STANDARD

-Dinesh Singh

PREFACE

The Importance of the book is for campus Recruitment for the Computer Science and Information Technology Students .The book of C++ is majorly concentrated on Operator Overloading and most of the Object Oriented Programming,and the book of DS is majorly Concentrated on how to manage the data in an efficient way though LinkedLists and Tree Structures . This also helps you in calculating the Time Complexities of widely used Algorithms for different purposes of Data Management .

-Dinesh Singh

Contents

1.Introduction (few c topics continuity by c+ + programming)

2.DataTypes and their sizes

3.Comma Operator

4.Member operator and Dot Operator

5.Upcasting and Downcasting

6.Pointers to arrays

7.Class and object with member functions and access specifiers

8.Scope resolution Operator

9.Constructor

i)NO argument Constructor

ii)Argumented |ParameterizedConstructor

iii) Copy Constructor

10.Destructor

11.Friend Functions

12.Inline Functions

13.Pointer to an object

14.Static Variables in Class

15.Static functions in class

16.Inheritance

- -Simple Inheritance
- -Multiple Inheritance
- -Hierarchical Inheritance
- -MultiLevel Inheritance
- -Hybrid Inheritance

17.Function Overloading

18.Operator Overloading

19.Polymorphism

20.Pure virtual Functions

21.Abstract Class

22.Files in c/c++

23.New Keyword Usage

1)Introduction (few C topics Continuity by C++ programming)

```
#include<iostream>
using namespace std;
//here in gcc compiler or bloodshed compiler cout is not direct
method so in order to use tht
//we define it as using namespace std; .. remove it and check error
for ur self
int main()
{
    int a;
    cout<<"hello world ";
    cout<<"\n enter a \n";
    //here in c++ there is no need of format specifiers
    cin>>a;
    //we can declare variable any where
    int b;
    cout<<"\n enter b\n";
    cin>>b;

    cout<<"\n values of a and b are "<<a<<"\t"<<b<<" respectively ";
    //here writing <<"/........."<<..<<..<<"...." is called
cascading

}
```

2)Datatypes and their sizes

```
#include<iostream>
using namespace std;
int main()
{   //datatypes and their sizes
    cout<<"Size of float : "<<sizeof(float)<< endl;
    cout<<"Size of double : "<<sizeof(double)<< endl;
    cout<<"Size of wchar_t : "<<sizeof(wchar_t)<< endl;
    cout<<"Size of short int : "<<sizeof(short int)<< endl;
    cout<<"Size of long int : "<<sizeof(long int)<< endl;
    cout<<"Size of char : "<<sizeof(char)<< endl;
    cout<<"Size of int : "<<sizeof(int)<< endl;
    return 0;
}
```

3)Comma Operator

```
#include<iostream>
using namespace std;
int main()
{   //example of , operator in c++
    int i, j;
    j =10;
    i =(j++, j+100);//her it will perform
    cout<< i;
    return 0;
} o/p->111
```

4)Member Operator and Dot Operator

```
#include<iostream>
using namespace std;
struct st
{
    int a;
};

int main()
{   struct st s;
    struct st *sample;
    sample=&s;
    sample->a=10;//this -> is called as member operator,for pointers
we use ->
    s.a=20;//this . is called as dot operator,for objects(instances)
we use .
    cout<<sample->a;
}
```

5)Upcasting and DownCasting (CASTING DATATYPES)

```
#include<iostream>
using namespace std;
int main()
{
    double a =21.09399;
    float b =10.20;
    int c ;
    c =(int) a;//down casting ->from higher value to lower value
    cout<< c << endl ;
    c =(int) b;//down casting ->from higher value to lower value
    cout<< c << endl ;
    int x=10;
    float y;
    y=x; //upcasting ,this is done by default by the compiler
itself,user need not write y=(float)x;
    cout<< y <<endl;
    return 0;
}
```

6)pointers to arrays

```
#include <iostream>
using namespace std;
int main()
{
    int a[10],*p,i,n;
    p=a;
    cout<<"enter size";
    cin>>n;
    cout<<"enter array";
    for(i=0;i<n;i++)
    {
        scanf("%d",p+i);//no need of & bcoz pointer point to address
only
        //cant use cin>>(p+i) bcoz it has no functionality for
pointers in my compiler
    }
    cout<<"\n elements are \n";
    for(i=0;i<n;i++)
    {
        cout<<*(p+i);
    }
}
```

7)<u>CLASS AND OBJECT WITH MEMBER FUNCTIONS AND ACCESS SPECIFIERS</u>

```
#include <iostream>
using namespace std;
class sample
{   //by default the access specifier is private so variables cant
be accessed directly so make it pubic for our example

                public: int a;
                        float b;

                private:int c;
    //member functions -,
public://this should be public bcoz we need to access from objects
    void set()
    {
        cout<<"\n enter c \n";
        cin>>c;
    }
    void get()
    {
        cout<<"c value is \n"<<c;
    }
};
int main()
{
  //ACTUAL C++ OBJECT ORIENTED PROGRAMMING
    sample s;
    s.a=10;
    cout<<s.a<<endl;
    s.b=10.0;
    cout<<s.b<<endl;
    //we cant access as s.c inorder to access c value we use member
functions
    s.set();
    s.get();
}
```

8)SCOPE RESOLUTION OPERATOR

```
#include <iostream>
using namespace std;
//EXAMPLE OF SCOPE RESOLUTION OPERATOR :: for member functions
//an example to calculate area of rect

class area
{
    int l,b;

public:     void set();
            void get();
};
//instead of defining member functions inside class lets define it
outside

//returntype class :: method(arg) is the syntax for it

void area :: set()
{
    cout<<"enter l and b \n";
    cin>>l>>b;
}
void area :: get()
{
    cout<<"area is "<<l*b<<endl;
}
int main()
{
    area a;
    a.set();
    a.get();
}
```

ACCESS SPECIFIERS IN C++ ARE
-> PUBLIC ,which can be accessed from any where
-> PRIVATE ,which can be accessed by its objects
-> PROTECTED,which can be accessed by its objects and in add with its derived class

CONSTRUCTORS

9)NO ARGUMENT CONSTRUCTOR

```
#include <iostream>
using namespace std;
//CONSTRUCTORS
class sample
{
    int a;

public:
    //if only one constructor is there which is a no-argument
contstructor as below ,its not neccessary to write the constructor
```

```
    //compiler itself writes in the above case assigning value of a 
as 0
    sample()
    {
        a=10;
    }
    void get()
    {
        cout<<endl<<a;
    }
};

int main()
{
    sample s;
    s.get();
}
```

<u>10) ARGUMENTED CONSTRUCTOR</u>

```
#include <iostream>
using namespace std;
//CONSTRUCTORS
class sample
{
    int a;

public:
    //if only one constructor is there which is a no-argument 
contstructor as below ,its not neccessary to write the constructor
    //compiler itself writes in the above case assigning value of a 
as 0
    sample()
    {
        a=10;
    }
    void get()
    {
        cout<<endl<<a;
    }
};
class example
{
        int b;
public:

        example(int x)
        {
            b=x;
        }
        //now if we dont write the default (no-arguments 
constructor) its an error ,this is called constructor overloading
        //so user has to explicitly define it
        example()
        {
            b=0;
        }

        void get()
        {
            cout<<endl<<b;
        }
};

int main()
{
    example e(5);
    e.get();
}
```

**IF YOU DEFINE A ARGUMENTED CONSTRUCTOR YOU DEFINETELY NEED TO WRITE THE CODE FOR
DEFAULT CONSTRUCTOR**

11)How to Define The constructor Outside

```
#include <iostream>
using namespace std;
//defining constructor out side the class
class sample
{
    int x;
public:
    sample();
    sample(int x);
    void get();
};
//syntax is same as member function but no return type
sample :: sample()
{
    x=0;
}
sample ::sample(int x)
{
    this->x=x;//here this indicates the present object
}
void sample :: get()
{
    cout<<endl<<x<<endl;
}
int main()
{
    sample s2(10);
    s2.get();
    sample s1;
    s1.get();
}
```

THERE ARE THREE TYPES OF CONSTRUCTORS
-> DEFAULT CONSTRUCTOR

-> PARAMETERIZED CONSTRUCTOR
-> COPY CONSTRUCTOR

copy constructor

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to:

Initialize one object from another of the same type. s1=s2;

Copy an object to pass it as an argument to a function. fun(s1)

Copy an object to return it from a function. return s1

12) copy constructor

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

public:
    int a;
    //constructor
    sample()
    {
        a=0;
    }
    sample(int a)
    {
        this->a=a;
    }
    //copy constructor
    sample(sample &s)
    {
        a=s.a;//copying the value
    }
    void get()
    {
        cout<<"\n value of a is "<<a<<endl;
    }

};
// inorder to do this we need to define the attributes as public
because we are using it from outside in functions
void get1(sample s2)
{
    cout<<"\n value of a is "<<s2.a<<endl;
}



int main()
{
    sample s1;
    sample s2(5);
    s2.get();
    s1=s2;
    s1.get();
    get1(s2);
}
```

13)DESTRCUTOR

```
#include<iostream>
using namespace std;
class sample
{
    int x;
public:
    sample()
    {
        x=0;
    }
    sample(int x)
    {
        this->x=x;
    }
    //destructor will be invoked automatically by the compiler at
end of program
    ~sample()
    {
        cout<<endl<<"object getting destroyed "<<endl;
    }
};
int main()
{
    sample s(5);

}
```

HOW TO ACCESS THE PRIVATE VARIABLES WITHOUT MEMBER FUNCTIONS ? ANS-> FRIEND FUNCTIONS

1)friend functions are the function which access the variables which are private with out member functions

2)we use friend keyword and declare it in class and define it outside , follow the below program

14)FRIEND FUNCTION

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

```
{
    int x;
public:
    sample()
    {
        x=0;
    }
    sample(int x)
    {
        this->x=x;
    }
    //destructor will be invoked automatically by the compiler at
end of program
    ~sample()
    {
        cout<<endl<<"object gettig destroyed "<<endl;
    }
    friend void get(sample s);
    void display()
    {
        cout<<endl<<x<<endl;
    }
};
void get(sample s)
{
    cout<<endl<<s.x<<endl;
}
int main()
{   //check the output here object gets destroyed because obj is
gettting created in get method also
    sample s1(5);
    sample s2(10);
    s1.display();
    get(s2);
}
```

INLINE FUNCTIONS

GENERALLY IN CASE OF NORMAL FUNCTIONS,THE CONTROL FLOW JUMPS FROM MAIN TO THE FUNCTION LOCATION
BUT IN CASE OF INLINE FUNCTIONS , THE INLINE FUNCTION BODY GETS SUBSTITUTED IN PLACE OF THE FUNCTION CALL
GENERALLY INLINE FUNCTIONS ARE USED FOR SMALLER BODY AND FOR FASTER EXECUTION

15)INLINE FUNCTIONS

```
#include<iostream>
using namespace std;
inline void hello()
{
    cout<<"\n hello world ";
}
inline void display(int x)
```

```
{
    cout<<endl<<x<<endl;
}
int main()
{
    cout<<"inline functions are equivalent to macros in c"<<endl;
    cout<<"inline functions are used to sub the functions body in
the main function";
    hello();
    display(10);
    display(20);
    display(30);
}
```

1)POINTER TO AN OBJECT

```
#include<iostream>
using namespace std;
class sample
{
    int x;
public:
    int y;
    sample()
    {
        x=0;
        y=0;
    }
    sample(int x)
    {
        this->x=x;
        y=x;
    }
    void display()
    {
        cout<<endl<<x<<endl;
    }
};
int main()
{
    sample s(10);
    sample *ptr;
    ptr=&s;
    ptr->display();
    cout<<endl<<ptr->y<<endl;
}
```

2)Static variables in class

THE STATIC VARIABLES ARE CREATED ONLY ONCE ,FROM THEN THEY ARE MODIFIED

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

class sample {
public:
    static int x;
    void incr()
    {
        x++;
        cout<<endl<<x<<endl;
    }

};
// Initialize static member of class Box
int sample::x= 0;
int main(void)
{
    sample s;
    s.incr();
    sample s2;
    s2.incr();
    sample s3;
    s3.incr();
}
```

3)<u>static functions in class</u>
THE STATIC FUNCTIONS ARE ACCESSED BY SCOPE RESOLUTION OPERATOR AS SHOWN IN THE PROGRAM

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

class sample {
public:
    static int x;
    void incr()
    {
        x++;
        cout<<endl<<x<<endl;
    }
    static int count()
    {
        return x;
    }

};
// Initialize static member of class Box
int sample::x= 0;
int main(void)
{
    sample s;
    s.incr();
    sample s2;
    s2.incr();
    sample s3;
    s3.incr();
    cout<<endl<<"using static functions "<<endl;
    int x=sample :: count();
    cout<<"the value of x at last is "<<x<<endl;
}
```

INHERITANCE

1)INHERITANCE IS A CONCEPT IN WHICH BASE CLASS PROPERTIES ARE INHERITED BY THEIR DERIVED CLASS DEPENDING ON THE ACCESS SPECIFIER SPECIFIED
2)THERE ARE DIFFERENT TYPES OF INHERITANCE
->SIMPLE INHERITANCE
->MULTIPLE INHERITANCE
->HIERARCHICAL INHERITANCE
->MULTILEVEL INHERITANCE
->HYBRID INHERITANCE
3)each time a derived class is invoked ,base class is implicitly

invoked by the compiler

4)SIMPLE INHERITANCE   base->derived

```
#include<iostream>
using namespace std;
class base
{
protected:int l,b;
public:
    void read()
    {
        cout<<endl<<"enter l and b\n";
        cin>>l>>b;
    }
};
class derived : public base
{
public:
    int cal()
    {
        return l*b;
    }
};
int main()
{
    derived d;
    d.read();
    int x=d.cal();
    cout<<x;
}
```

5)MULTIPLE INHERITANCE BASE1,BASE2->DERIVED

```
#include<iostream>
using namespace std;
//MULTIPLE INHERITANCE
class base1
{
protected:int l,b;
public:
    void read1()
    {
        cout<<endl<<"enter l and b\n";
        cin>>l>>b;
    }
};
class base2
{
protected:int h;
public:
    void read2()
    {
        cout<<endl<<"enter h\n";
        cin>>h;
    }
};
class derived :public base1,public base2
{
public:
    int cal()
    {
        return l*b*h;
    }
};

int main()
{
    derived d;
    d.read1();
    d.read2();
    int x=d.cal();
    cout<<endl<<x<<endl;
}
```

6)HIERARCHICAL INHERITANCE

```
#include<iostream>
using namespace std;
//HIERARCHICAL INHERITANCE
class base
{
public:
    base()
    {
        cout<<"\n base class"<<endl;
    }
};
class der1 :public base
{
public:
    der1()
    {
        cout<<"\n der1 class"<<endl;
    }
};
class der2 :public base
{
public:
    der2()
    {
        cout<<"\n der2 class"<<endl;
    }
};
class der3 :public base
{
public:
    der3()
    {
        cout<<"\n der3 class"<<endl;
    }
};
int main()
{
    der1 d1;
    der2 d2;
    der3 d3;
}
```

7)MULTILEVEL INHERITANCE

```
#include<iostream>
using namespace std;
//MULTILEVEL INHERITANCE
class base
{
public:
    base()
    {
        cout<<"\n base class"<<endl;
    }
};
class der1 :public base
{
public:
    der1()
    {
        cout<<"\n derived class 1"<<endl;
    }
};
class der2 :public der1
{
public:
    der2()
    {
        cout<<"\n derived class 2"<<endl;
    }
};
int main()
{
    der2 d;

}
```

8)HYBRID INHERITANCE

```
#include<iostream>
using namespace std;
//HYBRID INHERITANCE
class base
{
public:
    base()
    {
        cout<<"\n base class"<<endl;
    }
};
class der1 :public base
{
public:
    der1()
    {
        cout<<"\n derived class 1"<<endl;
    }
};
class der2 :public der1
{
public:
    der2()
    {
        cout<<"\n derived class 2"<<endl;
    }
};
class der3:public der2,public base
{
public:
    der3()
    {
        cout<<endl<<"\n derived class 3"<<endl;
    }
};
int main()
{
    der3 d3;

}
```

9)<u>function overloading</u>

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

//FUNCTION OVERLOADING
//same function name but diff defintions nd datatypes or diff number
of arguments
int sum(int x,int y)
{
    return x+y;
}
float sum(float x,float y)
{
    return x+y;
}
int sum(int x,int y,int z)
{
    return x+y+z;
}
int sum(int x,int y,int z,int a)
{
    return x+y+z+a;
}
```

```
int main()
{
    int a=sum(10,20);
    int c=sum(10,20,30);
    int d=sum(10,20,30,40);
    cout<<endl<<a<<endl<<c<<endl<<d<<endl;

}
```

OPERATOR OVERLOADING

usually +,-,*,/ only work with predefined datatypes,in order to make den work for
user defined datatypes like classes we must overload the operators

syntax:returntype(class) operator opr (class &obj)
eg:sample operator +(sample &s)

```
10)
#include <iostream>
using namespace std;
 //OPERATOR OVERLOADING FOR BINARY OPERATORS
class sample
{
    int x;
public:
    sample()
    {
        x=0;
    }
    sample(int x)
    {
        this->x=x;
    }
    sample operator+(sample &s)
    {
        sample s1;
        s1.x=this->x+s.x;
        return s1;
    }
    sample operator-(sample &s)
    {
        sample s1;
        s1.x=this->x-s.x;
        return s1;
    }
    sample operator*(sample &s)
    {
        sample s2;
        s2.x=this->x*s.x;
        return s2;
    }
    sample operator/(sample &s)
    {
        sample s3;
        s3.x=this->x/s.x;
        return s3;
    }
    void display()
    {
        cout<<endl<<x<<endl;
    }
};
int main()
{
    sample s1(100);
    sample s2(20);
    sample s3;
    s3=s1+s2;//here s1+(s2) is the function call type,always we
return the first variable in the operation
    s3.display();
    s3=s1-s2;
    s3.display();
    s3=s1*s2;
    s3.display();
    s3=s1/s2;
    s3.display();
}
```

11)<u>UNIARY OPERATOR OVERLOADING</u>

```
#include <iostream>
using namespace std;
 //OPERATOR OVERLOADING FOR UNARY OPERATORS
class sample
{
    int x;
public:
    sample()
    {
        x=0;
    }
    sample(int x)
    {
        this->x=x;
    }
    sample operator -()
    {
        x=-x;
        return sample(x);
    }
    sample operator ++()//this is for pre increment
    {
        x=x+1;
        return sample(x);
    }
    sample operator --()//this is for pre decrement
    {
        x=x-1;
        return sample(x);
    }
    sample operator ++(int)//this is for post increment
    {   x=x+1;
        return sample(x);
    }
    sample operator --(int)//this is for post decrement
    {
        x=x-1;
        return sample(x);
    }

    void display()
    {
        cout<<endl<<"value of x is "<<x<<endl;
    }
};
int main()
{
    sample s1(10);
    s1.display();
    sample s2;
    s2=-s1;
    s2.display();
    s2=-s1;
    s2.display();
    s2=++s2;
    s2.display();
    s2=--s2;
    s2.display();
    s2=s2++;
    s2.display();
    s2=s2--;
    s2.display();

}
```

12)<u>Relational Operators overloading</u>

```
#include <iostream>
using namespace std;
 //OPERATOR OVERLOADING FOR Relational OPERATORS
class sample
{
    int x;
public:
    sample()
    {
        x=0;
    }
    sample(int x)
    {
        this->x=x;
    }
    bool operator >(sample &s2)
    {
        sample s1;
        if(this->x > s2.x)
        {
            return true;
        }
        else
        {
            return false;
        }

    }
    bool operator <(sample &s2)
    {
        sample s1;
        if(this->x < s2.x)
        {
            return true;
        }
        else
        {
            return false;
        }

    }
    bool operator >=(sample &s2)
    {
        sample s1;
        if(this->x >= s2.x)
        {
            return true;
        }
        else
        {
            return false;
        }

    }
    bool operator <=(sample &s2)
    {
        sample s1;
        if(this->x <= s2.x)
        {
            return true;
        }
        else
        {
            return false;
        }

    }
};
int main()
{
    sample s1(10);
    sample s2(20);
    if(s1>s2)
    {
    cout<<endl<<"true \n";
    }
    else
    {
    cout<<endl<<"false \n";
    }
    sample s3(30);
    if(s3>=s2)
        cout<<endl<<"true \n";
    else
        cout<<endl<<"false \n";
}
```

13)<u>Overloading istream and ostream operators</u>

```
#include <iostream>
using namespace std;
 //OPERATOR OVERLOADING FOR istream and ostream OPERATORS
class sample
{

public:
    int y;
    friend istream &operator >> (istream &input,sample &s)
    {
        input >>s.y;
        return input;
    }
    friend ostream &operator << (ostream &output,sample &s)
    {
        output <<s.y <<endl;
        return output;
    }
};
int main()
{
    sample s;
    cout<<"\n enter object \n";
    cin>>s;
    cout<<"\n object is \n";
    cout<<s;
}
```

14)<u>overloading assignment operator (=)</u>

```
#include <iostream>
using namespace std;
 //OPERATOR OVERLOADING FOR assignment OPERATORS
class sample
{

public:
    int y;
    sample()
    {
        y=0;
    }
    sample(int y)
    {
        this->y=y;
    }
    void operator =(sample &s)
    {
        this->y=s.y;
    }
    void display()
    {
        cout<<endl<<y<<endl;
    }
};
int main()
{
    sample s1(10);
    sample s2;
    s2=s1;
    s2.display();
}
```

15)<u>OVERLOADING SUBSCRIPT OPERATOR [ ]</u>

```
#include <iostream>
using namespace std;
//OVERLOADING [ ] OF ARRAYS
class saray
{

public:
    int arr[10];
    saray()
    {
        int i;
        cout<<"enter elements \n";
        for(i = 0; i < 5 ; i++)
        {
            cin>>arr[i];
        }
    }
    int &operator[](int i)
    {
        return arr[i];
    }
};
int main()
{
    saray A;
    for(int i=0;i<5;i++)
        cout<<A[i]<<"\t";
    return 0;
}
```

assignment ,do similarly for % (binary), !(unary) ,!= (relational)

operators which can be overloaded in the same way are

1)+, -, !, ~, ++, --, true, false
2)+, -, *, /, %, &, |, ^, <<, >>
3)==, !=, <, >, <=, >=
4)&&, ||
5)+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
6)[]
7)(T)x

operators which cannot be overloaded are

.(Member Access or Dot operator)
?: (Ternary or Conditional Operator)
:: (Scope Resolution Operator)
.* (Pointer-to-member Operator)
sizeof (Object size Operator)
typeid (Object type Operator)

POLYMORPHISM

The following program doesn’t work because here the parent class function gets invoked

EXAMPLE 1

```
#include<iostream>
using namespace std;
class base
{
public:
    void display()
    {
        cout<<"\n this is base class \n";
    }
};
class der : public base
{
public:
    void display()
    {
        cout<<"\n this is der class \n";
    }
};
int main()
{
    base *b;
    der d;
    b=&d;
    b->display();
}
```

To rectify this we have a concept called as Virtual Functions and Pure Virtual Functions

WE JUST NEED TO MAKE THE getArea() method virtual

polymorphism

As you can see, each of the child classes has a separate implementation for the function area(). This is how polymorphism is generally used

VIRTUAL FUNCTION

1)A virtual function is a function in a base class that is declared using the keyword virtual

2)this is referred to as dynamic linkage, or late binding.

Problem Rectification

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

```
public:
    virtual void display()
    {
        cout<<"\n this is base class \n";
    }
};
class der : public base
{
public:
    void display()
    {
        cout<<"\n this is der class \n";
    }
};
int main()
{
    base *b;
    der d;
    b=&d;
    b->display();
}
```

PURE VIRTUAL FUNCTIONS

=0 in the pgm below ,is taken as pure virtual function by the compiler by seeing the =0 after the function header ;

```
#include<iostream>
using namespace std;
class base
{
public:
    virtual void display() =0;
    //this is pure virtual function

};
class der : public base
{
public:
    void display()
    {
        cout<<"\n this is der class \n";
    }
};
int main()
{
    base *b;
    der d;
    b=&d;
    b->display();
}
```

EXAMPLE 2

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

```
class base
{

public:
    virtual void getArea()
    {
       cout<<"base function";
    }
};
class rect : public base
{
public:

    void getArea()
    {
        cout<<"\n rectangle area \n";
    }

};

class tria : public base
{
public:
    void getArea()
    {   cout<<"\n triangle area \n";

    }
};

int main()
{
    base *b;
    rect r;
    b=&r;
    b->getArea();
    tria t;
    b=&t;
    b->getArea();
}
```

ABSTRACT CLASS

1) A class having one or more pure virtual function(which is virtual fn and has no body) then that type of class is called as abstract class

2) We cannot create an object for the abstract class

the following pgm gives an error because we are trying to create an object for an abstract class ,these abstract classes are also called as interfaces

eg)
```
#include <iostream>
using namespace std;
class base
```

```
{
public:
    virtual void getArea()=0;

};
int main()
{
    base b;
}
```

error:variable type “base” is an abstract class

so how to make it use den,we need to over ride the getArea() method by its derived classes as follows

```
#include <iostream>
using namespace std;
class base
{
public:
    virtual void getArea()=0;

};
class der : public base
{
public:
    void getArea()
    {
        cout<<"\n overridded successfully \n";
    }
};
int main()
{
    der d;
    d.getArea();
}
```

if we don’t write the getArea() method in der class it also becomes an abstract class,So in order to create an obj for der class we need to over ride the getArea()

this concept of interfaces has a large scope in OOPS

Files in C & C++

Reading A File and Displaying it on console

```
#include<iostream>

using namespace std;
int main()
{
    FILE *f1;
f1=fopen("/Users/dineshsingh/Documents/c++ Examples/c++_practise/c+
+_practise/File1.rtf","r");
    char ch;
    while((ch=fgetc(f1))!=EOF)
    {
        cout<<ch;//in c printf("%c",ch);
    }
    fclose(f1);
}
```

Reading A file and Copying the content into Another File

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

int main()
{
    FILE *f1,*f2;
    f1=fopen("File1.txt","r");
    f2=fopen("File2.txt","a");
    char c;
    while((c=fgetc(f1))!=EOF)
    {
        fputc(c,f2);
    }
    fclose(f1);
    fclose(f2);
}
```

Writing a file by entering the content through console

```
#include<iostream>
using namespace std;
int main()
{
    FILE *f;
    int n;
    f=fopen("File1.txt","w");
    cout<<"\n enter the input into file.. \n I will Write into the
File(File1.txt) \n";
    for(int i=0;i<12;i++)
    {
        scanf("%c",&n);
        putw(n,f);
    }
    fclose(f);
}
```

Usage of feof(file *) (specifies End of File or not)

```
#include<iostream>
using namespace std;
int main()
{
    //USAGE OF FEOF
    FILE *f;
    int n;
    f=fopen("File1.txt","r");
    while(!feof(f))
    {
        char ch=fgetc(f);
        cout<<ch;
    }
}
```

Usage of ferror(file *) (specifies error or not)

```
#include<iostream>
using namespace std;
int main()
{
    //USAGE OF FEOF
    FILE *f;
    f=fopen("File1.txt","r");
    if(ferror(f))
        cout<<"\n error \n";
    else
        cout<<"\n no error \n";

}
```

Usage of rewind(file *) (makes the file pointer point the first letter of file)

```
#include<iostream>
using namespace std;
int main()
{
    //  USAGE OF REWIND
    FILE *f1,*f2;
f1=fopen("File1.txt,"r");
    char ch;
while((ch=fgetc(f1))!=EOF)
{
    cout<<ch;
    //by the ned f1 is with EOF
}
    cout<<endl<<endl<<endl;
    rewind(f1);
    ch=fgetc(f1);//points to first letter in File f1
    cout<<ch;


}
```

Usage of ftell(file *) and fseek(file *,int,int |SEEK_pos) (ftell gives the length and fseek is used to search)

fseek(file *,6,1) -> sets pointer after 6 digits from current loc
fseek(file *,-6,1) ->sets pointer before 6 digits from current loc
fseek(file *,-5,2) ->sets pointer before 5 digits from end
fseek(file *,-5,SEEK_END) -> sets pointer before 5 digits from end
fseek(file *,5,0)->sets pointer after 5 digits from start

KeyWords
SEEK_SET ->from start
SEEK_CUR ->from current pos
SEEK_END ->End of File

all these functions returns zero if successful, else it returns nonzero value

```
#include<iostream>
using namespace std;
int main ()
{
        FILE *f;
        long len;
        f = fopen("file.txt", "r");
        if( f == NULL )
        {
            perror ("Error opening file");
            return(-1);
        }
        fseek(f, 0, SEEK_END);
        len = ftell(f);
        fclose(f);
        cout<<"size of file is \t "<<len;
        return(0);
}
```

Convert Binary Number to Decimal Number

```
#include<iostream>
#include<math.h>
using namespace std;
int bd(int n) /* Function to convert binary to decimal.*/

{
    int decimal=0, i=0, rem;
    while (n!=0)
    {
        rem = n%10;
        n=n/10;
        decimal =decimal + rem*pow(2,i);
        ++i;
    }
    return decimal;
}
int main ()
{
    //Binary number to decimal number
    cout<<"enter binary number";
    int n;
    cin>>n;
    int x=bd(n);
    cout<<"Decimal Equivalent is "<<x<<endl;

}
```

Exceptions

Creating our own Exception in c++

```
#include <iostream>
#include <exception>
using namespace std;
//Creating out own Exceptions in C++
class myexception: public exception
{
    virtual const char* what() const throw()
    {
        return "Please dont enter 9";
    }
} ex;

int main () {
    try
    {
        cout<<"enter a number \n";
        int n;
        cin>>n;
        if(n==9)
            throw ex;
    }
    catch (exception& e)
    {
        cout << e.what() << '\n';
    }
    return 0;
}
```

New Keyword

usage of new keyword
equivalent to malloc

```
#include <iostream>
#include <exception>
using namespace std;
//  USAGE OF NEW KEYWORD (instead of malloc ,realloc,calloc methods
alternative for tht is new keyword
struct sample
{
    int x;
};
struct sample *p;
int main()
{
    p=new sample;
    cout<<"enter digit \n";
    cin>>p->x;
    cout<<p->x<<endl;


}
```

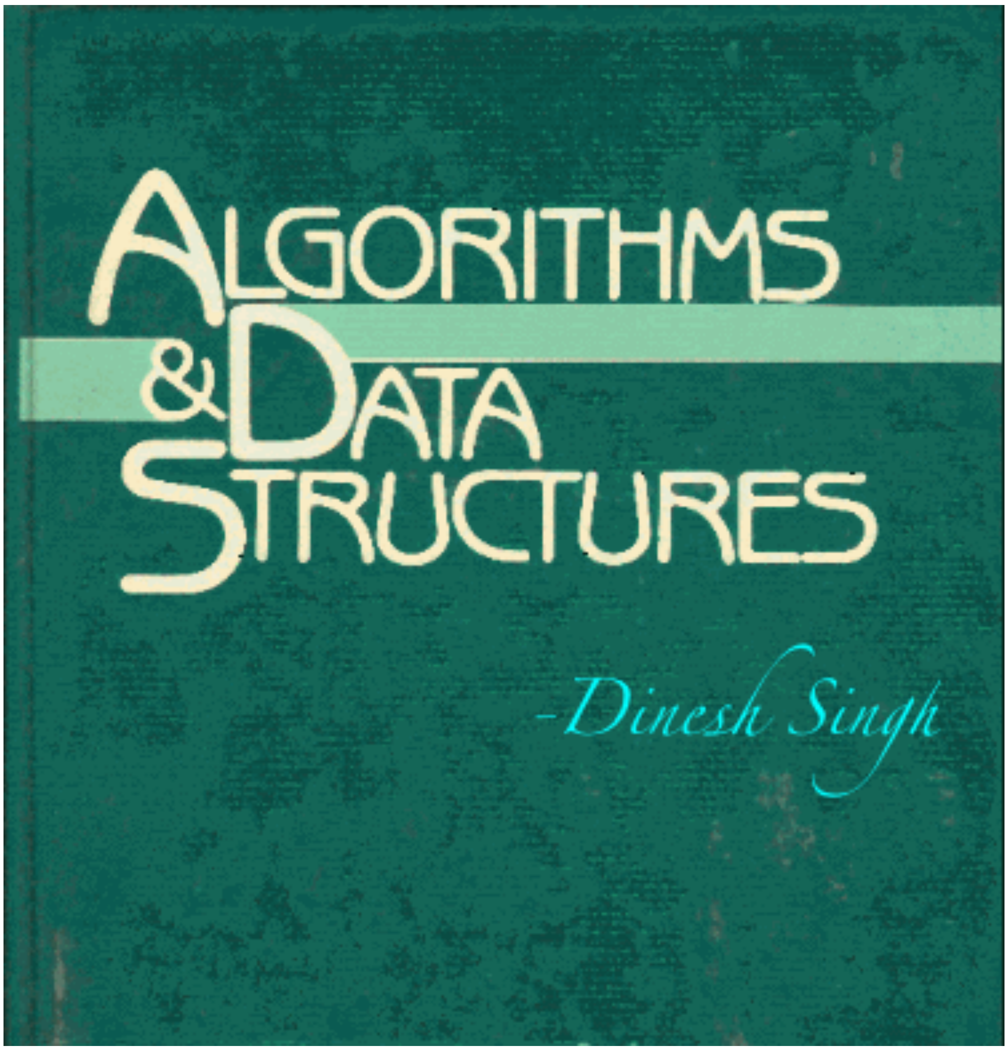
Self Referential Structures

```
#include <iostream>
#include <exception>
using namespace std;
//  Usage of Self Refrential Struct
struct sample
{
    int x;
    sample *s;
};
```

```
struct sample *p,*q;
int main()
{
    p=new sample;
    cout<<"enter digit \n";
    cin>>p->x;
    q=new sample;
    p->s=q;
    p=q;
    q->s=NULL;
    cout<<p->x<<endl;
    cout<<p->s<<endl;
    cout<<q->x<<endl;
    cout<<q->s<<endl;


}
```

Delete Keyword

```
#include <iostream>
#include <exception>
using namespace std;
//  Usage of Self refrential  Struct
struct sample
{
    int x;
    sample *s;
};
struct sample *p,*q;
int main()
{
    p=new sample;
    cout<<"enter digit \n";
    cin>>p->x;
    q=new sample;
    p->s=q;
    p=q;
    q->s=NULL;
    cout<<p->x<<endl;
    cout<<p->s<<endl;
    cout<<q->x<<endl;
    cout<<q->s<<endl;
    //delete keyword
    delete p;

    //delete is equivalent to free in c



}
```


ALGORITHMS
& DATA
STRUCTURES
-Dinesh Singh

CONTENTS

1.Stack

2.Queue

3.Circular Queue

4.Stack using LinkedList

5.Queue using LinkedList

6.Simple LinkedList

7.Circular LinkedList

8.Doubley Linked List

9.Double Ended Queue(DEQUE)

10.Linear Search

11.Binary Search

12.Bubble Sort

13.Selection Sort

14.Quick Sort

15.Merge Sort

16.Tree Traversals

17.Insertion of a node into a Binary Search Tree
18.Deletion of a node from a Binary Search Tree
19.A.V.L Tree(Self Balancing Tree Introduction)
20.Insertion of a node into an A.V.L tree Intro and Balancing of Tree

ABSTRACT DATATYPES

Stack -> Last In first Out DataStructure
push ->to insert an element
pop ->to delete an element
Advantages
1)Basic and Easy to Implement
2)Used in Parsing
3)Used in recursive function procedures
4)used in function calling
5)Expression Evaluation and Expression Conversion

- **Infix to Postfix**
- **Infix to Prefix**
- **Postfix to Infix**
- **Prefix to Infix**

6)used in towers of hanoi

```
#include<iostream>
using namespace std;
int main()
{
    //DEMO ON STACK
    int a[10],n=0,ele,i=0;
    char ch;
    while(1)
    {
        cout<<"\n enter your choice \n 1.Create 2.Display 3.push
4.pop 5.Exit \n";
        cin>>ch;

        switch(ch)
        {
        case '1':cout<<"enter the size of the stack \n";
                cin>>n;
                cout<<"enter elements -9 to quit";
            while(1)
            {
                cin>>ele;
                if(ele==-9)
                    break;
                else
                {   if(i==n)
                    {
                        cout<<"\n stack full ..\n sorry can't push
any more !";
                        break;
                    }

                    a[i]=ele;
                    i++;
                }
            }
        top==i;
                break;
        case '2':cout<<"\n Displaying your Stack .. \n";
                for(int k=i;k>0;k–)
                    cout<<a[k];
                break;
        case '3':if(i==n)
                {
                    cout<<" \n sorry stack size full !! cant push
any more \n";
                    break;
                }
                i++;
                cout<<" \n Enter element \n";
                cin>>ele;
                a[i]=ele;
                break;
        case '4':if(i==0)
                {
                    cout<<"\n Stack underflow ,can't pop anymore
\n";
                    break;
                }
                i--;
                break;
        case '5':exit(1);
                break;
        default:cout<<"\n enter valid input \n";
                break;
        }
    }
}
```

sample output

```
 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
1
enter the size of the stack
5
enter elements -9 to quit1
2
3
4
5
6

 stack full ..
 sorry can't push any more !
 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
2
```

```
 Displaying your Stack ..
54321
 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
4

 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
2

 Displaying your Stack ..
4321
 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
3

 Enter element
5

 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
3

 sorry stack size full !! cant push any more

 enter your choice
 1.Create 2.Display 3.push 4.pop 5.Exit
5
Program ended with exit code: 1
```

<u>2)Queue DataStructure (First IN First Out)</u>
insert->to insert an element;
delete->to delete an element;

front ->from here we will delete , i.e., starting of array
rear ->from here we will insert , i.e., end of array

<u>Advantages/Usage</u>

1)first-come-first-serve principle. Network packets, I/O requests
2)few operating systems are based on queue data structure
<u>Disadvantages</u>
1)In queues there is a fixed entry point and a fixed exit point. since structures are heavily used in sorting, searching and indexing these kind of restriction increases the time and space requirements of the algorithm.
2)It is difficult to implement recursive calls and function calls with queues,its much easier with stacks

```
#include<iostream>
using namespace std;
int main()
{
    int a[10],n,front=-1,rear=-1,t=0;
    char ch;
    int ele;
    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete 5.Exit \n";
        cin>>ch;
        switch(ch)
        {
            case '1':

                    cout<<"\n enter size of the queue \n";
                    cin>>n;
                    cout<<"\n enter elements ,-9 to exit \n";
                    while(1)
                    {
                        cin>>ele;
                        if(ele==-9)
                            break;
                        if(front==-1 && t==n)
                        {
                            cout<<"\n queue overflow \n ";
                            break;
                        }
                        t++;
                        rear++;
                        a[rear]=ele;
                    }
                break;

            case '2':

                for(int i=front+1;i<=rear;i++)
                    cout<<a[i]<<"\t";
                break;

            case '3':

                if(front==-1 && rear==n)
                {

                    cout<<"\n queue overflow \n ";
                    break;

                }
                rear++;
                cout<<"\n enter element \n";
                cin>>ele;
                a[rear]=ele;
                break;

            case '4':

                if(front==rear)
                {

                    cout<<"\n queue underflow \n ";
                    break;

                }
                front++;
                break;

           case '5':exit(1);
            break;
           default:cout<<"\n enter valid input \n";
                break;
        }
    }
}
```

output

```
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
1

 enter size of the queue
5

 enter elements ,-9 to exit
1
2
3
4
5
6

 queue overflow

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3

 enter element
```

```
6

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
2   3   4   5   6
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
5
Program ended with exit code: 1
```

Circular Queue

```
#include<iostream>
using namespace std;
//Circular Queue
int main()
{
    int a[10],n=0,front=-1,rear=-1,temp=0;
    char ch;
    int ele;
    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete 5.Exit \n";
        cin>>ch;
        switch(ch)
        {
            case '1':
                cout<<"\n Enter size of the Circular Queue \n";
                cin>>n;
                cout<<"\n Enter elements ,-9 to exit \n";
                while(1)
                {
                    cin>>ele;
                    if(ele==-9)
                        break;
                    if(temp==n)
                    {
                        cout<<"\n Circular Queue Overflow \n";
                        break;
                    }
                    rear++;
                    a[rear]=ele;
                    temp++;
                }
                break;

            case '2':cout<<front <<"\t"<<rear<<"\n";
                if(front<rear)
                {   for(int i=front+1;i<=rear;i++)
                    cout<<a[i];
                }
                else
                {
                    for(int i=front+1;i<=n-1;i++)
                        cout<<a[i];
                    for(int i=0;i<=rear;i++)
                        cout<<a[i];
                }

                break;
            case '3':cout<<front<<" "<<rear<<endl;
                if(temp==n)
                {
                    cout<<"\n Circular Queue Overflow \n";
                    break;
                }
                else if(temp==n-1)
                    rear=0;
                else
                    rear++;
                cout<<"\n enter element \n";
                cin>>ele;
                temp++;
                a[rear]=ele;
                break;
            case '4':
                if(temp==0)
                {
                    cout<<"\n Circular Queue UnderFlow \n";
                    break;
                }
                if(front==n-1)
                    front=-1;
                else
                    front++;
                temp--;
                break;
            case '5':exit(1);
                break;
            default:cout<<"\n please enter valid input \n";
                break;
        }

    }
}
```

<u>output</u>

```
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
1

 Enter size of the Circular Queue
5

 Enter elements ,-9 to exit
1
2
3
4
5
6
```

```
 Circular Queue Overflow

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
-1  4
12345
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3
-1 4

 Circular Queue Overflow

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
0   4
2345
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3
0 4

 enter element
6

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2
0   0
62345
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3
0 0

 Circular Queue Overflow

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
```

Advantages

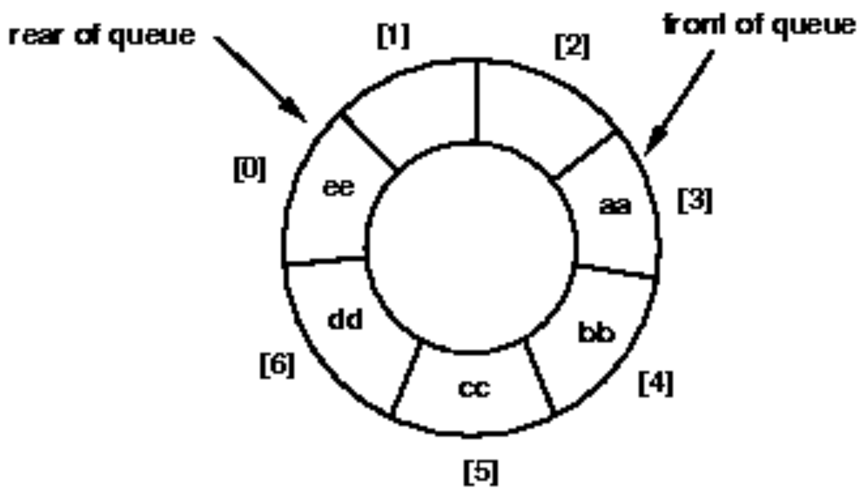
1)first-come-first-serve principle. Network packets, I/O requests
2)few operating systems are based on queue data structure
3)Blank Spaces are refilled in circular queues

Disadvantages

*1)*It is difficult to specify the front and rear positions
2)In queues there is a fixed entry point and a fixed exit point. since structures are heavily used in sorting, searching and indexing these kind of restriction increases the time and space requirements of the algorithm.
3)It is difficult to implement recursive calls and function calls with queues,its much easier with stacks

4)Elements can’t be inserted and deleted in between

5)searching and indexing is difficult



LINKED LISTS

Implementation of Stack with LinkedLists

```
#include<iostream>
using namespace std;
typedef struct stack
{   int x;
    struct stack *link;
}stack;
stack *p=NULL,*top=NULL;
int main()
{   char ch;
    int ele;
    while(1)
    {

        cout<<"\n 1.create 2.display 3.push 4.pop \n";
        cin>>ch;
        switch(ch)
        {

            case '1':

                cout<<"\n enter element ,-9 to quit \n";
                while(1)
                {
                    cin>>ele;
                    if(ele==-9)
                    {
                        break;
                    }
                    p=new stack;
                    p->x=ele;
                    p->link=top;
                    top=p;
                }
                break;

            case '2':

                p=top;
                while(1)
                {   if(p==NULL)
                    break;
                    cout<<"\t"<<p->x;
                    p=p->link;
                }
                break;

            case '3':

                cout<<" \n enter element \n";
                cin>>ele;
                p=new stack;
                p->x=ele;
```

```
                    p->link=top;
                    top=p;
                    break;

                case '4':
                    if(top==NULL)
                    {
                        cout<<"\n Stack Underflow \n";
                        break;
                    }
                    p=top;
                    top=top->link;
                    delete(p);
                    break;

                case '5':exit(1);
                default:cout<<"\n enter valid input \n";

            }
        }
}
```

output:

```
 1.create 2.display 3.push 4.pop
1

 enter element ,-9 to quit
1
2
3
-9

 1.create 2.display 3.push 4.pop
2
    3   2   1
 1.create 2.display 3.push 4.pop
3

 enter element
4

 1.create 2.display 3.push 4.pop
2
    4   3   2   1
 1.create 2.display 3.push 4.pop
4

 1.create 2.display 3.push 4.pop
2
    3   2   1
 1.create 2.display 3.push 4.pop

4
```

```
 1.create 2.display 3.push 4.pop
2
    2   1
 1.create 2.display 3.push 4.pop
4

 1.create 2.display 3.push 4.pop
2
    1
 1.create 2.display 3.push 4.pop
4

 1.create 2.display 3.push 4.pop
2

 1.create 2.display 3.push 4.pop
4

 Stack Underflow

 1.create 2.display 3.push 4.pop
```

Queue Implementation With LinkedList

```
#include<iostream>
using namespace std;
typedef struct queue
{
    int x;
    queue *link;
}queue;
queue *p=NULL,*front=NULL,*rear=NULL;
int main()
{   char ch;
    int ele;
    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete 5.Exit \n";
        cin>>ch;
        switch(ch)
        {
            case '1':
                cout<<"\n enter element ,-9 to exit \n";
                while(1)
                {
                    cin>>ele;
                    if(ele==-9)
                        break;
                    p=new queue;
                    p->x=ele;
                    p->link=NULL;
                    if(front==NULL && rear==NULL)
                    {
                        front=p;
                        rear=p;

                    }
                    else
                    {
                        rear->link=p;
                        rear=p;
                    }

                }
                break;
            case '2':
                cout<<"\n Displaying .. \n";
                p=front;
                while(1)
                {
                    if(p==NULL)
                    {
                        break;
                    }
                    cout<<"\t"<<p->x;
                    p=p->link;

                }
                break;
            case '3':
                cout<<"\n enter element to insert \n";
                cin>>ele;
                p=new queue;
                p->x=ele;
                p->link=NULL;
                rear->link=p;
                rear=p;

                break;
            case '4':
                if(front==NULL)
                {
                    cout<<"\n queue underflow ";
                    break;
                }
                p=front;
                front=front->link;
                if(front==NULL)
                    rear=NULL;
                delete(p);
                break;
            case '5':exit(1);
                break;
            default:cout<<"\n Please enter valid choice \n";
                break;
        }
    }
}
```

```
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
1

 enter element ,-9 to exit
1
2
3
4
5
-9

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 Displaying ..
    1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3

 enter element to insert
6

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 Displaying ..
    1   2   3   4   5   6
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 Displaying ..
    2   3   4   5   6
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 Displaying ..
    2   3   4   5   6
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
```

```
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 Displaying ..

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

 queue underflow
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
5
Program ended with exit code: 1
```

SimpleLinkedList

```
#include <iostream>
using namespace std;
typedef struct linkedlist
{
    int x;
    linkedlist *link;
}linkedlist;
linkedlist
*p,*head=NULL,*tail=NULL,*latest=NULL,*dummy=NULL,*help=NULL;
int main()
{
    int ele,k=0;
    char ch;
    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete 5.Search
6.Length 7.Revers 8.DeleteAll 9.exit \n ";
        cin>>ch;
        switch(ch)
        {
            case '1':
                cout<<"\n enter elements ,-9 to exit \n";
                    while(1)
                    {
                        cin>>ele;
                        if(ele==-9)
                            break;
                        p=new linkedlist;
                        p->x=ele;
                        p->link=NULL;
                        if(head==NULL)
                        {
                            head=p;
                            tail=p;

                        }
                        else
                        {
```

```
                tail->link=p;
                tail=p;
            }
        }
    break;
case '2':
    p=head;
    while(p!=NULL)
    {
        cout<<p->x<<" \t ";
        p=p->link;
    }
    break;
case '3':{
    int temp=1;
    cout<<"\n Enter position to Insert from start \n";
    cin>>k;
    p=head;
    while(1)
    {   if(temp==k)
            break;
        p=p->link;
        temp++;
    }
    latest=new linkedlist;
    if(k==0)
    {
        latest->link=head;
         head=latest;
        break;
    }
    cout<<"\n enter element to insert \n";
    cin>>ele;
    latest->x=ele;
    latest->link=p->link;
    p->link=latest;
    break;
}
case '4':
{
    int temp=1;
    cout<<"\n Enter Position to Delete \n";
    cin>>k;

    if(k==1)
    {
      p=head;
      head=head->link;
      delete(p);

    }
    else
    {
        p=head;
```

```
            dummy=NULL;
            temp=1;
            while(temp<k)
            {
                dummy=p;
                p=p->link;
                temp++;
            }

            dummy->link=p->link;
            cout<<"element delted is "<<p->x;
            delete(p);
        }
        break;
    }
    case '5':
        cout<<"\n enter element to search \n";
        cin>>ele;
        p=head;
        int flag;
        while(p!=NULL)
        {
            if(p->x==ele)
            {
                flag=1;
                break;
            }
            p=p->link;
        }
        if(flag==1)
            cout<<"\n element found !! \n";
        else
            cout<<"\n element not found !! \n";
        break;
    case '6':
    { int len=0;
        p=head;
        while(p!=NULL)
        {   len++;
            p=p->link;
        }
        cout<<"\n length is "<<len;
        break;
    }
    case '7':
    {
        p=head;
        dummy=NULL;
        while(p!=NULL)
        {
            help=dummy;
            dummy=p;
            p=p->link;
            dummy->link=help;
```

```
                }
                head=dummy;
                break;
            }
            case '8':
                while(head!=NULL)
                {
                    p=head;
                    head=head->link;
                    delete (p);
                }
                break;
            case '9':exit(1);
                break;
        }

    }
}
```

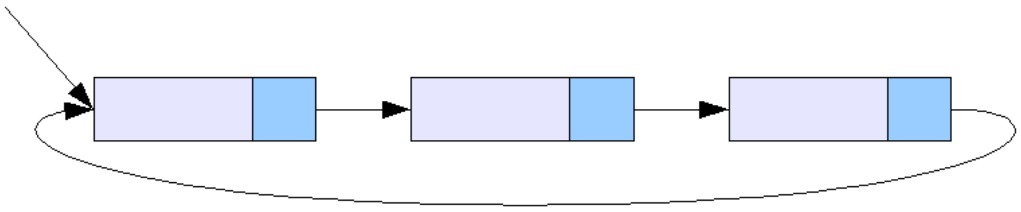
Output:

```
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 1

 enter elements ,-9 to exit
1
2
3
-9

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 2
1    2   3
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 3

 Enter position to Insert from start
2

 enter element to insert
99

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 2
1    2   99      3
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 4

 Enter Position to Delete
3
```

```
element delted is 99
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 2
1    2   3
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 5

 enter element to search
3

 element found !!

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 6

 length is 3
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 7

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 2
3    2   1
 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 8

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 2

 1.Create 2.Display 3.Insert 4.Delete 5.Search 6.Length 7.Revers
8.DeleteAll 9.exit
 9
Program ended with exit code: 1
```

Circular LinkedList

IN case of circular linked list ,the end of the linked list is not null but it points to HEADER
HEADER: it has the same fields like linked list but the element field is not present,only link node is available

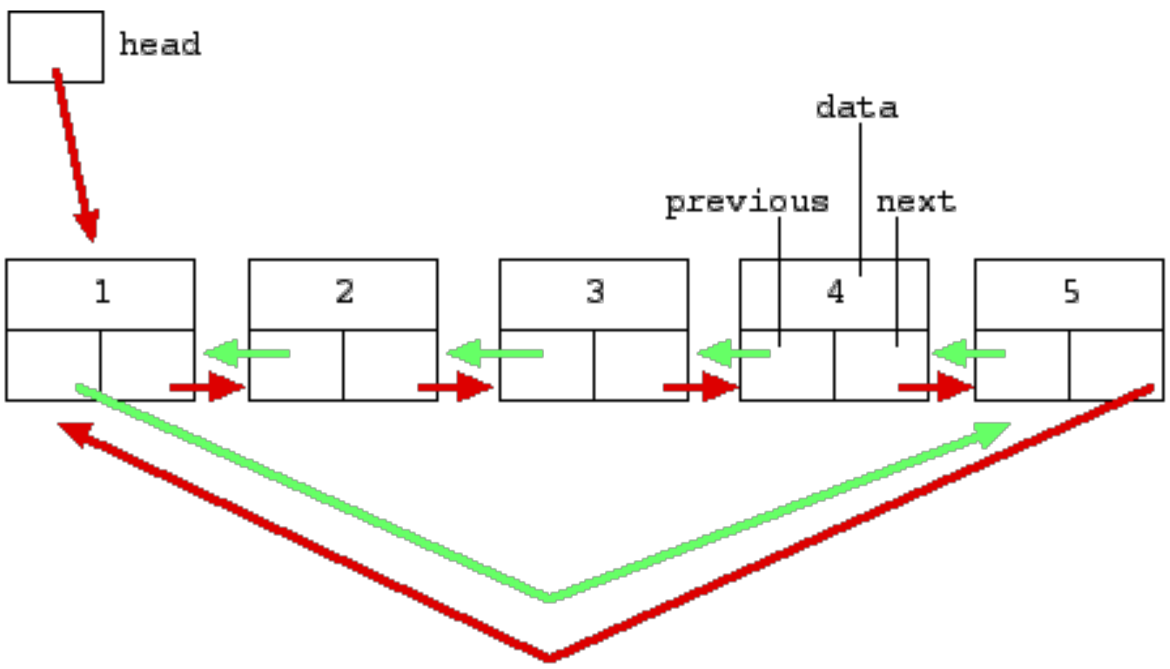


```
#include <iostream>
using namespace std;
typedef struct linkedlist
{
    int x;
    linkedlist *link;
}linkedlist;
linkedlist *header=NULL,*p=NULL,*temp=NULL,*dummy=NULL;


int main()
{   header =new linkedlist;
    header->link=header;
    temp=header;
    char ch;
    int ele,k=0;
    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete \n";
        cin>>ch;
        switch(ch)
        {
            case '1':
                cout<<"\n enter elelemts ,-9 to exit \n";
                while(1)
                {
                    cin>>ele;
                    if(ele==-9)
                        break;
                    p=new linkedlist;
                    p->x=ele;
                    temp->link=p;
                    temp=p;
                    p->link=header;
                }
                break;
            case '2':
                cout<<"\n displaying .. \n";
                p=header->link;
                while(p!=header)
                {
                    cout<<p->x<<"\t";
                    p=p->link;
                }
                break;
            case '3':
            {   cout<<"\n enter the position to insert \n";
                cin>>k;
                cout<<"\n enter element to insert \n ";
                cin>>ele;
                p=new linkedlist;
                p->x=ele;
                int i=0;
                temp=header;
                while(i<k)
                {
                    i++;
                    temp=temp->link;
                }
                p->link=temp->link;
                temp->link=p;
                break;
            }
            case '4':
            {   cout<<"\n enter the position to delete \n";
                cin>>k;
                dummy=header;
                temp=header->link;
                int i=1;
                while(i<k)
                {   i++;
                    dummy=temp;
                    temp=temp->link;
                }
                dummy->link=temp->link;
                delete(temp);
                break;
            }
            case '5':exit(1);
            default:cout<<"\n please enter valid choice \n";
                break;

        }
    }
}
output:

 1.Create 2.Display 3.Insert 4.Delete
1

 enter elelemts ,-9 to exit
1
2
3
4
5
-9
```

```
 1.Create 2.Display 3.Insert 4.Delete
2

 displaying ..
1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete
3

 enter the position to insert
3

 enter element to insert
 99

 1.Create 2.Display 3.Insert 4.Delete
2

 displaying ..
1   2   3   99  4   5
 1.Create 2.Display 3.Insert 4.Delete
4

 enter the position to delete
4

 1.Create 2.Display 3.Insert 4.Delete
2

 displaying ..
1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete
```

Doubley Linked List

In case of DOubley linkedList we can travel in both the direction



```
#include <iostream>
using namespace std;
typedef struct linkedlist
{
    int x;
    linkedlist *left,*right;
}linkedlist;
linkedlist *head=NULL,*temp=NULL,*dummy=NULL,*p=NULL;
int main()
{   int ele,k;
    char ch;

    while(1)
    {
        cout<<"\n 1.Create 2.Display 3.Insert 4.Delete 5.Exit \n";
        cin>>ch;
        switch(ch)
        {
            case '1':
            {
                cout<<"\n Enter elements -9 to quit \n";
                while(1)
                {
                    cin>>ele;
                    if(ele==-9)
                        break;
                    p=new linkedlist;
                    p->x=ele;
                    p->left=p->right=NULL;
                    if(head==NULL)
                    {
                        head=p;
                        temp=p;//for forward
                        dummy=p;//for backward
                    }
                    else
                    {

                        temp->right=p;
                        p->left=dummy;
                        temp=p;
                        dummy=p;
                    }
                }
                break;
            }
            case '2':
            {   char d;
                cout<<" \n 1.Forward 2.Backward \n";
                cin>>d;
                switch(d)
                {
                        case '1':
                        cout<<"\n forward displaying .. \n";
                        p=head;
                        while(p!=NULL)
                        {
                            cout<<p->x<<"\t";
                            p=p->right;
                        }
                        break;
                        case '2':
                        cout<<"\n Backward Displaying .. \n";
                        p=dummy;//dummy is the last node
                        while(p!=NULL)
                        {   cout<<p->x<<"\t";
                            p=p->left;
                        }
                        break;
                        default:cout<<"please enter valid input ";
                        break;
                }
                break;
            }
            case '3':
            {
                        cout<<"\n  Inserting .. \n";
                        p=head;
                        cout<<"\n enter position \n";
                        cin>>k;
                        cout<<"\n enter element \n";
                        cin>>ele;
                        p=new linkedlist;
                        p->x=ele;
                        p->left=NULL;
                        p->right=NULL;
                        if(k==0)
                        {   head->left=p;
                            p->right=head;
                            head=p;
                        }
                        else
                        {
                            int i=1;
                            temp=head;
                            while(i<k)
                            {
                                temp=temp->right;
                                i++;
                            }
                            if(temp==NULL)
                            {
                                cout<<"\n OUT OF RANGE \n";
                                break;
                            }
                            else
                            {
                                p->right=temp->right;
                                p->left=temp;
                                temp->right=p;
                                (p->right)->left=p;

                            }
                        }
                break;
                }
                case '4':
                {
                    cout<<"\n  Deleting .. \n";
                    p=head;
                    cout<<"\n enter position \n";
                    cin>>k;
                    if(k==1)
                    {  (head->right)->left=NULL;
                        temp=head;
                        head=head->right;
                        delete(temp);
                    }
                    else
                    {
                        int i=1;
                        temp=head;
                        while(i<k)
                        {   dummy=temp;
                            temp=temp->right;
                            i++;
                        }
                        dummy->right=temp->right;
                        (temp->right)->left=dummy;
                        delete(temp);
                    }
```

```
                        break;
                    }
                case '5':exit(1);
                    break;
                default :cout<<"\n enter valid input \n";
                    break;

        }
    }
}
```

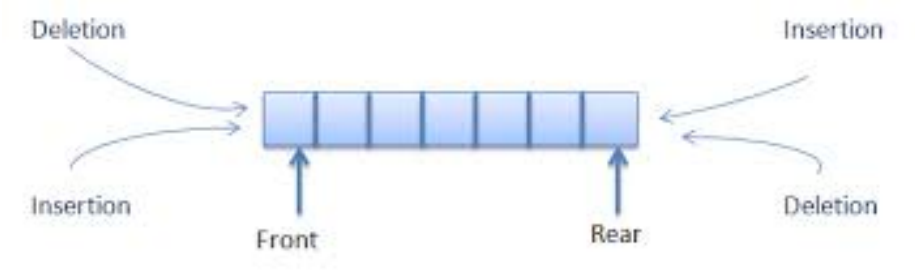
Output:

```
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
1

 Enter elements -9 to quit
1
2
3
4
5
-9

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 1.Forward 2.Backward
1

 forward displaying ..
1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 1.Forward 2.Backward
2

 Backward Displaying ..
5   4   3   2   1
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
3

  Inserting ..

 enter position
3

 enter element
99

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 1.Forward 2.Backward
```

```
1

 forward displaying ..
1   2   3   99  4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
4

  Deleting ..

 enter position
4

 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 1.Forward 2.Backward
1

 forward displaying ..
1   2   3   4   5
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
2

 1.Forward 2.Backward
2

 Backward Displaying ..
3   2   1
 1.Create 2.Display 3.Insert 4.Delete 5.Exit
5
Program ended with exit code: 1
```

Double Ended Queue (Deque)

here we can insert and delete from both the ends of the queue



```
#include <iostream>
using namespace std;
typedef struct doubleendedqueue
{
    int x;
    doubleendedqueue *left,*right;
}deq;
deq *p=NULL,*front=NULL,*rear=NULL,*temp=NULL;
void create1()
```

```
{   int ele;
    cout<<"\n enter elements -9 to quit \n";
    while(1)
    {
        cin>>ele;
        if(ele==-9)
        {
            return;
        }
        p=new deq;
        p->x=ele;
        p->left=NULL;
        p->right=NULL;
        if(front==NULL)
        {
            front=p;
            rear=p;
        }
        else
        {
            rear->right=p;
            p->left=rear;
            rear=p;
        }
    }
    return;
}
void display()
{
    char d;
    cout<<"\n 1.Forward Display 2.Backward Display \n";
    cin>>d;
    switch(d)
    {
            case '1':
            p=front;
            while(p!=NULL)
            {
                cout<<p->x<<"\t";
                p=p->right;
            }
            break;
            case '2':
            p=rear;
            while(p!=NULL)
            {
                cout<<p->x<<"\t";
                p=p->left;
            }
            break;
    }
    return;
}
void forwardIns()
```

```
{   int ele;
    cout<<"\n Forward Insertion .. \n ";
    cout<<"\n enter ele \n";
    cin>>ele;
    p=new deq;
    p->x=ele;
    p->right=p->left=NULL;
    if(front==NULL)
    {
        front=rear=p;
    }
    else
    {
        p->right=front;
        front->left=p;
        front=p;
    }
    return;
}

void backwardIns()
{
    int ele;
    cout<<"\n Backward Insertion .. \n ";
    cout<<"\n enter ele \n";
    cin>>ele;
    p=new deq;
    p->x=ele;
    p->right=p->left=NULL;
    if(front==NULL)
    {
        front=rear=p;
    }
    else
    {
        rear->right=p;
        p->left=rear;
        rear=p;
    }
    return;
}
void Insert()
{
    char d;
    cout<<"\n 1.Forward Insert 2.Backward Insert \n";
    cin>>d;
    switch (d)
    {
        case '1':forwardIns();
            break;
        case '2':backwardIns();
            break;
        default:cout<<"\n please enter valid input \n";
            break;
```

```
     }
    return;
}
void forwardDel()
{
    if(front==rear)
        cout<<"\n queue underflow !! \n";
    else
    {
        p=front;
        front=front->right;
        if(front!=NULL)
        {
            front->left=NULL;
        }
        else
        {
            rear=NULL;
        }
        delete(p);
    }
    return;
}
void backwardDel()
{
    if(front==rear)
        cout<<"\n queue underflow !! \n";
    else
    {
        p=rear;
        rear=rear->left;
        if(rear!=NULL)
        {
            rear->right=NULL;
        }
        else
        {
            front=NULL;
        }
        delete(p);
    }
    return;
}
void Delete1()
{
    char d;
    cout<<"\n 1.Forward Delete 2.Backward Delete \n";
    cin>>d;
    switch (d)
    {
        case '1':forwardDel();
            break;
        case '2':backwardDel();
            break;
        default:cout<<"\n please enter valid input \n";
            break;
    }
    return;


}
int main()
{

    char ch;
    while(1)
    {
        cout<<"\n 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit \n";
        cin>>ch;
        switch(ch)
        {
            case '1':create1();
                break;
            case '2':display();
                break;
            case '3':Insert();
                break;
            case '4':Delete1();
                break;
            case '5':exit(1);
                break;
            default:cout<<"\n please enter valid input \n";
                break;
        }

    }
}
```

output:

```
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
1

 enter elements -9 to quit
1
2
3
4
5
-9

 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
1
1   2   3   4   5
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2
```

```
 1.Forward Display 2.Backward Display
2
5   4   3   2   1
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
3

 1.Forward Insert 2.Backward Insert
1

 Forward Insertion ..

 enter ele
99

 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
3

 1.Forward Insert 2.Backward Insert
2

 Backward Insertion ..

 enter ele
100

 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
1
99  1   2   3   4   5   100
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
2
100 5   4   3   2   1   99
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
4

 1.Forward Delete 2.Backward Delete
1

 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
1
1   2   3   4   5   100
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
4

 1.Forward Delete 2.Backward Delete
```

```
2

 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
1
1   2   3   4   5
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
2

 1.Forward Display 2.Backward Display
2
5   4   3   2   1
 1.Create 2.Dispay 3.Insert 4.Delete 5.Exit
5
Program ended with exit code: 1
```

LinearSearch:

Linear search is usually very simple to implement, and is practical when the list has only a few elements, or when performing a single search in an unordered list.

1)Basic searching technique
2)Easy to Implement
Disadvantages
worst-case cost and the expected cost of linear search are both O(n).

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

int main()
{
    //linear search
    int a[10],n,i;
    cout<<"\n enter size of the array \n";
    cin>>n;
    cout<<"\n enter elements \n ";
    for(i=0;i<n;i++)
        cin>>a[i];

    cout<<"\n enter element to search \n";
    int k;
    cin>>k;
    int flag=0;
    for(i=0;i<n;i++)
    {
        if(a[i]==k)
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
        cout<<" sorry ! element not found ! ";
    else
        cout<<"Found the element "<<k<<" at position "<<i;
}
```

Binary Search :

1) Much Efficient than Linear Search and it follows Divide and Conquer Technique
2) Time Complexity is
```
    WorstCase ->O(logn)
    Avg Case  ->O(logn)
    Best Case ->O(1)
```
3) how to calculate it
```
        t(n)=T(n/2)+1 where n>1
            =1 where n=1
```

4)Its a must that elements should be in sorted order
Disadvantages
1)Must and should be in a sorted order

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

int main()
{
    //Binary Search
    int a[10];
    cout<<"\n enter size \n";
    int n;
    cin>>n;
    cout<<"\n enter elements \n";
    for(int i=0;i<n;i++)
    {
        cin>>a[i];
    }

    int mid,lb,ub,k,flag=0;
    lb=0;
    ub=n;
    cout<<"\n enter element to search \n";
    cin>>k;
    while(1)
    {
        mid=(lb+ub)/2;
        if(a[mid]==k)
        {
            cout<<"\n Element Found at "<<mid;
            break;
        }
        else if(a[mid]>k)
        {
            ub=mid;
        }
        else if(a[mid]<k)
        {
            lb=mid;
        }
        if(k>a[n])
        {
            cout<<"\n element not found \n";
            break;
        }
    }
}
```

Output:

```
 enter size
5

 enter elements
1
2
3
4
5

 enter element to search
2

 Element Found at 2Program ended with exit code: 0
```

BinarySearch Using Recursion

```
#include<iostream>
using namespace std;
int bs(int x[],int n,int lb,int ub,int k)
{
    int mid=(lb+ub)/2;
    if(x[mid]==k)
        return 1;
    else if(x[mid]<k)
        return bs(x,n,mid,ub,k);
    else if(x[mid]>k)
        return bs(x,n,lb,mid,k);
    else
        return 0;
}
int main()
{   int a[5];
    int n;
    cout<<"\n enter size of array \n";
    cin>>n;
    cout<<"\n enter array elements \n ";
    for(int i=0;i<n;i++)
        cin>>a[i];
    int k;
    cout<<"\n enter element to search \n";
    cin>>k;
    int d=bs(a,n,0,n,k);
    if(d==0)
        cout<<"\n sorry element not found \n";
    else
```

```
         cout<<"\n element found \n";
}
```
Output:

```
 enter size of array
5

 enter array elements
 1
2
3
4
5

 enter element to search
2

 element found
Program ended with exit code: 0
```

Sorting Techniques

Bubble Sort:

```
1)Also called as sinking sort
2)Time Complexity
    Worst Case :O(n^2)
    Avg Case   :O(n^2)
    Best case  :O(n^2)
3)how to calculate it
        t(n)=2T(n/2)+n where n>1
            =0 where n=1
```

Step-by-step example:

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort.
In each step, elements written in **bold** are being compared. Three passes will be required.

First Pass:

(**5** **1** 4 2 8) (**1** **5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 **5** **4** 2 8) (1 **4** **5** 2 8), Swap since 5 > 4

(1 4 **5** **2** 8) (1 4 **2** **5** 8), Swap since 5 > 2

(1 4 2 **5** **8**) (1 4 2 **5** **8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1** **4** 2 5 8) (**1** **4** 2 5 8)

(1 **4** **2** 5 8) (1 **2** **4** 5 8), Swap since 4 > 2

(1 2 **4** **5** 8) (1 2 **4** **5** 8)
(1 2 4 **5** **8**)

(1 2 4 **5** **8**)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:
(**1** **2** 4 5 8) (**1** **2** 4 5 8)
(1 **2** **4** 5 8) (1 **2** **4** 5 8)
(1 2 **4** **5** 8) (1 2 **4** **5** 8)
(1 2 4 **5** **8**) (1 2 4 **5** **8**)

pgm)
```
#include<iostream>
using namespace std;
int main()
{
    int a[10],n;
    cout<<"\n enter size of the array \n";
    cin>>n;
    cout<<"\n enter elements into the array \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    int temp;
    for(int i=0;i<n;i++)
        for(int j=0;j<n-i-1;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    cout<<"\n after sorting .. \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<"\t";
}
```

output

```
 enter size of the array
5

 enter elements into the array
99
```

```
14
17
11
50

 after sorting ..
11  14  17  50  99  Program ended with exit code: 0
```

Selection Sort

1)Also called as CockTail Sort
2)Time Complexity
    Worst Case :O(n^2)
    Avg Case   :O(n^2)
    Best case :O(n^2)
3)how to calculate it
        t(n)=2T(n/2)+n where n>1
            =0 where n=1

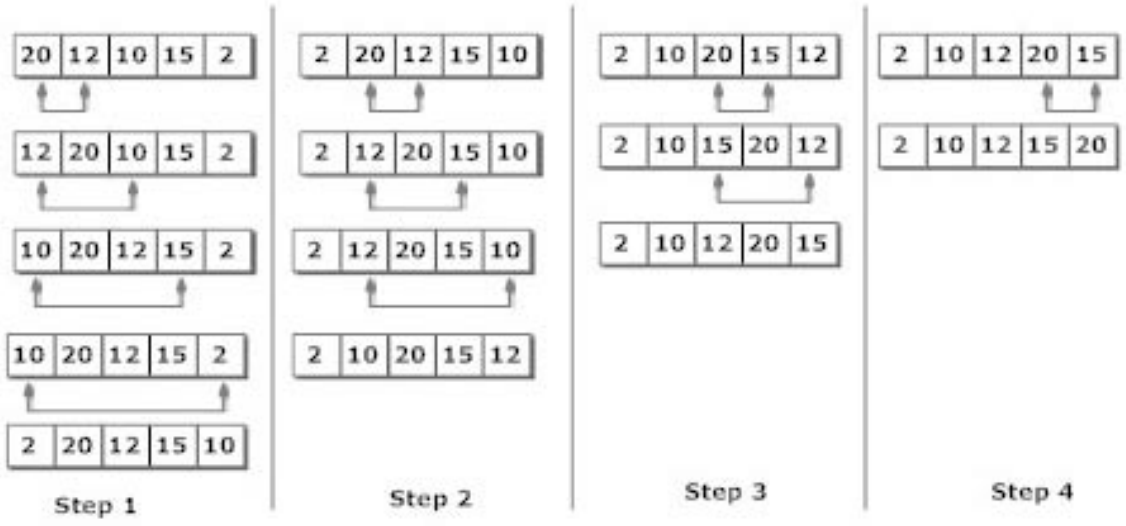


Figure: Selection Sort

```
#include<iostream>
using namespace std;
int main()
{
    int a[10],n;
    cout<<"\n enter size of the array \n";
    cin>>n;
    cout<<"\n enter elements into the array \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    int temp;
    for(int i=0;i<n;i++)
        for(int j=i+1;j<n;j++)
        {
            if(a[i]>a[j])
```

```
                {
                    temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
            }
        cout<<"\n after sorting .. \n";
        for(int i=0;i<n;i++)
            cout<<a[i]<<"\t";
}
```

output

```
 enter size of the array
5

 enter elements into the array
3
88
1
5
99

 after sorting ..
1   3   5   88  99  Program ended with exit code: 0
```

INSERTION SORT

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

int main()
{
    int a[10];
    int n;

    cout<<"\n ENTER SIZE OF THE ARRAY \n";
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>a[i];
    int d;
    for(int i=1;i<n;i++)
    {
        d=i;
        while(d>0 && a[d]<a[d-1])
        {
            int temp=a[d-1];
            a[d-1]=a[d];
            a[d]=temp;
            d--;

        }
    }
    for(int i=0;i<n;i++)
        cout<<a[i]<<"\t";
}
```

```
 ENTER SIZE OF THE ARRAY
5
1
3
7
2
10
1    2    3    7    10   Program ended with exit code: 0
```

<u>QuickSort:</u>
1)Fastes Sorting Tech.,
2)Time Complexity :

```
    Worst Case :O(n^2)
    Avg Case   :O(nlogn)
    Best case  :O(nlogn)
```

3)Uses Divide Nd Conquer
4)how to calculate it

```
        t(n)=2T(n/2)+n where n>1
            =0 where n=1
```

EG:

```
#include<iostream>
using namespace std;
void quicksort(int a[],int n,int lb,int ub)
{
    if(lb<ub)
    {
        
        int key=a[lb];
        int i=lb;
        int j=ub;
        int flag=0;
        while(flag==0)
        {
            
            i++;
            while(a[i]<key)
            {
                i++;
            }
            
            while(a[j]>key)
            {
                j--;
            }
            if(i<j)
            {
                int temp;
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
            else
                flag=1;
        }
        int temp=a[lb];
        a[lb]=a[j];
        a[j]=temp;
        quicksort(a, n, lb, j-1);
        quicksort(a,n,j+1,ub);

    }

}
int main()
{
    int a[10],n;
    cout<<"\n enter size of the array \n";
    cin>>n;
    cout<<"\n enter elements into the array \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    quicksort(a, n, 0, n-1);
    cout<<"displaying after getting sorted \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<"\t";

}
```

Output:

```

 enter size of the array
5

 enter elements into the array
100
74
21
15
11
displaying after getting sorted
11   15   21   74   100  Program ended with exit code: 0
```

MergeSort

1)Uses Divide Nd Conquer

2)Time Complexity:

```
    Worst Case :O(nlogn)
    Avg Case   :O(nlogn)
    Best case  :O(nlogn)
```

3)how to calculate it

```
        t(n)=2T(n/2)+Cn where n>1
            =a where n=1
```

pgm)

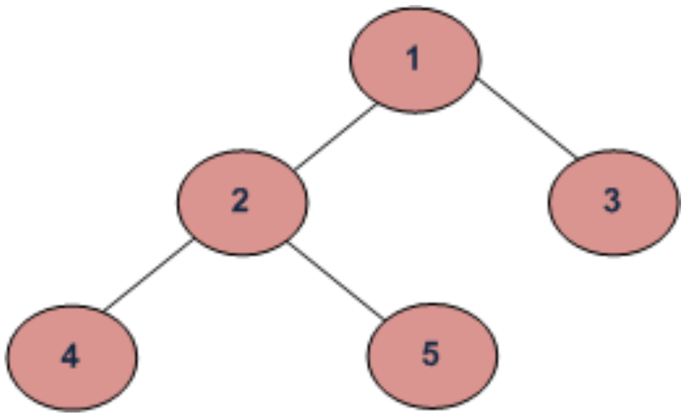
```
#include<iostream>
using namespace std;
int a[10],b[10];
void mergeIt(int ,int,int);
void merge(int low,int high)
{   int mid;

    if(low<high)
    {
        mid=low+high;
        mid=mid/2;
        merge(low,mid);
        merge(mid+1,high);
        mergeIt(low,mid,high);
    }
}

void mergeIt(int low,int mid,int high)
{   static int count=1;
    int h=low;
    int i=low;
    int j=mid+1;
    while(h<=mid && j<=high)
    {
        if(a[h]<=a[j])
        {
            b[i]=a[h];
            h++;
        }
        else
        {
            b[i]=a[j];
            j++;
        }
        i++;
    }//end of while

    if(h>mid)
    {
        for(int k=j;k<=high;k++)
        {
            b[i]=a[k];
            i++;
        }
    }
    else
    {
        for(int k=h;k<=mid;k++)
        {
            b[i]=a[k];
            i++;
        }
    }
    for(int k=low;k<=high;k++)
    {
        a[k]=b[k];
    }
    cout<<"step "<<count<<":\t";
    for(int k=low;k<=high;k++)
    {
        cout<<a[k]<<"\t";
    }
    count++;
    cout<<"\n";
}


int main()
{
    int n;
    cout<<"\n enter size of the array \n";
    cin>>n;
    cout<<"\n enter elements into the array \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    merge(0,n-1);

    for(int k=0;k<n;k++)
    {
        cout<<a[k]<<"\t";
    }

}
```

Output:

```
 enter size of the array
5

 enter elements into the array
6
3
9
2
17
step 1: 3   6
step 2: 3   6   9
step 3: 2   17
step 4: 2   3   6   9   17
2   3   6   9   17  Program ended with exit code: 0
```

Tree Traversals

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



1) Preorder 2)InOrder 3)PostOrder

Time Complexity: O(n)

Let us prove it:

Complexity function T(n) — for all problem where tree traversal is involved — can be defined as:

$T(n) = T(k) + T(n – k – 1) + c$

Where k is the number of nodes on one side of root and n-k-1 on the other side.

Let's do analysis of boundary conditions

Case 1: Skewed tree (One of the subtrees is empty and other subtree is non-empty)

k is 0 in this case.
$T(n) = T(0) + T(n-1) + c$
$T(n) = 2T(0) + T(n-2) + 2c$
$T(n) = 3T(0) + T(n-3) + 3c$
$T(n) = 4T(0) + T(n-4) + 4c$

…………………………………………
…………………………………………
$T(n) = (n-1)T(0) + T(1) + (n-1)c$

T(n) = nT(0) + (n)c

Insertion is quite simple , Once we find the element to be less or greater we will place accordingly
 if element is less than its root ,it gets placed towards left side
 if element is greater than its root,it gets placed towards right side

```
Program )
#include<iostream>
using namespace std;
typedef struct trees
{
    struct trees *left;
    int data;
    struct trees * right;
}trees;
void insert(trees **,int);
void preorder(trees *);
void inorder(trees *);
void postorder(trees *);
int main()
{
    int n;
    trees *bt=NULL;
    char ch;
    while(1)
    {
        cout<<"\n enter your choice 1.Create 2.Display\n";
        cin>>ch;
        switch(ch)
        {
            case '1':
            {
                while(1)
                {
                    cout<<"\n enter element,-9 to quit \n";
                    cin>>n;
                    if(n==-9)
                        break;
                    insert(&bt,n);//passing the address of a pointer
so we need to
                                          //receive with pointer to a
pointer


                }
            }
                break;
            case '2':
            {
                cout<<"\n enter your choice of display 1.PreOrder
2.InOrder 3.PostOrder \n";
                cin>>ch;
                switch(ch)
                {
                    case '1':preorder(bt);//passing a pointer ,so
call by reference
                        break;
                    case '2':inorder(bt);
                        break;
                    case '3':postorder(bt);
                        break;
                }
            }
                break;
        }
    }
}
void insert(trees **p,int n)
{   if(*p==NULL)
    {
        *p=new trees;
        (*p)->data=n;
        (*p)->left=(*p)->right=NULL;
    }
    else
    {
        if(n<(*p)->data)
        {
            insert(&(*p)->left,n);
        }
        else
        {
            insert(&(*p)->right,n);
        }

    }

}

void preorder(trees *p)
{
    if(p!=NULL)
    {
        cout<<"\t"<<p->data;
        preorder(p->left);
        preorder(p->right);
    }
}

void inorder(trees *p)
{
    if(p!=NULL)
    {

        preorder(p->left);
        cout<<"\t"<<p->data;
        preorder(p->right);
    }
}
void postorder(trees *p)
{
    if(p!=NULL)
    {

        preorder(p->left);
        preorder(p->right);
        cout<<"\t"<<p->data;

    }
}
```

```
 enter your choice 1.Create 2.Display 3.Insert 4.Exit
1

 enter element,-9 to quit
1

 enter element,-9 to quit
5

 enter element,-9 to quit
2

 enter element,-9 to quit
9

 enter element,-9 to quit
45

 enter element,-9 to quit
78

 enter element,-9 to quit
-9

 enter your choice 1.Create 2.Display 3.Insert 4.Exit
2

 enter your choice of display 1.PreOrder 2.InOrder 3.PostOrder
1
    1   5   2   9   45  78
 enter your choice 1.Create 2.Display 3.Insert 4.Exit
3

 enter element to insert
99

 enter your choice 1.Create 2.Display 3.Insert 4.Exit
2
```

```
 enter your choice of display 1.PreOrder 2.InOrder 3.PostOrder
1
    1   5   2   9   45  78  99
 enter your choice 1.Create 2.Display 3.Insert 4.Exit
4
Program ended with exit code: 1
```

Deleting a node from a tree

1)Deleting is not as simple as insertion because we need to re-arrange the tree again
2)this can be classified into 2 cases
a) A node having 2 sons (including root node)
b) A node having 0 or 1 sons

Node having 0 or 1 Son: total six sub cases

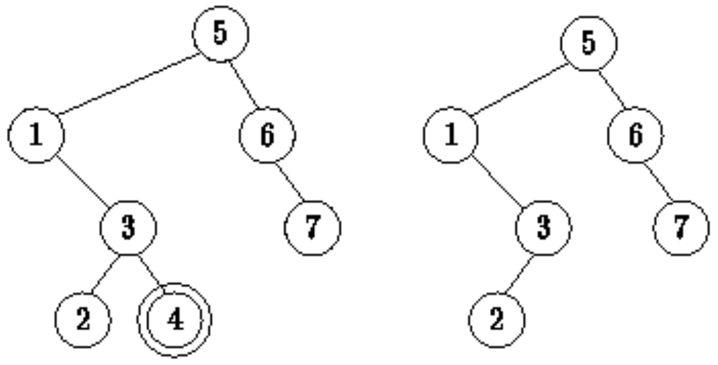
if we are deleting the root node **we need to check again 3 things**
->0 sons
->Only left son
->only right son

if we are deleting other than Root node **we need to check again 3 things**
->0 sons
->Only left son
->only right son

To make things simpler i have only given Create,Display and Insert in above program ,lets Continue with it now

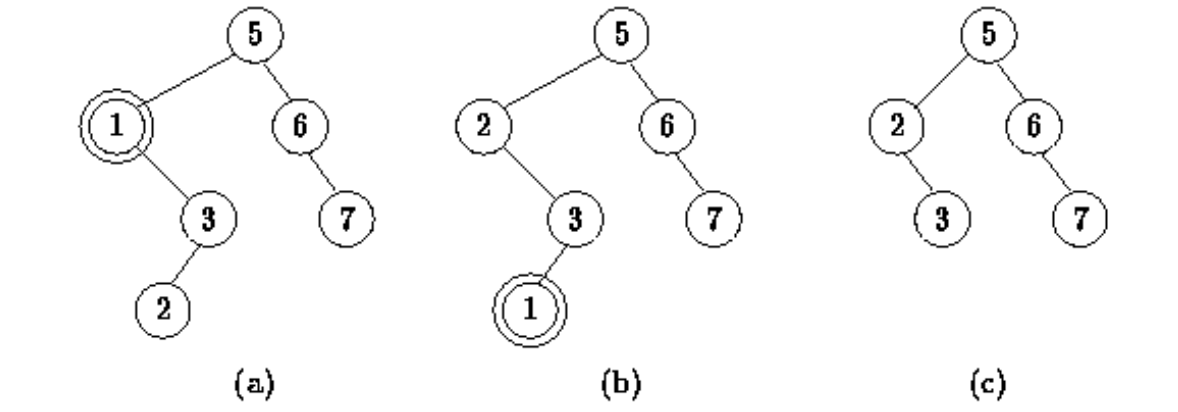
When removing an item from a search tree, it is imperative that the tree which remains satisfies the data ordering criterion. If the item to be removed is in a leaf node, then it is fairly easy to remove that item from the tree since doing so does not disturb the relative order of any of the other items in the tree.

Examples:1



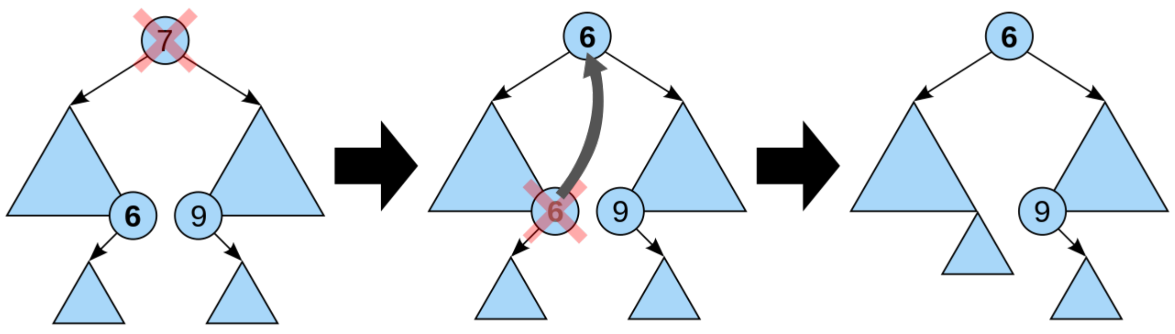
Removing a leaf Node from a BST

For example, consider the binary search tree shown in Figure (a). Suppose we wish to remove the node labeled 4. Since node 4 is a leaf, its subtrees are empty. When we remove it from the tree, the tree remains a valid search tree as shown in Figure (b).



Removing a NON-LEAF NODE (EITHER LEFT OR RIGHT IS EMPTY)

To remove a non-leaf node, we move it down in the tree until it becomes a leaf node since a leaf node is easily deleted. To move a node down we swap it with another node which is further down in the tree. For example, consider the search tree shown in Figure (a). Node 1 is not a leaf since it has an empty left subtree but a non-empty right subtree. To remove node 1, we swap it with the smallest key in its right subtree, which in this case is node 2, Figure (b). Since node 1 is now a leaf, it is easily deleted. Notice that the resulting tree remains a valid search tree, as shown in Figure (c).



Removing a NON-LEAF NODE (With 2 Children)

In case of 2 Children we just need to check the left-sub-trees right node and Right-sub-trees left node ,The Smaller one Becomes the root node and such of the smaller one will be adjusted accordingly to the tree

```
//Search Function
void search(trees *q,int n,trees **pp,trees **tp,int *fp)
{   //q and n holds search information
    //pp ,tp and fp holds the retriving information for deleting
   while(q)//returns true if q is not null
   {
       if(n==q->data)
       {
           *fp=1;//to indicate that element has been found
           *tp=q;
           return;
       }
       else if(n<q->data)
       {
           *pp=q;//to store the parent value
           q=q->left;
       }
       else
       {
           *pp=q;//to store the parent value
           q=q->right;
       }
   }
}

void delete1(trees **root,int n)
{
    int found;
    trees *parent,*x,*xsucc;
    if(*root==NULL)
    {
        cout<<"\n Tree is empty \n";
        return;
    }
    parent=x=NULL;
    search(*root,n,&parent,&x,&found);

    if(found==false)
    {
        cout<<"\n element not found to delete \n";
        return;
    }


    //IF THE ROOT HAS 2 CHILDREN
    if(x->left!=NULL && x->right!=NULL)
    {
        parent=x;
        xsucc=x->right;
        if(xsucc->left!=NULL)
        {
            parent=xsucc;
            xsucc=xsucc->left;
        }
        x->data=xsucc->data;
        x=xsucc;
    }

    //IF THE NODE HAS NO CHILDREN

    if(x->left == NULL && x->right == NULL)
    {
        if(parent->right==x)
            parent->right=NULL;
        else
            parent->left=NULL;
        free(x);
        return;
    }

    //IF IT HAS ONLY RIGHT CHILD

    if(x->left == NULL && x->right !=NULL)
    {
        if(parent->left==x)
            parent->left=x->right;
        else
            parent->right=x->right;
        free(x);
        return;
    }

    //IF IT HAS ONLY LEFT CHILD

    if(x->left !=NULL && x->right ==NULL)
    {
        if(parent->left==x)
            parent->left=x->left;
        else
            parent->right=x->left;
        free(x);
        return;
    }
}
```

As we have seen the individual code here,lets club up the entire menu driven

program and write the code for Complete Binary Search Tree

```
#include<iostream>
using namespace std;
typedef struct trees
{
    struct trees *left;
    int data;
    struct trees * right;
}trees;
void delete1(trees **,int);
void search(trees *,int ,trees **,trees **,int *);
void insert(trees **,int);
void preorder(trees *);
void inorder(trees *);
void postorder(trees *);
int main()
{
    int n;
    trees *bt=NULL;
    char ch;
    while(1)
    {
        cout<<"\n enter your choice 1.Create 2.Display 3.Insert 
4.Delete 5.exit\n";
        cin>>ch;
        switch(ch)
        {
            case '1':
            {
                while(1)
                {
                    cout<<"\n enter element,-9 to quit \n";
                    cin>>n;
                    if(n==-9)
                        break;
                    insert(&bt,n);//passing the address of a pointer 
so we need to
                                  //receive with pointer to a 
pointer


                }
            }
                break;
            case '2':
            {
                cout<<"\n enter your choice of display 1.PreOrder 
2.InOrder 3.PostOrder \n";
                cin>>ch;
                switch(ch)
                {
                    case '1':preorder(bt);//passing a pointer ,so
call by reference
                        break;
                    case '2':inorder(bt);
                        break;
                    case '3':postorder(bt);
                        break;
                }
            }
                break;
            case '3':
            {
                cout<<" \n enter element to insert \n";
                cin>>n;
                insert(&bt,n);
            }
                break;
            case '4':
            {
                cout<<"\n enter element to delete \n";
                cin>>n;
                delete1(&bt,n);
                break;
            }
            case '5':exit(1);
                break;
            default:cout<<"\n Please enter valid Input \n";
                break;
        }
    }
}
//Creation is Repeated Insertion
void insert(trees **p,int n)
{   if(*p==NULL)
    {
        *p=new trees;
        (*p)->data=n;
        (*p)->left=(*p)->right=NULL;
    }
    else
    {
        if(n<(*p)->data)
        {
            insert(&(*p)->left,n);
        }
        else
        {
            insert(&(*p)->right,n);
        }

    }

}

void preorder(trees *p)
```

```
{
    if(p!=NULL)
    {
        cout<<"\t"<<p->data;
        preorder(p->left);
        preorder(p->right);
    }
}

void inorder(trees *p)
{
    if(p!=NULL)
    {

        preorder(p->left);
        cout<<"\t"<<p->data;
        preorder(p->right);
    }
}
void postorder(trees *p)
{
    if(p!=NULL)
    {

        preorder(p->left);
        preorder(p->right);
        cout<<"\t"<<p->data;

    }
}

//Search Function
void search(trees *q,int n,trees **pp,trees **tp,int *fp)
{   //q and n holds search information
    //pp ,tp and fp holds the retriving information for deleting
   while(q)//returns true if q is not null
   {
       if(n==q->data)
       {
           *fp=1;//to indicate that element has been found
           *tp=q;
           return;
       }
       else if(n<q->data)
       {
           *pp=q;//to store the parent value
           q=q->left;
       }
       else
       {
           *pp=q;//to store the parent value
           q=q->right;
       }
   }
```

```
}

void delete1(trees **root,int n)
{
    int found;
    trees *parent,*x,*xsucc;
    if(*root==NULL)
    {
        cout<<"\n Tree is empty \n";
        return;
    }
    parent=x=NULL;
    search(*root,n,&parent,&x,&found);

    if(found==false)
    {
        cout<<"\n element not found to delete \n";
        return;
    }


    //IF THE ROOT HAS 2 CHILDREN

    if(x->left!=NULL && x->right!=NULL)
    {
        parent=x;
        xsucc=x->right;
        if(xsucc->left!=NULL)
        {
            parent=xsucc;
            xsucc=xsucc->left;
        }
        x->data=xsucc->data;
        x=xsucc;
    }

    //IF THE NODE HAS NO CHILDREN

    if(x->left == NULL && x->right == NULL)
    {
        if(parent->right==x)
            parent->right=NULL;
        else
            parent->left=NULL;
        free(x);
        return;
    }

    //IF IT HAS ONLY RIGHT CHILD

    if(x->left == NULL && x->right !=NULL)
    {
        if(parent->left==x)
            parent->left=x->right;
        else
            parent->right=x->right;
        free(x);
        return;
    }

    //IF IT HAS ONLY LEFT CHILD

    if(x->left !=NULL && x->right ==NULL)
    {
        if(parent->left==x)
            parent->left=x->left;
        else
            parent->right=x->left;
        free(x);
        return;
    }
}
```

OUTPUT:

```
 enter your choice 1.Create 2.Display 3.Insert 4.Delete 5.exit
1

 enter element,-9 to quit
4

 enter element,-9 to quit
9

 enter element,-9 to quit
2

 enter element,-9 to quit
6

 enter element,-9 to quit
3

 enter element,-9 to quit
-9

 enter your choice 1.Create 2.Display 3.Insert 4.Delete 5.exit
2

 enter your choice of display 1.PreOrder 2.InOrder 3.PostOrder
1
    4   2   3   9   6
 enter your choice 1.Create 2.Display 3.Insert 4.Delete 5.exit
4

 enter element to delete
3
```

```
 enter your choice 1.Create 2.Display 3.Insert 4.Delete 5.exit
2

 enter your choice of display 1.PreOrder 2.InOrder 3.PostOrder
1
    4   2   9   6
 enter your choice 1.Create 2.Display 3.Insert 4.Delete 5.exit
5
Program ended with exit code: 1
```

AVL TREE(SELF BALANCING TREE)

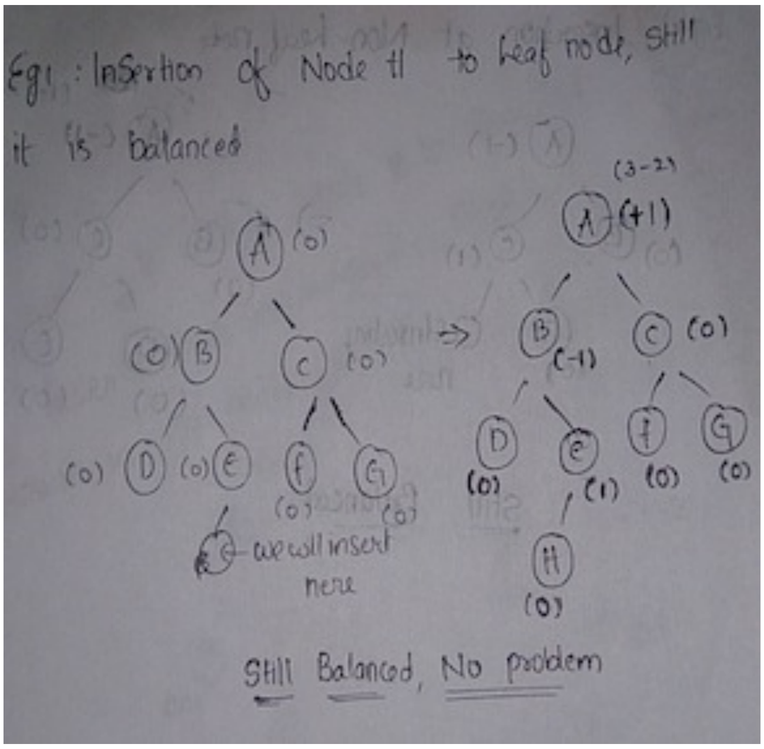
BST is termed as avl tree when the balancing factor of any node is 0,1,-1

balancing factor=height of LST-Height of RST

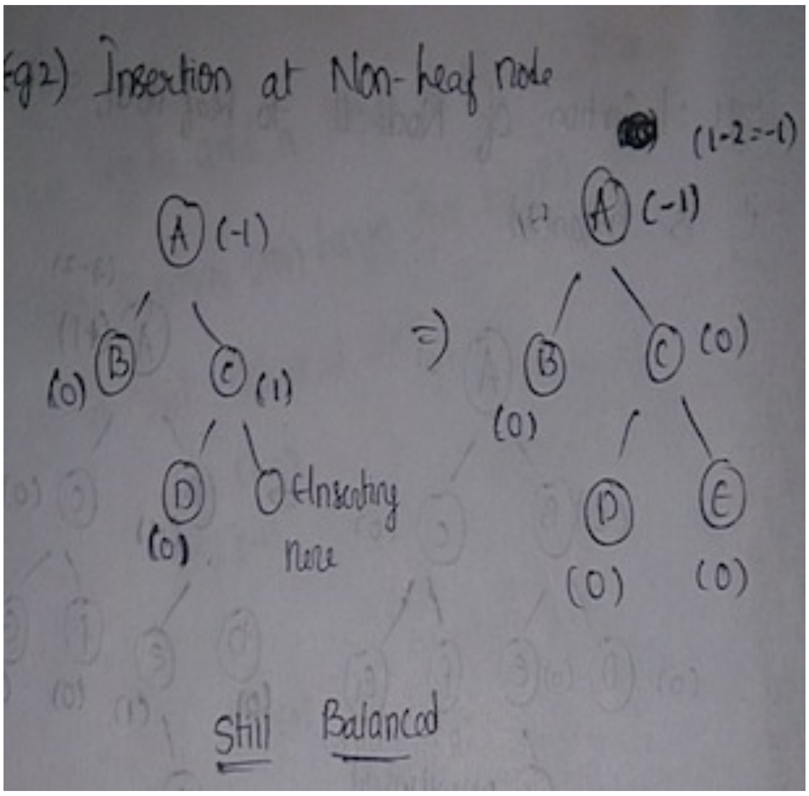
```
Ex:    (-1)                              A(-2) so not possible
here,
        A                              /   \
       / \                           b(0)   c +1
      B   c(-1)                              / \
    (0)    \                             d(1)  e (0)
            d (0)                        /
                                       f (0)

 a)A BALANCED TREE                b)Unbalanced Tree
```

A balanced Tree becomes unbalanced during insertion and deletion of nodes

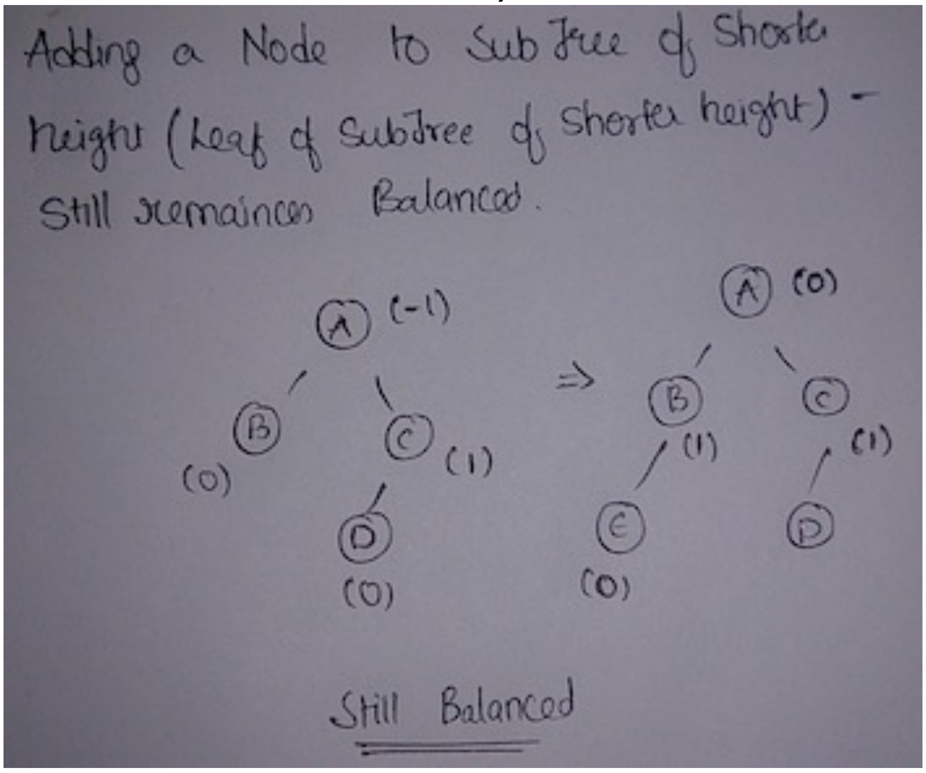
CASE 1) INSERTION OF NODE TO LEAF NODE (YET REMAINS BALANCED) –>NO PROBLEM



CASE 2) INSERTION AT NON LEAF NODE (YET REMAINS BALANCED) –>NO PROBLEM

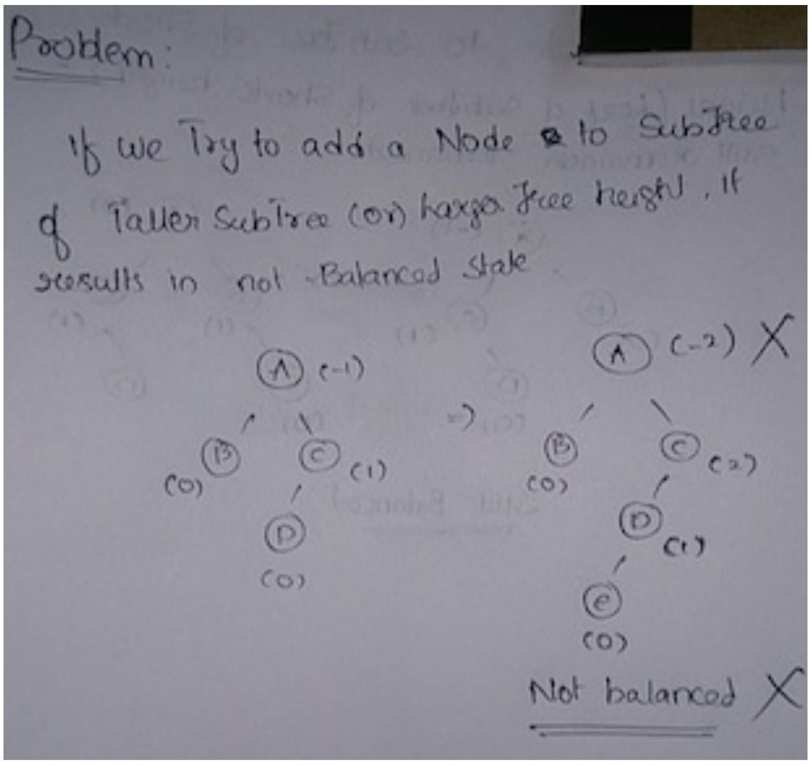


CASE 3)ADDING A NODE TO SUBTREE OF SHORTER HEIGHT (LEAF OF SUBTREE HAVING LESS HEIGHT) —>NO PROBLEM



Adding a Node to Sub Tree of Shorter height (Leaf of Subtree of Shorter height) - Still remainces Balanced.
A (-1)
B (0)
C (1)
D (0)
A (0)
B (1)
C (1)
E (0)
D
Still Balanced

CASE 4) INSERTION OF A NODE TO A SUBTREE WHERE THE SUBTREE IS TALLER —>PROBLEM ,WE NEED TO BALANCE IT



Problem:

If we Try to add a Node to SubTree of Taller SubTree (or) Larger Tree height, it results in not Balanced State

<u>we have seen how problem occurs,to solve it we need to perform Rotation</u>

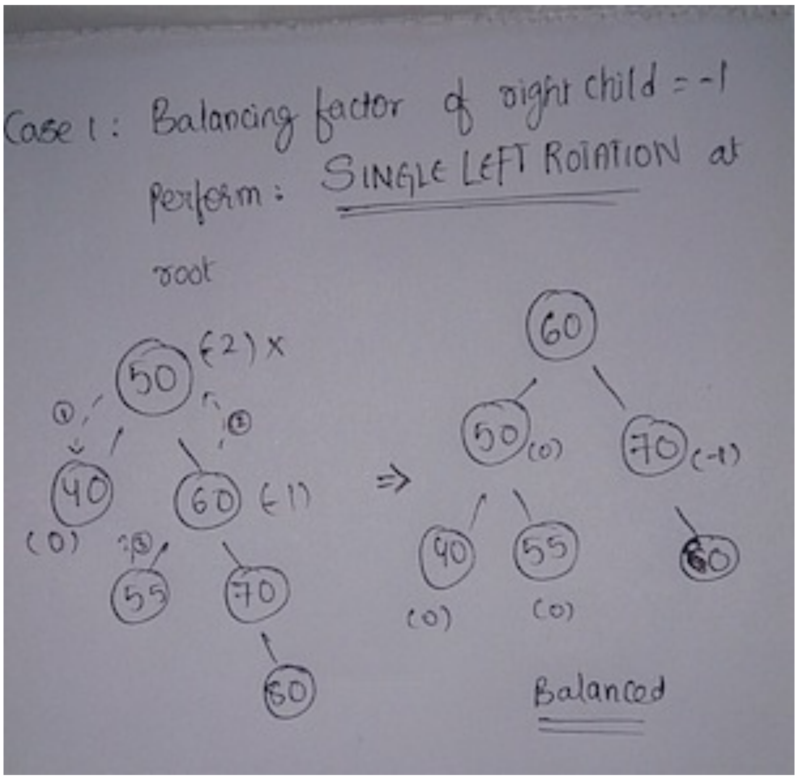
These are classified into 2 types

1)If the balancing factor =-2 (we need to consider the balancing factor of its right child)

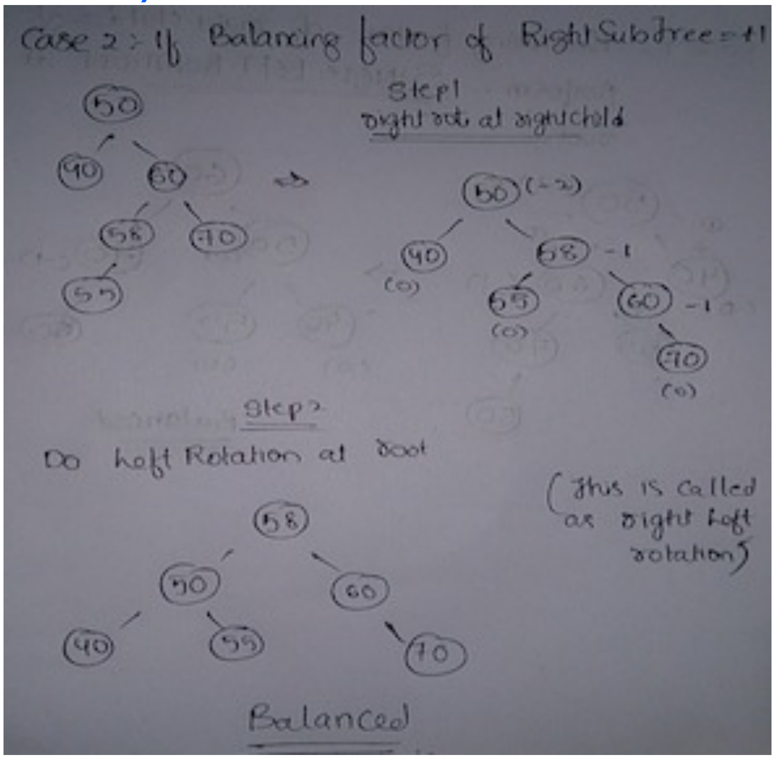
case 1: if balancing factor of right child is -1 ,perform SINGLE_LEFT_ROTATION at the root

case 2: if balancing factor of right child is +1 ,perform Right_Rotation at right child + left Rotation at root (RIGHT_LEFT_ROT)

CASE 1)


Case 1: Balancing factor of right child = -1
perform: SINGLE LEFT ROTATION at root

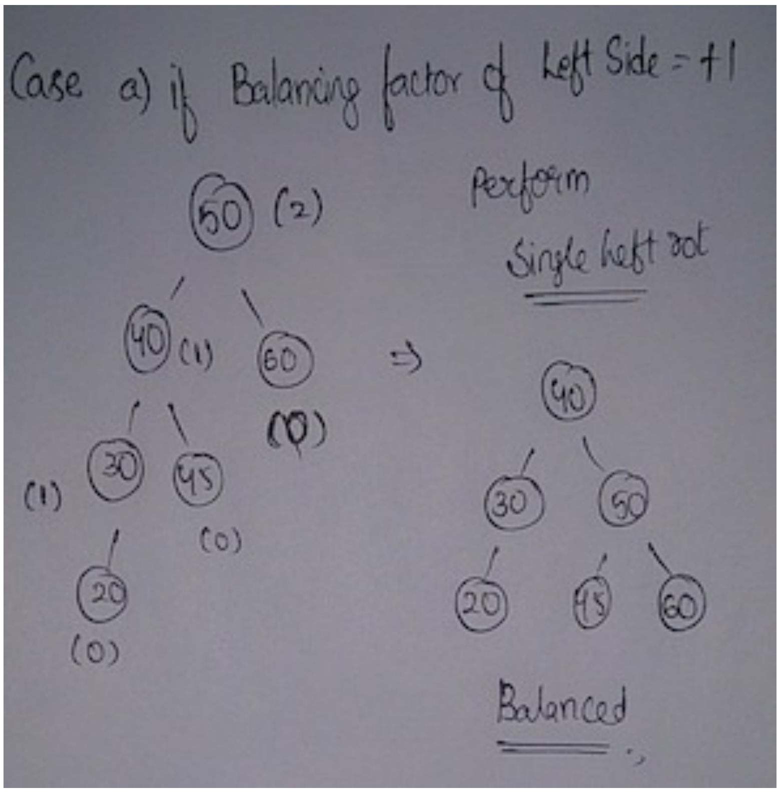
CASE 2)


Case 2: If Balancing factor of Right Subtree = +1
Step1
right rot. at rightchild
Step2
Do Left Rotation at root
(This is called as right left rotation)

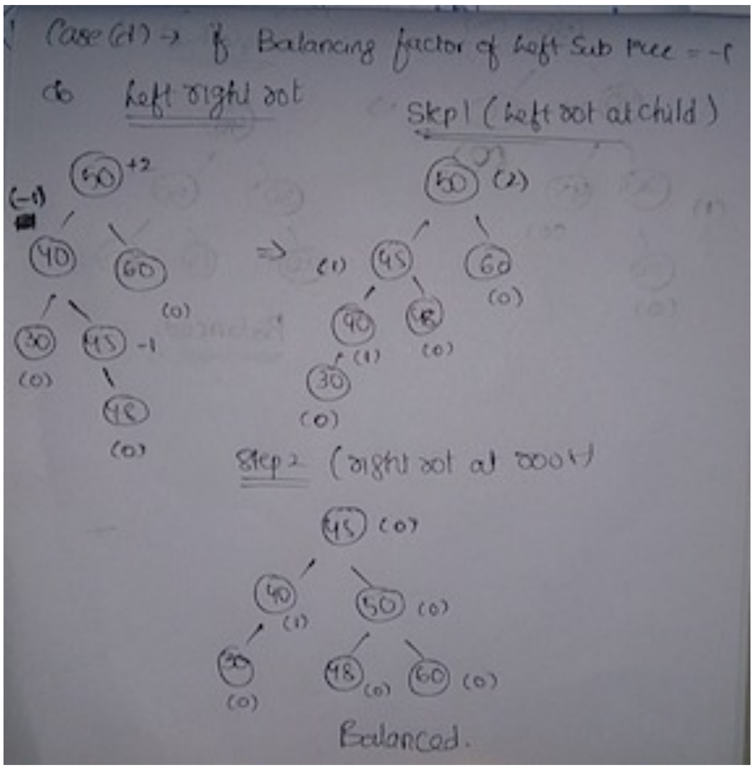
1)If the balancing factor =2 (we need to consider the balancing factor of its left child)
case a: if balancing factor of left child is 1 ,perform SINGLE_RIGHT_ROTATION at the root
case b: if balancing factor of left child is -1 ,perform Left_Rotation at Left child + Right Rotation at

root (Left_Right_ROT)

CASE a:



CASE b:


Case (ii) -> if Balancing factor of left Sub tree = -1
do left right rot
Step 1 (left rot at child)
Step 2 (right rot at root)
Balanced.

PROGRAM:

```
#include<iostream>
using namespace std;
typedef struct avltree
```

```
{
    avltree * left;
    int data;
    avltree *right;
}avltree;

void inorder(avltree *);
void preorder(avltree *);
void postorder(avltree *);

avltree* insert(avltree *,int); //return type is a pointer

avltree* balancing(avltree *);

int finalheight(avltree *);
int get_diff(avltree *);

avltree* singleleftrot(avltree *);
avltree* rightleftrot(avltree *);
avltree* singlerightrot(avltree *);
avltree* leftrightrot(avltree *);

int main()
{   char ch,d;
    int n;
    avltree *tree=NULL;
    while(1)
    {
        cout<<"\n enter choice 1.Create 2.Display 3.Insert 4.Exit \n
";
        cin>>ch;
        switch(ch)
        {
                case '1':
                {
                    cout<<"\n enter element ,-9 to exit \n";
                    while(1)
                    {
                        cin>>n;
                        if(n==-9)
                            break;
                        tree=insert(tree,n);
                    }
                    break;
                }
                case '2':
                {
                    cout<<"\n 1.Preorder 2.Inorder 3.PostOrder \n";
                    cin>>d;
                    switch(d)
                    {
                        case '1':preorder(tree);
                            break;
                        case '2':inorder(tree);
```

```
                            break;
                        case '3':postorder(tree);
                            break;
                        default:cout<<"\n please enter valid choice
\n";
                            break;
                    }
                    break;
                }
                case '3':
                {
                    cout<<"\n enter the element to insert \n";
                    cin>>n;
                    tree=insert(tree,n);
                    break;
                }
            case '4':exit(1);
                break;
            default:cout<<"\n please enter the valid input \n";
                break;
        }

    }
}

void inorder(avltree *tree)
{
   if(tree!=NULL)
    {
        inorder(tree->left);
        cout<<"\t"<<tree->data;
        inorder(tree->right);
    }
}

void preorder(avltree *tree)
{

    if(tree!=NULL)
    {
        cout<<"\t"<<tree->data;
        preorder(tree->left);
        preorder(tree->right);
    }
}

void postorder(avltree *tree)
{
    if(tree!=NULL)
    {
        postorder(tree->left);
        postorder(tree->right);
        cout<<"\t"<<tree->data;
    }
```

```
}

//INSERTING A NODE

avltree* insert(avltree *root,int vol)
{
    if(root==NULL)
    {
        root=new avltree;
        root->data=vol;
        root->left=root->right=NULL;
    }
    else if(vol<root->data)
    {
        root->left=insert(root->left,vol);
        root=balancing(root);
    }
    else
    {
        root->right=insert(root->right,vol);
        root=balancing(root);
    }
    return root;
}

//SINCE WE ARE ADDING ELEMENTS INTO THE TREE ,TREE GETS
UNBALANCED,WE NEED TO BALANCE IT

avltree *balancing(avltree *node)
{
    int balfactor=get_diff(node);
    if(balfactor==-2)
    {
        if(get_diff(node->right)==-1)
            node=singleleftrot(node);
        else
            node=rightleftrot(node);
    }
    else if(balfactor==+2)
    {
        if(get_diff(node->left)==+1)
            node=singlerightrot(node);
        else
            node=leftrightrot(node);
    }
return node;
}

//LET US NOW DEFINE THE RESPECTIVE FUNCTIONS FOR ROTATION and FOR
get_diff()

avltree* singleleftrot(avltree *parent)
{
    avltree *node1;
    node1=parent->right;
    parent->right=node1->left;
    node1->left=parent;
    return node1;
}


avltree* singlerightrot(avltree *parent)
{
    avltree *node1;
    node1=parent->left;
    parent->left=node1->right;
    node1->right=parent;
    return node1;
}

avltree* rightleftrot(avltree *parent)
{
    avltree *node1;
    node1=parent->right;
    parent->right=singlerightrot(node1);
    return singleleftrot(parent);
}


avltree* leftrightrot(avltree *parent)
{
    avltree *node1;
    node1=parent->left;
    parent->left=singleleftrot(node1);
    return singlerightrot(parent);
}

int get_diff(avltree *node)
{
    int lh=finalheight(node->left);
    int rh=finalheight(node->right);
    int balfactor=lh-rh;
    return balfactor;
}

//DEFINING THE FUNCTION TO CAL THE HEIGHT OF THE TREE

int finalheight(avltree *node)
{
    int height=0;
    if(node!=NULL)
    {
        int lh=finalheight(node->left);
        int rh=finalheight(node->right);
        int max=lh>rh ? lh:rh; //TERNARY OPERATOR
        height=max+1;//also include the root node
    }
    return height;
```

```
}
```

OUTPUT:

```
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 1

 enter element ,-9 to exit
4
9
3
7
8
2
-9

 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2

 1.Preorder 2.Inorder 3.PostOrder
1
    4   3   2   8   7   9
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2

 1.Preorder 2.Inorder 3.PostOrder
2
    2   3   4   7   8   9
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2

 1.Preorder 2.Inorder 3.PostOrder
3
    2   3   7   9   8   4
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 3

 enter the element to insert
99

 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2

 1.Preorder 2.Inorder 3.PostOrder
1
    4   3   2   8   7   9   99
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2

 1.Preorder 2.Inorder 3.PostOrder
2
    2   3   4   7   8   9   99
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 2
 1.Preorder 2.Inorder 3.PostOrder
3
    2   3   7   99  9   8   4
 enter choice 1.Create 2.Display 3.Insert 4.Exit
 4
Program ended with exit code: 1
```

ACKNOWLEDGMENTS

I sincerely thank my parents to encourage me to write the book and to the lecturer's and professors of GITAM UNIVERSITY who have gave the basics of programming .

I specially thank MR.Raju (dataPro) sir to teach me advance topics of programming .

This book was specially written to one special friend of mine which later is sold to other people in market

Hope you like this and let me continue my other Technical books related to other topics in Future

Thanking you

Yours FaithFully

Dinesh Singh