Data Structures Using C Lab List 2022 - 2023

3RD SEMESTER BCA AND BSC CS

1. *Sort a given list of strings*
2. *Reverse a string using pointers.*
3. *Implement Pattern matching algorithm.*
4. *Append 2 arrays*
5. *Search an element in the array using binary search.*
6. *Read a sparse matrix and display its triplet representation using array.*
7. *Create a singly linked list of n nodes and display it.*
8. *Delete a given node from a singly linked list.*
9. *Sort a singly linked list.*
10. *Create a singly linked list and search an element from that list.*
11. *Create a doubly linked list of integers and display in forward and backward direction.*
12. *Addition of 2 polynomials using array.*
13. *Implement Stack using array*
14. *Implement Stack using linked list*
15. *Evaluation of postfix expression.*
16. *Implement Queue using array.*
17. *Implement Queue using linked list.*
18. *Traverse a binary search tree in preorder*
19. *Traverse a binary search tree in inorder*
20. *Traverse a binary search tree in postorder.*
21. *Search an element in a binary search tree*
22. *Implement exchange sort*
23. *Implement selection sort.*
24. *Implement insertion sort.*
25. *Implement quick sort.*

1. Sort a given list of strings

ALGORITHM

Step 1. Start

Step 2. Declare str[20][20] and t[20] as character arrays. Also i, j and n as int

Step 3. Read n, number of strings

Step 4. Set i=0

Step 5. Repeat steps 6 and 7 until i<n

Step 6. Read str[i]

Step 7. Increment i by one.

Step 8. Set i=0

Step 9. Repeat steps 10 to 14 until i<n-1

Step 10. Set j=i+1

Step 11. Repeat steps 12 and 13 until j<n

Step 12. Check if strcmp(str[j],str[i])<0 then swap the strings str[j] and str[i]

Step 13. Increment j by one

Step 14. Increment i by one

Step 15. Set i=0

Step 16. Repeat steps 17 and 18 until i<n

Step 17. print str[i]

Step 18. Increment i by one.

Step 19. Stop

SOURCE CODE

```
#include<string.h>
#include<stdio.h>
void main()
{
    char str[20][20],t[20];
    int n, i, j;
    printf("How many strings : ");
    scanf("%d",&n);
    printf("Enter the strings : \n");
    for(i=0;i<n;i++)
    {
        scanf("%s",&str[i]);
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(strcmp(str[j],str[i])<0)
            {
                strcpy(t,str[i]);
                strcpy(str[i],str[j]);
                strcpy(str[j],t);
            }
        }
    }
    printf("Sorted strings are : \n");
    for(i=0;i<n;i++)
    {
        printf("\t%s\n",str[i]);
    }
    getch();
}
```

OUTPUT

```
How many strings : 5
Enter the strings :
banana
apple
orange
grapes
guava
Sorted strings are :
        apple
        banana
        grapes
        guava
        orange
```

2. Reverse a string using pointers.

ALGORITHM

Step 1. Start
Step 2. Declare str[100], ch as character, p and q as character pointers, len and i as int.
Step 3. Read str
Step 4. Store length of str into len
Step 5. Set p= address of first character of str.
Step 6. Set q= address of last character of str.
Step 7. Set i=0.
Step 8. Repeat the steps 9 to 12 until i<len/2
Step 9. Swap the characters pointed by p and q.
Step 10. increment p by one.
Step 11. Decrement q by one.
Step 12. Increment i by one.
Step 13. Print str, the reversed string.
Step 14. Stop

SOURCE CODE

```c
#include <stdio.h>
#include <string.h>
void main()
{
    char str[100], ch, *p, *q;
    int len, i;
    printf("Enter a string        :  ");
    gets(str);
    len = strlen(str);
```

```
    p=str;
    q= str+len-1;
    for (i = 0; i < len / 2; i++)
    {
        ch = *q;
        *q = *p;
        *p = ch;
        p++;
        q--;
    }
    printf("Reverse of the string : %s\n", str);
    getch();
}
```

OUTPUT

```
Enter a string        :  how are you
Reverse of the string : uoy era woh
```

3. Implement Pattern matching algorithm.

ALGORITHM

Step 1. Start
Step 2. Declare s[20], r[20] and q[20] as character arrays, ls, lr, i, j and count as int.
Step 3. Set count=0.
Step 4. Read s, the string
Step 5. Read r, the pattern to match
Step 6. Set ls=length of s and lr=length of r.
Step 7. Convert r and s to lower case
Step 8. set i=0.
Step 9. Repeat steps 10 to 12 until i<ls-lr
Step 10. Call strncpy(q, s+i, lr) , to copy substring of s of length lr to q
Step 11. if strcmp(q,r)=0, then print pattern is found at i+1 th position and increment count by one.
Step 12. Increment i by one
Step 13. If count = 0, print pattern not found.
Step 14. Stop.

SOURCE CODE

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s[50], r[20],q[20];
```

```
    int ls,lr, i, j, c, count=0;
    printf("Enter the string  : ");
    gets(s);
    printf("Enter the pattern : ");
    gets(r);
    ls = strlen(s);
    lr = strlen(r);
    strlwr(s);
    strlwr(r);
    for (i = 0; i <= ls-lr; i++)
    {
        strncpy(q,s+i,lr);
        if(strcmp(q,r)==0)
        {
            printf("\tPattern found at position %d \n", i+1);
            count++;
        }
    }
    if(count==0)
    {
        printf("Pattern not found..!");
    }
    getch();
}
```

OUTPUT

```
Enter the string  : have a nice day, eat an icecream..
Enter the pattern : ice
        Pattern found at position 9
        Pattern found at position 25
```

4. Append 2 arrays

ALGORITHM

Step 1. Start.
Step 2. Declare a[20], b[20], c[20], i, m, n as int.
Step 3. Read m, size of first array.
Step 4. Set i=0.
Step 5. Repeat steps 6 to 8 until i<m.
Step 6. Read a[i].
Step 7. Set c[i]=a[i].
Step 8. Increment i by one.
Step 9. Read n, size of second array.
Step 10. Set i=0.

Step 11. Repeat steps 12 to 14 until i<n.
Step 12. Read b[i], second array.
Step 13. Set c[m+i]=b[i].
Step 14. Increment i by one.
Step 15. Print first array a[m].
Step 16. Print second array b[n].
Step 17. Print appended array c[m+n].
Step 18. Stop.

SOURCE CODE

```
#include<stdio.h>
void main()
{
   int a[20],b[20],c[40],i,m,n;
   printf("Enter number of elements in first array : ");
   scanf("%d",&m);
   printf("\tEnter the elements : ");
   for(i = 0; i < m; i++)
   {
      scanf("%d",&a[i]);
      c[i]=a[i];
   }
   printf("Enter number of elements in second array : ");
   scanf("%d",&n);
   printf("\tEnter the elements : ");
   for(i = 0; i < n; i++)
   {
      scanf("%d",&b[i]);
      c[m+i]=b[i];
   }
   printf("\nArray one     : ");
   for(i=0;i<m;i++)
   {
      printf("%d ",a[i]);
   }
   printf("\nArray two     : ");
   for(i=0;i<n;i++)
   {
      printf("%d ",b[i]);
   }
   printf("\nAppended array : ");
   for(i=0;i<m+n;i++)
   {
      printf("%d ",c[i]);
   }
   getch();
}
```

OUTPUT

```
Enter number of elements in first array : 3
        Enter the elements : 3 6 4
Enter number of elements in second array : 4
        Enter the elements : 5 7 11 9

Array one     : 3 6 4
Array two     : 5 7 11 9
Appended array : 3 6 4 5 7 11 9
```

5. Search an element in the array using binary search.

ALGORITHM

Step 1. Start.
Step 2. Declare a[20], n, mid, big, i, end, x as int.
Step 3. Read n, size of the array.
Step 4. Repeat the steps 5 to 6 until i<n.
Step 5. Read a[i] in ascending order.
Step 6. Increment i by one.
Step 7. Read x, the element to search.
Step 8. Set big=0, end=n-1
Step 9. Repeat steps 10 to 12 until big<=end
Step 10. Set mid=(big+end)/2
Step 11. If x=a[mid] , then print “element found”. Go to step 14.
Step 12. if a[mid]<x then set big=mid+1, else set end= mid-1.
Step 13. print “not found”.
Step 14. Stop.

SOURCE CODE

```c
#include<stdio.h>
void main()
{
    int a[20],n,mid,big,i,end,x;
    printf("Enter the limit : ");
    scanf("%d",&n);
    printf("Enter the elements in ascending order :\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Enter the element to be searched : ");
    scanf("%d",&x);

    big=0;
```

```
    end=n-1;
    while(big<=end)
    {
        mid=(big+end)/2;
        if(x==a[mid])
        {
            printf("%d is found at position %d",x, mid+1);
            getch();
            return;
        }
        else if(a[mid]<x)
            big=mid+1;
        else
            end=mid-1;
    }
    printf("%d is not found in the array",x);
    getch();
}
```

OUTPUT

```
Enter the limit : 5
Enter the elements in ascending order :
3 8 9 11 14
Enter the element to be searched : 11
11 is found at position 4
```

6. Read a sparse matrix and display its triplet representation using array.

ALGORITHM

Step 1. Start
Step 2. Declare s[20][20], m[3][20], r, c, i, j and k as int.
Step 3. Read r,c – order of the matrix
Step 4. Set i=0.
Step 5. Repeat the steps 6 to 10 until i<r
Step 6. Set j=0.
Step 7. Repeat the steps 8 to 9 until j<c
Step 8. Read s[i][j]
Step 9. Increment j by one.
Step 10. Increment i by one.
Step 11. Set i=0.
Step 12. Repeat the steps 13 to 18 until i<r
Step 13. Set j=0.
Step 14. Repeat the steps 15 to 17 until j<c

Step 15. print s[i][j]

Step 16. if s[i][j] not equal to zero, then

Step 16.a) set m[0][k]=i

Step 16.b) set m[1][k]=j

Step 16.c) set m[2][k]=s[i][j]

Step 16.d) increment k by one.

Step 17. Increment j by one.

Step 18. Increment i by one.

Step 19. Set i=0.

Step 20. Repeat the steps 21 to 25 until i<3

Step 21. Set j=0.

Step 22. Repeat the steps 23 and 24 until j<k

Step 23. Print m[i][j], triplet form of sparse matrix

Step 24. Increment j by one.

Step 25. Increment i by one.

Step 26. Stop.

SOURCE CODE

```
#include<stdio.h>
void main()
{
    int s[20][20], m[3][20], r, c, i, j, k=0;
    char str[3][7]={"Row","Column","Value"};
    printf("Enter number of rows and columns in the matrix : ");
    scanf("%d%d",&r, &c);
    printf("Enter %d elements in the matrix : ",r*c);
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
          scanf("\t%d", &s[i][j]);
        }
    }
    printf("The sparse matrix : ");
    for(i=0; i<r; i++)
    {
        printf("\n\t");
        for(j=0; j<c; j++)
        {
          printf("%4d", s[i][j]);
          if(s[i][j]!=0)
          {
            m[0][k]=i;
            m[1][k]=j;
            m[2][k]=s[i][j];
            k++;
```

```
            }
        }
    }
    printf("\nTriplet representation of the sparse matrix :");
    for(i=0; i<3; i++)
    {
        printf("\n\t%7s :",str[i]);
        for(j=0; j<k; j++)
        {
          printf("%4d |", m[i][j]);
        }
    }
   getch();
}
```

OUTPUT

```
Enter number of rows and columns in the matrix : 4 3
Enter 12 elements in the matrix :
2 5 0 0 0 3 0 6 0 0 1 0
The sparse matrix :
          2   5   0
          0   0   3
          0   6   0
          0   1   0
Triplet representation of the sparse matrix :
           Row :   0 |   0 |   1 |   2 |   3 |
        Column :   0 |   1 |   2 |   1 |   1 |
         Value :   2 |   5 |   3 |   6 |   1 |
```

7. Create a singly linked list of n nodes and display it.

ALGORITHM

Global declarations

1. Define node as structure with elements next as pointer to structure node and data as int.
2. Declare start, temp, nw, last as pointers to structure node.
3. Declare n, x, i as int.

main() function

Step 1. Start.
Step 2. Declare ch as int.
Step 3. Set start=last=null.
Step 4. Read ch.
Step 5. If ch=1, call create(), go to step 4
Else if ch=2, call insert(),go to step 4
Else if ch=3, call display(),go to step 4

Else if ch=4, go to step 6

Step 6. Stop.

create() function

Step 1. Set start=last=null

Step 2. Read n, number of elements

Step 3. Call insert() n times

insert() function

Step 1. Create a new node and store its address in nw.

Step 2. Read nw->data

Step 3. Set nw->next=NULL

Step 4. If start=NULL then set start=last=nw,
else set last->next=nw and last=nw.

display() function

Step 1. Set temp=start

Step 2. Repeat steps 3 and 4 until temp!=NULL

Step 3. Print temp->data

Step 4. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *next;
    int data;
};
struct node *start, *nw, *temp, *last;
int n, x, i;
void display();
void create();
void insert();
void main()
{
int ch;
start=last=NULL;
    while(1)
    {
        printf("\nLinked List Operations\n----------------------\n");
        printf("1.Create New List\n2.Add Node\n3.Display List\n4.Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: create();
            break;
            case 2: insert();
```

```
                    printf("\t%d added..",x);
            break;
            case 3: display();
            break;
            case 4: return;
            default: printf("Invalid choice..");
        }
    }
}
void create()
{
    start=last=NULL;
    printf("how many elements : ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        insert();
    }
    printf("\tNew list created..");
}
void insert()
{
    nw = (struct node *) malloc(sizeof(struct node));
    printf("\tEnter the data :");
    scanf("%d",&x);
    nw->data=x;
    nw->next=NULL;
    if(start == NULL)
    {
        start = last = nw;
    }
    else
    {
        last->next= nw;
        last=nw;
    }
}
void display()
{
    printf("Elements in the linked list : ");
    for(temp=start;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
```

OUTPUT

```
Linked List Operations
----------------------
1.Create New List
2.Add Node
3.Display List
4.Exit
Enter your choice : 1
how many elements : 3
        Enter the data :6
        Enter the data :2
        Enter the data :9
        New list created..
Linked List Operations
----------------------
1.Create New List
2.Add Node
3.Display List
4.Exit
Enter your choice : 3
Elements in the linked list : 6->2->9->NULL
Linked List Operations
----------------------
1.Create New List
2.Add Node
3.Display List
4.Exit
Enter your choice : 2
        Enter the data :8
        8 added..
Linked List Operations
----------------------
1.Create New List
2.Add Node
3.Display List
4.Exit
Enter your choice : 3
Elements in the linked list : 6->2->9->8->NULL
Linked List Operations
----------------------
1.Create New List
2.Add Node
3.Display List
4.Exit
Enter your choice : 4
```

8. Delete a given node from a singly linked list.

ALGORITHM

Global declarations

1. Define node as structure with elements next as pointer to structure node and data as int.
2. Declare start, temp, nw, last as pointers to structure node.

main() function

Step 1. Start.
Step 2. Declare ch as int.
Step 3. Set start=last=null.
Step 4. Read ch.
Step 5. If ch=1, call create(), go to step 4
Else if ch=2, call delete_node(),go to step 4
Else if ch=3, call display(),go to step 4
Else if ch=4, go to step 6
Step 6. Stop.

create() function

Step 1. Declare n, x, i as int.
Step 2. Set start=last=null
Step 3. Read n, number of elements
Step 4. Repeat steps 5 to 8 n times
Step 5. Create a new node and store its address in nw.
Step 6. Read nw->data
Step 7. Set nw->next=NULL
Step 8. If start=NULL then set start=last=nw,
else set last->next=nw and last=nw.

delete_node() function

Step 1. declare prev as pointer to node, x as int.
Step 2. if start=NULL, print "list empty" and exit from function. Else go to step 3
Step 3. read x, node to delete.
Step 4. Set temp=start
Step 5. Repeat steps 6 to 7 until temp!=NULL
Step 6. If temp->data = x, then
a. If temp=start, set start=start->next, else set prev->next=temp->next
b. Call free(temp)- to deallocate memory
c. Exit from the function
Else set prev=temp
Step 7. Set temp=temp->next
Step 8. Print "element not found"

display() function

Step 1. Set temp=start
Step 2. Repeat steps 3 and 4 until temp!=NULL
Step 3. Print temp->data
Step 4. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
```

```c
#include<stdlib.h>
struct node
{
    struct node *next;
    int data;
};
struct node *start, *nw, *temp, *last;
void display();
void create();
void delete_node();
void main()
{
int ch;
start=last=NULL;
    while(1)
    {
        printf("\nLinked List Operations\n--------------------\n");
        printf("1.Create New List\n2.Delete a Node\n3.Display List\n4.Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: create();
            break;
            case 2: delete_node();
            break;
            case 3: display();
            break;
            case 4: return;
            default: printf("\tInvalid choice..");
        }
    }
}
void create()
{
   int n, x, i;
   start=last=NULL;
   printf("how many elements : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
    {
        nw = (struct node *) malloc(sizeof(struct node));
        printf("\tEnter the data :");
        scanf("%d",&x);
        nw->data=x;
        nw->next=NULL;
        if(start == NULL)
```

```
        {
            start = last = nw;
        }
        else
        {
            last->next= nw;
            last=nw;
        }
    }
    printf("New list is created..\n");
}
void display()
{
    printf("Elements in the linked list : ");
    for(temp=start;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
void delete_node()
{
    struct node*prev;
    int x;
    if(start==NULL)
    {
        printf("\tList is empty..!");
        return;
    }
    printf("\tEnter the node to delete : ");
    scanf("%d",&x);
    for(temp=start; temp!=NULL;temp=temp->next)
    {
        if(temp->data==x)
        {
            if(temp==start)
            {
                start=start->next;
            }
            else
            {
                prev->next=temp->next;
            }
            printf("\t%d is deleted..", x);
            free(temp);
            return;
        }
```

```
        prev=temp;
    }
    printf("\t%d not found..",x);
}
```

OUTPUT

```
Linked List Operations
----------------------
1.Create New List
2.Delete a Node
3.Display List
4.Exit
Enter your choice : 1
how many elements : 4
        Enter the data :3
        Enter the data :9
        Enter the data :4
        Enter the data :7
New list is created..

Linked List Operations
----------------------
1.Create New List
2.Delete a Node
3.Display List
4.Exit
Enter your choice : 3
Elements in the linked list : 3->9->4->7->NULL
Linked List Operations
----------------------
1.Create New List
2.Delete a Node
3.Display List
4.Exit
Enter your choice : 2
        Enter the node to delete : 4
        4 is deleted..
Linked List Operations
----------------------
1.Create New List
2.Delete a Node
3.Display List
4.Exit
Enter your choice : 3
Elements in the linked list : 3->9->7->NULL
```

```
Linked List Operations
--------------------
1.Create New List
2.Delete a Node
3.Display List
4.Exit
Enter your choice : 4
```

9. Sort a singly linked list.

ALGORITHM

Global declarations

1. Define node as structure with elements next as pointer to structure node and data as int.
2. Declare start, temp, nw, last as pointers to structure node and i, x, and n as int.

main() function

Step 1. Start.
Step 2. Declare ch as int.
Step 3. Set start=last=null.
Step 4. Read ch.
Step 5. If ch=1, call create(), go to step 4
   i. Else if ch=2, call sort(),go to step 4
   ii. Else if ch=3, call display(),go to step 4
   iii. Else if ch=4, go to step 6
Step 6. Stop.

create() function

Step 1. Declare n, x, i as int.
Step 2. Set start=last=null
Step 3. Read n, number of elements
Step 4. Repeat steps 5 to 8 n times
Step 5. Create a new node and store its address in nw.
Step 6. Read nw->data
Step 7. Set nw->next=NULL
Step 8. If start=NULL then set start=last=nw,
else set last->next=nw and last=nw.

sort() function

Step 1. declare p and q as structure node.
Step 2. if start=NULL, print “list empty” and exit from function. Else go to step 3
Step 3. Set p=start.
Step 4. Repeat the steps 5 to 9 untill p->next not equal to null.
Step 5. Set q=p->next.
Step 6. Repeat the steps 7 and 8 untill q not equal to null.
Step 7. Check if p->data>q->data, then swap p->data and q->data.
Step 8. Set q=q->next
Step 9. Set p=p->next.

Step 10. Stop.

display() function

Step 1. Set temp=start
Step 2. Repeat steps 3 and 4 until temp!=NULL
Step 3. Print temp->data
Step 4. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start, *last, *nw, *temp;
int i,x,n;
void create();
void display();
void sort();
void main()
{
    int ch;
    while(1)
    {
        printf("\nLinked list operations\n--------------------");
        printf("\n1.Create\n2.Sort\n3.Display\n4.Exit");
        printf("\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: create(); break;
            case 2: sort(); break;
            case 3: display(); break;
            case 4: return;
            default: printf("Invalid option..!");
        }
    }
}
void sort()
{
    struct node *p, *q;
    if(start==NULL)
    {
        printf("List is empty..!");
        return;
```

```
    }
    for(p=start;p->next!=NULL;p=p->next)
    {
        for(q=p->next;q!=NULL;q=q->next)
        {
            if(p->data>q->data)
            {
                x=p->data;
                p->data=q->data;
                q->data=x;
            }
        }
    }
    printf("List is sorted..");

}
void create()
{
    start=last=NULL;
    printf("How many elements : ");
    scanf("%d",&n);
    printf("Enter the elements : ");
    for(i=0;i<n;i++)
    {
        nw=(struct node*) malloc(sizeof(struct node));
        scanf("%d",&x);
        nw->data=x;
        nw->next=NULL;
        if(start==NULL)
        {
            start=last=nw;
        }
        else
        {
            last->next=nw;
            last=nw;
        }
    }
    printf("List created..!");
}
void display()
{
    printf("Elements in the linked list : ");
    for(temp=start;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
```

```
    printf("NULL");
}
```

OUTPUT

```
Linked list operations
------------------
1.Create
2.Sort
3.Display
4.Exit
Enter ur choice : 1
How many elements : 6
Enter the elements : 9 4 2 8 1 9
List created..!
Linked list operations
------------------
1.Create
2.Sort
3.Display
4.Exit
Enter ur choice : 2
List is sorted..
Linked list operations
------------------
1.Create
2.Sort
3.Display
4.Exit
Enter ur choice : 3
Elements in the linked list : 1->2->4->8->9->9->NULL
Linked list operations
------------------
1.Create
2.Sort
3.Display
4.Exit
Enter ur choice :4
```

10. Create a singly linked list and search an element from that list.

ALGORITHM

Global declarations

3. Define node as structure with elements next as pointer to structure node and data as int.

4. Declare start, temp, nw, last as pointers to structure node.

main() function

Step 7. Start.
Step 8. Declare ch as int.
Step 9. Set start=last=null.
Step 10. Read ch.
Step 11. If ch=1, call create(), go to step 4
i. Else if ch=2, call delete_node(),go to step 4
ii. Else if ch=3, call display(),go to step 4
iii. Else if ch=4, go to step 6
Step 12. Stop.

create() function

Step 9. Declare n, x, i as int.
Step 10. Set start=last=null
Step 11. Read n, number of elements
Step 12. Repeat steps 5 to 8 n times
Step 13. Create a new node and store its address in nw.
Step 14. Read nw->data
Step 15. Set nw->next=NULL
Step 16. If start=NULL then set start=last=nw,
i. else set last->next=nw and last=nw.

search() function

Step 11. declare p and flag as int, set p=flag=0.
Step 12. if start=NULL, print "list empty" and exit from function. Else go to step 3
Step 13. read x, node to delete.
Step 14. Set temp=start
Step 15. Repeat steps 6 to 7 until temp!=NULL
Step 16. If temp->data = x, then Set flag=1 and print "found".
Step 17. Set temp=temp->next
Step 18. If flag=0 then Print "element not found"

display() function

Step 5. Set temp=start
Step 6. Repeat steps 3 and 4 until temp!=NULL
Step 7. Print temp->data
Step 8. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
```

```c
struct node *start, *last, *nw, *temp;
int i,x,n;
void create();
void display();
void search();
void main()
{
    int ch;
    while(1)
    {
        printf("\nLinked list operations\n--------------------");
        printf("\n1.Create\n2.Search\n3.Display\n4.Exit");
        printf("\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: create(); break;
            case 2: search(); break;
            case 3: display(); break;
            case 4: return;
            default: printf("Invalid option..!");
        }
    }
}
void search()
{
    int p=0,flag=0;
    if(start==NULL)
    {
        printf("List is empty..!");
        return;
    }
    printf("Enter the data to be search : ");
    scanf("%d",&x);
    for(temp=start;temp!=NULL; temp=temp->next)
    {
        p++;
        if(temp->data==x)
        {
            flag=1;
            printf("\t%d is found at position %d..\n",x,p);
        }
    }
    if(flag==0)
        printf("\t%d is not found..!",x);
}
void create()
```

```
{
    start=last=NULL;
    printf("How many elements : ");
    scanf("%d",&n);
    printf("Enter the elements : ");
    for(i=0;i<n;i++)
    {
        nw=(struct node*) malloc(sizeof(struct node));
        scanf("%d",&x);
        nw->data=x;
        nw->next=NULL;
        if(start==NULL)
        {
            start=last=nw;
        }
        else
        {
            last->next=nw;
            last=nw;
        }
    }
    printf("List created..!");
}
void display()
{
    printf("Elements in the linked list : ");
    for(temp=start;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
```

OUTPUT

```
Linked list operations
------------------
1.Create
2.Search
3.Display
4.Exit
Enter ur choice : 1
How many elements : 6
Enter the elements : 3 9 2 1 3 5
List created..!
Linked list operations
------------------
1.Create
```

```
2.Search
3.Display
4.Exit
Enter ur choice : 3
Elements in the linked list : 3->9->2->1->3->5->NULL
Linked list operations
------------------
1.Create
2.Search
3.Display
4.Exit
Enter ur choice : 2
Enter the data to be search : 3
        3 is found at position 1..
        3 is found at position 5..
Linked list operations
------------------
1.Create
2.Search
3.Display
4.Exit
Enter ur choice : 2
Enter the data to be search : 7
        7 is not found..!
Linked list operations
------------------
1.Create
2.Search
3.Display
4.Exit
Enter ur choice : 4
```

11. Create a doubly linked list of integers and display in forward and backward direction.

ALGORITHM

Global declarations

4. Define node as structure with elements next and prev as pointer to structure node and data as int.
5. Declare start, temp, nw, last as pointers to structure node.

main() function

Step 7. Start.
Step 8. Declare ch as int.
Step 9. Set start=last=null.
Step 10. Read ch.
Step 11. If ch=1, set start=last=null and call addnodes(), go to step 4
Else if ch=2, call addnodes(),go to step 4
Else if ch=3, call traverseforward(),go to step 4

Else if ch=4, call traversebackward(),go to step 4

Else if ch=5, go to step 6

Step 12. Stop.

addnodes() function

Step 5. Declare n, x and i as int.

Step 6. Read n, number of nodes to add

Step 7. Repeat steps 4 to 7 n times.

Step 8. Create a new node and store its address in nw.

Step 9. Read nw->data

Step 10. Set nw->next=NULL

Step 11. If start=NULL then set nw->prev=NULL and set start=last=nw,
else set last->next=nw , nw->prev=last and last=nw.

traverseforward() function

Step 5. Set temp=start

Step 6. Repeat steps 3 and 4 until temp!=NULL

Step 7. Print temp->data

Step 8. Set temp=temp->next

traversebackward() function

Step 1. Set temp=last

Step 2. Repeat steps 3 and 4 until temp!=NULL

Step 3. Print temp->data

Step 4. Set temp=temp->prev

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *start, *nw, *temp, *last;
void addnodes();
void traverseforward();
void traversebackward();
void main()
{
int ch;
start=last=NULL;
    while(1)
    {
        printf("\nDoubly Linked List\n-------------------\n");
        printf("1.Create List\n2.Add Nodes\n3.Display in Forward");
        printf("\n4.Display in Backward\n5.Exit");
        printf("\nEnter your choice : ");
```

```
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: start=last=NULL;
                    addnodes();
                    printf("list created..");
            break;
            case 2: addnodes();
                    printf("nodes added..");
            break;
            case 3: traverseforward();
            break;
            case 4: traversebackward();
            break;
            case 5: return;
            default: printf("Invalid choice..");
        }
    }
}
void addnodes()
{
   int n, x, i;
   printf("how many elements : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
    {
        nw = (struct node *) malloc(sizeof(struct node));
        printf("\tEnter the data :");
        scanf("%d",&x);
        nw->data=x;
        nw->next=NULL;
        if(start == NULL)
        {
            nw->prev = NULL;
            start = last = nw;
        }
        else
        {
            last->next= nw;
            nw->prev=last;
            last=nw;
        }
    }
}
void traverseforward()
{
    printf("Elements in the linked list (forward) : ");
```

```
    for(temp=start;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
void traversebackward()
{
    printf("Elements in the linked list (backward) : ");
    for(temp=last;temp!=NULL;temp=temp->prev)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
```

OUTPUT

```
Doubly Linked List
------------------
1.Create List
2.Add Nodes
3.Display in Forward
4.Display in Backward
5.Exit
Enter your choice : 1
how many elements : 6
        Enter the data :9
        Enter the data :3
        Enter the data :8
        Enter the data :1
        Enter the data :4
        Enter the data :8
list created..
Doubly Linked List
------------------
1.Create List
2.Add Nodes
3.Display in Forward
4.Display in Backward
5.Exit
Enter your choice : 3
Elements in the linked list (forward) : 9->3->8->1->4->8->NULL
Doubly Linked List
------------------
1.Create List
2.Add Nodes
```

```
3.Display in Forward
4.Display in Backward
5.Exit
Enter your choice : 2
how many elements : 2
        Enter the data :7
        Enter the data :5
nodes added..
Doubly Linked List
------------------
1.Create List
2.Add Nodes
3.Display in Forward
4.Display in Backward
5.Exit
Enter your choice : 4
Elements in the linked list (backward) : 5->7->8->4->1->8->3->9->NULL
Doubly Linked List
------------------
1.Create List
2.Add Nodes
3.Display in Forward
4.Display in Backward
5.Exit
Enter your choice : 4
```

12. Addition of 2 polynomials using array.

ALGORITHM

main() function

Step 1. start
Step 2. declare p[20], q[20], r[20], m, n, k , i as int.
Step 3. initialize the all elements of the arrays p and q to zero
Step 4. call m=read(p,"first");
Step 5. call n=read(q,"second");
Step 6. call display(p,m,"first");
Step 7. call display(q,n,"second");
Step 8. check if m>n, then set k=m else set k=n
Step 9. Set i=0
Step 10. repeat step 11 and 12 until i<=k
Step 11. Set r[i]=p[i]+q[i]
Step 12. Increment i by one.
Step 13. call display(r,k,"sum of two")
Step 14. Stop

display(int a[], int t, char str[]) function

Step 1. declare i as int
Step 2. set i=t
Step 3. repeat steps 4 and 5 until i>0
Step 4. check if a[i] not equal to zero, then print (a[i] + “x^”+i)
Step 5. decrement i by one.
Step 6. Check if a[0] not equal to zero, then print(a[0] + “=0”) ,
else print (backspace+” =0”)

read(int a[], char str[]) function

Step 1. declare t and i as int.
Step 2. read t, degree of polynomial
Step 3. set i=t
Step 4. repeat steps 5 and 6 until i>=0
Step 5. read a[i], coefficient of x^i
Step 6. decrement i by one.
Step 7. Return t

SOURCE CODE

```
#include<stdio.h>
int read(int a[], char str[]);
void display(int a[],int m, char str[]);
void display(int a[],int t, char str[])
{
    int i;
    printf("\n%s polynomial is : ",str);
    for(i=t;i>0;i--)
    {
        if(a[i]!=0)
        {
            printf(" %dx^%d +",a[i],i);
        }
    }
    if(a[0]!=0)
        printf(" %d = 0",a[0]);
        else
        printf("\b = 0");
}
int read(int a[], char str[])
{
    int t, i;
    printf("Enter the degree of %s polynomial : ",str);
    scanf("%d",&t);
    for(i=t;i>=0;i--)
    {
        printf("\tEnter the coefficient of x^%d : ",i);
        scanf("%d",&a[i]);
    }
```

```
    return t;
}
void main()
{
    int p[20]={0},q[20]={0}, r[20], m, n, k, i;
    m=read(p, "first");
    n=read(q,"second");
    display(p,m,"First");
    display(q,n,"Second");
    k=m>n?m:n;
    for(i=0;i<=k;i++)
    {
        r[i]=p[i]+q[i];
    }
    display(r,k,"Sum of two");
    getch();
}
```

OUTPUT

```
Enter the degree of first polynomial : 3
        Enter the coefficient of x^3 : 3
        Enter the coefficient of x^2 : 0
        Enter the coefficient of x^1 : -2
        Enter the coefficient of x^0 : 9
Enter the degree of second polynomial : 4
        Enter the coefficient of x^4 : 4
        Enter the coefficient of x^3 : 1
        Enter the coefficient of x^2 : 2
        Enter the coefficient of x^1 : 0
        Enter the coefficient of x^0 : 0

First polynomial is :  3x^3 + -2x^1 + 9 = 0
Second polynomial is :  4x^4 + 1x^3 + 2x^2  = 0
Sum of two polynomial is :  4x^4 + 4x^3 + 2x^2 + -2x^1 + 9 = 0
```

13. Implement Stack using array

ALGORITHM

Global declarations

1. Declare stack[100], ch, n, top, x and i as int

main() function

Step 1. Start.

Step 2. Set top=-1

Step 3. Read n, size of stack

Step 4. Read ch.

Step 5. If ch=1, call push(), go to step 4
Else if ch=2, call pop(),go to step 4
Else if ch=3, call display(),go to step 4
Else if ch=4, go to step 6

Step 6. Stop.

push() function

Step 1. check if top=n-1, then print “stack overflow” and return to main(), else go to step 2

Step 2. read x

Step 3. increment top by one.

Step 4. Set stack[top]=x

pop() function

Step 1. check if top=-1, then print “stack underflow” and return to main(), else go to step 2

Step 2. Print stack[top], element to be delete

Step 3. decrement top by one.

display() function

Step 1. check if top=-1, then print “empty stack” and return to main(), else go to step 2

Step 2. Set i=top

Step 3. Repeat the steps 4 and 5 until i>=0

Step 4. Print stack[i]

Step 5. decrement i by one

SOURCE CODE

```
#include<stdio.h>
int stack[100],ch,n,top,x,i;
void push();
void pop();
void display();
void main()
{
    top=-1;
    printf("\n Enter the size of Stack : ");
    scanf("%d",&n);
    while(1)
    {
        printf("\n Stack Operations\n----------------");
        printf("\n 1.Push\n 2.Pop\n 3.Display\n 4.Exit");
        printf("\n Enter the Choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
```

```
            case 3: display();
                    break;
            case 4: return;
            default: printf("\tInvalid choice");
        }
    }
}
void push()
{
    if(top==n-1)
    {
        printf("\tStack Over Flow..!");
    }
    else
    {
        printf("\tEnter a value to be pushed : ");
        scanf("%d",&x);
        top++;
        stack[top]=x;
        printf("\t%d pushed..",x);
    }
}
void pop()
{
    if(top==-1)
    {
        printf("\tStack Under Flow..!");
    }
    else
    {
        printf("\tThe popped element is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top==-1)
    {
        printf("\tStack is empty..");
        return;
    }
    printf("\tThe Stack elements are : ");
    for(i=top; i>=0; i--)
    {
        printf("%d  ",stack[i]);
    }
}
```

OUTPUT

```
 Enter the size of Stack : 4

 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 8
        8 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 4
        4 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 5
        5 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 5
        5 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
```

```
 Enter the Choice : 1
        Stack Over Flow..!
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        The Stack elements are : 5  5  4  8
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        The popped element is 5
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        The Stack elements are : 5  4  8
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        The popped element is 5
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        The Stack elements are : 4  8
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
```

```
 4.Exit
 Enter the Choice : 2
        The popped element is 4
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        The popped element is 8
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        Stack Under Flow..!
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        Stack is empty..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 4
```

14. Implement Stack using linked list

ALGORITHM

Global declarations

1 Define node as structure with elements next as pointer to structure node and data as int.
2 Declare top, temp, nw as pointers to structure node.
3 Declare x as int

main() function

Step 1. Start.
Step 2. Declare ch as int

Step 3. Set top=NULL
Step 4. Read ch.
Step 5. If ch=1, call push(), go to step 4
Else if ch=2, call pop(),go to step 4
Else if ch=3, call display(),go to step 4
Else if ch=4, go to step 6
Step 6. Stop.

push() function

Step 1. Read x, the number to push
Step 2. Create a new node and store its address in nw.
Step 3. nw->data = x
Step 4. Set nw->next=top
Step 5. Set top=nw.

pop() function

Step 1. if top=NULL, print "stack empty" and return to main(). Else go to step 2
Step 2. Set temp=top
Step 3. Set top=top->next
Step 4. Print temp->data
Step 5. Call free(temp), to deallocate memory

display() function

Step 1. Set temp=top
Step 2. Repeat steps 3 and 4 until temp!=NULL
Step 3. Print temp->data
Step 4. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *top, *nw, *temp;
int x;
void push();
void pop();
void display();
void main()
{
    int ch;
    top=NULL;
    while(1)
    {
```

```
        printf("\n Stack Operations\n----------------");
        printf("\n 1.Push\n 2.Pop\n 3.Display\n 4.Exit");
        printf("\n Enter the Choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: return;
            default: printf("\tInvalid choice");
        }
    }
}
void push()
{
    printf("\tEnter a value to be pushed : ");
    scanf("%d",&x);
    nw=(struct node*)malloc(sizeof(struct node));
    nw->data=x;
    nw->next=top;
    top=nw;
    printf("\t%d pushed..",x);
}
void pop()
{
    if(top==NULL)
    {
        printf("\tStack is empty..!");
    }
    else
    {
        temp=top;
        top=top->next;
        printf("\tThe popped element is %d",temp->data);
        free(temp);
    }
}
void display()
{
    printf("\tThe Stack elements are : ");
    for(temp=top;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
```

```
    }
    printf("NULL");
}
```

OUTPUT

```
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 6
        6 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 7
        7 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 1
        Enter a value to be pushed : 3
        3 pushed..
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        The Stack elements are : 3->7->6->NULL
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
```

```
 Enter the Choice : 2
        The popped element is 3
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        The popped element is 7
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 3
        The Stack elements are : 6->NULL
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        The popped element is 6
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 2
        Stack is empty..!
 Stack Operations
----------------
 1.Push
 2.Pop
 3.Display
 4.Exit
 Enter the Choice : 4
```

15. Evaluation of postfix expression.

ALGORITHM

Global declarartions

1. Declare stack[50] and top as int

main() function

Step 1. start
Step 2. declare i, x, a, b and c as int, also str[50] as char
Step 3. set top=-1
Step 4. read str
Step 5. set i=0
Step 6. repeat steps 7 to 12 until str[i] not equal to null string
Step 7. check if str[i] is an alphabet, then go to step 7.1 else go to step 8
step 7.1 read x, value for str[i]
step 7.2 call push(x)
step 7.3 go to step 6
Step 8. check if str[i] is a digit,then go to step 8.1 else go to step 9
step 8.1 store str[i] into x as digit
step 8.2 call push(x)
step 8.3 go to step 6
Step 9. call a=pop()
Step 10. call b=pop()
Step 11. check if str[i]='+' then set c=b+a
else check if str[i]='-' then set c=b-a
else check if str[i]='*' then set c=b*a
else check if str[i]='/' then set c=b/a
else check if str[i]='%' then set c=b%a
else check if str[i]='^' then set c=b^a
Step 12. call push(c)
Step 13. print stack[top], evaluated value of postfix
Step 14. stop

push(int x) function

Step 1. increment top by one
Step 2. set stack[top]=x

pop() function

Step 1. set x=stack[top]
Step 2. decrement top by one
Step 3. return x to main() function

SOURCE CODE

```
#include<stdio.h>
#include<ctype.h>
#include<math.h>
int stack[50], top;
void push(int);
int pop();
void main()
{
    int i,x,a,b,c;
```

```
    char str[50];
    top=-1;
    printf("Enter the postfix expression : ");
    gets(str);
    for(i=0;str[i]!='\0';i++)
    {
        if(isalpha(str[i]))
        {
            printf("  Enter value for %c = ",str[i]);
            scanf("%d",&x);
            push(x);
        }
        else if(isdigit(str[i]))
        {
            x=str[i]-'0';
            push(x);
        }
        else
        {
            a=pop();
            b=pop();
            switch(str[i])
            {
                case '+' : c=b+a;
                           break;
                case '-' : c=b-a;
                           break;
                case '*' : c=b*a;
                           break;
                case '/' : c=b/a;
                           break;
                case '%' : c=b%a;
                           break;
                case '^' : c=pow(b,a);

                           break;
            }
            push(c);
        }
    }
    printf("The result of the postfix evaluation : %d",stack[top]);
    getch();
}
void push(int x)
{
    top++;
    stack[top]=x;
```

```
}
int pop()
{
    int x=stack[top];
    top--;
    return x;
}
```

OUTPUT

```
Enter the postfix expression : abc*+
  Enter value for a = 3
  Enter value for b = 11
  Enter value for c = 10
The result of the postfix evaluation : 113
```

```
Enter the postfix expression : 3425^7-*+
The result of the postfix evaluation : 103
```

16. Implement Queue using array.

ALGORITHM

Global declarations

1. Declare q[50], f, r, n and x as int

main() function

Step 1. Start.
Step 2. Declare ch as int
Step 3. Set f=r=-1
Step 4. Read n, size of queue
Step 5. Read ch.
Step 6. If ch=1, call enqueue(), go to step 5
Else if ch=2, call dequeue(),go to step 5
Else if ch=3, call display(),go to step 5
Else if ch=4, go to step 7
Step 7. Stop.

enqueue() function

Step 1. check if r=n-1, then print “queue overflow” and return to main(), else go to step 2
Step 2. read x
Step 3. Check if f=-1, then set f=r=0
Else set r=r+1
Step 4. Set q[r]=x

dequeue() function

Step 1. check if f=-1, then print “queue underflow” and return to main(), else go to step 2
Step 2. Set x=q[f]

Step 3. Check if f=r, then set f=r=-1
Else set f=f+1

display() function

Step 1. Declare i as int
Step 2. check if f=-1, then print “empty queue” and return to main(), else go to step 3
Step 3. Set i=f
Step 4. Repeat the steps 5 and 6 until i<=r
Step 5. Print q[i]
Step 6. increment i by one.

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
void enqueue();
void dequeue();
void display();
int f, r, n, q[50], x;
void main ()
{
    int ch;
    f=r=-1;
    printf("\n Enter the size of Queue : ");
    scanf("%d",&n);
    while(1)
    {
        printf("\n Queue Operations\n------------------");
        printf("\n 1.Insert\n 2.Delete\n 3.Display\n 4.Exit");
        printf("\n Enter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: enqueue();
                    break;
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            case 4: return;
            default:
            printf("\tInvalid choice");
        }
    }
}
void enqueue()
{
    if(r == n-1)
```

```
    {
        printf("\tQueue Overflow..!");
        return;
    }
    printf("\tEnter the element : ");
    scanf("%d",&x);
    if(f==-1)
    {
        f=r=0;
    }
    else
    {
        r=r+1;
    }
    q[r] = x;
    printf("\t%d inserted..",x);
}
void dequeue()
{

    if (f==-1)
    {
        printf("\tQueue Underflow.!");
        return;
    }
    x = q[f];
    if(f==r)
    {
        f=r=-1;
    }
    else
    {
        f = f + 1;
    }
    printf("\t%d deleted.. ",x);
}
void display()
{
    int i;
    if (f==-1)
    {
        printf("\tQueue is empty..");
        return;
    }
    printf("\tQueue Elements : ");
    for(i=f;i<=r;i++)
    {
```

```
        printf("%d  ",q[i]);
    }
}
```

OUTPUT

```
 Enter the size of Queue : 4

 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 6
        6 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 8
        8 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 2
        2 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 4
        4 inserted..
 Queue Operations
------------------
 1.Insert
```

```
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Queue Overflow..!
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 3
        Queue Elements : 6  8  2  4
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 2
        6 deleted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 2
        8 deleted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 3
        Queue Elements : 2  4
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 4
```

17. Implement Queue using linked list.

ALGORITHM

Global declarations

1. Define node as structure with elements next as pointer to structure node and data as int.
2. Declare f, r, temp, nw as pointers to structure node.
3. Declare x as int

main() function

Step 1. Start.
Step 2. Declare ch as int
Step 3. Set f=r=NULL
Step 4. Read ch.
Step 5. If ch=1, call enqueue(), go to step 4
Else if ch=2, call dequeue(),go to step 4
Else if ch=3, call display(),go to step 4
Else if ch=4, go to step 6
Step 6. Stop.

enqueue() function

Step 1. Read x, the number to insert
Step 2. Create a new node and store its address in nw.
Step 3. nw->data = x
Step 4. Set nw->next=NULL
Step 5. Check if f=NULL, then set f=r=nw
Else set r->next=nw and r=nw

dequeue() function

Step 1. if f=NULL, print "queue empty" and return to main(). Else go to step 2
Step 2. Set temp=f
Step 3. Set f=f->next
Step 4. Print temp->data
Step 5. Call free(temp), to deallocate memory

display() function

Step 1. Set temp=f
Step 2. Repeat steps 3 and 4 until temp!=NULL
Step 3. Print temp->data
Step 4. Set temp=temp->next

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *f, *r, *nw, *temp;
int x;
```

```
void enqueue();
void dequeue();
void display();
void main ()
{
    int ch;
    f=r=NULL;
    while(1)
    {
        printf("\n Queue Operations\n------------------");
        printf("\n 1.Insert\n 2.Delete\n 3.Display\n 4.Exit");
        printf("\n Enter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: enqueue();
                    break;
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            case 4: return;
            default:
            printf("\tInvalid choice");
        }
    }
}
void enqueue()
{
    printf("\tEnter the element : ");
    scanf("%d",&x);
    nw=(struct node*)malloc(sizeof(struct node));
    nw->data=x;
    nw->next=NULL;
    if(f==NULL)
    {
        f=r=nw;
    }
    else
    {
        r->next=nw;
        r=nw;
    }
    printf("\t%d inserted..",x);
}
void dequeue()
{
```

```
    if (f==NULL)
    {
        printf("\tQueue is empty.!");
        return;
    }
    temp=f;
    f=f->next;
    printf("\t%d deleted.. ",temp->data);
    free(temp);
}
void display()
{
    printf("\tQueue Elements : ");
    for(temp=f;temp!=NULL;temp=temp->next)
    {
        printf("%d->",temp->data);
    }
    printf("NULL");
}
```

OUTPUT

```
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 5
        5 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 9
        9 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
```

```
        Enter the element : 7
        7 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 3
        Queue Elements : 5->9->7->NULL
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 2
        5 deleted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 3
        Queue Elements : 9->7->NULL
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 1
        Enter the element : 11
        11 inserted..
 Queue Operations
------------------
 1.Insert
 2.Delete
 3.Display
 4.Exit
 Enter your choice : 3
        Queue Elements : 9->7->11->NULL
 Queue Operations
------------------
 1.Insert
 2.Delete
```

```
3.Display
4.Exit
Enter your choice : 4
```

18. Traverse a binary search tree in preorder

ALGORITHM

Global declarations

1. Define node as structure with elements left and right as pointer to structure node and key as int.
2. Declare root, temp, nw and p as pointers to structure node.

main() function

Step 1. Start.
Step 2. Declare ch, n, x, i as int
Step 3. Read ch.
Step 4. If ch=1, then go to step 4.1 else go to step 5
- 4.1) Set root=NULL
- 4.2) Read n, number of nodes
- 4.3) Repeat steps 4.4 and 4.5 n times
- 4.4) Read x
- 4.5) Call insert(x)
- 4.6) Go to step 3

Step 5. if ch=2, call preorder(root), go to step 3
Else if ch=3, go to step 6
Step 6. Stop

insert(int x) function

Step 1. Create a new node and store its address in nw.
Step 2. nw->key = x
Step 3. Set nw->left=nw->right=NULL
Step 4. Check if root=NULL, then set root=nw and return to main()
Else go to step 5
Step 5. Set temp=root and p=NULL
Step 6. Repeat steps 7 and 8 until temp!=NULL
Step 7. Set p=temp
Step 8. Check if x<temp->key, then set temp=temp->left
Else set temp=temp->right
Step 9. Check if x<p->key, then set p->left=nw
Else set p->right=nw.

preorder(struct node*rt) function

Step 1. check if rt=NULL, then return to main()
else go to step 2
Step 2. print rt->key
Step 3. call preorder(rt->left) recursively
Step 4. call preorder(rt->right) recursively

SOURCE CODE

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int key;
    struct node *left, *right;
};
struct node*root,*nw,*temp,*p;
void insert(int);
void preorder(struct node*);
void insert(int x)
{
    nw = (struct node *) malloc(sizeof(struct node));
    nw->key = x;
    nw->left = NULL;
    nw->right = NULL;
    if(root==NULL)
    {
        root=nw;
        return;
    }
    temp = root;
    p = NULL;
    while (temp != NULL)
    {
        p = temp;
        if (x < temp->key)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if (x < p->key)
        p->left = nw;
    else
        p->right = nw;
}
void preorder(struct node* rt)
{
    if (rt == NULL)
        return;
    else
    {
        printf("%d ",rt->key);
        preorder(rt->left);
        preorder(rt->right);
    }
```

```
}
void main()
{

    int ch,n,x,i;
    while(1)
    {
        printf("\nBST Operations\n1-Create\n2-Pre-order\n3-Exit");
        printf("\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: root=NULL;
                    printf("  How many elements : ");
                    scanf("%d",&n);
                    printf("  Enter elements : ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&x);
                        insert(x);
                    }
                    break;
            case 2: printf("  Pre-order traversal is : ");
                    preorder(root);
                    break;
            case 3: return;
            default: printf("  Invalid option..!");
        }
    }
}
```

OUTPUT

```
BST Operations
1-Create
2-Pre-order
3-Exit
Enter ur choice : 1
  How many elements : 7
  Enter elements : 11 4 9 3 1 2 6

BST Operations
1-Create
2-Pre-order
3-Exit
Enter ur choice : 2
  Pre-order traversal is : 11 4 3 1 2 9 6
BST Operations
```

```
1-Create
2-Pre-order
3-Exit
Enter ur choice : 4
```

19. Traverse a binary search tree in inorder

ALGORITHM

Global declarations

1. Define node as structure with elements left and right as pointer to structure node and key as int.
2. Declare root, temp, nw and p as pointers to structure node.

main() function

Step 1. Start.
Step 2. Declare ch, n, x, i as int
Step 3. Read ch.
Step 4. If ch=1, then go to step 4.1 else go to step 5
- 4.1) Set root=NULL
- 4.2) Read n, number of nodes
- 4.3) Repeat steps 4.4 and 4.5 n times
- 4.4) Read x
- 4.5) Call insert(x)
- 4.6) Go to step 3

Step 5. if ch=2, call inorder(root), go to step 3
Else if ch=3, go to step 6
Step 6. Stop

insert(int x) function

Step 1. Create a new node and store its address in nw.
Step 2. nw->key = x
Step 3. Set nw->left=nw->right=NULL
Step 4. Check if root=NULL, then set root=nw and return to main()
Else go to step 5
Step 5. Set temp=root and p=NULL
Step 6. Repeat steps 7 and 8 until temp!=NULL
Step 7. Set p=temp
Step 8. Check if x<temp->key, then set temp=temp->left
Else set temp=temp->right
Step 9. Check if x<p->key, then set p->left=nw
Else set p->right=nw.

inorder(struct node*rt) function

Step 1. check if rt=NULL, then return to main()
else go to step 2
Step 2. call inorder(rt->left) recursively

Step 3. print rt->key

Step 4. call inorder(rt->right) recursively

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int key;
    struct node *left, *right;
};
struct node*root,*nw,*temp,*p;
void insert(int);
void inorder(struct node*);
void insert(int x)
{
    nw = (struct node *) malloc(sizeof(struct node));
    nw->key = x;
    nw->left = NULL;
    nw->right = NULL;
    if(root==NULL)
    {
        root=nw;
        return;
    }
    temp = root;
    p = NULL;
    while (temp != NULL)
    {
        p = temp;
        if (x < temp->key)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if (x < p->key)
        p->left = nw;
    else
        p->right = nw;
}
void inorder(struct node* rt)
{
    if (rt == NULL)
        return;
    else
    {
        inorder(rt->left);
```

```
        printf("%d ",rt->key);
        inorder(rt->right);
    }
}
void main()
{

    int ch,n,x,i;
    while(1)
    {
        printf("\nBST Operations\n1-Create\n2-Inorder\n3-Exit");
        printf("\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: root=NULL;
                    printf("  How many elements : ");
                    scanf("%d",&n);
                    printf("  Enter elements : ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&x);
                        insert(x);
                    }
                    break;
            case 2: printf("  Inorder traversal is : ");
                    inorder(root);
                    break;
            case 3: return;
            default: printf("  Invalid option..!");
        }
    }
}
```

OUTPUT

```
BST Operations
1-Create
2-Inorder
3-Exit
Enter ur choice : 1
  How many elements : 7
  Enter elements : 11 4 9 3 1 2 6

BST Operations
1-Create
2-Inorder
3-Exit
```

```
Enter ur choice : 2
  Inorder traversal is : 1 2 3 4 6 9 11
BST Operations
1-Create
2-Inorder
3-Exit
Enter ur choice : 4
```

20. Traverse a binary search tree in postorder.

ALGORITHM

Global declarations

1. Define node as structure with elements left and right as pointer to structure node and key as int.
2. Declare root, temp, nw and p as pointers to structure node.

main() function

Step 1. Start.
Step 2. Declare ch, n, x, i as int
Step 3. Read ch.
Step 4. If ch=1, then go to step 4.1 else go to step 5
9Set root=NULL
4.7) Read n, number of nodes
4.8) Repeat steps 4.4 and 4.5 n times
4.9) Read x
4.10) Call insert(x)
4.11) Go to step 3
Step 5. if ch=2, call postorder(root), go to step 3
Else if ch=3, go to step 6
Step 6. Stop

insert(int x) function

Step 1. Create a new node and store its address in nw.
Step 2. nw->key = x
Step 3. Set nw->left=nw->right=NULL
Step 4. Check if root=NULL, then set root=nw and return to main()
Else go to step 5
Step 5. Set temp=root and p=NULL
Step 6. Repeat steps 7 and 8 until temp!=NULL
Step 7. Set p=temp
Step 8. Check if x<temp->key, then set temp=temp->left
Else set temp=temp->right
Step 9. Check if x<p->key, then set p->left=nw
Else set p->right=nw.

postorder(struct node*rt) function

Step 1. check if rt=NULL, then return to main()
else go to step 2
Step 2. call postorder(rt->left) recursively
Step 3. call postorder(rt->right) recursively
Step 4. print rt->key

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int key;
    struct node *left, *right;
};
struct node*root,*nw,*temp,*p;
void insert(int);
void postorder(struct node*);
void insert(int x)
{
    nw = (struct node *) malloc(sizeof(struct node));
    nw->key = x;
    nw->left = NULL;
    nw->right = NULL;
    if(root==NULL)
    {
        root=nw;
        return;
    }
    temp = root;
    p = NULL;
    while (temp != NULL)
    {
        p = temp;
        if (x < temp->key)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if (x < p->key)
        p->left = nw;
    else
        p->right = nw;
}
void postorder(struct node* rt)
{
    if (rt == NULL)
        return;
```

```
    else
    {
        postorder(rt->left);
        postorder(rt->right);
        printf("%d ",rt->key);
    }
}
void main()
{

    int ch,n,x,i;
    while(1)
    {
        printf("\nBST Operations\n1-Create\n2-Post-order\n3-Exit");
        printf("\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: root=NULL;
                    printf("  How many elements : ");
                    scanf("%d",&n);
                    printf("  Enter elements : ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&x);
                        insert(x);
                    }
                    break;
            case 2: printf("  Post-order traversal is : ");
                    postorder(root);
                    break;
            case 3: return;
            default: printf("  Invalid option..!");
        }
    }
}
```

OUTPUT

```
BST Operations
1-Create
2-Post-order
3-Exit
Enter ur choice : 1
  How many elements : 7
  Enter elements : 11 4 9 3 1 2 6

BST Operations
```

```
1-Create
2-Post-order
3-Exit
Enter ur choice : 2
  Post-order traversal is : 2 1 3 6 9 4 11
BST Operations
1-Create
2-Post-order
3-Exit
Enter ur choice : 3
```

21. Search an element in a binary search tree

ALGORITHM

Global declarations

1. Define node as structure with elements left and right as pointer to structure node and key as int.
2. Declare root, temp, nw and p as pointers to structure node.

main() function

Step 1. Start.
Step 2. Declare ch, n, x, i as int
Step 3. Read ch.
Step 4. If ch=1, then go to step 4.1 else go to step 5
- 4.12) Set root=NULL
- 4.13) Read n, number of nodes
- 4.14) Repeat steps 4.4 and 4.5 n times
- 4.15) Read x
- 4.16) Call insert(x)
- 4.17) Go to step 3

Step 5. if ch=2, call search(), go to step 3
Else if ch=3, call inorder(root), go to step 3
Else if ch=4, go to step 6
Step 6. Stop

insert(int x) function

Step 1. Create a new node and store its address in nw.
Step 2. nw->key = x
Step 3. Set nw->left=nw->right=NULL
Step 4. Check if root=NULL, then set root=nw and return to main()
Else go to step 5
Step 5. Set temp=root and p=NULL
Step 6. Repeat steps 7 and 8 until temp!=NULL
Step 7. Set p=temp
Step 8. Check if x<temp->key, then set temp=temp->left
Else set temp=temp->right

Step 9. Check if x<p->key, then set p->left=nw
Else set p->right=nw.

search() function

Step 1. Read x
Step 2. Set temp=root
Step 3. Repeat steps 4 and 5 until temp!=NULL
Step 4. Check if x=temp->key, then print “element found” and return to main()
Else go to step 5
Step 5. Check if x<temp->key, then set temp=temp->left
Else set temp=temp->right.
Step 6. Print “not found”
Step 7. Stop

inorder(struct node*rt) function

Step 1. check if rt=NULL, then return to main()
else go to step 2
Step 2. call inorder(rt->left) recursively
Step 3. print rt->key
Step 4. call inorder(rt->right) recursively

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int key;
    struct node *left, *right;
};
struct node*root,*nw,*temp,*p;
void insert(int);
void search();
void inorder(struct node*);
void insert(int x)
{
    nw = (struct node *) malloc(sizeof(struct node));
    nw->key = x;
    nw->left = NULL;
    nw->right = NULL;
    if(root==NULL)
    {
        root=nw;
        return;
    }
    temp = root;
    p = NULL;
    while (temp != NULL)
    {
```

```
        p = temp;
        if (x < temp->key)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if (x < p->key)
        p->left = nw;
    else
        p->right = nw;
}
void search()
{
    int x;
    printf("  Enter data to be search : ");
    scanf("%d",&x);
    temp = root;
    while (temp != NULL)
    {
        if(x==temp->key)
        {
            printf("  %d is found in BST..",x);
            return;
        }
        if (x < temp->key)
            temp = temp->left;
        else
            temp = temp->right;
    }
    printf("  %d is not found in BST..!",x);
}
void inorder(struct node* rt)
{
    if (rt == NULL)
        return;
    else
    {
        inorder(rt->left);
        printf("%d ",rt->key);
        inorder(rt->right);
    }
}
void main()
{

    int ch,n,x,i;
    while(1)
```

```
    {
        printf("\nBST Operations\n1-Create\n2-Search\n3-Inorder");
        printf("\n4-Exit\nEnter ur choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: root=NULL;
                    printf("  How many elements : ");
                    scanf("%d",&n);
                    printf("  Enter elements : ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&x);
                        insert(x);
                    }
                    break;
            case 2: search();
                    break;
            case 3: printf("  Inorder traversal is : ");
                    inorder(root);
                    break;
            case 4: return;
            default: printf("  Invalid option..!");
        }
    }
}
```

OUTPUT

```
BST Operations
1-Create
2-Search
3-Inorder
4-Exit
Enter ur choice : 1
  How many elements : 6
  Enter elements : 9 6 3 11 5 8

BST Operations
1-Create
2-Search
3-Inorder
4-Exit
Enter ur choice : 2
  Enter data to be search : 3
  3 is found in BST..
BST Operations
```

```
1-Create
2-Search
3-Inorder
4-Exit
Enter ur choice : 2
  Enter data to be search : 7
  7 is not found in BST..!
BST Operations
1-Create
2-Search
3-Inorder
4-Exit
Enter ur choice : 3
  Inorder traversal is : 3 5 6 8 9 11
BST Operations
1-Create
2-Search
3-Inorder
4-Exit
Enter ur choice : 4
```

22. Implement exchange sort

ALGORITHM

Step 1. Start
Step 2. Declare a[50], i, j, min, n and t as int
Step 3. Read n, size of array
Step 4. Set i=0
Step 5. Repeat steps 6 and 7 until i<n
Step 6. Read a[i]
Step 7. Increment i by one.
Step 8. Set i=0
Step 9. Repeat steps 10 to 14 until i<n-1
Step 10. Set j=i+1
Step 11. Repeat steps 12 and 13 until j<n
Step 12. Check if a[j]<a[i] then swap a[j] and a[i]
Step 13. Increment j by one
Step 14. Increment i by one
Step 15. Set i=0
Step 16. Repeat steps 17 and 18 until i<n
Step 17. print a[i]
Step 18. Increment i by one.
Step 19. Stop

SOURCE CODE

```c
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[50], i, j, min, n, t;
    printf("\n Enter the size of the Array : ");
    scanf("%d", &n);
    printf(" Enter Array Elements  : ");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0; i<n-1; i++)
    {
        for(j=i+1; j<n; j++)
        {
            if(a[j]<a[i])
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
        }
    }
    printf(" Sorted Array Elements : ");
    for(i=0; i<n; i++)
    {
        printf("%d  ",a[i]);
    }
    getch();
}
```

OUTPUT

```
Enter the size of the Array : 5
Enter Array Elements  : 9 3 1 6 3
Sorted Array Elements : 1  3  3  6  9
```

23. Implement selection sort.

ALGORITHM

Step 1. Start
Step 2. Declare a[50], i, j, min, n and t as int
Step 3. Read n, size of array
Step 4. Set i=0

Step 5. Repeat steps 6 and 7 until i<n
Step 6. Read a[i]
Step 7. Increment i by one.
Step 8. Set i=0
Step 9. Repeat steps 10 to 16 until i<n-1
Step 10. Set min=i
Step 11. Set j=i+1
Step 12. Repeat steps 13 and 14 until j<n
Step 13. Check if a[j]<a[min] then set min=j
Step 14. Increment j by one
Step 15. Check if min not equal to i, then swap a[i] and a[min]
Step 16. Increment i by one
Step 17. Set i=0
Step 18. Repeat steps 19 and 20 until i<n
Step 19. print a[i]
Step 20. Increment i by one.
Step 21. Stop

SOURCE CODE

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[50], i, j, min, n, t;
    printf("\n Enter the size of the Array : ");
    scanf("%d", &n);
    printf(" Enter Array Elements  : ");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0; i<n-1; i++)
    {
        min=i;
        for(j=i+1; j<n; j++)
        {
            if(a[j]<a[min])
                min=j;
        }
        if(min!=i)
        {
            t=a[i];
            a[i]=a[min];
            a[min]=t;
        }
    }
```

```
    printf(" Sorted Array Elements : ");
    for(i=0; i<n; i++)
    {
        printf("%d  ",a[i]);
    }
    getch();
}
```

OUTPUT

```
 Enter the size of the Array : 5
 Enter Array Elements  : 9 3 1 6 3
 Sorted Array Elements : 1  3  3  6  9
```

24. Implement insertion sort.

ALGORITHM

Step 1. Start
Step 2. Declare a[50], i, j, min, n and key as int
Step 3. Read n, size of array
Step 4. Set i=0
Step 5. Repeat steps 6 and 7 until i<n
Step 6. Read a[i]
Step 7. Increment i by one.
Step 8. Set i=0
Step 9. Repeat steps 10 to 16 until i<n
Step 10. Set key=a[i]
Step 11. Set j=i-1
Step 12. Repeat steps 13 and 14 until key>a[j] and j>=0
Step 13. Set a[j+1]=a[j]
Step 14. decrement j by one
Step 15. Set a[j+1]=key
Step 16. Increment i by one
Step 17. Set i=0
Step 18. Repeat steps 19 and 20 until i<n
Step 19. print a[i]
Step 20. Increment i by one.
Step 21. Stop

SOURCE CODE

```
#include<stdio.h>
void main()
{
    int a[50], i, j, min, n, key;
    printf("\n Enter the size of the Array : ");
    scanf("%d", &n);
```

```
    printf(" Enter Array Elements  : ");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0; i<n; i++)
    {
        key=a[i];
        for(j=i-1; key<a[j] && j>=0; j--)
        {
            a[j+1]=a[j];
        }
        a[j+1]=key;
    }
    printf(" Sorted Array Elements : ");
    for(i=0; i<n; i++)
    {
        printf("%d  ",a[i]);
    }
    getch();
}
```

OUTPUT

```
Enter the size of the Array : 5
Enter Array Elements  : 9 3 1 6 3
Sorted Array Elements : 1  3  3  6  9
```

25. Implement quick sort.

ALGORITHM

SOURCE CODE

```
#include<stdio.h>
void quicksort(int [],int,int);
void main()
{
    int a[15],n,i;
    printf("Enter the size of the Array : ");
    scanf("%d",&n);
    printf("Enter Array Elements  : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    quicksort(a,0,n-1);
```

```
    printf("Sorted Array Elements : ");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
void quicksort(int a[],int l,int h)
{
    int pivot,i,j,temp;
    if(l<h)
    {
        pivot=l;
        i=l;
        j=h;
        while(i<j)
        {
            while(a[i]<=a[pivot]&&i<=h)
            {
                i++;
            }
            while(a[j]>a[pivot]&&j>=l)
            {
                j--;
            }
            if(i<j)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
        temp=a[j];
        a[j]=a[pivot];
        a[pivot]=temp;
        quicksort(a,l,j-1);
        quicksort(a,j+1,h);
    }
}
```

OUTPUT

```
Enter the size of the Array : 6
Enter Array Elements  : 3 8 1 9 3 2
Sorted Array Elements : 1 2 3 3 8 9
```