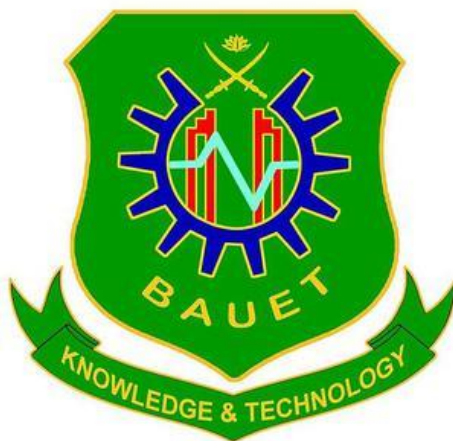


# Bangladesh Army University of Engineering & Technology (BAUET)

Qadirabad, Natore 6431, Bangladesh



Department of Information and Communication Engineering (ICE)

## Data Structures and Algorithms Lab Manual

Course Code	ICE-2112
Course Title	Data Structures and Algorithms Sessional
Credit Hours	1.5

Prepared by	Verified By
Nasirul Mumenin Lecturer, Dept. of ICE	Dr. Md. Rubel Bashar Head, Dept. of ICE

---

Knowledge and Technology

---

## Table of Content

Si.	Content	Page
1	General Guidelines & Safety Instructions	4
2	Course Description	5
3	Course Objectives	5
4	Statement of Course Outcomes	5
5	Assessment of Course Outcomes	6
6	Mark Distribution and Assessment Criteria	6
7	Summary of Lab Report	7
8	Format of Lab Report	9
9	Instruction for Lab Report Writing	9
S. No.	List of Experiments	Page No.
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.	1
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.	6
3	Write a C program that uses stack operations to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.	11
4	Write C programs to implement a double ended queue ADT using i) Array and ii) Doubly linked list respectively.	14
5	Write a C program that uses functions to perform the following: a) Create a binary search tree of characters. b) Traverse the above Binary search tree recursively in Postorder.	23
6	Write a C program that uses functions to perform the following: a) Create a binary search tree of integers. b) Traverse the above Binary search tree non recursively in inorder.	26
7	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: a) Insertion sort b) Merge sort	29
8	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: a) Quick sort      b) Selection sort	32
9	Write a C program to perform the following operation: i) Insertion into a B-tree ii) Implementing Heap sort algorithm for sorting a given list of integers in ascending order.	35
10	Write a C program to implement all the functions of a dictionary (ADT) using hashing.	41
11	Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.	46
12	Write C programs for implementing the following graph traversal algorithms: a) Depth first traversal b) Breadth first traversal	49

Content Beyond Syllabi		
1	*Write a C Program to check whether two given lists are containing the same data.	52
2	*Write a C program to find the largest element in a given doubly linked list.	54
3	*Write a C program to reverse the elements in the stack using recursion.	56
4	*Write a C program to implement stack using linked list.	58
5	*Write a C program to count the number of nodes in the binary search tree.	62
6	*Write a C program to sort an array of integers in ascending order using radix sort.	64
7	*Write a C program to sort a given list of strings.	66

\*Content beyond the university prescribed syllabi

ANNEXURE		
	Assessment Rubric	67
	Program Outcomes	70
	Knowledge Profile	72
	Complex Engineering Problem	72
	Complex Engineering Activities	73

## General Guideline and Safety Instructions

1. Strictly follow the written and verbal instructions given by the teacher /Lab Instructor. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teacher.
  2. Students are required to attend all labs with official dress code and wearing ID card.
  3. Mobile phones should be switched off in the lab. Keep bags in the bag shelf.
  4. Keep the labs clean at all times, no food and drinks allowed inside the lab.
  5. Students should work individually/team in the hardware and software task.
  6. Students have to bring the lab manual cum lab report file along with them whenever they come for lab work.
  7. Should take only the lab manual, calculator (if needed) and a pen or pencil to the work area.
  8. Should utilize 3 hours' time properly to perform the experiment and to record the readings. Do the calculations, draw the graphs and take signature from the instructor.
  9. If the experiment is not completed in the stipulated time, the pending work has to be carried out in the leisure hours or extended hours.
  10. Intentional misconduct will lead to expulsion from the lab.
  11. Do not handle any equipment without reading the safety instructions. Read the handout and procedures in the Lab Manual before starting the experiments.
  12. Do your wiring, setup, and a careful circuit checkout before applying power. Do not make circuit changes or perform any wiring when power is on.
  13. Avoid contact with energized electrical circuits.
  14. Do not insert connectors forcefully into the sockets.
  15. **NEVER** try to experiment with the power from the wall plug.
  16. Immediately report dangerous or exceptional conditions to the Lab instructor / teacher: Equipment that is not working as expected, wires or connectors are broken, the equipment that smells or "smokes". If you are not sure what the problem is or what's going on, switch off the Emergency shutdown.
  17. Never use damaged instruments, wires or connectors. Hand over these parts to the Lab instructor/Teacher.
  18. After completion of Experiment, return the bread board, trainer kits, wires, oscilloscope probes and other components to lab staff. Do not take any item from the lab without permission.
  19. Handling of Semiconductor Components: Sensitive electronic circuits and electronic components have to be handled with great care. The inappropriate handling of electronic component can damage or destroy the devices. The devices can be destroyed by driving to high currents through the device, by overheating the device, by mixing up the polarity, or by electrostatic discharge (ESD). Therefore, always handle the electronic devices as indicated by the handout, the specifications in the data sheet or other documentation.
  20. Special Precautions during soldering practice
    - a. Hold the soldering iron away from your body. Don't point the iron towards you.
    - b. Don't use a spread solder on the board as it may cause short circuit.
    - c. Do not overheat the components as excess heat may damage the components/board.
    - d. In case of burn or injury seek first aid available in the lab or at the college dispensary.
-

## Course Description

**Introduction:** Data types & data structures, data structure operations, Introduction to algorithms, performance analysis.

**Arrays, Records and Pointer:** Linear arrays, Relationships of arrays, Operation on arrays, Multidimensional arrays, pointer arrays, Record structures, representation of records, Sparse matrices.

**Stacks, Queues and Recursion:** Fundamentals, Different types of stacks and queues: circular, de-queues, etc., Evaluation of expressions, recursion, direct and indirect recursion, depth of recursion, Implementation of recursive procedures by stacks.

**Linked List:** Linked lists, Representation of linked list, Traversing & searching a linked list, Doubly linked list & dynamic storage management, Generalized list, Garbage collection & compaction.

**Trees and Graphs:** Basic terminology, Binary trees, Binary tree representation, Tree traversal, Extended binary tree, Huffman codes/algorithm, Graphs, Graph representation, Shortest path and transitive closure, Traversing a graph.

**Sorting & Searching algorithms:** Basic strategies of Algorithm design, Time and space analysis of algorithms, Average, best and worst case analysis, different notations. Sorting, Insertion sort, Shell sort, Heap sort, Radix sort, the general method of divide & conquer method, Merge sort, Quick sort, Selection sort, binary search, spanning trees, single source shortest paths.

**Symbol Tables:** Static tree tables, Dynamic tree tables, Hash tables overflow handling, Theoretical evaluation of overflow techniques.

**Dynamic programming:** The general method, multistage graphs, all pairs shortest paths, single source shortest paths problems.

## Course Objective:

- i) **Develop knowledge** of basic data structures for storage and retrieval of ordered or unordered data. **Apply** information and concepts in data structure and algorithms fundamentals with the familiarity of issues.
- ii) **Design** algorithms for the creation, insertion, deletion, searching, and sorting of each data structure.
- iii) **Formulate** new solutions for programming problems or improve existing code using learned algorithms and data structures.

## Statement of Course Outcomes (CO):

- 1. Develop knowledge** of basic data structures for storage and retrieval of ordered or unordered data. **Apply** information and concepts in data structure and algorithms fundamentals with the familiarity of issues.
  - 2. Formulate** new solutions for programming problems or improve existing code using learned algorithms and data structures.
  - 3. Design** or develop algorithms for creation, insertion, deletion, searching, and sorting of each data structure in any complex engineering problem.
-

### Statement of Course Outcome (CO) and their mapping

Knowledge Profile(s): WK3, WK4, WK5						
Course Outcomes (CO)	Statements	PO/WA	POI Code	Bloom's Taxonomy Level	Delivery methods and activities	Assessment Tools
CO1	<b>Develop knowledge</b> of basic data structures for storage and retrieval of ordered or unordered data. <b>Apply</b> information and concepts in data structure and algorithms fundamentals with the familiarity of issues.	1	1.3.1	C2,C1,C4	Lectures, Handouts, White Board Writing, Textbook,	Class Observation/ Homework, Assignments, Class Tests, Semester Final Examination (Written)
CO2	<b>Formulate</b> new solutions for programming problems or improve existing code using learned algorithms and data structures.	2	2.2.4	C4, C5		
CO3	<b>Design</b> or develop algorithms for creation, insertion, deletion, searching, and sorting of each data structure in any complex engineering problem.	3	3.2.6	C6		

### Assessment Criteria and Marks Distribution

Si. No.	Particulars	Marks
1	Lab Performance	10
2	Lab Reports	20
3	Lab Test	40
4	Quiz Test	20
5	Lab Viva	10
<b>Total</b>		100

## Summary of Lab Report

Ex. No.	Experiment Title	Date of Exp.	Date of Subm	Marks	Teacher's Signature	Remarks
01	Write a C program to analyze the structure of array. a) Create an array b) Display the array in reverse order c) Find the size of the array using sizeof function d) Display the addition operation in array e) Traverse the array using pointer.					
02	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.					
03	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.					
04	Write a C program that uses stack operations to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.					
05	Write C programs to implement a double ended queue ADT using i) array and ii) doubly linked list respectively.					
06	Write a C program that uses functions to perform the following: a) Create a binary search tree of characters. b) Traverse the above Binary search tree recursively in Postorder.					
07	Write a C program that uses functions to perform the following: a) Create a binary search tree of integers. b) Traverse the above Binary search tree non recursively in inorder.					
08	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order: a) Insertion sort      b) Merge sort					
09	Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order:					

	a) Quick sort                      b) Selection sort					
10	Write a C program to perform the following operation: i) Insertion into a B-tree ii) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.					
11	Write a C program to implement all the functions of a dictionary (ADT) using hashing.					
12	Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.					
13	Write C programs for implementing the following graph traversal algorithms: a) Depth first traversal b) Breadth first traversal					
	Average marks in lab Report (out of 20)					



## **Format of Lab Report**

All lab reports should have to follow the common format as below

- Experiment No
- Experiment Title
- Objectives
- Theory Overview (with formula/equations and/or figure if required)
- Diagram (with adequate labeling and figure caption)
- Results
- Discussion
- Conclusion

## **Instruction for Lab Report Writing**

- Lab report must be hand written without copying from other works.
- Writing should be neat and clean with proper caption and labeling in figure and table.
- The title page of report should contain all the basic information such as experiment no & title, course code & title, student's information, teacher's information, experiment date, submission date.
- Result should include calculated and/or simulated and/or measured data with proper unit.
- Table and/or graph of result should be neat and clear with axis label and units where applicable.
- The discussion should present your findings from the experiment. Evaluate the outcome objectively, taking a candid and unbiased point of view. Suppose that the outcome is not close to what you expected. Even then, after checking your results, give reasons why you believe that outcome is not consistent with the expected.
- In discussion, state the discrepancies between the experimental results and the model (theory), and discuss the sources of the differences in terms of the errors by offering logical inferences and suggest improvements.
- Conclusion should present, a brief summary of what was done, how it was done, show the results and conclusions of the experiment.
- Report should be submitted timely, late submission will cause reduction of marking.

- All lab reports have to be maintained in a single file which has to bring in every laboratory class.

## EXPERIMENT 1

### 1.1 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.

### 1.2 RESOURCE:

Turbo C

### 1.3 PROGRAM LOGIC

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

### 1.4 PROCEDURE

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

### 1.5 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 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 node 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;
}

```

## 1.6 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

## 1.7 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

## 1.8 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?

## 1.9 INPUT AND OUTPUT

```
geetha@iare:~  
[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
```

```
geetha@iare:~  
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

### 2.1 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.

### 2.2 RESOURCE

Turbo C

### 2.3 PROGRAM LOGIC

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

### 2.4 PROCEDURE

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

### 2.5 SOURCE CODE

**Program to create a double linked list to inserting, deleting and displaying the contents**

```
#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;
}

```

## 2.6 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

## 2.7 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

## 2.8 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

## 2.9 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

### 3.1 OBJECTIVE

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

### 3.2 RESOURCE:

Turbo C

### 3.3 PROGRAM LOGIC

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

### 3.4 PROCEDURE:

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

### 3.5 SOURCE CODE:

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

```
#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 for 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;
}

```

### 3.6 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

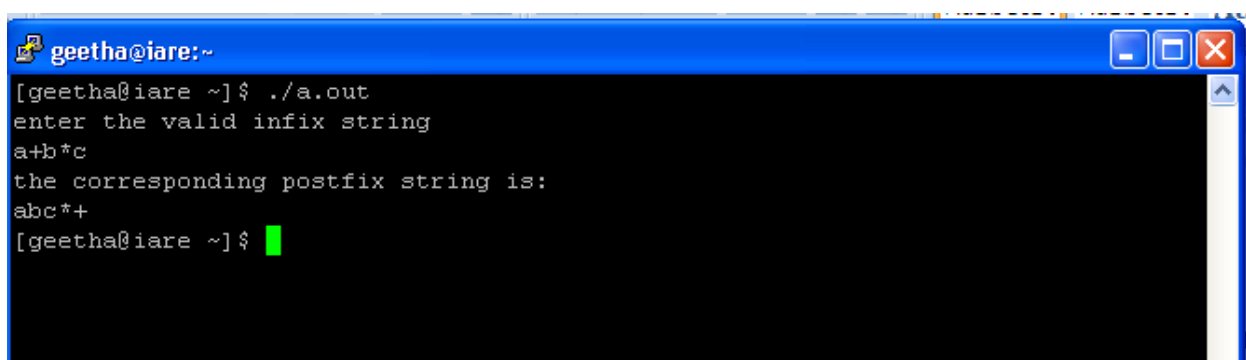
### 3.7 LAB ASSIGNMENT

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.

### 3.8 POST LAB QUESTIONS

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

### 3.9 INPUT AND OUTPUT



```

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

```

## EXPERIMENT 4

### 4.1 OBJECTIVE

1. To implement a double ended queue ADT using arrays.
2. To implement a double ended queue ADT using doubly linked list.

### 4.2 RESOURCE:

Turbo C

### 4.3 PROGRAM LOGIC

#### Double ended queue ADT using arrays

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

#### Double ended queue ADT using doubly linked list

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.

### 4.4 PROCEDURE:

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

### 4.5 SOURCE CODE:

#### Programs to implement a double ended queue ADT using arrays

```
#include<stdio.h>
#define SIZE 30
int dequeue[SIZE];
int front=-1,rear=-1; /* initializing front and rear*/
void insertrear(int);
void deletefront();
void insertfront(int);
void deleterear();
void traverse();
/*main program*/
void main()
{
    int choice,item;
    char ch;
    do
    {
        printf("\n options are");
        printf("\n press 1 to insert at rear");
        printf("\n press 2 to delete at front");
        printf("\n press 3 to insert at front");
        printf("\n press 4 to delete at rear");
        scanf("%d",&choice);
        switch(choice) /*switch case*/
        {
            case 1: printf("\n enter the element:");
                    scanf("%d",&item);
                    insertrear(item); /*function call for inserting element at rear*/
                    break;
            case 2: deletefront(); /*function call for deleting element at front*/
```



```

        break;
    case 3: printf("enter the element:");
            scanf("%d",&item);
            insertfront(item); /*function call for inserting element at front*/
            break;
    case 4: deleterear(); /*function call for deleting element at rear*/
            break;
    case 5: traverse(); /*traversing the list*/
            break;
    default : printf("wrong choice");
} /*end of switch case*/
printf("\n do you want to perform more operations?(Y/N):");
fflush(stdin);
scanf(" %c",&ch);
} while(ch=='Y'||ch=='y');
}

```

**/\*insertion at rear\*/**

void insertrear(int value) **/\*function definition\*/**

```

{
    if(rear==(SIZE-1))
    {
        printf("overflow");
        return;
    }
    else
    {
        if(front== -1)
        {
            printf("underflow so front will be modified");
            front=front+1;
        }
        rear=rear+1;
        dequeue[rear]=value;
    }
}

```

**/\*deletion at front\*/**

void deletefront() **/\*function definition\*/**

```

{
    int value;
    if(front== -1)
    {
        printf("queue is already empty");
        value=-1;
    }
    else
    {
        value=dequeue[front];
        if(front==rear)
        {
            printf("queue contains only one item");
            rear=-1;
            front=-1;
        }
        else
            front=front+1;
    }
}

```

```

    printf("removed element from front is %d",value);
}

/*insertion at front*/
void insertfront(int value) /*function definition*/
{
    if(front==0)
    {
        printf("front is at the beginning");
        printf("here insertion is not possible");
        return;
    }
    else
    {
        if(front==-1)
        {
            printf("queue is empty so both pointers will modified");
            front=front+1;
            rear=rear+1;
        }
        else
        {
            front=front-1;
        }
        dequeue[front]=value;
    }
}

/*deletion at rear*/
void deleterear() /*function definition*/
{
    int value;
    if(front==-1)
    {
        printf("queue is already empty");
        return;
    }
    else
    {
        value=dequeue[rear];
        if(rear==front)
        {
            printf("queue contains only one item");
            printf("rear and front will be modified");
            rear=-1;
            front=-1;
        }
        else
        {
            rear=rear-1;
        }
    }
    printf("\n the removed element from rear is:%d",value);
}

/*traverse operation*/
void traverse() /*function definition*/
{
    int i;

```

```

        if(front==-1)
        {
            printf("queue empty");
            return;
        }
        else
        {
            printf("\n value in the queue are as follow:");
            for(i=front;i<=rear;i++)
                printf("\n%d",dequeue[i]);
        }
    }
}

```

### **program to implement double ended queue adt using doubly linked list**

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

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

```
/*declaring a structure to create a node*/
```

```

struct node
{
    int data;
    struct node *prev, *next;
};

```

```
struct node *head = NULL, *tail = NULL;
```

```

struct node * createNode(int data)
{

```

```
/*allocating implicit memory to the node*/
```

```

    struct node *newnode = (struct node *)malloc(sizeof (struct node));
    newnode->data = data;
    newnode->next = newnode->prev = NULL;
    return (newnode);
}

```

**/\* create sentinel(dummy head & tail) that helps us to do insertion and deletion operation at front and rear so easily. And these dummy head and tail wont get deleted till the end of execution of this program \*/**

```
void createSentinels() /*creating a head and tail*/
```

```

{
    head = createNode(0);
    tail = createNode(0);
    head->next = tail;
    tail->prev = head;
}

```

**/\* insertion at the front of the queue \*/**

```
void enqueueAtFront(int data)
```

```

{
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = head->next;
    head->next = newnode;
    newnode->prev = head;
    newnode->next = temp;
    temp->prev = newnode;
}

```

```

}

/*insertion at the rear of the queue */
void enqueueAtRear(int data)
{
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = tail->prev;
    tail->prev = newnode;
    newnode->next = tail;
    newnode->prev = temp;
    temp->next = newnode;
}

/* deletion at the front of the queue */
void dequeueAtFront()
{
    struct node *temp;
    if (head->next == tail)
    {
        printf("Queue is empty\n");
    }
    Else
    {
        temp = head->next;
        head->next = temp->next;
        temp->next->prev = head;
        free(temp);
    }
    return;
}

/* deletion at the rear of the queue */

void dequeueAtRear()
{
    struct node *temp;
    if (tail->prev == head)
    {
        printf("Queue is empty\n");
    }
    Else
    {
        temp = tail->prev;
        tail->prev = temp->prev;
        temp->prev->next = tail;
        free(temp);
    }
    return;
}

/* display elements present in the queue */
void display()
{
    struct node *temp;
    if (head->next == tail)
    {

```

```

        printf("Queue is empty\n");
        return;
    }

    temp = head->next;
    while (temp != tail)
    {
        printf("%-3d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

/*main program*/
int main()
{
    int data, ch;
    createSentinels();
    while (1)
    {
        printf("1. Enqueue at front\n2. Enqueue at rear\n");
        printf("3. Dequeue at front\n4. Dequeue at rear\n");
        printf("5. Display\n6. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch (ch) /*switch case*/
        {
            case 1:
                printf("Enter the data to insert:");
                scanf("%d", &data);
                enqueueAtFront(data);
                break;

            case 2:
                printf("Enter ur data to insert:");
                scanf("%d", &data);
                enqueueAtRear(data);
                break;

            case 3:
                dequeueAtFront();
                break;

            case 4:
                dequeueAtRear();
                break;

            case 5:
                display();
                break;

            case 6:
                exit(0);

            default:
                printf("Pls. enter correct option\n");
                break;
        } /*end of switch case*/
    }
}

```

```

    }
    return 0;
}

```

#### 4.6 PRE LAB QUESTIONS

1. what is queue and its operations
2. what is double ended queue
3. differentiate queue and double ended queue

#### 4.7 LAB ASSIGNMENT

1. Write a program to insert an element when rear is at last position
2. Write a program to delete an element when front is at last position

#### 4.8 POST LAB QUESTIONS

1. Write the condition for queue full
2. Write the condition for queue empty
3. List the advantages of double ended queue over queue

#### 4.9 INPUT AND OUTPUT

##### A double ended queue ADT using arrays

```

geetha@iare:~
options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear1

enter the element:23
underflow so front will be modified
do you want to perform more operations?(Y/N):y

options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear3
enter the element:24
front is at the beginning here insertion is not possible
do you want to perform more operations?(Y/N):y

options are
press 1 to insert at rear
press 2 to delete at front
press 3 to insert at front
press 4 to delete at rear1

```

```

press 3 to insert at front
press 4 to delete at rear4
queue contains only one item rear and front will be modified
the removed element from rear is:50
do you want to perform more operations?(Y/N):n
[geetha@iare ~]$

```

```
geetha@iare:~  
enter the element:50  
  
do you want to perform more operations?(Y/N):y  
  
options are  
press 1 to insert at rear  
press 2 to delete at front  
press 3 to insert at front  
press 4 to delete at rear3  
enter the element:12  
front is at the beginninghere insertion is not possible  
do you want to perform more operations?(Y/N):y  
  
options are  
press 1 to insert at rear  
press 2 to delete at front  
press 3 to insert at front  
press 4 to delete at rear2  
removed element from front is 23  
do you want to perform more operations?(Y/N):y  
  
options are  
press 1 to insert at rear  
press 2 to delete at front
```

### double ended queue adt using doubly linked list

```
geetha@iare:~  
[geetha@iare ~]$ ./a.out  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:1  
enter data to be inserted:12  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:2  
enter data to be inserted:25  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:█
```

```
geetha@iare:~  
enter choice:1  
enter data to be inserted:26  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:2  
enter data to be inserted:65  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:3  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:█
```

```
enter choice:4  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:5  
12 25  
1.enqueueatfront  
2.enqueueatrear  
3.dequeueatfront  
4.dequeueatrear  
5.display  
6.exit  
enter choice:6  
[geetha@iare ~]$ █
```



## EXPERIMENT 5

### 5.1 OBJECTIVE

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

### 5.2 RESOURCE:

Turbo C

### 5.3 PROGRAM LOGIC

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

### 5.4 PROCEDURE:

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

### 5.5 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.resursive 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));
                    printf("\n enter the elements");
                    nn->lc=NULL;
```

```

        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);
    }
}
}

```

## 5.6 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.

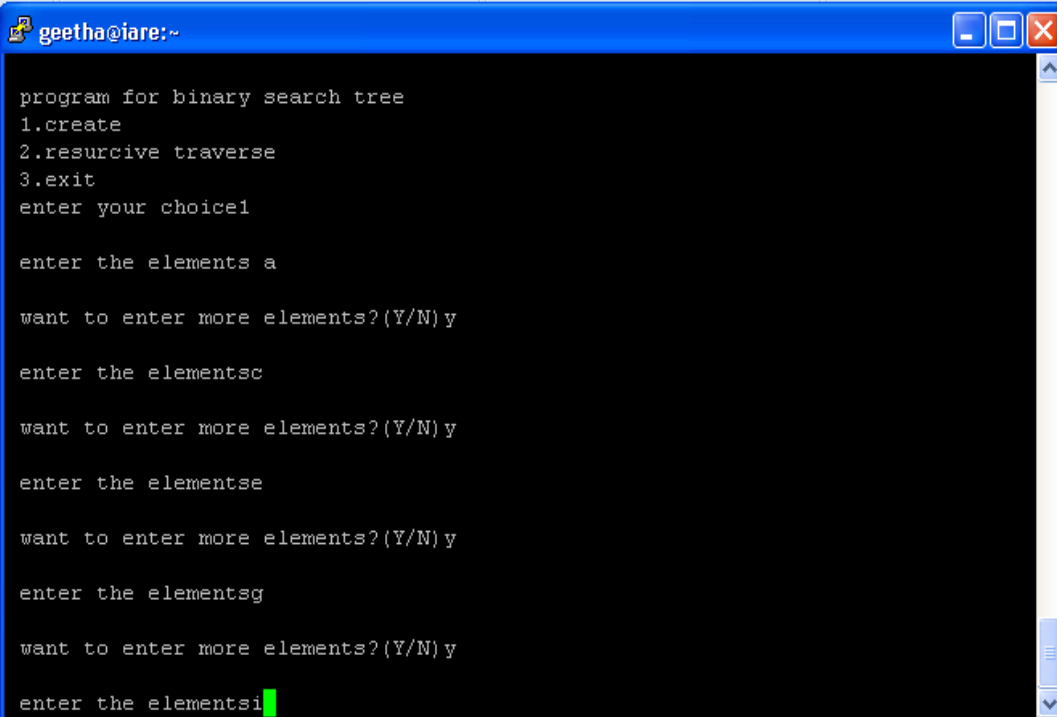
## 5.7 LAB ASSIGNMENT

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

## 5.8 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

## 5.9 INPUT AND OUTPUT



```

geetha@iare:~
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 elementshi

```

## EXPERIMENT 6

### 6.1 OBJECTIVE

1. Create a binary search tree of integers.
2. Traverse the above Binary search tree non recursively in inorder.

### 6.2 RESOURCE:

Turbo C

### 6.3 PROGRAM LOGIC

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

### 6.4 PROCEDURE

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

### 6.5 SOURCE CODE:

```
/*creating binary search tree of integer*/

#include<stdio.h>
#include<conio.h>
typedef struct bint
{
/*declaring a structure to create a node*/

    int data,flag;
    struct bint *left,*right;
}node;
node * create(node *r,int d)
{
    if(r == NULL)
    {
/*allocating implicit memory to the node*/

        r = (node *)malloc(sizeof(node));
        r->data = d;
        r->left = r->right = NULL;
    }
    else
    {
        if(r->data <= d)
            r->right = create(r->right,d);
        else
            r->left = create(r->left,d);
    }
    return r;
}
void non_in(node *r)
{
    int top=0;
    node *s[20],*pt=r;
    s[0]=NULL;
```

```

while(pt != NULL)
{
    s[++top] = pt;
    pt = pt->left;
}
pt = s[top--];
while(pt != NULL)
{
    printf("%d\t",pt->data);
    if(pt->right != NULL)
    {
        pt = pt->right;
        while(pt != NULL)
        {
            s[++top] = pt;
            pt = pt->left;
        }
    }
    pt = s[top--];
}
}
/*main program*/
void main()
{
    int d;
    char ch = 'Y';
    node *root = NULL;
    clrscr();
    while(toupper(ch) == 'Y')
    {
        printf("\n Enter the item to insert");
        scanf("%d",&d);
        head = create(root,d);
        printf("\n Do you want to continue(y/n)");
        fflush(stdin);
        ch = getchar();
    }

    printf("\n inorder non recursive\n");
    non_in(head);
}

```

## 6.6 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.

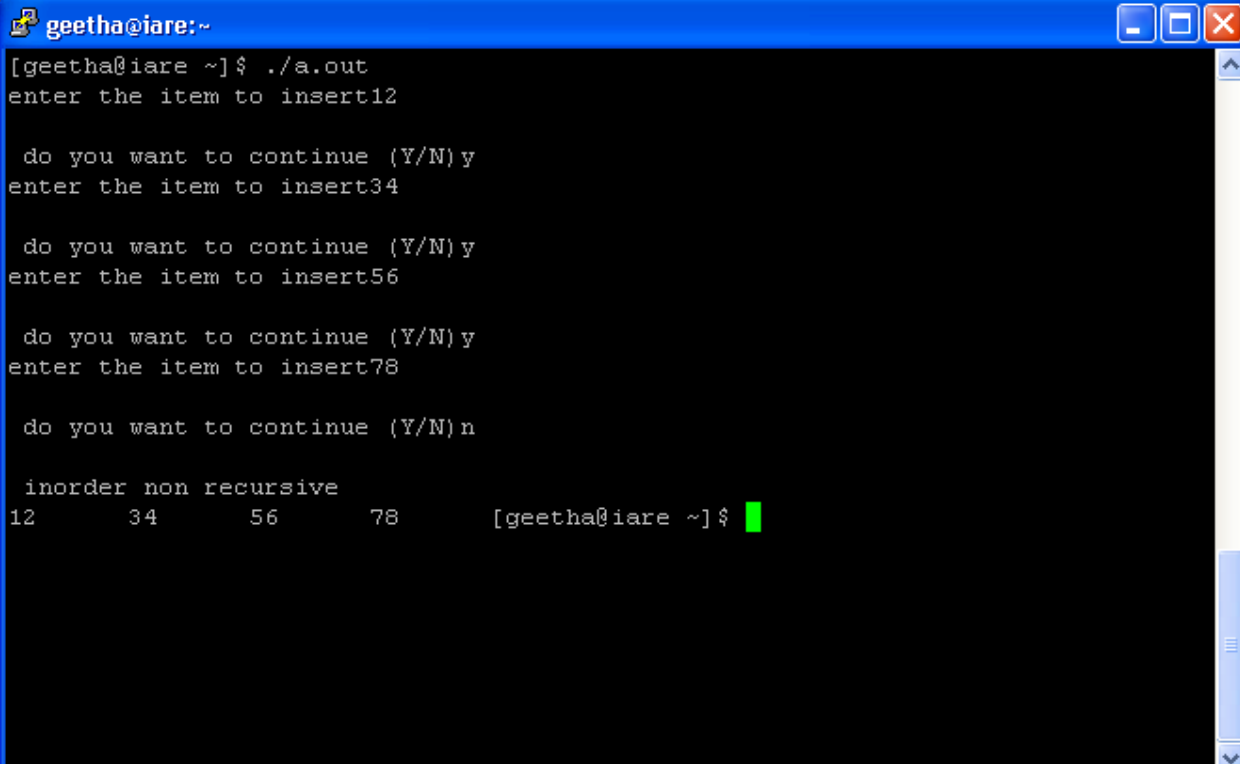
## 6.7 LAB ASSIGNMENT

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

## 6.8 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

## 6.9 INPUT AND OUTPUT

A terminal window with a blue title bar and standard window controls. The text inside shows a program being executed with several inputs and a final output.

```
geetha@iare:~  
[geetha@iare ~]$ ./a.out  
enter the item to insert12  
  
do you want to continue (Y/N)y  
enter the item to insert34  
  
do you want to continue (Y/N)y  
enter the item to insert56  
  
do you want to continue (Y/N)y  
enter the item to insert78  
  
do you want to continue (Y/N)n  
  
inorder non recursive  
12      34      56      78      [geetha@iare ~]$
```

## EXPERIMENT 7

### 7.1 OBJECTIVE

1. Arrange a list of integers in ascending order using Insertion sort
2. Arrange a list of integers in ascending order using Merge sort

### 7.2 RESOURCE

Turbo C

### 7.3 PROGRAM LOGIC

#### INSERTION SORT

1. Start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].
2. Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].
3. To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.

#### MERGE SORT

##### 1. Divide Step

If a given array  $A$  has zero or one element, simply return; it is already sorted. Otherwise, split  $A[p .. r]$  into two subarrays  $A[p .. q]$  and  $A[q + 1 .. r]$ , each containing about half of the elements of  $A[p .. r]$ . That is,  $q$  is the halfway point of  $A[p .. r]$ .

##### 2. Conquer Step

Conquer by recursively sorting the two subarrays  $A[p .. q]$  and  $A[q + 1 .. r]$ .

##### 3. Combine Step

Combine the elements back in  $A[p .. r]$  by merging the two sorted subarrays  $A[p .. q]$  and  $A[q + 1 .. r]$  into a sorted sequence. To accomplish this step, we will define a procedure MERGE ( $A, p, q, r$ ).

### 7.4 PROCEDURE:

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

### 7.5 SOURCE CODE:

#### Insertion sort

```
#include<stdio.h>
void inst_sort(int[]);
void main() {
    int num[5],count;
    printf("\n enter the five elements to sort:\n");
    for(count=0;count<5;count++)
        scanf("%d",&num[count]);
    inst_sort(num); /*function call for insertion sort*/
    printf("\n\n elements after sorting:\n");
    for(count=0;count<5;count++)
        printf("%d\n",num[count]);
}
void inst_sort(int num[]) { /* function definition for insertion sort*/
    int i,j,k;
    for(j=1;j<5;j++) {
        k=num[j];
        for(i=j-1;i>=0&& k<num[i];i--)
```

```

        num[i+1]=num[i];
        num[i+1]=k;
    } }

```

### **Merge sort**

```

#include<stdio.h>
void mergesort(int[],int,int);
void mergearray(int[],int,int,int);
main() {
    int a[50],n,i;
    printf("\n enter size of an array:");
    scanf("%d",&n);
    printf("\n enter elements of an array:\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    mergesort(a,0,n-1);
    printf("\n\nafter sorting:\n");
    for(i=0;i<n;i++)
        printf("\n%d",a[i]);
}
/*merge operation*/
void mergesort(int a[],int beg,int end) {
    int mid;
    if(beg<end) {
        mid=(beg+end)/2;
        mergesort(a,beg,mid);
        mergesort(a,mid+1,end);
        mergearray(a,beg,mid,end);
    } }
void mergearray(int a[],int beg,int mid,int end) {
    int i,leftend,num,temp,j,k,b[50];
    for(i=beg;i<=end;i++)
        b[i]=a[i];
    i=beg;
    j=mid+1;
    k=beg;
    while((i<=mid)&&(j<=end)) {
        if(b[i]<=b[j])
        {
            a[k]=b[i];
            i++;
            k++;
        }
        else {
            a[k]=b[j];
            j++;
            k++;
        } }
    if(i<=mid) {
        while(i<=mid) {
            a[k]=b[i];
            i++;
            k++;
        } }
    else {
        while(j<=end) {

```



```

        a[k]=b[j];
        j++;
        k++;
    } } }

```

## 7.6 PRE LAB QUESTIONS

1. what is sorting
2. List the various sorting algorithms
3. What is the advantage of insertion sort
4. How merge sort works
5. Is insertion sort a stable algorithm

## 7.7 LAB ASSIGNMENT

1. Apply the selection sort on the following elements 21,11,5,78,49, 54,72,88
2. Apply the merge sort on the following elements 21,11,5,78,49, 54,72,88 and 56,28,10

## 7.8 POST LAB QUESTIONS

1. What is the time complexity of insertion sort
2. What is the time complexity of merge sort
3. Why sorting is required
4. Is merge sort in place
5. List the application of merge sort

## 7.9 INPUT AND OUTPUT

### Insertion sort

```

geetha@iare:~
[geetha@iare ~]$ gcc insert.c
[geetha@iare ~]$ ./a.out

enter the five elements to sort:
5 4 3 2 1

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

```

### Merge sort

```

geetha@iare:~
[geetha@iare ~]$ gcc week7b.c
[geetha@iare ~]$ ./a.out

enter size of an array:5

enter elements of an array:
5 4 3 2 1

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

```

## EXPERIMENT 8

### 8.1. OBJECTIVE

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

### 8.2 RESOURCE:

Turbo C

### 8.3 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

### 8.4 PROCEDURE:

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

### 8.5 SOURCE CODE:

#### QUICK SORT

```
#include<stdio.h>
main()
{
    int x[10],i,n;
    printf("enter number of elements:");
    scanf("%d",&n);
    printf("enter %d elements:\n");
    for(i=0;i<n;i++)
        scanf("%d",&x[i]);
    quicksort(x,0,n-1);/*function call*/
    printf("sorted elements are:");
    for(i=0;i<n;i++)
        printf("%3d",x[i]);
}
/*called function*/
quicksort(int x[10],int first,int last)
{
    int pivot,i,j,t;
    if(first<last)
    {
        pivot=first;
        i=first;
        j=last;
        while(i<j)
        {
            while(x[i]<=x[pivot]&& i<last)
                i++;
            while(x[j]>x[pivot])
                j--;
            if(i<j)
```

```

        {
            t=x[i];
            x[i]=x[j];
            x[j]=t;
        }
    }
    t=x[pivot];
    x[pivot]=x[j];
    x[j]=t;
    quicksort(x,first,j-1);
    quicksort(x,j+1,last);
}
}

```

## **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]);
    }
}

```

## **8.6 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

## **8.7 LAB ASSIGNMENT**

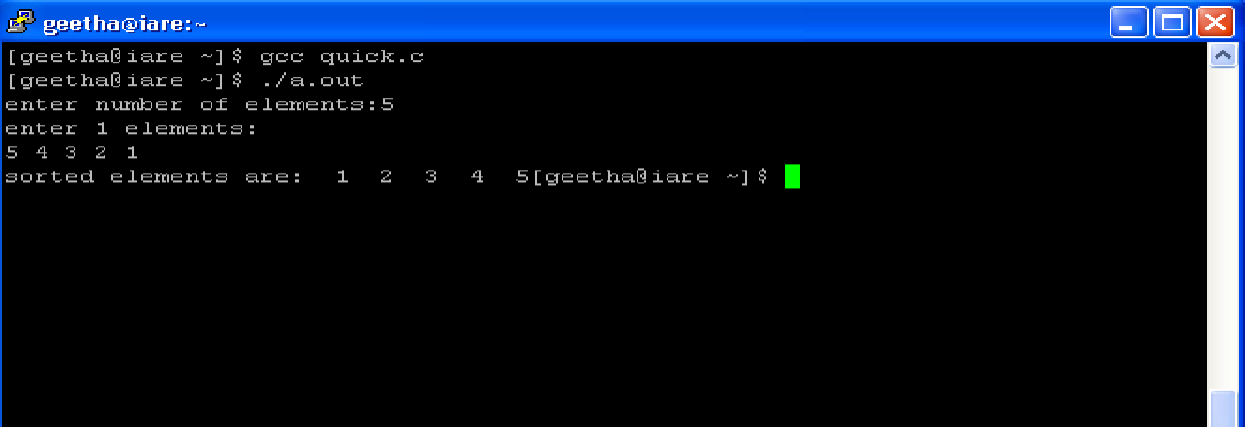
1. Rearrange the following numbers using Quick sort procedure. 42, 12, 18, 98, 67, 83, 8, 10, 71
2. Apply the selection sort on the following elements 21, 11, 5, 78, 49, 54, 72, 88

## **8.8 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

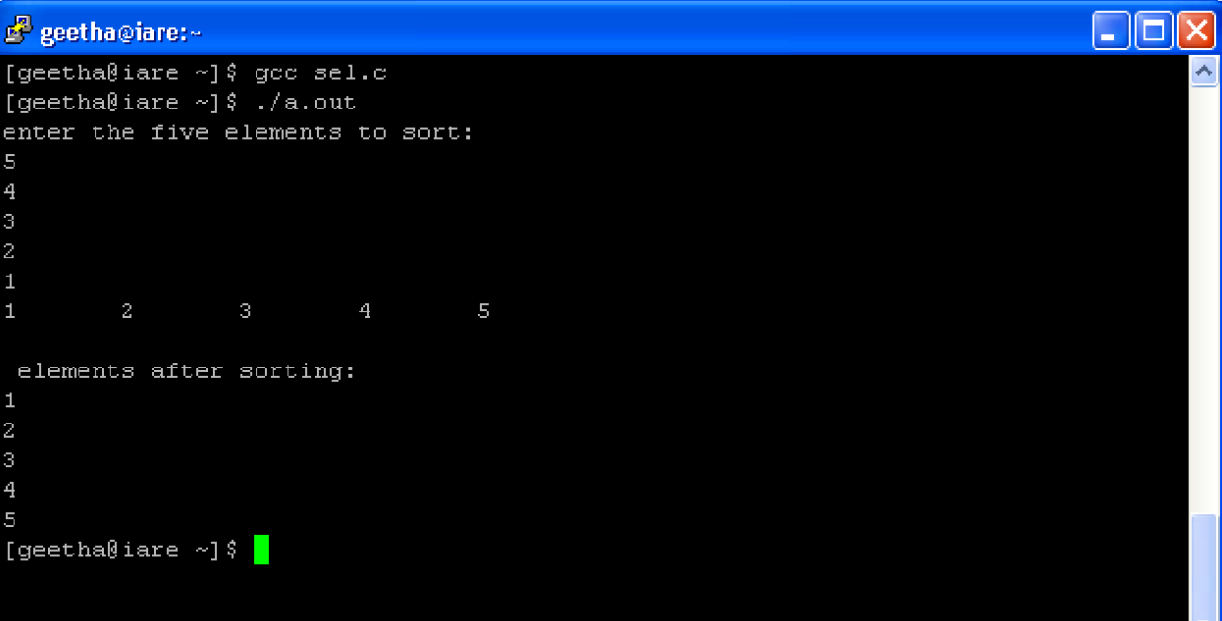
## **8.9 INPUT AND OUTPUT,**

### **QUICK SORT**



```
geetha@iare:~  
[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:~  
[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

### 9.1 OBJECTIVE

1. To perform the Insertion into a B-tree
2. Implement Heap sort algorithm

### 9.2 RESOURCE:

Turbo C

### 9.3 PROGRAM LOGIC

#### B-tree

1. Read the elements
2. Use the properties B-Tree and construct B-Tree
3. Select the required operation

#### Heap sort

4. Read the elements to be sort
5. Construct the heap
6. Sort the above heap using deletion

### 9.4 PROCEDURE:

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

### 9.5 SOURCE CODE:

#### **B-TREE**

```
#include<stdio.h>
#include <stdlib.h>
#define M 5
struct node
{
    int n; /* n < M No. of keys in node will always less than order of B
           tree */
    int keys[M-1]; /*array of keys*/
    struct node *p[M]; /* (n+1 pointers will be in use) */
} *root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };
void insert(int key);
void display(struct node *root,int);
void search(int x);
enum KeyStatus ins(struct node *r, int x, int* y, struct node** u);
int searchPos(int x,int *key_arr, int n);
int main()
{
    int key;
    int choice;
    printf("Creation of B tree for node %d\n",M);
    while(1)
    {
        printf("1.Insert\n");
        printf("3.Search\n");
        printf("4.Display\n");
        printf("5.Quit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
```

```

switch(choice)
{
    case 1:
        printf("Enter the key : ");
        scanf("%d",&key);
        insert(key);
        break;

    case 3:
        printf("Enter the key : ");
        scanf("%d",&key);
        search(key);
        break;

    case 4:
        printf("Btree is :\n");
        display(root,0);
        break;

    case 5:
        exit(1);
    default:
        printf("Wrong choice\n");
        break;
}/*End of switch*/
}/*End of while*/
return 0;
}/*End of main()*/
void insert(int key)
{
    struct node *newnode;
    int upKey;
    enum KeyStatus value;
    value = ins(root, key, &upKey, &newnode);
    if (value == Duplicate)
        printf("Key already available\n");
    if (value == InsertIt)
    {
        struct node *uproot = root;
        root=malloc(sizeof(struct node));
        root->n = 1;
        root->keys[0] = upKey;
        root->p[0] = uproot;
        root->p[1] = newnode;
    }/*End of if */
}/*End of insert()*/

enum KeyStatus ins(struct node *ptr, int key, int *upKey,struct node**newnode)
{
    struct node *newPtr, *lastPtr;
    int pos, i, n,splitPos;
    int newKey, lastKey;
    enum KeyStatus value;
    if (ptr == NULL)
    {
        *newnode = NULL;
        *upKey = key;
        return InsertIt;
    }

```

```

    }
    n = ptr->n;
    pos = searchPos(key, ptr->keys, n);
    if (pos < n && key == ptr->keys[pos])
        return Duplicate;
    value = ins(ptr->p[pos], key, &newKey, &newPtr);
    if (value != InsertIt)
        return value;
    /*If keys in node is less than M-1 where M is order of B tree*/
    if (n < M - 1)
    {
        pos = searchPos(newKey, ptr->keys, n);
        /*Shifting the key and pointer right for inserting the new key*/
        for (i=n; i>pos; i--)
        {
            ptr->keys[i] = ptr->keys[i-1];
            ptr->p[i+1] = ptr->p[i];
        }
        /*Key is inserted at exact location*/
        ptr->keys[pos] = newKey;
        ptr->p[pos+1] = newPtr;
        ++ptr->n; /*incrementing the number of keys in node*/
        return Success;
    } /*End of if */
    /*If keys in nodes are maximum and position of node to be inserted is last*/
    if (pos == M - 1)
    {
        lastKey = newKey;
        lastPtr = newPtr;
    }
    else /*If keys in node are maximum and position of node to be inserted is not last*/
    {
        lastKey = ptr->keys[M-2];
        lastPtr = ptr->p[M-1];
        for (i=M-2; i>pos; i--)
        {
            ptr->keys[i] = ptr->keys[i-1];
            ptr->p[i+1] = ptr->p[i];
        }
        ptr->keys[pos] = newKey;
        ptr->p[pos+1] = newPtr;
    }
    splitPos = (M - 1)/2;
    (*upKey) = ptr->keys[splitPos];

    (*newnode)=malloc(sizeof(struct node)); /*Right node after split*/
    ptr->n = splitPos; /*No. of keys for left splitted node*/
    (*newnode)->n = M-1-splitPos; /*No. of keys for right splitted node*/
    for (i=0; i < (*newnode)->n; i++)
    {
        (*newnode)->p[i] = ptr->p[i + splitPos + 1];
        if(i < (*newnode)->n - 1)
            (*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
        else
            (*newnode)->keys[i] = lastKey;
    }

```

```

(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}/*End of ins()*/

void display(struct node *ptr, int blanks)
{
if (ptr)
{
int i;
for(i=1;i<=blanks;i++)
printf(" ");
for (i=0; i < ptr->n; i++)
printf("%d ",ptr->keys[i]);
printf("\n");
for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);
}/*End of if*/
}/*End of display()*/

void search(int key)
{
int pos, i, n;
struct node *ptr = root;
printf("Search path:\n");
while (ptr)
{
n = ptr->n;
for (i=0; i < ptr->n; i++)
printf(" %d",ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
{
printf("Key %d found in position %d of last dispalyed node\n",key,i);
return;
}
ptr = ptr->p[pos];
}
printf("Key %d is not available\n",key);
}/*End of search()*/

int searchPos(int key, int *key_arr, int n)
{
int pos=0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}/*End of searchPos()*/

```

## **HEAP SORT**

```

#include<stdio.h>
int p(int);
int left(int);
int right(int);
void heapify(int[],int,int);
void buildheap(int[],int);

```



```

void heapsort(int[],int);
void main()
{
    int x[20],n,i;
    printf("enter the no. of elements to b sorted");
    scanf("%d",&n);
    printf("enter the elements ");
    for(i=0;i<n;i++)
        scanf("%d",&x[i]);
    heapsort(x,n);
    printf("sorted array is");
    for(i=0;i<n;i++)
        printf("%d",x[i]);
}
int p(int i)
{
    return i/2;
}
int left(int i)
{
    return 2*i+1;
}
int right(int i)
{
    return 2*i+2;
}
void heapify(int a[],int i,int n)
{
    int l,r,large,t;
    l=left(i);
    r=right(i);
    if((l<=n-1)&&(a[l]>a[i]))
        large=l;
    else
        large=i;
    if((r<=n-1)&&(a[r]>a[large]))
        large=r;
    if(large!=i)
    {
        t=a[i];
        a[i]=a[large];
        a[large]=t;
        heapify(a,large,n);
    }
}
void buildheap(int a[],int n)
{
    int i;
    for(i=(n-1)/2;i>=0;i--)
        heapify(a,i,n);
}
void heapsort(int a[],int n)
{
    int i,m,t;
    buildheap(a,n);
    m=n ;
    for(i=n-1;i>=1;i--)
    {

```

```

        t=a[0];
        a[0]=a[i];
        a[i]=t;
        m=m-1;
        heapify(a,0,m);
    }
}

```

## 9.6 PRE LAB QUESTIONS

1. what is B –tree
2. What is heap
3. What are the properties of B-tree
4. What is sorting

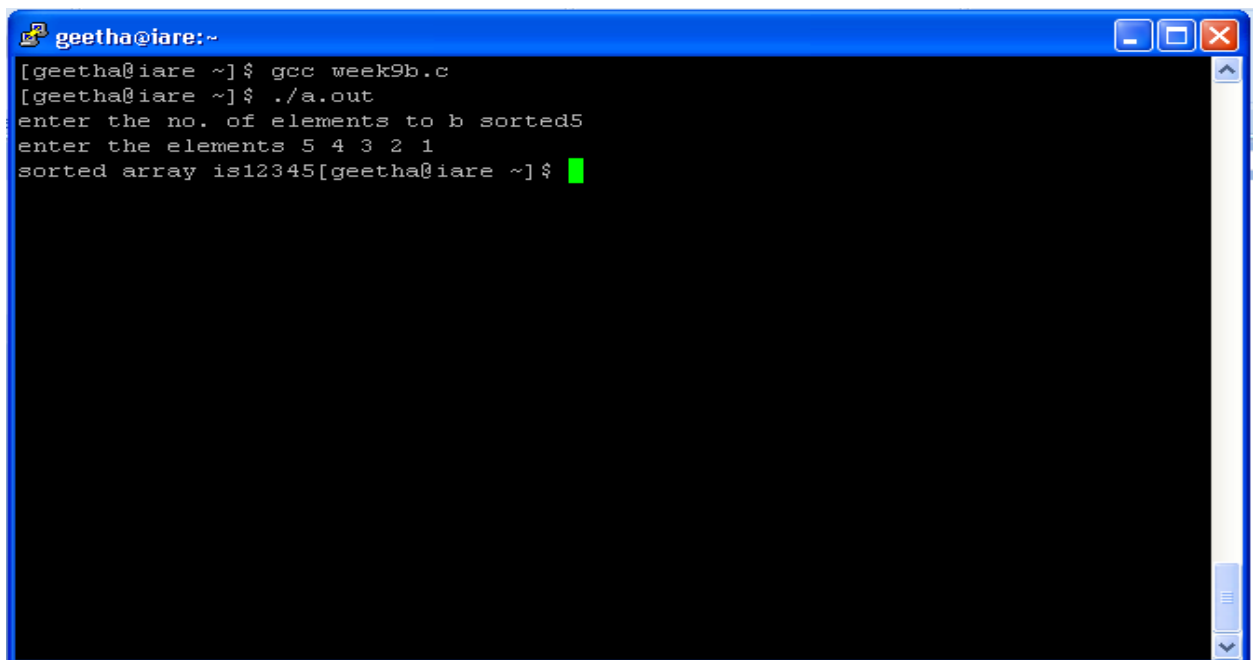
## 9.7 LAB ASSIGNMENT

1. Apply heap sort on list of elements 14,12,9,8,7,10,18,20,30
2. Construct a B-tree of order 3 with the following elements 10,20,15,3,2,16,21,25,30,40

## 9.8 POST LAB QUESTIONS

1. What is time complexity of heap sort
2. Write the condition to insert and delete an element into B-tree
3. Is heap sort is stable or not

## 9.9 INPUT AND OUTPUT



```

geetha@iare:~
[geetha@iare ~]$ gcc week9b.c
[geetha@iare ~]$ ./a.out
enter the no. of elements to b sorted5
enter the elements 5 4 3 2 1
sorted array is12345[geetha@iare ~]$

```

## EXPERIMENT 10

### 10.1 OBJECTIVE

To implement all the functions of a dictionary (ADT) using hashing

### 10.2 RESOURCE:

Turbo C

### 10.3 PROGRAM LOGIC

1. Read the key elements from the dictionary
2. Use the hash function to implement dictionary
3. Apply required operation on the dictionary

### 10.4 PROCEDURE:

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

### 10.5 SOURCE CODE:.

#### DICTIONARY USING HASHING

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int b;

int hsearch(int key,int d,int *ht,int *empty)
{
    int i=key%(d);
    int j=i;
    int c=0;
    do
    {
        if(empty[j]||(*ht+j)==key))
            return j;
        c++;
        j=(i+c)%(d);
    }while(j!=i);
    return 0;
}

int search(int key,int d,int *ht,int *empty)
{
    b=hsearch(key,d,ht,empty);
    printf("%d",b);
    if(empty[b]==1)
        return -1;
    else if(b==0)
        return 1;
    else
        return b;
}

/*insertion operation*/
void insert(int key,int d,int *ht,int *empty)
{
    b=hsearch(key,d,ht,empty);
    if(empty[b])
    {
```

```

        empty[b]=0;
        *(ht+b)=key;
        printf("elements is inserted\n");
    }
}
/*deletion operation*/
void delete(int key,int d,int *ht,int *empty)
{
    int b=hsearch(key,d,ht,empty);
    *(ht+b)=0;
    empty[b]=1;
    printf("element is deleted\n");
}
void display(int d,int *ht,int *empty)
{
    int i;
    printf("hash table elements are\n");
    for(i=0;i<d;i++)
    {
        if(empty[i])
            printf(" 0");
        else
            printf("%5d",*(ht+i));
    }
    printf("\n");
}
/*main program*/
void main()
{
    int choice=1;
    int key;
    int d,i,s;
    int *empty,*ht;
    printf("enter the hash table size:");
    scanf("%d",&d);
    ht=(int *)malloc(d *sizeof(int));
    empty=(int *)malloc(d *sizeof(int));
    for(i=0;i<d;i++)
        empty[i]=1;
    while(1)
    {
        printf("\n");
        printf("\n LINEAR PROBING");
        printf("\n 1:insert hash table");
        printf("\n 2:delete hash table");
        printf("\n 3:search hash table");
        printf("\n 4:display hash table");
        printf("\n 5:exit");
        printf("enter your choice");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("enter the elemants:");
                    scanf("%d",&key);
                    insert(key,d,ht,empty);
                    break;
            case 2:printf("enter to remove from hash table:");
                    scanf("%d",&key);

```

```

        delete(key,d,ht,empty);
        break;
    case 3:printf("enter the search elements:");
        scanf("%d",&key);
        s=search(key,d,ht,empty);
        if(s==-1||s==0)
            printf("not found\n");
        else
            printf("element found at index %d",hsearch(key,d,ht,empty));
        break;
    case 4:display(d,ht,empty);
        break;
    case 5:exit(0);
    }
}return;
}

```

### 10.6 PRE LAB QUESTIONS

1. what is dictionary
2. What is hashing
3. List types of hash functions
4. Define linear probing
5. What is quadratic probing

### 10.7 LAB ASSIGNMENT

1. Use quadratic probing to fill the Hash table of size 11. Data elements are 23,0,52,61,78,33,100,8,90,10,14.
2. Analyze input (371, 323, 173, 199, 344, 679, 989) and hash function  $h(x)=x \bmod 10$ , Show the result Separate Chaining, linear probing

### 10.8 POST LAB QUESTIONS

1. List the application of hashing
2. Explain the condition to insert element into hash table
3. What is double hashing

## 10.9 INPUT AND OUTPUT

```
geetha@iare:~  
[geetha@iare ~]$ gcc week9.c  
[geetha@iare ~]$ ./a.out  
enter the hash table size:3  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice1  
enter the elemants:12  
elements is inserted  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice1  
enter the elemants:22  
elements is inserted
```

```
geetha@iare:~  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice1  
enter the elemants:22  
elements is inserted  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice1  
enter the elemants:31  
elements is inserted  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice
```

```
geetha@iare:~  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice4  
hash table elements are  
12 22 31  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice2  
enter to remove from hash table:22  
element is deleted  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table
```

```
geetha@iare:~  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice3  
enter the search elements:22  
1not found  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice4  
hash table elements are  
12 0 31  
  
LINEAR PROBING  
1:insert hash table:  
2:delete hash table  
3:search hash table  
4:display hash table  
5:exitenter your choice
```

## EXPERIMENT 11

### 11.1 OBJECTIVE

To implementing Knuth-Morris- Pratt pattern matching algorithm

### 11.2 RESOURCE:

Turbo C

### 11.3 PROGRAM LOGIC

1. Read the pattern and text
2. Compute failure function
3. Apply the KMP algorithm

### 11.4 PROCEDURE

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

### 11.5 SOURCE CODE

#### Knuth-Morris- Pratt pattern matching algorithm

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

void computefailure(char *pat, int M, int *f);

void KMPSearch(char *pat, char *txt) //to implement kmp search
{
    int M = strlen(pat);//length of pattern
    int N = strlen(txt);//length of text

    int *f = (int *)malloc(sizeof(int)*M); //dynamic allocation
    int j = 0; // index for pat[]

    computefailure(pat, M, f);

    int i = 0; // index for txt[]
    while(i < N)
    {
        if(pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)
        {
            printf("Found pattern at index %d \n", i-j);
            j = f[j-1];
        }
        else if(pat[j] != txt[i])
        {
            if(j != 0)
            {
                j = f[j-1];
            }
            else
            {
                i = i+1;
            }
        }
    }
}
```



```

    }
    free(f); }

void computefailure (char *pat, int M, int *f) //compute the failure function
{
    int len = 0;
    int i;

    f[0] = 0; // f[0] is always 0
    i = 1;

    while(i < M)
    {
        if(pat[i] == pat[len])
        {
            len++;
            f[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if( len != 0 )
            {
                len = f[len-1];
            }
            else // if (len == 0)
            {
                f[i] = 0;
                i++;
            }
        }
    }
}

int main()
{
    int match;
    printf("\nEnter the Text: ");
    gets(text); //reading the text
    printf("\nEnter the Pattern: ");
    gets(pattern); //reading the pattern
    m=strlen(text);
    n=strlen(pattern);
    match=KMPSearch(pat, txt);

    if(match>=0)
    {
        printf("\nMatch found at position %d\n",match);
    }
    else
    {
        printf("\nNo Match found!!!\n");
    }
    return 0;
}

```

### 11.6 PRE LAB QUESTIONS

1. what is pattern matching
2. List the various pattern matching algorithms
3. What is failure function

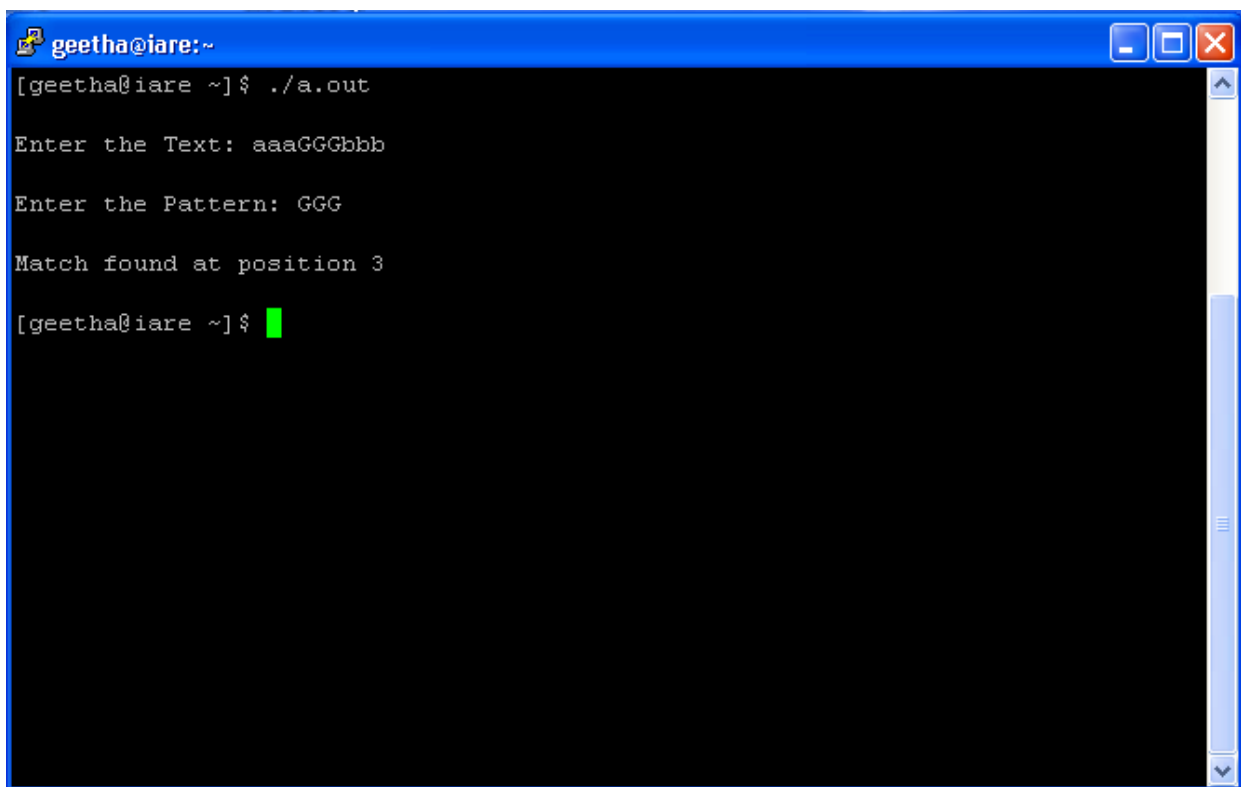
### 11.7 LAB ASSIGNMENT

1. Apply KMP algorithm on pattern “abacab” and text “abacaabaccabacabaabb”
2. Apply KMP algorithm on pattern “abaa” and text “abbbaababaab”

### 11.8 POST LAB QUESTIONS

1. Which pattern matching algorithm is better
2. Write the significance of Knuth-Morris- Pratt pattern matching algorithm
3. List the applications of pattern matching algorithms

### 11. 9 INPUT AND OUTPUT



```
geetha@iare:~  
[geetha@iare ~]$ ./a.out  
Enter the Text: aaaGGGbbb  
Enter the Pattern: GGG  
Match found at position 3  
[geetha@iare ~]$
```

The image shows a terminal window with a blue title bar. The window title is "geetha@iare:~". The terminal content shows the execution of a program. The user enters the text "aaaGGGbbb" and the pattern "GGG". The program outputs "Match found at position 3". The prompt "[geetha@iare ~]" is followed by a green cursor.

## EXPERIMENT 12

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

### 12.1 OBJECTIVE

To traverse graph

### 12.2 RESOURCE:

Turbo C

### 12.3 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

### 12.4 PROCEDURE:

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

### 12.5 SOURCE CODE;

#### DFS

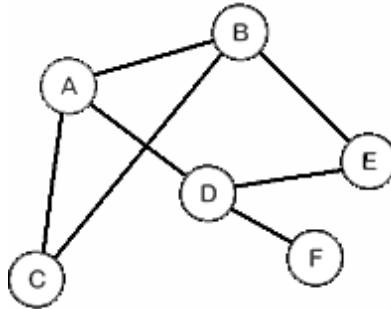
```
#include <stdio.h>
void dfs(int);
int g[10][10],visited[10],n,vertex[10];
void main()
{
    int i,j;
    printf("enter number of vertices:");
    scanf("%d",&n);
    printf("enter the val of vertex:");
    for(i=0;i<n;i++)
        scanf("%d",&vertex[i]);
    printf("\n enter adjecency matrix of the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

## 12.6 PRE LAB QUESTIONS

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

## 12.7 LAB ASSIGNMENT

1. Find DFS traversal of the following graph



2. Deduce the time complexity of DFS algorithm

## 12.8 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:~  
[geetha@iare ~]$ gcc dfs.c  
[geetha@iare ~]$ ./a.out  
enter number of vertices:4  
enter the val of vertex:1 2 3 4  
  
enter adjacency 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 SYLLABI**

## EXPERIMENT 1

### 1.1 OBJECTIVE

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

### 1.2 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

### 2.1 OBJECTIVE

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

### 2.2 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

### 3.1 OBJECTIVE

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

### 3.2 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

### 4.1 OBJECTIVE

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

### 4.2 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 - Display  
 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

### 5.1 OBJECTIVE

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

### 5.2 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

```

## **INPUT OUTPUT**

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

### 6.1 OBJECTIVE

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

### 6.2 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 (ladradox 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 (ladradox 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 (ladradox sort) : 27 64 91 129 325 345 985

## EXPERIMENT 7

### 7.1 OBJECTIVE

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

### 7.2 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);
    fflush();
    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

**ANNEXURE-I**  
**Assessment Rubric**

**1. Laboratory Report**

<b>Category</b>	<b>Outstanding (Up to 100%)</b>	<b>Accomplished (Up to 75%)</b>	<b>Developing (Up to 50%)</b>	<b>Beginner (Up to 25%)</b>
Write up format	Aim, Apparatus, material requirement, theoretical basis, procedure of experiment, sketch of the experimental setup etc. is demarcated and presented in clearly labeled and neatly organized sections.	The write up follows the specified format but a couple of the specified parameters are missing.	The report follows the specified format but a few of the formats are missing and the experimental sketch is not included in the report	The write up does not follow the specified format and the presentation is shabby.
Observations and Calculations	The experimental observations and calculations are recorded in neatly prepared table with correct units and significant figures. One sample calculation is explained by substitution of values	The experimental observations and calculations are recorded in neatly prepared table with correct units and significant figures but sample calculation is not shown	The experimental observations and calculations are recorded neatly but correct units and significant figures are not used. Sample calculation is also not shown	The experimental observations and results are recorded carelessly. Correct units significant figures are not followed and sample calculations not shown

Results and Graphs	Results obtained are correct within reasonable limits. Graphs are drawn neatly with labeling of the axes. Relevant calculations are performed from the graphs. Equations are obtained by regression analysis or curve fitting if relevant	Results obtained are correct within reasonable limits. Graphs are drawn neatly with labeling of the axes. Relevant calculations from the graphs are incomplete and equations are not obtained by regression analysis or curve fitting	Results obtained are correct within reasonable limits. Graphs are not drawn neatly and or labeling is not proper. No calculations are done from the graphs and equations are not obtained by regression analysis or curve fitting	Results obtained are not correct within reasonable limits. Graphs are not drawn neatly and or labeling is not proper. No calculations are done from the graphs and equations are not obtained by regression analysis or curve fitting
Discussion of results	All relevant points of the result are discussed and justified in light of theoretical expectations. Reasons for divergent results are identified and corrective measures discussed.	Results are discussed but no theoretical reference is mentioned. Divergent results are identified but no satisfactory reasoning is given for the same.	Discussion of results is incomplete and divergent results are not identified.	Neither relevant points of the results are discussed nor divergent results identified

## 2. Individual Presentation

Category	Outstanding (Up to 100%)	Accomplished (Up to 75%)	Developing (Up to 50%)	Beginner (Up to 25%)
Content	A concise summary of the topic; Convincing justification for choice of topic; Comprehensive and complete coverage of information	A good summary of the topic; Acceptable justification for choice of topic; Most important information covered; Little irrelevant information	Informative but much of the information irrelevant; Confused justification for choice of topic; Coverage of some of the major points	A brief look at the topic; Little justification for choice of topic; Majority of information irrelevant and significant points left out
Organization	Clear purpose and subject; Pertinent examples, facts, and/or statistics; Supports conclusions/ideas with evidence	Somewhat clear purpose and subject; Some examples, facts, and/or statistics that support the subject; Some data or evidence that supports conclusions	Attempts to define purpose and subject; Weak examples, facts, and/or statistics not adequately supporting the subject; Very thin data or evidence to support conclusion	Subject and purpose not clearly defined; Weak or no support of subject; Insufficient support for ideas or conclusions
Visual Aids	Information is clear and concise with proper key information in points or phrases; Visually appealing/engaging	Too much information in complete sentences on slides along with proper key information in phrases; Significant visual appeal	Too much information in complete sentences on many slides; Some proper key information; Minimal effort made to make slides appealing	Too much information in complete sentences on slides; No or few proper key information; Repetition of the same information on multiple slides; No visual appeal
Delivery Style	Regular eye contact; Appropriate speaking volume & body language; Proper pace and diction; Fluent avoidance of repetitions, hesitations, gap fillers	Steady eye contact; Adequate volume and energy; Generally good pace and diction; Few or no distracting gestures; Few repetitions, hesitations, gap fillers	More volume or energy needed at times; Pace too slow or fast; Some distracting gestures or posture; Some repetitions, hesitations, gap fillers	Low volume and energy; Pace too slow or fast; Poor diction; Lots of distracting gestures or posture; Frequent repetitions, hesitations, gap fillers

Question-answer Session	Demonstrates knowledge by answering all types of questions with explanations and elaboration in professional manner	Is at ease with expected answers to all questions without elaboration in somewhat professional manner	Is uncomfortable with information and can answer only rudimentary questions	Does not have grasp of information and cannot answer questions about subject
-------------------------	---	---	---	--

## ANNEXURE-II

### Program Outcomes

**Program Outcomes (POs)** represent the knowledge, skills and attitudes the students should have at the end of a four year engineering program. CSE program of BAUET has 12 Program Outcomes. They are briefly described in the following table.

Sl. No	PO	Category	Description
1	PO 1	Engineering Knowledge	Apply the <b>knowledge</b> of <b>mathematics, science, engineering fundamentals</b> , and an <b>engineering specialization</b> to the solution of complex engineering problems.
2	PO 2	Problem Analysis	<b>Identify</b> , formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions <b>using first principles</b> of mathematics, natural sciences, and engineering sciences.
3	PO 3	Design/Development of Solutions	<b>Design solutions</b> for complex engineering problems and <b>design system</b> components or processes that meet the specified needs with appropriate consideration for public health and safety as well as cultural, social and environmental concerns.
4	PO 4	Investigation	<b>Conduct investigations</b> of complex problems, considering design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
5	PO 5	Modern Tool Usage	<b>Create, select, and apply</b> appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.



6	<b>PO 6</b>	<b>The Engineer and Society</b>	<b>Apply reasoning</b> 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	<b>PO 7</b>	<b>Environment and Sustainability</b>	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of need for sustainable development.
8	<b>PO 8</b>	<b>Ethics</b>	Apply ethical principles and commit to professional Ethics and responsibilities and norms of the engineering practice.
9	<b>PO 9</b>	<b>Individual and Team Work</b>	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10	<b>PO 10</b>	<b>Communication</b>	Communicate effectively on complex engineering activities with the engineering community and with society at large. Some of them are, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11	<b>PO 11</b>	<b>Project Management and Finance</b>	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	<b>PO 12</b>	<b>Life Long Learning</b>	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

### ANNEXURE-III

## Knowledge Profile, Complex Engineering Problem and Complex Engineering Activities

### Knowledge Profile (WK/K)- CHARACTERISTIC

<b>WK1</b>	<b>Natural Sciences</b>	A systematic, theory-based understanding of the natural sciences applicable to the discipline
<b>WK2</b>	<b>Mathematics</b>	Conceptually-based mathematics, numerical analysis, statistics and formal aspects of computer and information science to support analysis and modelling applicable to the discipline
<b>WK3</b>	<b>Engineering fundamentals</b>	A systematic, theory-based formulation of engineering fundamentals required in the engineering discipline
<b>WK4</b>	<b>Specialist knowledge</b>	Engineering specialist knowledge that provides theoretical frameworks and bodies of knowledge for the accepted practice areas in the engineering discipline; much is at the forefront of the discipline.
<b>WK5</b>	<b>Engineering design</b>	Knowledge that supports engineering design in a practice area.
<b>WK6</b>	<b>Engineering practice</b>	Knowledge of engineering practice (technology) in the practice areas in the engineering discipline
<b>WK7</b>	<b>Comprehension</b>	Comprehension of the role of engineering in society and identified issues in engineering practice in the discipline: ethics and the professional responsibility of an engineer to public safety; the impacts of engineering activity: economic, social, cultural, environmental and sustainability
<b>WK8</b>	<b>Research literature</b>	Engagement with selected knowledge in the research literature of the discipline

### Complex Engineering Problem (WP/P)

<b>WP</b>	<b>Preamble</b>	<b>COMPLEX PROBLEMS have characteristic of WP1 and some or all of WP2 to WP7</b>
<b>WP1</b>	<b>Depth of Knowledge</b>	<b>In-depth engineering knowledge</b> at the level of one or more of WK3, WK4, WK5, WK6 or WK8 which allows a fundamental based, first principles analytical approach
<b>WP2</b>	<b>Conflicting requirement</b>	Wide-ranging or <b>conflicting</b> technical, engineering and other issues

<b>WP3</b>	<b>Depth of analysis</b>	<b>no obvious solution</b> and require abstract thinking, originality in analysis to formulate suitable models
<b>WP4</b>	<b>Familiarity of issues</b>	<b>infrequently</b> encountered issues
<b>WP5</b>	<b>Extent of applicable codes</b>	outside problems encompassed by standards and codes of practice for professional engineering
<b>WP6</b>	<b>Extent of stakeholder</b>	<b>diverse groups</b> of stakeholders with widely varying needs
<b>WP7</b>	<b>Interdependence</b>	high level problems including <b>many component parts</b> or sub-problems

### Complex Engineering Activities (EA)

Activ ities	Preamble	<b>Complex activities</b> means (engineering) activities or projects that have <b>some or all</b> of the following characteristics listed below
<b>EA1</b>	<b>Range of resources</b>	<b>Diverse resources</b> (people, money, equipment, materials, information and technologies).
<b>EA2</b>	<b>Level of interaction</b>	Require resolution of significant problems arising from interactions between <b>wide ranging</b> or <b>conflicting</b> technical, engineering or other issues.
<b>EA3</b>	<b>Innovation</b>	Involve creative use of engineering principles and <b>research-based</b> knowledge in <b>novel</b> ways.
<b>EA4</b>	<b>Consequences to society and the environment</b>	Have <b>significant consequences</b> in a <b>range of contexts</b> , characterised by <b>difficulty</b> of prediction and mitigation.
<b>EA5</b>	<b>Familiarity</b>	Can extend <b>beyond previous</b> experiences by applying <b>principles-based</b> approaches.

\*\*\*The End\*\*\*