**C++ ASSIGNMENT**

**LAB 1:**

**Write a program to adddition of two numbers.**

#include <iostream>

using namespace std;

int main2(){

int no1,no2,sum;

cout<<"Enter two numbers: "<<endl;

cin>>no1>>no2;

sum = no1+no2;

cout<<"Sum is: "<<sum<<endl;

}

A screen shot of a computer

Description automatically generated

**Write a program to swap two numbers.**

#include <iostream>

using namespace std;

int main3(){

int no1,no2,temp;

cout<<"Enter two number for swap:"<<endl;

cin>>no1>>no2;

temp=no1;

no1=no2;

no2=temp;

cout<<"The swapped number are: "<<no1<<" "<<no2<<endl;

}

A screen shot of a computer

Description automatically generated

**Write a program to accept an integer and check if it is even or odd**

#include<iostream>

using namespace std;

int main4()

{

int num;

cout<<"Enter the number: ";

cin>>num;

if(num%2==0){

cout<<"The number is even"<<endl;

}

else{

cout<<"The number is odd"<<endl;

}

}

A black screen with white text

Description automatically generated

**Write a program to accept a number and check if it is divisible by 5 and 7.**

#include<iostream>

using namespace std;

int main5(){

int num;

cout<<"Enter the number: ";

cin>>num;

if (num%5==0 && num%7==0){

cout<<"The number is divisible by both 5 and 7"<<endl;

}

else if (num%5==0){

cout<<"The number is divisble by only 5"<<endl;

}

else if(num%7==0){

cout<<"The number is divisbile by only 7"<<endl;

}

else{

cout<<"The number is not divisble by 5 and 7"<<endl;

}

}

A black screen with white text

Description automatically generated

**Write a program, which accepts annual basic salary of an employee and calculates and displays the**

**Income tax as per the following rules.**

Basic: < 1, 50,000 Tax = 0

1, 50,000 to 3,00,000 Tax = 20%

3,00,000 Tax = 30%

#include<iostream>

using namespace std;

int main6(){

int basic\_salary;

cout<<"Enter basic salary: ";

cin>>basic\_salary;

if(basic\_salary<=150000){

cout<<"The salary is: "<<basic\_salary<<endl;

}

else if(basic\_salary>150000 || basic\_salary<300000){

basic\_salary = (basic\_salary\*20)/100;

cout<<"The salary is: "<<basic\_salary<<endl;

}

else{

basic\_salary = (basic\_salary\*30)/100;

cout<<"The salary is: "<<basic\_salary<<endl;

}

}

A black screen with white text

Description automatically generated

**Accept a lowercase character from the user and check whether the character is a vowel or consonant.**

**(Hint: a, e, i, o, u are vowels)**

#include<iostream>

using namespace std;

int main7()

{

char ch;

cout<<"Enter the alphabet: ";

cin>>ch;

if (ch=='a'|| ch=='e' || ch=='i'|| ch=='o'||ch=='u'){

cout<<"The"<<" "<<ch<<" "<<"is a vowel"<<endl;

}

else{

cout<<"The"<<" "<<ch<<" "<<"""is a consonant"<<endl;

}

}

A black screen with white text

Description automatically generated

**Write a program to input angles of a triangle and check whether triangle is valid or not.**

#include<iostream>

using namespace std;

int main8(){

int n1,n2,n3,sum;

cout<<"Enter three angles of Triangle:"<<endl;

cin>>n1>>n2>>n3;

if(n1!=0&&n2!=0&&n3!=0){

sum=n1+n2+n3;

if(sum==180){

cout<<"Triangle is valid!"<<endl;

}

}

else{

cout<<"Triangle is not valid!"<<endl;

}

}

A black screen with white text

Description automatically generated

**Write a program to find factorial of a given number. ex:no5 fact=5\*4\*3\*2\*1=120**

#include<iostream>

using namespace std;

int main18(){

int i,num,fact=1;

cout<<"Enter the number: "<<endl;

cin>>num;

for(i=1;i<=num;i++){

fact = fact\*i;

}

cout<<"The factorial is:"<<fact<<endl;

}

A screen shot of a computer

Description automatically generated

**Write a program to find m to the power n. m=3 and n=4 so 3\*3\*3\*3**

#include<iostream>

using namespace std;

int main9(){

int i,m,n,result=1;

cout<<"Enter the Number and power:";

cin>>m>>n;

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

result = result \*m;

}

cout<<"Answer:"<<result;

}

A screen shot of a computer

Description automatically generated

**Check if number is a prime number or not.:**

#include<iostream>

using namespace std;

int main10(){

int i, num;

bool flag=true;

cout<<"Enter the number: ";

cin>>num;

for(i=2;i<num;i++){

if (num%i==0){

flag = false;

break;

}

else

{

flag = true;

}

}

if(flag){

cout<<"Its prime";

}

else{

cout<<"Its not prime";

}

}

A black screen with white text

Description automatically generated

**Sum of series : 1+2+3+….+n**

#include<iostream>

using namespace std;

int main14(){

int n,sum=0;

cout<<"Enter the number: ";

cin>>n;

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

sum = sum+i;

}

cout<<"The sum is: "<<sum<<endl;

}

A black screen with white text

Description automatically generated

**Check whether the number is palindrome or not?**

#include<iostream>

using namespace std;

int main()

{

int n, temp,m, rev=0;

cout<<"Enter your number: ";

cin>>n;

temp=n;

while(temp!=0)

{

m=temp%10;

rev=rev\*10+m;

temp=temp/10;

}

if(rev==n)

cout<<n<<" Is a palindrome."<<endl;

else

cout<<n<<" Is not a palindrome."<<endl;

}

A black screen with white text

Description automatically generated

**Write a program to find sum of all even and odd numbers between 1 to n.**

#include<iostream>

using namespace std;

int main15(){

int n,sum=0,sum1=0;

cout<<"Enter the number: ";

cin>>n;

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

if(i%2==0){

sum = sum +i;

}

}

cout<<"The addition of even num is: "<<sum<<endl;

for(int j=0;j<=n;j++){

if(j%2==1){

sum1 = sum1 + j;

}

}

cout<<"The addition of odd num is: "<<sum1;

}

A screen shot of a computer

Description automatically generated

**Write a program to enter a number and print its reverse.**

**#**include<iostream>

using namespace std;

int main16()

{

int n,temp,rev=0;

cout<<"Enter the number to be reversed:"<<endl;

cin>>n;

temp=n;

while(temp!=0)

{

rev=rev\*10+temp%10;

temp=temp/10;

}

cout<<"Reverse of "<<n<<" = "<<rev<<endl;

}

A black screen with white text

Description automatically generated

**Write a program to print all Prime numbers between 1 to n.**

**#**include<iostream>

using namespace std;

int main20()

{

int n;

bool status;

cout<<"Enter the range"<<endl;

cin>>n;

cout<<"Prime Numbers from 1 to "<<n<<":"<<endl;

cout<<"1 2 ";

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

{

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

{

if(i%j==0)

{

status=false;

break;

}

else

{

status=true;

}

}

if(status)

cout<<i<<" ";

}

}

A screenshot of a computer

Description automatically generated

**Write a program to check entered number is Armstrong number or not.**

#include<iostream>

using namespace std;

int main21(){

int num,rem,sum=0,temp;

cout<<"Enter the number:";

cin>>num;

temp=num;

for (int i=0;i<=num;i++){

rem= num%10;

sum = sum + (rem\*rem\*rem);

num = num/10;

}

if (temp==sum){

cout<<"The number is amstrong number";

}

else{

cout<<"The is not amstrong number";

}

}

A black background with white text

Description automatically generated

**Write a program to find greatest of three numbers using nested if-else**

**#**include<iostream>

using namespace std;

int main22(){

int n1,n2,n3;

cout<<"Enter three numbers: ";

cin>>n1>>n2>>n3;

if(n1>n2 && n1>n3){

cout<<n1<<" "<<"is the greatest number";

}

else if(n2>n1 && n2>n3){

cout<<n2<<" "<<"is the greatest number";

}

else{

cout<<n3<<" "<<"is the greatest number";

}

}

A black screen with white text

Description automatically generated

**Create menu driven program for Pizza Shop.And display total amount**

#include<iostream>

using namespace std;

int main23(){

int i,n,ch,tot;

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

cout<<"1.Margretia Pizza Rs250 2.Cheese Burst Pizza Rs300 3.Plain Pizza Rs100"<<endl;

while(ch!=4){

cout<<"Enter your choice:";

cin>>ch;

switch(ch){

case 1:

cout<<"Enter the quantity of pizza: ";

cin>>n;

tot+=250\*n;

break;

case 2:

cout<<"Enter the quantity of pizza: ";

cin>>n;

tot+=300\*n;

break;

case 3:

cout<<"Enter the quantity of pizza: ";

cin>>n;

tot+=100\*n;

break;

}

}

cout<<"Total bill: "<<tot;

}

A black screen with white text

Description automatically generated

**Accept a single digit from the user and display it in words. For example, if digit entered is 9, display Nine.**

**#**include<iostream>

using namespace std;

int main24(){

int ch;

cout<<"Enter the number: ";

cin>>ch;

switch(ch){

case 0:

cout<<"Zero";

break;

case 1:

cout<<"One";

break;

case 2:

cout<<"Two";

break;

case 3:

cout<<"Three";

break;

case 4:

cout<<"Four";

break;

case 5:

cout<<"Five";

break;

case 6:

cout<<"Six";

break;

case 7:

cout<<"Seven";

break;

case 8:

cout<<"Eight";

break;

case 9:

cout<<"Nine";

break;

}

}

A screen shot of a computer

Description automatically generated

**Write a program, which accepts two integers and an operator as a character (+ - \* / ), performs the corresponding operation and displays the result.**

#include<iostream>

using namespace std;

int main25(){

int a,b;

char ch;

cout<<"Enter the operator: ";

cin>>ch;

switch(ch){

case '+':

cout<<"Enter a and b: ";

cin>>a>>b;

cout<<"Sum is: "<<a+b<<endl;

break;

case '-':

cout<<"Enter a and b: ";

cin>>a>>b;

cout<<"Sub is: "<<a-b<<endl;

break;

case '/':

cout<<"Enter a and b: ";

cin>>a>>b;

cout<<"Div is: "<<a/b<<endl;

break;

case'\*':

cout<<"Enter a and b: ";

cin>>a>>b;

cout<<"Mul is: "<<a\*b<<endl;

break;

default:

cout<<"Enter valid input";

break;

}

}

A screen shot of a computer

Description automatically generated

**LAB 2:**

**Write a program that accepts numbers continuously as long as the number is positive and prints the sum of the given numbers.**

#include<iostream>

using namespace std;

void positiveNumber(){

int num,sum=0;

for (int i=0;i>=0;i++){

cout<<"Enter the positive number: "<<endl;

cin>>num;

if(num>=0){

sum = sum+num;

}

else{

break;

}

}

cout<<"The sum of all positive num: "<<sum;

}

int main1(){

positiveNumber();

}

A computer screen with white text

Description automatically generated

**Write a program to accept two integers x and n and compute x raised to n.**

#include<iostream>

using namespace std;

int powerOfNumber(int m, int n){

int result=1;

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

result = result \*m;

}

cout<<"Answer:"<<result;

}

int main2(){

int x,y;

cout<<"Enter the number: "<<endl;

cin>>x;

cout<<"Enter the power: "<<endl;

cin>>y;

powerOfNumber(x,y);

}

A black screen with white text

Description automatically generated

**Write a program to accept a character, an integer n and display the next n characters**.

#include<iostream>

using namespace std;

void incrementOfChar(char c, int n){

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

c=c+n;

}

cout<<"Character is incremented to : "<<c;

}

int main3(){

char cha;

int num;

cout<<"Enter the Character: "<<endl;

cin>>cha;

cout<<"Enter the number you want to increment the character: "<<endl;

cin>>num;

incrementOfChar(cha,num);

}

A screen shot of a computer

Description automatically generated

**Write a program to calculate factorial of a number. For e.g. factorial of 5 = 5! = 5 \*4\*3\*2\*1 = 120**

#include<iostream>

using namespace std;

int factorial(int num)

{

int fact=1;

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

fact = fact\*i;

}

cout<<"Factorial of "<<num<<" is: "<<fact;

}

int main4(){

int x;

cout<<"Enter the num of which you want factorial: ";

cin>>x;

factorial(x);

}

A black screen with white text

Description automatically generated

**Write a program to calculate factors of a given number.**

#include<iostream>

using namespace std;

int factorOfNum(int n, int i){

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

if(n%i==0){

cout<<"Factors are: "<<i<<endl;

}

}

}

int main5(){

int num;

int fact;

cout<<"Enter the number of which you want to find factors: ";

cin>>num;

factorOfNum(num,fact);

}

A screenshot of a computer

Description automatically generated

**Accept two numbers and calculate GCD of them.**

#include<iostream>

using namespace std;

int gcdOfTwoNumber(int n1, int n2){

int gcd;

for (int i=1;i<=n1&&i<=n2;i++){

if(n1%i==0 && n2%i==0){

gcd=i;

}

}

cout<<"Greatest common divsior: "<<gcd<<endl;

}

int main6()

{

int num1,num2;

cout<<"Enter the two number of which you want to find gcd: ";

cin>>num1>>num2;

gcdOfTwoNumber(num1,num2);

}

A black screen with white text

Description automatically generated

**Write a menu driven program to do following operations:**

**a) Compute area of circle**

**b) Compute area of rectangle**

**c) Compute area of triangle**

**d) Exit**

**Display menu, ask choice to the user, depending on choice accept the parameters and perform the**

**operation. Continue this process until user selects exit option.**

#include<iostream>

#define PI 3.14

using namespace std;

int areaOfCircle(int r){

float a;

a = PI \* r \* r;

cout<<"Area of circle is: "<<a<<endl;

}

int areaOfRectangle(int l, int b){

int a;

a = l\*b;

cout<<"Area of rectangle is: "<<a<<endl;

}

int areaOfTriangle(int h, int b){

float a;

a = (h \* b)/2;

cout<<"Area of triangle is: "<<a<<endl;

}

int main7(){

int radius,length,breadth,height,base;

int ch;

cout<<"1.Area of circle"<<endl;

cout<<"2.Area of rectangle"<<endl;

cout<<"3.Area of triangle"<<endl;

cout<<"Exit"<<endl;

do{

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

cout<<"Enter Radius of circle: ";

cin>>radius;

areaOfCircle(radius);

break;

case 2:

cout<<"Enter length and breadth of recctangle: ";

cin>>length>>breadth;

areaOfRectangle(length,breadth);

break;

case 3:

cout<<"Enter the height and base of triangle: ";

cin>>height>>base;

areaOfTriangle(height,base);

break;

case 4:

cout<<"Finish!!";

exit(0);

default:

cout<<"Enter valid input!"<<endl;

}

}while(ch!=4);

}

A black screen with white text

Description automatically generated

**Write a program to print all prime numbers between 1 to n**

#include <iostream>

using namespace std;

bool isPrime9(int n)

{

for (int i = 2; i < n; i++) {

if (n % i == 0)

return false;

}

return true;

}

int main()

{

int num;

cout<<"Enter the num till you want prime num: ";

cin>>num;

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

if (isPrime(i))

cout<<i << " ";

}

return 0;

}

A black background with white text

Description automatically generated

**LAB 2(3):**

**Write a program to create an array of integers and perform following operations on that array like finding the sum, average, maximum and minimum number in that array. Accept the numbers of the array from user.**

#include<iostream>

using namespace std;

int sumarr(int arr[],int n){

int sum=0;

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

sum=sum+arr[i];

}

return sum;

}

double avgarr(int arr[],int n){

int sum=0;

double avg;

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

sum=sum+arr[i];

}

avg=(double)sum/n;

return avg;

}

int maxarr(int arr[],int n){

int max=arr[0];

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

if(max<arr[i]){

max=arr[i];

}

}

return max;

}

int minarr(int arr[],int n){

int min=arr[0];

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

if(min>arr[i]){

min=arr[i];

}

}

return min;

}

int main1(){

int arr[25],n;

cout<<"Enter the limit of array: "<<endl;

cin>>n;

cout<<"Enter the element of the array: "<<endl;

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

{

cin>>arr[i];

}

cout<<"Sum of element of array: "<<sumarr(arr,n)<<endl;

cout<<"Average of element of array: "<<avgarr(arr,n)<<endl;

cout<<"Maximum element of array: "<<maxarr(arr,n)<<endl;

cout<<"Minimum element of array: "<<minarr(arr,n)<<endl;

}

A computer screen with white text

Description automatically generated

**Write a program to Accept a number and display its sum of digits.:ex 568 5+6+8**

#include<iostream>

using namespace std;

int sumdigit(int n)

{

int sum=0;

while(n>0)

{

sum+=n%10;

n=n/10;

}

return sum;

}

int main2()

{

int n;

cout<<"Enter a number: ";

cin>>n;

cout<<"Sum of digits of "<<n<<" : "<<sumdigit(n);

}

A black screen with white text

Description automatically generated

**Write a program to find sum of all even and odd numbers between 1 to n.**

#include<iostream>

using namespace std;

int sumo(int n)

{

int sumodd=0;

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

{

if(i%2!=0)

sumodd+=i;

}

return sumodd;

}

int sume(int n)

{

int sumeven=0;

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

{

if(i%2==0)

sumeven+=i;

}

return sumeven;

}

int main3()

{

int n;

cout<<"Enter the range: ";

cin>>n;

cout<<"Sum of odd number from 1 to "<<n<<" is "<<sumo(n)<<endl;

cout<<"Sum of even number from 1 to "<<n<<" is "<<sume(n)<<endl;

}

A black screen with white text

Description automatically generated

**Write a program to print all Prime numbers between 1 to n.**

#include<iostream>

using namespace std;

void printPrime(int n)

{

bool status;

cout<<"1 2 ";

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

{

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

{

if(i%j==0)

{

status=false;

break;

}

else

{

status=true;

}

}

if(status)

cout<<i<<" ";

}

}

int main4()

{

int m;

cout<<"Enter the range: ";

cin>>m;

printPrime(m);

}

A screenshot of a computer

Description automatically generated

**Write a program to accept array from user .Accept number from user and search number is present in array or not.**

#include<iostream>

using namespace std;

int main5()

{

int a[20],n,m;

bool status=false;

cout<<"Enter the length of the array: ";

cin>>n;

cout<<"Enter array elements: "<<endl;

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

cin>>a[i];

cout<<"Enter the number to be searched: ";

cin>>m;

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

{

if(a[i]==m)

{

status=true;

}

}

if(status)

cout<<"Entered element is in array!";

else

cout<<"Entered element is not available in array!"<<endl;

}

A black screen with white text

Description automatically generated

**Write a program to print following pattern.**

#include<iostream>

using namespace std;

int main6(){

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

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

cout<<"\* ";

}

cout<<endl;

}

}

A black screen with white text

Description automatically generated

**Write a program to create student class with data members rollno, marks1,mark2,mark3. Accept data (acceptInfo()) and display using display member function.Also display total,percentage and grade**

#include<iostream>

using namespace std;

class student{

private: int rollno,m1,m2,m3;

public:void acceptInfo();

public:void display();

};

void student::acceptInfo(){

int rollno,m1,m2,m3;

cout<<"Enter the Roll no of student: "<<endl;

cin>>rollno;

cout<<"Enter marks of three subjects: "<<endl;

cin>>m1>>m2>>m3;

}

void student::display(){

int rollno,m1,m2,m3;

float tot,per;

char c;

cout<<"Roll no of student: "<<rollno<<endl;

cout<<"Marks of three subjects are: "<<m1<<endl;

cout<<m2<<endl;

cout<<m3<<endl;

tot=m1+m2+m3;

cout<<"Total marks: "<<tot<<endl;

per = (tot/300)\*100;

cout<<"Percentage: "<<per<<endl;

if(per>=90){

cout<<"Grade A"<<endl;

}

else if(per>=80 && per<90){

cout<<"Grade B"<<endl;

}

else{

cout<<"Grade C";

}

}

int main7(){

student s;

s.acceptInfo();

s.display();

}

A computer screen with white text

Description automatically generated

**LAB 3:**

**Create a class Person with data members as name, age, city. Write getters and setters for all the data members. Also add the display function. Create Default and Parameterized constructors. Create the object of this class in main method and invoke all the methods in that class.**

#include<iostream>

#include<string.h>

using namespace std;

class person{

private:string name;

int age;

string city;

public:person(){

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

this->name="xyz";

this->age=10;

this->city="abc";

cout<<"Person got created";

}

person(string name, int age, string city){

this->name=name;

this->age=age;

this->city=city;

cout<<"Person created";

}

void assignValue(string name,int age,string city){

this->name= name;

this->age=age;

this->city=city;

}

void display(){

cout<<"\nName: "<<name<<endl;

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

cout<<"city: "<<city<<endl;

}

void setName(string name){

this->name=name;

}

void setAge(int age){

this->age=age;

}

void setCity(string city){

this->city=city;

}

string getName(){

return name;

}

int getAge(){

return age;

}

string getCity(){

return city;

}

};

int main1(){

person p1("Kunal",23,"Surat");

p1.display();

cout<<"\n\n-----After setter called-------"<<endl;

p1.setName("Harshal");

p1.setAge(28);

p1.setCity("Nagpur");

p1.display();

cout<<"\n\n-------After getter called------"<<endl;

cout<<p1.getName()<<endl;

cout<<p1.getAge()<<endl;

cout<<p1.getCity()<<endl;

}

A screenshot of a computer

Description automatically generated

**Write a program to create student class with data members rollno, marks1,mark2,mark3. Accept data (acceptInfo()) and display using display member function. Also display total,percentage and grade**

#include<iostream>

using namespace std;

class student{

private: int rollno,m1,m2,m3;

public:void acceptInfo();

public:void display();

};

void student::acceptInfo(){

int rollno,m1,m2,m3;

cout<<"Enter the Roll no of student: "<<endl;

cin>>rollno;

cout<<"Enter marks of three subjects: "<<endl;

cin>>m1>>m2>>m3;

}

void student::display(){

int rollno,m1,m2,m3;

float tot,per;

char c;

cout<<"Roll no of student: "<<rollno<<endl;

cout<<"Marks of three subjects are: "<<m1<<endl;

cout<<m2<<endl;

cout<<m3<<endl;

tot=m1+m2+m3;

cout<<"Total marks: "<<tot<<endl;

per = (tot/300)\*100;

cout<<"Percentage: "<<per<<endl;

if(per>=90){

cout<<"Grade A"<<endl;

}

else if(per>=80 && per<90){

cout<<"Grade B"<<endl;

}

else{

cout<<"Grade C";

}

}

int main(){

student s;

s.acceptInfo();

s.display();

}

A computer screen with white text

Description automatically generated

**Create a class Date with data members as dd, mm, yy. Write getters and setters for all the data members. Also add the display function. Create Default and Parameterized constructors. Create the object of this class in main method and invoke all the methods in that class.**

#include<iostream>

using namespace std;

class Date{

private: int d,m,y;

public:Date(){

d=0;

m=0;

y=0;

}

Date(int d, int m, int y){

this->d=d;

this->m=m;

this->y=y;

}

void Input(){

cout<<"Enter the Day: ";

cin>>this->d;

cout<<"Enter the Month: ";

cin>>this->m;

cout<<"Enter the Year: ";

cin>>this->y;

}

void display(){

cout<<"Date :"<<" "<<d<<"/"<<m<<"/"<<y<<endl;

}

void setD(int d){

this->d=d;

}

void setM(int m){

this->m=m;

}

void setY(int y){

this->y=y;

}

int getD(){

return d;

}

int getM(){

return m;

}

int getY(){

return y;

}

};

int main3(){

Date datObj;

//datObj.Input();

datObj.display();

Date datObj1;

datObj1.Input();

datObj1.display();

datObj1.setD(10);

datObj1.setM(12);

datObj1.setY(1998);

datObj1.getD();

datObj1.getM();

datObj1.getY();

cout<<"\n\n------------------After Getter and Setter Called-----------------------"<<endl;

datObj1.display();

}

A screen shot of a computer

Description automatically generated

**Create a class Book with data members as bname,id,author,price. Write getters and setters for all the data members. Also add the display function. Create Default and Parameterized constructors. Create the object of this class in main method and invoke all the methods in that class.**

#include<iostream>

#include<string.h>

using namespace std;

class Book{

private:char bname[20];

int id;

char author[20];

double price;

public:

Book(){

strcpy(bname,"Default Book");

id=5;

strcpy(author,"None");

price=00.00;

}

Book(char\*bname,int id,char\*author,double price){

strcpy(this->bname,bname);

this->id=id;

strcpy(this->author,author);

this->price=price;

}

void display(){

cout<<"Book Name: "<<bname<<endl;

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

cout<<"Author: "<<author<<endl;

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

}

void setBname(char\* bname){

strcpy(this->bname,bname);

}

void setId(int id){

this->id=id;

}

void setAuth(char\* author){

strcpy(this->author,author);

}

void setPri(double price){

this->price=price;

}

char\* getBname(){

return bname;

}

int getId(){

return id;

}

char\* getAuth(){

return author;

}

double getPri(){

return price;

}

};

int main4()

{

cout<<"---------Default Constructor---------"<<endl;

Book bookObj;

bookObj.display();

cout<<"\n\n------------Parameter constrcutor--------"<<endl;

Book bookObj1("HarryPotter",5,"John Wick",54.25);

bookObj1.display();

cout<<"\n\n---------------After Getter & Setter Function--------------"<<endl;

bookObj1.setBname("The Archer");

bookObj1.setId(27);

bookObj1.setAuth("Harry");

bookObj1.setPri(500);

bookObj1.getBname();

bookObj1.getId();

bookObj1.getAuth();

bookObj1.getPri();

bookObj1.display();

}

A screenshot of a computer

Description automatically generated

**Create a class Point with data members as x,y. Create Default and Parameterized constructors. Write getters and setters for all the data members. Also add the display function. Create the object of this class in main method and invoke all the methods in that class.**

#include<iostream>

#include<string.h>

using namespace std;

class Point{

private:

double x,y,z;

public:

Point()

{

x=0;

y=0;

}

Point(double x,double y)

{

this->x=x;

this->y=y;

}

double getx()

{

return x;

}

double gety()

{

return y;

}

void setx(double x)

{

this->x=x;

}

void sety(double y)

{

this->y=y;

}

void display()

{

cout<<"X= "<<x<<"\tY= "<<y<<endl;

}

};

int main5()

{

Point pt;

pt.display();

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

Point pt1(1.3,6.2);

pt1.display();

pt1.setx(1.5);

pt1.sety(6.5);

cout<<"X= "<<pt1.getx()<<"\tY= "<<pt1.gety()<<endl;

}

A black screen with white text

Description automatically generated

**Create a class ComplexNumber with data members real, imaginary. Create Default and Parameterized constructors. Write getters and setters for all the data members. Also add the display function. Create the object of this class in main method and invoke all the methods in that class.**

#include<iostream>

using namespace std;

class ComplexNum{

private:double real,img;

public:

ComplexNum(){

real=0.0;

img=1.1;

}

ComplexNum(double real,double img){

this->real=real;

this->img=img;

}

void display(){

cout<<"Real Part: "<<real<<endl;

cout<<"Imaginary Part: "<<img<<endl;

}

void setR(double real){

this->real=real;

}

void setI(double img){

this->img=img;

}

double getR(){

return real;

}

double getI(){

return img;

}

};

int main6(){

ComplexNum cm;

cm.display();

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

ComplexNum cm1(4.25,5.12);

cm1.display();

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

cm1.setI(15.25);

cm1.setR(10.2);

cm1.getI();

cm1.getR();

cm1.display();

}

A screenshot of a computer

Description automatically generated

**LAB 4:**

**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.**

#include<iostream>

using namespace std;

class Animal{

public:int size;

string name;

string place;

public:

Animal(){ //default constructor

// cout<<"--------Default constructor---------"<<endl;

//data members

size=35;

name="xxxx";

place="Earth";

}

Animal(int size,string name,string place){

//iniatlize paramter constructor

//this is used to avoid conflict of name

this->size=size;

this->name=name;

this->place=place;

}

void Input(){

cout<<"Enter the size: "<<endl;

cin>>size;

cout<<"Enter the name: "<<endl;

cin>>name;

cout<<"Enter the place: "<<endl;

cin>>place;

}

virtual void display(){

//display function for displaying value

cout<<"Size of animal: "<<size<<endl;

cout<<"Name of animal: "<<name<<endl;

cout<<"Place where animal lives: "<<place<<endl;

}

};

class Mammal:public Animal{ //1st derived class for mammals

public: int legs;

string food;

Mammal(){

legs=4;

food="rice";

}

Mammal(int size,string name,string place,int legs,string food):Animal(size,name,place)

{

this->legs=legs;

this->food=food;

}

void Input(){

Animal::Input();

cout<<"Enter the number of legs: "<<endl;

cin>>legs;

cout<<"Enter the food mammals eat: "<<endl;

cin>>food;

}

void display(){

Animal::display();

cout<<"Legs of Mammal "<<legs<<endl;

cout<<"Food Mammsl eat "<<food<<endl;

}

};

class Bird:public Animal{

public:int wing;

float height;

public:

Bird(){

wing=0;

height=10;

}

Bird(int size ,string name,string place,int wing,float height):Animal(size,name,place){

this->wing=wing;

this->height=height;

}

void Input(){

Animal::Input();

cout<<"Enter the number of wings: "<<endl;

cin>>wing;

cout<<"Enter the height bird can fly: "<<endl;

cin>>height;

}

void display(){

Animal::display();

cout<<"wings of bird are "<<wing<<endl;

cout<<"height of flying "<<height<<endl;

}

};

class Fish:public Animal{

private:string colour;

int age;

public:Fish(){

colour="NA";

age= 10;

}

Fish(string colour,int age){

this->colour=colour;

this->age=age;

}

void Input(){

Animal::Input();

cout<<"Enter the colour of fish: "<<endl;

cin>>colour;

cout<<"Enter the age of fish: "<<endl;

cin>>age;

}

void display(){

Animal::display();

cout<<"colour of fish "<<colour<<endl;

cout<<"age of fish "<<age<<endl;

}

};

int main1(){

cout<<"\n\n------------Bird-----------------"<<endl;

Bird biObj1;

biObj1.Input();

biObj1.display();

cout<<"\n\n---------------Mammal--------------"<<endl;

Mammal mamObj1;

mamObj1.Input();

mamObj1.display();

cout<<"\n\n---------------Fish--------------"<<endl;

Fish fisObj1;

fisObj1.Input();

fisObj1.display();

}

**A screenshot of a computer program

Description automatically generated**

**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.**

#include <iostream>

#define PI 3.14

using namespace std;

class Shape{

public:

double calArea(){

return 0;

}

double calPerimeter(){

return 0;

}

};

class Circle:public Shape{

private: double radius;

public:

Circle(double radius){

this->radius=radius;

}

double calArea(){

return PI\*radius\*radius;

}

double calPerimeter(){

return 2\*PI\*radius;

}

};

class Rectangle:public Shape{

private: double length,breadth;

public:

Rectangle(double length,double breadth){

this->length=length;

this->breadth=breadth;

}

double calArea(){

return length\*breadth;

}

double calPerimeter(){

return 2\*(length+breadth);

}

};

class Triangle:public Shape{

private: double a,b,c,ba,h;

public:

Triangle(double a,double b,double c,double ba,double h){

this->a=a;

this->b=b;

this->c=c;

this->ba=ba;

this->h=h;

}

double calArea(){

return (ba\*h)/2;

}

double calPerimeter(){

return a+b+c;

}

};

int main3(){

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

Circle ciObj(5);

cout << "Area: " << ciObj.calArea() <<endl;

cout << "Perimeter: " << ciObj.calPerimeter() <<endl;

cout<<"\n\n-----------Rectangle----------------"<<endl;

Rectangle reObj(7,9);

cout << "Area: " << reObj.calArea() <<endl;

cout << "Perimeter: " << reObj.calPerimeter() <<endl;

cout<<"\n\n-----------Triangle----------------"<<endl;

Triangle triObj(5,9,7,10,12);

cout << "Area: " << triObj.calArea() <<endl;

cout << "Perimeter: " << triObj.calPerimeter() <<endl;

}

A computer screen shot of a black screen

Description automatically generated

**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.**

#include<iostream>

#include<string>

using namespace std;

class Employee{

public:int id;

string name;

double salary;

public:Employee(){

id=0;

name="NA";

salary=100.00;

}

Employee(int id,string name,double salary){

this->id=id;

this->name=name;

this->salary=salary;

}

void calSalary(){

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

}

void display(){

cout<<"ID: "<<this->id<<endl;

cout<<"Name: "<<this->name<<endl;

}

}

class Manager: virtual public Employee{

protected:string dname;

double incentives;

public:Manager(){

dname="No department assigned";

incentives=0.00;

}

Manager(int id,string name,double salary,string dname,double incentives):Employee(id,name,salary){

this->dname=dname;

this->incentives=incentives;

}

void calSalary(){

cout<<"Manager Salary: "<<salary+incentives<<endl;

}

void display(){

Employee::display();

cout<<"Department Name: "<<this->dname<<endl;

}

};

class Developer: virtual public Employee{

protected: double pa,fa,ta;

public:Developer(){

pa=25.25;

fa=20.00;

ta=12.50;

}

Developer(int id,string name,double salary,double pa,double fa, double ta):Employee(id,name,salary){

this->pa=pa;

this->fa=fa;

this->ta=ta;

}

void calSalary(){

cout<<"Developer Salary: "<<salary+pa+fa+ta<<endl;

}

void display(){

Employee::display();

cout<<"Pa: "<<this->pa<<endl;

cout<<"Fa: "<<this->fa<<endl;

cout<<"Ta: "<<this->ta<<endl;

}

};

int main4(){

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

Employee emp1;

emp1.display();

emp1.calSalary();

cout<<"\n\n--------------Default constructor of Manager-----------"<<endl;

Manager mgr1;

mgr1.display();

mgr1.calSalary();

cout<<"\n\n--------------Parameter Constructor of Manager-----------"<<endl;

Manager mgr2(12,"Harshal",500.25,"SDE",250);

mgr2.display();

mgr2.calSalary();

cout<<"\n\n--------------Default constructor of Developer-----------"<<endl;

Developer dvp1;

dvp1.display();

dvp1.calSalary();

cout<<"\n\n--------------Parameter Constructor of Developer-----------"<<endl;

Developer dvp2(1,"Kunal",900.25,51.23,41.30,87.21);

dvp2.display();

dvp2.calSalary();

}

A screenshot of a computer

Description automatically generated

**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().**

#include<iostream>

#include<string>

#include<typeinfo.h>

using namespace std;

class Vehicle{

public: string name;

private: double reg\_no;

public:

Vehicle(){

name="NA";

reg\_no=2525;

}

Vehicle(string name, double reg\_no){

this->name=name;

this->reg\_no=reg\_no;

}

Virtual void display(){

cout<<"Name of the vehicle: "<<name<<endl;

cout<<"Registration number of vehicle: "<<reg\_no<<endl;

}

};

class Car: public Vehicle{

private: int wheels;

public:

Car(){

wheels=0;

}

Car(string name,double reg\_no,int wheels):Vehicle(name,reg\_no)

{

this->wheels=wheels;

}

void display(){

Vehicle::display();

cout<<"Wheels of car: "<<wheels<<endl;

}

void StartEngine(string engine){

cout<<"Your Car Engine is: "<<engine<<endl;

}

};

class MotorCycle:public Vehicle{

private: string color;

public:

MotorCycle(){

color="None";

}

MotorCycle(string name,double reg\_no,string color):Vehicle(name,reg\_no){

this->color=color;

}

void display(){

Vehicle::display();

cout<<"Color of Motorcycle: "<<color<<endl;

}

void TopSpeed(double top\_speed){

cout<<"Top Speeed of motorcycle: "<<top\_speed<<endl;

}

};

class Truck:public Vehicle{

private:string size;

public:

Truck(){

size="Small";

}

Truck(string name,double reg\_no,string size):Vehicle(name,reg\_no){

this->size=size;

}

void display(){

Vehicle::display();

cout<<"Size of the truck: "<<size<<endl;

}

void TruckWeight(double weight){

cout<<"The "<<this->name<<" truck can load weight upto: "<<weight<<"kg"<<endl;

}

};

int main5()

{

int i=0;

Vehicle vObj("bike",2632);

vObj.display();

Car cObj("Honda City",52645,4);

MotorCycle mObj("Kawasaki",5213,"Red");

//mObj.TopSpeed(60);

Truck tObj("Volvo",4562,"Large");

//tObj.TruckWeight(5000.25);

Vehicle \*fun[4];

fun[0]=&vObj;

fun[0]=&cObj;

fun[1]=&mObj;

fun[2]=&tObj;

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

fun[i]->display();

if(typeid(\*fun[i])==typeid(Car)){

Car\*c=dynamic\_cast<Car\*>(fun[i]);

c->StartEngine("ON");

}

if(typeid(\*fun[i])==typeid(MotorCycle)){

MotorCycle\*s=dynamic\_cast<MotorCycle\*>(fun[i]);

s->TopSpeed(220);

}

if(typeid(\*fun[i])==typeid(Truck)){

Truck\*w=dynamic\_cast<Truck\*>(fun[i]);

w->TruckWeight(500);

}

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

}

}

A screenshot of a computer

Description automatically generated

**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().**

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

protected:

string account\_number;

double balance;

public:

BankAccount(string account\_number, double balance = 0) : account\_number(account\_number), balance(balance) {}

void deposit(double amount) {

balance += amount;

}

virtual void withdraw(double amount) = 0;

virtual void display() {

cout << "Account Number: " << account\_number << endl;

cout << "Balance: " << balance << endl;

}

};

class SavingsAccount : public BankAccount {

public:

SavingsAccount(string account\_number, double balance = 0) : BankAccount(account\_number, balance) {}

void withdraw(double amount) override {

if (balance >= amount) {

balance -= amount;

} else {

cout << "Insufficient funds" << endl;

}

}

void calculate\_interest(double rate) {

double interest = balance \* rate / 100;

deposit(interest);

}

void display() override {

BankAccount::display();

cout << "Account Type: Savings" << endl;

}

};

class CheckingAccount : public BankAccount {

private:

double overdraft\_limit;

public:

CheckingAccount(string account\_number, double balance = 0, double overdraft\_limit = 0) : BankAccount(account\_number, balance), overdraft\_limit(overdraft\_limit) {}

void withdraw(double amount) override {

if (balance + overdraft\_limit >= amount) {

balance -= amount;

} else {

cout << "Transaction denied: Insufficient funds" << endl;

}

}

void display() override {

BankAccount::display();

cout << "Account Type: Checking" << endl;

cout << "Overdraft Limit: " << overdraft\_limit << endl;

}

};

int main7() {

// Creating a savings account

SavingsAccount savings\_acc("Kunal", 1000);

savings\_acc.display();

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

savings\_acc.deposit(500);

savings\_acc.calculate\_interest(1.5);

savings\_acc.display();

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

// Creating a checking account

CheckingAccount checking\_acc("Ajith", 2000, 500);

checking\_acc.display();

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

checking\_acc.withdraw(2500); // Insufficient funds

checking\_acc.withdraw(1500);

checking\_acc.display();

return 0;

}

A screenshot of a computer

Description automatically generated

**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.**

#include <iostream>

#include<math.h>

using namespace std;

class Shape {

public:

virtual double area() = 0;

virtual double perimeter() = 0;

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double radius)

{

this->radius=radius;

}

double area() {

return 3.14159 \* radius \* radius;

}

double perimeter() {

return 2 \* 3.14159 \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double length, double width) {

this->length=length;

this->width=width;

}

double area() {

return length \* width;

}

double perimeter() {

return 2 \* (length + width);

}

};

class Triangle : public Shape {

private:

double side1, side2, side3;

public:

Triangle(double side1, double side2, double side3)

{

this->side1=side1;

this->side2=side2;

this->side3=side3;

}

double area() {

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double perimeter() {

return side1 + side2 + side3;

}

};

int main() {

Shape\* shapes[] = {

new Circle(5),

new Rectangle(4, 6),

new Triangle(3, 4, 5)

};

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

cout << "Shape " << i + 1 << " - Area: " << shapes[i]->area() << ", Perimeter: " << shapes[i]->perimeter() << endl;

}

return 0;

}

A screen shot of a black screen

Description automatically generated

**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.**

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

Person(string name, int age) : name(name), age(age) {}

void display() {

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

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

}

};

class Student : public Person {

private:

string student\_id;

double gpa;

public:

Student(string name, int age, string student\_id, double gpa) : Person(name, age), student\_id(student\_id), gpa(gpa) {}

void display() {

Person::display();

cout << "Student ID: " << student\_id << endl;

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

}

};

int main8() {

Person person("Gaurav", 25);

Student student("Harshal", 20, "2023001", 8.87);

cout << "Person:" << endl;

person.display();

cout << endl;

cout << "Student:" << endl;

student.display();

return 0;

}

A computer screen with white text

Description automatically generated

**Library Catalog with Books and Journals:**

**Problem Statement: Build a library catalog system. Create a base class LibraryItem with properties like title and author. Then, derive classes like Book and Journal, each with their unique properties. Implement methods to check out and return items in the derived classes.**

#include <iostream>

#include <string>

using namespace std;

class LibraryItem {

protected:

string title;

string author;

bool checked\_out;

public:

LibraryItem(string title, string author) : title(title), author(author), checked\_out(false) {}

virtual void display() = 0;

bool isCheckedOut() {

return checked\_out;

}

void checkOut() {

checked\_out = true;

}

void returnItem() {

checked\_out = false;

}

};

class Book : public LibraryItem {

private:

int num\_pages;

public:

Book(string title, string author, int num\_pages) : LibraryItem(title, author), num\_pages(num\_pages) {}

void display() {

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

cout << "Author: " << author << endl;

cout << "Number of Pages: " << num\_pages << endl;

cout << "Checked Out: " << (checked\_out ? "Yes" : "No") << endl;

}

};

class Journal : public LibraryItem {

private:

int issue\_number;

public:

Journal(string title, string author, int issue\_number) : LibraryItem(title, author), issue\_number(issue\_number) {}

void display() {

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

cout << "Author: " << author << endl;

cout << "Issue Number: " << issue\_number << endl;

cout << "Checked Out: " << (checked\_out ? "Yes" : "No") << endl;

}

};

int main9() {

Book book("The Great Gatsby", "F. Scott Fitzgerald", 180);

Journal journal("Science", "Nature Publishing Group", 123);

cout << "Book Information:" << endl;

book.display();

cout << endl;

cout << "Journal Information:" << endl;

journal.display();

cout << endl;

book.checkOut();

cout << "Book Checked Out: " << book.isCheckedOut() << endl;

return 0;

}

A computer screen shot of a black screen

Description automatically generated

**Shape Sorting with Interfaces:**

**Problem Statement: Implement a shape sorting program. Define a base class Shape with properties like area and perimeter. Create derived classes like Circle, Rectangle, and Triangle. Implement an interface Sortable with a method to compare shapes by area. Use this interface to sort a list of shapes.**

#include <iostream>

#include <algorithm>

#include <math.h>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double radius) : radius(radius) {}

double area() const {

return 3.14159 \* radius \* radius;

}

double perimeter() const override {

return 2 \* 3.14159 \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double length, double width) : length(length), width(width) {}

double area() const override {

return length \* width;

}

double perimeter() const override {

return 2 \* (length + width);

}

};

class Triangle : public Shape {

private:

double side1, side2, side3;

public:

Triangle(double side1, double side2, double side3) : side1(side1), side2(side2), side3(side3) {}

double area() const override {

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double perimeter() const override {

return side1 + side2 + side3;

}

};

bool compareShapesByArea(const Shape\* shape1, const Shape\* shape2) {

return shape1->area() < shape2->area();

}

int main10() {

Circle circle(5);

Rectangle rectangle(4, 6);

Triangle triangle(3, 4, 5);

Shape\* shapes[] = {&circle, &rectangle, &triangle};

int size = sizeof(shapes) / sizeof(shapes[0]);

sort(shapes, shapes + size, compareShapesByArea);

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

cout << "Shape " << i + 1 << ": Area = " << shapes[i]->area() << ", Perimeter = " << shapes[i]->perimeter() << endl;

}

return 0;

}

A black screen with white text

Description automatically generated

**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.**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

int employeeId;

public:

Employee(string name, int employeeId) : name(name), employeeId(employeeId) {}

virtual double calculatePay() const = 0;

virtual void display() const {

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

cout << "Employee ID: " << employeeId << endl;

}

};

class HourlyEmployee : public Employee {

private:

double hourlyRate;

double hoursWorked;

public:

HourlyEmployee(string name, int employeeId, double hourlyRate, double hoursWorked)

: Employee(name, employeeId), hourlyRate(hourlyRate), hoursWorked(hoursWorked) {}

double calculatePay() const override {

return hourlyRate \* hoursWorked;

}

void display() const override {

Employee::display();

cout << "Pay: " << calculatePay() << endl;

}

};

class SalariedEmployee : public Employee {

private:

double salary;

public:

SalariedEmployee(string name, int employeeId, double salary)

: Employee(name, employeeId), salary(salary) {}

double calculatePay() const override {

return salary;

}

void display() const override {

Employee::display();

cout << "Pay: " << calculatePay() << endl;

}

};

int main11() {

HourlyEmployee hourlyEmp("yash", 1001, 15.5, 40);

SalariedEmployee salariedEmp("Smit", 1002, 50000);

cout << "Hourly Employee:" << endl;

hourlyEmp.display();

cout << endl;

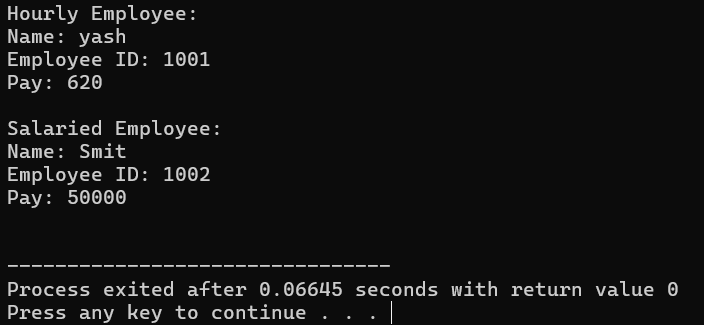
cout << "Salaried Employee:" << endl;

salariedEmp.display();

cout << endl;

return 0;

}



**LAB 5:**

**Fresh business scenario to apply inheritance , polymorphism to emp based organization scenario.**

**Create Emp based organization structure --- Emp , Mgr , Worker**

**1.1 Emp state--- id(int), name, deptId , basicSalary(double). Accept all of above in constructor arguments.**

**Methods ---**

**1.2. compute net salary ---ret 0 (eg : public double computeNetSalary(){return 0;})**

**1.2 Mgr state ---id,name,basic,deptId , perfBonus**

**Add suitable constructor**

**Methods ----**

**1. compute net salary (formula: basic+perfBonus) -- override computeNetSalary**

**Methods :**

**1.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate**

**1. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary**

**2. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter). Create suitable array to store organization details.**

**Provide following options**

**1. Hire Manager**

**I/P : all manager details**

**2. Hire Worker**

**I/P : all worker details**

**3. Display information of all employees net salary (by invoking computeNetSal),**

**4. Exit**

#include<iostream>

#include<string>

#include<typeinfo>

using namespace std;

class Employee{

private: int id;

string name;

int deptId;

protected: double basic\_salary;

public:Employee(){ //Default constructor of employee class

id=0;

name="Unknown";

deptId=0;

basic\_salary=0.00;

}

Employee(int id,string name,int deptId, double basic\_salary) //Parameter constructor of Employee

{

this->id=id;

this->name=name;

this->deptId=deptId;

this->basic\_salary=basic\_salary;

virtual void calNetSal(){ //virtual function of calculating salary is created so it can be overrriden by child class

cout<<"Employee salary: "<<basic\_salary<<endl;

}

virtual void Input(){ //virtual function of Taking input is created so it can be overrriden by child class

cout<<"Id: ";

cin>>this->id;

cout<<"Name: ";

cin>>this->name;

cout<<"Dept Id: ";

cin>>this->deptId;

cout<<"Basic Salary: ";

cin>>this->basic\_salary;

}

virtual void display(){ //virtual function of Display is created so it can be overrriden by child class

cout<<"Id is "<<this->id<<endl;

cout<<"Name is "<<this->name<<endl;

cout<<"Dept id is "<<this->deptId<<endl;

cout<<"Basic salary is "<<this->basic\_salary<<endl;

}

};

class Manager:public Employee{

private: double perfBonus;

public:Manager(){ //Default constructor of Manager class

this->perfBonus=100;

}

Manager(int id,string name,int deptId, double basic\_salary,double perfBonus):Employee(id,name,deptId,basic\_salary){ //parameter constructor of manager class

this->perfBonus=perfBonus;

}

void Input(){ //Input function is created to take bonus as input

Employee::Input();

cout<<"Perfomace bonus: ";

cin>>perfBonus;

}

void display(){ //Display function is created to print Output

Employee::display();

cout<<"Manager Bonus: "<<perfBonus<<endl;

}

void calNetSal(){ //calNetSal function is created to calculate net salary

cout<<"Manager Salary: "<<perfBonus+this->basic\_salary<<endl;

}

};

class Worker:public Employee{

private: int hoursWork;

double hourlyRate;

public:

Worker(){

hoursWork=25;

hourlyRate=00.00;

}

Worker(int id,string name,int deptId, double basic\_salary,int hoursWork, double hourlyRate):Employee(id,name,deptId,basic\_salary){ //parameter constructor of manager class

this->hoursWork=hoursWork;

this->hourlyRate=hourlyRate;

}

void Input(){ //Input function is created to take bonus as input

Employee::Input();

cout<<"Hours Work: ";

cin>>hoursWork;

cout<<"Hourly Rate: ";

cin>>hourlyRate;

}

void display(){ //Display function is created to print Output

Employee::display();

cout<<"Hours Work: "<<hoursWork<<endl;

cout<<"Hourly Rate: "<<hourlyRate<<endl;

}

void calNetSal(){ //calNetSal function is created to calculate net salary

cout<<"Woker Salary: "<<this->basic\_salary+(hoursWork\*hourlyRate)<<endl;

}

};

int main1(){

Manager \*m;

Worker \*w;

cout<<"1.Hire Manager 2.Hire Worker 3.Display Info 4.exit "<<endl;

int ch,index=0,id,deptId,hoursWork,i=0;

string name;

double basic\_salary,perfBonus,hourlyRate;

Employee\* hire[5];

do{

cout<<"\nEnter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

if(index<2){

m=new Manager;

m->Input();

hire[index]=m;

index++;

}

else{

cout<<"No Vacant Place for placement!"<<endl;

}

break;

case 2:

if(index<2){

w=new Worker;

w->Input();

hire[index]=w;

index++;

}

else{

cout<<"No vacant place for placement!"<<endl;

}

break;

case 3:

for(i=0;i<index;i++){

cout<<"\n---------------------Details of Employee Hired---------------"<<endl;

hire[i]->display();

hire[i]->calNetSal();

}

break;

case 4:

cout<<"Exiting.."<<endl;

break;

default:

cout<<"Invalid Choice!";

break;

}

}while(ch!=4);

}

A screenshot of a computer

Description automatically generated

**2:Create cpp application for bank account handling.**

**2.1. Create a class BankAccount -- acct no(int),customer name(string),balance(double)**

**Add constr. (2 constrs : first to accept all details )**

**2.2 Add Business logic methods**

**Methods**

**public void withdraw(double amt)**

**public void deposit(double amt)**

**2.3: Create object of account class and test withdraw and deposit methods.**

#include<iostream>

#include<string>

using namespace std;

class BankAccount{

private: int acct\_no;

string cust\_name;

double bal;

public:

BankAccount(){

acct\_no=000;

cust\_name="NO NAME";

bal=0.0;

}

BankAccount(int acct\_no,string cust\_name,double bal){

this->acct\_no=acct\_no;

this->cust\_name=cust\_name;

this->bal=bal;

}

void Withdraw(double draw){

if(bal>=0){

bal=bal-draw;

cout<<"Remainig Balance: "<<bal<<endl;

}

else if(draw>bal){

cout<<"Your account not have sufficient balance"<<endl;

}

else{

cout<<"Insufficient Fund!"<<endl;

}

}

void Deposit(double depo)

bal=bal+depo;

cout<<"Balance after deposit: "<<bal<<endl;

}

};

int main2(){

int acct\_no,ch;

string cust\_name;

double bal,depo,draw;

cout<<"Enter the details of your bank account"<<endl;

cout<<"Enter the Account Number:"<<endl;

cin>>acct\_no;

cout<<"Enter the Customer Name: "<<endl;

cin>>cust\_name;

cout<<"Enter the Balance of your account: "<<endl;

cin>>bal;

BankAccount bObj(acct\_no,cust\_name,bal)

do{

cout<<"Enter your choice: "<<endl;

cout<<"1 Withdraw 2 Deposit 3 Check balance 4 exit"<<endl;

cin>>(ch);

switch(ch){

case 1:

cout<<"Enter the amount You want to Withdraw: "<<endl;

cin>>draw;

bObj.Withdraw(draw);

cout<<"Amount Withdrawed Successfully"<<endl;

break;

case 2:

cout<<"Enter the amount You want to deposit: "<<endl;

cin>>depo;

bObj.Deposit(depo);

cout<<"Amount Deposited Successfully"<<endl;

break;

case 3:

cout<<"Balance of "<<acct\_no<<" "<<"is: "<<bal<<endl;

break;

case 4:

cout<<"Exiting..."<<endl;

break;

default:

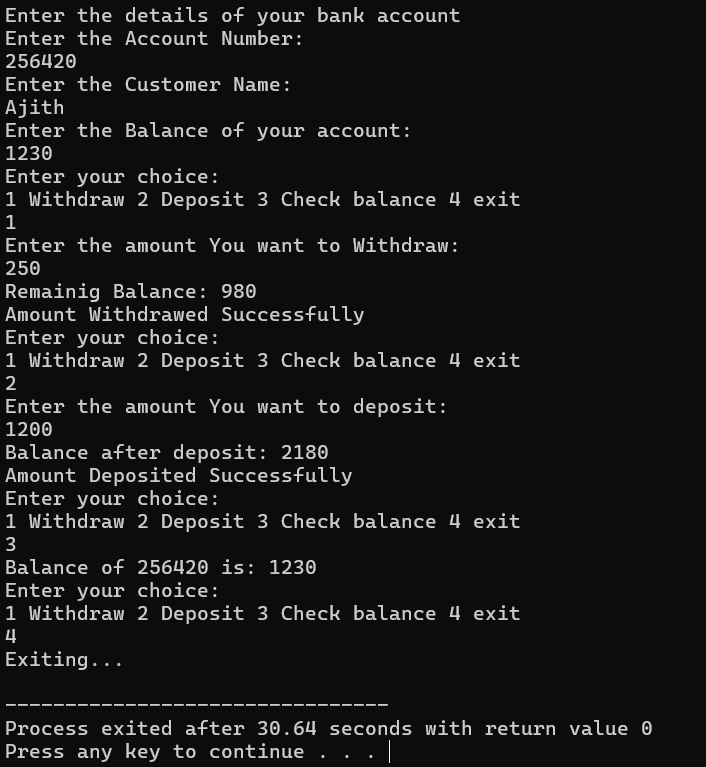
cout<<"Invalid choice!"<<endl;

break;

}

}while(ch!=4);

}



**Create a abstract class Shape with pure virtual method area; Create Rectangle,Circle,Square class..inherit them from Shape class..Override area method. Test these all classes by creating object of respective class.**

#include<iostream>

#define PI 3.14

using namespace std;

class Shape{

public:

virtual void Calarea()=0;

};

class Circle:public Shape{

private: double r;

public:

Circle(){

r=5;

}

Circle(double r){

this->r=r;

}

void Calarea(){

cout<<"Area of Circle is: "<<PI\*r\*r<<endl;

}

};

class Rectangle:public Shape{

private: double l,b;

public:

Rectangle(){

l=5;

b=8;

}

Rectangle(double l,double b){

this->l=l;

this->b=b;

}

void Calarea(){

cout<<"Area of Rectangle is: "<<l\*b<<endl;

}

};

class Square:public Shape{

private: double a;

public:

Square(){

a=5;

}

Square(double a){

this->a=a;

}

void Calarea(){

cout<<"Area of Square is: "<<a\*a<<endl;

}

};

int main3(){

Circle cObj(5);

cObj.Calarea();

Rectangle rObj(8,9);

rObj.Calarea();

Square sObj(3);

sObj.Calarea();

}

A black screen with white text

Description automatically generated

**LAB 6:**

#include<iostream>

using namespace std;

class Stack{

private:int\* arr;

int top;

public:

Stack(){

arr = new int[5];

top=-1;

}

int Push(int ele){

if(top<5){

top++;

arr[top]= ele;

}

else{

throw 20;

}

}

int Pop(){

if(top<0){

throw 30;

}

else{

arr[top--];

}

}

void display(){

int i =0;

while(i<=top){

cout<<arr[i]<<" ";

i++;

}

cout<<endl;

}

};

int main(){

try{

Stack sobj;

cout<<"Elements in stack are: "<<endl;

sobj.Push(3);

sobj.Push(5);

sobj.display();

sobj.Pop();

sobj.display();

sobj.Pop();

sobj.Push(5);

sobj.Pop();

sobj.Push(5);

sobj.Pop();

sobj.Push(10);

sobj.Push(20);

sobj.display();

//sobj.Push(6);

//sobj.Push(5);

//sobj.Push(4);

//sobj.Push(3);

//sobj.Push(2);

//sobj.display();

//sobj.Push(1);

}

catch( int no){

if(no==20){

cout<<"Stack is full"<<endl;

}

if(no==30){

cout<<"Stack is empty"<<endl;

}

}

}

A black screen with white text

Description automatically generated

**LAB 7:**

**Diamond Inheritance of Employee, Manager,SalesPerson,SalesManager**

#include<iostream>

#include<string>

using namespace std;

class Employee{

public:int id;

string name;

double salary;

public:Employee(){

id=0;

name="NA";

salary=100.00;

}

Employee(int id,string name,double salary){

this->id=id;

this->name=name;

this->salary=salary;

}

void display(){

cout<<"ID: "<<this->id<<endl;

cout<<"Name: "<<this->name<<endl;

cout<<"Salaray: "<<this->salary<<endl;

}

};

class Manager: virtual public Employee{

protected:string dname;

double incentives;

public:Manager(){

dname="No department assigned";

incentives=0.00;

}

Manager(int id,string name,double salary,string dname,double incentives):Employee(id,name,salary){

this->dname=dname;

this->incentives=incentives;

}

void display(){

cout<<"Department Name: "<<this->dname<<endl;

cout<<"Manager Total Salary: "<<this->salary+incentives<<endl;

}

};

class SalesPerson: virtual public Employee{

protected: double pa,fa,ta;

public:SalesPerson(){

pa=25.25;

fa=20.00;

ta=12.50;

}

SalesPerson(int id,string name,double salary,double pa,double fa, double ta):Employee(id,name,salary){

this->pa=pa;

this->fa=fa;

this->ta=ta;

}

void display(){

cout<<"Pa: "<<this->pa<<endl;

cout<<"Fa: "<<this->fa<<endl;

cout<<"Ta: "<<this->ta<<endl;

cout<<"Sales Person Salary: "<<this->salary+pa+fa+ta<<endl;

}

};

class SalesManager:public Manager,public SalesPerson{

private: string position;

public:

SalesManager(){

position="Position Not Assigned!";

}

SalesManager(int id,string name,double salary,string dname,double incentives,double pa,double fa,double ta,string position):Employee(id,name,salary),Manager(id,name,salary,dname,incentives),SalesPerson(id,name,salary,pa,fa,ta){

this->position=position;

}

void display(){

cout<<id<<" "<<name<<" "<<salary<<" "<<dname<<" "<<incentives<<" "<<pa<<" "<<fa<<" "<<ta<<" "<<position<<endl;

}

};

int main(){

Employee emp1;

emp1.display();

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

SalesManager sm1(012,"Ajit",5000.5,"CSE",1500,500,400,300,"Senior");

sm1.display();

// Employee emp(5,"Gauraw",50000);

// emp.display();

}

A screen shot of a computer

Description automatically generated