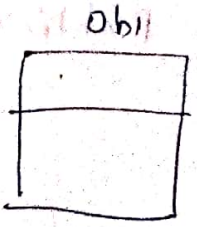
Write a pgm oop to add two objects.

```
#include <iostream.h>
#include <conio.h>
class num
{
private: int a;
public: void display()
        {
        cout<<"The value is; "<<a;
        }
        void input()
        {
        cout<<"The value of a";
        cin>>a;
        }
        void add(ob1, ob2)
        {
        ob3 = ob1 + ob2
        cout<<"The value = "<<ob3;
        }
int b;
};
```

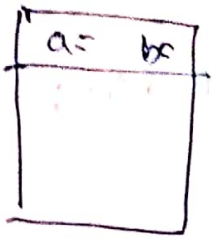


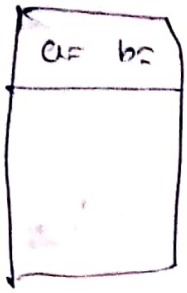
```
int main()
{
num ob1, ob2, ob3;
ob1.input();
ob1.add();
ob1.show();
ob1.input();
ob2.input();
```

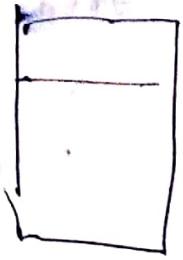
Write a OOP to add two complex numbers;

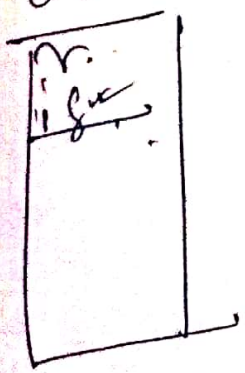
```
#include <iostream.h>
#include <conio.h>
class num
{
private: int a, b;
public: void display()
        {
        if(b<0)
        cout<<"\n"<<a<<"i"<<b*(-1)
        else
        cout<<"\n"<<a<<"+i"<<b
        }
      void input()
      {
      cout<<"Enter the value";
      cin>>a>>b;
```

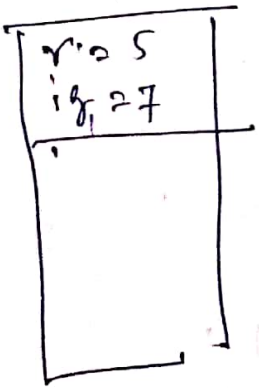
```
}
void add(num t1, num t2)
{
  a = t1.a + t2.a;
  b = t1.b + t2.b;
}
};int main()
{num ob1, ob2, ob3;
 ob1.input();
 ob1.show();
 ob2.input();
 ob2.show();
 ob3.add(ob1, ob2);
 ob3.show();
 return 0;
}
```





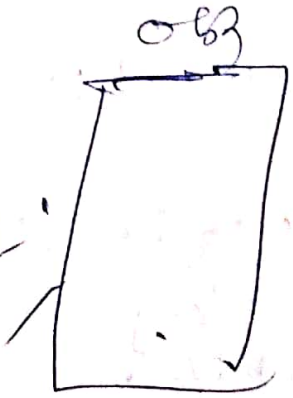






5 + i7

5 - i7



ob3 = ob1 + ob2

5 - i7

// Programme to add two rational nos

```
class rational
 {
  int n;
  int d;
Public: void input();
        void show();
        void add(rational, rational)
        int hcf(int, int);
 };
void rational :: input()
 {
  cout << " In enter numbers ";
  cin >> n;
  cout << " In enter dinemiator ";
  cin >> d;
 }
void rational :: show()
 {
  int c = hcf(n, d);
  n = n/c;
  d = d/c;
  Cout << "the value = " << n << "/" << d
```

```
int rational :: hcf (int x, int y)
{
if (y==0)
return (x);
else hcf
return ( y, x%y );
}

void rational :: add (rational a, rational b)
{
n=(a.n * b.d) + ( a.d * b.n)
d = (a.d * b.d)
}

int main( )
{
rational ob1, ob2, ob3;
ob1.input();
ob1.show();
ob2.input();
ob2.show();
ob3.add (ob1, ob2);
ob3.show( );
return (0);
}
```

temp
n = a.n * b.d + b.d * a.n
d = a.d * b.d

```
rational rational :: add(rational b)
{
rational temp;
temp.n= n*t.d + d *t.n
temp.d = d *t.d ;
return (temp);
}
```

ob1 | ob2

ob3 = ob1.add (ob2);

C++ // object oriented programming

```
#include <iostream>
#include <stdio.h>
using namespace std;  // ⊗ when Dev C++ use
```

Insertion put to operator

```
cout << "          ";
cout << endl << "_____";
cout << a;
cout << "\n The value = " << a;
```

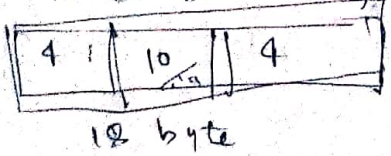
ostream

```
cin >> a >> b >> c;
```

istream extraction get in operator / get from

```
int a;
```


```
struct book ⊗
{
  int page;  ④
  char name[10];  ⑩
  float price;  4
};
struct book b;
```




```
b.name
b.page
b.price
```

⊙ ⟹ Member access operator

```
class book
{
  ═══
  ═══
  ═══
void read();
};
```

object = Data + function

object is an instance of a class.

cout << " " << a;

cin >> a >> b >> c;

int a = 32770;

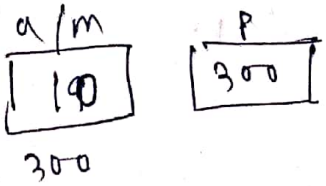
int a = 5;

int a = strlen("sudip");

int a=5; int *p;

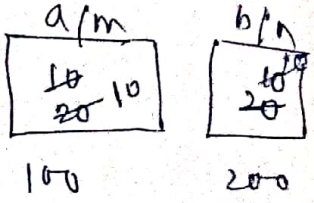
Reference Variable

int a=10;
int *p= &a;
int &m = a

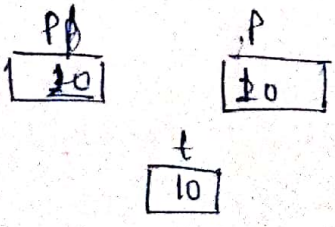


m = 10

```
main()
{
  int a=10, b=20;
  void swapv(int ,int) ;
  void swapa(int *, int*);
  void swapr(int &, int &);

  cout << "value of a ="<< a;
  cout << " value of b = " << b;

  swapv( a, b) ; // call by value.
  cout << " value of a=" << a;
  cout << " value of b = " << b;
  swapa(&a, &b) ; // call by address
  cout << " value of a = " << a;
  cout << " value of b = " << b;
  swapr( a, b);
  cout << "value of a=" << a;
  cout << " value of b = " << b;
}
```

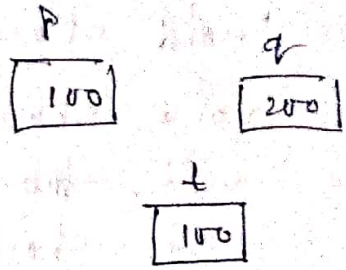



```
void swapr ( int p, int q)
{
  int t
    t = p;
    p = q;
    q = t
}
```



// No change in main

```
void swapp(int *p, int *q)
{
  int t
  t= *p
  *p= *q
  *q= t
}
```



// change in main

```
void swapr(int &m, int &n)
{
  int t;
  t=m;
  m=n;
  n=t;
}
```

// change in main

Function overloading :—

In c++, it is possible to define different function with same name but their parameter type or nos of parameter are different.

In a particular block, when there are two or more function with same name having different parameters or no of parameter is different is known as function overloading. It is an example of compile time polymorphism

```
main()
{
 void show();
 void show(int);
 void show(int, int);
 void show(int, char);
 show();
 show(10);
 show(10,20);
 show(10,'c');
}
 void show()
 {
   cout << "No parameter";
 }
 void show(int a)
 {
  cout << "\n value = " << a;
 }
 void show(int a, int b)
 {
  cout << "\n value = " << a;
  cout << "\n value = " << b;
 }
void show(int a, char b)
 {
   cout << "value = " << a;
   cout << " value of b = " << b;
 }
```

[2016]
1)a) Default Argument

```
main()
{ void show (int=0, int=0, int=0)
  show (10);
  show ( );
 show (10, 20);
  show (10, 20 30);
}
void show (int x, int y, int z)
 {
  cout << x << y << z;
 }
```

The function assign a default value to the parameter which doesn't have a matching argument in the function call is called default argument. Here default values are must specified when the function is declared. The compiler looks at the prototype to see how many arguments a function uses and alerts the program for possible default values, one important point to note is that we must add default from Right to left.

Constructor :- constructor is a function whose name is same as class name. It has no return type not even void. Its basic purpose is to allocate memory and initialized object when it is declared. They should be declared in the public section. It may have parameter or no parameter. If the constructor has no parameter, it is known as default constructor. when it has parameter, then it is known as parametric constructor or overloaded constructor. It invokes automatically when object is declared. we can use constructor by two ways ~~e.g. in~~ —

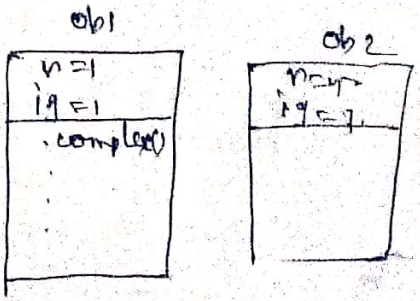
1) Implicit Call
   exmp: num obl (10);

2) Explicit Call
   examp: num obl = 10;

~~copy cpy~~

```
#include <iostream.h>
#include <conio.h>
Using namespace std;
class complex
{
    private:   int a,b;
    public:    complex() // Default constructor
               {
                 a= 1;
                 b= 1;
```

```
Complex( int m, int n)          // parametric constructor or
{                               // overloaded
  a = m;
  b = n;
}
void input()
{
cout << " Enter value " ;
cin >> a >> b;
}
void show()
{
if (b<0)
cout << "\n" << a << "-i" << b*(-1);
else
cout << "\n" << a << "+i" << b;
}
};
void main()
{
  Complex ob1; // Invokes default constructor
  ob1.show();
  Complex ob2(5,7); // " parametric "
  Complex ob3 = Complex(6,8) // " " "
  ob2.show();
  ob3.show();
  ob1.input();
  ob1.show();
getch();
}
```



num obj, o
obj = obj;

copy constructor

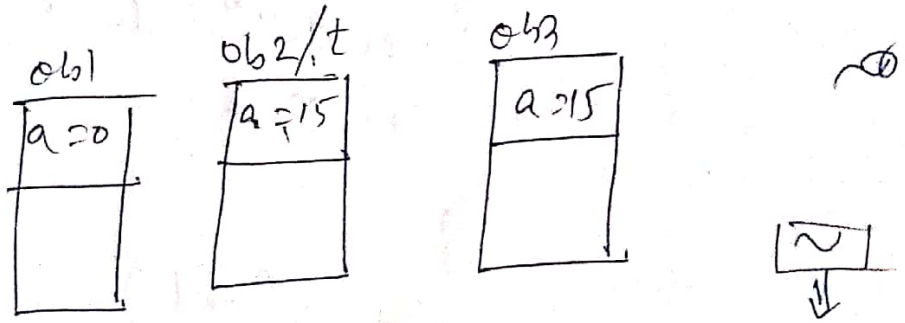
int a;
int b = 5;
int c = b;

```
class num
{
    private : int a;

    public:
        num ( )
        { a = 0;
        }
        num ( int x ) // parametric con
        { a = x;
        }
        num ( num & t ) // copy cons
        { a = t.a;
        }
        void input( )
        { cout << " Enter a number ! ";
          cin >> a;
        }
        void show( )
        { cout << "\n The value = " << a;
        }
};
```

// shallow copy

~num( )
{
cout << " object destroyed ";
}

```
void main()
{
    num ob1; // invokes default constructor.
    num ob2(15); // " parametric "
    num ob3(ob2); // " copy "
    ob1.show();
    ob2.show();
    ob3.show();
}
```



It is a function whose name is same as class name preceded by tield (~) operator. It has no return type and no parameter. It's purpose to deallocate memory (objects) when control goes outside of the scope where the objects are declared.

int a[5];

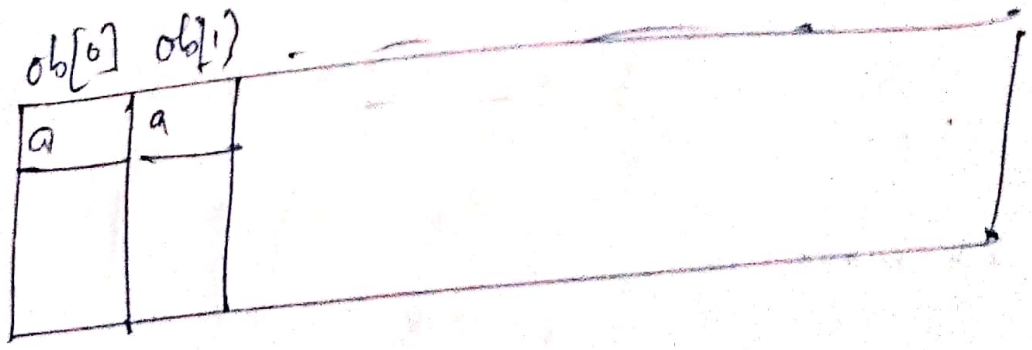
char a[5];

num ob[5];

Array of object

```
class num
{
    private:  int a;
    public:
        void input()
        { cout<<"Enter a no:";
          cin>>a;
        }
        void show()
        { cout<<"Value"<<a;
        }
};
```

```
main()
{
    num ob[10];
    for(int i=0; i<10; i++)
    {
        ob[i].input();
    }
    for(i=0; i<10; i++)
        ob[i].show();
}
```



```
class num
{
 Private: int a;
 Public: void input()
         {
         cout << "Enter the value ;
         cin >> a ;
         }

        void show()
        {
        cout << "\t" << a ;
        }

};
int main()
{
  num ob[10] ;
  for(int i=0 ; i<10; i++)
     {
       ob[i].input();
     }
  for(i= 0; i<10 ; i++)
    {
      ob[i]. show ();
    }
```

```
for (i=0; i<10; i++)
  {
  for (int j=0; j<9; j++)
    {
    if (ob[j] < ob[j+1]
```

```
class num
{
  private:  int a;
  public:   void input()
            {
            }

            }
            void show()
            }
              cout<<"\t"<<a;

            }
int         gthan (num t)
{           @ if (a > t.a)
              return 1;
            else
            ®   return 0;

};        }
```

```
void main()
{
    num ob[10];
    int i, j;
    num temp;
    for(i=0; i<10; i++)
        ob[i].input();
    cout<<"\nBefore Sorting:";
    for(i=0; i<10; i++)
        ob[i].show();
    for(i=0; i<=8; i++)
    {
        for(j=0; j<=8-i; j++)
        {
            int k = ob[j].gthan(ob[j+
            if(k==1)
            {
                temp = ob[j];
                ob[j] = ob[j+1];
                ob[j+1] = temp;
            }
        }
    }
    cout<<"After Sorting!";
}
```

```
for (i=0; i<10; i++)
    ob[i].show();
```

}

→ if(ob[j].gtthan (ob[j+1]))
{

}

)

Q1) WAP in c++ Implement string using constructor.

Q2) why assignment operator is assigned in C++.

// Multiplication of two complex no //

```
#include <iostream.h>
#include <conio.h>

class complex
{
private : int re, ig;

public :
        void input()
        {
        cout << " Enter the value : ";
        cin >> re >> ig ;
        }
        void show()
        {
        if( ig < 0)
        cout << re << "-i" << (-1)* ig ;
        else
        cout << re << "+i" << ig ;
        }
complex operator+ ( complex t1, complex t2)
{ complex temp;
temp.re = t1.re + t2.re;
temp.ig = t1.ig + t2.ig; return(temp);
}
```

```
complex operator * (complex t1, complex t2)
{ complex temp;
temp.r = (t1.r * t2.r) - (t2.ig * t1.ig)
temp.ig = (t1.r * t2.ig) + (t1.ig * t2.r)
return temp;
}
};
void main()
{ complex ob1, ob2, ob3, ob4
ob1.input();
ob2.input();
ob3 operator * (ob1, ob2);
ob3 operator + (ob1, ob2);
ob3.show();
ob4.show();
}
```

ob3 = ob1 operator*(ob2)
ob4 = ob1 operator*(ob2)
ob3 = ob1 * ob2;
ob4 = ob1 + ob2;

ob1 ob2

Q.) why assignment operator is overloaded in the overload

Q.) Array subscript [] operator ?

Q.) overload get in & put to operator ?

Q.) overload pre & post increament operator

Q.) Write a programe to add to complex number using non member function

Q.) WAP about Fibbonacci sequence using con

```
class complex
{
    private: int r, ig;
    public: void input()
            {

            }
            void show()
            {

            }
    friend complex add (complex, complex);
};

complex add (complex t1, complex t2)
{ complex temp;
  temp.r = t1.r + t2.r;
  temp.ig = t1.ig + t2.ig; return (temp); }
```

```
main()
{
  complex ob1, ob2, ob3;
  ob1.input();
  ob2.input();
  ob1.show();
  ob2.show();
ob3=add(ob1 * ob2);
  ob3.show();
return 0;
}
```

Dynamic constructor

```
/* program for dynamic constructor */
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
class name
{
 private : char *s; int l;
 public :  name()
           {
            l=0
            s= new char [l+1]
           }
```

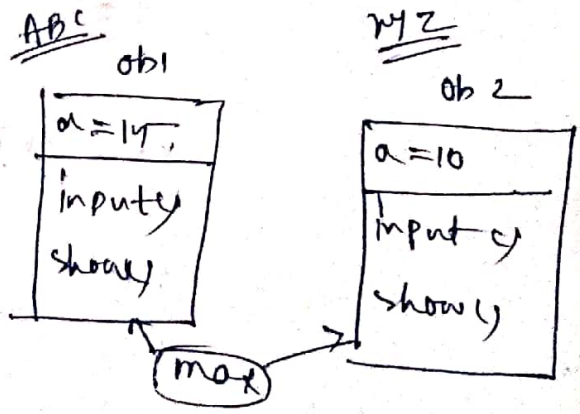
```
num (char *x)
{
 l = strlen(x);
 s = new char[l+1];      // Deep copy
 strcpy(s, x);
}

num (num &t)
{
 l = strlen(t.s);        // l = t.l;
 s = new char[l+1];
 strcpy(s, t.s);
}

void input()
{
 cout << "\n Enter a name:";
 cin >> s;
 l = strlen(s);
}

void show()
{
 cout << "\n The string is : " << s;
 cout << "\n length = " << l;
}
};

int main()
{
 num ob1;
 num ob2("Pandip");
 num ob3(ob2);
```

```
    obj.show();
    obj2.show();
    obj3.show();
    obj.input();
    obj.show();
  return 0;
}
```

Friend function :—

```
class ABC;  // Forward declaration
class XYZ
{
  private : int a;
  public :    void input()
              {
              ==
              }
              void show()
              {
              ==
              ==
              }
    friend int max(ABC, XYZ);
};

class ABC
{
    private : int a;
    public : void input()
             {
             ==
             }
```

```
    void show()
    {
    ==
    }
    friend int max(ABC, XYZ);
};

int max(ABC t1, XYZ t2)
{
    if(t1.a > t2.a)
        return 1;
    else
        return 0;
}
main()
{
    ABC ob1;
    XYZ ob2;
    ob1.input();
    ob1.show();
    ob2.input();
    ob2.show();
    int c;
    c = max(ob1, ob2);
    if(c==1)
        ob1.show();
    else
        ob2.show();
}
```




```
if(c==1)
{
    cout << "Maximum=";
    ob1.show();
}
```

8.) overloaded get in & put to operator.

cout << obj; obj.show();
cin >> obj

= operator<<(obj);

```
#include<iostream>

class num
{
  private: int a;
  public:
          num()
          {
          a=0;
          }
          num(int x)
          {
          a=x;
          }
  friend ostream& operator<<(ostream&, num&);
  friend istream& operator>>(istream&, num&);

};

ostream& operator<<(ostream& dout, num& t)
{
  dout << t.a;
  return(dout);
}
istream& operator>>(istream& din, num& t)
{
  din>>t.a;
  return(din);
}
```

```
main()
{
   num ob;
   cout << "Enter a object. ";
   cin >> ob;   // operator >> (cin, ob);
   cout << " The value of object = " << ob;
```

// operator << (cout,

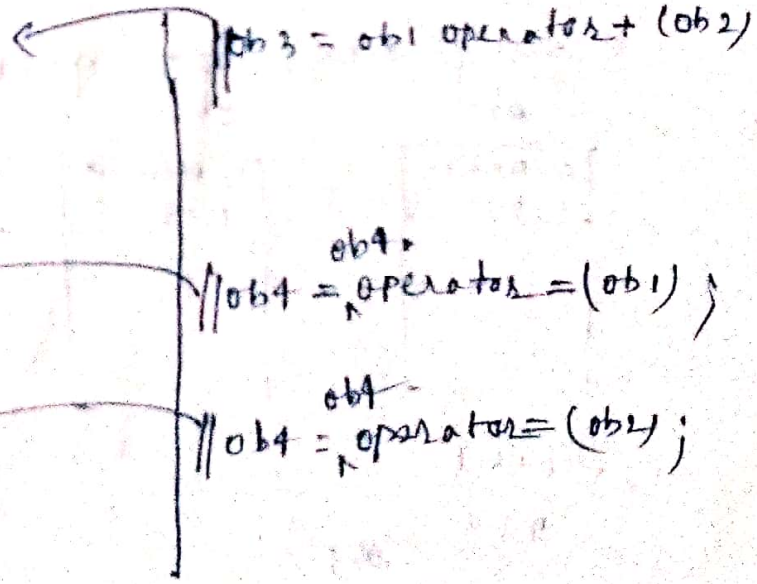
overloading +, = for string

$ob_3 = ob_1 + ob_2$

$ob_3 = ob_2$

```
#include <iostream>

class num
{
  private:  char *s;
            int l;

  Public:   num()
            {
              l=0;
              s= new char[l+1];
            }
          void input()
          {
            cout << " Enter string";
            cin >> s;
          } l = strlen(s);
```

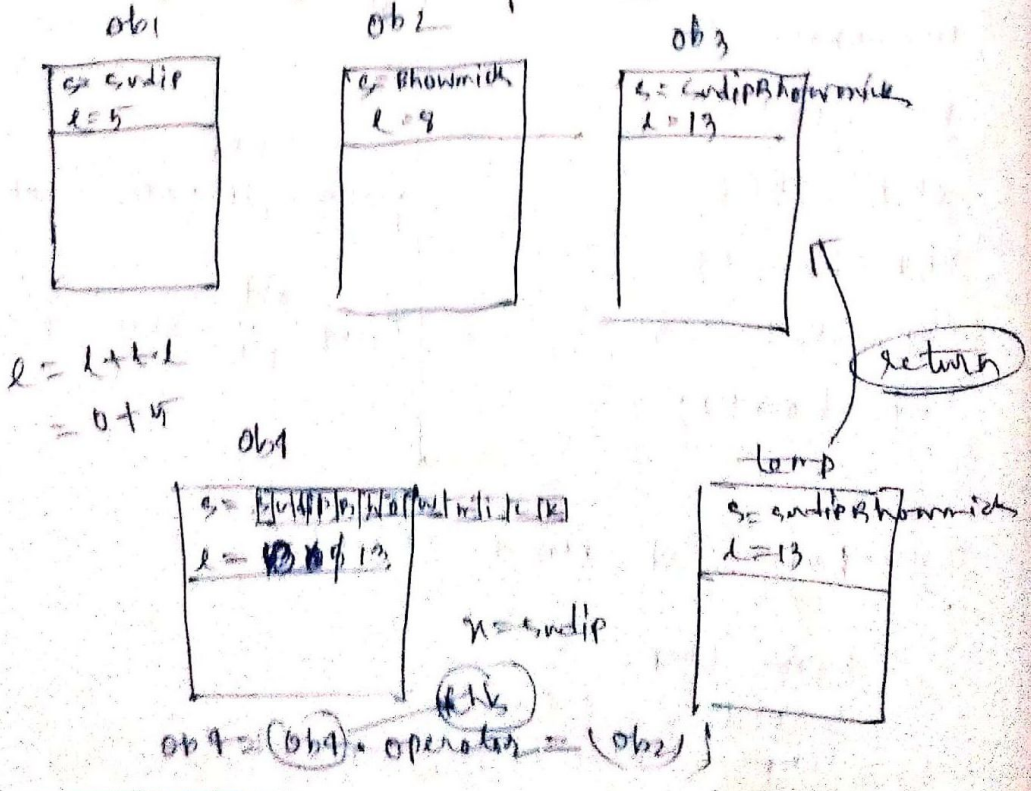
```
void show()
{
  cout << " The string is : " << s;
  cout << " Length = " << l;
}

num operator+ (num);
num operator= (num);

};

main()
{
num ob1, ob2, ob3
ob1.input();
ob2.input();
ob1.show();
ob2.show();
ob3 = ob1 + ob2;
ob3.show();
num ob4;
ob4 = ob1;
ob4.show();
ob4 = ob2;
ob4.show();
}

num num :: operator+ (num t)
{
    num temp;
    temp.l = l + t.l;
    temp.s = new char[temp.l+1];
```



```
    strcpy ( temp->s , s );
    strcat ( temp->s , t.s );
    return (temp);
}

num num :: operator = ( num t )
{
    char *x = s;
    l = l + t.l;
    s = new char [ l+1 ];
    strcpy ( s , x );
    strcat ( s , t.s );
    return *this;
}
```

```
* num num :: operator=
        (num
{
    num temp;
    temp.l = l+t.s
    temp.s = new char
        [temp.l
strcpy ( temp.s , s );
strcat ( temp.s , ts);
    return (temp);
}
```

8.) ~~with assignment operator is overload~~



1. Write a program in c++ to obtain fibonacci sequence using constructor.

```
#include <iostream>

class fibonacci
{
  private: int a, b;
  public: fibonacci ()
          {
            a=0;
            b=1;
          }

          void series(int x)
          {
            int c;
            cout<<x<<"\n";
            for (int i=1; i<=x; i++)
              {
                cout<<" "<<a;
                c=a+b;
                cout<<c;
                a=b;
                b=c;
              }
          }
};

int main()
{
```

```
int n;
cout << "How many n";
cin >> n;
ob.series(n);
getch();
}
```

Type conversion

Ob2 = ob1 + 10

① Basic type to class type.

```
#include <iostream>

class num
{
  public: int a;
  private: void input()
           {
            cout << "Enter the value";
            cin >> a;
           }
          void display()
          {
           cout << a;
          }
```

```
num operator+ (int y)
{
    num temp;
    temp.a = a + y;
    return temp;
}
};
int main()
{
num ob1, ob2;
int n;
cout << "Enter the value of n";
cin >> n;
ob1.input();
ob1.show();
ob2 = ob1+n;        // ob2 = ob1.operator+(n);
ob2.show();
return 0;
}
```

(11) class type to Basic type.

```
#include <iostream>

class num
{
  public: int a;
  private: void input()
           {

           }

  void show()
  {

  }

int operator+(num t)
{
  int b;
  b = a + t.a;
  return b;
}
```

num operator + (num t)

{

}

};

int main()

{

Int a;

num ob1, ob2, ob3;

ob1.input();

ob2.input();

a ob3 = ob1 + ob2; // a = obj1.operator(obj2);

a = ob3;

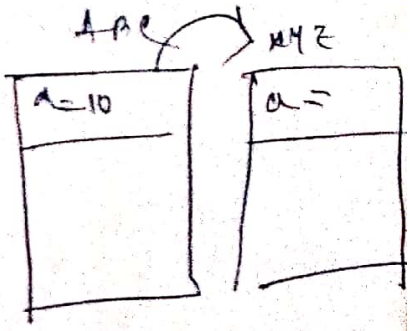
//a = a operator + (ob3);

Cout << a;

(iv) one class type to another class type,

```
#include <iostream>

class XYZ ;
class ABC
{
  public: int a;
  private: void input()
           {

           }
           void show()
           {
           }
    friend operator
    friend int copy(ABC, XYZ);
};
class XYZ
{
  public: int a;
  private: void input()
           {
           }
```

```
void show()
{
}
num operator=
friend int copy(ABC, XYZ);
};
num operator=
int copy(ABC t1, XYZ t2)
{ temp;
  temp.a=
}
int main()
{
ABC ob1;
XYZ ob2;
ob1.input();
ob2.input();
ob2 = ob1; // ob2 = ob2.operator=(ob1);
```



@52 = @51 — Addr
↓
XYZ

```
class ABC;
class XYZ
{
    private: int a;
    public:
        XYZ()
        {
            a=0;
        }
        void input()
        {
        }
        void show()
        {
        }
        friend XYZ operator=(ABC);
};
```

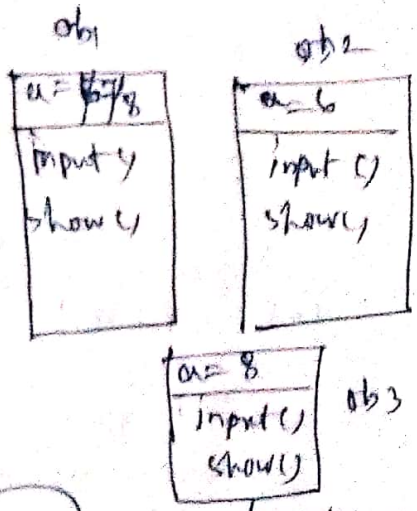
```
class ABC
{
    private: int b;
    public: ABC()
            { b=0;
            }
            void getdata()
            {
            }
            void display()
            {
            }
            friend XYZ operator=(ABC);
};
XYZ operator=(ABC t)
{
    XYZ temp;
    temp.a = t.b;
    return temp;
}
```

```
main()
{
    XYZ ob2;
    ABC ob1;
    ob1.getData();
    ob1.display();
    ob2 = ob1; // ob2 = operator=
    ob2.show();
}
```

/* overloading pre and post increament operator */

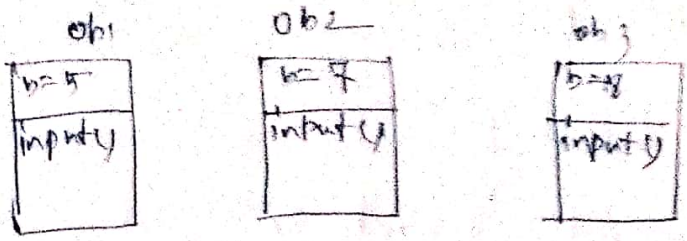
```
#include <iostream>

class num
{
    public: int a;
    private: int a;
    public: void input()
            {
             cout << "Enter the value of a";
             cin >> a;
            }
```

```
void display()
{
cout << "The value of a = " << a;
}

num operator++(int=0)
{
  num temp;                    // post increment
  temp.a = a++;
  return(temp);
}

num operator++()
{
  num temp;                    // Pre increment
  temp.a = ++a;
  return(temp);
}
};
int main()
{
num ob1, ob2, ob3;
ob1.input(); → (6)
ob2 = ob1++;  ------------> ob3 = ob1.operator++();
ob2.show(); → (5)
ob1.show(); → (4)
ob3 = ++ob1; ------------> ob3 = ob1.operator++();
ob3.show(); → (3)
ob1.show(); → (2)
return(0);
}
```



/* static data & static function */

```
class num
{
    private:  static int a;
              int b;
    public:   void input()
              {
                cout << "Enter no";
                cin >> b;
                a++
              }
              static void show()
              {
                cout << "value of a=" << a;
                cout << "value of b=" << b;
              }
};
int num:: a;
main()
{
    num ob1, ob2, ob3;
    ob1.input();
    ob1.show(); // Error num:: show()
    ob2.input();
    ob2.show(); // Error num:: show();
    ob3.input();
    ob3.show(); // Error num:: show();
}
```



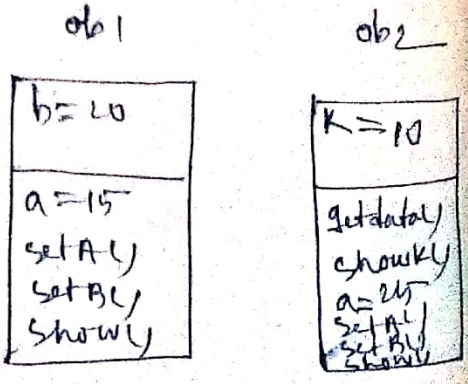
a = p1.y - 3

show();

/* single Inheritance */

```
class A
{
    private : int b;
    public : int a;
        void seta ( int x)
        {
          a=x;
        }
        void setb ( int y)
        {
          b=y;
        }
        void show()
        {
          cout << a;
        }
};
```

```
class B : public A
{
    private : int k;
    public : void getdata()
             {
                cout << "Enter value";
                cin >> k;
             }
    void showk()
    {
        cout << k;
    }
};
main()
{
    A obj;
    obj.setA(15);
    obj.setB(20);
    obj.show();
    B obj2;
    obj2.setA(25);
    obj2.show();
    obj2.getdata();
    obj2.showk();
}
```



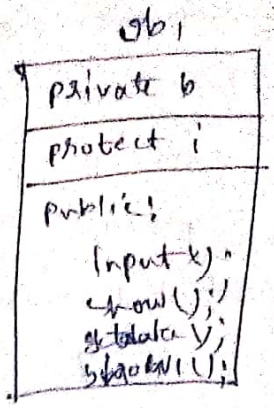
Base class access control (access specifier)

~~When a base class~~

When the access specifier for a base class is public, all the public members of the base class become public members of derived class and all protected members of ~~derive~~ base class become protected members of derive class. But private member of the base class is not accessible by derived class.

```
class A
{
  protected : int i;
  public :    void input()
              {
              }
              void show()
              {
              }
};
class B : public A
{
  private:  int b;
  public :  void getdata()
            {
              cin >> b;
            }
            void show()
            {
            }
};
```

```
main()
{
    b obj;
    obj.input();
    obj.show();
    obj.getdata();
    obj2.show();
}
```



When the access specifier for a base class is private, all the public members and protected members of the base class become private members of the derived class.

```
class A
{
    protected: int i;
    public:    void input()
               {
               }
               void show()
               {
               }
};
```

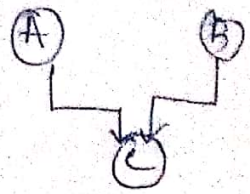
```
class B : private A
{
    private: int b;
    public: void getdata()
            {
            }
            void show()
            {
            }
};
main()
{
    B obj;
    obj.input(); // error because (or the access
    obj.show(); // " specifier is private) it
                becomes private.
    obj.getdata();
    obj.show();
}
```

~~**~~ when the access specifier for a base class is protected, all the public member and protected member of the base class become protected member of the derived class.

```
class A
{
    protected: int i;
    public: void input()
    {
```

```
    }
    void show()
    {
    }
};
class B : protected A
{
    protected: int b;
    public: void getdata1()
            {
            }
            void show1()
            {
            }
};
main()
{
  B ob1;
  ob1.input(); // error because it is protected
  ob1.show(); // " " " " "
  ob1.getdata();
  ob1.show();
}
```

Multiple Inheritance :-




```
class base1
{
  protected: int x;
  public: void showX()
          {
            cout << "The value of x: " << x;
          }
};

class base2
{
  protected: int y;
  public: void showY()
          {
            cout << y;
          }
};

class derived : public base1, public base2 -
{
  public: void set(int i, int j)
          {
            x = i;
            y = j;
          }
};
```

```
main()
{
  derived ob;
  ob.setx(10,20);
  ob.showx();
  ob.showy();
}
```

Constructor, destructor and Inheritance :=

** Q) When constructor and destructor are executed in case of inheritance?

Ans:⇒ It is possible for a base class, ~~a derived~~ a derived class or both to contain constructors and/or destructors. It's important to understand the order in which these functions are executed when an object of a derived class comes into existence and when it goes out of existence.

When we declared ~~declared~~ a derived class object it invokes base class constructor and then derived class constructor. Destructor will be executed in opposite order when control goes outside of the scope where object is declared

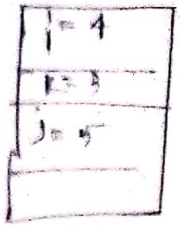
```
/* passing parameter to base class constructor */

class base
{
    protected : int i;
    public:  base ()
             {
               i=0;
             }

            base (int x)
             {
               i=x;
               cout << " constructing Base ";
             }

           ~base ()
             {
               cout << " destructing base ";
             }

};

class derived : public base
{
   private : int j;
   public:   derived ()
             {
               j=0;
             }

             derived (int x, int y) : base (x)
             {
               j=2;
               cout << " constructing derived ";
             }
```

passing parameter to base class constructor through derived class

```
~derived ( )
{
cout <<" destructing derived";
}

void show( )
{
cout <<" The value of i,j" << i;
cout << j ;
}

};
int main( )
{
derived ob (3, 4);
ob.show( );
return 0;
}

/* same topic */
class base1
{ protected : int i ;
public: base1( )
{ cout <<" constructing base
i=0;
}
base1 ( int x )
{ i= x;
cout <<" constructing base1 "; }
```

```
    }
};

class base2
{
  protected : int k;
  public :  base2()
            {
              k=0;
            }
           base2(int x)
            {
              k=x;
              cout<<" constructing base2 ";
            }
};
class derived : public base1 , public base2
{
  private :  int j ;
  Public :  derived ()
            {
              j=0 ;
            }
           derived ( int x, int y , int z ) : base1(y), base2(
            {
              j=z;
              cout<<" constructing derived ";
            }
```

```
    void show()
    {
      cout << i << j << k;
    }
};
main()
{
derived ob(1,4,5);
  ob.show();
}
```




```
/*  */

class base
{
  public: int j;
};
class derived : private base
{
  public:
      base :: j; // make j public again
};
```

```
/* Ambiguity due to multiple base class */

class base
{
  public: int i;
};

class derived1 : public base
{
  public: int j;
};

class derived2 : public base
{
  public: int k;
};

class derived3 : public derived1, public derived2
{
  public: int sum;
};

main()
{
  derived3 ob;
  ob.i = 10; // error — this is ambiguous which i ?
  ob.j = 20;
  ob.k = 30;
  ob.sum = ob.i + ob.j + ob.k; // This is error.
}
```

```
cout << ob.i; // Error.
cout << ob.j;
cout << ob.k;
cout << ob.sum;
}
```

/* This ambiguity can be solved by ~~the~~ using two ways.

① By using scop resolution operator */

```
ob.derived1::i = 10;
ob.derived2::i = 15;
cout << ob.derived1::i;
```

~~/* ① By declaring virtual base class */~~

/* ① By inheriting as virtual */

```
class base
{
public: int i;
};
class derived1: virtual public base
{
public: int j;
}
```

```
class derived2 : virtual public base
{ public : int k;
};
class derived3 : public derived1, public derived2
                 // only single copy of i will be inherited
{ public : int sum;
};
main()
{
derived3 ob;
ob.i = 10;
ob.j = 20;
ob.k = 30;
ob.sum = ob.i + ob.j + ob.k;
cout << ob.i;
cout << ob.j;
cout << ob.k;
cout << ob.sum;
}
```

(Function overriding)

virtual function :— A virtual function is a member function he declared within base class and redefined by derived class. To create a virtual function precede the function declaration in the base class with the key word virtual. Virtual function implement one name multiple form (runtime Polymorphism). In case of multi level inheritence, after overriding

virtual nature is also inherited in all level of derived class,

```
class A
{
    public: virtual void show()
            {
              cout<<" This is Kolkata";
            }
};
class B : public A
{
    public: void show()
            {
              cout<<" This is Mumbai";
            }
};
class C: public B
{
    public: void show()
            {
              cout<<" This is Goa";
            }
};
int main()
{
    A obj, *P;
    B obj2;
    P = &obj;
```

```
    p->show();
    p=&ob2;
    p->show();
    (* ob3;
   p=&ob;
   P->show();
}
```

calling a virtual function through base class reference :—

```
class A
{
  Public: virtual void show()
          {
            cout << "This is Kolkata"
          }
};

class B : public A
{ public: void show()
          {
            cout << "This is Mumbai";
          }
};

class C: public B
{
  public: void show()
          {
            cout << "this is Goa";
          }
};
```

```
void function (A &t)
{
    t.show();
}
void main()
{
    A ob1;
    B ob2;
    C ob3
    function (ob1);
    function (ob2);
    function (ob3);
}
```

/* virtual function are hierarchical */

```
class A
{
    public:
        virtual void show()
        {
            cout << "This is Kolkata";
        }
};
class B : public A
{
    public:
        void show()
        {
            cout << "This is Goa";
        }
};
```

```
class c : public B
{
  public:
};
main()
{                                  //output: this is c,oda
  c obj;
  obj.show();
}
```

Pure virtual function :=

A pure virtual function is a virtual function that has no definition with in the base class. To declare a pure virtual function →

```
virtual  return type  function name (parameter list) =0;
```

~~void print~~

When a virtual function is made pure, any derived class must provide its own definition. If the derived class fails to override the pure virtual function, a compile time error will result.

```
class number
{
  protected : int val;
  public : void setval (int i);
           {
             val = i;
```

```
virtual void show() = 0;
};
class hextype : public number
{
public: void show()
    { cout << "Hexadecimal = " << val;
      cout << val
    }
};
class octtype : public number
{
  public: void show()
    { cout << " octa
    }
};
class Bintype : public number
{
  Public: void show()
    { convertion();
    }
main()
{ number
  setval(4);
}
```

~~#include <iostream>~~

~~#define~~

```
main()
{
    number *p;
    p->setval(20);
    hextype ob1;
    octtype ob2;
    bintype ob3;
    p=&ob1; p->set(24);
    p->show();
    p=&ob2
    p->set(30)
    p->show();
    p=&ob3
     p->setval(40);
     p->show()
}
```

/* Abstract class */

```
class convert
{
    protected:
        double val1, val2;
    public: convert(double i)
        {
            val1 = i;
        }
    double getconvert()
    {
        return(val2);
    }
    double getinit()
    {
        return(val1);
    }
    virtual void compute() = 0;
};
class ltog : public convert
{
    public: ltog(double i): convert(i)
        {
        }
    void compute()
    {
        val2 = val1/3.7894;
    }
```

we have to define default constructor while defining parameterized constructor

```
class ftoc : public convert
{
public : ftoc ( double i ) ; convert(i)
        {
        }
      void compute ()
        {
          val2 = ((val1 - 32) * 5) / 9 ;   // (val1-32)/1.8;
        }
};
main()
{ convert *p;
  ltog obj (4);
  ftoc obj2(98);
  p = &obj;
  cout << " p -> getinity " << " Litre " ;
  p -> compute. ();
  cout << p -> getconvert () << " Gallon " ;
  p = &obj2
  cout << p -> getinit () << " Farenhite " ;
  p -> compute ();
  cout << p -> getconvert () << " Celcius " ;
}
```

fit to meter add

Q.) figure class

2016

2) ⓒ Abstract class:= A class that contains atleast one pure virtual function is said to be abstract. An abstract class contain one or more function for which there is no definition and no object of the abstract class may be created. But we can create pointers and references to an abstract class. It supports runtime polymorphism

Ex:

Early binding Vs Late binding
(Static) (Dynamic)
(Compile time) (Run time)

① Early binding refers to ~~early~~ event that occurs at compile time. In essence early binding occurs when all information needed to call a function is known at compile time.

Ex: function overloading, operator overloading.

② The main advantage to early binding is efficiency.

③ As all information to call a function is determined at compile time, the function calls are very fast.

Late binding refers to function call that are not resolved until runtime. virtual functions are used to achieve late binding. When access is via a base class pointer or reference the virtual function

is actually called is determined by the type of object pointed by the pointer, so it is determined at runtime.

The main advantage of late binding is flexibility, unlike early binding, late binding allows us to create program that can respond to events occurring while the program executes without having to create a large amount of "contingency code" Its slower than early binding.

Template :-

Generic function :- A generic function defines a general set of operations that will be applied to various types of data. The type of data that the func" will operate upon is passed to it as a parameter.

A Generic function is created using the key word Template.

```
template <class type> return-type function name(
                                    parameter list)
{
  // Body of func"
}
```

```
#include <iostream>
using namespace std;
template <class X>
void swap(X &a, X &b)
{
  X t;
  t = a;
  a = b; b = t; }
```

```
int main()
{
int i=10, j=20;
double p=10.4, q=20.3;
char c='A', d='B';
cout << "Before swaping" << endl;
cout << "original i, j = " << i << j;
cout << " Original p, q = " << p << q;
cout << " Original c, d = " << c << D;
swap (i, j);
swap (p, q);
swap (c, d);
cout << " After swaping " << endl;
cout << " i = " << i << " j = " << j;
cout << "p = " << p << " q = " << q;
cout << " c = " << c << "d = " << d;
return 0;
}
```

Generic class :—

```
template < class T >
class stack
{
private : T a[10];
          int top;
public: stack()
        {
        top = -1;
        }
```

```
    void push (T);
    T pop();
};
template <class T>
void stack <T>:: push (T ob)
  {
    if (top == 9)
    {cout << " stack is full ";
     return;
    }
   top++
   a[top] = ob;
  }
template <class T>
T stack <T> :: pop ()
{
   if (top == -1)
     {cout << " stack is empty ";
      return null;
     }
 T item = a[top];
   top --;
   return item;
}
int main()
{
stack <char> ob1;
ob1.push ('A');
ob1.push ('B');
```

```
char c = ob1.pop();
cout << "Item poped :" << c;
stack <int> ob2;
ob2.push(10);
ob2.push(20);
int s = ob2.pop();
cout << "Item poped =" << s;
return 0;
}
```

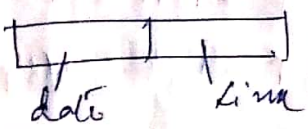
#

```
Template <class T1, class T2>
class num
{
    private:  T1 i;
              T2 j;
    Public:   num(T1, T2);
              void show();
};
Template <class T1, class T2>
num <T1, T2> :: num (T1 x, T2 y)
{
    i = x;
    j = y;
}
Template <class T1, class T2>
num <T1, T2> :: show()
{
```

```
    cout << i;
    cout << j;
  }
int main()
{
  mem <int, float> obj (5, 15.5);
   obj.show();
   return 0;
}
```

/* Single Linked list operation using class */

```
typedef struct node
{
int data;
struct node *link;
}NODE;
```




```
class slink
{
  private: NODE *p;
  public:  slink()
           {
             p=NULL;
           }
        slink (int);
        void addend (int);
        void addbeg (int);
        void addafter (int, int);
        void display ( );
        int count ();
        void del (int);
};
```

```
slink :: slink(int K)
{
    p = new NODE();
    p->data = K;
    p->link = NULL;
}

void slink :: addend(int K)
{
    NODE *temp;
    if (P == NULL)
    {
        P = new NODE();
        p->data = K
        p->link = NULL;
    }
    else
    {
        temp = P;
        while (temp->link != NULL)
            temp = temp->link;
        temp->link = new NODE();
        temp = temp->link;
        temp->data = K;
        temp->link = NULL;
    }
}

void slink :: display()
{
    NODE *temp = P;
    cout << "\n";
    while (temp)        // while (temp) means while (temp != NU
```

```
    {
      cout << " \t " << temp -> data;
      temp = temp -> link;
    }
}
void slink :: add beg (int k)
{
    NODE * temp = new NODE ();
    temp -> data = k;
    temp -> link = p;
    p = temp;
}
void slink :: add after (int pos, int k)
{
    int i;
    NODE * temp, *p;
    if (pos == 0)
    {
      addbeg (k);
    }
  temp = p;
  for (i=1; i<pos; i++)
  {
    if (temp == NULL)
    {
      cout << " \n Invalid position";
      return;
    }
    temp = temp -> link;
}
```

```
    r = new NODE();
    r->data = k;
    r->link = temp->link;
    temp->link = r;
}
void slink::del(int k)
{
    NODE *temp, *r;
    temp = p;
    while(temp)
    {
        if(temp->data == k)
        {
            if(temp == p)
                p = p->link;
            else
                p->link = temp->link;
            delete(temp);
            return;
        }
        else
        {   p = temp;
            temp = temp->link;
        }
    }
    cout << "\n Item not found to delete";
}
```

```
int slink :: count( )
{
    int c=0;
    NODE * temp = P;
    while (temp)
    {
     c++
     temp= temp->link;
    }
  return(c);
}

int main ()
{
      slink ob;
      int ch;
      do
      {
          cout<< "\n 1. Add node at end: ";
          cout << "\n 2. Add node at begining: ";
          cout << "\n 3. Add node after a specified
                        position: ";
          cout<< "\n 4. Display linked list ";
          cout<< "\n 5. Count nodes ";
          cout << "\n 6. Delete node";
          cout<< "\n 7. Exit ";
          cout << "Enter your choice: ";
          cin >> ch;                    // if (ch<1 || ch>7)
      switch (ch)                          continue;
      {
```

```
case 1: int a;
        cout << "\n Enter value for node:";
        cin >> a;
        addend (a);
        break;

case 2: int a;
        cout << "\n Enter value for node:";
        cin >> a;
        addbeg(a);
        break;

case 3: int pos, a;
        cout << "\n Enter the position:";
        cin >> pos;
        cout << "\n Enter value for node:";
        cin >> a;
        addafter (pos, a);
        break;

case 4: display ();
        break;

case 5: int c;
        c = count ();
        cout << "\n Total nos of node =" << c;
        break;

case 6: int a;
        cout << "\n Enter a value to delete:";
        cin >> a;
        Del (a);
```

```
        break;
    case 7: cout << "THE END";
            break;
    default: cout << "\nInvalid choice";
    } while (ch != 7);
    return 0;
}
```

(i) Stack using linked list & using template
(ii) Doubly linked list
(iii) circular queue using array

```
/* Linear Queue Using Array */
#include <iostream>
using namespace std;
#define MAX 10
class lq
{
    private: int front, rear, q[MAX];
    Public: lq()
            {
                rear = front = -1;
            }
        void input(int);
        int del();
        void display();
};
```

```
void Iq :: input ( int K)
{
   if ( rear == MAX - 1)
   {
     cout << " In Queue is full " ;
     return;
   }

  q[++rear] = K;
   if ( front == -1)
      front=0;
}
int Iq :: del ()
{
   int temp;
   if (front == -1)
   {
     cout << " In Queue is empty " ;
     return 0;
   }

 temp = q[front] ;
  if ( front == rear)
  {
    front = rear = -1;
  }
  else
     front++ ;
     return temp;
}
```

```
void lq :: display ( )
{
   if (front == -1 && rear == -1)
    {
      cout << "\n Queue is empty !!!";
      return;
    }
   cout << "\n The queue is : ";
   for ( int i = 0; i <= rear ; i++)
        cout << q[i] << " \t";
}

int main( )
{
 lq ob;
 int ch, a;
 do
 {
    cout << "\n \t Linear Queue operation ";
    cout << "\n 1. Inser \n 2. delet \n 3. Display \n
                4. Exit ";
    cout << " \n Enter Your choice ";
    cin >> ch;
    if ( ch < 1 || ch > 3 )
      continue;
    switch (ch)
    {
```

```
case 1:
    cout << "\n Enter a data to insert:";
    cin >> a;
    ob.input(a);
    break;

case 2:
    a = ob.del();
    cout << "\n Item deleted :" << a;
    break;

case 3:
    ob.display();
    break;

case 4:
    cout << "\n Exit";

}
} while(ch != 4);

return 0;
}
```

```
/* Queue using Array (template) */
#include <iostream>
using namespace std;
template <class T>
class queue
{
  private : int rear, front;
            T a[10];

  public : queue()
           {
             rear = front = -1;
           }
           void insert(T);
           T delet();
           void display();
};
template <class T>
void queue<T> :: insert(T ob)
{ if (rear == 9)
  { cout << "\n The queue is full";
    return;
  }
  rear ++;
  a[rear] = ob;
}
```

```
template <class T>
int queue <T> :: delet ( )
{
    if (front == -1)
    {
        cout << " \n The queue is empty ";
        return NULL;
    }
    T item = a[front];
if (rear == front)
    {
        rear = -1;
        front = -1;
    }
else
    front ++;
    return item;
}

template <class T>
void queue <T> :: display ( )
{
    if (front == -1 && rear == -1)
    {
        cout << " \n The queue is empty ";
        return;
    }
```

```
cout << "\n The queue is :" ;
for (int i= front; i<= rear; i++)
    cout<< a[i] << "\t";
}

main()
{
    queue <char> ob1;
    ob1.insert ('A');
    ob1. insert ('B');
    ob1. insert ('C');
    char d = ob1. delet ();
    cout << "\n The Item popped " << d ;
    queue <int> ob2;
    ob2. insert (15);
    ob2. insert (20);
    ob2. insert (25);
    int l = ob2. delet ();
    cout << "\n The Item popped " << l;
}
```

```
/* program for split a sentence and check it is
weak / strong password */

#include <iostream>
#include <string.h>
using namespace std;
class num
{
  char *s;
  int l;
public:
    num()
    {
      l=0;
      s=new char[l+1];
    }

    num(char *x)
    {
      l=strlen(x);
      s=new char[l+1];
      strcpy(s,x);
    }
  void input()
  {
    char *x;
    cout<<"Enter string ";
    gets(x);
    l=strlen(x);
    s=new char[l+1];
  strcpy(s,x)
```

```
    void display ()
    {
     cout<<" The string is "<<s;
    }
    void strengthpass ();
     void split ();
};
void num :: split ()
{
 int i, j=0;
 cout<<"\n\t";
 for (i=0; i<l; i++)
    {
      if ( s[i]==' ' || s[i+1]=='\0')
        {
          for (j ; j<=i; j++)
             cout << s[j];
             cout<<"\n\t";
        }
    }
}
void num :: strengthpass ()
  {
    int f=0, k;
    for (int i=0; i<l; i++)
      {
        k=s[i];
        if((k>=65 && k<=90) || (s[i]==' '))
```

```
            f=1;
        else
            {
                f=0;
                break;
            }
        }
    if (f==0)
        cout<<" strong password ";
    else
        cout<<" weak password ";
    }
int main()
{
    num obj(" Sudip Bhowmick ");
    obj.display();
    obj.split();
    obj.strength();
    obj.display();
    return(0);
}
```

/* Infix to postfix and

```
#include <iostream>
#include <math.h>
#include <string.h>
using namespace std;
#define MAX 20
typedef struct stack
{
  float a[MAX];
  int top;
} STACK;
class implement
{
  private: char P[30], q[30];
           STACK s;
  public: implement()
          {
            s.top = -1;
          }
          void push (float);
          float pop();
          int pred(char, char);
          void evalute();
          void input();
          void show();
          void intopost();
};
```

```
void implement :: input()
{
  cout << " Enter Infix Expression";
  cin >> q;
}
void implement :: show()
{
  cout << " The Infix Expression is " << q;
}
void implement :: push(float item)
{
  if (s.top == MAX-1)
    {
      cout << " stack is overflow";
      return;
    }
  s.top++;
  s.a[s.top] = item;
}
void implement :: pop()
{
  float data;
  if (s.top == -1)
  {
    cout << " stack is empty";
    return (NULL);
  }
  data = s.a[s.top];
  s.top--;
  return data;
}
```

```
void implement();  into post()
{
    char symb, t;
    int i, j=0, k;
    for( i=0; q[i] != NULL; i++)
    {
        symb = q[i];
        k = symb;
        if ((k>=65 && k<=90) || (k>=48 && k<=57)
                || (k>=97 && k<=122))
        {
            p[j] = symb;
            j++;
        }
        else
        {
            while( s.top != -1 && ( pred (s.a[s.top], symb)
                    == 1))
            {
                t = pop();
                p[j] = t;
                j++;
            }
            if ( s.top == -1 || symb != ')')
            {
                push (symb);
            }
            else
            {
                t = pop();
            }
```

```
while (s.top!=-1)
    {
     t=pop();
     P[j]=t;
     j++;
    }
P[j]='\0';
cout<<"The postfix "<<P;
}
int implement::pred(char c1, char c2)
{
  if (c1=='(' || c2=='(')
    {
     return 0;
    }
  if (c2==')')
    { return 1;
    }
  if (c1==c2)
    { return 1;
    }
  if (c1=='$')
    { return 1;
    }
  if((c1=='/')||(c1=='*')&&(c2!='$'))
  { return 1;
```

```
if ((c1=='*') && (c2=='+' || c2=='-'))
 {
  return 1;
 }
if ((c1=='+') && (c2=='-') || (c1=='-') &&
                              (c2=='+'))
   {
     return 1;
   }
 else
   {return 0;
   }
}
void implement :: evaluate()
 {int i;
  float value, result, k1, k2;
  char t;
  for(i=0; p[i]!='\0'; i++)
   {
    t=p[i];
    switch(t)
    {
      case '+':
            k2 = pop();
            k1 = pop();
            value = (k1-48) + (k2-48);
            value = value + 48;
            push (value);
            break;
```

```
case '-':
        k2=pop();
        k1=pop();
        value=(k1-48)-(k2-48);
        value=value+48;
        push(value);
        break;

case '*':
        k2=pop();
        k1=pop();
        value=(k1-48)*(k2-48);
        value=value+48;
        push(value);
        break;

case '/':
        k2=pop();
        k1=pop();
        value=(k1-48)/(k2-48);
        value=value+48;
        push(value);
        break;

default:
        push(t);
    }
}
result=pop();
result=result-48;
cout<<"The result="<<result;
}
```

```
int main()
{
    Implement ob;
    ob.input();
    ob.show();
    ob.intopost();
    ob.evalute();
    return 0;
}
```

/* circular linked list using ~~template~~ */

/* BST */ ⇒ All operation

/* polynomial */ ⇒ mul, Addition

```
/* polynomial */
#include <iostream>
#include <malloc>
using namespace std;
typedef struct node
{
  int coeff;
  int exp;
  struct node * link;
} NODE;
class poly
{
  private : NODE *p;
  public:
    poly()
    {
      p=NULL;
    }
    ~~poly(int, int);~~
    void display();
    void padd(int, int);
    void add(poly, poly);
    void mul(poly, poly);
};
void poly :: padd (int c, int e)
{
  NODE *t, *temp = p;
```

```
if (p== NULL || e > p→exp)
 {
   p = new NODE;
   p→coeff = c;
   p→exp = e;
   p→link = temp;
 }

else
 {
   while (temp != NULL)
     {
       if ( temp→exp == e)
         {
           temp→coeff = temp→coeff + c
           return;
         }
       if ( temp→exp > e && ( temp→link→exp < e
                              || temp→link == NULL))
         {
           z = new NODE;
           z→coeff = c;
           z→exp = e;
           z→link = temp→link
           temp→link = z;
           return;
         }
       temp = temp→link;
     }
   z→link = NULL;
   temp→link = z;
```

```
    }
  }
void poly :: add (poly t1, poly t2)
{
  NODE *z; NODE *x = t1.p, *y = t2.p;
  if (t1.p == NULL && t2.p == NULL)
    return;
  while (x != NULL && y != NULL)
  {
    if (p == NULL)
    {
      p = new NODE;
      z = p;
    }
    else
    {
      z -> link = new NODE;
      z = z -> link;
    }
  if (x -> exp < y -> exp)
  {
    z -> coeff = y -> coeff;
    z -> exp = y -> exp;
    y = y -> link;
  }
  else if (x -> exp > y -> exp)
  {
    z -> coeff = x -> coeff;
    z -> exp = x -> coeff;
    x = x -> link = .
```

```
}
else
{
    z->coeff = x->coeff + y->coeff;
    z->exp = x->exp;
    y = y->link;
    x = x->link;
}
}
while (x != NULL)
{
    if (p == NULL)
    {
        p = new NODE;
        z = p;
    }
    else
    {
        z->link = new NODE;
        z = z->link;
    }
    z->coeff = x->coeff;
    z->exp = x->exp;
    x = x->link;
}
while (y != NULL)
{
    if (p == NULL)
    {
        p = new NODE;
        z = p;
```

```
        }
        else
        {
            z -> link = new NODE;
            z = z -> link;
        }
        z -> exp = y -> exp;
        z -> coeff = y -> coeff;
        y = y -> link;
    }
    z -> link = NULL;
}
void poly :: mul ( poly t1, poly t2 )
{
    NODE *z, *x, *y;
    x = t1.p;
    y = t2.p;
    int c, e;
    z = y;
    if ( x == NULL && y == NULL )
        return;
    if ( x == NULL )
        p = y;
    else
    {
        if ( y == NULL )
            p = x;
        else
```

```
{
while (x != NULL)
    {
    while (y != NULL)
        {
        c = x -> coeff * y -> coeff;
        e = x -> exp + y -> exp;
        y = y -> link;
        Padd (c, e);
        }
    y = z;
    x = x -> link;
    }
}
}
void poly :: display ( );
{
    NODE *x = p;
    while (x != NULL)
    {
    if (x -> coeff < 0)
        {
        if (x -> exp != 0)
        {
        cout << x -> coeff << "x^" << x -> exp;
        }
        else
        {
        cout << " " << x -> coeff;
        }
    }
```

```
else
{
  if (n->coeff == 0)
    {
      n = n->link;
      continue;
    }
  else
    {
      if (n->exp != 0)
      cout << " + " << n->coeff << " x^ " <<
                       n->exp;
      else
      cout << " + " << n->coeff;
    }
  }
  n = n->link;
}
}
int main()
{
  poly ob1, ob2, ob3, ob4;
  int i, a, b, m;
  char ch, ch.a;

  cout << "Enter input for first polynomial ";
  do
  {
```

```
cout << " Enter coefficient";
cin >> a;
cout << " Enter exponent ";
cin >> b;
obj.padd (a, b);
cout << " Do you want to continue (Y/N)?";
fflush (stdin);
cin >> ch;
} while ( ch == 'y' || ch == 'Y' );
cout << " Enter input for 2nd polynomial:";
do
{
  cout << " Enter coefficient :";
  cin >> a;
  cout << " Enter exponent :";
  cin >> b;
  obj2.padd (a, b);
  cout << " Do you want to continue (Y/N)? ";
} while ( ch == 'y' || ch == 'Y' );
cout << " The first polynomial : ";
  obj.display ( );
cout << " The 2nd polynomial : ";
  obj2.display ( );
do
{
  cout << " Menu for polynomial operation ";
  cout << " 1. Polynomial addition 2. In 2. Polynomial
            Multiplication ";
```

```
cout << "Enter your choice:";
cin >> m;
switch (m)
{
case 1 : obj.display(); ob2.display();
         ob3.add (ob1, ob2);
         cout << "The polynomial addition
                 result is:";
         ob3.display();
         break;
case 2: ob1.display();
        ob2.display();
        ob4.mul(ob1, ob2);
        cout << "The polynomial multiplica
                result is:";
        ob4.display();
        break;
default: cout << "invalid choice";
}
cout << "Do you want to continue more
        operation (Y/N)?";
fflush (stdin);
cin >> cha;
} while (cha == 'y' || cha == 'Y');
return 0;
}
```

```
/* BST */

#include <iostream>
#include <malloc.h>
#include <stdio.h>
using namespace std;
Typedef struct node
{
  struct node *left;
  int data;
  struct node *right;
}NODE;
class BST
{
  private: NODE *root;
           int c;
  public: BST()
          {
            root = NULL;
            c = 0;
          }
          void maketree(NODE **, NODE *);
          void del1(NODE **, NODE *);
          void del(int);
          void getnode(int);
          void pretrav1(NODE *);
          void postrav1(NODE *);
          void intrav1(NODE *);
          void pretrav()
          {
            pretrav1(root);
```

```
    void inorder()
    {
      inorder1(root);
    }
    void postorder()
    {
      postorder1(root);
    }
    int count()
    {
      return c;
    }
};
int main()
{
  BST obj;
  int item, ch;
  do
  {
  cout<<"\n Menu for tree operation :";
  cout<<"\n 1. Insert. \n 2. Delete \n 3. Preorder
        display. \n 4. Inorder display. \n 5. Posto
        display. \n 6. Count non-leaf node. \n
        0. Exit.";
  cout<<"Enter your choice :";
  cin>>ch;
  if(ch<0 || ch>6)
    continue;
  switch(ch)
  {
```

```
case 1: cout << "\n Enter item to insert:";
        cin >> item;
        obj.getnode(item);
        break;
case 2: cout << "\n Enter item to delete;"
        cin >> item;
        obj.del(item);
        getch();
        break;
case 3: cout << "\n\n";
        obj.pretrav();
        getch();
        break;
case 4: cout << "\n\n";
        obj.intrav();
        getch();
        break;
case 5: cout << "\n\n";
        obj.postrav();
        getch();
        break;
case 6: printf("\n No. of Non leaf node
                = %d", obj.count());
        getch();
        break;
case 0: cout << "\n The END";
        getch();
}
}while(ch!=0);
return 0;
}
```

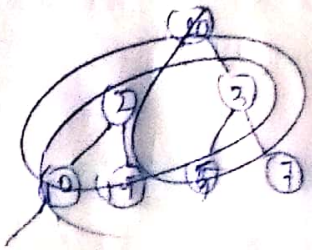
```
void BST :: getnode( int item)
{
  NODE *n = (NODE *) malloc(sizeof(NODE)); // new NODE
    n -> data = item;
    n -> left = n -> right = NULL;
    maketree ( &root, n);
}

void BST :: maketree (NODE **p, NODE *n)
{
    if ( *p == NULL)
      *p = n;
    else if ( (*p) -> data > n -> data)
        maketree (&((*p) -> left), n);
    else if ((*p) -> data <= n -> data)
        maketree (&((*p) -> right), n);
}

void BST :: introv1 (NODE *q)
{ if(q)                                  // if (q!=NULL
  {
    if ( q -> left != NULL || q -> right != NULL)
      {
        c++;
        introv1 ( q -> left);
        printf("%d", q -> data);
        introv1 (q -> right);
      }
  }
void BST :: pretrov1(NODE *n)
  {
    if (n)
    {
      printf("%d", n -> data);
      pretrov1 ( n -> left);
      pretrov1 ( n -> right);
```

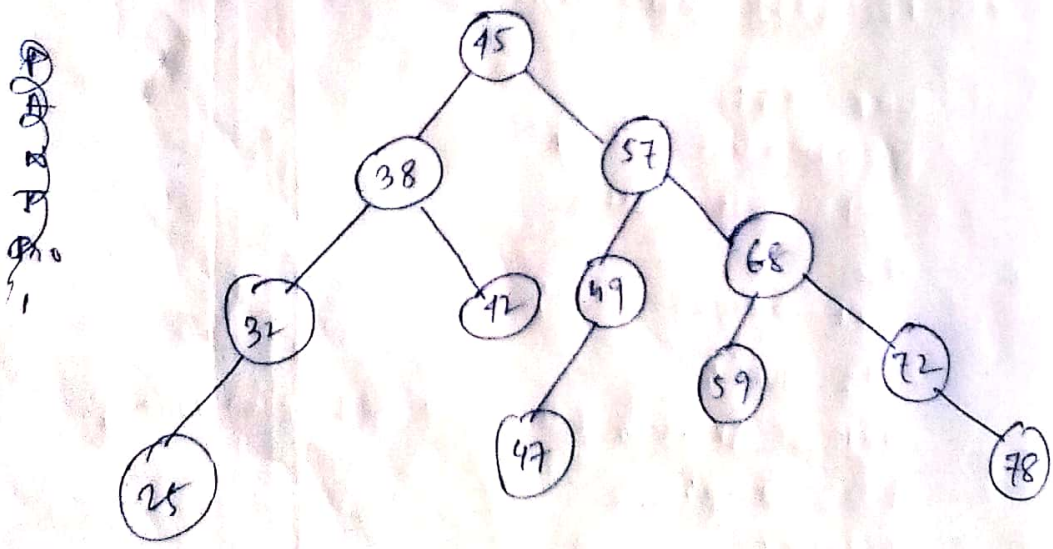
```
    }
}
void BST :: postorder1(NODE *s)
{
  if(s)
  {
    postorder1(s->left);
    postorder1(s->right);
    printf("%d", s->data);
  }
}
void BST :: del1(NODE **q, NODE *r1)
{
  NODE *cur, *prev, *temp;
  if(*q==NULL)
  {
    cout<<"Deletion is not possible";
  }
  else if((*q)->left==NULL && (*q)->right==NULL)
  {
    temp=*q;
    if(r1->left==*q)
      r1->left=NULL;
    else
      r1->right=NULL;
    *q=NULL;
    cout<<"\n Item deleted :"<<temp->data;
    delete temp;
    return;
  }
  else if((*q)->left==NULL)
  {
    temp=*q;
    (*q)->data=(*q)->right->data;
    *q=(*q)->right;
    del1(q, temp);
  }
```

```
else
{
  cur = (*q) -> right;
  prev = NULL;
  while (cur -> left != NULL)
    {
      prev = cur;
      cur = cur -> left;
    }
  (*q) -> data = cur -> data;
  *q = cur;
  del1 (q, prev);
}
}
void BST :: del (int item)
{
  NODE **p = &root;
  NODE *n, *n1;
  int f = 0;
  if (root == NULL)
    cout << "\n Item not found to delete";
  else
  {
    while (*p != NULL && f != 1)
      {
        if ((*p) -> data == item)
          {
            f = 1;
            del1 (p, n1);
          }
        else
          {
            n = *q;
            if (n -> data > item)
              {
                n1 = n;
                n = n -> left;
              }
```

```
else
{
    n1 = n;
    n = n -> right;
}
}
p = &n;
}
}
}
```

Food – (V)
water – (VII)
Surviving Tools – (III)
Rope – (II)
RAM – (IV)
Mirror – (VI)
chip – (I)
compose (X)
Map (IX)
Binocular (VIII)





Preorder ⇒ 45, 38, 32, 25, 42, 57, 49, 47, 68, 59, 72, 78

Inorder ⇒ 25, 32, 38, 42, 45, 47, 49, 57, 59, 68, 72, 78,

Postorder 25, 32, 42, 38, 47, 49, 59, 78, 72, 68, 57, 45,