MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DATA STRUCTURES
LAB MANUAL
IInd YEAR CIVIL
(2022-2023)

Prepared by,

Dr.Hussain Sharif

Associate Professor
CSE Dept

CIVIL ENGINEERING

Program Outcomes	
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**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.
PO3	**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.
PO4	**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.
PO5	**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.
PO6	**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.
PO7	**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.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**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.
PO11	**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.
PO12	**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	
PSO1	**Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and Networking for efficient analysis and design of computer-based systems of varying complexity.
PSO2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies.

DATA STRUCTURES LAB SYLLABUS

Recommended Systems/Software Requirements:
Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100MB free disk space. C compiler.

***Content beyond the university prescribed syllabus**

ATTAINMENT OF PROGRAM OUTCOMES & PROGRAM SPECIFIC OUTCOMES

Exp. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
1	Write a C program that uses functions to perform the following: a) Create a singly linked list of integers. b) Delete a given integer from the above linked list. c) Display the contents of the above list after deletion. d) Traversal the contents of the above list.	PO1, PO2	PSO1
2	Write a C program that uses functions to perform the following: a) Create a doubly linked list of integers. b) Delete a given integer from the above doubly linked list. c) Display the contents of the above list after deletion.	PO1, PO2	PSO1
3	Write a C program that uses functions to perform the following: a) Create a circular linked list of integers. b) Delete a given integer from the above circular linked list. c) Display the contents of the above list after deletion.	PO1, PO2	PSO1, PSO2
4	Write a C program that implement stack operation using i) Arrays ii) Pointers	PO1, PO2	PSO1
5	Write a C programs to implement infix to postfix conversion using stack	PO1, PO2, PO3	PSO1, PSO2
6	Write a C programs to implement postfix evolution	PO1, PO2, PO3	PSO1
7	Write a C program that implements queue operations using i) Arrays ii) Pointers	PO1, PO2	PSO1
8	Write a C programs for implementing the following sorting methods to arrange alist of integers in ascending order: a) Insertion sort b) Bubble sort c) Selection Sort	PO1, PO2	PSO1
9	Write C programs for implementing the following sorting methods to arrange alist of integers in ascending order: a) Quick sort b) Merge sort	PO1, PO2, PO3	PSO1
10	Write a C program that use both recursive and non-recursive functions to perform the following searching operations for key value in a given list of integers i) Linear Search ii) Binary Search	PO1, PO2, PO3	PSO1, PSO2
11	Write a C program to implement the tree traversal methods using both recursive and non-recursive	PO1, PO2, PO4, PO12	PSO1, PSO2
12	Write C programs for implement the graph traversal methods	PO1, PO2, PO3	PSO1
	Content Beyond Syllabi		
1	*Write a C Program to check whether two given lists are containing the same data.	PO1, PO2	PSO1
2	*Write a C program to find the largest element in a given doubly linked list.	PO1, PO2	PSO1
3	*Write a C program to reverse the elements in the stack using recursion.	PO1, PO2	PSO1, PSO2

Exp. No.	**Experiment**	**Program Outcomes Attained**	**Program Specific Outcomes Attained**
4	*Write a C program to implement stack using linked list.	PO1	PSO1
5	*Write a C program to count the number of nodes in the binary search tree.	PO1, PO2	PSO1
6	*Write a C program to sort an array of integers in ascending order using radix sort.	PO1, PO2	PSO1
7	*Write a C program to sort a given list of strings.	PO1	PSO1

*****Content beyond the University prescribed syllabus

DATA STRUCTURES LABORATORY

OBJECTIVE:

The objective of this lab is to teach students various data structures and to explain them algorithms for performing various operations on these data structures. This lab complements the data structures course. Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables and search trees.

OUTCOMES:

Upon the completion of Data Structures practical course, the student will be able to:

1. **Design** and analyze the time and space efficiency of the data structure.
2. **Identity** the appropriate data structure for given problem.
3. **Understand** the applications of data structures.
4. **Choose** the appropriate data structure and algorithm design method for a specified application.
5. **Understand** which algorithm or data structure to use in different scenarios.
6. **Understand** and apply fundamental algorithmic problems including Tree traversals, Graph traversals.
7. **Compare** different implementations of data structures and to recognize the advantages and disadvantages of them.
8. **Write** complex applications using structured programming methods.

EXPERIMENT 1

OBJECTIVE

1. To create a singly linked list of integers.
2. Delete a given integer from the above linked list.
3. Display the contents of the above list after deletion.

RESOURCE:

Turbo C

PROGRAM LOGIC

1. Create a node using structure
2. Dynamically allocate memory to node
3. Create and add nodes to linked list

PROCEDURE

Go to debug -> run or press CTRL + F9 to run the program.

SOURCE CODE

program to create a single linked list, delete the contents and display the contents

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
/*declaring a structure to create a node*/
struct node
{
    int data;
    struct node *next;
};
struct node *start;

/* inserting nodes into the list*/
/*function to insert values from beginning of the the single linked list*/
void insertbeg(void)
{
    struct node *nn;
    int a;
    /*allocating implicit memory to the node*/
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    a=nn->data;
    if(start==NULL)             /*checking if List is empty*/
    {
        nn->next=NULL;
        start=nn;
    }
    else
    {
        nn->next=start;
        start=nn;
    }
    printf("%d succ. inserted\n",a);
    return;
}
 /*function to insert values from the end of the linked list*/
```

```
void insertend(void)
{
    struct node *nn,*lp;int b;
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    b=nn->data;
    if(start==NULL)
    {
        nn->next=NULL;
        start=nn;
    }
    else
    {
        lp=start;
        while(lp->next!=NULL)
        {
            lp=lp->next;
        }
        lp->next=nn;
        nn->next=NULL;
    }
    printf("%d is succ. inserted\n",b);
    return;
}
```
/*function to insert values from the middle of the linked list*/
```
void insertmid(void)
{
    struct node *nn,*temp,*ptemp;int x,v;
    nn=(struct node *)malloc(sizeof(struct node));
    if(start==NULL)
    {
        printf("sll is empty\n"); return;
    }
    printf("enter data before which no. is to be inserted:\n");
    scanf("%d",&x);
    if(x==start->data)
    {
        insertbeg();
        return;
    }
    ptemp=start;
    temp=start->next;
    while(temp!=NULL&&temp->data!=x)
    {
        ptemp=temp;
        temp=temp->next;
    }
    if(temp==NULL)
    {
        printf("%d data does not exist\n",x);
    }
    else
    {
            printf("enter data:");
            scanf("%d",&nn->data);
            v=nn->data;
            ptemp->next=nn;
            nn->next=temp;
            printf("%d succ. inserted\n",v);
    }
    return;
}
 /*deletion operation*/
void deletion(void)
{
    struct node *pt,*t;
    int x;
    if(start==NULL)
    {
        printf("sll is empty\n");
        return;
    }
    printf("enter data to be deleted:");
    scanf("%d",&x);
    if(x==start->data)
    {
        t=start;
        /* assigning first node pointer to next nod pointer to delete a data
        from the  starting of the node*/
        start=start->next;
        free(t);
        printf("%d is succ. deleted\n",x);
        return;
    }
    pt=start;
    t=start->next;
    while(t!=NULL&&t->data!=x)
    {
        pt=t;t=t->next;
    }
    if(t==NULL)
    {
        printf("%d does not exist\n",x);return;
     }
    else
    {
        pt->next=t->next;
    }
    printf("%d is succ. deleted\n",x);
    free(t);
    return;
}
void display(void)
{
    struct node *temp;
    if(start==NULL)
    {
        printf("sll is empty\n");
        return;
    }
    printf("elements are:\n");
    temp=start;
    while(temp!=NULL)
    {
        printf("%d\n",temp->data);
```

```
            temp=temp->next;
        }
        return;
    }
     /*  main program*/
    int main()
    {
        int c,a;    start=NULL;
        do
        {
            printf("1:insert\n2:delete\n3:display\n4:exit\nenter choice:");
            scanf("%d",&c);
            switch(c)
            {
                case 1:
                printf("1:insertbeg\n2:insert end\n3:insert mid\nenter choice:");
                scanf("%d",&a);
                switch(a)
                {
                     case 1:insertbeg(); break;
                    case  2:insertend();  break;
                    case 3:insertmid(); break;
                }
                break;
                case 2:deletion(); break;
                case 3:display(); break;
                case 4:printf("program ends\n");break;
                default:printf("wrong choice\n");
                break;
            }
        }while(c!=4);return 0;
    }
```

PRE LAB QUESTIONS

1. What is data structure
2. How the memory is allocated dynamically
3. What is linked list
4. What is node
5. What are the types of linked list

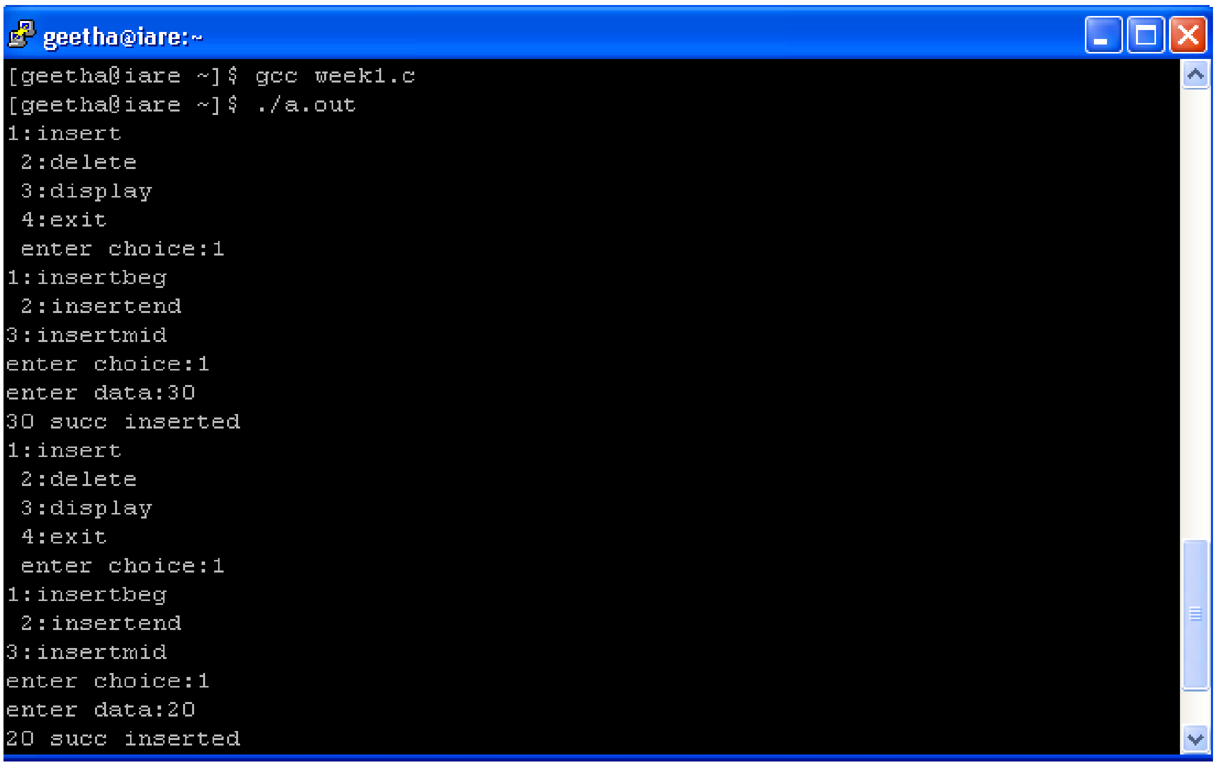
LAB ASSIGNMENT

1. Write a program to insert a node at first , last and at specified position of linked list
2. Write a program to delete a node from first, last and at specified position of linked list

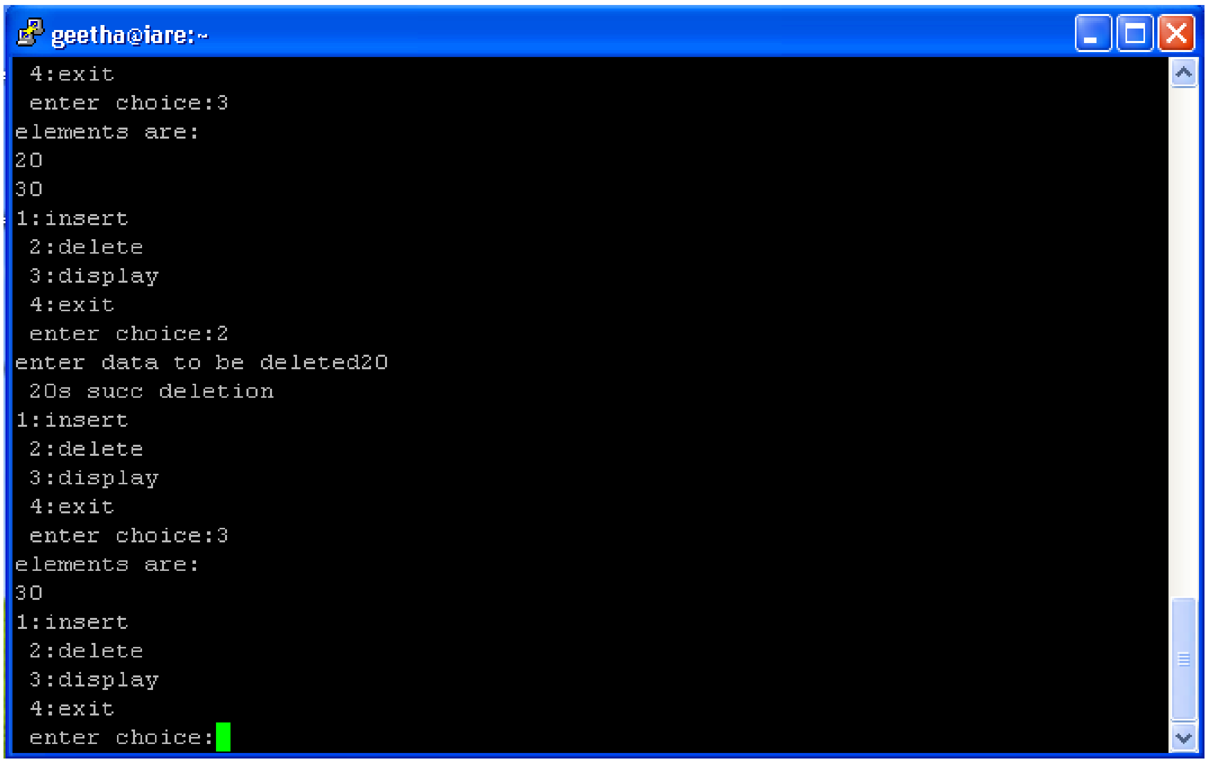
POST LAB QUESTIONS

1. How to represent linked list
2. How will you traverse linked list in reverse order
3. List the advantages and disadvantages of linked list?

INPUT AND OUTPUT



```
[geetha@iare ~]$ gcc week1.c
[geetha@iare ~]$ ./a.out
1:insert
 2:delete
 3:display
 4:exit
 enter choice:1
1:insertbeg
 2:insertend
3:insertmid
enter choice:1
enter data:30
30 succ inserted
1:insert
 2:delete
 3:display
 4:exit
 enter choice:1
1:insertbeg
 2:insertend
3:insertmid
enter choice:1
enter data:20
20 succ inserted
```



```
 4:exit
 enter choice:3
elements are:
20
30
1:insert
 2:delete
 3:display
 4:exit
 enter choice:2
enter data to be deleted20
 20s succ deletion
1:insert
 2:delete
 3:display
 4:exit
 enter choice:3
elements are:
30
1:insert
 2:delete
 3:display
 4:exit
 enter choice:
```

EXPERIMENT 2

OBJECTIVE

1. Create a doubly linked list of integers.
2. Delete a given integer from the above doubly linked list.
3. Display the contents of the above list after deletion.

RESOURCE

Turbo C

PROGRAM LOGIC

1. Create a node using structure
2. Dynamically allocate memory to node
3. Create and add nodes to linked list

PROCEDURE

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE

<u>Program to create a double linked list to inserting, deleting and displaying the contents</u>

```
#include<stdio.h>
#include<stdlib.h>
/*declaring a structure to create a node*/
struct node
{
     struct node *prev;
     int data;
     struct node *next;
};
struct node *start,*nt;
/* inserting nodes into the list*/
/*function to insert values from beginning of the the double linked list*/

void insertbeg(void)
{
    int a;
    struct node *nn,*temp;
/*allocating implicit memory to the node*/
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    a=nn->data;
   if(start==NULL)     /*checking if List is empty*/
  {
      nn->prev=nn->next=NULL;
      start=nn;
  }
  else
 {
     nn->next=start;
     nn->prev=NULL;
     start->prev=nn;
     start=nn;
  }
  printf("%d succ inserted \n",a);
}
```
/*function to insert values from the end of the linked list*/

```
void insertend(void)
{
    int b;
    struct node *nn,*lp;
    nn=(struct node *)malloc(sizeof(struct node));
    printf("enter data:");
    scanf("%d",&nn->data);
    b=nn->data;
    if(start==NULL)
   {
```
/* assigning first node pointer to next nod pointer to delete a data from the starting of the node*/

```
       nn->prev=nn->next=NULL;
       start=nn;
   }
   else
  {
     lp=start;
    while(lp->next!=NULL)
   {
       lp=lp->next;
    }
    nn->prev=lp;
    lp->next=nn;
    nn->next=NULL;
  }
 printf("%d succ inserted\n",b);
}
```
/*function to insert values from the middle of the linked list*/

```
void insertmid(void)
{
    struct node *nn,*temp,*ptemp;
    int x,c;
   if(start==NULL)
   {
       printf("dll is empty\n");
      return;
   }
    printf("enter data before which nn is to be inserted\n");
    scanf("%d",&x);
    if(x==start->data)
    {
       insertbeg();
     }
     ptemp=start;
     temp=start->next;
     while(temp->next!=NULL&&temp->data!=x)
    {
       ptemp=temp;
       temp=temp->next;
    }
    if(temp==NULL)
   {
        printf("%d does not exit\n",x);
      }
      else
     {
/*allocating implicit memory to the node*/

         nn=(struct node *)malloc(sizeof(struct node));
         printf("enter data");
         scanf("%d",&nn->data);
        c=nn->data;
        nn->data;
       nn->prev=ptemp;
       nn->next=temp;
       ptemp->next=nn;
       temp->prev=nn;
       printf("%d succ inserted \n",c);
   }
}
/*end of insertion operation*/
/*deletion operation*/
void deletion()
{
     struct node *pt,*t;
     int x;
     t=pt=start;
     if(start==NULL)
    {
        printf("dll is empty\n");
     }
     printf("enter data to be deleted:");
    scanf("%d",&x);
    if(x==start->data)
   {
     t=start;
     t=t->next;
     free(start);
     start=t;
     start=pt;
   }
   else
  {
     while(t->next!=NULL&&t->data!=x)
    {
       pt=t;    /*logic for traversing*/
       t=t->next;
    }
    if(t->next==NULL&&t->data==x)
   {
     free(t);
     pt->next=NULL;
   }
   else
   {
     if(t->next==NULL&&t->data!=x)
         printf("data not found");
     else
      {
        pt->next=t->next;
        free(t);
      }
   }
   printf("%d is succ deleted\n",x);
  }
}
```
/*end of deletion operation*/
/*display operation*/
```
void display()
{
     struct node *temp;
     if(start==NULL)
     printf("stack is empty ");
     temp=start;
     while(temp->next!=NULL)
    {
        printf("%d",temp->data);
        temp=temp->next;
    }
    printf("%d",temp->data);
 }
```
/*end of display operation*/
/*main program*/
```
int main()
{
     int c,a;
    start=NULL;
    do
   {
        printf("1.insert\n2.delete\n3.display\n4.exit\nenter choice:");
        scanf("%d",&c);
        switch(c)
       {
           case 1:printf("1.insertbeg\n2.insertend\n3.insertmid\nenter choice:");
                      scanf("%d",&a);
                      switch(a)
                      {
                             case 1:insertbeg();
                                        break;
                             case 2:insertend();
                                        break;
                             case 3:insertmid();
                                        break;
                      }
                    break;
          case 2:deletion();
                     break;
         case 3:display();
                     break;
        case 4:printf("program ends\n");
                    break;
        default:printf("wrong choice\n");
                    break;
     }
 }
  while(c!=4);
  return 0;
 }
```

PRE LAB QUESTIONS

1. What is double linked list
2. How to represent a node in double linked list
3. Differentiate between single and double linked list

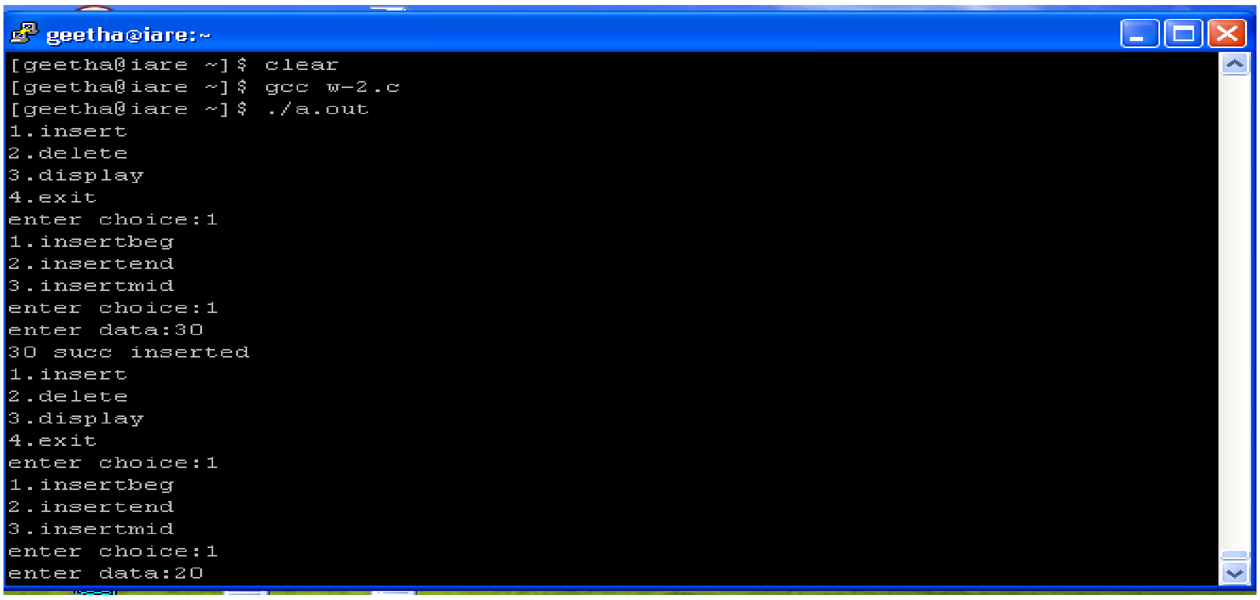
LAB ASSIGNMENT

1. Write a program to insert a node at first , last and at specified position of double linked list
2. Write a program to eliminate duplicates from double linked list
3. Write a program to delete a node from first, last and at specified position of double linked list

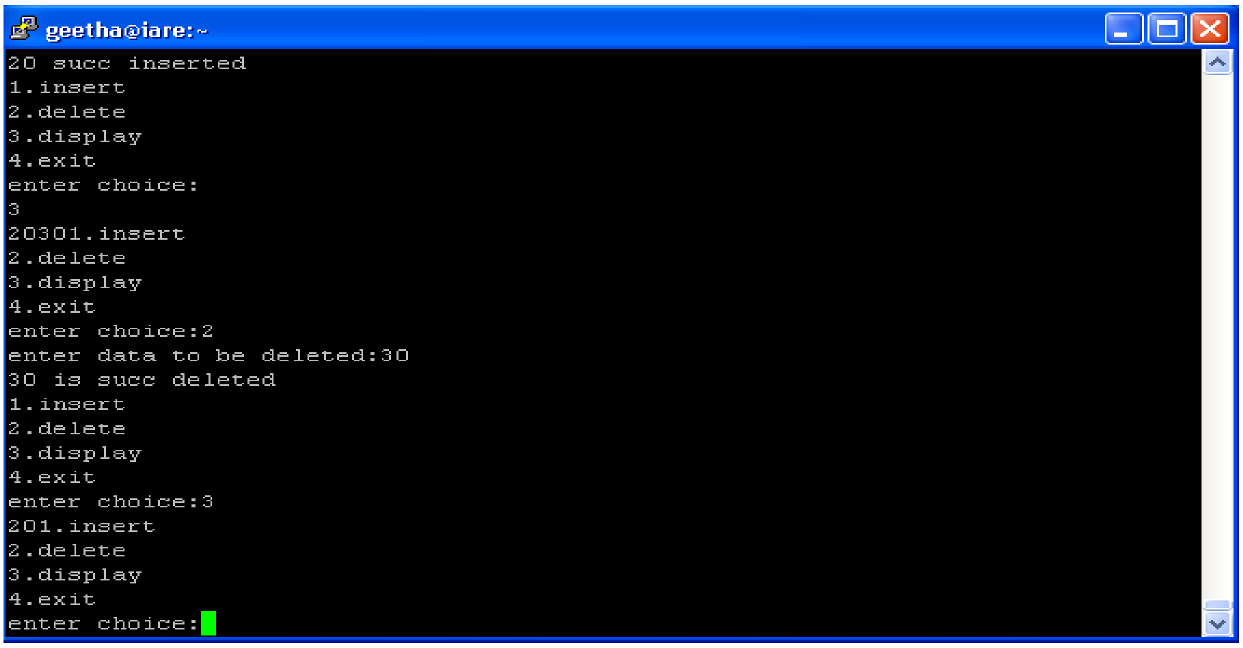
POST LAB QUESTIONS

1. How to represent double linked list
2. How will you traverse double linked list
3. List the advantages of double linked list over single list

INPUT AND OUTPUT



```
geetha@iare:~
[geetha@iare ~]$ clear
[geetha@iare ~]$ gcc w-2.c
[geetha@iare ~]$ ./a.out
1.insert
2.delete
3.display
4.exit
enter choice:1
1.insertbeg
2.insertend
3.insertmid
enter choice:1
enter data:30
30 succ inserted
1.insert
2.delete
3.display
4.exit
enter choice:1
1.insertbeg
2.insertend
3.insertmid
enter choice:1
enter data:20
```



```
geetha@iare:~
20 succ inserted
1.insert
2.delete
3.display
4.exit
enter choice:
3
20301.insert
2.delete
3.display
4.exit
enter choice:2
enter data to be deleted:30
30 is succ deleted
1.insert
2.delete
3.display
4.exit
enter choice:3
201.insert
2.delete
3.display
4.exit
enter choice:
```

EXPERIMENT 3

OBJECTIVE

To convert a given infix expression into its postfix Equivalent, Implement the stack using an array**.**

RESOURCE:

Turbo C

PROGRAM LOGIC

1. Create a stack
2. Read an Circular Q
3. convert infix exression into postfix expression

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

Program to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.

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

struct node {              /* Basic structure of Node */
    int data;
    struct node * prev;
    struct node * next;
}*head, *last;

int main()
{
    int n, data;
    head = NULL;
    last = NULL;

    printf("Enter the total number of nodes in list: ");
    scanf("%d", &n);
    createList(n);                    // function to create double linked list
    displayList();                    // function to display the list

    printf("Enter the position and data to insert new node: ");
    scanf("%d %d", &n, &data);
    insert_position(data, n);     // function to insert node at any position
    displayList();
    return 0;
}
void createList(int n)
{
    int i, data;
    struct node *newNode;
    if(n >= 1){            /* Creates and links the head node */
        head = (struct node *)malloc(sizeof(struct node));
        printf("Enter data of 1 node: ");
        scanf("%d", &data);
        head->data = data;
        head->prev = NULL;
        head->next = NULL;

        last = head;

        for(i=2; i<=n; i++){     /* Creates and links rest of the n-1 nodes */
            newNode = (struct node *)malloc(sizeof(struct node));
            printf("Enter data of %d node: ", i);
            scanf("%d", &data);

            newNode->data = data;
            newNode->prev = last;    //Links new node with the previous node
            newNode->next = NULL;

            last->next = newNode; //Links previous node with the new node
            last = newNode; //Makes new node as last/previous node
        }
        printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
    }
}
void insert_position(int data, int position)
{
    struct node * newNode, *temp;
    if(head == NULL){
        printf("Error, List is empty!\n");
    }
    else{
        temp = head;
        if(temp!=NULL){
            newNode = (struct node *)malloc(sizeof(struct node));

            newNode->data = data;
            newNode->next = temp->next; //Connects new node with n+1th node
            newNode->prev = temp;       //Connects new node with n-1th node

            if(temp->next != NULL)
            {
                  temp->next->prev = newNode; /* Connects n+1th node with new node */
            }
            temp->next = newNode;          /* Connects n-1th node with new node */
            printf("NODE INSERTED SUCCESSFULLY AT %d POSITION\n", position);
        }
        else{
            printf("Error, Invalid position\n");
        }
    }
}
void displayList()
{
    struct node * temp;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        temp = head;
        printf("DATA IN THE LIST:\n");

        while(temp != NULL)
        {
            printf("DATA of %d node = %d\n", n, temp->data);
            n++;

            /* Moves the current pointer to next node */
            temp = temp->next;
        }
    }
}
```

PRE LAB QUESTIONS

1. what is an Circular linked list
2. what are Circular linked list operations

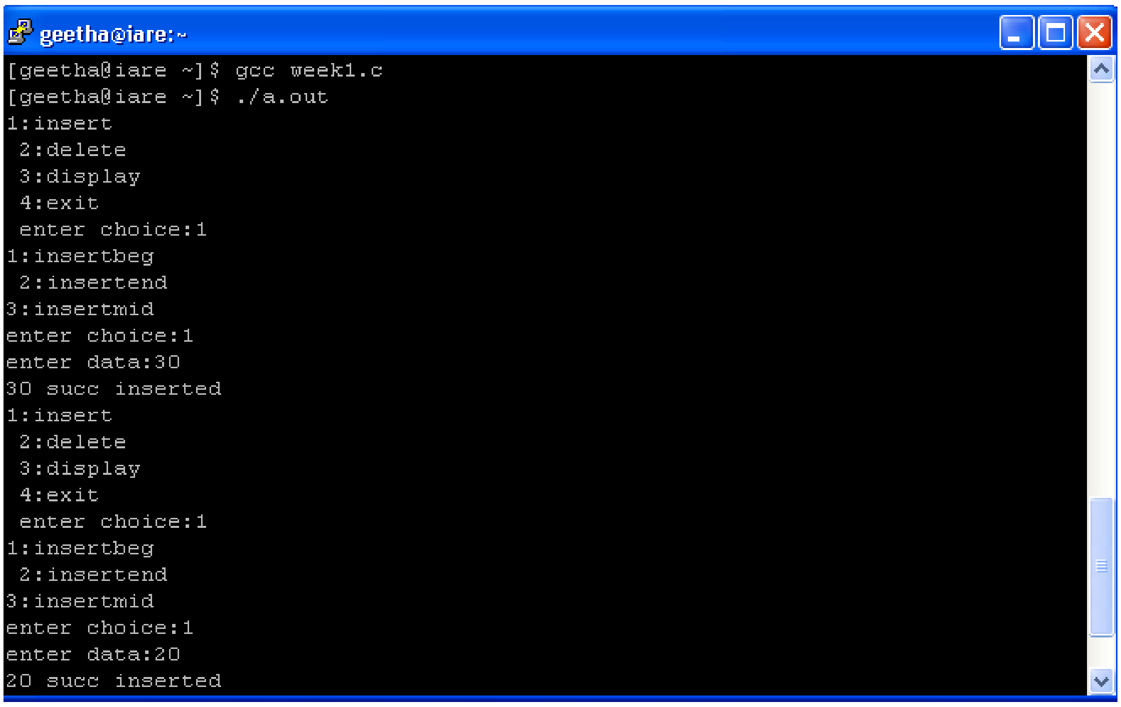
LAB ASSIGNMENT

1. Write a program to insert a node at first , last and at specified position of circular linked list
2. Write a program to eliminate duplicates from circular linked list

POST LAB QUESTIONS

1. how to represent stack
2. why reverse polish notation is required
3. can we evaluate polish notation

INPUT AND OUTPUT



```
[geetha@iare ~]$ gcc week1.c
[geetha@iare ~]$ ./a.out
1:insert
 2:delete
 3:display
 4:exit
 enter choice:1
1:insertbeg
 2:insertend
3:insertmid
enter choice:1
enter data:30
30 succ inserted
1:insert
 2:delete
 3:display
 4:exit
 enter choice:1
1:insertbeg
 2:insertend
3:insertmid
enter choice:1
enter data:20
20 succ inserted
```

EXPERIMENT 4

OBJECTIVE
1. To implement a Stack using arrays.
2. To implement a Stack using Pointers.

RESOURCE:
Turbo C

PROGRAM LOGIC
Stack using arrays
1. Create a Stack
2. Perform all the operation of double ended queue using arrays
3. Display the content of Stack at last.

Stack using Pointers
1. Create a linked list
2. Perform all the operation of double ended queue using linked list
3. Display the content of queue at last.

PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

Write a Programs to implement a Stack operation using Arrays and Pointers

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

struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
```

```
{
   printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
   if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct node));
      top->ptr = NULL;
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct node));
      temp->ptr = top;
      temp->info = data;
      top = temp;
   }
   count++;
}

/* Display stack elements */
void display()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("Stack is empty");
      return;
   }

   while (top1 != NULL)
   {
      printf("%d ", top1->info);
      top1 = top1->ptr;
   }
 }

/* Pop Operation on stack */
void pop()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("\n Error : Trying to pop from empty stack");
      return;
   }
   else
      top1 = top1->ptr;
   printf("\n Popped value : %d", top->info);
   free(top);
   top = top1;
   count--;
}
```

```
/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}
```

INPUT/OUTPUT:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stack
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2

Popped value : 80
Enter choice : 3

Top element : 56
Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
90 78 56
Enter choice : 7

No. of elements in stack : 3
Enter choice : 8

All stack elements destroyed
Enter choice : 4

Stack is empty
Enter choice : 5
```

EXPERIMENT 5

OBJECTIVE

To convert a given infix expression into its postfix Equivalent, Implement the stack using an array**.**

RESOURCE:

Turbo C

PROGRAM LOGIC

1. Create a stack
2. Read an infix expression
3. convert infix exression into postfix expression

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

<u>Program to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.</u>

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 20
char stack[MAX];
int top=1;
char pop();  /*declaration of pop function*/
void push(char item); /*declaration of  push function*/
int prcd(char symbol)  /*checking the precedence*/
{
  switch(symbol) /*assigning values for symbols*/
  {
    case '+':
    case '-': return 2;
    break;
    case '*':
    case '/': return 4;
    break;
    case '^':return 6;
    break;
    case '(':
    case ')':
    case '#':return 1;
   break;
 }
}
int(isoperator(char symbol))  /*assigning  operators*/
{
  switch(symbol)
  {
     case '+':
     case '*':
```

```
     case '-':
    case '/':
    case '^':
    case '(':
    case ')':return 1;
    break;
    default:return 0;
  }
 }
/*converting infix to postfix*/
void convertip(char infix[],char postfix[])
{
 int i,symbol,j=0;
 stack[++top]='#';
 for(i=0;i<strlen(infix);i++)
 {
  symbol=infix[i];
  if(isoperator(symbol)==0)
  {
     postfix[j]=symbol;
      j++;
  }
  else
  {
     if(symbol=='(')
     push(symbol);  /*function call  for pushing elements into the stack*/
     else if(symbol==')')
     {
        while(stack[top]!='(')
       {
          postfix[j]=pop();
          j++;
       }
       pop();     /*function call  for popping elements into the stack*/
   }
   else
   {
       if(prcd(symbol)>prcd(stack[top]))
       push(symbol);
       else
       {
         while(prcd(symbol)<=prcd(stack[top]))
         {
           postfix[j]=pop();
            j++;
         }
        push(symbol);
       }/*end of else loop*/
    }/*end of else loop*/
   } /*end of else loop*/
  }/*end of for loop*/
  While (stack[top]!='#')
  {
    postfix[j]=pop();
     j++;
   }
   postfix[j]='\0'; /*null terminate string*/
 }
```

```
/*main program*/
void main()
 {
     char infix[20],postfix[20];
     printf("enter the valid infix string \n");
     gets(infix);
     convertip(infix,postfix);    /*function call  for converting infix to postfix */
     printf("the corresponding postfix string is:\n");
    puts(postfix);
  }
/*push operation*/
void push(char item)
{
   top++;
   stack[top]=item;
 }
/*pop operation*/
char pop()
{
   char a;
   a=stack[top];
   top--;
   return a;
 }
```

PRE LAB QUESTIONS

1. what is an expression
2. what are infix, prefix and postfix notations
3. what are polish and reverse polish notations
4. which data structure is used for infix to postfix conversion

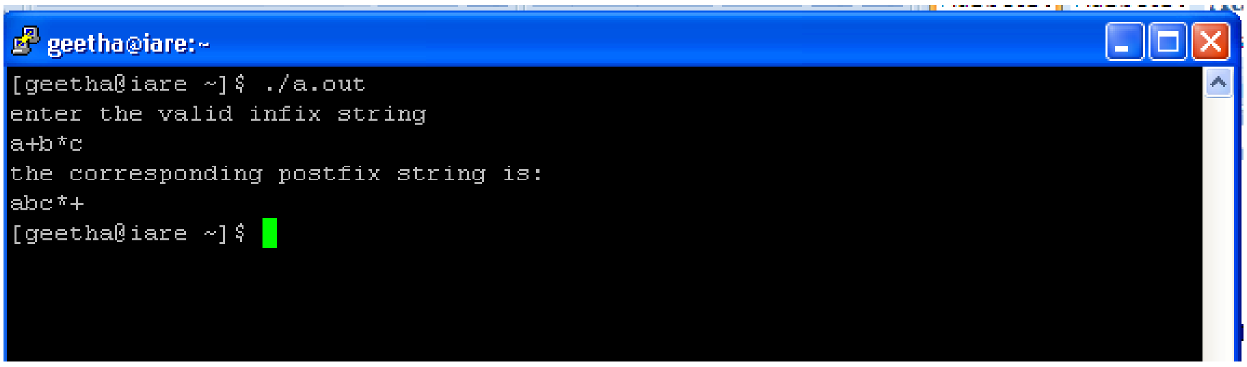
LAB ASSIGNMENT

1. Convert the infix expression (a+b)-(c*d) into post fix form?
2. Convert the following expression A + (B * C) - ((D * E + F) / G) into post form.

POST LAB QUESTIONS

1. how to represent stack
2. why reverse polish notation is required
3. can we evaluate polish notation

INPUT AND OUTPUT



```
[geetha@iare ~]$ ./a.out
enter the valid infix string
a+b*c
the corresponding postfix string is:
abc*+
[geetha@iare ~]$
```

EXPERIMENT 6

OBJECTIVE

Write a Program to implement postfix Evaluation**.**

RESOURCE:

Turbo C

PROGRAM LOGIC

4. Create a stack
5. Read an infix expression
6. convert infix exression into postfix expression

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

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

char stack[100];
int top = -1;

void push(char x)
{
stack[++top] = x;
}

char pop()
{
if(top == -1)
return -1;
else
return stack[top--];
}

int priority(char x)
{
if(x == '(')
return 0;
if(x == '+' || x == '-')
return 1;
if(x == '*' || x == '/')
return 2;
return 0;
}

int main()
{
charexp[100];
char *e, x;
printf("Enter the expression : ");
scanf("%s",exp);
printf("\n");
    e = exp;
```

```
while(*e != '\0')
    {
if(isalnum(*e))
printf("%c ",*e);
else if(*e == '(')
push(*e);
else if(*e == ')')
        {
while((x = pop()) != '(')
printf("%c ", x);
        }
else
        {
while(priority(stack[top]) >= priority(*e))
printf("%c ",pop());
push(*e);
        }
e++;
    }

while(top != -1)
    {
printf("%c ",pop());
}return 0;
}
```

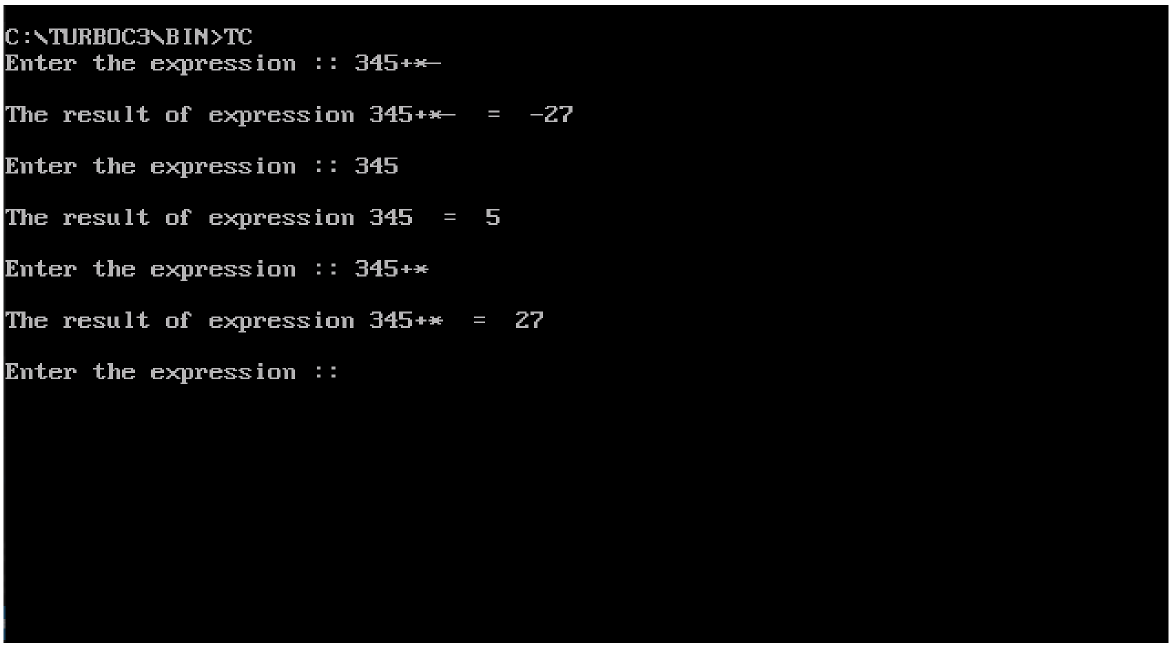
PRE LAB QUESTIONS

1. what is an Postfix expression
2. what are infix, prefix and postfix notations
3. what are polish and reverse polish notations
4. which data structure is used for infix to postfix conversion

LAB ASSIGNMENT

1. Convert the Postfix expression (ab)-(cd)*/ into post fix form?
2. Convert the following expression A BC* - DE F/ G+* into post form.

INPUT AND OUTPUT



```
C:\TURBOC3\BIN>TC
Enter the expression :: 345+*-

The result of expression 345+*-  =  -27

Enter the expression :: 345

The result of expression 345  =  5

Enter the expression :: 345+*

The result of expression 345+*  =  27

Enter the expression ::
```

EXPERIMENT 7

OBJECTIVE

1. To implement a Queue using arrays.
2. To implement a Queue using Pointers.

RESOURCE:

Turbo C

PROGRAM LOGIC

<u>Queue using arrays</u>

1. Create a Queue
2. Perform all the operation of double ended queue using arrays
3. Display the content of Queue at Rear.

<u>Queue using Pointers</u>

4. Create a linked list
5. Perform all the operation of double ended queue using linked list
6. Display the content of queue at Front.

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

```
#include<stdio.h>

#define n 5

int main()

{

int queue[n],ch=1,front,rear=0,i,j=1,x=n;

printf("Queue using Array");

printf("\n1.Rear \n2.Front \n3.Display \n4.Exit");

while(ch)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else
```

```
{
printf("\n Enter no %d:",j++);
scanf("%d",&queue[rear++]);
}
break;
case 2:
if(front==rear)
{
printf("\n Queue is empty");
}
else
{
printf("\n Deleted Element is %d",queue[front++]);
x++;
}
break;
case 3:
printf("\nQueue Elements are:\n ");
if(front==rear)
printf("\n Queue is Empty");
else
{
for(i=front; i<rear; i++)
{
printf("%d",queue[i]);
printf("\n");
}
break;
case 4:
exit(0);
default:

printf("Wrong Choice: please see the options");

}

}

}

return 0;

}
```

PRE LAB QUESTIONS

1. what is queue
2. List the various queue operations
3. What is the advantage of queue

INPUT AND OUTPUT:

```
8           - Rear
9           - Front
10          - Overflow
11          - Underflow
12          - Exit
13          - Display
14          -Queue
15          -Count
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2

Popped value : 80
Enter choice : 3

Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
90 78 56
Enter choice : 7

No. of elements in Queue:3
Enter choice : 8

All queue elementsdestroyed
Enter choice : 4

queue is empty
Enter choice : 5
```

EXPERIMENT 8

OBJECTIVE

1. Sorting the list of integers in ascending order using Bubble sort
2. Sorting the list of integers in ascending order using Selection sort
3. Sorting the list of integers in ascending order using Insertion sort

RESOURCE:

Turbo C

PROGRAM LOGIC

Quick sort

1. Read the elements to be sort
2. Find the proper pivot element
3. Apply quick sort method to sort the remaining elements

Selection sort

1. Read the elements to be sort
2. Select the minimum element
3. Apply the selection sort to sort the remaining elements

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

BUBBLE SORT

```
   #include <stdio.h>

   int main()
   {
   int array[100], n, c, d, swap;

   printf("Enter number of elements\n");
scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)
scanf("%d", &array[c]);

for (c = 0 ; c < n - 1; c++)
  {
for (d = 0 ; d < n - c - 1; d++)
   {
if (array[d] > array[d+1]) /* for decreasing order use '<' instead of '>' */
   {
   swap       = array[d];
     array[d]   = array[d+1];
array[d+1] = swap;
        }
   }
  }

   printf("Sorted list in ascending order:\n");

for (c = 0; c < n; c++)
printf("%d\n", array[c]);

      return 0;
}
```

SELECTION SORT

```
#include<stdio.h>
void sel_sort(int[]);
void main()
{
     int num[5],count;
    printf("enter the five elements to sort:\n");
    for(count=0;count<5;count++)
       scanf("%d",&num[count]);
    sel_sort(num); /*function call*/
    printf("\n\n elements after sorting:\n");
    for(count=0;count<5;count++)
       printf("%d\n",num[count]);
}
/*called function*/
void sel_sort(int num[])
{
    int i,j,min,temp;
    for(j=0;j<5;j++)
   {
      min=j;
      for(i=j;i<5;i++)
      if(num[min]>num[i])
      min=i;
     if(min<5)
    {
         temp=num[j];
         num[j]=num[min];
         num[min]=temp;
    }
     printf("%d\t",num[j]);
  }
}
```

INSERTION SORT

```
#include<stdio.h>
int main()
{
int i, j, count, temp, number[25];

printf("How many numbers u are going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
for(i=0;i<count;i++)
scanf("%d",&number[i]);
for(i=1;i<count;i++){
temp=number[i];
      j=i-1;
while((temp<number[j])&&(j>=0)){
number[j+1]=number[j];
         j=j-1;
      }
number[j+1]=temp;
   }
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);

return 0;
}
```

PRE LAB QUESTIONS

1. what is sorting
2. List the various sorting algorithms
3. What is the advantage of selection sort
4. How to find pivot element in quick sort
5. Is quick sort is stable algorithm

LAB ASSIGNMENT

1 . Rearrange the following numbers using Bubble sort procedure. 42, 12, 18, 98, 67, 83, 8, 10, 71

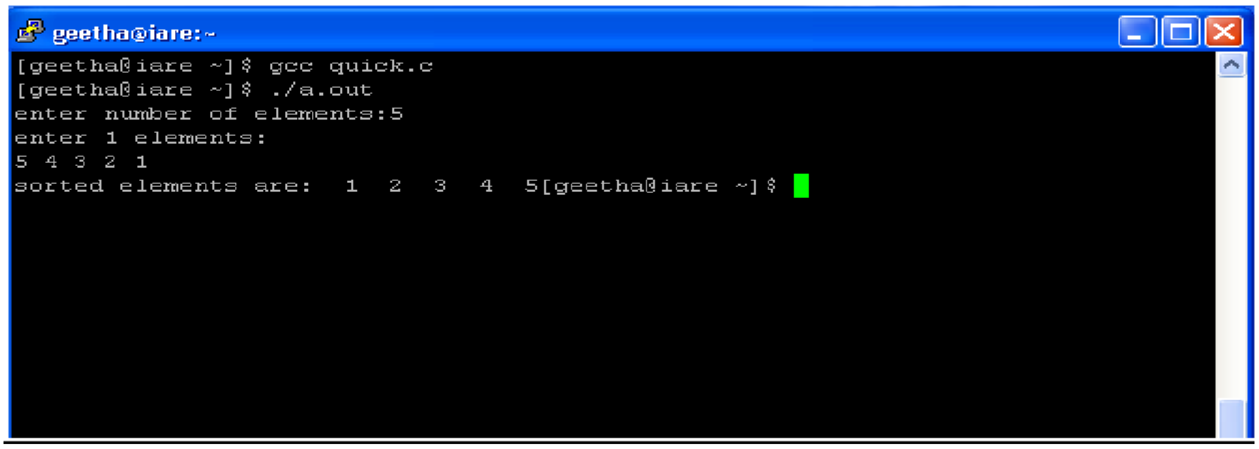
2.Apply the selection sort on the following elements21,11,5,78,49, 54,72,88

POST LAB QUESTIONS

6. What is the time complexity of selection sort
7. What is the time complexity of quick sort
8. Why sorting is required
9. Is selection sort is stable
10. What is the worst case for quick sort

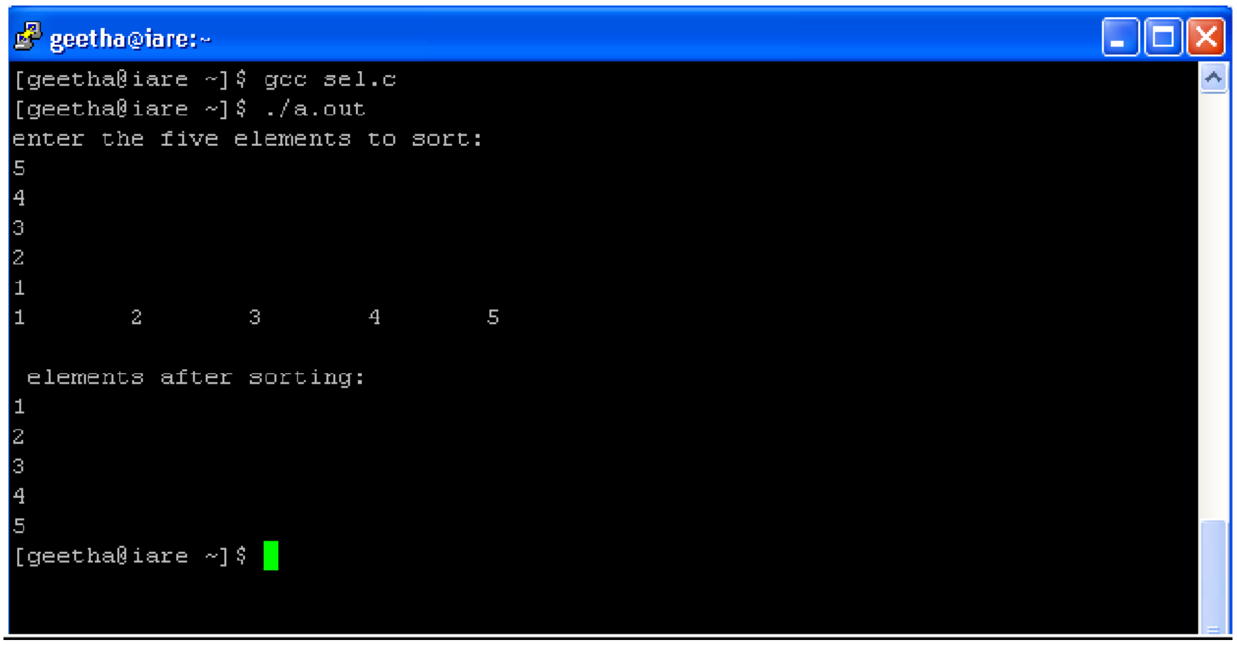
8. 9 INPUT AND OUTPUT

<u>QUICK SORT</u>



```
[geetha@iare ~]$ gcc quick.c
[geetha@iare ~]$ ./a.out
enter number of elements:5
enter 1 elements:
5 4 3 2 1
sorted elements are:  1  2  3  4  5[geetha@iare ~]$
```

SELECTION SORT



```
[geetha@iare ~]$ gcc sel.c
[geetha@iare ~]$ ./a.out
enter the five elements to sort:
5
4
3
2
1
1       2       3       4       5

 elements after sorting:
1
2
3
4
5
[geetha@iare ~]$ 
```

EXPERIMENT 9

OBJECTIVE

1. Sorting the list of integers in ascending order using Merge sort
2. Sorting the list of integers in ascending order using Quick sort

RESOURCE:

Turbo C

PROGRAM LOGIC

Quick sort

1. Read the elements to be sort
2. Find the proper pivot element
3. Apply quick sort method to sort the remaining elements

Selection sort

4. Read the elements to be sort
5. Select the minimum element
6. Apply the selection sort to sort the remaining elements

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

MERGE SORT :

```
#include<stdio.h>
#include<conio.h>
#define MAX_SIZE 5
void merge_sort(int, int);
void merge_array(int, int, int, int);
int arr_sort[MAX_SIZE];

int main() {
  int i;
  printf("Simple Merge Sort Example - Functions and Array\n");
  printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);
  for (i = 0; i < MAX_SIZE; i++)
  scanf("%d", &arr_sort[i]);
  printf("\nYour Data   :");
  for (i = 0; i < MAX_SIZE; i++) {
    printf("\t%d", arr_sort[i]);
  }
  merge_sort(0, MAX_SIZE - 1);
  printf("\n\nSorted Data :");
  for (i = 0; i < MAX_SIZE; i++) {
    printf("\t%d", arr_sort[i]);
  }
  getch();
}
void merge_sort(int i, int j) {
  int m;
  if (i < j) {
    m = (i + j) / 2;
    merge_sort(i, m);
    merge_sort(m + 1, j);
        merge_array(i, m, m + 1, j);
  }
}
void merge_array(int a, int b, int c, int d) {
  int t[50];
  int i = a, j = c, k = 0;
  while (i <= b && j <= d) {
    if (arr_sort[i] < arr_sort[j])
      t[k++] = arr_sort[i++];
    else
      t[k++] = arr_sort[j++];
  }
  while (i <= b)
    t[k++] = arr_sort[i++];
  while (j <= d)
    t[k++] = arr_sort[j++];
  for (i = a, j = 0; i <= d; i++, j++)
    arr_sort[i] = t[j];
}
```

QUICK SORT:

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
```

```
     j=last;
     while(i<j){
       while(number[i]<=number[pivot]&&i<last)
       i++;
       while(number[j]>number[pivot])
       j--;
       if(i<j){
         temp=number[i];
         number[i]=number[j];
         number[j]=temp;
       }
     }
     temp=number[pivot];
     number[pivot]=number[j];
     number[j]=temp;
     quicksort(number,first,j-1);
     quicksort(number,j+1,last);
   }
}
int main(){
   int i, count, number[25];
   printf("How many elements are u going to enter?: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
   scanf("%d",&number[i]);
   quicksort(number,0,count-1);
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
   printf(" %d",number[i]);
   return 0;
}
```

PRE LAB QUESTIONS

1. what is sorting
2. List the various sorting algorithms
3. What is the advantage of selection sort
4. How to find pivot element in quick sort
5. Is quick sort is stable algorithm

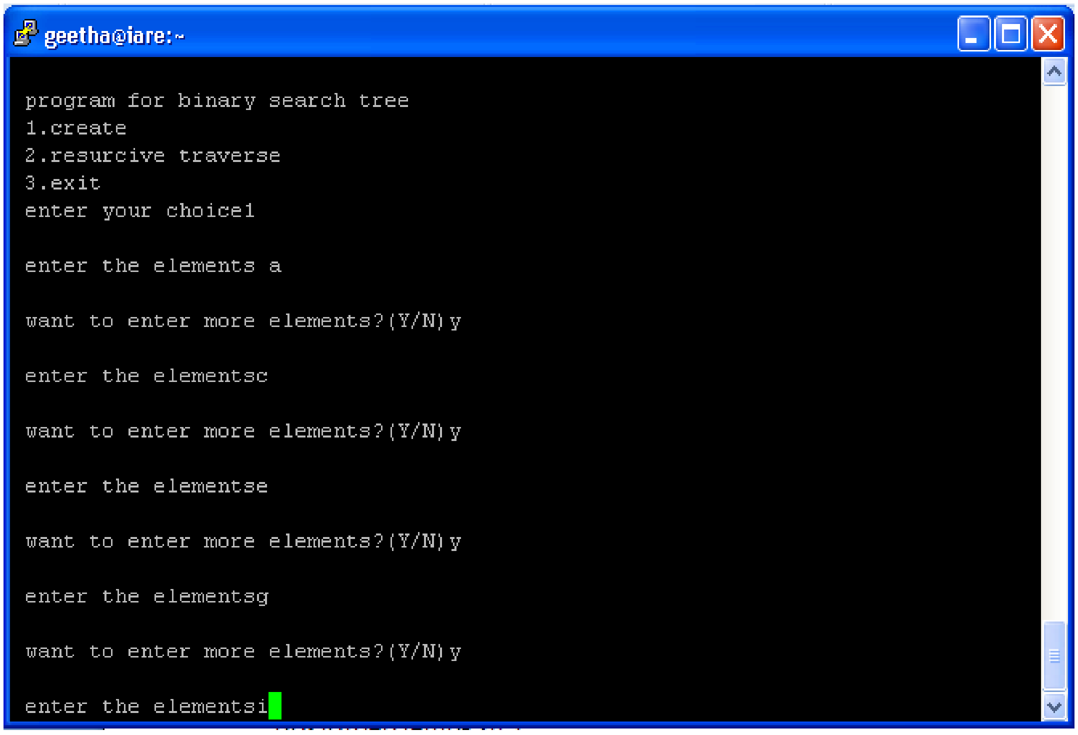
LAB ASSIGNMENT

1 . Rearrange the following numbers using Merge sort procedure. 42, 12, 18, 98, 67, 83, 8, 10, 71

2.Apply the Quick sort on the following elements21,11,5,78,49, 54,72,88

POST LAB QUESTIONS

11. What is the time complexity of merge sort
12. What is the time complexity of quick sort
13. Why sorting is required
14. What is the worst case for quick sort



```
program for binary search tree
1.create
2.resurcive traverse
3.exit
enter your choice1

enter the elements a

want to enter more elements?(Y/N)y

enter the elementsc

want to enter more elements?(Y/N)y

enter the elementse

want to enter more elements?(Y/N)y

enter the elementsg

want to enter more elements?(Y/N)y

enter the elementsi
```

EXPERIMENT 10

OBJECTIVE

1. Program to implement Linear Search
2. Program to implement Binary Search

RESOURCE:

Turbo C

PROGRAM LOGIC

1. Create a linear and binary search with the property binary search

SOURCE CODE:

LINEAR SEARCH:

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
 {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

BINARY SEARCH:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i, arr[10], search, first, last, middle;
    printf("Enter 10 elements (in ascending order): ");
```

```
        for(i=0; i<10; i++)
            scanf("%d", &arr[i]);
        printf("\nEnter element to be search: ");
        scanf("%d", &search);
        first = 0;
        last = 9;
        middle = (first+last)/2;
        while(first <= last)
        {
            if(arr[middle]<search)
                first = middle+1;
            else if(arr[middle]==search)
            {
                printf("\nThe number, %d found at Position %d", search, middle+1);
                break;
            }
            else
                last = middle-1;
            middle = (first+last)/2;
        }
        if(first>last)
            printf("\nThe number, %d is not found in given Array", search);
        getch();
        return 0;
    }
```

INPUT AND OUTPUT

```
How many Elements you want to
Enter?
6
Enter the Key element to search! 56
Enter the elements: 23
 67
 89
 56
 19
 12
Element found in position 4


How many Elements you
enter? 6
Enter the Key Element to
search

 8

Enter the Elements: 2

 4

6

8

10

12

Element Found in 4 positions.
```

EXPERIMENT 11

OBJECTIVE

1. To create a binary search tree of characters.
2. Traverse the above Binary search tree recursively in Post order.

RESOURCE:

Turbo C

PROGRAM LOGIC

2. Create binary tree with the property binary search tree
3. Visit the tree in post order
4. Visit in the order left, right, root
5. Display the visited nodes

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

/*program for creating and traversing the binary search tree*/

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST
{
    char d;
/*declaring a structure to create a node*/

   struct BST *lc,*rc;
}node;
/*main program*/
void main()
{
   int choice;
   char ans='N';
   int key;
    node *nn,*root,*parent;
    root=NULL;
    printf("\n program for binary search tree");
    do
   {
     printf("\n 1.create");
     printf("\n 2.resurcive traverse");
     printf("\n 3.exit");
     printf("\n enter your choice");
     scanf("%d",&choice);
     switch(choice)  /*switch case*/
    {
      case 1:
                        do
                        {
                               nn=(node *)malloc(sizeof(node));
```

```
                                          nn->rc=NULL;
                                          scanf(" %c",&nn->d);
                                          if(root==NULL)
                                          root=nn;
                                         else
                                             insert(root,nn);
                                          printf("\n want to enter more elements?(Y/N)");
                                          scanf(" %c",&ans);
                                   }while(ans=='y');
                                  break;
        case 2:
                                   if(root==NULL)
                                  printf("tree is not created");
                                  else
                                  {
                                         printf("\n the inorder display:");
                                         inorder(root);
                                        printf("\n the preorder display:");
                                        preorder(root);
                                       printf("\n the postorder display:");
                                       postorder(root);
                                  }
                                  break;
       } /*end of switch case*/
    }while(choice!=3);
 }
/*insertion operation*/
void insert(node *root,node *nn)
{
    int c,d;
    c=nn->d;
    d=root->d;
    if(c<d)
   {
      if(root->lc==NULL)
      root->lc=nn;
     else
       insert(root->lc,nn);
   }
}
/*inorder traversal*/
void inorder(node *temp)
{
    if(temp!=NULL)
    {
      inorder(temp->lc);
      printf(" %c",temp->d);
      inorder(temp->rc);
    }
}
/*preorder traversal*/
void preorder(node *temp)
{
   if(temp!=NULL)
   {
     printf(" %c",temp->d);
     preorder(temp->lc);
     preorder(temp->rc);
   }
}
/*postorder traversal*/
void postorder(node *temp)
{
   if(temp!=NULL)
   {
     postorder(temp->lc);
     postorder(temp->rc);
     printf(" %c",temp->d);
   }
}
```

PRE LAB QUESTIONS

1. Differentiate between BST and complete BST
2. What are the properties of BST
3. How many nodes will be there in given nth level.

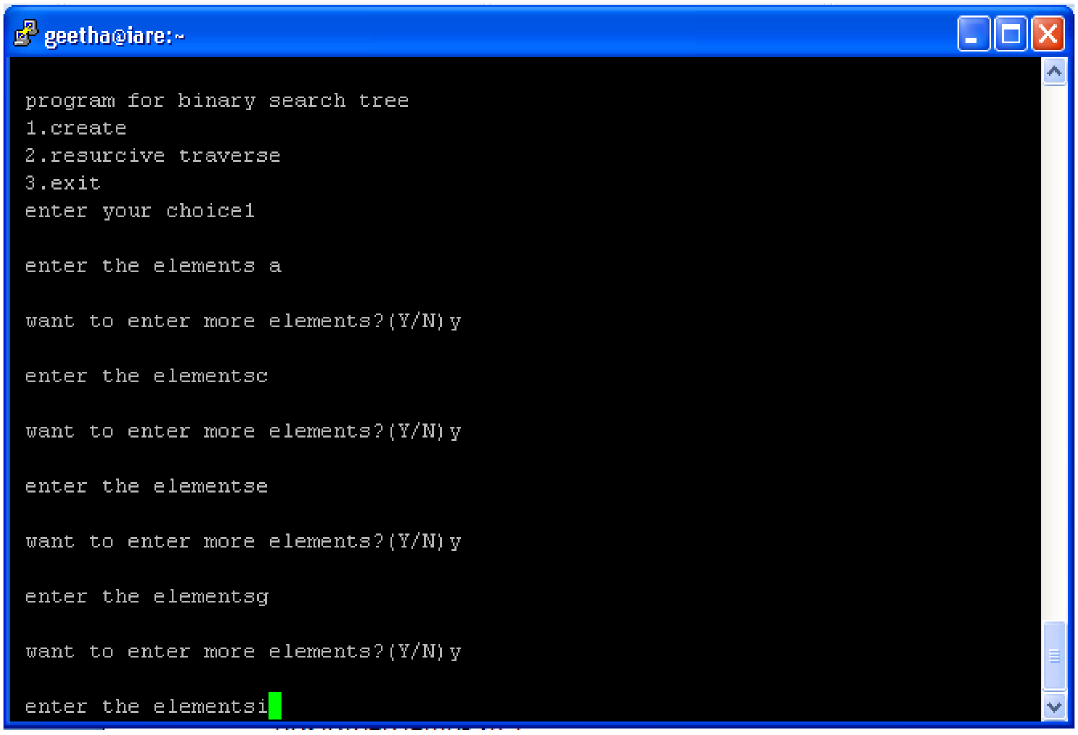
LAB ASSIGNMENT

1. Construct a binary search tree for the following 80, 40, 75, 30, 20, 90, 50

POST LAB QUESTIONS

1. List the various tree traversal techniques are there
2. Write the necessary condition for inserting element into BST
3. List the applications of BST

INPUT AND OUTPUT



```
program for binary search tree
1.create
2.resurcive traverse
3.exit
enter your choice1

enter the elements a

want to enter more elements?(Y/N)y

enter the elementsc

want to enter more elements?(Y/N)y

enter the elementse

want to enter more elements?(Y/N)y

enter the elementsg

want to enter more elements?(Y/N)y

enter the elementsi
```

EXPERIMENT 12

A: Write C programs for implementing the following graph traversal algorithm for Depth first traversal

OBJECTIVE

To traverse graph

RESOURCE:
Turbo C

PROGRAM LOGIC

1. Take the graph as a input
2. Start at some vertex and traverse it using DFS
3. Apply the above procedure for all nodes

PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

SOURCE CODE:

DFS

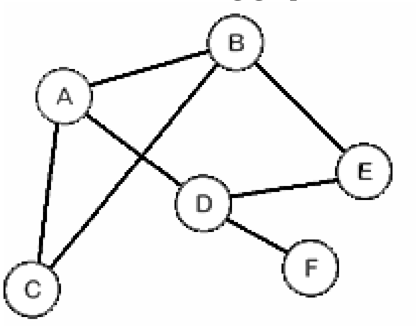
```
#include <stdio.h>
void dfs(int);
int g[10][10],visited[10],n,vertex[10];
void main()
{
    int i,j;
    printf("enter number of vertices:");
     scanf("%d",&n);
     printf("enter the val of vertex:");
     for(i=0;i<n;i++)
                scanf("%d",&vertex[i]);
   printf("\n enter adjecency matrix of the graph:");
   for(i=0;i<n;i++)
                 for(j=0;j<n;j++)
                scanf("%d",&g[i][j]);
  for(i=0;i<n;i++)
   visited[i]=0;
   dfs(0);
}
void dfs(int i)
{
   int j;
    printf("%d",vertex[i]);
 visited[i]=1;         for(j=0;j<n;j++)
 if(!visited[j]&&g[i][j]==1)
   dfs(j);
}
```

PRE LAB QUESTIONS

1. What is graph
2. List various way of representations of graph
3. How many graph traversal algorithms are there

LAB ASSIGNMENT

1. Find DFS traversal of the following graph

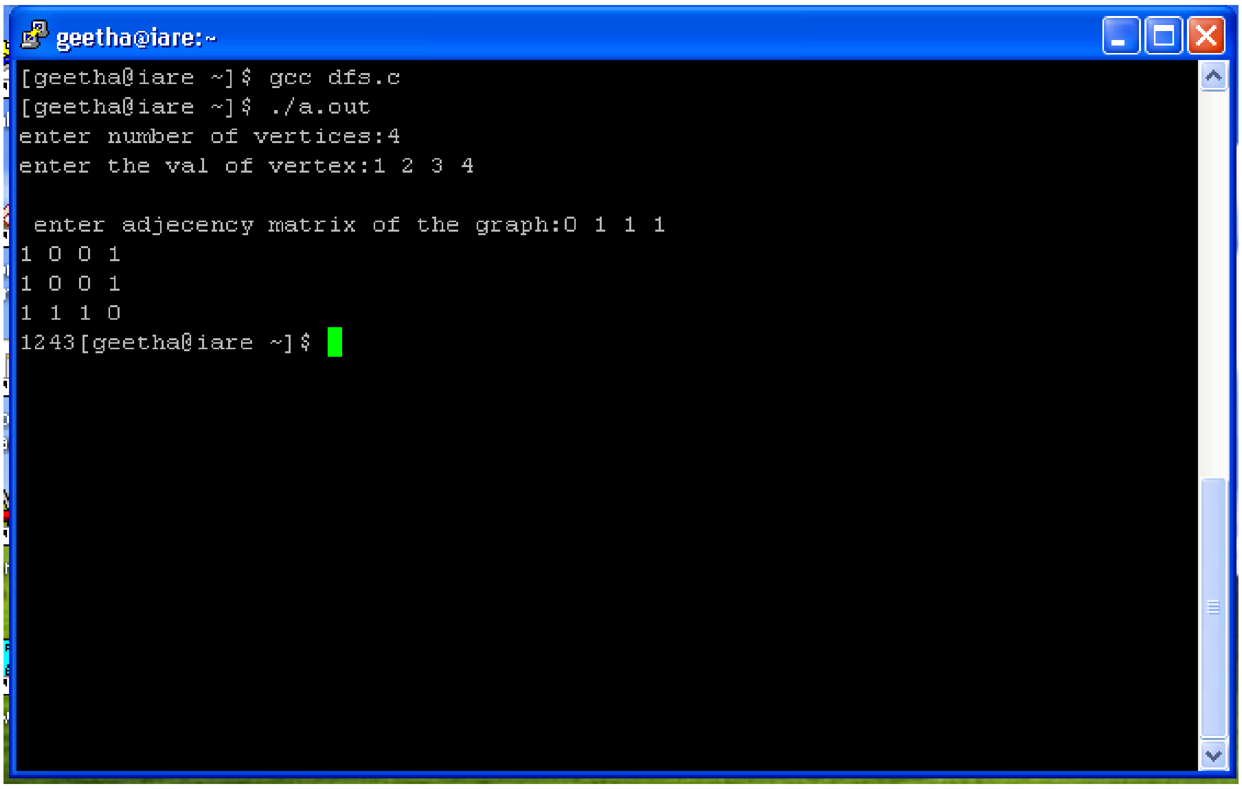


2. Deduce the time complexity of DFS algorithm

POST LAB QUESTIONS

1. Applications of graph traversals
2. Define minimum spanning tree
3. What is the time complexity of DFS

12. 9 INPUT AND OUTPUT



```
[geetha@iare ~]$ gcc dfs.c
[geetha@iare ~]$ ./a.out
enter number of vertices:4
enter the val of vertex:1 2 3 4

 enter adjecency matrix of the graph:0 1 1 1
1 0 0 1
1 0 0 1
1 1 1 0
1243[geetha@iare ~]$
```

CONTENT BEYOND SYLLABUS

EXPERIMENT 1

OBJECTIVE

*Write a C Program to check whether two given lists are containing the same data.

SOURCE CODE

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

void feedmember(struct node **);
int compare (struct node *, struct node *);
void release(struct node **);

int main()  {
   struct node *p = NULL;
   struct node *q = NULL;
   int result;

   printf("Enter data into first list\n");
   feedmember(&p);
   printf("Enter data into second list\n");
   feedmember(&q);
   result = compare(p, q);
   if (result == 1)       {
      printf("The 2 list are equal.\n");
   }
   else     {
      printf("The 2 lists are unequal.\n");
   }
   release (&p);
   release (&q);

   return 0;
}

int compare (struct node *p, struct node *q) {
   while (p != NULL && q != NULL)
   {
      if (p->num != q-> num)          {
         return 0;
      }
      else
      {
         p = p->next;
         q = q->next;
      }
   }
   if (p != NULL || q != NULL)      {
      return 0;
   }
```

```
    else    {
        return 1;
    }
}

void feedmember (struct node **head)
{
    int c, ch;
    struct node *temp;

    do      {
        printf("Enter number: ");
        scanf("%d", &c);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = c;
        temp->next = *head;
        *head = temp;
        printf("Do you wish to continue [1/0]: ");
        scanf("%d", &ch);
    }while (ch != 0);
    printf("\n");
}

void release (struct node **head) {
    struct node *temp = *head;

    while ((*head) != NULL)
    {
        (*head) = (*head)->next;
        free(temp);
        temp = *head;
    }
}
```

INPUT/ OUTPUT

```
Enter data into first list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 3
Do you wish to continue [1/0]: 1
Enter number: 28
Do you wish to continue [1/0]: 1
Enter number: 9
Do you wish to continue [1/0]: 0

Enter data into second list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 3
Do you wish to continue [1/0]: 1
Enter number: 28
Do you wish to continue [1/0]: 1
Enter number: 9
Do you wish to continue [1/0]: 0

The 2 list are equal.
```

EXPERIMENT 2

OBJECTIVE

*Write a C program to find the largest element in a given doubly linked list.

SOURCE CODE

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

struct node
{
   int num;
   struct node *next;
   struct node *prev;
};

void create(struct node **);
int max(struct node *);
void release(struct node **);

int main()
{
   struct node *p = NULL;
   int n;

   printf("Enter data into the list\n");
   create(&p);
   n = max(p);
   printf("The maximum number entered in the list is %d.\n", n);
   release (&p);

   return 0;
}

int max(struct node *head)
{
   struct node *max, *q;

   q = max = head;
   while (q != NULL)
   {
      if (q->num > max->num)
      {
         max = q;
      }
      q = q->next;
   }

   return (max->num);
}

void create(struct node **head)
{
   int c, ch;
   struct node *temp, *rear;
```

```
    do
    {
        printf("Enter number: ");
        scanf("%d", &c);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = c;
        temp->next = NULL;
        temp->prev = NULL;
        if (*head == NULL)
        {
            *head = temp;
        }
        else
        {
            rear->next = temp;
            temp->prev = rear;
        }
        rear = temp;
        printf("Do you wish to continue [1/0]: ");
        scanf("%d", &ch);
    } while (ch != 0);
    printf("\n");
}

void release(struct node **head)
{
    struct node *temp = *head;
    *head = (*head)->next;
    while ((*head) != NULL)
    {
        free(temp);
        temp = *head;
        (*head) = (*head)->next;
    }
}
```

INPUT/ OUTPUT:

```
Enter data into the list
Enter number: 12
Do you wish to continue [1/0]: 1
Enter number: 7
Do you wish to continue [1/0]: 1
Enter number: 23
Do you wish to continue [1/0]: 1
Enter number: 4
Do you wish to continue [1/0]: 1
Enter number: 1
Do you wish to continue [1/0]: 1
Enter number: 16
Do you wish to continue [1/0]: 0

The maximum number entered in the list is 23.
```

EXPERIMENT 3

OBJECTIVE

*Write a C program to reverse the elements in the stack using recursion.

SOURCE CODE

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

struct node
{
    int a;
    struct node *next;
};

void generate(struct node **);
void display(struct node *);
void stack_reverse(struct node **, struct node **);
void delete(struct node **);

int main()
{
    struct node *head = NULL;

    generate(&head);
    printf("\nThe sequence of contents in stack\n");
    display(head);
    printf("\nInversing the contents of the stack\n");
    if (head != NULL)
    {
        stack_reverse(&head, &(head->next));
    }
    printf("\nThe contents in stack after reversal\n");
    display(head);
    delete(&head);

    return 0;
}

void stack_reverse(struct node **head, struct node **head_next)
{
    struct node *temp;

    if (*head_next != NULL)
    {
         temp = (*head_next)->next;
        (*head_next)->next = (*head);
        *head = *head_next;
        *head_next = temp;
        stack_reverse(head, head_next);
    }
}

void display(struct node *head)
{
    if (head != NULL)
    {
        printf("%d ", head->a);
        display(head->next);
    }
}

void generate(struct node **head)
{
    int num, i;
    struct node *temp;

    printf("Enter length of list: ");
    scanf("%d", &num);
    for (i = num; i > 0; i--)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        temp->a = i;
        if (*head == NULL)
        {
            *head = temp;
            (*head)->next = NULL;
        }
        else
        {
            temp->next = *head;
            *head = temp;
        }
    }
}

void delete(struct node **head)
{
    struct node *temp;
    while (*head != NULL)
    {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}
```

INPUT/OUTPUT:

```
Enter length of list: 10

The sequence of contents in stack
1 2 3 4 5  6 7 8  9 10

Reversing the contents of the stack

The contents in stack after reversal
10 9 8 7  6 5 4  3 2 1
```

EXPERIMENT 4

OBJECTIVE

*Write a C program to implement stack using linked list.

SOURCE CODE

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

struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}

/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;
    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}
```

INPUT/OUTPUT:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stack
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2

Popped value : 80
Enter choice : 3

Top element : 56
Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
90 78 56
Enter choice : 7

No. of elements in stack : 3
Enter choice : 8

All stack elements destroyed
Enter choice : 4

Stack is empty
Enter choice : 5
```

EXPERIMENT 5

OBJECTIVE

*Write a C program to count the number of nodes in the binary search tree.

SOURCE CODE

```
#include < stdio.h >
#include < conio.h >
#include < alloc.h >
#define new_node (struct node*)malloc(sizeof (struct node))

struct node
{
		int data;
		struct node *lchild;
		struct node *rchild;
};

struct node *create_bin_rec();
void print_bin_pre_rec(struct node *t);
void cnt_nodes(struct node *t, int *l, int *nl);

void main()
{
		struct node *root;
		int leaf,nonleaf;
		clrscr();
		printf("\nCreate a binary tree \n");
		root = create_bin_rec();
		printf("\n Binary tree is ");
		print_bin_pre_rec(root);
		leaf = nonleaf = 0;
		cnt_nodes(root,&leaf,&nonleaf);
		printf("\n Total no. of leaf nodes are : %d ",leaf);
		printf("\n Total no. of nonleaf nodes are : %d ",nonleaf);
		printf("\n Total no. of nodes are : %d ", (leaf+nonleaf));

} // main

struct node *create_bin_rec()
{
		int  data;
		struct node *t;
		printf("\nData ( -1 to exit ) : ");
		scanf("%d",&data);
		if( data == -1)
				return(NULL);
		else
		{
				t = new_node;
				t->data = data;
				t->lchild =create_bin_rec();
				t->rchild =create_bin_rec();
				return(t);
		} //else
} // create

void print_bin_pre_rec(struct node *t)
{
        if( t != NULL)
        {
                printf("%4d",t->data);
                print_bin_pre_rec(t->lchild);
                print_bin_pre_rec(t->rchild);
        } // if
} // print bin pre rec

void cnt_nodes(struct node *t, int *l, int *nl)
{
        if( t != NULL)
        {
                if( t->lchild == NULL && t->rchild == NULL)
                        (*l)++;
                else
                        (*nl)++;
                        cnt_nodes(t->lchild,l,nl);
                        cnt_nodes(t->rchild,l,nl);
        } // if
} // cnt nodes
```

<u>INPUT OUTPUT</u>

```
Create binary tree
Data (-1 to exit) 10
Data (-1 to exit) 20
Data (-1 to exit)  -1

Binary tree is 10 20
Total no. of leaf nodes are 1
Total no. of non leaf nodes are 1
Total no. of nodes are 2
```

EXPERIMENT 6

OBJECTIVE

*Write a C program to sort an array of integers in ascending order using radix sort.

SOURCE CODE

```
#include <stdio.h>
int min = 0, count = 0, array[100] = {0}, array1[100] = {0};

void main()
{
   int k, i, j, temp, t, n;

   printf("Enter size of array :");
   scanf("%d", &count);
   printf("Enter elements into array :");
   for (i = 0; i < count; i++)
   {
      scanf("%d", &array[i]);
      array1[i] = array[i];
   }
   for (k = 0; k < 3; k++)
   {
      for (i = 0; i < count; i++)
      {
         min = array[i] % 10;        /* To find minimum lsd */
         t = i;
         for (j = i + 1; j < count; j++)
         {
            if (min > (array[j] % 10))
            {
               min = array[j] % 10;
               t = j;
            }
         }
         temp = array1[t];
         array1[t] = array1[i];
         array1[i] = temp;
         temp = array[t];
         array[t] = array[i];
         array[i] = temp;

      }
      for (j = 0; j < count; j++)        /*to find MSB */
         array[j] = array[j] / 10;
   }
   printf("Sorted Array (lSdradix sort) : ");
   for (i = 0; i < count; i++)
      printf("%d ", array1[i]);
}
```

INPUT/ OUTPUT:

```
/* Average Case */
Enter size of array :7
Enter elements into array :170
45
90
75
802
24
2
Sorted Array (ladradix sort) : 2 24 45 75 90 170 802

/*Best case */
Enter size of array :7
Enter elements into array :22
64
121
78
159
206
348
Sorted Array (ladradix sort) : 22 64 78 159 121 206 348

   /* Worst case */
Enter size of array :7
Enter elements into array :985
27
64
129
345
325
091
Sorted Array (ladradix sort) : 27 64 91 129 325 345 985
```

EXPERIMENT 7

OBJECTIVE

*Write a C program to sort a given list of strings.

SOURCE CODE

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main() {
char *str[5], *temp;
 int i, j, n;
 printf("\nHow many names do you want to have?");
 scanf("%d", &n);
 for (i = 0; i < n; i++) {
      printf("\nEnter the name %d: ", i);
      flushall();
      gets(str[i]);
}
for (i = 0; i < n; i++) {
     for (j = 0; j < n - 1; j++) {
        if (strcmp(str[j], str[j + 1]) > 0) {
            strcpy(temp, str[j]);
            strcpy(str[j], str[j + 1]);
            strcpy(str[j + 1], temp);
} }  }
printf("\nSorted List : ");
for (i = 0; i < n; i++)
      puts(str[i]);
   return (0);
}
```

INPUT /OUTPUT:

```
Enter any  strings :
pri
pra
pru
pry
prn

Strings in order are :
pra
pri
prn
pru
pry
```