Data Structures Faculty Lab Manual

I Year II Sem

2020-21

Prepared by

Mr.I.Lakshmi Narayana
Assistant Professor

Mr.B.Sobhan Babu
Assistant Professor

Mr.K.Kranthi Kumar
Assistant Professor

GUDLAVALLERU ENGINEERING COLLEGE
(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521356
DEPARTMENT OF INFORMATION TECHNOLOGY

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadrirao Knowledge Village, Gudlavalleru – 521356

INSTITUTE VISION & MISSION

Institute Vision:

To be a leading institution of engineering education and research, preparing students for leadership in their fields in a caring and challenging learning environment.

Institute Mission:

- To produce quality engineers by providing state-of-the-art engineering education.
- To attract and retain knowledgeable, creative, motivated and highly skilled individuals whose leadership and contributions uphold the college tenets of education, creativity, research and responsible public service.
- To develop faculty and resources to impart and disseminate knowledge and information to students and also to society that will enhance educational level, which in turn, will contribute to social and economic betterment of society.
- To provide an environment that values and encourages knowledge acquisition and academic freedom, making this a preferred institution for knowledge seekers.
- To provide quality assurance.
- To partner and collaborate with industry, government, and R&D institutes to develop new knowledge and sustainable technologies and serve as an engine for facilitating the nation’s economic development.
- To impart personality development skills to students that will help them to succeed and lead.
- To instil in students the attitude, values and vision that will prepare them to lead lives of personal integrity and civic responsibility.
- To promote a campus environment that welcomes and makes students of all races, cultures and civilizations feel at home.
- Putting students face to face with industrial, governmental and societal challenges.

DEPARTMENT VISION & MISSION

VISION

To be a centre of innovation by adopting changes in Information Technology, imparting quality education, research to produce visionary computer professionals and entrepreneurs.

MISSION

- To provide an academic environment in which students are given the essential resources for solving real-world problems and work in multidisciplinary teams.
- To impart value based education and research among students, particularly belonging to rural areas, for their sustained growth in technological aspects and leadership.
- To collaborate with the industry for making the students adoptable to evolving changes in Information Technology and related areas.

PROGRAMME EDUCATIONAL OBJECTIVES(PEOs):-

PEO1: To exhibit analytical skills in modeling and solving computing problems by applying mathematical, scientific and engineering knowledge and to pursue their higher studies.

PEO2: To communicate effectively with multi-disciplinary teams to develop quality software systems with an orientation towards research and development for lifelong learning.

PEO3: To address industry and societal needs for the growth of global economy using emerging technologies by following professional ethics.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

Students will be able to

PSO1: Organize, maintain and protect IT Infrastructural resources.

PSO2: Design and Develop web, mobile, and smart apps based software solutions to the real world problems

Course Objectives

- To implement different searching and sorting algorithms.
- To implement linear and non-linear data structures.

Course Outcomes

Upon successful completion of the course, the students will be able to

- implement searching and sorting
- develop code to simulate the operations on linked lists.
- implement the operations on stacks and queues.
- evaluate postfix expression using stack.
- demonstrate the operations on Binary Search Trees and Graphs.
- determine the use of hashing in implementing dictionaries.

DATA STRUCTURES LAB	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**PSO 1**	**PSO 2**
CO1: implement searching and sorting	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**
CO2: develop code to simulate the operations on linked lists .	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**
CO3: implement the operations on stacks and queues.	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**
CO4: evaluate postfix expression using stack.	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**
CO5: demonstrate the operations on Binary Search Trees and Graphs.	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**
CO6: determine the use of hashing in implementing dictionaries.	**3**	**3**	**3**	**2**	**2**			**2**	**2**	**2**		**2**	**2**	**3**

Index

1. (i) (a) Write C programs that use recursive functions to perform Linear search for a Key value in a given list.

Algorithm:

Algorithm Linearsearch_Recursion(a<array>,n,ele,i)

Input: a is an array with n elements, ele is the element to be search, i is starting index.

Output: position of required element in array, if it is available.

1. found =0
2. if(i < n and found = = 0)
    - a) if(a[i] = =ele)
        - i) print(required element was found at positioni)
        - ii) found =1
    - b) else
        - i) i = i+1
        - ii) Linearsearch_Recursion(a,n,ele,i)
    - c) endif
3. end if
4. if(found !=1)
    - a) print(required element is notavailable)
5. end if

End Linearsearch_Recursion

Program:

```
/* LINEAR SEARCH USING RECUSION */
int linearrec(int [],int,int,int);
void main()
{
        int a[20],n,i,flag=0,ele;
        clrscr();
        printf("Enter number of element to array");
        scanf("%d",&n);
        printf("\n Enter elements to array");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
```

```
	printf("\n Enter element to search");
	scanf("%d",&ele);
	flag=linearrec(a,n,ele,0);
	if(flag==1)
	{
		printf("\n Successful search");
	}
	else
	{
		printf("\n The given element was not found in the array");
	}
	getch();
}
int linearrec(int a[],int n,int ele,int i)
{
	if(i<n)
	{
		if(a[i]==ele)
		{
			printf("\n Element found at %d location",i);
			return 1;
		}
		else
		{
			i=i+1;
			linearrec(a,n,ele,i);
		}
	}
}
```

Output:

Enter number of element to array 5

Enter elements to array

10 2 20 3 11

Enter element to search 2

Element found at 1 location

Successful search

Enter number of element to array 6

Enter elements to array

12 36 14 10 2 6

Enter element to search

42

The given element was not found in the array

1. (i) (b) Write C programs that use non recursive functions to perform Linear search for a Key value in a givenlist.

Algorithm:

Algorithm Linearsearch(a<array>,n,ele)

Input: a is an array with n elements, ele is the element to be search.

Output: position of required element in array, if it is available.

1. i = 0
2. found =0
3. while(i < n and found = =0)
   a) if(a[i] = =ele)
      i) print(required element was found at positioni)
      ii) found=1
   b) else
      i) i = i+1
   c) endif
4. end if
5. if(found!=1)
   a) print(required element is notavailable)
6. end if

End Linearsearch

Program:

```
/* LINEAR SEARCH USING NON RECUSION */
int linear(int [],int,int);
void main()
{
    int a[20],n,i,flag=0,ele;
    clrscr();
    printf("Enter number of element to array");
    scanf("%d",&n);
    printf("\n Enter elements to array");
    for(i=0;i<n;i++)
    {
```

```
                scanf("%d",&a[i]);
        }
        printf("\n Enter element to search");
        scanf("%d",&ele);
        flag=linear(a,n,ele);
        if(flag!=0)
        {
                printf("\n Successful search");
        }
        else
        {
                printf("\n The given element was not found in the array");
        }
        getch();
}

int linear(int a[],int n,int ele)
{
        int i;
        for(i=0;i<n;i++)
        {
                if(a[i]==ele)
                {
                        printf("\n Element found at %d location",i);
                        return 1;
                }
        }
        return 0;
}
```

Output:

Enter number of element to array 6

Enter elements to array

15 22 10 3 4 6

Enter element to search 10

Element found at 2 location

Successful search

Enter number of element to array 5

Enter elements to array

1 6 4 3 7

Enter element to search 10

The given element was not found in the array

1. (ii) (a) Write C programs that use recursive functions to perform Binary search for a Key value in a givenlist.

<u>Algorithm:</u>

Algorithm Binarysearch_Recursion(a<array>,ele,low,high)

Input: a is array with n elements, ele is the element to be search, low is starting index, high is endingindex.

Output: position of required element in array, if it is available.

1. found=0.
2. if(low<=high)
    - a) mid=(low+high)/2
    - b) if(ele==a[mid])
        - i) print(required element was found at midposition)
        - ii) found=1
    - c) elseif(ele<a[mid])
        - i) Binarysearch_Recursion(a,ele,low,mid-1)
    - d) elseif(ele>a[mid])
        - i) Binarysearch_Recursion(a,ele,mid+1,high)
    - e) end if
3. end if
4. if(found!=1)
    - a) print(required element is notavailable)
5. end if

End Binarysearch_Recursion

<u>Program:</u>

```
/* BINARY SEARCH USING RECUSION */
int binaryrec(int [],int,int,int,int);
void main()
{
	int a[20],n,i,flag=0,ele;
	clrscr();
	printf("Enter number of element to array");
	scanf("%d",&n);
```

```
	printf("\n Enter elements to array");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("\n Enter element to search");
	scanf("%d",&ele);
	flag=binaryrec(a,n,ele,0,n-1);
	if(flag==1)
	{
		printf("\n Successful search");
	}
	else
	{
		printf("\n The given element was not found in the array");
	}
	getch();
}

int binaryrec(int a[],int n,int ele,int first,int last)
{
	int mid;
	if(first<=last)
	{
		mid=(first+last)/2;
		if(ele==a[mid])
		{
			printf("\n The given element was found at %d location",mid);
			return 1;
		}
		 else if(ele<a[mid])
		{
			binaryrec(a,n,ele,first,mid-1);
		}
```

```
		else if(ele>a[mid])
		{
			binaryrec(a,n,ele,mid+1,last);
		}
	}
}
```

Output:

Enter number of element to array 5

Enter elements to array

22 33 44 55 66

Enter element to search

33

The given element was found at 1 location

Successful search

Enter number of element to array 6

Enter elements toarray

10 20 112 123 145 368

Enter element to search 23

The given element was not found in the array

1. (ii) (b) Write C programs that non recursive functions to perform Binary search for a Key value in a givenlist.

Algorithm:

Algorithm Binarysearch (a<array>, ele)

Input: a is array with n elements, ele is the element to be search.

Output: position of required element in array, if it is available.

1. found=0.
2.while(low<=high)
   - a) mid=(low+high)/2
   - b) if(ele==a[mid])
     - i) print(required element was found at midposition)
     - ii) found=1
   - c) elseif(ele<a[mid])
     - i) high = mid -1
   - d) elseif(ele>a[mid])
     - i) low = mid +1
   - e) end if
3. end if
4. if(found !=1)
   - a) print(required element is notavailable)
5. end if

End Binarysearch

Program:

```
/* BINARY SEARCH USING NON RECUSION */
int binarysearch(int [],int,int);
void main()
{
        int a[20],n,i,flag=0,ele;
        clrscr();
        printf("Enter number of element to array");
        scanf("%d",&n);
        printf("\n Enter elements to array");
```

```
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("\n Enter element to search");
	scanf("%d",&ele);
	flag=binarysearch(a,n,ele);
	if(flag!=0)
	{
		printf("\n Successful search");
	}
	else
	{
		printf("\n The given element was not found in the array");
	}
	getch();
}

int binarysearch(int a[],int n,int ele)
{
	int first,last,mid;
	first=0;
	last=n-1;
	while(first<=last)
	{
		mid=(first+last)/2;
		if(ele==a[mid])
		{
			printf("\n The given element was found at %d location",mid);
			return 1;
		}
		elseif(ele<a[mid])
		{
			last=mid-1;
```

```
        }
        else if(ele>a[mid])
        {
            first=mid+1;
        }
    }
    return 0;
}
```

Output:

```
Enter number of element to array 5
Enter elements to array
12 15 17 18 25
Enter element to search18
The given element was found at 3 location
Successful search


Enter number of element to array 4
Enter elements to array
1 2 10 15
Enter element to search 22
The given element was not found in the array
```

2. (i) Write C program to sort a given list of integers in ascending order using Bubble sort.

Algorithm:

Algorithm bubblesort(a<array>, n)

Input: a is array with n elements to be sort.

Output: array elements in ascending order.

```
1. i = 0
2. while( i <n)
      a) j =0
      b) while ( j <n-i-1)
            i) if ( a[j] > a[j+1] )
                  A) t =a[j]
                  B) a[j] = a[j+1]
                  C) a[j+1] = t
            ii) endif
            iii) j = j+1
      c) endloop
      d) i = i +1
3. end loop
```

End bubblesort

Program:

```
/* BUBBLE SORT */
#include<stdio.h>
void bubblesort(int [],int);
void main()
{
        int a[20],n,i;
        clrscr();
        printf("Enter number of element to array");
        scanf("%d",&n);
        printf("\n Enter elements to array");
        for(i=0;i<n;i++)
```

```
	{
		scanf("%d",&a[i]);
	}
	bubblesort(a,n);
	printf("\n After sorting\n");
	for(i=0;i<n;i++)
	{
		printf("\t %d",a[i]);
	}
	getch();
}

void bubblesort(int a[],int n)
{
	int i,j,t;
	for(i=0;i<n;i++)
	{
		for(j=0;j<n-i-1;j++)
		{
			if(a[j]>a[j+1])
			{
				t=a[j];
				a[j]=a[j+1];
				a[j+1]=t;
			}
		}
	}
}
```

Output:

Enter number of element to array 5

Enter elements to array

12 10 11 13 9

After sorting

9 10 11 12 13

Enter number of element to array 4

Enter elements to array

-9 -3 0 14

After sorting

-9 -3 0 14

2. (ii) Write C program to sort a given list of integers in ascending order using Insertion sort.

Algorithm:

Algorithm insertionsort(a<array>, n)

Input: a is array with n elements to be sort.

Output: array elements in ascending order.

1. i = 1
2. while(i <n)
    a) x = a[i]
    b) j = i-1
    c) while(j >= 0 and a[j] > x)
        i) a[j] =a[j+1]
        ii) j = j -1
    d) end loop
    e) a[j+1] =x
    f) i = i +1
3. end loop

End insertionsort

Program:

```
/* SORT THE GIVEN ELEMENTS USING INSERTION SORT */
void insertion(int [],int);
void main()
{
        int a[2],n,i;
        clrscr();
        printf("\n Enter no.of elements to array");
        scanf("%d",&n);
         printf("\n Enter elements to array");
         for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("\n After sorting");
```

```
        insertion(a,n);
        for(i=0;i<n;i++)
        {
                printf("\t %d",a[i]);
        }
        getch();
}

void insertion(int a[],int n)
{
        int i,j,x;
        for(i=1;i<n;i++)
        {
                x=a[i];
                 j=i-1;
                while(j>=0 && a[j]>x)
                {
                        a[j+1]=a[j];
                        j=j-1;
                }
                a[j+1]=x;
        }
}
```

Output:

```
Enter no.of elements to array 5
 Enter elements to array
12 10 4 6 9
 Aftersorting 4     6     9     10     12

 Enter no.of elements to array 4
 Enter elements to array-3 -1 0-6
 Aftersorting -6     -3     -1     0
```

2. (iii) Write C program to sort a given list of integers in ascending order using Selection sort.

Algorithm:

Algorithm selectionsort (a<array>, n)

Input: a is array with n elements to be sort.

Output: array elements in ascending order.

```
    1. i = 0
    2. while( i <n)
        a) pos =i
        b) j =i+1
        c) while ( j <n)
            i) if ( a[j] < a[pos])
                A) pos =j
            ii) endif
        d) end loop
        e) t =a[pos]
        f) a[pos] = a[i]
        g) a[i] =t
        h) i=i+1
    3. end loop
```

End selectionsort

Program:

```
/* SELECTION SORT */
#include<stdio.h>
void selectionsort(int [],int);
void main()
{
    int a[20],n,i;
    clrscr();
    printf("Enter number of element to array");
    scanf("%d",&n);
    printf("\n Enter elements to array");
```

```
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	selectionsort(a,n);
	printf("\n After sorting\n");
	for(i=0;i<n;i++)
	{
		printf("\t %d",a[i]);
	}
	getch();
}

void selectionsort(int a[],int n)
{
	int i,j,pos,t;
	for(i=0;i<n;i++)
	{
		pos=i;
		for(j=i+1;j<n;j++)
		{
			if(a[j]<a[pos])
			{
				pos=j;
			}
		}
		t=a[pos];
		a[pos]=a[i];
		a[i]=t;
	}
}
```

Output:

Enter number of element to array 6

Enter elements to array

2 1 9 3 4 0

After sorting

0 1 2 3 4 9

Enter number of element to array 4

Enter elements to array-6 -4 2 1

After sorting

-6 -4 1 2

3 (i) Write a C program that uses functions to create a singly linked list.

Algorithm:

Algorithm SLL_Create(header,x)

Input: header is a header node, x is data part of new node to be insert.

Output: SLL with new node inserted at beginning.

1. new = getnewnode()
2. if(new = =NULL)
    a) print(required node was not available in memory, so unable toprocess)
3. else
    a) new.link =header.link /* 1 */
    b) header.link =new /* 2 */
    c) new.data =x
4. end if

End SLL_Create

Program:

```
/*Creation of a single linked list*/
#include<stdio.h>
#include<alloc.h>
void  creation();
void traversal();
struct node
{
	int data;
	struct node *link;
}*ptr,*header,*new;
void main()
{
	int ch;
	clrscr();
	header =NULL;
	while(1)
	{
		printf("\n Enter the choice of operation 1.creation 2.traversal:");
```

```
                scanf("%d",&ch);
                switch(ch)
                {
                        case1:creation();
                                break;
                        case2:traversal();
                                break;
                        default:exit(0);
                }
        }
}
void creation()
{
        int item,x,key,pos;
        printf("enter the data value to insert");
        scanf("%d",&x);
        new=malloc(sizeof(struct node));
        new->link= header->link;
        header->link=new;
        new->data=x;
}
void traversal()
{
        printf("\nelements in the list are");
        ptr=header;
        while(ptr->link!=NULL)
        {
                ptr=ptr->link;
                printf("\t%d",ptr->data);
        }
}
```

Output:

enter the choice of operation 1.Creation 2.traversal: 1

enter the data value to insert 10

enter the choice of operation 1.Creation2.traversal: 2

elements in thelist are 10

enter the choice of operation 1.Creation 2.traversal: 1

enter the data value to insert 20

enter the choice of operation 1.Creation 2.traversal: 1

enter the data value to insert 30

enter the choice of operation 1.Creation2.traversal: 2

elements in thelistare 30 20 10

enter the choice of operation 1.Creation 2.traversal: 1

enter the data value to insert 40

enter the choice of operation 1.Creation 2.traversal: 2

elements in thelistare 40 30 20 10

enter the choice of operation 1.Creation 2.traversal:0

3. (ii) Write a C program that uses functions to insert an element into a singly linked list

Algorithm:

Algorithm SLL_Insert_Begin(header,x)

Input: header is a header node, x is data part of new node to be insert.

Output: SLL with new node inserted at beginning.

1. new = getnewnode()
2. if(new = =NULL)
   a) print(required node was not available in memory, so unable toprocess)
3. else
   a) new.link =header.link /* 1 */
   b) header.link =new /* 2 */
   c) new.data =x
4. end if

End SLL_Insert_Begin

Algorithm SLL_Insert_Ending(header,x)

Input: header is header node,x is data part of new node to be insert.

Output: SLL with new node at ending.

1. new =getnewnode()
2. if(new ==NULL)
   a) print(Required node was not available in memory bank, so unable to process)
3. else
   a) ptr=header
   b) while(ptr.link!=NULL)
      i) ptr = ptr.link go tostep(b)
   c) end loop
   d) ptr.link=new /* 1 */
   e) new.link =NULL /* 2 */
   f) new.data =x
4. end if

End_SLL_Insert_Ending

Algorithm SLL_Insert_ANY(header,x,key)

Input: header is header node,x is data part of new node to be inserting, key is the data part of a node, after this node we want to insert new node.

Output: SLL with new node at ANY

1. new =getnewnode()
2. if(new ==NULL)
    a) print(required node was not available in memory bank,so unable toprocess)
3. else
    a) ptr =header
    b) while(ptr.data != key and ptr.link !=NULL)
        i) ptr =ptr.link
    c) end loop
    d) if(ptr.link == NULL and ptr.data !=key)
        i) print(required node with data part as key value is not available, so unable toprocess)
    e) else
        ii) new.link =ptr.link /* 1 */
        iii) ptr.link=new /* 2 */
        iv) new.data = x
    f) end if
4. end if

End SLL_insert_ANY

Program:

```
/*insertion of a single linked list*/
#include<stdio.h>
#include<alloc.h>
void insertion();
void traversal();
struct node
{
	int data;
	struct node *link;
}*ptr,*header,*new;
```

```
void main()
{
        int ch;
        clrscr();
        header =NULL;
        while(1)
        {
                printf("\nenter the choice of operation 1.insert 2.traversal: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case1:insertion();
                                break;
                        case2:traversal();
                                break;
                        default:exit(0);
                }
        }
}
void insertion()
{
        int item,x,key,pos;
        printf("enter the data value to insert");
        scanf("%d",&x);
        new=malloc(sizeof(struct node));
        printf("enter the position for insertion 1.begining 2.ending 3.At any position");
        scanf("%d",&pos);
        /* insertion at beginning*/
        if(pos==1)
        {
                new->link= header->link;
                header->link=new;
                new->data=x;
        }
```

```
    /* insertion at ending*/
    else if(pos==2)
    {
            ptr=header;
            while(ptr->link!=NULL)
            {
                    ptr=ptr->link;
            }
            ptr->link=new;
            new->link=NULL;
            new->data=x;
    }
/* insertion at any pos*/
else if(pos==3)
{
            printf("\nenter key value");
            scanf("%d",&key);
            ptr=header;
            while(ptr->link!=NULL && ptr->data!=key)
            {
                    ptr=ptr->link;
            }
            if(ptr->link==NULL)
            {
             /* Special case i.e. insertion of a node at any position that leads to insertion at end*/
                    if(ptr->data==key)
                    {
                            new->link=ptr->link;
                            ptr->link=new;
                            new->data=x;
                    }
                    else
                    {
                            printf("\n Key not available");
```

```
                        }
                }
                else
                {
                        new->link=ptr->link;
                        ptr->link=new;
                        new->data=x;
                }
        }
}
void traversal()
{
        printf("\n elements in the list are");
        ptr=header;
        while(ptr->link!=NULL)
        {
                ptr=ptr->link;
                printf("\t%d",ptr->data);
        }
}
```

Output:

```
enter the choice of operation 1.insert 2.traversal: 1
enter the data value to insert 10
enter the position for insertion 1.begining 2.ending 3.At any position1
enter the choice of operation 1.insert 2.traversal:2
elements in thelist are         10
enter the choice of operation 1.insert 2.traversal: 1
enter the data value to insert 20
enter the position for insertion 1.begining 2.ending 3.At any position2
enter the choice of operation 1.insert 2.traversal:2
elements in thelistare          10      20
enter the choice of operation 1.insert 2.traversal: 1
enter the data value to insert 30
```

enter the position for insertion 1.begining 2.ending 3.At any position 1

enter the choice of operation 1.insert 2.traversal:2

elements in the list are 30 10 20

enter the choice of operation 1.insert 2.traversal: 1

enter the choice of operation 1.insert 2.traversal: 1

enter the data value to insert 40

enter the position for insertion 1.begining 2.ending 3.At any position 3

enter key value 10

enter the choice of operation 1.insert 2.traversal: 2

elements in thelistare 30 10 40 20

enter the choice of operation 1.insert2.traversal:0

3. (iii) Write a C program that uses functions to delete an element from a singly linked list

<u>Algorithm:</u>

Algorithm SLL_Delete_Begin(header)

Input: Header is a header node.

Output: SLL with node deleted at Beginning.

1. if(header.link ==NULL)
   a) print(SLL is empty, so unable to delete node fromlist)
2. else /* SLL is notempty*/
   a) ptr=header.link /* ptr points to first node into list*/
   b) header.link =ptr.link /* 1 */
   c) return(ptr) /* send back deleted node to memorybank*/
3. end if

End SLL_Delete_Begin

Algorithm SLL_ Delete_End (header)

Input: header is a header node

Output: SLL with node deleted at ending.

1. if(header.link = =NULL)
   a) print(SLL is empty, so unable to delete the node fromlist)
2. else /*SLL is notempty*/
   a) ptr=header /*ptr initially points to headernode*/
   b) while(ptr.link != NULL)
      i)ptr1=ptr
      ii)ptr=ptr.link /*go to step b*/
   c) end loop
   d) ptr1.link=NULL /* 1 */
   e) return(ptr)
3. end if

End SLL_ Delete_ End

Algorithm SLL_ Delete_ ANY (header,key)

Input: header is a header node, key is the data part of the node to be delete.

Output: SLL with node deleted at Any position. i.e. Required element.

1. if(header.link = =NULL)
    a) print(SLL is empty, so unable to delete the node fromlist)
2. else /*SLL is notempty*/
    a) ptr=header /*ptr initially points to headernode*/
    b) while(ptr.link != NULL and ptr.data !=key)
        i) ptr1 =ptr
        ii) ptr=ptr.link go to stepb
    c) end loop
    d) if(ptr.link = = NULL and ptr.data !=key)
        i) print(Required node with data part as key value is notavailable)
    e) else /* node with data part as key value available*/
        i) ptr1.link= ptr.link /* 1 */
        ii) return(ptr)
    f) end if
3. end if

End SLL_ Delete_ ANY

Program:

```
/* SLL DELETION */
#include<stdio.h>
#include<malloc.h>
void create();
void delete();
void traverse();
struct node
{
	int data;
	struct node *link;
}*header,*new,*ptr,*ptr1;
void main()
{
	int ch;
	clrscr();
	header=NULL;
```

```
    while(1)
    {
        printf("\n\nEnter the choice of operation");
        printf("\n1.Create \t 2.Delete \t 3. Traversal\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:create();
                break;
            case 2:delete();
                break;
            case 3: traverse();
                break;
            default:exit(0);
        }
    }
    getch();
}
void create()
{
    int x;
    new=malloc(sizeof(struct node));
    printf("\nEnter the data value");
    scanf("%d",&x);
    if(header->link==NULL)
    {
        header->link=new;
        new->link=NULL;
        new->data=x;
    }
    else
    {
        ptr=header;
        while(ptr->link!=NULL)
```

```
            {
                ptr=ptr->link;
            }
            ptr->link=new;
            new->link=NULL;
            new->data=x;
        }
}
void delete()
{
        int pos,x,key;
        printf("\nEnter the position for deletion");
        printf("\n 1.Begining 2.Ending\t3.At any Position\n");
        scanf("%d",&pos);
        if(pos==1) /*Deletion at beginning*/
        {
            ptr=header;
            if(ptr->link==NULL)
            {
                printf("\n SLL is empty");
            }
            else
            {
                ptr1=ptr;
                ptr=ptr->link;
                ptr1->link=ptr->link; /* Address of second node i.e link part of first node is copied to link part of header node*/
                printf("\nNode deleted is %d",ptr->data);
                free(ptr);
            }
        }
        else if (pos==2) /* Deletion at Ending */
        {
            ptr=header;
            if(ptr->link==NULL)
```

```
        {
            printf("\n SLL is empty, unable to perform deletion");
        }
        else
        {
            ptr1=ptr;
            ptr=ptr->link;
            while(ptr->link!=NULL)
            {
                ptr1=ptr;
                ptr=ptr->link;
            }
            ptr1->link=NULL;    /* Last but one node link part is replaced with NULL */
            printf("\nDeleted Node is%d",ptr->data);
            free(ptr);
        }
    }
    elseif(pos==3)                          /* Deletion at any position*/
    {
        ptr=header;
        if(ptr->link==NULL)
        {
            printf("\n SLL is empty, unable to perform deletion");
        }
        else
        {
            printf("\nEnter the data value to delete");
            scanf("%d",&key);
            while(ptr->data!=key && ptr->link!=NULL)
            {
                ptr1=ptr;
                ptr=ptr->link;          /* Move to the nextnode*/
            }
```

```
                if(ptr->link==NULL)
                {
                        printf("\n Node with key was not found");
                }
                else
                {
                        ptr1->link=ptr->link;
                        printf("\nDeleted node is %d",ptr->data);
                        free(ptr);
                }
            }
        }
}

void traverse()
{
        printf("\n Elements in the list are:\n");
        ptr=header;
        while(ptr->link!=NULL)
        {
                ptr=ptr->link;
                printf("\t%d",ptr->data);
        }
}
```

Output:

```
Enter the choice of operation
1.Create        2.Delete        3. Traversal
1
Enter the data value10

Enter the choice of operation
1.Create        2.Delete        3. Traversal
1
```

Enter the data value 20

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 30

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 40

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

10 20 30 40 50 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

1

Node deleted is 10

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 30 40 50 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

2

Deleted Node is 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 30 40 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

3

Enter the data value to delete 30

Deleted node is 30

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 40 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

3

Enter the data value to delete65

Node with key was not found

Enter the choice of operation

1.Create 2.Delete 3. Traversal

0

4. (i) Write a C program that uses functions to Create a Doubly linked list.

Algorithm:

Algorithm DLL_Creation(header,X)

Input: header is a header node.

Output: DLL with new node at begin.

1.new = getnewnode()

2.if(new = = NULL)

a) print(required node is not available inmemory)

3.else

a) ptr =header.rlink

b) new.rlink=ptr /* 1 */

c) new.llink=header /* 2*/

d) header.rlink =new /* 3 */

e) ptr.llink=new /* 4 */

4. end if

End DLL_Creation

Program:

```
/* Creation of double linked list*/
#include<stdio.h>
#include<alloc.h>
void  creation();
void traversal();
struct node
{
        int data;
        struct node *llink;
        struct node *rlink;
}*ptr,*ptr1,*header,*new1;
void main()
{
        int ch;
        clrscr();
        header->llink=NULL;
```

```
	header->rlink=NULL;
	header->data=NULL;
	while(1)
	{
		printf("\n Enter the choice of operation 1.creation 2.traversal: ");
		scanf("%d",&ch);
		switch(ch)
		{
			case1:creation();
				break;
			case2:traversal();
				break;
			default:exit(0);
		}
	}
}
void creation()
{
	int item,x,key,pos;
	printf("enter the data value to insert");
	scanf("%d",&x);
	new1=malloc(sizeof(struct node));
	ptr=header;
	ptr1=ptr->rlink;
	new1->llink=header;
	new1->rlink= ptr1;
	ptr->rlink=new1;
	ptr1->llink=new1;
	new1->data=x;
}
void traversal()
{
	printf("\nelements in the list are");
	ptr=header;
```

```
	while(ptr->rlink!=NULL)
	{
		ptr=ptr->rlink;
		printf("\t%d",ptr->data);
	}
}
```

Output:

```
enter the choice of operation 1.creation 2.traversal: 1
enter the data value to insert10

enter the choice of operation 1.creation 2.traversal: 1
enter the data value to insert20

enter the choice of operation 1.creation 2.traversal: 1
enter the data value to insert30

enter the choice of operation 1.creation 2.traversal: 1
enter the data value to insert40

enter the choice of operation 1.creation 2.traversal:2

elements in thelistare        40      30      20      10
enter the choice of operation 1.creation 2.traversal:0
```

4. (ii) Write a C program that uses functions to Insert an element into a Doubly linked list.

<u>Algorithm:</u>

Algorithm DLL_insertion_Begin(header,X)

Input: header is a header node.

Output: DLL with new node at begin.

1.new = getnewnode()

2.if(new = = NULL)

a) print(required node is not available inmemory)

3.else

a) ptr =header.rlink

b) new.rlink=ptr /* 1 */

c) new.llink=header /* 2*/

d) header.rlink =new /* 3 */

e) ptr.llink=new /* 4 */

4. end if

End DLL_insertion_Begin

Algorithm DLL_Insert_Ending(Header,x)

Input: Header is the header node, x is the data part of new node to be inserted.

Output: DLL with new node inserted at the ending.

1.new = getnewnode()

2.if(new = = NULL)

a) print(Required node was not available)

3.else

a) ptr=header

b) while(ptr.rlink !=NULL)

i) ptr=ptr.rlink gotostep(b)

c) end whileloop

d) ptr.rlink=new /* 1 */

e) new.llink=ptr /* 2 */

f) new.rlink =NULL /* 3 */

g) new.data = x

4.end if

End DLL_Insertion_Ending

Algorithm DLL_Insertion_ANY(header,x,key)

Input: Header is a header node, key is the data part of a node, after that node new node is inserted, x is data part of new node to be insert.

Output: DLL with new node inserted after the node with data part as key value

1. new =getnewnode()
2. if(new = =NULL)
    a) print (required node is not available inmemory)
3. else
    a) ptr=header
    b) while(ptr.data != key and ptr.rlink != NULL)
        i)ptr=ptr.rlinkgo to step(b)
    c) end loop
    d) if(ptr.rlink = =NULL and ptr.data !=key)
        i) print(required node with key value was notavailable)
    e) else
        i) ptr1=ptr.rlink
        ii) new.rlink=ptr1 /* 1 */
        iii) new.llink=ptr /* 2 */
        iv) ptr.rlink=new /* 3 */
        v) ptr1.llink=new /* 4 */
        vi) new.data=x
    f) end if
4. endif

EndDLL_Insertion_ANY

Program:

```
/*insertion of a node into double linked list*/
#include<stdio.h>
#include<alloc.h>
void insertion();
void traversal();
struct node
{
	int data;
```

```
        struct node *llink;
        struct node *rlink;
}*ptr,*ptr1,*header,*new1;

void main()
{
        int    ch;
        clrscr();
        header->llink=NULL;
        header->rlink=NULL;
        header->data=NULL;
        while(1)
        {
                printf("\nenter the choice of operation 1.insert 2.traversal: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case1:insertion();
                                break;
                        case2:traversal();
                                break;
                        default:exit(0);
                }
        }
}
void insertion()
{
        int item,x,key,pos;
        printf("enter the data value to insert");
        scanf("%d",&x);
        new1=malloc(sizeof(struct node));
        printf("enter the position for insertion 1.begining 2.ending 3.At any position");
        scanf("%d",&pos);
        /* insertion at beginning*/
```

```
if(pos==1)
{
        ptr=header;
        ptr1=ptr->rlink;
        new1->llink=header;
        new1->rlink= ptr1;
        ptr->rlink=new1;
        ptr1->llink=new1;
        new1->data=x;
}
/* insertion at ending*/
else if(pos==2)
{
        ptr=header;
        while(ptr->rlink!=NULL)
        {
                ptr=ptr->rlink;
        }
        ptr->rlink=new1;
        new1->llink=ptr;
        new1->rlink=NULL;
        new1->data=x;
}
/* insertion at any pos*/
else if(pos==3)
{
        printf("\nenrter key value");
        scanf("%d",&key);
        ptr=header;
        while(ptr->rlink!=NULL && ptr->data!=key)
        {
                ptr=ptr->rlink;
        }
        if(ptr->rlink==NULL)
```

```
			{
				printf("\n Key not available");
			}
			else
			{
				ptr1=ptr->rlink;
				new1->llink=ptr;
				new1->rlink=ptr1;
				ptr->rlink=new1;
				ptr1->llink=new1;
				new1->data=x;
			}
		}
}
void traversal()
{
		printf("\nelements in the list are");
		ptr=header;
		while(ptr->rlink!=NULL)
		{
			ptr=ptr->rlink;
			printf("\t%d",ptr->data);
		}
}
```

Output:

enter the choice of operation 1.insert 2.traversal: 1

enter the data value to insert 10

enter the position for insertion 1.begining 2.ending 3.At any position1

enter the choice of operation 1.insert 2.traversal: 1

enter the data value to insert 20

enter the position for insertion 1.begining 2.ending 3.At any position1

enter the choice of operation 1.insert 2.traversal:2

elements in thelistare 20 10

enter the choice of operation 1.insert 2.traversal: 1
enter the data value to insert 30
enter the position for insertion 1.begining 2.ending 3.At any position2

enter the choice of operation 1.insert 2.traversal: 2
elements in thelistare 20 10 30

enter the choice of operation 1.insert 2.traversal: 1
enter the data value to insert 55
enter the position for insertion 1.begining 2.ending 3.At any position3
enter key value10

enter the choice of operation 1.insert 2.traversal: 2
elements in thelistare 20 10 55 30

enter the choice of operation 1.insert 2.traversal:0

4. (iii) Write a C program that uses functions to Delete an element from a Doubly linked list.

Algorithm:

Algorithm DLL_Deletion_Begin(header)

Input: header is a header node

Output: DLL with node deleted at begin

1. if(header.rlink = =NULL)
    a) print(DLL is empty, not possible to perform deletionoperation)
2. else
    a) ptr =header.rlink
    b) ptr1 =ptr.rlink
    c) header.rlink =ptr1 /* 1 */
    d) ptr1.llink =header /* 2 */
    e) return(ptr)
3. endif

EndDLL_Deletion_Begin

Algorithm DLL_Deletion_End(header)

Input: header is a header node.

Output: DLL with deleted node at ending.

1. if(header.rlink = = NULL)
    a) print(DLL is empty, not possible to perform deletionoperation)
2. else
    a) ptr =header
    b) while(ptr.rlink !=NULL)
        i) ptr1 =ptr
        ii) ptr =ptr.rlink
    c) endloop
    d) ptr1.rlink=NULL /* 1 */
    e) return(ptr)
3. end if

End DLL_Deletioon_Ending

Algorithm DLL_Deletion_Any(header,key)

Input: header is header node, key is the data part of a node to be delete.

Output: DLL without node as data part is key value.

```
1.if(header.rlink = = NULL)
        a) print(DLL is empty, not possible for deletionoperation)
2.else
        a) ptr=header
        b) while(ptr.data != key and ptr.rlink != NULL)
                i)ptr =ptr.rlink
        c) end loop
        d) if(ptr.rlink == NULL and ptr.data !=key)
                i) print(required node was not available inlist)
        e) else
                i) ptr1 =ptr.llink
                ii) ptr2=ptr.rlink
                iii) ptr1.rlink=ptr2            /* 1 */
                iv) ptr2.rlink=ptr1             /* 2 */
        f) end if
3.endif
```

End DLL_Deletion_Any

<u>Program:</u>

```
/* DELETION OF A NODE FROM DLL */
#include<stdio.h>
#include<malloc.h>
void create();
void delete();
void traverse();
struct node
{
        int data;
        struct node *llink,*rlink;
}*header,*new1,*ptr,*ptr1,*ptr2;
```

```
void main()
{
	int    ch;
	clrscr();
	header->data=NULL;
	header->llink=NULL;
	header->rlink=NULL;

	while(1)
	{
		printf("\n\nEnter the choice of operation");
		printf("\n1.Create \t 2.Delete \t 3. Traversal\n");
		scanf("%d",&ch);
		switch(ch)
		{
			case1:create();
				break;
			case2:delete();
				break;
			case 3:traverse();
				break;
			default:exit(0);
		}
	}
	getch();
}
voidcreate()
{
	int x;
	new1=malloc(sizeof(struct node));
	printf("\nEnter the data value");
	scanf("%d",&x);
	if(header->rlink==NULL)
	{
```

```
                header->rlink=new1;
                new1->llink=header;
                new1->rlink=NULL;
                new1->data=x;
        }
        else
        {
                ptr=header;
                while(ptr->rlink!=NULL)
                {
                        ptr=ptr->rlink;
                }
                ptr->rlink=new1;
                new1->llink=ptr;
                new1->rlink=NULL;
                new1->data=x;
        }
}
void delete()
{
        int pos,x,key;
        printf("\nEnter the position for deletion");
        printf("\n 1.Begining 2.Ending\t3.At any Position\n");
        scanf("%d",&pos);
        if(pos==1)              /*Deletion atbeginning*/
        {
                ptr=header;
                if(ptr->rlink==NULL)
                {
                        printf("\n DLL is empty");
                }
                else
                {
                        ptr1=ptr;
```

```
                ptr=ptr->rlink;
                ptr2=ptr->rlink;
                ptr1->rlink=ptr2;       /*Addressofsecondnodeiscopiedtorightlinkpartofheadernode*/ ptr2-
                >llink=ptr1;            /* Address of header node is copied to left link part of second node*/
                printf("\nNode deleted is %d",ptr->data);
                free(ptr);
        }
}
else if (pos==2) /* Deletion at Ending */
{
        ptr=header;
        if(ptr->rlink==NULL)
        {
                printf("\n DLL is empty, unable to perform deletion");
        }
        else
        {
                while(ptr->rlink!=NULL)
                {
                        ptr1=ptr;
                        ptr=ptr->rlink;
                }
                 ptr1->rlink=NULL;    /* Last but one node link part is replaced with NULL*/
                printf("\nDeleted Node is %d",ptr->data);
                free(ptr);
        }
}
elseif(pos==3)          /* Deletion at any position*/
{
        ptr=header;
        if(ptr->rlink==NULL)
        {
                printf("\n DLL is empty, unable to perform deletion");
        }
```

```
		else
		{
			printf("\n Enter the data value to delete");
			scanf("%d",&key);
			while(ptr->data!=key && ptr->rlink!=NULL)
			{
				ptr1=ptr;
				ptr=ptr->rlink;			/* Move to the nextnode*/
			}
			if(ptr->rlink==NULL)
			{
				printf("\n Node with key was not found");
			}
			else
			{
				ptr2=ptr->rlink;
				ptr1->rlink=ptr2; /* Next node address is copied to the rlink part of previous node */
				ptr2->llink=ptr1; /* Previous node address is copied to the llink part of next node */
				printf("\n Deleted node is %d",ptr->data);
				free(ptr);
			}
		}
	}
}
void traverse()
{
	printf("\n Elements in the list are:\n");
	ptr=header;
	while(ptr->rlink!=NULL)
	{
		ptr=ptr->rlink;
		printf("\t%d",ptr->data);
	}
}
```

Output:

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 10

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 20

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 30

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 40

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

1

Enter the data value 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

10 20 30 40 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

1

Node deleted is 10

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 30 40 50 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 30 40 50 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

2

Deleted Node is 60

Enter the choice of operation

1.Create 2.Delete 3. Traversal

3

Elements in the list are:

20 30 40 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

3

Enter the data value to delete30

Deleted node is 30

Enter the choice of operation

1.Create 2.Delete 3. Traversal

Elements in the listare:

20 40 50

Enter the choice of operation

1.Create 2.Delete 3. Traversal

2

Enter the position for deletion

1.Begining2.Ending 3.At any Position

55

Enter the choice of operation

1.Create 2.Delete 3. Traversal

0

5. Write a C program that implement stack (its operations) using arrays.

Algorithm:

Algorithm Stack_PUSH(item)

Input: item is new item to push into stack

Output: pushing new item into stack at top whenever stack is not full.

1. if(top >=size)
    a) print(stack is full, not possible to perform pushoperation)
2. else
    a) top=top+1
    b) s[top]=item
3. end if

End Stack_PUSH

Algorithm Stack_POP( )

Input: Stack with some elements.

Output: item deleted at top most end.

1. if(top <1)
    a) print(stack is empty not possible topop)
2. else
    a) item=s[top]
    b) top=top-1
    c) print(deleteditem)
3. end if

End Stack_POP

Program:
```
#includ
e<stdio.h>
#define size 5
int top=0,s[5];
void push(int);
void pop();
void traverse();
```

```
void main()
{
        int i,item,ch;
        clrscr();
        while(1)
        {
                printf("\n Enter your choice 1.push 2.pop 3.traverse");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\n Enter the item ");
                                scanf("%d",&item);
                                push(item);
                                break;
                        case 2: pop();
                                break;
                        case 3: traverse();
                                break;
                        default:exit(0);
                 }
        }
 }
 void push(int item)
 {
        if(top==size)
        {
                printf("\n stack is full ");
        }
        else
        {
                top=top+1;
                s[top]=item;
        }
 }
```

```
void pop()
{
	if(top < 1)
	{
		printf("\n Stack is empty");
	}
	else
	{
		printf("Poped item is %d",s[top]);
		top=top-1;
	}
}
void traverse()
{
	int i;
	if(top <1)
	{
		printf("\n Stack is empty");
	}
	else
	{
		printf("\n Items of stack are ");
		for(i=1;i<=top;i++)
		{
			printf("%d\t",s[i]);
		}
	}
}
```

Output:

Enter your choice 1.push 2.pop 3.traverse1

Enter the item 10

Enter your choice 1.push 2.pop 3.traverse1

Enter the item20

Enter your choice 1.push 2.pop 3.traverse1

Enter the item30

Enter your choice 1.push 2.pop 3.traverse3

Items of stack are1020 30

Enter your choice 1.push 2.pop 3.traverse1

Enter the item 40

Enter your choice 1.push 2.pop 3.traverse3

Items of stack are1020 30 40

Enter your choice 1.push 2.pop 3.traverse2

Poped item is 40

Enter your choice 1.push 2.pop 3.traverse3

Items of stack are1020 30

Enter your choice 1.push 2.pop 3.traverse2

Poped item is 30

Enter your choice 1.push 2.pop 3.traverse3

Items of stack are 10 20

Enter your choice 1.push 2.pop 3.traverse1

Enter the item 55

Enter your choice 1.push 2.pop 3.traverse3

Items of stack are1020 55

Enter your choice 1.push 2.pop 3.traverse0

6. Write C programs that implement Queue (its operations) using linked lists.

Algorithm:

Algorithm Enqueue _LL(item)

Input: item is new item to be insert.

Output: new item i.e new node is inserted at rear end.

1.new = getnewnode()

2.if(new = =NULL)

a) print(required node is not available in memory)

3.else

a) if(front = =NULL and rear==NULL) /* Q is EMPTY*/

i) header.link=new

ii) new.link=NULL

iii) front=new

iv) rear=new

v) new.data=item

b) else /* Q is not EMPTY*/

i) rear.link=new /* 1 */

ii) new.link=NULL /* 2 */

iii) rear=new /* 3 */

iiv) new.data=item

c) end if

4.end if

End_Enqueue_LL

Algorithm Dequeue_LL( )

Input: Queue with some elements

Output: Element is deleted at front end if queue is not empty.

1.if(front==NULL andrear==NULL)

a) print(queue is empty, not possible to perform dequeueoperation)

2.else

a) if(front==rear) /* Q has only one element*/

i) header.link=NULL

ii) item=front.data

iii) front=NULL

iv) rear=NULL

b) else /* Q has more than one element*/

i) header.link=front.link /* 1 */

ii) item=front.data

iii) free(front)

iv)front=header.link /* 2 */

c) end if

d) print(deleted element is item)

3.endif

End_Dequeue_LL

Program:

```
/* QUEUE OPERATIONS USING LINKED LISTS */
#include<stdio.h>
#include<malloc.h>
void enqueue();
void dequeue();
void traverse();
struct node
{
        int data;
        struct node *link;
}*front,*rear,*ptr,*ptr1,*header,*new;
void main()
{
        int i,ch;
        clrscr();
        header->data=NULL;
        header->link=NULL;
        front=NULL;
        rear=NULL;
        while(1)
        {
```

```
        printf("\n Enter your choice 1.Enqueue 2.Dequeue 3.traverse");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:enqueue();
                break;
            case 2:dequeue();
                break;
            case 3:traverse();
                break;
            default:exit(0);
         }
    }
}
void enqueue()
{
    int item;
    new=malloc(sizeof(struct node));
    printf("\n enter item to enqueue");
    scanf("%d",&item);
    if(front==NULL && rear==NULL)
    {
        header->link=new;
        new->link=NULL;
        front=new;
        rear=new;
        new->data=item;
    }
    else
    {
        rear->link=new;
        new->data=item;
        new->link=NULL;
        rear=rear->link;
```

```
		}
	}
	voiddequeue()
	{
		if(front==NULL && rear==NULL)
		{
			printf("\n Queue is empty");
		}
		else
		{
			if(front==rear) /* Q has only one item */
			{
				ptr=front->link;
				header->link=ptr;
				printf("Dequeue item is %d",front->data);
				front=rear=NULL;
			}
			else
			{
				header->link=front->link;
				printf("Dequeue item is %d",front->data);
				free(front);
				front=header->link;
			}
		}
	}
	void traverse()
	{
		ptr=header;
		if(front==NULL &&rear==NULL)
		{
			printf("\n Queue isempty");
		}
		else
```

```
        {
                printf("\n Items ofQueue are     ");
                while(ptr->link!=NULL)
                {
                        ptr=ptr->link;
                        printf("%d\t",ptr->data);
                }
        }
}
```

Output:

Enter your choice 1.Enqueue 2.Dequeue 3.traverse1

enter item toenqueue10

Enter your choice 1.Enqueue 2.Dequeue 3.traverse1

enter item toenqueue20

Enter your choice 1.Enqueue 2.Dequeue 3.traverse1

enter item toenqueue30

Enter your choice 1.Enqueue 2.Dequeue 3.traverse1

enter item toenqueue40

Enter your choice 1.Enqueue 2.Dequeue 3.traverse3

Items ofQueueare 10 20 30 40

Enter your choice 1.Enqueue 2.Dequeue 3.traverse2

Dequeue item is 10

Enter your choice 1.Enqueue 2.Dequeue 3.traverse3

Items ofQueueare 20 30 40

Enter your choice 1.Enqueue 2.Dequeue 3.traverse1 enter item to enqueue55

Enter your choice 1.Enqueue 2.Dequeue 3.traverse3

Items ofQueueare 20 30 40 55

Enter your choice 1.Enqueue 2.Dequeue 3.traverse2

Dequeue item is 20

Enter your choice 1.Enqueue 2.Dequeue 3.traverse3

Items ofQueueare 30 40 55

Enter your choice 1.Enqueue 2.Dequeue 3.traverse0

7. Write a C program that uses Stack operations to convert infix expression into postfix expression.

Algorithm:

Algorithm Conversion of infix to postfix

Input: Infix expression.

Output: Postfix expression.

1. Perform the following steps while reading of infix expression is notover
   - a) if symbol is left parenthesis then push symbol intostack.
   - b) if symbol is operand then add symbol to post fix expression.
   - c) if symbol is operator then check stack is empty ornot.
     - i) if stack is empty then push the operator intostack.
     - ii) if stack is not empty then check priority of theoperators.
       - (I) if priority of current operator >priority of operator present at top of stack then push operator into stack.
       - (II) else if priority of operator present at top of stack>=priority of current operator then pop the operator present at top of stack and add popped operator to postfix expression (go to stepI)
   - d) if symbol is right parenthesis then pop every element form stack up corresponding left parenthesis and add the poped elements to postfixexpression.
2. After completion of reading infix expression, if stack not empty then pop all the items from stack and then add to post fix expression.

End conversion of infix to postfix

Program:
```
#include
<stdio.h>
#include<ctype.h>
#include<string.h>
char s[20];
int top=0;
int priority(char);
void main()
{
        char infix[20],postfix[20],ch;
        int i,j,l;
```

```
clrscr();
printf("\n Enter infix expression: ");
scanf("%s",infix);
l=strlen(infix);
for(i=0,j=0;i<l;i++)
{
	if(infix[i]=='(')
	{
		top=top+1;
		s[top]=infix[i];
	}
	else if(isalpha(infix[i]) || isdigit(infix[i]))
	{
		postfix[j]=infix[i];
		 j=j+1;
	}
	else if(infix[i]=='+' || infix[i]=='-' || infix[i]=='*' || infix[i]=='/'|| infix[i]=='%' || infix[i]=='$')
	{
		if(top<1)
		{
			top=top+1;
			s[top]=infix[i];
		}
		else
		{
			if(priority(infix[i]) > priority(s[top]))
			{
				top=top+1;
				s[top]=infix[i];
			}
			else if(priority(s[top]) >= priority(infix[i]))
			{
				while(priority(s[top]) >= priority(infix[i]))
				{
```

```
                        postfix[j]=s[top];
                        top=top-1;
                        j=j+1;
                    }
                    top=top+1;
                    s[top]=infix[i];
                }
            }
        }
        else if(infix[i]==')')
        {
            while(s[top]!='(')
            {
                postfix[j]=s[top];
                top=top-1;
                j=j+1;
            }
            top=top-1;
        }
        else
        {
            printf("\n Invalid Infix Expression");
        }
    }
    while(top!=0)
    {
        postfix[j]=s[top];
        top=top-1;
        j=j+1;
    }
    postfix[j]='\0';
    printf("\nEquivalent infix to postfix expression is %s",postfix);
    getch();
}
```

```
int priority(char c)
{
	switch(c)
	{
		case'(':	return 0;
		case '+':
		case'-':	return 1;
		case '*':
		case '/':
		case'%':	return2;
		case'$':	return3;
	}
	return 0;
}
```

Output:

Enter infix expression: (a+b)

Equivalent infix to postfix expression is ab+

Enter infix expression: m+n-o

Equivalent infix to postfix expression is mn+o-

Enter infix expression: (a+b*(c-d))

Equivalent infix to postfix expression is abcd-*+

8. Write a C program that uses Stack operations to evaluate postfix expression.

Algorithm:

Algorithm PostfixExpressionEvaluation

Input: Postfix expression

Output: Result of Expression

1. Repeat the following steps while reading the postfixexpression.

    a) if the read symbol is operand, then push the symbol intostack.

    b) if the read symbol is operator then pop the top most two items of the stack and apply the operator on them, and then push back the result to thestack.

2. Finally stack has only one item, after completion of reading the postfix expression. That item is the result ofexpression.

End PostfixExpressionEvaluation

Program:

/* EVALUATION OF POSTFIX EXPRESSION */

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<math.h>
char s[20];
int top=0;
void main()
{
	char postfix[20],symb;
	int i=0,l,op1,op2,res;
	clrscr();
	printf("\n Enter infix expression");
	scanf("%s",postfix);
	l=strlen(postfix);
	for(i=0;i<l;i++)
	{
		if(isdigit(postfix[i]))
		{
			top=top+1;
```

```
				s[top]=postfix[i]-48;
			}
			else
			{
				op2=s[top];
				top=top-1;
				op1=s[top];
				top=top-1;
				switch(postfix[i])
				{
					case '+':
						res=op1+op2;
						break;
					case '-':
						res=op1-op2;
						break;
					case '*':
						res=op1*op2;
						break;
					case '/':
						res=op1/op2;
						break;
					case '%':
						res=op1%op2;
						break;
					case '$':
						res=pow(op1,op2);
						break;
				}
				top=top+1;
				s[top]=res;
			}
		}
		printf("\nResult postfix expression is %d",s[top]);
```

```
	getch();
}
```

Output:

Enter infix expression56-

Result postfix expression is-1

Enter infix expression56*2-

Result postfix expression is28

Enter infix expression56+37-*

Result postfix expression is-44

9. Write a C program to create a Binary Search Tree of integers and perform insert, traversaloperations.

Algorithm:

Algorithm BST_Insert(item)

Input: item is data part of new node to be insert into BST.

Output: BST with new node has data part item.

1. ptr =Root
2. flag =0
3. while(ptr != NULL and flag == 0)
    - a) if(item ==ptr.data)
        - i) flag =1
        - ii) print(item alreadyexist)
    - b) else if(item <ptr.data)
        - i) ptr1 =ptr
        - ii) ptr =ptr.LCHILD
    - c) else if(item >ptr.data)
        - i) ptr1 =ptr
        - ii) ptr =ptr.RCHILD
    - d) end if
4. end loop
5. if(ptr ==NULL)
    - a) new =getnewnode()
    - b) new.data = item
    - c) new.lchild =NULL
    - d) new.rchild =NULL
    - e) if(root.data ==NULL)
        - i) root=new;
            - A) print(New node inserted successfully as ROOTNode)
    - f) else if(item<ptr1.data) /* inserting new node as left child to itsparent*/
        - i) ptr1.lchild=new
        - ii) print(New Node is inserted successfully as LEFTchild)
    - g) else /* inserting new node as right child to itsparent*/
        - i) ptr1->rchild=new;
        - ii) print(New Node is inserted successfully as RightChild)

h) end if

6. end if

End BST_Insert

<u>Program:</u>

```
/* BST INSERTION AND TRAVERSAL */
#include<stdio.h>
#include<malloc.h>
struct node
{
	int data;
	struct node *lchild,*rchild;
}*ptr,*ptr1,*root,*new,*parent,*ptr2,*ptr3,*ptr4;
void insertion();
void preorder (struct node *);
void inorder (struct node *);
void postorder (struct node *);

void main()
{
	int ch,c,item;
	clrscr();
	root->data=NULL;
	root->lchild=NULL;
	root->rchild=NULL;
	ptr=root;
	while(1)
	{
		printf("\n enter your choice of operation");
		printf("\n 1. insertion \t 2.Traverse");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
```

```
                    insertion();
                    break;
                case 2:
                    printf("\n Enter your traversal");
                    printf("\n 1. Preorder 2. Inorder 3. Postorder");
                    scanf("%d",&c);
                    if(c==1)
                    {
                        printf("\n Nodes in BST are");
                        preorder(root);
                    }
                    else if(c==2)
                    {
                        printf("\n Nodes in BST are");
                        inorder(root);
                    }
                    else if(c==3)
                    {
                        printf("\n Nodes in BST are");
                        postorder(root);
                    }
                    break;
                    default : exit(0);
            }
        }
        getch();
}

voidinsertion()
{
        int item,flag=0;
        ptr=root;
        printf("\n Enter data part of node to be insert");
        scanf("%d",&item);
```

```
while(ptr!=NULL && flag==0)
{
        if(item==ptr->data)
        {
                printf("\n Item already exist in BST");
                flag=1;
        }
        elseif(item<ptr->data)
        {
                ptr1=ptr;
                ptr=ptr->lchild;
        }
        elseif(item>ptr->data)
        {
                ptr1=ptr;
                ptr=ptr->rchild;
        }
}

if(ptr==NULL)
{
        new=malloc(sizeof(struct node));
        new->data=item;
        new->lchild=NULL;
        new->rchild=NULL;
        if(root->data==NULL)
        {
                root=new;
                printf("\n Node %d is inserted successfully as ROOT Node",item);
        }
        elseif(item<ptr1->data)         /* inserting new node as left child to itsparent*/
        {
                ptr1->lchild=new;
                printf("\n Node %d is inserted successfully LEFT child",item);
```

```
		}
		else			/* inserting new node as right child to itsparent*/
		{
			ptr1->rchild=new;
			printf("\n Node %d is inserted successfully Right Child",item);
		}
	}
}
void preorder(struct node *p)
{
	if(p!=NULL)
	{
		printf("\t %d",p->data);
		preorder(p->lchild);
		preorder(p->rchild);
	}
}
void inorder(struct node *p)
{
	if(p!=NULL)
	{
		inorder(p->lchild);
		printf("\t %d",p->data);
		inorder(p->rchild);
	}
}

void postorder(struct node *p)
{
	if(p!=NULL)
	{
		postorder(p->lchild);
		postorder(p->rchild);
		printf("\t %d",p->data);
```

```
	}
}
```

Output:

enter your choice of operation

1.insertion   2. Traverse1

Enter data part of node to be insert 45

Node 45 is inserted successfully as ROOT Node

enter your choice of operation

1.insertion   2. Traverse1

Enter data part of node to be insert 15

Node 15 is inserted successfully LEFT child

enter your choice of operation

1.insertion   2. Traverse1

Enter data part of node to be insert 48

Node 48 is inserted successfully Right Child

enter your choice of operation

1.insertion   2. Traverse1

Enter data part of node to be insert 22

Node 22 is inserted successfully Right Child

enter your choice of operation

1.insertion   2. Traverse1

Enter data part of node to be insert 46

Node 46 is inserted successfully LEFT child

enter your choice of operation

1.insertion   2. Traverse 2

Enter yourtraversal

1. Preorder 2. Inorder 3. Postorder 1

Nodes inBSTare 45 15 22 48 46

enter your choice of operation

1.insertion 2. Traverse1

Enter data part of node to be insert 11

Node 11 is inserted successfully LEFT child

enter your choice of operation

1.insertion 2. Traverse1

Enter data part of node to be insert 43

Node 43 is inserted successfully Right Child

enter your choice of operation

1.insertion 2. Traverse1

Enter data part of node to be insert 55

Node 55 is inserted successfully Right Child

enter your choice of operation

1.insertion 2. Traverse 2

Enter yourtraversal

1. Preorder 2. Inorder 3. Postorder 3

Nodes inBSTare 11 43 22 15 46 55 48 45

enter your choice of operation

1.insertion 2. Traverse 2

Enter yourtraversal

1. Preorder 2. Inorder 3. Postorder 2

Nodes inBSTare 11 15 22 43 45 46 48 55

enter your choice of operation

1.insertion 2. Traverse 2

Enter yourtraversal

1. Preorder 2. Inorder 3. Postorder 1

Nodes inBSTare 45 15 11 22 43 48 46 55

10. (i) Write a C program to implement Depth First Search for agraph.

Algorithm:

Algorithm DFS( )

Input: Adjacent matrix representation of a graph.

Output: DFS traversal of graph.

1. PUSH the starting vertex intostack
2. While stack is notempty
   a) POP a vertex v from stack
   b) if vertex v is notvisited
      i) visit the vertex v
      ii) PUSH all the adjacent vertices of v into stack
   c) endif
3. end loop

EndDFS

Program:

```
/* DFS USING ADJACENT MATRIX using RECURSION*/
#include<stdio.h>
#include<conio.h>
int a[20][20],s[20],visited[20],n,i,j,item,st,v, top=0,size=20;
void push(int);
int pop();
void dfs();
void main()
{
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                visited[i]=0;
        }
        printf("\n Enter Adjacency matrix:\n");
```

```
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n Enter the starting vertex:");
    scanf("%d",&st);
    push(st);                               /* PUSH the starting vertex into the STACK*/
    printf("\n DFS for Given graph is");
    dfs();
    getch();
}
void dfs()
{
    if(top!=0)                              /* While the STACK is not EMPTY*/
    {
        v=pop();                            /* POP vertex from STACK*/
        if(visited[v]==0)                   /* If the poped vertex is not visited*/
        {
            visited[v]=1;                   /* Visit the poped vertex */
            printf("\n%d",v);
            for(i=1;i<=n;i++)
            {
                if(a[v][i]==1)
                {
                    push(i);                /* PUSH ALL THE ADJACENT VERTICES OF vertex V in to STACK*/
                }
            }
        }
        dfs();
    }
}
```

```
void push(int item)
{
        if(top==size)
        {
                printf("\n Stack is full ");
        }
        else
        {
                top=top+1;
                s[top]=item;
        }
 }

int pop()
{
        int item;
        if(top==0)
        {
                printf("\n Stack is empty");
                return 0;
        }
        else
        {
                item=s[top];
                top=top-1;
                return item;
        }
}
```

/* DFS USING ADJACENT MATRIX NON RECURSION*/

```
#include<stdio.h>
#include<conio.h>
int top=0,size=20,v;
int a[20][20],s[20],visited[20],n,i,j,item,st;
void push(int);
int pop();
void dfs();
void main()
{
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                visited[i]=0;
        }
        printf("\n Enter Adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("\n Enter the starting vertex:");
         scanf("%d",&st);
         push(st);                          /* PUSH the starting vertex into the STACK */
        printf("\n DFS for Given graph is");
        dfs();
        getch();
}
```

```
void dfs()
{
        while(top!=0)                          /* While the STACK is not EMPTY*/
        {
                v=pop();                       /* POP vertex from STACK*/
                if(visited[v]==0)              /* If the poped vertex is not visited*/
                {
                        visited[v]=1;          /* Visit the poped vertex */
                        printf("\n%d",v);
                        for(i=1;i<=n;i++)
                        {
                                if(a[v][i]==1)
                                {
                                        push(i);    /* PUSH ALL THE ADJACENT VERTICES OF vertex V in to STACK*/
                                }
                        }
                }
        }
}

void push(int item)
{
        if(top==size)
        {
                printf("\n Stack is full ");
        }
        else
        {
                top=top+1;
                s[top]=item;
        }
 }
```

```
int pop()
{
	int item;
	if(top==0)
	{
		printf("\n Stack is empty");
		return 0;
	}
	else
	{
		item=s[top];
		top=top-1;
		return item;
	}
}
```

Output 1:

```
Enter the number of vertices:5
 Enter Adjacency matrix:
0	1	0	0	1
1	0	1	0	0
0	1	0	1	1
0	0	1	0	1
1	0	1	1	0

Enter the starting vertex:1
 DFS for Given graph is
 1
 5
 4
 3
 2
```

Output 2:

Enter the number of vertices:5

Enter Adjacency matrix:

0	1	1	0	0
1	0	1	1	1
1	1	0	1	1
0	1	1	0	1
0	1	1	1	0

Enter the starting vertex:1

DFS for Given graph is

1

3

5

4

2

10. (ii) Write a C program to implement Breadth First Search for a graph.

Algorithm BFS()

Input: Adjacent matrix representation of a graph.

Output: BFS traversal of graph.

1. ENQUEUE the starting vertex intoqueue
2. While queue is notempty
   a) DEQUEUE a vertex v from queue
   b) if vertex v is notvisited
      i) visit the vertex v
      ii) ENQUEUE all the adjacent vertices of v into queue
   c) endif
3. end loop

End BFS

```
/* BFS using RECURSION*/
#include<stdio.h>
#include<conio.h>
int front=0,rear=0,size=20;
int a[20][20],q[20],visited[20],n,i,j,item;
void bfs();
void enqueue(int);
int dequeue();
voidmain()
{
        int s;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                visited[i]=0;
        }
        printf("\n Enter Adjacency matrix:\n");
        for(i=1;i<=n;i++)
```

```
	{
		for(j=1;j<=n;j++)
		{
			scanf("%d",&a[i][j]);
		}
	}
	printf("\n Enter the starting vertex:");
	scanf("%d",&s);
	enqueue(s);				/* Enqueue the starting vertex into the QUEUE*/
	printf("\n BFS for Given graph is");
	bfs(s);
	getch();
}
void bfs()
{
	int v;
	if(front!=0 &&rear !=0)			/* While the QUEUE is not EMPTY*/
	{
		v=dequeue();			/* Dequeue vertex from QUEUE */
		if(visited[v]==0)		/* If Dequeued vertex is not visited*/
		{
			visited[v]=1;
			printf("\n %d",v);
			for(i=1;i<=n;i++)
			{
				if(a[v][i]==1)
				{
					enqueue(i);	/* Enqueue ALL THE ADJACENT VERTICES OF vertex V*/
				}
			}
		}
		bfs();
	}
}
```

```
void enqueue(int item)
 {
	if(rear==size)
	{
		printf("\n Queue is full ");
	}
	else
	{
		if(front==0 && rear==0)
		{
			front=1;
			rear=rear+1;
			q[rear]=item;
		}
		else
		{
			rear=rear+1;
			q[rear]=item;
		}
	}
 }

int dequeue()
{
	int item;
	if(front==0 && rear==0)
	{
		printf("\n Queue is empty");
		return 0;
	}
	else
	{
		if(front==rear)
		{
```

```
                item=q[front];
                front=0;
                rear=0;
                returnitem;
            }
            else
            {
                item=q[front];
                front=front+1;
                returnitem;
            }
    }
}


/* BFS using NON RECURSION*/
#include<stdio.h>
#include<conio.h>
int front=0,rear=0,size=20;
int a[20][20],q[20],visited[20],n,i,j,item;
void bfs();
void enqueue(int);
int dequeue();
voidmain()
{
    int s;
    clrscr();
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        visited[i]=0;
    }
    printf("\n Enter Adjacency matrix:\n");
```

```
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			scanf("%d",&a[i][j]);
		}
	}
	printf("\n Enter the starting vertex:");
	scanf("%d",&s);
	enqueue(s);				/* Enqueue the starting vertex into the QUEUE*/
	printf("\n BFS for Given graph is");
	bfs();
	getch();
}
void bfs()
{
	int v;
	while(front!=0 &&rear!=0)		/* While the QUEUE is not EMPTY*/
	{
		v=dequeue();			/* Dequeue vertex from QUEUE */
		if(visited[v]==0)		/* If Dequeued vertex is not visited*/
		{
			visited[v]=1;
			printf("\n %d",v);
			for(i=1;i<=n;i++)
			{
				if(a[v][i]==1)
				{
					enqueue(i);	/* Enqueue ALL THE ADJACENT VERTICES OF vertex V*/
				}
			}
		}
	}
}
```

```
void enqueue(int item)
 {
	if(rear==size)
	{
		printf("\n Queue is full ");
	}
	else
	{
		if(front==0 && rear==0)
		{
			front=1;
			rear=rear+1;
			q[rear]=item;
		}
		else
		{
			rear=rear+1;
			q[rear]=item;
		}
	}
 }
int dequeue()
{
	int item;
	if(front==0 && rear==0)
	{
		printf("\n Queue is empty");
		return 0;
	}
	else
	{
		if(front==rear)
		{
			item=q[front];
```

```
                        front=0;
                        rear=0;
                        returnitem;
                }
                else
                {
                        item=q[front];
                        front=front+1;
                        returnitem;
                }
        }
}
```

Output 1:

Enter the number of vertices: 5

Enter Adjacency matrix:

0	1	0	0	1
1	0	1	0	0
0	1	0	1	1
0	0	1	0	1
1	0	1	1	0

Enter the starting vertex:1

BFS for Given graph is

1

2

5

3

4

Output 2:

Enter the number of vertices:5

Enter Adjacency matrix:

0	1	1	0	0
1	0	1	1	1

1	1	0	1	1
0	1	1	0	1
0	1	1	1	0

Enter the starting vertex:1

BFS for Given graph is

1

2

3

4

5

11. Write a C program to create a hash table and perform insert, display and search operations.

<u>Algorithm:</u>

Algorithm Hash_Division(key)

Input: Key is new is inserting into hash table.

Output: index for inserting key into hash table.

1. return (key %m)

End Hash_Division

Algorithm Hash_Multiplication(key)

Input: Key is new is inserting into hashtable.

Output: index for inserting key into hash table.

1. A = 0.61804
2. t1 = key* A /* t1 is integer variable*/
3. t2 = key* A /* t2 is floating point variable*/
4. t2 = t2– t1 /* getting fractional part*/
5. pos = t2* m /* Multiplying fractional part with m value*/
6. return(pos)

End Hash_Multiplication

Algorithm Hash_Universal(key)

Input: Key is new is inserting into hash table.

Output: index for inserting key into hash table.

1. return (((k*a + b) % p) % m)

End Hash_Universal

<u>Program:</u>

```
/* Operations on Hash tables i.e. Insertion, Display, Search
Using Different Hash methods, i.e. DIVISION METHOD, MULTIPLCATION METHOD,UNIVERSAL Hashing */
#include<stdio.h>
#include<conio.h>

int htable[50],h,ch,k,m=11,i,pos,count=0,flag;
```

```
void insert(int);
int search(int);
void display();
void main()
{
	clrscr();
	printf("-------------HASHFUNCTIONS ------------- ");
	printf("\n[1].Division\n[2].Multiplication\n[3].Universal");
	printf("\nChoose Hash Function:");
	scanf("%d",&h);
	while(1)
	{
		printf("\nEnter choice of OERATIONS ON DICTIONARIES");
		printf("\n[1].INSERT\t[2].SEARCH\t[3].DISPLAY\t[0].EXIT");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				if(count>=m)
				{
					printf("\nHash Table is Full, SO INSERTION NOT POSSIBLE");
				}
				else
				{
					printf("Enter Key to be insert:");
					scanf("%d",&k);
					insert(k);
				}
				break;
			case 2:
				if(count==0)
				{
					printf("\nHash Table is Empty");
				}
```

```
else
{ printf("Enter Key to be search:");
 scanf("%d",&k); pos=search(k);
if(htable[pos]==k)
{
printf("\nKey Found At Location:%d",pos);
}
else
 {
 printf("\nKey Not Found in the Hash Table");
 }
 }
 break;
case 3:
 if(count==0)
{
 printf("\nHash Table is Empty");
} else
 {
display();
 }
break;
default:
exit(0);
}
 }
 getch();
}
```

```
int division(int k)

{

 return (k%m);

}

int multiply(int k)

{

 float A=0.61804,t2,fa;

int t1;

 t1=(k*A);

 t2=(k*A);

fa=t2-t1;

 return fa*m;

 }

int universal(int k)

{

 int p=13,a=7,b=5;

 pos=((k*a+b)%p)%m;

return pos;

 }

void insert(int k)
 {
 pos=search(k);
 if(htable[pos]==k)
 {
 printf("\nKey Already Found in the Hash Table");
}
Else
 {
 htable[pos]=k;
printf("\nKey Inserted at Location: %d",pos);
 count++; /* Increment number of keys present in hash table bu 1 */
 }
}
```

```
int search(int k)
{
int tc;
 if(h==1) { pos=division(k);
}
 else if(h==2)
 { pos=multiply(k);
 }
 Else
 {
pos=universal(k);
}
if(ch==1)
 {
flag=0; while(flag==0)
{
 if(htable[pos]==k || htable[pos]==0)
{
 flag=1;
break;
 }
else
 {
printf("\n Collision Occured at Location: %d",pos);
pos=(pos+1)%m;
}
 }
 } else /* Other than insertion operation */
{
flag=0;
 tc=0; /* Temporary count whenever u r started for next index to be search*/
while(flag==0)
{
 if(htable[pos]==k || tc==m)
{
 flag=1; break;
}
```

```
else
 {
 pos=(pos+1)%m; tc++;
}
 }
 }
 return pos;
}
void display()
{ printf("\nElements in the Hash Table are:\n");
 printf("\nIndex \t Key");
for(i=0;i<m;i++)
{
 printf("\n%d\t %d",i,htable[i]);
}
}
```

Output:

```
-------------HASH FUNCTIONS--------------
[1].Division
[2].Multiplication
[3].Universal

Enter choice of OERATIONS ON DICTIONARIES

 [1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT 1

Enter Key to be insert:23

Key Inserted at Location: 1

Enter choice of OERATIONS ON DICTIONARIES

 [1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:42 Key Inserted at Location: 9

Enter choice of OERATIONS ON DICTIONARIES
```

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:31

Collision Occurred at Location: 9

Key Inserted at Location: 10

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:66

Key Inserted at Location: 0
Enter choice of OERATIONS ON DICTIONARIES
[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT
1
Enter Key to be insert:44
Collision Occurred at Location: 0
Collision Occurred at Location: 1

Key Inserted at Location: 2

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:23

Key Already Found in the Hash Table

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:34

Collision Occurred at Location: 1

Collision Occurred at Location: 2

Key Inserted at Location: 3 Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:45

Collision Occurred at Location: 1

Collision Occurred at Location: 2

Collision Occurred at Location: 3

Key Inserted at Location: 4

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

1

Enter Key to be insert:77

Collision Occurred at Location: 0
Collision Occurred at Location: 1
Collision Occurred at Location: 2
Collision Occurred at Location: 3
Collision Occurred at Location: 4

Key Inserted at Location: 5

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

3

Elements in the Hash Table are:

Index Key

0 66

1 23

2 44

3 34

4 45

5 77

6 0

7 0

8 0

9 42

10 31

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

2

Enter Key to be search:34

Key Found At Location:3

Enter choice of OERATIONS ON DICTIONARIES

[1].INSERT [2]. SEARCH [3].DISPLAY [0].EXIT

2

Enter Key to be search:99

Key Not Found in the Hash Table

Enter choice of OERATIONS ON DICTIONARIES

Additional Programs

1. Write C programs that implement stack (its operations) using Linked list

Description:

In this program we have to implement the stack operation by using the pointers. Here they stack operation are push and pop. Push operation is used to insert the elements into a stack and pop operation is used to remove the elements in to a stack.

Algorithm:

Step 1: Start

Step 2: Declare the structure for the stack pointers.

Step 3: Define the push function

Step 4: Define the pop function

Step 5: Define the display function

Step 6: Read the choice

Step 7: if choice = push
    Create a cell for the TOP cell in the stack.
    Place the date in the TOP cell
    Place the TOP pointer to the new cell

Step 8: if choice=pop
    Check if empty stack. If so, print stack is empty.
    Otherwise, remove the TOP cell.

Step 9: if choice=display
    Display all the elements in the Stack.

Step 10: Stop

PROGRAM:

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

struct node
{
int data;
struct node *link;
};
struct node *top=NULL,*temp;
void main()
{
int choice,data;
clrscr();

while(1)//infinite loop is used to insert/delete infinite number of nodes
```

```
{

printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
printf("\nEnter ur choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
temp=(struct node *)malloc(sizeof(struct node));
printf("Enter a node data :");
scanf("%d",&data);
temp->data=data;
temp->link=top;
top=temp;
break;
case 2:
if(top!=NULL)
{
printf("The poped element is %d",top->data);
top=top->link;
}
else
{
printf("\nStack Underflow");
}
break;
case 3:
temp=top;
if(temp==NULL)
{
printf("\nStack is empty\n");
}

while(temp!=NULL)
{
printf("->%d->",temp->data);
temp=temp->link;
}
break;
case 4:
exit(0);
}
}
}
```

INPUT AND OUTPUT:

```
1.Push
2.Pop
3.Display
4.Exit

Enter ur choice:1
Enter a node data :11

1.Push
2.Pop
```

3.Display
4.Exit

Enter ur choice:1
Enter a node data :22

1.Push
2.Pop
3.Display
4.Exit

Enter ur choice:3
->22->->11->
1.Push
2.Pop
3.Display
4.Exit

Enter ur choice:2
The poped element is 22
1.Push
2.Pop
3.Display
4.Exit

Enter ur choice:3
->11->
1.Push
2.Pop
3.Display
4.Exit

Enter ur choice:4
EXIT

Viva Questions:

1.How to allocate memory for a node in linked list?
2.What are the operations on stack?

2. **Write C programs that implement Queue (its operations) using arrays.**

Description:

In this program we have to implement the Queue operation by using the arrays. Here they Queue operation are push and pop. Push operation is used to insert the elements into a Queue and pop operation is used to remove the elements in to a Queue.

ALGORITHM FOR INSERTING AN ELEMENT IN TO A QUEUE:

```
Function QINSERET(Q,F,R,N,Y)
    Step 1: [overflow]
              If R>=N
              Then printf(" overflow")
               Return
    Step 2: [Increment rear pointer]
             R=R+1

    Step 3: [ Insert element]
             Q[R]=y

    Step 4: [Is front pointer properly set?]
              If F=0
              Then f=1
              Return
```

ALGORITHM FOR DELETING AN ELEMENT FROM A Queue:

```
Function  QDELETE(Q,F,R)

    Step 1: [Underflow]
              If F=0
              Then printf("Queue underflow")
               Return(0)
    Step 2: [Delete element]
             Y=q[f]

    Step 3: [Is Queue Empty?]
              If F=R
              Then  F=R=0
              Else
                    F=F+1

    Step 4:[Return element]
             Return(r)
```

Program:

```
# include <stdio.h>
# define size 4
int front=0,rear=-1,item,choice,a[size];
main()
{
clrscr();
while(1)
{
        printf("*** MENU ***\n 1. INSERTION\n 2. DELETION\n
      3.TRAVERSE\n 4. EXIT\n");
        printf("enter your choice:");
```

```
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:insertion();
				break;
			case 2:deletion();
				 break;
			case 3:traverse();
				   break;
			case 4:exit();
			default:printf("*** wrong choice ***\n");
		}
	}
	getch();
	}
	insertion()
	{
		if(rear==size-1)
		printf("*** queue is full ***\n");
		else
		{
			printf("enter item into queue:");
			scanf("%d",&item);
			rear++;
			a[rear]=item;
		}
	}
	deletion()
	{
		if(front==rear+1)
		printf("*** queue is empty ***\n");
		else
		{
		item=a[front];
		front++;
		printf("the deleted item from queue is %d\n",item);
		}
	}
	traverse()
	{
		int i;
		if(front==rear+1)
		printf("*** queue is empty ***\n");
		else
		{
			for(i=front;i<=rear;i++)
			if(i==front && rear==i)
			printf("%d at %d ->front=rear\n",a[i],i);
			else
			if(i==rear)
			printf("%d at %d ->rear\n",a[i],i);
			else
			if(i==front)
			printf("%d at %d ->front\n",a[i],i);
			else
			printf("%d at %d\n",a[i],i);
		}
```

}

Input/Output:

```
*** MENU ***
 1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:1
enter item into queue:11
*** MENU ***
 1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:1
enter item into queue:12
*** MENU ***
1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:1
enter item into queue:13
*** MENU ***
1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:1
enter item into queue:14
*** MENU ***
1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:1
*** queue is full ***
*** MENU ***
 1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:3
11 at 0 ->front
12 at 1
13 at 2
14 at 3 ->rear
*** MENU ***
 1. INSERTION
 2. DELETION
 3. TRAVERSE
 4. EXIT
enter your choice:2
the deleted item from queue is 11
*** MENU ***
```

1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:2
the deleted item from queue is 12
*** MENU ***
1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:2
the deleted item from queue is 13

*** MENU ***
1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:2
the deleted item from queue is 14
*** MENU ***
1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:2
*** queue is empty ***
*** MENU ***
1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:3
*** queue is empty ***
*** MENU ***
1. INSERTION
2. DELETION
3. TRAVERSE
4. EXIT

enter your choice:4

Exit

Viva Questions:

1.What are the conditions for queue full and queue empty?
2.What are the operations on queue?