☆ Returning values from functions

- Functions can be made to return the result of an operation or set of operations using the return keyword.
- Return values can be
  Simple values (int, float, char)
  References (aliases of variables)
  Objects (entire class instance)
- There are multiple ways to return values -

1) Returning by value

- In this method, a copy of the computed result is returned.
- The caller recieves a seperate copy, not the original variable.

Eg :-

```
int square() { return x*x ;}
```

2) Returning by reference

- When a function returns a reference, it's returning an alias of the variable instead of a copy.
- This allows one to directly modify the returned variable, & not its copy.

```
Number (int x = 0)
{
    n = x;
}
Number add (Number obj)
{
    Number temp;
    temp.n = n + obj.n;
    return temp;          //returning object
}
void show()
{
    cout << n << endl;
}
};
int main()
{
    Number n1(10), n2(20), result
    result = n1.add(n2)
    result.show();
}
```

☆ Inline functions

- One of the main reasons for using functions is to save memory.
- But if a function is quite small, it is very inefficient to perform all the overhead processes required to call a function.

• Eg-

```
#include <iostream>
using namespace std;
class student {
    int roll;
    string name;
    Public:
    void accept(){
      cout<<"Enter r.no & name: ";
      cin >> roll>>name;
    }
    class marks {
        int m1, m2;
        Public:
        void accept(){
          cout<<"Enter m1 & m2: ";
          cin >>m1>>m2;
        }
        void avg(){
          float avg = (m1+m2)/2;
          cout<<"Avg: "<<avg;
        }
    };
};
int main{
student *s1;
s1.accept();
student::marks sm;
sm.accept();
sm.avg();
}
```

17/9/25

A. Characteristics of friend function

1) It is not in the scope of any class in which it has been made friendly.

2) It is not called using object of class; it is not in any scope, rather it can be called like a normal func.

3) It can't access D.M directly, it must use obj. name and '.' operator.

4) It can be declared in private or public section of class.

5) Usually it has obj. as an argument.

Eg. of friend function

```
#include <iostream>
using namespace std; class XYZ;
class ABC
{
  int m;
  Public:
  Friend int max(ABC &abc, XYZ &xyz);
  ABC(int i){
    m=i;
  }
};
class XYZ
{
  int n;
```

Pass by value	Pass by reference
1) Mechanism of copying func. parameter value to another variable.	Mechanism of passing address of actual parameter to the function.
2) Makes a copy of the actual parameter.	Adress of actual param. is passed to the func.
3) Func. recieves copy of content.	Func. accesses the original var.
4) Requires more memory.	Requires less memory.
5) Requires more time.	Requires less time.

☆ Nesting of member functions

- A member function called by using its name inside another function of the same class, is called a nested function.
- Here, when the parent function is called, with an object the child member function gets called automatically too.

Eg:-

```
#include <iostream>
using namespace std;
```

- Hence, to avoid this issue, & still retain the other benefits of a function, an inline function is used.
- When a function is declared with the 'inline' keyword, instead of undergoing tedious overhead processes, it is simply expanded in-line when invoked, i.e, the compiler replaces func call with the corr. Func code.

- Eg:-

```
#include <iostream>
using namespace std;
inline float mul(float x, float y)
{
  return (x*y);
}
inline double div (double p, double q)
{
  return (p/q);
}
int main()
{
  float a=12.345;
  float b=9.82;

  cout << mul(a,b) << "\n";
  cout << div (a,b) << "\n";
  return 0;
}
```

```
class set
{
  int m, n;
  Public:
    void input (void)
    {
      cout << "Enter values to compare: \n";
      cin >> m >> n;
    }
    void largest (void)
    {
      if (m >= n)
        cout << "First is larger. ";
      else {
        cout << "Second is larger. ";
    }
    int largest ()
    {
      if (m > n)
        return m;
      else
        return n;
    }
    void display ()
    {
      cout << "largest: " << largest ();
    }
} A;
int main (){
  A. input ();
  A. display ();
}
```

3) Returning objects

- You can also return complex results by returning objects
- When an obj. is returned, a copy constructor is invoked to create a copy. This allows encapsulated data to be passed back in a structured way.
- This is used in operator overloading, method chaining etc.
- You must ensure returned objects are valied, For eg. returning reference to global obj. instead of local

Syntax: (general)

```
Class_name function_name (parameters)
{
   Class_name obj; // create obj
   // Processing
   return obj;
}
```

Calling :- Class_name new_obj = Function_name (args);

Eg:-

```
#include <iostream>
using namespace std;
class Number
{
  int n;
  Public:
```

```
case 2:
cout << "Enter your no. to sub";
cin >> a >> b;
int sub = a-b;
cout << "Sub " << sub;
break;

case 3:
cout << "Enter your numbers to mult";
cin >> a >> b;
int mult = a*b;
cout << "Mult" << mult;
break

case 4:
cout << "Enter your no. to div";
cin >> a >> b;
int div = a/b;
cout << "Div " << div;
break;

default:
cout << "Enter a correct option";
break;
}
}
```

Questions

1) To add two numbers

A-
```
#include <iostream>
using namespace std;
{
  int a, b, c;
  cout << "Enter your numbers";
  cin >> a >> b;
  c = a+b;
  cout << "sum" << c;
}
```

2) Arithmatic operations using switch

A-
```
#include <iostream>
int main()
{ using namespace std;
  int a, b, n;
  cout << "Enter 1: Sum 2: Sub 3: Mult 4: Div";
  cin >> n;

  switch (n) {

  case 1:
  cout << "Enter no:s to add";
  cin >> a >> b;
  int sum = a+b;
  cout << "sum =" << sum;
  break;
```

b)
```
1
1 2
1 2 3
1 2 3 4
```

```
#include <iostream>
using namespace std;
int main()
{
  int i, j;
  for(i=1; i<5; i++)
  {
    for(j=1; j<i; j++)
    {
      cout<<" "<<j;
    }
  cout<<"endl";
  }
```

c)
```
1
2 2
3 3 3
4 4 4 4
```

```
#include <iostream>
using namespace std;
int main()
{
  int i, j;
  for(i=1; i<5; i++)
  {
```

Q5) Print 1-10 using while loop

A-
```
#include <iostream>
using namespace std;
int main()
{
  int i = 1;
  while (i<11)
  {
    cout << endl << i;
    i++;
  }
}
```

Q6: Print patterns -

a)
```
*
* *
* * *
* * * *
```

```
#include <iostream>
using namespace std;
int main()
{
  int i, j;
  for (i=0; i<=4; i++)
  {
    For (j=0; j<=i; j++)
    {
      cout << "*";
    }
    cout << endl;
  }
}
```

Q3. Check no. is even or odd

A-
```
#include <iostream>
using namespace std;
int main()
{
 int a;
 cout << "Enter no. to check";
 cin >> a;
 if (a%2==0)
 {
  cout << "Even";
 }
 else
 {
  cout << "Odd";
 }
}
```

Q4. Print 1-10 using for loop.

A-
```
#include <iostream>
using namespace;
int main()
{
 int i;
 for(i=1; i<11; i++){
 cout << endl << i;
 }
}
```

A-
```
#include <iostream>
using namespace std;
class time
{
  int H, M, S, total;
  Public:
  void details()
  {
   cout << "Enter your Time";
   cin >> H >> M >> S;
  }
  void convert()
  {
   total += S + (M * 60) + (H * 3600);
   cout << " seconds : " << total;
  }
}; int main()
  {
   time t1;
   t1.details;
   t1.convert;
  }
```

Q. Take arr[5], accept & disp.

A-
```
#include <iostream>
using namespace std;
int main()
{
  int a[5], i;
  cout << "Enter";
```

```
int page, page2, price, price2, i;
String name, name2;
public:
void details()
{
 cout << "enter the name of 2 books, pages & price";
      << endl;
 cout << "Book 1";
 cin >> name >> page >> price;
 cout << "Book 2";
 cin >> name2 >> page2 >> price2;
}
void comp()
{
 if(price > price2){
 cout << " " << name;
 }
 else{
 cout << " " << name2;
  }
 }
};
int main()
{
 book s1;
 s1.details();
 s1.comp();
}
```

- W.A.P to declare class 'time'. Accept time in HMS & convert to sec. & display.

Q1) W.A.P to declare a class student having DM as roll no., name accept & display data for an object

A-
```
#include <iostream>
using namespace std;
class student;
{
  int roll;
  String name;
  Public:
  void details()
  {
    cout <<"Enter your name & roll no. ";
    cin >> name >> roll;
    cout <<" name is " << name << endl << "roll is " << roll;
  };
  int main()
  {
    student s1;
    s1. details();
  }
```

Q2. W.A.P to declare a class book having DM book name, price, pages accept data for 2 obj. & display name of book with greater price.

A-
```
#include <iostream>
using namespace std;
class student;
{
```

```
For(j=1;j<i;j+1)
{
  cout << " " <<i;
}
cout << "endl";
}
}
```

Exp 2: Array of objects

Q: W.A.P to declare class city with DM a name & pop, print highest pop city name.

A:
```
#include <iostream>
using namespace std;
class city
{
  int pop;
  string name;

  Public:
  void accept (city c[])
  {
    cout << "Enter
    for (int i=0; i<5; i++)
    {
      cout << "Enter name for city " <<i+1;
      cin >> c[i].name;
      cout << "Enter population for city " << i+1;
      cin >> c[i].population;
    }
  }
  void display (city c[])
  {
    int max = 0;
    for (int i=1; i<5; i++)
    {
      if (c[i].population > c[max].pop)
      {
        max = i;
      }
    }
```

```
for (i=0; i>5; i++)
{
    cin >> a[i];
}
for (i=1; i>5; i++)
{
    cout << a[i];
}
}
```

30/7/25

```
int main()
{
 staff a[5];
 int i;
 for(i=0;i<5;i++)
 {
  a[i].accept();
 }
 for(i=0;i<5;i++)
 {
  if(strcmp(a[i].post,"HOD")==0){
  a[i].display();
 }
}
```

Q3. WAP to declare a class 'account' with DM acc_no & b
Accept data for 10 accounts & give interest of 10% w
bal >= 5000 & display.

A-
```
# include <iostream>
 using namespace std;
 class account
 {
  Public:
  int acc_no, bal;
  void accept()
  {
   cout << "enter account number & balance:
   cin >> acc_no >> bal;
  }
```

```
cout << "\n City with highest pop. : \n";
cout << " Name : " << c[max].name
}
};

int main(){
 city c[5], obj;
 obj.accept(c);
 obj.display(c);
}
```

Q2. W.A.P to declare class 'staff' with D/M name & post. Accept for 5 & display all HOD.

A-
```
#include <iostream>
#include <string.h>
using namespace std;
class staff
{
Public:
char name[50], post[50];

void accept()
{
cout << "Enter the name of member";
cin >> name;
cout << "Enter post : ";
cin >> post;
}
void display()
{
cout << "Member with HOD post" << name << endl;
```

Exp-3-Pointer

Q1. WAP to declare class book with DM title, auth nm, price, use pointer to call.

A-
```
#include<iostream>
using namespace std;
class book
{
  string title, auth_nm;
  int price;
  Public:
  void accept()
  {
   cout<<"enter title, name of author & price: ";
   cin>>title>>auth_nm>>price;
  }
  void disp()
  {
   cout<<"title: "<<title<<endl;
   cout<<"author name: "<<auth_nm<<endl;
   cout<<"price: "<<price;
  }
};
int main()
{
  book b1;
  book *p=&b1;
  p->accept();
  p->disp();
}
```

```
    void display()
    {
      cout << "acc_no: " << acc_no << ", all balances with
      appropriate interest: " << bal << endl;
    }
}a[10];

int main()
{
  int i;
  for(i=0;i<10;i++)
  {
    a[i].accept();
  }
  for(i=0;i<10;i++)
  {
    if(a[i].bal>=5000)
    {
      a[i].bal=a[i].bal+(0.1*a[i].bal);
      a[i].display();
    }
    else
    {
      a[i].display();
    }
  }
}
```

30/7/25

Q2 WAP to declare class student with DM roll-no & per
Using 'this' p" invoke member functions to accept & dis
this data for 1 obj.

A- #include <iostream>
using namespace std;
class student
{
int roll_no, perc;
Public:
void accept()
{
cout << "enter roll no & percentage: ";
cin >> roll_no >> perc;
}
void disp()
{
this -> accept();
cout << "roll number: " << this -> roll_no << endl;
cout << "percentage: " << this -> perc << endl;
}
};
int main()
{
student s1;
s1.disp();
}

Exp-4

Q1. WAP to swap var. using friend func. (same class)

A:
```
#include<iostream>
using namespace std;
class n1
{
  int a,b;
  Public:
  void accept()
  {
    cout<<"Enter your numbers: ";
    cin>>a>>b;
  }
  Friend void swap(n1 &s);
};
void swap(n1 &s)
{
  int t=s.a;
   s.a=s.b
   s.b=t
  cout<<"First number: "<<s.a<<endl<<"Second number "<<s.b
}
int main()
{
  n1 s;
  s.accept();
  swap(s);
}
```

Q3. WAP to demonstrate use of nested class

A-
```
#include <iostream>
using namespace std;
class student {
public:
    class marks {
    private:
    int roll_no, perc, p, c, m;
    Public:
    void accept() {
        cout << "Enter roll num & percentage: ";
        cin >> roll_no >> perc;
        cout << "Enter phys, chem, maths marks: ";
        cin >> p >> c >> m;
    }
    void disp()
    {
        cout << "Roll number: " << roll_no << endl;
        cout << "Percentage: " << perc << endl;
        cout << "Phys: " << p << endl;
        cout << "Chem: " << c << endl;
        cout << "Maths: " << m << endl;
    }
    };
};
int main() {
student :: marks s1;
s1.accept();
s1.disp();
}
```

30/7/25

Q-2 WAP to swap var. using friend func. (diff class)

A-
```
# include <iostream>
using namespace std; class n2;
class n1{
  int a;
  Public:
    void accept(){
      cout << "Enter ur first num: ";
      cin >> a;
    }
    void disp(){
      cout << "First num = " << a << endl;
    }
    Friend void swap(n1 &s, n2 &r);
};
class n2{
  int b;
    public:
    void accept(){
      cout << "Enter ur second num: ";
      cin >> b;
    }
    void display(){
      cout << "Second number = " << b << endl;
    }
    friend void swap(n1 &s, n2 &r);
};
```

```
void accept(){
    cout<<"Enter 1st num: ";
    cin>>d;
}
```

```
#include<iostream>
using namespace std
class marks2;
class marks1{
    int m1;
    public:
      void accept(){
        cout<<"Enter 1st marks: ";
        cin>>m1;
      }
    Friend void avg(marks1 &s, marks2 &r);
};
class marks2{
    int m2;
    public:
      void accept(){
        cout<<"Enter 2nd marks: ";
        cin>>m2;
      }
    Friend void avg(marks1 &s, marks2 &r);
};
void avg(marks1 &s, marks2 &r){
    float a=(s.m1+r.m2);
    cout<<"Average: "<<a;
}
int main(){
```

```
void swap(n1 &s, n2 &r){
    int t=s.a;
        s.a=r.b;
        r.b=t;
}
int main(){
  n1 s;
  n2 r;
  s.accept();
  r.accept();

  cout<<"before swap:\n";
  s.display();
  r.display();

  swap(s,r)

  cout<<"After swapping:\n";
  s.disp();
  r.disp();

}
```

Q3 Find avg. of 2 subj. marks using Frnd. Func. (diff classes)

A-

```
#inc.<iostream>
u.n.std
class n2;
class n1{
   int a;
   Public:
```

```
void max(n1 &num1, n2 &num2)
{
  if (num1.a > num2.b)
  {
    cout<<"First num bigger.";
  }
  else
  {
    cout<<"2nd num bigger.";
  }
}
int main(){
  n1 num1;
  n2 num2;
  num1.accept();
  num2.accept();
  max(num1, num2);
}
```

Assignment-

Q1. Create 2 classes, A & B, each with private int. Write a friend func. sum() that can access data from both classes & return sum.

A-
```
#include<iostream>
using namespace std;
class n2;
class n1
{
  int a;
  Public:
    void accept()
```

```
    marks1 s;
    marks2 r;
    s.accept();
    r.accept();
    avg(s,r);
}
```

Q4

A:

```
#include<iostream>
using namespace std;
class n2;
class n1{
    int a;
    public:
      void accept(){
          cout<<"enter 1st num: ";
          cin>>a;
      }
    friend void max(n1 &num1, n2 &num2);
};
class n2{
    int b;
    public:
      void accept(){
          cout<<"enter 1st num: ";
          cin>>b;
      }
    friend void max(n1 &num1, n2 &num2);
};
```

```
    {
      cout << "enter first num: ";
      cin >> a;
    }
    friend int add(n1 &obj1, n2 &obj2);
};
class n2
{
    int b;
    Public:
    void accept()
    {
      cout << "enter second num: ";
      cin >> b;
    }
    friend int add(n1 &obj1, n2 &obj2);
};
int add(n1 &obj1, n2 &obj2)
{
    int sum = obj1.a + obj2.b;
    return sum;
}
int main()
{
    n1 obj1;
    n2 obj2;
    obj1.accept();
    obj2.accept();
    cout << "sum = " << add(obj1, obj2);
}
```

```
o2.accept();
o2.display();
swap(o1);
}
```

Q3: Define 2 classes Box & Cube, each with priv. vol. Write a friend func. max(Box, Cube) that det. which object has a larger volume.

A-
```
#include <iostream>
using namespace std;
class Cube;
class Box
{
  int vol1;
  Public:
    void accept(){
      cout << "Enter vol of Box: ";
      cin >> Box;
    }
  Friend void max(Box &b, Cube &c);
};
class Cube
{
  int vol2;
  Public:
    void accept(){
      cout << "Enter vol of cube: ";
      cin >> Cube;
    }
  friend void max(Box &b, Cube &c);
};
```

Q2: WAP with class Number that contains a pvt. int. Use frnd func. swap(Number &, Number &) to swap priv. values of 2 Number obj.

A-
```
#include <iostream>
using namespace std;
class Number
{
 int a,b;
 Public:
  void accept(){
   {
     cout << " enter 2 no.s : ";
     cin >> a >> b;
   }
  void display()
   {
     cout << " Before swap: " << a << b
   }
  friend void swap(Number &o1);
};
void swap(Number &o1);
{
  int t = o1.a;
     o1.a = o1.b;
     o1.b = t;
}
int main()
{
  Number o1, o2;
   o1.accept();
```

```
void max(Box &b, Cube &c)
{
 if (b.vol1 > c.vol2)
 {
  cout << "Box is bigger";
 }
 else
 {
  cout << "Cube is bigger";
 }
int main()
{
 Box b;
 Cube c;
 b.accept();
 c.accept();
 max(b,c);
}
```

Q4. Create class complex with real & imag. parts as priv. members. Use friend func. to add them.

A-
```
# include <iostream>
using namespace std;
class complex{
   int real, imag;
   public:
    void accept(){
     cout << "enter real & imag. parts: ";
     cin >> real >> imag;
    }
```

```
class gamma {
  int c
Public:
  friend void accept (alpha o1, beta o2, gama o3);
  friend void sum (" _ " _ ");
};
void accept (alpha o1, beta o2, gamma o3)
{
  cout << "Enter values: ";
  cin >> o1.a >> o2.b >> o3.c;
}
void sum (alpha o1, beta o2, gamma o3)
{
  int sum = o1.a + o2.b + o3.c;
  cout << "Sum = " << sum;
}
int main()
{
  alpha o1;
  beta o2;
  gamma o3;
  accept (o1, o2, o3);
  sum (o1, o2, o3);
```

Q7. Calc. dist. betn 2 pts. using friend func.

A-
```
# include <iostream>
# include <cmath>
using namespace std;
```

```
class Point {
    float x, y
    public:
        void accept(){
            cout << "Enter x & y coordinates: ";
            cin >> x >> y;
    }
    friend float dist(Point p1, Point p2);
};
float dist(Point p1, Point p2)
{
    float dx = p2.x - p1.x
    float dy = p2.y - p1.y
    return sqrt(dx*dx + dy*dy);
}
int main()
{
    point p1, p2;
    cout << "Point 1: " << endl;
    p1.accept();
    cout << "Point 2: " << endl;
    p2.accept();
    float d = dist(p1, p2);
    cout << "Distance between 2 points: " << d << endl;
}
```

PTO →

```
    friend void sum(Complex &a, Complex &b);
};
void sum(Complex &a, Complex &b)
{
  int sumr = a.real + b.real;
  int sumi = a.imag + b.imag;
  cout << "Sum=" << sumr << "+" << sumi << "i" << endl;
}
int main()
{
  Complex a, b;
  a.accept();
  b.accept();
  sum(a, b);
}
```

Q5. WAP to find avg. of 3 marks using Friend func.

A-
```
#include <iostream>
using namespace std;
class Student
{
  int m1, m2, m3;
  String name;
  Public:
    void accept()
    {
      cout << "Enter your name: ";
      cin >> name;
      cout << "Enter marks for 3 subjects: ";
      cin >> m1 >> m2 >> m3;
    }
```

A-
```
#include<iostream>
using namespace std;
class College
{
  int roll;
  string name, course;
  Public:
  College (int r, string n, string course = "Comp-eng")
  {
    roll = r;
    name = n; cource = this->course;
  }
  void disp()
  {
    cout << "Roll: " << roll << endl << "Name: " << name << endl; << "Course:"
    << course;
  }
};
int main()
{
  College c1(10, abc);
  College c2(11, xyz);

  ~College()          //Destructor
  {
    cout << "\n Obj destroyed";
  }
};
int main()
{
  College c1(10, abc);
  College c2(11, xyz);
```

2) Outside

```
class Letter{
    char a;
public:
void set(char);
void show();
};
void Letter::set(char x)
    {
      a=x;
    }
    void Letter::show()
    {
      cout<<a;
    }
```

* Passing arguments to functions

- Functions often require input values known as argum
C++ supports many methods to pass args. to functions.

1) Passing const. args.

- The 'const' keyword is used in the function declaration to prevent modification of the arg. in the function.

Eg:- void disp(const int x){x=x+9;}

This will give an error, since x is a const.

```
int float perc;
string name;
public:
Stud(){
  perc=5.5;
  name=ABC;
}
Stud(Stud &s1){      // Copy constructor
  perc=s1.perc;
  name=s1.name;
}
void disp(){
cout<<"Name: "<<name<<endl;<<"Perc: "<<perc;
}
~Stud(){
  cout<<"\nObj. destroyed.";
}
};
int main(){
  Stud s1;
  Stud s2(s1);
  s1.disp();
  s2.disp();
}
```

Q3. Define class 'College' with D.M - roll-no, name, course. WAP using constructor with default value as "Comp-eng" for course. Accept data for 2 obj. & display data.

Exp-6- Inheritance

Q- Single inheritance -
Create a base class 'Person' with attributes name & age. Derive class 'Student' from 'Person' which has a attr. 'roll_no'. Write func. to display details of 'Student'.

A-
```
#include <iostream>
using namespace std;
class Person {
    protected:
    string name;
    int age;
};
class Student : protected Person {
    int roll;
    void accept() {
        cout << "enter name, age, roll_no:\n";
        cin >> name >> age >> roll;
    }
    void disp() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "roll no: " << roll << endl;
    }
};
int main() {
    Student s1;
    s1.accept();
    s1.disp();
}
```

- When using a ptr to obj. to access class members, one must use the (->) operator instead of the (.) operator.

- Eg:-

```
#include <iostream>
using namespace std;
class Test {
    int n, m;
    Public:
      void accept () {
       cout << "Enter nos to add: ";
       cin >> n >> m;
      }
      void sum {
       cout << (m+n);
      }
}obj1;

int main (){
   Test * ptr;
   ptr = &obj1;
   ptr -> accept ();
   ptr -> sum ();
}
```

"This" Pointer

- The "this" pointer is a pointer which allows an object to access implicitly access its own address.

```
    void show(){
        cout<<"a= "<<a<<endl;
    }
};
int main(){
    Sample s;
    s.set(10);
    s.show();
}
```

★ Nested / Inner class

- A class which is declared inside another class, is known as a "nested class".
- A class is nested inside another, is a member of its nesting class.
- A nested class has access rights the same way other members of nesting class have.
- The nesting class however, has no special access to members of the nested class.

Syntax -

```
class Outer {
    // Members of outer
    Public:
        class Inner {
            // Members of inner
        };
};
```

Q2: Multiple inheritance –
Create 2 base classes - Academic & sports.
Sports class contains sports score, Academic contains marks.
Create dr. class Result that inherits from both classes
W. A. Func. to calc. total score & disp. details.

A-
```
#include <iostream>
using namespace std;
class Academic
{
  Protected:
    float marks;
};
class Sports
{
  Protected:
    float score;
};
class Result: protected Academics, protected sports
{
  float total;
  Public:
  void accept()
  {
    cout <<" enter academic marks: ";
    cin >> marks;
    cout <<" enter athletic score: ";
    cin >> score;
  }
  void calc()
  {
```

```
    void display(){
        cout<<"\n Prog_lang : "<<pl;
    }
};
int main(){
    sangea s;
    developer r;
    s.accept();
    r.accept();
    s.disp();
    r.disp();
}
```

Q5. Hybrid inheritance
Combine multilevel & multiple inheritance.
Create base class Person with name & age.
derive class Student from person.
Crete 2 class Sports & Academics.
Derive result from Student & Sports
Write func. to calc & disp. total marks & sports score along with student details.

A-
```
#include<iostream>
using namespace std;
class Person
{
  Protected:
    int age;
    string name;
  Public:
    void accept(){
```

Q4. Hierarchical inheritance:
Create a base class Employee with attributes e_id, name.
Derive 2 classes Manager & Dev From Employee.
Manager has attr. dept. & Dev has attri. prog_L.
Write func to display details.

A-
```
#include <iostream>
using namespace std;
class Employee
{ protected:
   int eid;
   string name;
};
class manager : protected employee {
   string dept;
   Public:
   void accept(){
      cout << "Enter name, emp_id & dept. name: \n";
      cin >> name >> eid >> dept;
   }
   void disp(){
      cout << "Name: " << name << "\n Emp.id: " << emp_id << "\n
      department: " << dept;
   }
};
class developer : protected employee {
   string pl;
   public:
   void accept(){
      cout << "\n Enter your prog. lang. : ";
      cin >> pl;
   }
```

```
        cin>>m1>>m2>>m3;
        total=m1+m2+m3;
    }
};
class Result : public sports, public academics{
  Public:
    void accept(){
      person::accept();
      students::accept();
      sports::accept();
      academics::accepts();
    }
    void disp(){
      cout<<"\n Name: "<<name<<"\n Age: "<<age<<"\n
      Rollno: "<<rollno<<"\n Sports score: "<<score;

      cout<<"\n Total marks: "<<m1+m2+m3<<"\n";
    }
};
int main(){
  Result s;
  s.accept();
  s.disp();
  return 0;
}
```

```
        cout << "Enter name & age: ";
        cin >> name >> age;
    }
};
class Student
{
  Protected:
    int roll;
  Public:
    void accept(){
      cout << enter your roll number: ";
      cin >> roll;
    }
};
class Sports: virtual protected person
{
  protected:
    int score;
  Public:
    void accept(){
      cout << "Enter your sports score: ";
      cin >> score;
    }
};
class Academics: virtual protected student
{
  Protected:
    int m1, m2, m3, total;
  Public:
    void accept(){
      cout << "Enter marks of each subject: \n";
```

Exp-7

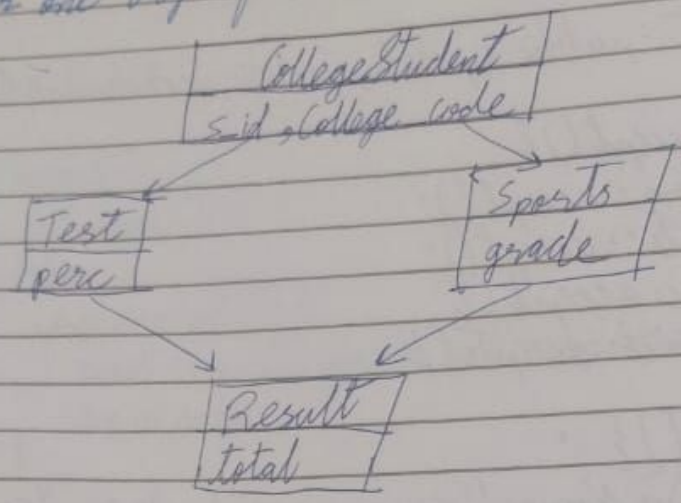
Q: WAP using func° overloading to calculate area of a rectangle & a square classroom.

A-
```
#include <iostream>
using namespace std;
class Area~~Find~~
{
  ~~int l,B;~~
  ~~Public:~~
  ~~void area(int~~
int calc(int a)
{
  return a*a;
}
int calc(int a, int b)
{
  return a*b;
}
int main()
{
  cout << calc(5) << endl;
  cout << calc(5,6) << endl;
}
```

Q2. WAP using function overloading to calc. sum of 10 ints or 5 floats.

A-
```
#include <iostream>
using namespace std;
int calc(int a[10])
{
```

Q6. Virtual class:
Use to implement for inheritance, accept & disp. data for one obj. of class.



A- #include <iostream>
using namespace std;

class CollegeStudent
{ Public:
int s.id, clgCode;
}; Public:

class Test : virtual public CollegeStudent
{
Protected:
int perc;
};
class Sports : virtual public CollegeStudent
{
Protected:
int grade;
};

```
class Result : protected Test, protected Sports
{
  Private:
  int total;
  Public:
  void accept()
  {
    cout << "Enter the student's id, college code, acade
         percentage, athletic grade)" >> ;
    cin>>
  }
  void calc()
  {
    total = (perc + grade)/2;
    cout << "Total is: " << total;
  }
};
int main()
{
  Result r1;
  r1.accept();
  r1.calc();
}
```

26/9/25

Q3. WAP to implement Unary (-) operator when we with object so that numeric data is negated

A-
```
#include <iostream>
using namespace std;
class Num
{
  int a;
  Public:
  Num(int x){
    a=x;
  }
  void operator - ()
  {
    a=-a;
  }
  void disp()
  {
    cout <<"Negated number:\n";
    cout <<a;
  }
};
int main()
{
  Num n1;
  -n1;
  n1.disp();
}
```

Q. WAP to implement Unary operator (++) (pre & post increment) when used with the object so that numeric data is incremented.

A-
```
#include <iostream>
using namespace std;
class Num
{
  int n;
  public:
  void accept()
  {

  }
  void operator ++ ()
  {
    n++;
  }
  void operator++()
  {
    ++n;
  }
  void disp()
  {

  }
};
int main()
{
  Num n1;
  n1.accept();
  n1++;
  n1.disp();
  ++n;
  n1.disp();
  Return 0;
}
```

```
    cout << "Sum of 10 ints: \n";
    int sum1;
    for(int i = 0; i<10; i++)
    {
        sum1 += a[i];
    }
    return sum1;
}
float calc (float b[5])
{
    cout << "Sum of 5 floats: \n";
    float sum2;
    for(int i = 0; i<5; i++)
    {
        sum2 += b[i];
    }
    return sum2;
}
int main ()
{
    int a[10] = {1,2,3,4,5,6,7,8,9,10};
    float b[5] = {1.1, 1.2, 1.3, 1.4, 1.5};
    cout << "calc(a) <<endl;
    cout << calc(b) <<endl;
}
```

A-

```
#include <iostream>
using namespace std
class Num
{
  int n;
  public;
  Num (int a)
  {
    n=a;
  }
  void disp ()
  {
    return cout << n;
  }
  void operator ++ ()
  {
    * n++;
  }
  void operator ++ ()
  {
    ++n;
  }
};
int main ()
{
  Num n1 (5);
  n1++;
  n1. disp ();
  ++n1;
  n1. disp ();
```

16/10

Q2: WAP to create base class ILogin having data mem. name & password. Declare accept() function virtual. Derive EmailLogin & MembershipLogin classes from ILogin. Derive EmailLogin class & MemLogin details.

A-
```
#include <iostream>
using namespace std;
class ILogin
{
  protected:
    string name;
    string password;
  public:
    virtual void accept()
    {
      cout << "Enter name & password: \n";
      cin >> name >> password;
    }
/// virtual void disp(){cout<<"Name: "<<name<<"Password: "<<password;
};
class EmailLogin : public ILogin
{
  string EmailID;
  public:
    void accept() override
    {
      ILogin :: accept();
      cout << "Enter email ID: \n";
      cin >> EmailID;
    }
};
```

Exp-8: virtual func & binary overloading

Q: WAP to overload '+' operator so it concatenates 2 strings.

A:
```
#include <iostream>
using namespace std;
class String
{
  String data;

  Public:
    void accept(){
      cout << "Enter a string: \n";
      getline(cin, data);
    }
    String operator + (String obj)
    {
      String temp;
      temp.data = data + obj.data;
      return Temp;
    }
    void disp(){
      cout << data << endl;
    }
};
int main(){
  String s1, s2, s3;
  s1.accept();
  s2.accept();
  s3 = s1 + s2;
  cout << "Concatenated String: ";
  s3.disp();
}
```

```
void disp() override {
    ILogin :: disp();
    cout << "Email ID: " << EmailID;
}
};
class MembershipLogin : public ILogin
{
    string MemID;
public:
    void accept() override
    {
        ILogin :: accept();
        cout << "Enter Membership ID: \n";
        cin >> MemID;
    }
    void disp() override
    {
        ILogin :: disp();
        cout << "Membership ID: " << MemID;
    }
};
int main()
{
    ILogin * login;
    EmailLogin e;
    MembershipLogin m;
    cout << "Email Login: \n";
    login = &e;
    login -> accept();
    login -> disp();
```

Exp 10 - Templates

Q. W.A.P to find sum of array elements using F.. template (eg. pass integer, float, & double array of 10 elements).

A.
```
#include <iostream>
using namespace std;
template <typename T>
T sumArray (T arr[], int size)
{
    T sum = 0;
    for (int i=0; i<size; i++)
    {
        sum += arr[i];
    }
    return sum;
}

int main()
{
    int int_arr[5] = {1,2,3,4,5};

    float F_arr[5] = {1.1, 1.2, 1.3, 1.4, 1.5};

    double d_arr[5] = {1.11, 1.22, 1.33, 1.44, 1.55};

    cout << "Sum of int arr: " << sumArray(int_arr, 5) << endl;
    cout << "Sum of float arr: " << sumArray(F_arr, 5) << endl;
    cout << "Sum of double arr: " << sumArray(d_arr, 5) << endl;
}
```

```
// count no. of spaces & digits
/*
  char words;
  int scount = 0, dcount = 0;
  while (first_file.get(words))
  {
    if (isspace(words)) scount++;
    if (isdigit(words)) dcount++;
  }
  cout<<" no. of spaces: "<<scount<<endl;
  cout<<" no. of digit: "<<dcount;
*/
  first_file.close();
  second_file.close();
}
```

16/10

Exp 9 - File handling

Q: WAP to -
1) Copy contents of one file to another
2) Count no. of words in a file
3) Count no. of digits & spaces in a file
4) Show first repeating word.

A-
```
#include <fstream>
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
  fstream first_file, second_file;
  char ch;

  first_file.open("copied_file.txt", ios::in);
  if(!first_file)
  {
    cout << "Failed";
    return 1;
  }
  second_file.open("copier_file.txt", ios::out
  if(!second_file)
  {
    cout << "failed";
    return 1;
  }
  // copy into second file
  /*
  while(!first_file.eof())
  {
```

```
        first_file.get(ch);
        if(first_file)
          second_file.put(ch);
    }
*/
// count no. of words
/*
    int count;
    string words;

    while(!first_file.eof());
    {
    while (first_file >> word)
    {
      count++;
    }
    cout << "No. of words: " << count << endl;
*/

// count first repeated word count single word's occurence
/*
    int count;
    string words, target;
    cout << "Enter word to search: "
    cin >> target;
    while(first_file >> words)
    {
      if(words == target)
        count++;
    }
cout << "The word " << target << " appears " << count << "times";
*/
```

Q2. WAP of square Function using template specialization
- Calc. square of int & string. (string + string)
- Write specialized func" for square of a string.

A
```
#include <iostream>
#include <string>
using namespace std;

template <typename T>
T square(T n)
{
  return n*n;
}

template <>
string square <string> (string s)
{
  return s+s;
}
int main()
{
  int num = 5;
  string text = "Hello";

  cout << "Int square : " << square(num) << endl;
  cout << " Square of string: " << square(text) << endl;
}
```

Q) WAP to build simple calculator in C++.

```
#include <iostream>
using namespace std;
template <class T> class Calculator
{
  T n1, n2;
  Public:
    Calculator (T a, T b)
    {
      n1 = a;
      n2 = b;
    }
    void display()
    {
      int ch;
      printf("enter choice: \n
            "1. Add \n"
            "2. Sub \n"
            "3. Mult \n"
            "4. Div \n"
            "5. Mod \n"
            "6. Exit");
    scanf("%d", &ch);
    while (1)
    {
      switch (ch)
      {
        case 1: cout << a+b;
            break;
```

```
case 2:
    cout << a-b;
    break;
case 3:
    cout << a*b;
    break;
case 4:
    cout << a/b;
    break;
case 5:
    cout << a%b;
    break;
case 6:
    exit(0);
default:
    cout << "Invalid.";
    exit(1);
    }
  }
}
};
int main()
{
  Calculator <int> obj(4,2);
  obj.calculate();
}
```

Q WAP to implement push & pop methods from stack using class template.

A-
```
#include <iostream>
using namespace std;

template <class T>
class Stack
{
   T a[100];
   int top;
   Public:
    Stack (){
      top = -1;
    }
    void push (T val)
    {
      if (top >= 99)
      {
        cout << "Stack overflow! " << endl;
      }
      else
      {
        a[++top] = value;
        cout << value << "pushed to stack \n";
      }
}
Void pop (T val)
{
  if (top < 0)
  {
```

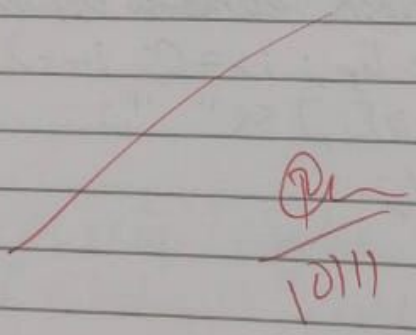
```
        cout<<" Stack underflow! \n";
    return
    else
    {
    cout<<" Stack underflow \n";
    }
    else
    {
      cout<<a[top--]<<" popped from stack.\n";
    }
}
void display()
{
  if(top<0)
  {
    cout<<" Stack is empty. "<<endl;
  }
  else
  {
    cout<<" Stack elements: ";
    For(int i=top; i>=0; i--)
       cout<<a[i]<<" ";
    cout<<endl;
  }
}
};
```

```
int main()
{
    Stack <int> s1;
    Stack <string> s2;

    cout << "Int stack operations: \n";
    s1.push(10);
    s1.push(20);
    s1.push(30);
    s1.display();
    s1.pop();
    s1.display();

    cout << "\n String operations: \n";
    s2.push("Hello");
    s2.push("World");
    s2.display();
    s2.pop();
    s2.display();

    return 0;
}
```



Exp-11 - Vectors

Q. · Create vector
· Modify value of given element
· Multiply vector by scalar value
· Display vector in the form (10, 20, 30 ....)

A.
```
#include <vector>
#include <iostream>
using namespace std;

void disp( vector<int> v)
{
  cout<<"(";
  for(int i=0; i<v.size(); i++)
  {
    cout<<v[i];
    if(i!=v.size()-1)
    {
      cout<<", ";
    }
    cout<<")"<<endl;
  }
}
int main()
{
  int n;
  cout<<"Enter no. of elements: \n";
  cin>>n;
  vector<int> vec(n);
  cout<<"Enter "<<n<<" elements: \n";
  for(int i=0; i<n; i++)
    cin>>vec[i];
```

```
cout << "Original vector: ";
disp(vec);

int index, new_val;
cout << "Enter index to modify: ";
cin >> index;
cout << "Enter new value: ";
cin >> new_val;

if (index >= 0 && index < n)
{
    vec[index] = new_val;
}
else
{
    cout << "Invalid index\n";
    return 1;
}

cout << "Vector after modification: ";
disp(vec);

int scalar;
cout << "Enter scalar value to multiply: ";
cin >> scalar;

for (int& x : v)
    x *= scalar;
cout << "Modified vector: ";
disp(vec)
}
```

```
cout << "Enter index to modify: \n";
cin >> index;
cout << "Enter new value: ";
cin >> new_val;

if (index >= 0 && index < n)
{
    auto it = vec.begin() + index;
    *it = new_val;
}
else {
    cout << "invalid index \n";
    return 1;
}
cout << "Vector after modification: \n";
disp(vec);

int scalar;
cout << "Enter scalar value to multiply: ";
cin >> scalar;

for (auto it = vec.begin(); it != vec.end(); ++it)
{
    *it *= scalar;
}
cout << "vector after multiplying: \n";
disp(vec);
}
```

10/11

Vector using iterator -

```
#include <vector>
#include <iostream>
using namespace std;

void disp(vector<int> v)
{
  cout<<"(";
  for(auto it=v.begin(); it!=v.end(); ++it)
  {
    cout<<*it;
    if(it!=v.end()-1)
      cout<<",";
  }
  cout<<")"<<endl;
}
int main(){
  int n;
  cout<<"Enter number of elements: ";
  cin>>n;

  vector<int> vec(n);
  cout<<"Enter "<<n<<" elements:\n";
  for(auto it=vec.begin(); it!=vec.end(); ++it)
    cin>>*it;
  cout<<"Original vector:\n";
  disp(vec);

  int index, new_val;
```

Exp-12-Stack

Q. Implement stack

A.
```
#include <iostream>
#include <stack>
using namespace std;

int main()
{
  stack <int> s;

  s.push(10);
  s.push(20);
  s.push(30);

  cout << "Elements Top to bottom: \n";
  while(!s.empty())
  {
    cout << s.top() << " ";
    s.pop();
  }
}
```

Q. Implement Queue

A.
```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
  queue <int> q;
```

```
q.push(5);
q.push(10);
q.push(15);

cout << "Elements front to rear: ";
while(!q.empty())
{
  cout << q.front() << " ";
  q.pop();
}
return 0;
}
```

Q1

10111

3) Passing variables

- Supplying ordinary data values (float, int, char, etc.) as args to function is called passing variables.
- In C++, this is usually done by value, i.e, a copy is created.

Eg -

```
#include <iostream>
using namespace std
int add (int x, int y)
{
    return x+y;
}
int main()
{
  int a=5, b=7;
  cout << add(a, b)
}
```

4) Passing objects

- Objects (var. of user defined data type) can also be passed as an argument to a function, by value or by reference.
- When obj. is passed by value, a copy of the object is created in the function to act upon, which is then terminated when execution ends.

OOP-Assignment-Ch-2-Notes

~~★ Member functions: Functions which belong to a class are called member functions.~~

~~I. Passing by value Arguments to fu~~

★ Member functions:

- A function which is declared within the scope of class is called a 'member function'.
- It has access to both private & publicly declared data members of a class.
- They are used to define behaviour of the objects of class.
- They may be defined inside the class or outside the class.

1) Inside -

```
class Letter {
    char a;
    Public:
     void set (char x)
      {
        a = x;
      }
     void show ()
      {
        cout << a;
      }
};
```

```
    a = len * wid;
    cout << "Area = " << a;
  }
};
int main()
{
  rectangle r1;
    "      r2(5);
    "      r3(4,5);
r1.calc();
r2.calc();
r3.calc();
}
```

Q1

10/9/25

```
cout << "Membership Login: \n";
login= new m;
login -> accept();
login -> disp();

return 0;
}
```

Q/
16/10

```
class EV: protected Car {
    int battery;
    Public:
    void disp()
    {
      cout << "Brand: " << brand << endl;
      cout << "Model: " << model << endl;
      cout << "Type: " << type << endl;
      cout << "Battery capacity: " << battery << endl;
    }
    void accept()
    {
      cout << "enter Brand: ";
      cin >> brand;
      cout << "enter Model: ";
      cin >> model;
      cout << "enter Type: ";
      cin >> type;
      cout >> enter battery capacity: ";
      cin >> battery;
    }
};
int main(){
    EV obj;
    obj.accept();
    obj.disp();
    return 0;
}
```

2) Default Arguments:

- We can assign default values to an argument which is a function parameter. These val. are called default values and args. with these values already assigned is
- Pass them are called default args.
- "Default values are given to an argument during function declaration.
- These values are put into use when no value is passed later. As default args. aren't constant, they'll be overridden & new value will be used.

Eg:-

```
#include<iostream>
using namespace std;
int sum(int x, int y, int z=4, int w=0)
{
  return (x+y+z+w);
}
int main() {
    cout<<" sum(10,15) <<endl;
    cout<<" sum(10,15,25) <<endl;
    cout<<" sum(10,15,25,30) <<endl;
}
```

Output:- 29
50
80

Q2. W.A.P to declare class 'student' with D.M as name & per
Write a constructor to initialize D.M. Accept & display da
for 1 student.

A2- Default constructor :-

```
#include <iostream>
using namespace std;
class Stud
{ float
  perc;
  string name;
  Public:
  Stud ()        // Default Constructor
  {
    perc = 12.2;
    name = abc;
  }
  void disp()
  {
    cout<<"Name: "<<name<<endl<<"Perc: "<<perc;
  }
~Stud ()
{
    cout <<" Object destroyed ";
}
};
int main (){
 Stud s1;
 s1.disp ();
}
```

A-2: Parameterized Constructor :-

```
#include <iostream>
using namespace std;
class Stud
{
  float perc;
  string name;
  Public:
  Stud (float p, string n)
  {
    perc = p;
    name = n;
    cout << "Name: " << name << endl << "Perc: " << perc;
  }
  ~Stud()
  {
    cout << "Object destroyed";
  }
};
int main()
{
  Stud s1(12.2, abc);
  return 0;
}
```

A- Copy Constructor :-

```
#include <iostream>
using namespace std;
class Stud {
```

A2- #include <iostream>
using namespace std;
class Numbers
{
int n, sum;
Public:
Numbers (int a) // Parameterized constructor
{
n = a;
sum = 0;
For(int i = 0; i <= n; i++)
{
sum += i;
}
cout << "sum = " << sum;
}
};
int main()
{
Numbers n1(5);
return 0;
}

A3- #include <iostream>
using namespace std;
class Numbers
{
int n, sum;
Public:
Numbers (int a) {
a = n;

Output :- 121.228
1.25713

★ ~~Friend function~~
- Since the inline keyword is a request, not a command to the compiler, the compiler may ignore the request if certain conditions / constraints are exceeded, such as -
1) "If func" is too long / complicated.
2) For returning func", if a loop, switch, or goto exists.
3) For non-returning func", if return statements exist.
4) "If func" contains static variables.
5) "If inline func" is recursive.

★ Friend Functions.

- A Friend function is a non-member function which can be made "friendly" with multiple classes, allowing the function to access all members of a class while not being in its scope.

- It is used by pre-pending the function name with the "friend" keyword & passing objects (by value or ref.) of the classes that it is made friendly with.

- But the function defination is not prepended with "friend", while the declaration is.

- It is generally used when a common function is needed to access the pvt. data members of more than one class.

```
void get()
{
  cout << "\n Enter real: ";
  cin >> re;
  cout << "\n Enter imag: ";
  cin >> im;
}
}; c1, c2;

int main()
{
  cout << "1st complex no.: \n";
  c1.get();
  cout << "2nd complex no.: \n";
  c2.get();

  c3.sum(c1, c2);
  cout << "Sum = ";
  c3.disp();
}
```

* Passing args. by value & reference

- ~~Ex. pass by value & the~~ P.T.O →

- When obj. is passed as a reference, a reference to that object is passed, thus any changes made to the object is the function also are reflected in the actual object.
- Whenever obj. of a class is passed to its class's member function, class data members can be accessed inside the func. using obj.name & dot operator.

Eg -

```
#include <iostream>
using namespace std;
class complex
{
  int re, im;
  Public:
  void get()
  {
    cout << re << "+" << im << "i" << endl;
  }
  void sum (complex c1, complex c2)   // By value
  {
    re = c1.re + c2+re;
    im = c1.im + c2.im;
  }
};
int main()
{
  complex c1, c2, c3;
```

Q8. Create 2 classes: Bankacc & audit. Bank Acc holds bal. info. Write friend func. in Audit that accesses & prints bal. info for auditing.

A-
```
#include <iostream>
using namespace std;
class Audit;
class Bankacc{
    int bal;
    Public:
      Bankacc(int b){
          balance=b;
      }
    friend void AuditAccount(Bankacc &acc, Audit &auditor);
};
class Audit{
    Public:
      void printAudit(Bankacc &acc){
          AuditAcc(acc, *this);
      }
};
void AuditAccount(Bankacc &acc, Audit &auditor)
{
  cout<<" Auditing Account: "<<endl;
  cout<<" Balance :₹"<<acc.bal<<endl;
}
int main(){
    Bankacc account1(50,000);
    Audit auditor;
    auditor.printAudit(account1);
}
```

14/8

```
    c1.disp();
    c2.disp();
    return 0;
}
```

Q4. WAP to demonstrate constructor overloading.

```
#include<iostream>
using namespace std;
class rectangle
{
  int len, wid;
  Public:
  rectangle (int a)
  {
    len = a;
    wid = a;
  }
  rectangle (int a, int b)
  {
    len = a;
    wid = b;
  }
  rectangle ()
  {
    len = 1;
    wid = 2;
  }
  void calc()
  {
    int a;
```

```
Sum = 0;
for(int i=0; i<n; i++)
{
  sum += i;
}
cout << " sum = " << sum;
}
Numbers (Numbers n1)    // Copy constructor
{
   n = n1.n
   sum = 0;
   for (int i = 0; i<n; i++)
   {
     sum += i;
   }
   cout << " sum = " << sum;
}
~Numbers()                    // Destructor
{
  cout << "\n object destroyed";
}
};
int main()
{
  Numbers n1(5);
  Numbers n2(n1);
  return 0;
}
```

```
friend void avg(Student s);
};
void avg(Student s){
float avg = (s.m1+s.m2+s.m3)/3.0;
cout << "The avg. marks are: " <<avg;
}
int main(){
Student s;
s.accept();
avg(s);
}
```

Q- WAP to add gamma, beta, Alpha classes, each with a priv. D.M. using friend func.

A-
```
#include <iostream>
using namespace std;
class beta;
class gamma;
class alpha{
    int a;
    Public:
    friend void accept(alpha a1, beta a2, gamma a3);
    friend void sum(alpha a1, beta a2, gamma a3);
};
class beta{
    int b;
    Public:
    friend void accept(alpha a1, beta a2, gamma a3);
    friend void sum(alpha a1, beta a2, gamma a3);
};
```

(Q1 & 2 with all cons, 3, 4 ~~one~~ it is, All with destructors)

classmate

Date
Page

Exp 5 - Constructor & Decons

Q1. WAP to find sum of no.s bet" 1 to n using cons. where val of n will be passed to the constructor.

A-

```
#include <iostream>
using namespace std;
class Numbers
{
  int n; ~~int sum;~~ ← Public
  Numbers ()   // default constructor
  {
    n=5;
    sum = 0;
    For(int i=0; i<n; i++)
    {
      sum += i;
    }
  }
  void disp()
  {
    cout << "Sum till n: " << sum;
  }
  ~Numbers()
  {
    cout << " Object deleted ";
  }
};
int main()
{
  Numbers n1;
  n1.disp();
}
```

```
        total = ((marks + score)/200)*100;
        cout << "Total percentage: " << total << "%" << endl;
    }
};
int main(){
    Result obj;
    obj.accept();
    obj.calc();
    return 0;
}
```

Q3. Multilevel Inheritance:
Create a class vehicle with attributes like brand & model.
Derive a class Car from vehicle adding "Type" (eg. SUV).
Derive a class E.V from Car which adds battery capacity.
Write functions to display details.

A-
```
#include <iostream>
using namespace std;
class Vehicle
{
  Public:
    string brand, model;
};
class Car : public Vehicle
{
  Protected:
    string type;
}
```

```
Public:
  XYZ (int j) {
    n=j;
  }
  friend int max(ABC &abc, XYZ &xyz);
};
int max(ABC &abc, XYZ &xyz);
{
  if(abc.m > xyz.n)
  {
    return abc.m;
  }
  else
  {
    return xyz.m;
}
int main()
{
  ABC abc(10);
  XYZ xyz(20);
  cout << max(abc, xyz);
  return 0;
}
```

☆ Pointer to Object

- A pointer variable which points to the address of an object is known as a pointer to object.
- This ptr. var. can be used to access ~~the~~ members of the class to which the obj. it points at belongs to.

- It allows an object to access a data member in a member function when said object is used to call function.
- Only members of a class can have a "this" pointer. It is why a friend function does not have a "this" pointer.
- To access data members of a class using the "this" pointer, an arrow operator (→) must be used.
- Uses-
1) It is used to get data members & passed parameters in a function when both share the same identifier.
2) It is used to reference a mem. func. inside another mem. function, so ~~two~~ functions can be ~~called~~ used by calling 1 function.
- "This" ptr. is also not usable with static member func; as they are associated with the class itself rather than it's object.

Eg:-

```
#include <iostream>
using namespace std;
class Sample {
    int a;
    Public:
        void set (int a) {
            This → a = a;
        }
```

- Used when a value that needs to be changed every time the function is called must be returned.
- Here the return type must be declared with (&).
- It avoids creating unnecessary copies, but if you try to return a local variable by reference the results may be wrong, as it is destroyed post-exec. of function.
- Hence you must always return static var., global var, or class Data. Mem. by reference.

Eg:-

```
#include <iostream>
using namespace std;
int & counter()
{
  static int count = 0;
  count++;
  return count;
}
int main() {
    cout << counter() << endl;
    cout << counter() << endl;
    cout << counter() << endl;
}
```

Outputs:- 1
2
3