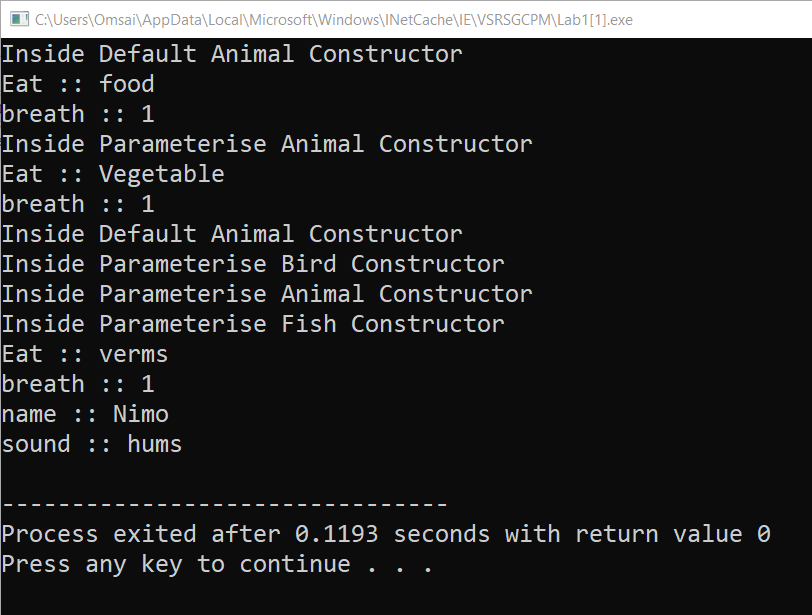
// f1.DisplayBase();

f1.DisplayFish();

return 0;

}

Output:



2. Shape Hierarchy:

Problem Statement: Design a hierarchy of shape classes. Begin with a base class Shape and then create derived classes like Circle, Rectangle, and Triangle. Each shape should have methods for calculating area and perimeter specific to its geometry.

code:

#include<iostream>

using namespace std;

class shape

{

private:

string name;

public:

shape()

{

name="shym";

}

shape(string name)

{

this->name=name;

}

void display()

{

cout<<"\*\*\*shape information\*\*\*\n";

cout<<"shape name:"<<name<<endl;

}

};

class circle : public shape

{

private:

int r;

int a,p;

public:

circle()

{

r=12;

}

circle(string name, int r):shape(name)

{

this->r=r;

}

void area()

{

a=3.14\*r\*r;

}

void peri()

{

p=2\*3.14\*r;

}

void display()

{

shape::display();

cout<<"radius of circle:"<<r<<endl;

cout<<"area of circle:"<<a<<endl;

cout<<"perimeter of circle:"<<p<<endl;

}

};

class rectangle : public shape

{

private:

int l,b;

int a,p;

public:

rectangle()

{

l=12;

b=45;

}

rectangle(string name, int l,int b):shape(name)

{

this->l=l;

this->b=b;

}

void area()

{

a=l\*b;

}

void peri()

{

p=2\*l+2\*b;

}

void display()

{

shape::display();

cout<<"length of rectangle:"<<l<<endl;

cout<<"breadth of rectangle"<<b<<endl;

cout<<"area of rectangle:"<<a<<endl;

cout<<"perimeter of rectangle :"<<p<<endl;

}

};

class triangle : public shape

{

private:

int h,b;

int a,p;

public:

triangle()

{

h=11;

b=47;

}

triangle(string name, int h,int b):shape(name)

{

this->h=h;

this->b=b;

}

void area()

{

a=0.5\*h\*b;

}

void peri()

{

p=3\*b;

}

void display()

{

shape::display();

cout<<"height of triangle:"<<h<<endl;

cout<<"breadth of triangle"<<b<<endl;

cout<<"area of triangle:"<<a<<endl;

cout<<"perimeter of triangle :"<<p<<endl;

}

};

int main()

{

circle c1("circle",10);

c1.area();

c1.peri();

c1.display();

cout<<"\n";

rectangle r1("rectangle",10,10);

r1.area();

r1.peri();

r1.display();

cout<<"\n";

triangle t1("triangle",20,10);

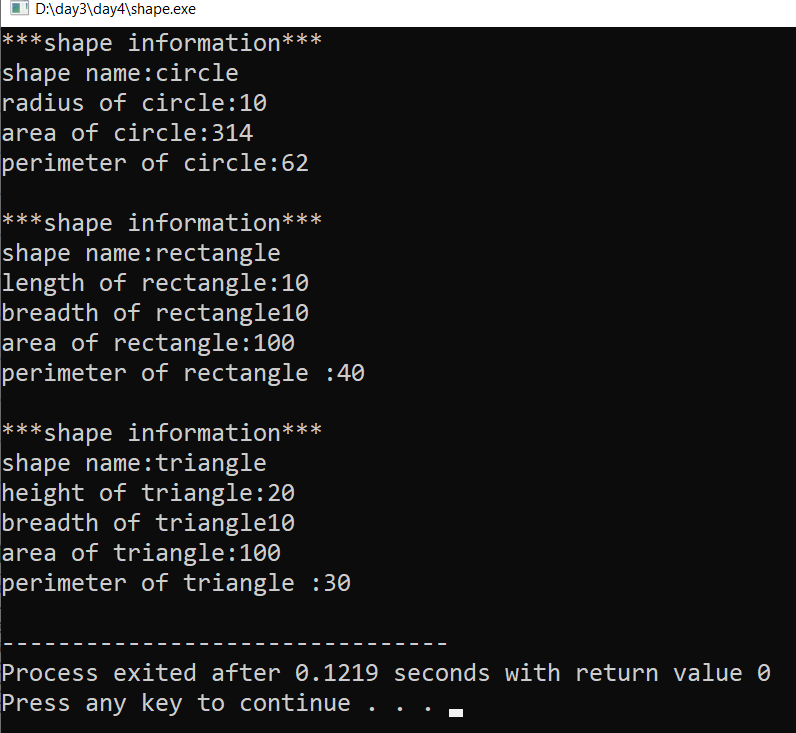
t1.area();

t1.peri();

t1.display();

}

Output:



3.Employee Inheritance:

Problem Statement: Build a system for managing employees. Create a base class Employee with attributes such as name, employee ID, and salary. Then, derive classes like Manager and Developer, each with its own attributes and methods. Implement a common method, like calculate\_salary(), in the base class.

Code:

#include<iostream>

#include<string.h>

using namespace std;

class emp

{

private:

string name;

int id;

protected:

double sal;

public:

emp()

{

//cout<<"emp class default constructor called\n";

name="not given";

id=1;

sal=10000;

}

emp(string name,int id,double sal)

{

//cout<<"emp class parameterized constructor called\n";

this->name=name;

this->id=id;

this->sal=sal;

}

void display()

{

cout<<"\*\*\*\*\*\*\*emp information\*\*\*\*\*\*\*\n";

cout<<"name:"<<name<<endl;

cout<<"id:"<<id<<endl;

cout<<" total salary:"<<sal<<endl;

}

};

class manager: public emp

{

private:

int deptno;

double incentive;

public:

manager()

{

//cout<<"manager class default constructor called\n";

deptno=10;

incentive=2500;

}

manager(string name,int id,double sal,int deptno,int incentive):emp(name,id,sal)

{

//cout<<"manager class parameterized constructor called\n";

this->deptno=deptno;

this->incentive=incentive;

}

void calSalary()

{

sal=sal+incentive;

}

void display()

{

emp::display();

cout<<"manager dept no:"<<deptno<<endl;

cout<<"manager incentive:"<<incentive<<endl;

}

};

class Developer:public emp

{

private:

int bonus;

public:

Developer()

{

bonus=10;

}

Developer(string name,int id,double sal,int bonus):emp(name,id,sal)

{

this->bonus=bonus;

}

void calsalary()

{

sal=sal+bonus;

}

void display()

{

emp::display();

cout<<"bonus:"<<bonus;

}

};

int main()

{

emp e1;

e1.display();

emp e2("Aniket",100,50000);

e2.display();

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

manager m1("ram",12,80000,20,2500);

m1.calSalary();

m1.display();

cout<<"\n\n";

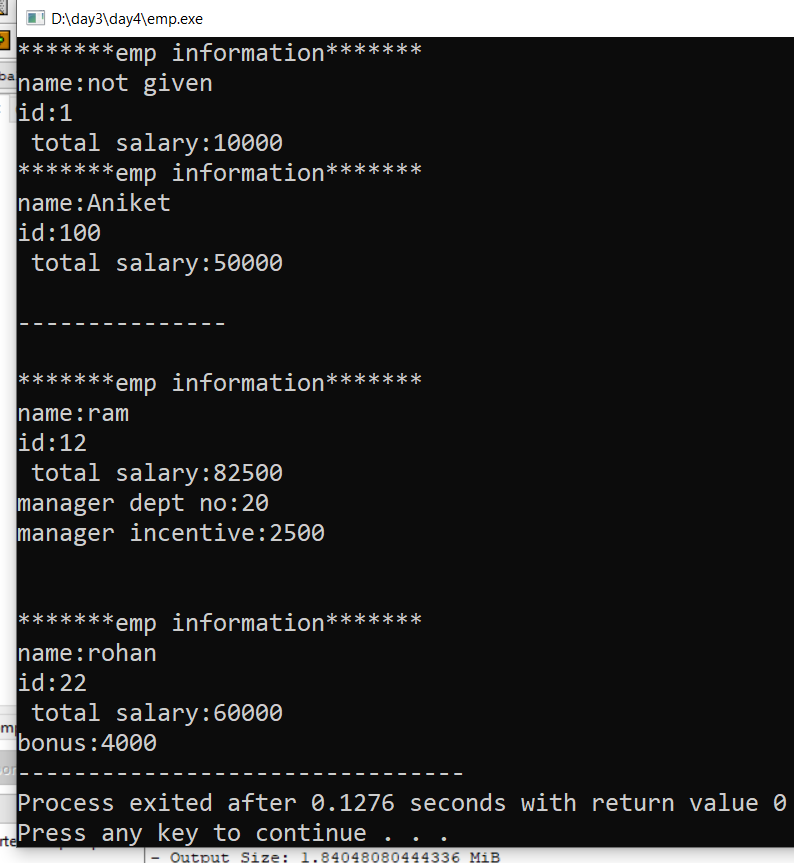
Developer d1("rohan",22,56000,4000);

d1.calsalary();

d1.display();

}

Output:



4. Vehicle Inheritance:

Problem Statement: Develop a class hierarchy for vehicles. Start with a base class Vehicle and create derived classes like Car, Motorcycle, and Truck. Each derived class should have unique properties like the number of wheels and specific methods like start\_engine().

Code:

#include<iostream>

#include<string.h>

using namespace std;

class vehicle

{

private:

string name;

int price;

string fuel\_type;

int wheel;

public:

vehicle()

{

name="abc";

price=13455;

fuel\_type="asd";

wheel=6;

}

vehicle(string name,int price,string fuel\_type,int wheel)

{

this->name=name;

this->price=price;

this->fuel\_type=fuel\_type;

this->wheel=wheel;

}

void display()

{

cout<<"\*\*\*\*\*\*\*\*\*Vehicle Information\*\*\*\*\*\*\*\*\*\*\n";

cout<<"vehicle company name :"<<name<<"\n";

cout<<"vehicle price :"<<price<<"\n";

cout<<"vehicle fuel type :"<<fuel\_type<<"\n";

cout<<"vehicle total wheel :"<<this->wheel<<"\n";

}

};

class car :public vehicle

{

public:

car(string name,int price,string fuel\_type,int wheel):vehicle(name,price,fuel\_type,wheel)

{

}

void start\_engine()

{

cout<<"car engine start with switch on the key\n";

}

void display()

{

vehicle::display();

}

};

class motorcycle :public vehicle

{

public:

motorcycle(string name,int price,string fuel\_type,int wheel):vehicle(name,price,fuel\_type,wheel)

{

}

void start\_engine()

{

cout<<"motorcycle engine start with press the start button\n";

}

void display()

{

vehicle::display();

}

};

class truck :public vehicle

{

public:

truck(string name,int price,string fuel\_type,int wheel):vehicle(name,price,fuel\_type,wheel)

{

}

void start\_engine()

{

cout<<"Truck engine start with switch on the key\n";

}

void display()

{

vehicle::display();

}

};

int main()

{

car c1("creata",1500000,"Diesel",4);

c1.display();

c1.start\_engine();

cout<<"\n";

motorcycle m1("TVS",92000,"petrol",2);

m1.display();

m1.start\_engine();

cout<<"\n";

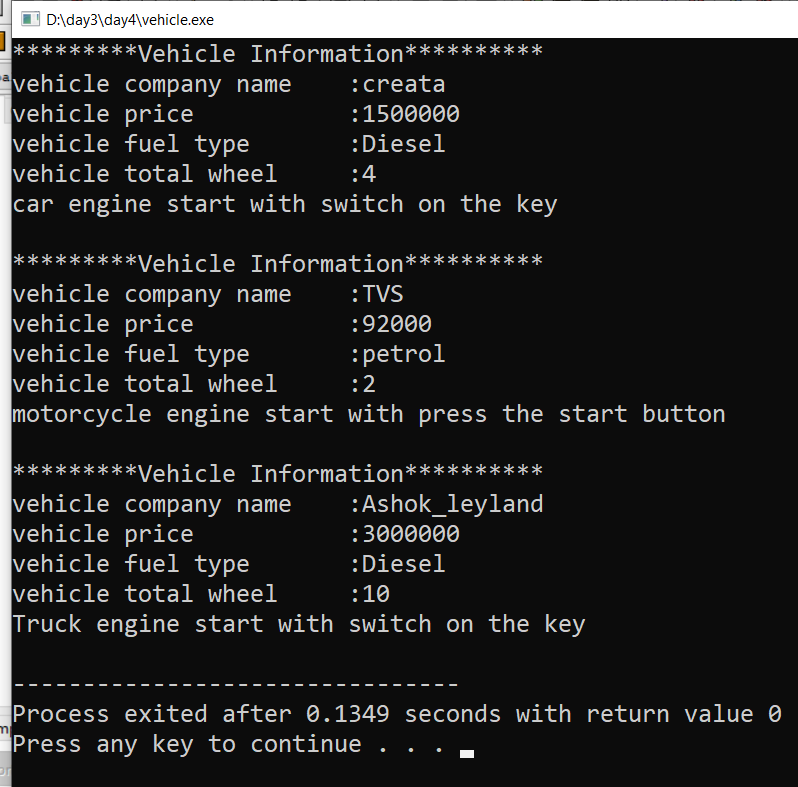
truck t1("Ashok\_leyland",3000000,"Diesel",10);

t1.display();

t1.start\_engine();

}

Output:



5.Bank Account Inheritance:

Problem Statement: Design a system for managing bank accounts. Create a base class BankAccount with attributes like account number and balance. Derive classes like SavingsAccount and CheckingAccount, each with specialized methods like withdraw() and calculate\_interest().

Code:

#include<iostream>

using namespace std;

class bank\_account

{

private:

int account\_no;

protected:

double balance;

public:

bank\_account()

{

account\_no=11;

balance=100;

}

bank\_account(int account\_no ,double balance)

{

this->account\_no=account\_no;

this->balance=balance;

}

void display()

{

cout<<"\*\*\*\*\*bank account detail\*\*\*\*\*\*\n";

cout<<"bank account number:"<<account\_no<<endl;

cout<<" bank balance:"<<balance<<endl;

}

};

class saving\_account:public bank\_account

{

private:

double withdraw,w;

public:

saving\_account()

{

withdraw=123;

}

saving\_account(int account\_no ,double balance,double withdraw):bank\_account(account\_no,balance)

{

this->withdraw=withdraw;

}

void withdrawamount()

{

w=balance-withdraw;

}

void display()

{

bank\_account::display();

cout<<"withdraw amount:"<<withdraw<<endl;

cout<<"after withdraw balance:"<<w<<endl;

}

};

int main()

{

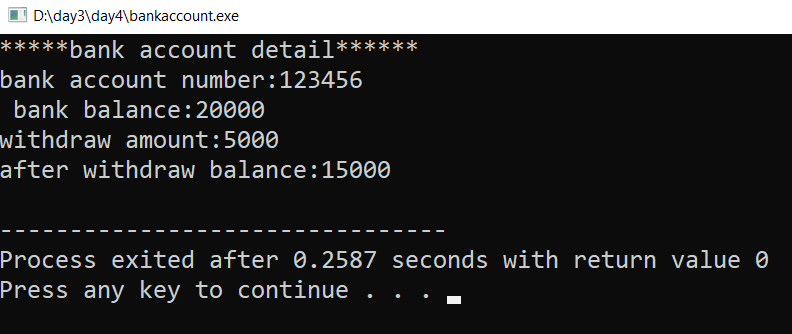
saving\_account s1(123456,20000,5000);

s1.withdrawamount();

s1.display();

}

Output:



6. Geometric Shapes with Polymorphism:

Problem Statement: Extend the shape hierarchy example by implementing polymorphism. Define a base class Shape with methods to calculate area and perimeter. Then, create derived classes like Circle, Rectangle, and Triangle, each with its own implementation of these methods.

Code:

#include<iostream>

#include<string>

using namespace std;

class shape

{

private:

string name;

public:

shape()

{

name="shym";

}

shape(string name)

{

this->name=name;

}

virtual void calarea()=0;

virtual void calperi()=0;

virtual void display()

{

cout<<"\*\*\*shape information\*\*\*\n";

cout<<"shape name:"<<name<<endl;

}

};

class circle : public shape

{

private:

int r;

public:

circle()

{

r=12;

}

circle(string name, int r):shape(name)

{

this->r=r;

}

void calarea()

{

int a=3.14\*r\*r;

cout<<"area of circle:"<<a<<endl;

}

void calperi()

{

int p=2\*3.14\*r;

cout<<"perimeter of circle:"<<p<<endl;

}

void display()

{

shape::display();

cout<<"radius of circle:"<<r<<endl;

}

};

class rectangle : public shape

{

private:

int l,b;

public:

rectangle()

{

l=12;

b=45;

}

rectangle(string name, int l,int b):shape(name)

{

this->l=l;

this->b=b;

}

void calarea()

{

int a=l\*b;

cout<<"area of rectangle:"<<a<<endl;

}

void calperi()

{

int p=2\*l+2\*b;

cout<<"perimeter of rectangle :"<<p<<endl;

}

void display()

{

shape::display();

cout<<"length of rectangle:"<<l<<endl;

cout<<"breadth of rectangle"<<b<<endl;

}

};

class triangle : public shape

{

private:

int h,b;

public:

triangle()

{

h=11;

b=47;

}

triangle(string name, int h,int b):shape(name)

{

this->h=h;

this->b=b;

}

void calarea()

{

int a=0.5\*h\*b;

cout<<"area of triangle:"<<a<<endl;

}

void calperi()

{

int p=3\*b;

cout<<"perimeter of triangle :"<<p<<endl;

}

void display()

{

shape::display();

cout<<"height of triangle:"<<h<<endl;

cout<<"breadth of triangle"<<b<<endl;

}

};

int main()

{

shape \*s1;

circle c1("circle",10);

s1=&c1;

s1->display();

s1->calarea();

s1->calperi();

cout<<"\n";

rectangle r1("rectangle",10,10);

s1=&r1;

s1->display();

s1->calarea();

s1->calperi();

cout<<"\n";

triangle t1("triangle",20,10);

s1=&t1;

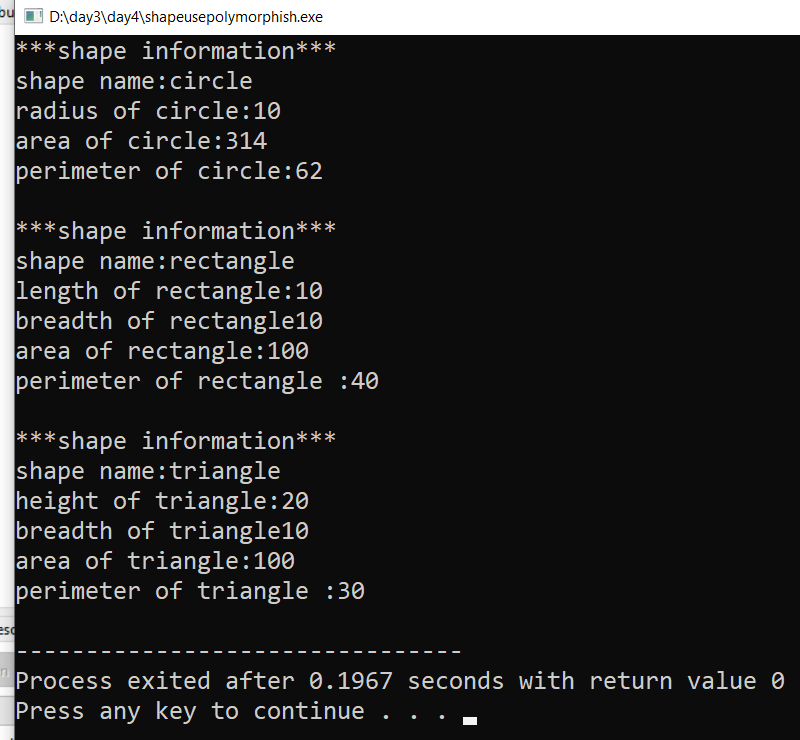
s1->display();

s1->calarea();

s1->calperi();

}

Output:



7. Person and Student Inheritance:

Problem Statement: Model a system for handling individuals and students within an educational institution. Create a base class Person with attributes like name and age. Derive a Student class with additional attributes like student ID and GPA, inheriting the common attributes from the Person class.

Code:

#include<iostream>

#include<string>

using namespace std;

class person

{

private:

string name;

int age;

public:

person()

{

name="abc";

age=13;

}

person(string name,int age)

{

this->name=name;

this->age=age;

}

void display()

{

cout<<"\*\*\*\*\*\*\*information\*\*\*\*\*\*\*\n";

cout<<"person name:"<<name<<endl;

cout<<"person age:"<<age<<endl;

}

};

class student : public person

{

private:

int id,gpa;

public:

student()

{

id=356;

gpa=58;

}

student(string name,int age,int id,int gpa):person(name,age)

{

this->id=id;

this->gpa=gpa;

}

void display()

{

person::display();

cout<<"student ID:"<<id<<endl;

cout<<"student GPA:"<<gpa<<endl;

}

};

int main()

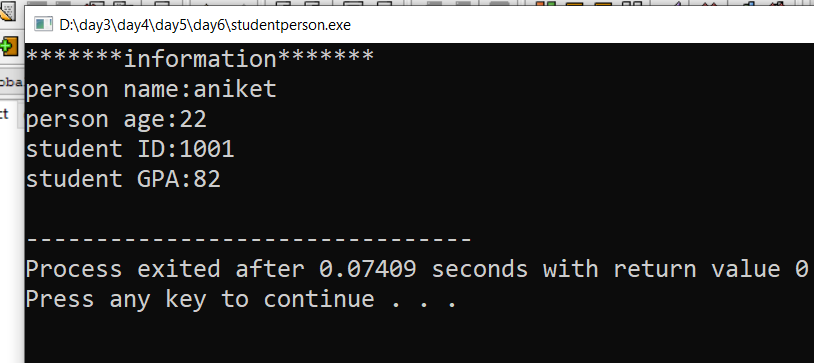
{

student s1("aniket",22,1001,82);

s1.display();

}

Output:



10.Employee Payroll System with Abstract Classes:

Problem Statement: Design an employee payroll system. Create an abstract class Employee with attributes like name and employee ID. Derive concrete classes like HourlyEmployee and SalariedEmployee. Define abstract methods for calculating pay in the base class and implement them in the derived classes.

Code:

#include<iostream>

#include<string>

using namespace std;

class employee

{

protected:

int id;

string name;

int deptid;

double basic\_sal;

public:

employee()

{

id=1;

name="asd";

deptid=23;

basic\_sal=35657;

}

employee(int id,string name,int deptid,double basic\_sal)

{

this->id=id;

this->name=name;

this->deptid=deptid;

this->basic\_sal=basic\_sal;

}

virtual void calsalary()=0;

virtual void display()

{

cout<<"\*\*\*\*\*\*employee information\*\*\*\*\*\*\*\*\n";

cout<<"employee id:"<<id<<endl;

cout<<"employee name:"<<name<<endl;

cout<<"employee deptid:"<<deptid<<endl;

cout<<"employee basic\_sal:"<<basic\_sal<<endl;

}

};

class hourlyemployee : public employee

{

private:

int hourlywork,hourlyrate;

public:

hourlyemployee()

{

hourlywork=2;

hourlyrate=300;

}

hourlyemployee(int id,string name,int deptid,double basic\_sal,int hourlywork,int hourlyrate):employee(id,name,deptid,basic\_sal)

{

this->hourlywork=hourlywork;

this->hourlyrate=hourlyrate;

}

void calsalary()

{

double sum;

sum=basic\_sal+(hourlywork\*hourlyrate);

cout<<"total salary:"<<sum<<endl;

}

void display()

{

employee::display();

cout<<"hourlyrate:"<<hourlyrate<<endl;

cout<<"hourlywork:"<<hourlywork<<endl;

}

};

class salariedemployee : public employee

{

private:

double bonus;

public:

salariedemployee()

{

bonus=0;

}

salariedemployee(int id,string name,int deptid,double basic\_sal ,double bonus ):employee(id,name,deptid,basic\_sal)

{

this->bonus=bonus;

}

void calsalary()

{

double sum;

sum=basic\_sal+bonus;

cout<<"total salary:"<<sum<<endl;

}

void display()

{

employee::display();

cout<<"bonus:"<<bonus<<endl;

}

};

int main()

{

employee \*e1;

hourlyemployee h1(12,"aniket",345,80000,4,500);

e1=&h1;

e1->display();

e1->calsalary();

cout<<"\n";

salariedemployee s1(34,"ram",567,50000,5000);

e1=&s1;

e1->display();

e1->calsalary();

}

Output: 