C++ & DATA STRUCTURE LAB (PGDCAL02) (PG - DIPLOMA)

ACHARYA NAGARJUNA UNIVERSITY

CENTRE FOR DISTANCE EDUCATION

NAGARJUNA NAGAR,

GUNTUR

ANDHRA PRADESH

DATA STRUCTURES USING C++ Lab Manual

1.write a program for implementing the operations on complex numbers using classes

Procedure:

1.Here we read two complex numbers .

2.Perform addition and subtraction on those two complex numbers

Program:

```
#include <iostream.h>
class Complex
{
    private:
        int real;
        int imag;
    public:
        void readComplex()
        {
            cout << "Enter the real and imaginary parts : ";
            cin >> this->real;
            cin >> this->imag;
        }
        Complex add(const Complex& c)
        {
            Complex comp;
            comp.real = this->real + c.real;
            comp.imag = this->imag + c.imag;
            return comp;
        }
```

```
        Complex subtract(const Complex& c)
        {
            Complex comp;
            comp.real = this->real - c.real;
            comp.imag = this->imag - c.imag;
            return comp;
        }
        void printComplex()
        {
            cout << "Real      : " << this->real << endl;
            cout << "Imaginary : " << this->imag << endl;
        }
};
int main()
{
    Complex a, b, c, d;
    cout << "Setting first complex number " << endl;
    a.readComplex();
    cout << "Setting second complex number " << endl;
    b.readComplex();
    /* Adding two complex numbers */
    cout << "Addition of a and b    : " << endl;
    c = a.add(b);
    c.printComplex();
    /* Subtracting two complex numbers */
    cout << "Subtraction of a and b : " << endl;
    d = a.subtract(b);
    d.printComplex();
}
```

Output:

CALCULATION OF AREA AND VOLUME

1. area of circle

2. area of rectangle

3. area of triangle

4. area of square

5. area of the room

Enter your choice 1

enter the value of radius of the circle

3

area of the circle is 28.26

Program 2:Program for finding the area of circle, rectangle and room using function overloading

Procedure:

By using function overloading , we perform the area of circle,rectangle and room.

1.we can find the area of circle by using the following equation:
Area = 3.14 * (radius)2

2.we can find the area of rectangle by using the following equation:
Area = Breadth * Length

3.we can find the area of room by using the following equation:
(2*h)*(l+b)

Program:

```
#include <iostream.h>

#include<conio.h>

class cal_area

{

        public:

                void area(int r);

                void area(int l,int b);

                void area(float t,int d,int e);
```

```
                void area(long a);
                void area(int,int,float);
};
void cal_area::area(int r)
{
        cout<<"area of the circle is "<<3.14*r*r;
}
void cal_area::area(int l,int b)
{
        cout<<"area of the rectangle is"<<l*b;
}
void cal_area::area(float t,int d,int e)
{
        cout<<"area of the triangle is"<<t*d*e;
}
void cal_area::area(long a)
{
        cout<<"area of the square is"<<a*a;
}
void  cal_area::area(int l,int b,float h)
{
        cout<<"area of the room is"<<(2*h)*(l+b);
}
int main()
{
        int r,d,e,l,b;
        float t,c,h;
        long a;
        int ch;
```

```
double j,f;
long int g;
cal_area ob;
cout<<"\tCALCULATION OF AREA AND VOLUME";
cout<<"\n\n1. area of circle";
cout<<"\n2. area of rectangle";
cout<<"\n3. area of triangle";
cout<<"\n4. area of square";
cout<<"\n5. area of the room";
cout<<"\n\tEnter your choice ";
cin>>ch;
switch(ch)
{
	case 1:
		cout<<"enter the value of radius of the circle \n";
		cin>>r;
		ob.area(r);
		break;
	case 2:
		cout<<"enter the sides of rectangle \n";
		cin>>l>>b;
		ob.area(l,b);
		break;
	case 3:
		cout<<"enter the sides of triangle \n";
		cin>>d>>e;
		ob.area(0.5,d,e);
		break;
	case 4:
```

```
                cout<<"enter the sides of square\n";
                cin>>a;
                ob.area(a);
                break;
        case 5:
                cout<<"enter the length, width and height of the room\n";
                cin>>l>>b>>h;
                ob.area(l,b,h);
                break;
        default:
                cout<<"\nThe choice entered is a wrong choice";
    }
    getch();
}
```

Output:

```
enter the first complex number
Enter the real and imaginary parts : 2
4
enter the second complex number
Enter the real and imaginary parts : 3
5
Addition of a and b    :
Real      : 5
Imaginary : 9
Subtraction of a and b :
Real      : -1
Imaginary : -1
```

3.Program for finding the volume of box using constructor overloading

Procedure:

we can find volume of box by using the following equation:
length*bredth*height

Program:

```
#include<iostream.h>

class volume

{

public:

int length,bredth,height;

volume()

{

cout<<"enter length,bredth and height\n";

cin>>length>>bredth>>height;

}

volume(int v)

{

length=bredth=height=v;

}

volume(int l,int b,int h)

{

length=l;

bredth=b;

height=h;

}

void cal_volume()

{

int a;

cout<<"volume of the box=";

a=(length*bredth*height);
```

```
cout<<a;

}

};

void main()

{

volume v;

volume v1(3,6,9);

volume v2(4);

v.cal_volume();

v1.cal_volume();

v2.cal_volume();

}
```

Output:

```
enter length,bredth and height

4

3

5

volume of the box=60volume of the box=162volume of the box=64
```

4. program for sorting n elements using bubble sort technique

Procedure:

Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary.

1. if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, you mustn't swap the element.

2. Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped. This completes the first step of bubble sort.

3.If there are *n* elements to be sorted then, the process mentioned above should be repeated *n-1* times to get required result.

Program:

```
#include<iostream.h>
#include<conio.h>
#define SIZE 20
class bubblesort
{
 int array[SIZE],n;
 public:
      void readarray();
      void bubble_sort();
      void displayarray();
 };
 void bubblesort::readarray()
 {
 cout<<"read n";
 cin>>n;
 for (int i=0;i<n;i++)
           cin>>array[i];
 }
void bubblesort::displayarray()
 {
 cout<<"elements in an array\n";
 for (int i=0;i<n;i++)
           cout<<array[i]<<endl;
 }
 void bubblesort::bubble_sort()
 {
 int temp;
 for(int i=0;i<n;i++)
```

```
{
        for(int j=i+1;j<n;j++)
        {
                if(array[i]>array[j])
                {
                        temp=array[i];
                        array[i]=array[j];
                        array[j]=temp;
                }
        }
 }
}
void main()
{
clrscr();
bubblesort s;
s.readarray();
cout<<"after sorting";
s.bubble_sort();
s.displayarray();
}
```

Output:

```
read n5
25
37
21
46
63
after sortingelements in an array
```

21

25

37

46

63

5.Sort given elements using selection sort

Procedure:

Consider an array A [1 . . n] containing a sequence of elements with length n that is to be sorted.

1. The list is divided into two sub lists, sorted and unsorted, which are divided by an imaginary wall.

2. We find the smallest element from the unsorted sub list and swap it with the element at the beginning of the unsorted data.

3.After each selection and swapping, the imaginary wall between the two sub lists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.

4. A list of n elements requires n-1 passes to completely rearrange the data.

Program:

```
#include<iostream.h>

#include<conio.h>

#define SIZE 20

class selectionsort

{

 int array[SIZE],n;

 public:

      void readarray();

      void selection_sort();

      void displayarray();

 };

 void selectionsort::readarray()

 {
```

```
 cout<<"read n";
 cin>>n;
 for (int i=0;i<n;i++)
        cin>>array[i];
 }
void selectionsort::displayarray()
 {
 cout<<"elements in an array\n";
 for (int i=0;i<n;i++)
        cout<<array[i]<<endl;
 }
 void selectionsort::selection_sort()
 {
 int min,temp;
 min=0;
 for (int i=0;i<n;i++)
 {
        for(int j=i;j<n;j++)
                if(array[j]<array[min])
                {
                        min=j;
                }
        temp=array[i];
        array[i]=array[min];
        array[min]=temp;
 }
 }
void main()
{
```

```
clrscr();
selectionsort s;
s.readarray();
cout<<"after sorting";
s.selection_sort();
s.displayarray();
}
```

Output:

```
read n5
22
45
12
76
63
after sortingelements in an array
12
22
76
45
63
```

6.Sort given elements using insertion sort

Procedure:

Consider an array A[1 . . n] containing a sequence of elements with length n that is to be sorted.

1. The second element of an array is compared with the elements that appear before it (only first element in this case). If the second element is smaller than first element, second element is inserted in the position of first element. After first step, first two elements of an array will be sorted.
2. The third element of an array is compared with the elements that appear before it (first and second element). If third element is smaller than first element, it is inserted in the position of first element. If third element is larger than first element but, smaller than second element, it is inserted in the position of second element. If third element is

larger than both the elements, it is kept in the position as it is. After second step, first three elements of an array will be sorted.

3. Similarly, the fourth element of an array is compared with the elements that appears before it (first, second and third element) and the same procedure is applied and that element is inserted in the proper position. After third step, first four elements of an array will be sorted.

If there are *n* elements to be sorted. Then, this procedure is repeated *n-1* times to get sorted list of array.

Program:

```
#include<iostream.h>

#include<conio.h>

#define SIZE 20

class insertionsort

{

 int array[SIZE],n;

 public:

     void readarray();

     void insertion_sort();

     void displayarray();

 };

 void insertionsort::readarray()

 {

 cout<<"read n";

 cin>>n;

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

          cin>>array[i];

 }

void insertionsort::displayarray()

 {

 cout<<"elements in an array\n";

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

```
        cout<<array[i]<<endl;
}
void insertionsort::insertion_sort()
{
int temp;
  for (int i=1;i<n;i++)
  {
        temp=array[i];
        for(int j=i-1;array[j]>temp&&j>=0;j=j-1)
        {
        array[j+1]=array[j];
        }
        array[j+1]=temp;
  }
}
void main()
{
clrscr();
insertionsort s;
s.readarray();
cout<<"after sorting";
s.insertion_sort();
s.displayarray();
}
```

Output:

enter the size of array

5

enter the array

34

23

56

54

17

sorted list

17 23 34 54 56

7.Sort given elements using merge sort

Procedure:

1. The length of the list is 0 or 1, and then it is considered as sorted.
2. Otherwise, divide the unsorted list into 2 lists each about half the size.
3. Sort each sub list recursively. Implement the step 2 until the two sub lists are sorted.
4. As a final step, combine (merge) both the lists back into one sorted list.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

class merge_sort

{

public:

void msort(int [],int [],int ,int );

void merge(int [],int [],int ,int ,int );

};

int n,a[10],b[10];

void main()

{

merge_sort m;

int i;

cout<<"enter the size of array\n";

cin>>n;
```

```
cout<<"enter the array\n";
for(i=0;i<n;i++)
cin>>a[i];
m.msort(a,b,0,n-1);
cout<<"sorted list\n";
for(i=0;i<n;i++)
cout<<a[i];
getch();
}
void merge_sort::msort(int a[],int b[],int begin,int end)
{
int center;
if(begin<end)
{
center=(begin+end)/2;
msort(a,b,begin,center);
msort(a,b,center+1,end);
merge(a,b,begin,center+1,end);
}
}
void merge_sort::merge(int a[],int b[],int lpos,int rpos,int rightend)
{
int i,leftend,num,tpos;
leftend=rpos-1;
tpos=lpos;
num=rightend-lpos+1;
while(lpos<=leftend&&rpos<=rightend)
{
        if(a[lpos]<=a[rpos])
```

```
        {
        b[tpos]=a[lpos];
        tpos++;
        lpos++;
        }
        else
        {
        b[tpos]=a[rpos];
        tpos++;
        rpos++;
        }
}
while(lpos<=leftend)
{
        b[tpos]=a[lpos];
        tpos++;
        lpos++;
        }
while(rpos<=rightend)
{
        b[tpos]=a[rpos];
        tpos++;
        rpos++;
}
for(i=0;i<num;i++,rightend--)
a[rightend]=b[rightend];
}
```

Output:

enter the size of array

5

enter the array

30

20

50

49

10

sorted list

10

20

30

49

50

8.Sort given elements using quick sort

Procedure:

1. Select an element, pivot, from the list.
2. Rearrange the elements in the list, so that all elements those are less than the pivot are arranged before the pivot and all elements those are greater than the pivot are arranged after the pivot. Now the pivot is in its position.
3. Sort the both sub lists – sub list of the elements which are less than the pivot and the list of elements which are more than the pivot recursively

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

class quick_sort

{

public:

int n,a[10];

void quicksort(int[],int,int);
```

```
void exchange(int[],int,int);
};
void main()
{
quick_sort q;
int i;
cout<<"enter the size of array\n";
cin>>q.n;
cout<<"enter the array\n";
for(i=0;i<q.n;i++)
cin>>q.a[i];
q.quicksort(q.a,0,q.n-1);
cout<<"sorted list\n";
for(i=0;i<q.n;i++)
cout<<q.a[i]<<"\t";
getch();
}
void quick_sort::quicksort(int a[],int begin,int end)
{
int i,j,pivot,c;
if(begin<end)
{
i=begin;
j=end;
c=(i+j)/2;
exchange(a,c,i);
pivot=begin;
while(i<j)
{
```

```
while(a[i]<=a[pivot])
i++;
while(a[j]>a[pivot])
j--;
if(i<j)
exchange(a,i,j);
}
exchange(a,pivot,j);
quicksort(a,begin,j-1);
quicksort(a,j+1,end);
}
}
void quick_sort::exchange(int a[],int i,int j)
{
int temp;
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
```

Output:

```
enter the size of array
5
enter the array
35
28
69
57
26
sorted list
```

26

28

35

57

69

9.Implement the following operations on single linked list.

(i) creation II)insertion iii)deletion iv)display

procedure:

1.creating a node

Step 1: Allocating memory for the node head by using the malloc function.

Step 2: Read the head-> value

Step 3:makes the head->next value NULL.

2.insert at begin

Step 1: Allocate memory for the temp node p by using the malloc function.

Step 2: read the p-> value.

Step 3: store the address of head in p-> next.

Step 4: make p as head.

3.insertion at middle

Step 1: Read the position(pos) and the value to insert at that position.

Step 2:Allocate memory for node p and insert the value (p->value).

Step 2:Traverse the list from head until reaches to the pos.

Step 3:if the position obtained then insert the node at that position.

4.insertion at end

Step 1: Allocate memory for the temp node p by using the malloc function.

Step 2: read the p-> value.

Step 3:Traverse the list until reaches to last node(q).

Step 4:Assign the address of p in q->next.

5.Deletion at begin

Step 1: Initially head contains the starting node address

Step 2 : Assign the head reference value to head then head contains the address of 2nd node

6.Deletion at middle

Step 1:read the element (elem) to delete in the list.

Step 2:Traverse the list from head and compare every node(q) in the list with elem if not assign q to p and q->next to q until it reaches to the elem

step 3:Assign q reference(q->next) to p reference(p->next).

7.Deletion at end

Step 1: Initially head contains the starting node address

Step 2 :Traverse the list from head to Last but one node(p).

Step 3:Assign null to p reference.

8.Traverse the list

Step 1:Assign the starting address to p.

Step 2:Traverse the list from p and print the value of each node until it reaches to the last node

Program:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>
#include<stdio.h>
struct node
{
int value;
struct node *next;
};
class single
{
```

```
public:
node *head;
single()
{
head=NULL;
}
void create();
void insertbegin();
void insertmiddle();
void insertend();
void deletebegin();
void deletemiddle();
void deleteend();
void traverse();
};
void single::create()
{
head=(struct node *)malloc(sizeof(struct node));
cout<<"enter the  value";
cin>>head->value;
head->next=NULL;
}
void single::insertbegin()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at begin\n";
cin>>p->value;
p->next=head;
```

```
head=p;
}
void single::insertmiddle()
{
struct node *p,*q;
int pos,count=1,flag;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at middle\n";
cin>>p->value;
cout<<"enter the position\n";
cin>>pos;
q=head;
while(q->next!=NULL)
{
count++;
if(count==pos)
break;
q=q->next;
}
p->next=q->next;
q->next=p;
}
void single::insertend()
{
struct node *p,*q;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at end\n";
cin>>p->value;
q=head;
```

```
while(q->next!=NULL)
    q=q->next;
q->next=p;
p->next=NULL;
}
void single::deletebegin()
{
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
head=head->next;
}
void single::deletemiddle()
{
struct node *q;
int ele;
if(head==NULL)
cout<<"there is no item to delete\n";
else
{
cout<<"enter the element to delete\n";
cin>>ele;
q=head;
while(q->value==ele)
{
q=q->next;
}
```

```
q->next=q->next->next;
}
}
void single::deleteend()
{
struct node *p,*q;
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
{
q=head;
while(q->next!=NULL)
{
p=q;
q=q->next;
}
p->next=NULL;
}
}
void single::traverse()
{
struct node *p;
p=head;
while(p->next!=NULL)
{
cout<<p->value;
p=p->next;
```

```
}
cout<<p->value;
}
int n,i;
void main()
{
single sll;
int ch;
char c;
do
{
cout<<"single linked list operations\n";
cout<<"1.create a list\n";
cout<<"2.insert begin\n";
cout<<"3.insertmiddle\n";
cout<<"4.insert end\n";
cout<<"5.delete begin\n";
cout<<"6.delete middle\n";
cout<<"7.delete end\n";
cout<<"8.traverse\n";
cout<<"9.exit\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
        case 1:sll.create();
                break;
        case 2:sll.insertbegin();
                break;
```

```
        case 3:sll.insertmiddle();
                break;
        case 4:sll.insertend();
                break;
        case 5:sll.deletebegin();
                break;
        case 6:sll.deletemiddle();
                break;
        case 7:sll.deleteend();
                break;
        case 8:sll.traverse();
                break;
        case 9:exit(0);
                break;
        default:cout<<"enter correct choice";
        }
        cout<<"do you want to continue\n";
        fflush(stdin);
        cin>>c;
        }while(c=='y');

getch();
}
```

Output:

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

1

enter the value 10

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

2

enter the value to insert at begin

20

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

4

enter the value to insert at end

30

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

3

enter the value to insert at middle

40

enter the position

2

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

8

20

10

40

30

do you want to continue

n

10.Implement the following operations on double linked list

i)creation ii)insertion iii)deletion iv)display

Procedure:

Procedure for creating the first node:-
1. Allocating memory for the first node.
2. Assign data and consider that node as head and tail node.

1. Insert an item in the list:

Insert an item at the beginning of the list:-

Procedure:-
1. Allocate a new node
2. Insert new element
3. Make new node point to old head and old head pointing to the new node

4. Update head to point to new node

Insert an item at the end of the list:-

Procedure:-

1. Allocate a new node
2. Insert new element
3. Have new node **next**point to null
4. Have tail(last) node point to new node and new node previous pointing to the tail node
5. Update tail to point to new node

Insert an item at the middle of the list:-

Procedure:-

1. Allocate a new node
2. Insert new element
3. Break pointer connection at the corresponding position(ex:position=2)
4. Re-connect the pointers(previous and next) for the new node.

Delete an item from the list :-

Delete an item from the beginning of the list:-

Procedure:-

1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node

Delete an item at the end of the list:-

Procedure:-

1.Update tail to point to previous node in the list

2. Remove the link of tail node (deallocate memory) and store the null value.

Delete an item at the middle of the list:-

Here we must specify the position of the node or value of the node in the linked list to remove the node.

Procedure:-

1.Traverse the double linked list upto position-1.

2.Update the (previous and next)links of position-1 and position +1.

3.Remove the node with the given position.

Search an item from the double linked list:-

In double linked list, we can search an item from the starting point(head node) and moving the pointer to the next node or from the ending point(tail node) and moving the pointer to the previous node.

Procedure:-

1.Compare the search item with the current node in the list. If the info of the current node is equal to the search item, stop the search; otherwise, make the next node the current node.

2.Repeat Step one until either an item in the list equal to the search item is found, or no more data is left in the list to compare with the search item.

Find the length of the double linked list:-

We can find the length of the double linked list in 2 ways, from head to tail or from tail to head. We use count variable to store the length of a double linked list. Initially count is initialized to zero.

Procedure:-

1.Traverse the list from head(tail) node to tail(head) node and increment the count by 1.

2.Finally count contains the length of the list.

Reverse the list:-

We can print the double linked list in reverse order by traversing the list from tail node to head node.

Procedure:-

1. Start from the tail node.
2. Fetch the data from that node and print it.
3. Advance pointer to the previous node and perform the previous steps.

Traverse the list:-

We can traverse the double linked list in 2 ways, from head to tail and tail to head node.

Procedure for traverse the list:-

1. Firstly move to the head(tail) node.
2. Fetch the data from the node and perform the operations such as arithmetic operation or any operation depending on data type.
3. After performing operation, advance pointer to next(previous) node and perform all above steps on Visited node.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<alloc.h>

struct node

{

int value;

struct node *next;

struct node *prev;

};

class doublell

{

public:

node *head,*tail;

doublell()

{

head=NULL;

tail=NULL;
```

```
}
void create();
void insertbegin();
void insertmiddle();
void insertend();
void deletebegin();
void deletemiddle();
void deleteend();
void traverse();
void nthposition();
};
int n,i;
void main()
{
doublell dll;
int ch;
char c;
do
{
cout<<"double linked list operations\n1.create a list\n2.insert begin\n3.insertmiddle\n4.insert end\n5.delete begin\n6.delete middle\n7.delete end\n8.traverse\n9.nthposition\n10.exit\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
        case 1:dll.create();
        break;
        case 2:dll.insertbegin();
        break;
```

```
        case 3:dll.insertmiddle();
        break;
        case 4:dll.insertend();
        break;
        case 5:dll.deletebegin();
        break;
        case 6:dll.deletemiddle();
        break;
        case 7:dll.deleteend();
        break;
        case 8:dll.traverse();
        break;
        case 9:dll.nthposition();
        break;
        case 10:exit(0);
        break;
        default:cout<<"enter correct choice";
        }
        cout<<"do you want to continue\n";
        fflush(stdin);
        cin>>c;
        }while(c=='y');

getch();
}
void doublell::create()
{
head=(struct node *)malloc(sizeof(struct node));
if(head!=NULL)
```

```
{
cout<<"enter the  value";
cin>>head->value;
head->next=NULL;
head->prev=NULL;
tail=head;
}
else
cout<<"can not create element\n";
}
void doublell::insertbegin()
{
        struct node *p;
        p=(struct node *)malloc(sizeof(struct node));
        if(p!=NULL)
        {
                cout<<"enter the value to insert at begin\n";
                cin>>p->value;
                p->next=head;
                p->prev=NULL;
                head->prev=p;
                head=p;
        }
        else
        cout<<"cannot insert\n";
}
void doublell::insertmiddle()
{
struct node *p,*q;
```

```
p=(struct node *)malloc(sizeof(struct node));
int pos,count=1;
if(p!=NULL)
{
cout<<"enter the value to insert at middle\n";
cin>>p->value;
cout<<"enter the position\n";
cin>>pos;
q=head;
while(q->next!=NULL)
{
count++;
if(count==pos)
break;
q=q->next;
}
p->next=q->next;
p->next->prev=p;
q->next=p;
p->prev=q;
}
else
cout<<"can not insert\n";
}
void doublell::insertend()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
if(p!=NULL)
```

```
{
cout<<"enter the value to insert at end\n";
cin>>p->value;
tail->next=p;
p->prev=tail;
p->next=NULL;
tail=p;
}
else
cout<<"can not insert\n";
}
void doublell::deletebegin()
{
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
{
head=NULL;
tail=NULL;
}
else
{
head=head->next;
head->prev=NULL;
}
}
void doublell::deletemiddle()
{
struct node *q;
```

```
int pos,count=1;
if(head==NULL)
cout<<"there is no item to delete\n";
else
{
cout<<"enter the element to delete\n";
cin>>pos;
q=head;
while(q->next!=tail)
{
count++;
if(count==pos)
break;
q=q->next;
}
q->next=q->next->next;
q->next->prev=q;
}
}
void doublell::deleteend()
{
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
{
head=NULL;
tail=NULL;
}
else
```

```
{
tail=tail->prev;
tail->next=NULL;
}
}
void doublell::traverse()
{
struct node *p;
p=head;
while(p->next!=NULL)
{
cout<<p->value<<endl;
p=p->next;
}
cout<<p->value<<endl;
}
void doublell::nthposition()
{
struct node *p;
int count=1,pos;
printf("enter the position from the last\n");
scanf("%d",&pos);
p=tail;
while(p->prev!=NULL)
{
if(count==pos)
printf("the element at the %d position from the end is %d",pos,p->value);
count++;
p=p->prev;
```

```
}
}
```

Output:

```
double linked list operations
1.create a list
2.insert begin
3.insertmiddle
4.insert end
5.delete begin
6.delete middle
7.delete end
8.traverse
9.exit
enter your choice
1
enter the value 5
do you want to continue
y
double linked list operations
1.create a list
2.insert begin
3.insertmiddle
4.insert end
5.delete begin
6.delete middle
7.delete end
8.traverse
9.exit
enter your choice
```

2

enter the value to insert at begin

10

do you want to continue

y

double linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

4

enter the value to insert at end

20

do you want to continue

y

double linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

3

enter the value to insert at middle

30

enter the position

2

do you want to continue

y

double linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

8

10

5

30

20

do you want to continue

n

11.Implement the following operations on circular linked list

i)creation ii)insertion iii)deletion iv)display

Procedure:

A linked list in which the last node points to the first node is called a circular single linked list. In a circular linked list with more than one node,it is convenient to make the pointer of last node pointing to the first node.Instead of making the last pointer as null, we place the address of first node.

The usual operations on a circular list are as follows: Initialize the list (to an empty state), determine if the list is empty, destroy the list, print the list, find the length of the list, search the list for a given item, insert an item in the list, delete an item from the list, and copy the list.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<alloc.h>

struct node

{

int value;

struct node *next;

}*head;

class circular

{

public:

circular()

{

head=NULL;

}

void create();

void insertbegin();
```

```
void insertmiddle();
void insertend();
void deletebegin();
void deletemiddle();
void deleteend();
void traverse();
};
void circular::create()
{
head=(struct node *)malloc(sizeof(struct node));
cout<<head;
if(head!=NULL)
{
cout<<"enter the  value";
cin>>head->value;
head->next=head;
}
else
cout<<"can not create element\n";
}
void circular::insertbegin()
{
        struct node *p,*q;
        q=head;
        p=(struct node *)malloc(sizeof(struct node));
        if(p!=NULL)
        {
                cout<<"enter the value to insert at begin\n";
                cin>>p->value;
```

```
                cout<<p;
                while(q->next!=head)
                q=q->next;
                p->next=head;
                q->next=p;
                head=p;
        }
        else
        cout<<"cannot insert\n";
}
void circular::insertmiddle()
{
struct node *p,*q;
int pos,count=1,flag;
p=(struct node *)malloc(sizeof(struct node));
if(p!=NULL)
{
cout<<"enter the value to insert at middle\n";
cin>>p->value;
cout<<"enter the position\n";
cin>>pos;
q=head;
while(q->next!=head)
{
count++;
if(count==pos)
break;
q=q->next;
}
```

```
p->next=q->next;
q->next=p;
}
else
cout<<"can not insert\n";
}
void circular::insertend()
{
struct node *p,*q;
p=(struct node *)malloc(sizeof(struct node));
if(p!=NULL)
{
cout<<"enter the value to insert at end\n";
cin>>p->value;
q=head;
while(q->next!=head)
    q=q->next;
q->next=p;
p->next=head;
}
else
cout<<"can not insert\n";
}
void circular::deletebegin()
{
struct node *p;
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==head)
```

```
head=NULL;
else
{
p=head;

while(p->next!=head)
        p=p->next;
p->next=head->next;
head=head->next;
}
}
void circular::deletemiddle()
{
struct node *p;
int ele;
if(head==NULL)
cout<<"there is no item to delete\n";
else
{
cout<<"enter the element to delete\n";
cin>>ele;
p=head;
while(p->value==ele)
{
p=p->next;
}
p->next=p->next->next;
}
}
```

```
void circular::deleteend()
{
struct node *p,*q;
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==head)
head=NULL;
else
{
q=head;
while(q->next!=head)
{
p=q;
q=q->next;
}
p->next=q->next;
}
}
void circular::traverse()
{
struct node *p;
p=head;
while(p->next!=head)
{
cout<<p->value;
p=p->next;
}
cout<<p->value;
```

```
}
int n,i;
void main()
{
circular cll;
int ch;
char c;
do
{
cout<<"circular linked list operations\n";
cout<<"1.create a list\n";
cout<<"2.insert begin\n";
cout<<"3.insertmiddle\n";
cout<<"4.insert end\n";
cout<<"5.delete begin\n";
cout<<"6.delete middle\n";
cout<<"7.delete end\n";
cout<<"8.traverse\n";
cout<<"9.exit\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
        case 1:cll.create();
        break;
        case 2:cll.insertbegin();
        break;
        case 3:cll.insertmiddle();
        break;
```

```
        case 4:cll.insertend();
        break;
        case 5:cll.deletebegin();
        break;
        case 6:cll.deletemiddle();
        break;
        case 7:cll.deleteend();
        break;
        case 8:cll.traverse();
        break;
        case 9:exit(0);
        break;
        default:cout<<"enter correct choice";
        }
        cout<<"do you want to continue\n";
        fflush(stdin);
        cin>>c;
        }while(c=='y');

getch();
}
```

Output:

```
circular linked list operations
1.create a list
2.insert begin
3.insertmiddle
4.insert end
5.delete begin
6.delete middle
```

7.delete end

8.traverse

9.exit

enter your choice

1

enter the value 80

do you want to continue

y

circular linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

2

enter the value to insert at begin

20

do you want to continue

y

circular linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

4

enter the value to insert at end

40

do you want to continue

y

circular linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

3

enter the value to insert at middle

50

enter the position

2

do you want to continue

y

circular linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

8

20

80

50

40

do you want to continue

n

12.Program for splitting given linked list

Procedure:

We perform all single linked list operations here also.

After performing all operations , we will give the number elements in the first list.Based on the number, it splits the list into two parts.

Program:

```
#include<iostream.h>

#include<conio.h>

#include<process.h>

#include<alloc.h>

#include<stdio.h>
```

```
struct node
{
int value;
struct node *next;
};
class single
{
public:
node *head,*head1;
single()
{
head=NULL;
head1=NULL;
}
void create();
void insertbegin();
void insertmiddle();
void insertend();
void deletebegin();
void deletemiddle();
void deleteend();
void traverse();
void traverse(struct node *);
void splitlist();
};
void single::create()
{
head=(struct node *)malloc(sizeof(struct node));
cout<<"enter the  value";
```

```
cin>>head->value;
head->next=NULL;
}
void single::insertbegin()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at begin\n";
cin>>p->value;
p->next=head;
head=p;
}
void single::insertmiddle()
{
struct node *p,*q;
int pos,count=1,flag;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at middle\n";
cin>>p->value;
cout<<"enter the position\n";
cin>>pos;
q=head;
while(q->next!=NULL)
{
count++;
if(count==pos)
break;
q=q->next;
}
```

```
p->next=q->next;
q->next=p;
}
void single::insertend()
{
struct node *p,*q;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the value to insert at end\n";
cin>>p->value;
q=head;
while(q->next!=NULL)
    q=q->next;
q->next=p;
p->next=NULL;
}
void single::deletebegin()
{
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
head=head->next;
}
void single::deletemiddle()
{
struct node *q;
int ele;
if(head==NULL)
```

```
cout<<"there is no item to delete\n";
else
{
cout<<"enter the element to delete\n";
cin>>ele;
q=head;
while(q->value==ele)
{
q=q->next;
}
q->next=q->next->next;
}
}
void single::deleteend()
{
struct node *p,*q;
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
{
q=head;
while(q->next!=NULL)
{
p=q;
q=q->next;
}
p->next=NULL;
```

```
}
}
void single::traverse()
{
struct node *p;
p=head;
while(p->next!=NULL)
{
cout<<p->value<<endl;
p=p->next;
}
cout<<p->value;
}
void single::traverse(struct node *H)
{
struct node *p;
p=H;
while(p->next!=NULL)
{
cout<<p->value<<endl;
p=p->next;
}
cout<<p->value;
}
void single::splitlist()
{
int n1,count;
cout<<"enter how many number of elements you need in the first list";
cin>>n1;
```

```
struct node *p,*q;
p=(struct node *)malloc(sizeof(struct node));
q=(struct node *)malloc(sizeof(struct node));
p=head;
count=1;
while(p!=NULL)
{
q=p;
p=p->next;
if(count==n1)
{
 q->next=NULL;
 head1=p;
}
count++;
}
cout<<"after splitting";
cout<<"first list\n";
traverse(head);
cout<<"second list\n";
traverse(head1);
}
int n,i;
void main()
{
single sll;
int ch;
char c;
do
```

```
{
cout<<"single linked list operations\n";
cout<<"1.create a list\n";
cout<<"2.insert begin\n";
cout<<"3.insertmiddle\n";
cout<<"4.insert end\n";
cout<<"5.delete begin\n";
cout<<"6.delete middle\n";
cout<<"7.delete end\n";
cout<<"8.traverse\n";
cout<<"9.split list\n";
cout<<"10.exit\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
        case 1:sll.create();
                break;
        case 2:sll.insertbegin();
                break;
        case 3:sll.insertmiddle();
                break;
        case 4:sll.insertend();
                break;
        case 5:sll.deletebegin();
                break;
        case 6:sll.deletemiddle();
                break;
        case 7:sll.deleteend();
```

```
                break;
        case 8:sll.traverse();
                break;
        case 9:sll.splitlist();
                break;
        case 10:exit(0);
                break;
        default:cout<<"enter correct choice";
        }
        cout<<"do you want to continue\n";
        fflush(stdin);
        cin>>c;
        }while(c=='y');

getch();
}
```

Output:

```
single linked list operations
1.create a list
2.insert begin
3.insertmiddle
4.insert end
5.delete begin
6.delete middle
7.delete end
8.traverse
9.split list
10.exit
enter your choice
```

1

enter the value 10

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.split list

10.exit

enter your choice

2

enter the value to insert at begin

20

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.split list

10.exit

enter your choice

4

enter the value to insert at end

30

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.split list

10.exit

enter your choice

3

enter the value to insert at middle

40

enter the position

2

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.split list

10.exit

enter your choice

8

20

10

40

30

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.split list

10.exit

enter your choice

9

enter how many number of elements you need in the first list2

after splitting first list

20

10

second list

40

30

do you want to continue

n

13.Program for traversing the given linked list in reverse order

Procedure:

In single linked list , we print the list in normal order. If we want to print the list in reverse order then we must use recursion. We know only the starting address and in only one direction we can traverse the single linked list. So, we cannot traverse the list from last to first because we cannot know the address of last node.

Using recursion, we cannot print the value of the first node until we have printed the value of the second node in the list. Similarly, we cannot print the value of the second node until we have printed the value of the third node. Repeat the same until the size of the list is reduced to zero.

The recurrence relation for printing the list in reverse order is

reverse = { no action if List is empty
{ Print the tail and print the element if list is nonempty

Program:

```
#include <iostream.h>


typedef struct node {
    int data;            // will store information
    node *next;          // the reference to the next node
};
node *head;
```

```
int printList(node *traverse) {
    if (traverse->next == NULL) {
        return -1;
    }
    traverse=traverse->next;
    printList(traverse);
    cout << traverse->data << endl;
    return 0;
}
int main() {
    node *temp = NULL;
    node *begin = NULL;
cout<<"enter elements";
    for (int i = 0; i < 5; i++) {
        temp = new node;
        temp->data = i;
        if (begin == NULL) {
            begin = temp;
        }
        if (head != NULL) {
            head->next = temp;
        }
        head = temp;
        head->next = NULL;
    }
    head = begin;
cout<<"list in reverse order";
    printList(head);
```

```
    return 0;
}
```

Output:

```
enter elements
10
20
30
40
50
elements in reverse order
50
40
30
20
10
```

14.Merge two given linked lists

procedure:

1.create two lists in sorted order using single linked list.

2.Performing the merge operation on two lists using the merge sort algorithm.

Program:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct merg
{
int val;
struct merg *next;
```

```
};
class merging
{
public:
struct merg *createlist(int[],int);
void display(struct merg *);
struct merg *merglist(struct merg *,struct merg *);
 };
void main()
{
merging m;
struct merg *A,*B,*C,*D;
int a[10],b[10],i,n1,n2;
clrscr();
cout<<"enter the number of elements in first list\n";
cin>>n1;
cout<<"enter the values for the first list in sorted order\n";
for(i=0;i<n1;i++)
cin>>a[i];
cout<<"enter the number of elements in second list\n";
cin>>n2;
cout<<"enter the values for the second list in sorted order\n";
for(i=0;i<n2;i++)
cin>>b[i];
A=m.createlist(a,n1);
cout<<"first list\n";
m.display(A);
B=m.createlist(b,n2);
cout<<"second list\n";
```

```
m.display(B);
C=m.merglist(A,B);
cout<<"after merging\n";
m.display(C);
getch();
}
struct merg *createnode(int a)
{
struct merg *E;
E=(struct merg *)malloc(sizeof(struct merg));
E->val=a;
E->next=NULL;
return(E);
}
struct merg * merging::createlist(int c[],int n)
{
struct merg *A,*B,*C,*H=NULL;
int i;
for(i=0;i<n;i++)
{
A=createnode(c[i]);
if(H==0)
{
H=A;
B=A;
}
else
{
B->next=A;
```

```
B=A;
}
}
return(H);
}
void merging::display(struct merg *head)
{
struct merg *p;
p=head;
while(p->next!=NULL)
{
cout<<p->val<<endl;
p=p->next;
}
cout<<p->val<<endl;
}
struct merg * merging::merglist(struct merg *A,struct merg *B)
{
struct merg *C=NULL,*p,*q,*E,*X;
p=A;
q=B;
while(p&&q)
{
if(p->val<q->val)
{
E=createnode(p->val);
p=p->next;
}
else if(p->val>q->val)
```

```
{
E=createnode(q->val);
q=q->next;
}
else
{
E=createnode(p->val);
p=p->next;
q=q->next;
}
if(C==NULL)
{
C=E;
X=E;
}
else
{
X->next=E;
X=E;
}
}
while(p!=NULL)
{
E=createnode(p->val);
if(C==NULL)
{
C=E;
X=E;
}
```

```
else
{
X->next=E;
X=E;
}
p=p->next;
}
while(q!=NULL)
{
E=createnode(q->val);
if(C==NULL)
{
C=E;
X=E;
}
else
{
X->next=E;
X=E;
}
q=q->next;
}
return(C);
}
```

Output:

```
Enter number of elements in the first list
3
Enter the values for the first list in the sorted order
1
```

3

5

Enter number of elements in the second list

3

Enter the values for the second list in the sorted order

2

4

6

First list

1

3

5

Second list

2

4

6

After merging

1

2

3

4

5

6

15.Create a linked list to store the names of colors

Procedure:

The procedure is as same as the single linked list, but instead of numbers we insert colors.

Program:

#include<iostream.h>

```
#include<conio.h>
#include<process.h>
#include<alloc.h>
#include<stdio.h>
struct node
{
char colornames[15];
struct node *next;
};
class single
{
public:
node *head;
single()
{
head=NULL;
}
void create();
void insertbegin();
void insertmiddle();
void insertend();
void deletebegin();
void deletemiddle();
void deleteend();
void traverse();
void sorting();
};
void single::create()
{
```

```
head=(struct node *)malloc(sizeof(struct node));
cout<<"enter the  color name";
cin>>head->colorname;
head->next=NULL;
}
void single::insertbegin()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the colorname to insert at begin\n";
cin>>p->colorname;
p->next=head;
head=p;
}
void single::insertmiddle()
{
struct node *p,*q;
int pos,count=1,flag;
p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the colorname to insert at middle\n";
cin>>p->colorname;
cout<<"enter the position\n";
cin>>pos;
q=head;
while(q->next!=NULL)
{
count++;
if(count==pos)
break;
```

```
q=q->next;
}
p->next=q->next;
q->next=p;
}
void single::insertend()
{
struct node *p,*q;

p=(struct node *)malloc(sizeof(struct node));
cout<<"enter the colorname to insert at end\n";
cin>>p->colorname;
q=head;
while(q->next!=NULL)
    q=q->next;
q->next=p;
p->next=NULL;
}
void single::deletebegin()
{
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
head=head->next;
}
void single::deletemiddle()
{
```

```
struct node *q;
int ele;
if(head==NULL)
cout<<"there is no item to delete\n";
else
{
cout<<"enter the element to delete\n";
cin>>ele;
q=head;
while(q->value==ele)
{
q=q->next;
}
q->next=q->next->next;
}
}
void single::deleteend()
{
struct node *p,*q;
if(head==NULL)
cout<<"there is no item to delete\n";
else if(head->next==NULL)
head=NULL;
else
{
q=head;
while(q->next!=NULL)
{
p=q;
```

```
q=q->next;
}
p->next=NULL;
}
}
void single::traverse()
{
struct node *p;
p=head;
while(p->next!=NULL)
{
cout<<p->value<<endl;
p=p->next;
}
cout<<p->value<<endl;
}
void single::sorting()
{
struct node *p,*q;
int temp;
for(p=head;p!=NULL;p=p->next)
{
for(q=p;q!=NULL;q=q->next)
{
if(p->value>q->value)
{
temp=p->value;
p->value=q->value;
q->value=temp;
```

```
}
}
}
}
int n,i;
void main()
{
single sll;
int ch;
char c;
do
{
cout<<"single linked list operations\n";
cout<<"1.create a list\n";
cout<<"2.insert begin\n";
cout<<"3.insertmiddle\n";
cout<<"4.insert end\n";
cout<<"5.delete begin\n";
cout<<"6.delete middle\n";
cout<<"7.delete end\n";
cout<<"8.traverse\n";
cout<<"9.sorting\n";
cout<<"10.exit\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
        case 1:sll.create();
                break;
```

```
        case 2:sll.insertbegin();
                break;
        case 3:sll.insertmiddle();
                break;
        case 4:sll.insertend();
                break;
        case 5:sll.deletebegin();
                break;
        case 6:sll.deletemiddle();
                break;
        case 7:sll.deleteend();
                break;
        case 8:sll.traverse();
                break;
        case 9:sll.sorting();
                break;
        case 10:exit(0);
                break;
        default:cout<<"enter correct choice";
        }
        cout<<"do you want to continue\n";
        fflush(stdin);
        cin>>c;
        }while(c=='y');

getch();
}
```

Output:

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

1

enter the colorname red

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

2

enter the colorname to insert at begin

green

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

4

enter the colorname to insert at end

blue

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

3

enter the value to insert at middle

pink

enter the position

2

do you want to continue

y

single linked list operations

1.create a list

2.insert begin

3.insertmiddle

4.insert end

5.delete begin

6.delete middle

7.delete end

8.traverse

9.exit

enter your choice

8

green

pink

red

blue

do you want to continue

n

16.Implement stack operations using arrays

Procedure:

push operation

Step 1:if(top==size)

Write”stack overflow”

Step 2:Read “element”

Increment top by 1 and insert element at top of the stack

Pop operation

Step 1:if top==-1

Then write “stack underflow”

Step 2:Decrement top by 1

Program:

```
#include<iostream.h>
#define SIZE 10
class stack
{
int s[SIZE],top;
public:
stack()
{
top=-1;
}
void traverse()
{
for(int i=0;i<=top;i++)
        cout<<s[i]<<endl;
}
void push()
{
if (top==SIZE)
        cout<<"stack overflow\n";
else
        {
```

```
        int a;
        top=top+1;
        cout<<"enter the element\n";
        cin>>a;
        s[top]=a;
        }
}
void pop()
{
if(top==-1)
        cout<<"stack is underflow\n";
else
        {
        int a;
        a=s[top];
        cout<<"deleted element is"<<a;
        top=top-1;
        }
}
};
void main()
{
stack st;
int ch;
char c;
do
{
cout<<"list of operations performed on stack:\n";
cout<<"1.push\n";
```

```
cout<<"2.pop\n";
cout<<"3.traverse\n";
cout<<"enter choice\n";
cin>>ch;
switch(ch)
{
case 1:st.push();
      break;
case 2:st.pop();
      break;
case 3:st.traverse();
      break;
default:cout<<"enter correct choice\n";
}
cout<<"do you want to continue\n";
cin>>c;
}while(c=='y'||c=='Y');
}
```

Output:

```
List of operations performed on stack:
1.push
2.pop
3.traverse
enter choice
1
enter the element
10
do you want to continue
y
```

List of operations performed on stack:

1.push

2.pop

3.traverse

enter choice

1

enter the element

20

do you want to continue

y

List of operations performed on stack:

1.push

2.pop

3.traverse

enter choice

1

enter the element

30

do you want to continue

y

List of operations performed on stack:

1.push

2.pop

3.traverse

enter choice

3

10

20

30

do you want to continue

y

List of operations performed on stack:

1.push

2.pop

3.traverse

enter choice

2

the deleted element is 30

do you want to continue

n

17.Implement stack operations using Linked List

Procedure:

We can perform the same push and pop operations in stack but we are implementing using linked list.

For push , we use insertatbegin() function.

For pop , we use deletionatbegin() function.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<alloc.h>

struct node

{

int value;

struct node *next;

}*head;

class stackll
```

```
{
public:
stackll()
{
head=NULL;
}
void create();
void push();
void pop();
void display();
};
void main()
{
stackll st;
int n,ch,c;
do
{
cout<<"stack operations\n";
cout<<"enter your choice\n";
cout<<"1.creation\n";
cout<<"2.push\n";
cout<<"3.pop\n";
cout<<"4.display\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
case 1:
      st.create();
```

```
            break;
case 2:
        st.push();
        break;
case 3:
            st.pop();
            break;
case 4:
            st.display();
            break;
case 5:
            exit(0);
            break;
default:
            cout<<"enter correct choice\n";
            break;
}
            cout<<"do you want to continue y=1  or n=0\n";
            cin>>c;
       } while(c==1);
getch();
}
void stackll::create()
{
head=(struct node *)malloc(sizeof(struct node));
cout<<"enter value\n";
cin>>head->value;
head->next=NULL;
}
```

```
void stackll::push()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
if(p==NULL)
cout<<"stack overflow\n";
else
{
cout<<"enter value\n";
cin>>p->value;
p->next=head;
head=p;
}
}
void stackll::pop()
{
if(head==NULL)
cout<<"stack underflow\n";
else
{
cout<<"the deleted element is"<<head->value;
head=head->next;
}
}
void stackll::display()
{
struct node *p;
p=head;
if(head==NULL)
```

```
cout<<"no elements to display\n";
else
{
while(p!=NULL)
{
cout<<p->value;
p=p->next;
}
}
}
```

Output:

```
stack operations:
1.creation
2.push
3.pop
4.display
enter your choice
1
enter value
10
do you want to continue  y=1 or n=0
1
stack operations:
1.creation
2.push
3.pop
4.display
enter your choice
2
```

enter value

20

do you want to continue y=1 or n=0

1

stack operations:

1.creation

2.push

3.pop

4.display

enter your choice

2

enter value

30

do you want to continue y=1 or n=0

1

stack operations:

1.creation

2.push

3.pop

4.display

enter your choice

4

10

20

30

do you want to continue y=1 or n=0

1

stack operations:

1.creation

2.push

3.pop

4.display

enter your choice

3

the deleted element is 30do you want to continue y=1 or n=0

0

18.Implement queue operations using arrays

Procedure:

insert into the queue

Step 1: if rear is equal to size
Then write”queue is full”
Step 2: Increment rear by 1 and insert element at rear position

Delete element from the queue

Step 1:if(front==-1)

Then write ”queue underflow”

Step 2:increment front by 1

Program:

```
#include<iostream.h>

#define SIZE 10

class queue

{

int q[SIZE],front,rear;

public:

queue()

{

front=rear=-1;

}

void traverse()
```

```
{
for(int i=front;i<=rear&&front!=-1;i++)
        cout<<q[i]<<endl;
}
void insert()
{
if (rear==SIZE-1)
        cout<<"queue overflow\n";
else
        {
        if(front==-1)
                front=0;
        int a;
        rear=rear+1;
        cout<<"enter the element\n";
        cin>>a;
        q[rear]=a;
        }
}
void delete1()
{
if(front==SIZE)
        cout<<"there is no element to delete\n";
else if (front==rear)
        {
        cout<<"deleted element is"<<q[front];
        front=rear=-1;
        }
else
```

```
        {
        cout<<"deleted element is"<<q[front];
        front=front+1;
        }
}
};
void main()
{
queue qu;
int ch;
char c;
do
{
cout<<"list of operations performed on queue:\n";
cout<<"1.insert\n";
cout<<"2.delete\n";
cout<<"3.traverse\n";
cout<<"enter choice\n";
cin>>ch;
switch(ch)
{
case 1:qu.insert();
    break;
case 2:qu.delete1();
    break;
case 3:qu.traverse();
    break;
default:cout<<"enter correct choice\n";
}
```

```
cout<<"do you want to continue\n";
cin>>c;
}while(c=='y'||c=='Y');
}
```

Output:

```
list of operations performed on queue:
1.insert
2.delete
3.traverse
enter choice
1
enter the element
10
do you want to continue
y
list of operations performed on queue:
1.insert
2.delete
3.traverse
enter choice
1
enter the element
20
do you want to continue
y
list of operations performed on queue:
1.insert
2.delete
3.traverse
```

enter choice

1

enter the element

30

do you want to continue

y

list of operations performed on queue:

1.insert

2.delete

3.traverse

enter choice

3

10

20

30

do you want to continue

y

list of operations performed on queue:

1.insert

2.delete

3.traverse

enter choice

2

deleted element is 10 do you want to continue

y

19.Implement queue operations using Linked List

Procedure:

We can perform the same insertion and deletion operations , but using linked list.

For insertion, we use insertionatend() function.

For deletion, we use deletionatbegin() function

Program:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>
struct node
{
int value;
struct node *next;
}*head;
class queuell
{
public:
queuell()
{
head=NULL;
}
void create();
void insert();
void delete1();
void display();
};
void main()
{
queuell qu;
int n,ch,c;
```

```
do
{
cout<<"queue operations\n";
cout<<"enter your choice\n";
cout<<"1.creation\n";
cout<<"2.insert\n";
cout<<"3.delete\n";
cout<<"4.display\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
case 1:
      qu.create();
          break;
case 2:
      qu.insert();
      break;
case 3:
          qu.delete1();
          break;
case 4:
          qu.display();
          break;
case 5:
          exit(0);
          break;
default:
          cout<<"enter correct choice\n";
```

```
            break;
}
            cout<<"do you want to continue y=1  or n=0\n";
            cin>>c;
       } while(c==1);
getch();
}
void queuell::create()
{
head=(struct node *)malloc(sizeof(struct node));
cout<<"enter value\n";
cin>>head->value;
head->next=NULL;
}
void queuell::insert()
{
struct node *p;
p=(struct node *)malloc(sizeof(struct node));
if(p==NULL)
cout<<"queue overflow\n";
else
{
cout<<"enter value\n";
cin>>p->value;
q=head;
while(q->next!=NULL)
            q=q->next;
q->next=p;
}
```

```
}
void stackll::delete1()
{
if(head==NULL)
cout<<"queue underflow\n";
else
{
cout<<"the deleted element is"<<head->value;
head=head->next;
}
}
void stackll::display()
{
struct node *p;
p=head;
if(head==NULL)
cout<<"no elements to display\n";
else
{
while(p!=NULL)
{
cout<<p->value;
p=p->next;
}
}
}
```

Output:

queue operations

enter your choice

1.creation

2.insert

3.delete

4.display

enter your choice

1

enter value

10

do you want to continue y=1 or n=0

1

queue operations

enter your choice

1.creation

2.insert

3.delete

4.display

enter your choice

2

enter value

20

do you want to continue y=1 or n=0

1

queue operations

enter your choice

1.creation

2.insert

3.delete

4.display

enter your choice

2

enter value

30

do you want to continue y=1 or n=0

1

queue operations

enter your choice

1.creation

2.insert

3.delete

4.display

enter your choice

4

10

20

30

do you want to continue y=1 or n=0

1

queue operations

enter your choice

1.creation

2.insert

3.delete

4.display

enter your choice

3

the deleted element is 10 do you want to continue y=1 or n=0

0

20.Implement operations on Circular queue

Procedure:

A circular queue is an abstract data type that contains a collection of data which allows addition of data at the end of the queue and removal of data at the beginning of the queue. Circular queues have a fixed size.
Circular queue follows FIFO principle. Queue items are added at the rear end and the items are deleted at front end of the circular queue.

A circular queue is a queue in which all nodes are treated as circular such that the first node follows the last node.

Program:

```
#include<stdio.h>

#include<conio.h>

#define size 5

void insertcq();

void deletecq();

void display();

int cqueue[size],front=-1,rear=-1;

void main()

{

int ch,c;

do

{

printf("enter the circular queue operations\n");

printf("1.insert\n");

printf("2.delete\n");

printf("3.display\n");

printf("enter your choice\n");

scanf("%d",&ch);

switch(ch)

{
```

```
case 1:insertcq();
        break;
case 2:deletecq();
        break;
case 3:display();
        break;
default:printf("enter correct choice\n");
}
printf("do you want continue y=1 n=0\n");
scanf("%d",&c);
}while(c==1);
getch();
}
void insertcq()
{
int v;
if((rear==size-1&&front==0)||(rear+1==front))
printf("queue overflow\n");
else
{
if(front==-1)
front=0;
rear=(rear+1)%size;
printf("enter the element into the queue\n");
scanf("%d",&v);
cqueue[rear]=v;
}
}
void deletecq()
```

```
{
if(front==-1)
printf("queue underflow\n");
else
{
        if(front==rear)
        {
        printf("the deleted element from the queue is %d",cqueue[front]);
        front=-1;
        rear=-1;
        }
        else
        {
        printf("the deleted element from the queue is %d",cqueue[front]);
        front=(front+1)%size;
        }
}
}
void display()
{
int i;
printf("queue elements\n");
if(front>rear)
{
for(i=front;i<size;i++)
printf("%d\t",cqueue[i]);
for(i=0;i<=rear;i++)
printf("%d\t",cqueue[i]);
}
```

```
else
{
for(i=front;i<=rear;i++)
printf("%d\t",cqueue[i]);
}
}
```

Output:

```
enter the circular queue operations
1.insert
2.delete
3.display
enter your choice
1
enter the element into the queue
10
do you want to continue
y
enter the circular queue operations
1.insert
2.delete
3.display
enter your choice
1
enter the element into the queue
20
do you want to continue
y
enter the circular queue operations
```

1.insert

2.delete

3.display

enter your choice

1

enter the element into the queue

30

do you want to continue

y

enter the circular queue operations

1.insert

2.delete

3.display

enter your choice

3

queue elements

10

20

30

do you want to continue

y

enter the circular queue operations

1.insert

2.delete

3.display

enter your choice

2

the deleted element from the queue is 10

do you want to continue

n

21.Construct and implement operations on priority queue

Procedure:

A priority queue support the following operations:

- insert_with_priority: add an element to the queue with an associated priority.
- pull_highest_priority_element: remove the element from the queue that has the highest priority, and return it.

    This is also known as "pop_element(Off)", "get_maximum_element" or "get_front(most)_element".

Program:

```
 #include<iostream.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

class priority_queue

{

public:

int pri_que[MAX];

int front, rear;

void insert_by_priority(int);

void delete_by_priority(int);

void create();

void check(int);

void display_pqueue();

};

void main()

{

    int n, ch;

    priority_queue pqu;

    cout<<"\n1 - Insert an element into queue";
```

```
cout<<"\n2 - Delete an element from queue";
cout<<"\n3 - Display queue elements";
cout<<"\n4 - Exit";
pqu.create();
while (1)
{
	cout<<"\nEnter your choice : ";
	cin>>ch;

	switch (ch)
	{
	case 1:
	   cout<<"\nEnter value to be inserted : ";
	   cin>>n;
	   pqu.insert_by_priority(n);
	   break;
	case 2:
	   cout<<"\nEnter value to delete : ";
	   cin>>n;
	   pqu.delete_by_priority(n);
	   break;
	case 3:
	   pqu.display_pqueue();
	   break;
	case 4:
	   exit(0);
	default:
	   cout<<"\nChoice is incorrect, Enter a correct choice";
	}
```

```
    }
}
/* Function to create an empty priority queue */
void priority_queue::create()
{
    front = rear = -1;
}
/* Function to insert value into priority queue */
void priority_queue::insert_by_priority(int data)
{
    if (rear >= MAX - 1)
    {
            cout<<"\nQueue overflow no more elements can be inserted";
            return;
    }
    if ((front == -1) && (rear == -1))
    {
            front++;
            rear++;
            pri_que[rear] = data;
            return;
    }
    else
            check(data);
    rear++;
}

/* Function to check priority and place element */
void priority_queue::check(int data)
```

```
{
    int i,j;

    for (i = 0; i <= rear; i++)
    {
        if (data >= pri_que[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}
/* Function to delete an element from queue */
void priority_queue::delete_by_priority(int data)
{
    int i;
    if ((front==-1) && (rear==-1))
    {
        cout<<"\nQueue is empty no elements to delete";
        return;
    }
    for (i = 0; i <= rear; i++)
    {
        if (data == pri_que[i])
```

```
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }

        pri_que[i] = -99;
        rear--;
        if (rear == -1)
            front = -1;
        return;
        }
    }
    cout<<endl<<data<<" not found in queue to delete"<<endl;
}
/* Function to display queue elements */
void priority_queue::display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
        cout<<"\nQueue is empty";
        return;
    }
    for (; front <= rear; front++)
    {
        cout<<pri_que[front];
    }
    front = 0;
}
```

Output:

1 - Insert an element into queue

2 - Delete an element from queue";

3 - Display queue elements";

4 - Exit

Enter your choice :

1

Enter value to be inserted :

10

Enter your choice :

1

Enter value to be inserted :

20

Enter your choice :

1

Enter value to be inserted :

30

Enter your choice :

1

Enter value to be inserted :

40

Enter your choice :

3

40 30 20 10

Enter your choice :

2

Enter value to delete :

20

Enter your choice :

3

40 30 10

Enter your choice :

4

22.Implement operations on double ended queue

Procedure:

This differs from the queue abstract data type or First-In-First-Out List (FIFO), where elements can only be added to one end and removed from the other. This general data class has some possible sub-types:

- An input-restricted deque is one where deletion can be made from both ends, but insertion can be made at one end only.
- An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only.

Both the basic and most common list types in computing, queues and stacks can be considered specializations of deques, and can be implemented using deques.

Program:

```
#include<iostream.h>

#include<stdio.h>

#define N 5

class dequeue

{

public:

int deque[N],left,right;

dequeue()

{

left=right=0;

}

void inp();

void outp();

void insertleft()

{
```

```
int item;
if(((left==1)&&(right==N))||(left==right+1))
cout<<"overflow\n";
else
{
cout<<"Enter the element to be inserted\n";
cin>>item;
if(left==0)
{
left=1;
right=1;
}
else if(left==1)
left=N;
else
left=left-1;

deque[left]=item;
cout<<deque[left];
}
}
void display()
{
int i=0;
if((left==0)&&(right==0))
cout<<"The queue is empty\n";
else
{
if(left<=right)
```

```
{
for(i=left;i<=right;i++)
cout<<deque[i];
}
if(left>right)
{
cout<<"the deque is ";
for(i=left;i<=N;i++)
cout<<deque[i];
for(i=1;i<=right;i++)
cout<<deque[i];
}
}
}
void insertright()
{
int item;
if(((left==1)&&(right==N))||(left==right+1))
            cout<<"Overflow\n";
else
{
          if(left==0)
              {
              left=1;
              right=1;
              }
              else if(right==N)
              right=1;
              else
```

```
            right=right+1;
cout<<"Enter the element to be inserted\n";
cin>>item;
deque[right]=item;
}
}
 void deleteright()
{
int item;
if(left==0)
cout<<"Underflow\n";
else
{
item=deque[right];
if(left==right)
{
left=0;
right=0;
}
else if(right==1)
right=N;
else
right=right-1;
cout<<"The deleted element is"<<item;
}
}
void deleteleft()
{
int item;
```

```
if(left==0)
cout<<"Underflow\n";
else
{
item=deque[left];
if(left==right)
{
left=0;
right=0;
}
else if(left==N)
left=1;
else
left=left+1;
cout<<"The deleted element is"<<item;
}
}
};
void dequeue::inp()
{
int k;
do
{
cout<<"Choose:1.insert right 2.delete right 3.delete left 4.display 5.exit\n";
cin>>k;
switch(k)
{
case 1:insertright();
        break;
```

```
case 2:deleteright();
            break;
case 3:deleteleft();
            break;
case 4:display();
            break;
}
}while(k!=5);
}
void dequeue::outp()
{
int m;
do
{
cout<<"Choose:1.insert left 2.delete left 3.insert right 4.display 5.exit\n";
cin>>m;
switch(m)
{
case 1:insertleft();
            break;
case 2:deleteleft();
        break;
case 3:insertright();
        break;
case 4: display();
        break;
}
}while(m!=5);
}
```

```
main()
{
dequeue dqu;
int ch;
do{
cout<<"Choose:1.input restricted deque 2.output restricted deque 3.exit\n";
cin>>ch;
switch(ch)
{
case 1:dqu.inp();
        break;
case 2:dqu.outp();
        break;
}
}while(ch!=3);
}
```

Output:

```
choose:1. input restricted deque 2.output restricted deque 3.exit
1
Choose:1.insert right 2.delete right 3.delete left 4.display 5.exit
1
enter the element to be inserted
10
Choose:1.insert right 2.delete right 3.delete left 4.display 5.exit
1
enter the element to be inserted
20
Choose:1.insert right 2.delete right 3.delete left 4.display 5.exit
```

4

10 20

Choose:1.insert right 2.delete right 3.delete left 4.display 5.exit

5

choose:1. input restricted deque 2.output restricted deque 3.exit

3

23.Convert infix expression to postfix expression by using stack.

Procedure:

STEP 1 : Read the given infix expression into string called infix.

STEP 2 : Reverse the infix string and read one character at a time and perform the following operations :

If the read character is an operand, then add the operand to the prefix string.
If the read character is not an operand, then check
If the stack is not empty and precedence of the top of the
stack operator is higher than the read operator,
then pop the operator from stack and add this
operator to the prefix string.
Else push the operator onto the stack.

STEP 3 : Repeat **STEP 2** till all characters are processed from the input string.

STEP 4 : If stack is not empty, then pop the operator from stack and add this operator to the prefix string.

STEP 5 : Repeat **STEP 4** till all the operators are popped from the stack.

STEP 6 : Reverse the prefix string and display the result of the given infix expression or the resultant prefix expression stored in a string called prefix from this algorithm.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

class intopost

{
```

```
public:
void push(int);
void pop();
int symbol(char);
//void evaluate(char[]);
//void push1(int);
//int pop1();
int operation(int,int,char);
};
int i=0,j=0;
char po[20],c[20];
char s[10];
int top=-1;
void main()
{
intopost post;
cout<<"enter the infix expression\n";
gets(c);
while(c[i]!='\0')
{
if(isdigit(c[i]))
{
po[j]=c[i];
j++;
i++;
}
else if(c[i]=='(')
post.push(c[i]);
else if(c[i]==')')
```

```
{
while(s[top]!='(')
post.pop();
post.pop();
i++;
}
else
{
if(top==-1)
{
top++;
s[top]=c[i];
i++;
}
else
{
        while(post.symbol(s[top])>=post.symbol(c[i]))
        {
        post.pop();
        }
        post.push(c[i]);
        }
}
}
while(s[top]>-1)
post.pop();
po[j]='\0';
cout<<"the postfix expression is \n";
puts(po);
```

```
//post.evaluate(po);
getch();
}
int intopost::symbol(char op)
{
if(op=='^')return 3;
else if(op=='*'||op=='/')return 2;
else if(op=='+'||op=='-')return 1;
else return 0;
}
void intopost::pop()
{
if(s[top]!='(')
{
po[j]=s[top];
top--;
j++;
}
else
top--;
}
void intopost::push(int c)
{
top++;
s[top]=c;
i++;
}
```

Output:

enter the infix expression

1+2*3

the postfix expression is

123*+

24.Write a program for evaluate postfix expression

Procedure:

- Scan the Postfix string from left to right.
- Initialise an empty stack.
- If the scannned character is an operand, add it to the stack. If the scanned character is an operator, there will be atleast two operands in the stack.
    - If the scanned character is an Operator, then we store the top most element of the stack(topStack) in a variable temp. Pop the stack. Now evaluate topStack(Operator)temp. Let the result of this operation be retVal. Pop the stack and Push retVal into the stack.

    Repeat this step till all the characters are scanned.

- After all characters are scanned, we will have only one element in the stack. Return topStack.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

class post_eval

{

public:

void evaluate(char[]);

void push(int);

int pop();

int operation(int,int,char);

};

char po[20];
```

```
int stack[10],top=-1;
void main()
{
post_eval post;
cout<<"enter the postfix expression  \n";
gets(po);
puts(po);
post.evaluate(po);
getch();
}
void post_eval::evaluate(char e[])
{
int i=0,a,b,c,z;
while(e[i]!='\0')
{
if(isdigit(e[i]))
{
z=e[i]-48;
push(z);
i++;
}
else
{
b=pop();
a=pop();
c=operation(a,b,e[i]);
push(c);
i++;
}
```

```
}
cout<<"postfix evaluation="<<stack[top];
}
void post_eval::push(int x)
{
top++;
stack[top]=x;
}
int post_eval::pop()
{
int z;
z=stack[top];
top--;
return(z);
}
int post_eval::operation(int a,int b,char op)
{
int c;
if(op=='*')
c=a*b;
else if(op=='/')
c=a/b;
else if(op=='+')
c=a+b;
else
c=a-b;
return(c);
}
```

Output:

enter the postfix expression

123*+

postfix evaluation=7

25.Implement operations on two way stack

Procedure:

Create a data structure *twoStacks* that represents two stacks. Implementation of *twoStacks*should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported by *twoStacks*.

push1(int x) –> pushes x to first stack
push2(int x) –> pushes x to second stack

pop1() –> pops an element from first stack and return the popped element
pop2() –> pops an element from second stack and return the popped element

Implementation of *twoStack* should be space efficient.

Method 1 (Divide the space in two halves)
A simple way to implement two stacks is to divide the array in two halves and assign the half half space to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[n/2+1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

Method 2 (A space efficient implementation)
This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack2 starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks. This check is highlighted in the below code.

Program:

```
#include<iostream.h>

#include <stdio.h>

#define SIZE 10

class twowaystack
```

```
{
public:
int ar[SIZE];
int top1 ;
int top2 ;
twowaystack()
{
top1=-1;
top2=SIZE;
}
//Functions to push data
void push_stack1 (int data)
{
  if (top1 < top2 - 1)
  {
    ar[++top1] = data;
  }
  else
  {
    cout<<"Stack Full! Cannot Push\n";
  }
}
void push_stack2 (int data)
{
  if (top1 < top2 - 1)
  {
    ar[--top2] = data;
  }
  else
```

```
  {
   cout<<"Stack Full! Cannot Push\n";
  }
}
//Functions to pop data
void pop_stack1 ()
{
  if (top1 >= 0)
  {
    int popped_value = ar[top1--];
    cout<<popped_value<<"is being popped from Stack 1\n";
  }
  else
  {
    cout<<"Stack Empty! Cannot Pop\n";
  }
}
void pop_stack2 ()
{
  if (top2 < SIZE)
  {
    int popped_value = ar[top2++];
    cout<<popped_value<<"is being popped from Stack 2\n";
  }
  else
  {
    cout<<"Stack Empty! Cannot Pop\n";
  }
}
```

```
//Functions to Print Stack 1 and Stack 2
void print_stack1 ()
{
  int i;
  for (i = top1; i >= 0; --i)
  {
    cout<<ar[i];
  }
  cout<<"\n";
}
void print_stack2 ()
{
  int i;
  for (i = top2; i < SIZE; ++i)
  {
    cout<<ar[i];
  }
  cout<<"\n";
}
};
int main()
{
  //int ar[SIZE];
  twowaystack tws;
  int i;
  int num_of_ele;

  cout<<"We can push a total of 10 values\n";
```

```
//Number of elements pushed in stack 1 is 6
//Number of elements pushed in stack 2 is 4
for (i = 1; i <= 6; ++i)
{
 tws. push_stack1 (i);
  cout<<"Value Pushed in Stack 1 is "<< i<<"\n";
}
for (i = 1; i <= 4; ++i)
{
  tws.push_stack2 (i);
  cout<<"Value Pushed in Stack 2 is "<< i<<"\n";
}
//Print Both Stacks
tws.print_stack1 ();
tws.print_stack2 ();
//Pushing on Stack Full
cout<<"Pushing Value in Stack 1 is "<< 11<<"\n";
tws.push_stack1 (11);
//Popping All Elements From Stack 1
num_of_ele = tws.top1 + 1;
while (num_of_ele)
{
 tws.pop_stack1 ();
  --num_of_ele;
}
//Trying to Pop From Empty Stack
tws.pop_stack1 ();
```

```
  return 0;
}
```

Output :

```
We can push a total of 10 values
Value Pushed in Stack 1 is 1
Value Pushed in Stack 1 is 2
Value Pushed in Stack 1 is 3
Value Pushed in Stack 1 is 4
Value Pushed in Stack 1 is 5
Value Pushed in Stack 1 is 6
Value Pushed in Stack 2 is 1
Value Pushed in Stack 2 is 2
Value Pushed in Stack 2 is 3
Value Pushed in Stack 2 is 4
6 5 4 3 2 1
4 3 2 1
Pushing Value in Stack 1 is 11
Stack Full! Cannot Push
6 is being popped from Stack 1
5 is being popped from Stack 1
4 is being popped from Stack 1
3 is being popped from Stack 1
2 is being popped from Stack 1
1 is being popped from Stack 1
Stack Empty! Cannot Pop
```

26.Add two polynomials using linked list

Procedure:

We use a linked list to dynamically store user input of polynomial expressions and then we add two polynomials using some simple arithmetic. For this, we follow the simple strategy:

- Make a polynomial abstract datatype using ***struct*** which basically implements a linked list.
- We write different functions for Creating (ie, adding more nodes to the linked list) a polynomial function, Adding two polynomials and Showing a polynomial expression.

Program:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct poly
{
int con,pw;
struct poly *next;
};
class polynomial
{
public:
struct poly *createpoly(int[],int);
void display(struct poly *);
struct poly *addpoly(struct poly *,struct poly *);
 };
void main()
{
polynomial pol;
struct poly *A,*B,*C,*D;
int a[10],b[10],i,n1,n2;
clrscr();
cout<<"enter the power of first polynomial\n";
cin>>n1;
cout<<"enter the constant values\n";
for(i=0;i<=n1;i++)
```

```
cin>>a[i];
cout<<"enter the power of second polynomial\n";
cin>>n2;
cout<<"enter the constant values\n";
for(i=0;i<=n2;i++)
cin>>b[i];
A=pol.createpoly(a,n1);
pol.display(A);
B=pol.createpoly(b,n2);
pol.display(B);
C=pol.addpoly(A,B);
Cout<<”addition is”;
pol.display(C);
getch();
}
struct poly *createnode(int a,int i)
{
struct poly *E;
E=(struct poly *)malloc(sizeof(struct poly));
E->con=a;
E->pw=i;
E->next=NULL;
return(E);
}
struct poly * polynomial::createpoly(int c[],int n)
{
struct poly *A,*B,*C,*H=NULL;
int i;
for(i=n;i>=0;i--)
```

```
{
if(c[i]!=0)
{
A=createnode(c[i],i);
if(H==0)
{
H=A;
B=A;
}
else
{
B->next=A;
B=A;
}
}
}
return(H);
}

void polynomial::display(struct poly *head)
{
struct poly *p;
p=head;
while(p->next!=NULL)
{
//printf("%dX^%d+",p->con,p->pw);
cout<<p->con<<"X^"<<p->pw<<"+";
p=p->next;
}
```

```
cout<<p->con<<endl;
}

struct poly * polynomial::addpoly(struct poly *h1,struct poly *h2)
{
struct poly *p,*q,*A,*H=NULL,*B;
p=h1;
q=h2;
while(p&&q)
{
if(p->pw>q->pw)
{
A=createnode(p->con,p->pw);
p=p->next;
}
else if(p->pw<q->pw)
{
A=createnode(q->con,q->pw);
q=q->next;
}
else
{
A=createnode(q->con+p->con,q->pw);
p=p->next;
q=q->next;
}
if(H==0)
{
H=A;
```

```
B=A;
}
else
{
B->next=A;
B=A;
}
}
while(p!=NULL)
{
A=createnode(p->con,p->pw);
if(H==0)
{
H=A;
B=A;
}
else
{
B->next=A;
B=A;
}
p=p->next;
}
while(q!=NULL)
{
A=createnode(q->con,q->pw);
if(H==0)
{
H=A;
```

```
B=A;
}
else
{
B->next=A;
B=A;
}
p=p->next;
}
return(H);
}
```

Output:

```
enter the power of first polynomial
3
enter the constant valuess
1
2
3
4
enter the power of second polynomial
3
enter the constant values
4
3
2
1
4X^3+3X^2+2X^1+1
1X^3+2X^2+3X^1+4
addtion is
```

5X^3+5X^2+5X^1+5

27.Multiply two polynomials using linked list

Procedure:

We use a linked list to dynamically store user input of polynomial expressions and then we add two polynomials using some simple arithmetic. For this, we follow the simple strategy:

- Make a polynomial abstract datatype using ***struct*** which basically implements a linked list.
- We write different functions for Creating (ie, adding more nodes to the linked list) a polynomial function, multiply two polynomials and Showing a polynomial expression.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

struct poly

{

int con,pw;

struct poly *next;

};

class polynomial

{

public:

struct poly *createpoly(int[],int);

void display(struct poly *);

struct poly *mulpoly(struct poly *,struct poly *);

 };

void main()

{

polynomial pol;

struct poly *A,*B,*C,*D;

int a[10],b[10],i,n1,n2;

clrscr();
```

```
cout<<"enter the power of first polynomial\n";
cin>>n1;
cout<<"enter the constant values\n";
for(i=0;i<=n1;i++)
cin>>a[i];
cout<<"enter the power of second polynomial\n";
cin>>n2;
cout<<"enter the constant values\n";
for(i=0;i<=n2;i++)
cin>>b[i];
A=pol.createpoly(a,n1);
pol.display(A);
B=pol.createpoly(b,n2);
pol.display(B);
D=pol.mulpoly(A,B);
Cout<<”multiplication is”;
pol.display(D);
getch();
}
struct poly *createnode(int a,int i)
{
struct poly *E;
E=(struct poly *)malloc(sizeof(struct poly));
E->con=a;
E->pw=i;
E->next=NULL;
return(E);
}
struct poly * polynomial::createpoly(int c[],int n)
```

```
{
struct poly *A,*B,*C,*H=NULL;
int i;
for(i=n;i>=0;i--)
{
if(c[i]!=0)
{
A=createnode(c[i],i);
if(H==0)
{
H=A;
B=A;
}
else
{
B->next=A;
B=A;
}
}
}
return(H);
}
void polynomial::display(struct poly *head)
{
struct poly *p;
p=head;
while(p->next!=NULL)
{
//printf("%dX^%d+",p->con,p->pw);
```

```
cout<<p->con<<"X^"<<p->pw<<"+";
p=p->next;
}
cout<<p->con<<endl;
}
struct poly * polynomial::mulpoly(struct poly *A,struct poly *B)
{
struct poly *H=NULL,*p,*q,*E,*C,*F,*D;
int flag=0;
q=A;
while(q)
{
if(H==0)
{
        p=B;
        while(p)
        {
                E=createnode(q->con*p->con,q->pw+p->pw);
                if(H==0)
                {
                H=E;
                C=E;
                }
                else
                {
                C->next=E;
                C=E;
                }
        p=p->next;
```

```
        }
}
else
{
        p=B;
                while(p)
        {
          E=createnode(q->con*p->con,q->pw+p->pw);
          D=H;
          while(D)
          {
          if(D->pw==E->pw)
          {
          D->con=D->con+E->con;
          flag=1;
          break;
          }
          D=D->next;
          }
                D=H;
                if(flag!=1)
                {
                while(D->next!=NULL)
                        D=D->next;
                D->next=E;
                }
                flag=0;
                p=p->next;
                }
```

```
}
q=q->next;
}
return(H);
}
```

Output:

```
enter the power of first polynomial
3
enter the constant valuess
1
2
3
4
enter the power of second polynomial
3
enter the constant values
4
3
2
1
4X^3+3X^2+2X^1+1
1X^3+2X^2+3X^1+4
multiplication is
4X^6+11X^5+20X^4+30X^3+20X^2+11X^1+4
```

28.Construct BST and implement traversing techniques recursively

Procedure:

Insertion:

The basic idea for inserting the element into the Binary Search Tree is:

From the root node onwards, we compare with the value being inserted. If the value is lesser then we traverse through the left sub tree and if the value is greater we traverse through the right subtree and insert the node at proper position.

Inorder Traversal

In this traversal the left sub tree of the given node is visited first, then the value at the given node is printed and then the right sub tree of the given node is visited. This process is applied recursively all the node in the tree until either the left sub tree is empty or the right sub tree is empty.

Preorder traversal

In this traversal the value at the given node is printed first and then the left sub tree of the given node is visited and then the right sub tree of the given node is visited. This process is applied recursively all the node in the tree until either the left sub tree is empty or the right sub tree is empty.

Postorder Traversal

In this traversal the left sub tree of the given node is traversed first, then the right sub tree of the given node is traversed and then the value at the given node is printed. This process is applied recursively all the node in the tree until either the left sub tree is empty or the right sub tree is empty.

Program:

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

struct bstree

{

int data;

struct bstree *prev;

struct bstree *next;

}*root;


class bst_traverse

{

public:

bst_traverse()

{
```

```
root=NULL;
}
void insertion();
void preorder(struct bstree*);
void inorder(struct bstree*);
void postorder(struct bstree*);
struct bstree* createnode(int);
void create(struct btree*);
};
struct bstree *p,*q,*ptr;
int a,in[10],k=0,min,max;
void main()
{
bst_traverse b;
int ch,item;
char c,ch1;
do
{
cout<<"binary search tree operations\n";
cout<<"1.insertion\n";
cout<<"2.traversals\n";
cout<<"enter your choice\n";
cin>>ch;
switch(ch)
{
case 1:b.insertion();
            break;
case 2:cout<<"a.preorder traversals\n";
       cout<<"b.inorder traversals\n";
```

```
        cout<<"c.postorder traversals\n";
        cout<<"enter your choice\n";
        fflush(stdin);
        cin>>ch1;
        switch(ch1)
        {
        case 'a':b.preorder(root);
                        break;
        case 'b':b.inorder(root);
                        break;
        case 'c':b.postorder(root);
                        break;
        default:cout<<"enter correct choice\n";
        }
        break;
default:printf("enter correct choice\n");
}
cout<<"do you want to continue  yes=y no=n\n";
fflush(stdin);
cin>>c;
}while(c=='y');
getch();
}
void bst_traverse::insertion()
{
int n,i,x;
struct bstree *p,*e;
cout<<"enter the value\n";
cin>>x;
```

```
e=createnode(x);
if(root==NULL)
root=e;
else
{
p=root;
while(p)
{
if(p->data>e->data)
{
if(p->prev!=NULL)
p=p->prev;
else
{
p->prev=e;
break;
}
}
else
{
if(p->next!=NULL)
p=p->next;
else
{
p->next=e;
break;
}
}
}
```

```
}
}
struct bstree* bst_traverse::createnode(int a)
{
struct bstree *p;
p=(struct bstree*)malloc(sizeof(struct bstree));
p->data=a;
p->prev=NULL;
p->next=NULL;
return(p);
}
void bst_traverse::preorder(struct bstree *q)
{
if(q)
{
cout<<q->data<<"\t";
if(q->prev)
        preorder(q->prev);
if(q->next)
        preorder(q->next);
}
}
void bst_traverse::inorder(struct bstree *q)
{
if(q->prev)
        inorder(q->prev);
cout<<q->data<<"\t";
if(q->next)
        inorder(q->next);
```

```
}
void bst_traverse::postorder(struct bstree *q)
{
if(q->prev)
        postorder(q->prev);
if(q->next)
        postorder(q->next);
cout<<q->data<<"\t";
}
```

Output:

```
binary search tree operations:
1.insertion
2.traversal
enter your choice
1
enter the value
15
do you want to continue yes=y or no=n
y
binary search tree operations:
1.insertion
2.traversal
enter your choice
1
enter the value
20
do you want to continue yes=y or no=n
y
binary search tree operations:
```

1.insertion

2.traversal

enter your choice

1

enter the value

30

do you want to continue yes=y or no=n

y

binary search tree operations:

1.insertion

2.traversal

enter your choice

1

enter the value

10

do you want to continue yes=y or no=n

y

binary search tree operations:

1.insertion

2.traversal

enter your choice

1

enter the value

25

do you want to continue yes=y or no=n

y

binary search tree operations:

1.insertion

2.traversal

enter your choice

2

a. preorder traversal

b. inorder traversal

c.postorder traversal

enter your choice

a

15 10 20 30 25

do you want to continue yes=y or no=n

n

29.Implement preorder traversal on BST non recursively

Procedure:

1) Create an empty stack *nodeStack* and push root node to stack.
2) Do following while *nodeStack* is not empty.
2. a) Pop an item from stack and print it.

2.b) Push right child of popped item to stack

2.c) Push left child of popped item to stack

Right child is pushed before left child to make sure that left subtree is processed first.

Program:

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

struct node
{
        int ele;
        node *left;
        node *right;
};

typedef struct node *nodeptr;
class stack
{
        private:
```

```
        struct snode
        {
                nodeptr ele;
                snode *next;
        };
        snode *top;
public:
        stack()
        {
                top=NULL;
        }
        void push(nodeptr p)
        {
                snode *temp;
                temp = new snode;
                temp->ele = p;
                temp->next = top;
                top=temp;
        }

        void pop()
        {
                if (top != NULL)
                {
                nodeptr t;
                snode *temp;
                temp = top;
                top=temp->next;
                delete temp;
                }
        }

        nodeptr topele()
        {
                if (top !=NULL)
                        return top->ele;
                else
                        return NULL;
        }



        int isempty()
        {
        return ((top == NULL) ? 1 : 0);
        }

};

class bstree
```

```
{
        public:
                void insert(int,nodeptr &);

                void preorder(nodeptr);


};

void bstree::insert(int x,nodeptr &p)
{
        if (p==NULL)
        {
                p = new node;
                p->ele=x;
                p->left=NULL;
                p->right=NULL;
        }
        else
        {
                if (x < p->ele)
                        insert(x,p->left);
                else if (x>p->ele)
                        insert(x,p->right);
                else
                        cout<<"Element already Exits !";
        }
}



void bstree::preordernr(nodeptr p)
{
        stack s;
        while (1)
        {
        if  (p != NULL)
        {
                cout<<p->ele<<"-->";
                s.push(p);
                p=p->left;
        }
        else
        if (s.isempty())
        {
                cout<<"Stack is empty";
                return;
        }
        else
        {
                nodeptr t;
```

```
                t=s.topele();
                p=t->right;
                s.pop();
        }
        }
}

int main()
{
int ch,x;
bstree bst;
char c='y';
nodeptr root;
root=NULL;

do
{
        clrscr();
        cout<<"Binary Search Tree operations";

        cout<<"1.INSERTION";

        cout<<" 2. PREORDER";
        COUT<<" 3. EXIT";
COUT<<"Enter your choice :";
        cin>>ch;

        switch(ch)
        {
        case 1:
                cout<<"Enter the new element to get inserted :";
                cin>>x;
                bst.insert(x,root);

        case 2:
                if (root==NULL)
                        cout<<"Tree is empty";
                else
                {
                        cout<<"Preorder traversal (Non-Recursive) is :";
                        bst.preordernr(root);

                }
                break;

        case 3:
                exit(0);
        }
        cout<<"DO YOU WANT TO CONTINUE YES OR NO ";
        cin>>c;
        }while (c=='y' || c == 'Y');
```

```
        return 0;
}
```

Output:

```
Binary Search Tree operations

        1.INSERTION

         2. PREORDER
         3. EXIT
Enter your choice :
1
Enter the new element to get inserted :

10

DO YOU WANT TO CONTINUE YES OR NO

Y

Binary Search Tree operations

        1.INSERTION

         2. PREORDER
         3. EXIT
Enter your choice :
1
Enter the new element to get inserted :

5

DO YOU WANT TO CONTINUE YES OR NO

Y


Binary Search Tree operations

        1.INSERTION

         2. PREORDER
         3. EXIT
Enter your choice :
1
Enter the new element to get inserted :

20

DO YOU WANT TO CONTINUE YES OR NO
```

Y

Binary Search Tree operations

1.INSERTION

2. PREORDER
3. EXIT

Enter your choice :
1
Enter the new element to get inserted :

15

DO YOU WANT TO CONTINUE YES OR NO

Y

Binary Search Tree operations

1.INSERTION

2. PREORDER
3. EXIT

Enter your choice :
2

Preorder traversal (Non-Recursive) is :

5 10 20 15

DO YOU WANT TO CONTINUE YES OR NO

n

30.Implement inorder traversal on BST non recursively

procedure:

1.Create an empty stack S.

2.Initialize current node as root

3.Push the current node to S and set current = current->left until current is NULL

4.If current is NULL and stack is not empty then

a) Pop the top item from stack.
b) b) Print the popped item, set current = popped_item->right
c) c) Go to step 3.

5) If current is NULL and stack is empty then we are done.

Program:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class node
{
public:
class node *left;
class node *right;
int data;
};
class tree: public node
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
                root->left=NULL;
```

```
                root->right=NULL;
                return;
        }
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
        temp=search(root,ch);
        if(temp->data>ch)
                temp->left=temp1;
        else
                temp->right=temp1;
}
node *search(node *temp,int ch)
{
        if(root== NULL)
        {
                cout <<"no node present";
                return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
                return temp;

        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
             { if(temp->right==NULL) return temp;
             search(temp->right,ch);
```

```
}            }

void display(node *temp)
{
        if(temp==NULL)
         return ;
        display(temp->left);
      cout<<temp->data << " ";
        display(temp->right);
}
void inorder( node *root)
{
        node *p;
        p=root;
        top=0;
        do
        {
                while(p!=NULL)
                {
                        stk[top]=p->data;
                        top++;
                        p=p->left;
                }
                if(top>0)
                {
                        p=pop(root);
                        cout << p->data;
                        p=p->right;
                }
```

```
        }while(top!=0 || p!=NULL);
}
node * pop(node *p)
{
int ch;
ch=stk[top-1];
if(p->data==ch)
{
top--;
return p;
}
if(p->data>ch)
pop(p->left);
else
pop(p->right);
}
};
void main()
{
        tree t1;
        int ch,n,i;
        clrscr();
        while(1)
        {
                cout <<"\n1.INSERT\n2.DISPLAY 3.INORDER TRAVERSE \n 4.EXIT\nEnter
your choice:";
                cin >> ch;
                switch(ch)
                {
```

```
                case 1:   cout <<"enter no of elements to insert:";
                          cout<<"\n enter the elements";
                          cin >> n;
                          for(i=1;i<=n;i++)
                          {  cin >> ch;
                             t1.insert(ch);
                          }
                           break;
                case 2:   t1.display(t1.root);break;
                case 3:   t1.inorder(t1.root);break;
                case 4:   exit(1);
                }
        }
}
```

Output:

binary search tree operations

1.INSERT

2.DISPLAY

3.INORDER TRAVERSE

4.EXIT

Enter your choice:

1

enter no of elements to insert

5

enter elements

34

23

56

43

17

binary search tree operations

1.INSERT

2.DISPLAY

3.INORDER TRAVERSE

4.EXIT

Enter your choice:

3

17 23 34 43 56

binary search tree operations

1.INSERT

2.DISPLAY

3.INORDER TRAVERSE

4.EXIT

Enter your choice:4

31.Implement postorder traversal on BST non recursively

Procedure:

1.Create an empty stack.

2.1. Do the following while root is not NULL

a) Push root's right child and then root to stack.

b) Set root as root's left child.

2.2 Pop an item from stack and set it as root.

a) If the popped item has a right child and the right child is at the top of stack, then remove the right child from stack, push the root back and set root as root's right child.

b)else print root's data and set root as NULL.

2.3 Repeat steps 2.1 and 2.2 while stack is not empty.

Program:

```
#include<iostream.h>

#include<conio.h>
```

```
#include<stdlib.h>
class node
{
public:
class node *left;
class node *right;
int data;
};
class tree: public node
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
                root->left=NULL;
                root->right=NULL;
                return;
        }
```

```
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
        temp=search(root,ch);
        if(temp->data>ch)
                temp->left=temp1;
        else
                temp->right=temp1;

}
node *search(node *temp,int ch)
{
        if(root== NULL)
        {
                cout <<"no node present";
                return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
                return temp;

        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
             { if(temp->right==NULL) return temp;
             search(temp->right,ch);

}            }
```

```
void display(node *temp)
{
        if(temp==NULL)
         return ;
        display(temp->left);
      cout<<temp->data << " ";
        display(temp->right);
}
void postorder( node *root)
{
        node *p;
        p=root;
        top=0;
        while(1)
        {
        while(p!=NULL)
        {
                stk[top]=p->data;
                top++;
                if(p->right!=NULL)
                        stk[top++]=-p->right->data;
                p=p->left;
        }
        while(stk[top-1] > 0 || top==0)
        {
           if(top==0) return;
           cout << stk[top-1] <<" ";
           p=pop(root);
        }
```

```
            if(stk[top-1]<0)
            {
              stk[top-1]=-stk[top-1];
              p=pop(root);
      }     }
}
node * pop(node *p)
{
int ch;
ch=stk[top-1];
if(p->data==ch)
{
top--;
return p;
}
if(p->data>ch)
pop(p->left);
else
pop(p->right);
}
};
void main()
{
            tree t1;
            int ch,n,i;
            clrscr();
            while(1)
            {
```

```
                cout <<"\n1.INSERT\n2.DISPLAY 3.POSTORDER TRAVERSE\n 4.EXIT\nEnter
your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:   cout <<"enter no of elements to insert:";
                          cout<<"\n enter the elements";
                          cin >> n;
                          for(i=1;i<=n;i++)
                          {  cin >> ch;
                             t1.insert(ch);
                          }
                           break;
                case 2:   t1.display(t1.root);break;
                case 3:   t1.postorder(t1.root); break;
                case 4:   exit(1);
                }
        }
}
```

Output:

binary search tree operations

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:

1

enter no of elements to insert

5

enter elements

34

23

56

43

17

binary search tree operations

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:

3

17 23 43 56 34

binary search tree operations

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:4

32.Implement binary search techniques recursively.

Procedure:

1. In binary search, we first compare the key with the item in the middle position of the array.

2. If there's a match, we can return immediately.

3. If the key is less than the middle element, then the item must lie in the lower half of the array.

4. If the key is greater than the middle item, then the item must lie in the upper half of the array. So we repeat the procedure on the lower (or upper) half of the array.

Program:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
class bin_search
{
public:
int binary(int a[],int n,int m,int l,int u);
};
void main()
{
    bin_search bin;
    int a[10],i,n,m,c,l,u;
    cout<<"Enter the size of an array: ";
    cin>>n;
    cout<<"Enter the elements of an array in sorted order: " ;
    for(i=0;i<n;i++)
    {
            cin>>a[i];
    }
    cout<<"Enter the element to search: ";
    cin>>m;
    l=0,u=n-1;
    c=bin.binary(a,n,m,l,u);
    if(c==0)
            cout<<"Number is not found.";
    else
            cout<<"Number is found.";
    getch();
 }
```

```
int bin_search::binary(int a[],int n,int m,int l,int u)
{
    int mid,c=0;
    if(l<=u)
    {
            mid=(l+u)/2;
            if(m==a[mid])
            {
               c=1;
            }
            else if(m<a[mid])
            {
               return binary(a,n,m,l,mid-1);
            }
            else
               return binary(a,n,m,mid+1,u);
    }
    else
            return c;
}
```

Output:

```
Enter the size of an array:
7
Enter the elements of an array in sorted order:
10
20
30
40
```

50

60

70

enter the element to search

40

number is found