GLOBAL ACADEMY OF TECHNOLOGY
BENGALURU-560098

(Autonomous Institution affiliated to VTU, Belagavi)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

DATA STRUCTURES LABORATORY
SUB CODE: BADL24305
Scheme: 2024

III SEMESTER B.E

LAB INSTRUCTORS MANUAL

2025-26

Prepared by: Prof. Madhunandana and Prof. Shaik Abdul Wahab

ACADEMIC YEAR 2025-26

DEPARTMENT VISION

To achieve high levels of quality education by utilizing cutting-edge technology, cultivating a collaborative culture, and propagating customer-focused innovations in order to benefit society as a whole.

DEPARTMENT MISSION

1. To strengthen the theoretical and practical aspects of the learning process by strong research culture in collaboration with communities to build healthy and sustainable world.

2. To Contribute towards greater association between academia and businesses to establish entrepreneurship among young minds.

3. Concentrate efforts towards application areas in health care, agriculture, transport and environment.

PROGRAM EDUCATIONAL OBJECTIVES(PEOs)

1. Ability to exhibit professional skills as Data Analysts to develop intelligent solutions for various engineering and science applications. ‘

2. Imbibe passion for life-long learning, innovation, career-growth, leadership qualities, and professional ethics.

3. Efficient team leaders, effective communicators and capable of working in multidisciplinary environment as entrepreneurs and contribute to cutting edge technologies and society.

PROGRAM SPECIFIC OUTCOMES

1. Design, develop and implement applications and system software.

2. Develop intelligent solutions using Data Science technologies to cater the societal needs.

PROGRAM OUTCOMES

PO1: Engineering Knowledge:

Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem Analysis:

Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable
development (WK1 to WK4).

PO3: Design/Development of Solutions:

Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs

with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required (WK5).

PO4: Conduct Investigations of Complex Problems:

Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions (WK8).

PO5: Engineering Tool Usage:

Create, select, and apply appropriate techniques, resources, and modern engineering & IT tools, including prediction, and modelling recognizing their limitations to solve complex engineering problems (WK2 and WK6).

PO6: The Engineer and The World:

Analyse and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment (WK1, WK5, and WK7).

PO7: Ethics:

Apply ethical principles and commit to professional ethics, human values, diversity, and inclusion; adhere to national & international laws (WK9).

PO8: Individual and Collaborative Teamwork:

Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication:

Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences.

PO10: Project Management and Finance:

Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning:

Recognize the need for and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies, and iii) critical thinking in the broadest context of technological change.

COURSE OUTCOMES

CO1	Apply the concepts of Stacks to solve a problem.
CO2	Demonstrate the implementation and application of queue data structures, including their variations, to solve real-world problems
CO3	Investigate the various types of lists and their application in real world.
CO4	Analyze and implement binary tree structures to optimize searching, traversal, and hierarchical data representation.
CO5	Design and develop efficient algorithms using advanced data structures like hash tables, heaps, and balanced trees.

Low-1 Medium-2 High-3

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11
CO1	3	3	3	1	2			1			2
CO2	3	3	3	1	2			1			2
CO3	3	3	3	1	2			1			2
CO4	3	3	3	1	2			1			2
CO5	3	3	3	1	2			1			2

LIST OF PROGRAMS

Sl. No.	Program	Page No
1	Develop a menu driven Program in C for the following operations on STACK of Integers(Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate Overflow and Underflow situations on Stack d. Display the status of Stack f. Exit.Support the program with appropriate functions for each of the above operations	
2	Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands	
3.	Develop a Program in C to evaluate of Suffix expression with single digit operands and operators: +, -, *, /, % and ^.	
4.	Develop recursive program in C to i) To Find GCD of 2 numbers ii) To Solve the Tower of Hanoi Problem	
5.	Develop a menu driven Program in C for the following operations on QUEUE of integers (Array Implementation of QUEUE with maximum size MAX) a. Enqueue an Element on to Queue b. Dequeue an Element from Queue c. Demonstrate Overflow and Underflow situations on Queue d. Display the status of Queue f. Exit. Support the program with appropriate functions for each of the above operations	
6.	Implement a program to multiply two polynomials using singly linked list.	
7.	Design a doubly linked list to represent sparse matrix. Each node in the list can have the row and column index of the matrix element and the value of the element. Print the complete matrix as the output.	
8.	Write a C program to create Binary Tree and to traverse the tree using In-order, Preorder and Post order (recursively).	
9.	Write a C program to implement priority queue using Heap.	
10.	Write a C program to implement Hashing using Linear probing. Implement insertion, deletion, search and display.	

1. Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a. Push an Element on to Stack

b. Pop an Element from Stack

c. Demonstrate Overflow and Underflow situations on Stack

d. Display the status of Stack

f. Exit.Support the program with appropriate functions for each of the above operations

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
struct stack
{
    int top;
    int data[SIZE];
};
typedef struct stack STACK;

void push(STACK *s,int item)
{
    if(s->top==SIZE-1)
        printf("\n Stack Overflow");
    else
    {
        s->top=s->top+1;
        s->data[s->top]=item;
     }
}
void pop(STACK *s)
{
    if(s->top==-1)
    {
        printf("\n Stack Underflow");
    }
    else
    {
        printf("\n Element popped is %d",s->data[s->top]);
        s->top=s->top-1;
    }
}

void display(STACK s)
{
    int i;
    if(s.top==-1)
        printf("\n Stack is Empty");
    else
    {
        printf("\n Stack content are\n");
        for(i=s.top;i>=0;i--)
            printf("%d\n",s.data[i]);
    }
}
```

```
int main()
{
    char item,del;
    int ch;
    STACK s;
    s.top=-1;
    for(;;)
    {
    printf("\n1. Push\n2. POP\n3. Display\n4.Exit");
    printf("\nRead Choice :");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:printf("\n Read element to be Pushed :");
               scanf("%d",&item);
               push(&s,item);
               break;
        case 2:pop(&s);
                break;
        case 3:display(s);
                break;
        default:exit(0);
    }
    }
    return 0;

}
```

OUTPUT :

```
1. Push
2. POP
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :10

1. Push
2. POP
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :20

1. Push
2. POP
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :30

1. Push
2. POP
```

```
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :40

1. Push
2. POP
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :50

1. Push
2. POP
3. Display
4.Exit
Read Choice :1

 Read element to be Pushed :60

 Stack Overflow
1. Push
2. POP
3. Display
4.Exit
Read Choice :3

 Stack content are
50
40
30
20
10

1. Push
2. POP
3. Display
4.Exit
Read Choice :2

 Element popped is 50
1. Push
2. POP
3. Display
4.Exit
Read Choice :2

 Element popped is 40
1. Push
2. POP
3. Display
4.Exit
```

2. Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define SIZE 20
struct stack
{
    int top;
    char data[SIZE];
};
typedef struct stack STACK;

void push(STACK *s,char item)
{
    s->data[++(s->top)]=item;
}

char pop(STACK *s)
{
    return s->data[(s->top)--];
}

int preced(char sym)
{
    switch(sym)
    {
        case '^': return 5;
        case '*':
        case '/': return 3;
        case '+':
        case '-': return 1;
    }
}
void infixtopostfix(STACK *s,char infix[15])
{
    int i,j=0;
    char symbol,temp,postfix[15];
    for(i=0;infix[i]!='\0';i++)
    {
        symbol=infix[i];
        if(isalnum(symbol))
            postfix[j++]=symbol;
        else
        {
            switch(symbol)
            {
                case '(':push(s,symbol);
                break;
                case ')':temp=pop(s);
                while(temp!='(')
                {
```

```
postfix[j++]=temp;
                    temp=pop(s);
                }
                break;
                case '+':
                case '-':
                case '*':
                case '/':
                case '^': if(s->top == -1 || s->data[s->top] == '(')
                            push(s,symbol);
                          else
                          {
                              while(preced(s->data[s->top]) >= preced(symbol) &&
s->top!=-1 && s->data[s->top] != '(')
                                        postfix[j++]=pop(s);
                              push(s,symbol);
                          }
                          break;
            }
        }
    }
    while(s->top != -1)
        postfix[j++]=pop(s);
    postfix[j]='\0';
    printf("\n The postfix expression is : %s\n",postfix);
}

int main()
{
    char infix[15];
    STACK s;
    s.top=-1;
    printf("\n Read the infix expression\n");
    scanf("%s",infix);
    infixtopostfix(&s,infix);
    return 0;
}
```

OUTPUT :

Read the infix expression
a+b*c

The postfix expression is : abc*+

3. Develop a Program in C to evaluate of Suffix expression with single digit operands and operators: +, -, *, /, % and ^.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#define SIZE 20
struct stack
{
    int top;
    float data[SIZE];
};
typedef struct stack STACK;
void push(STACK *s,float item)
{
    s->data[++(s->top)]=item;
}

float pop(STACK *s)
{
    return s->data[(s->top)--];
}

float operate(float op1,float op2,char symbol)
{
    switch(symbol)
    {
        case '+':return op1+op2;
        case '-':return op1-op2;
        case '*':return op1*op2;
        case '/':return op1/op2;
        case '^':return pow(op1,op2);
    }
}

float eval(STACK *s,char postfix[SIZE])
{
    int i;
    char symbol;
    float res,op1,op2;
    for(i=0;postfix[i]!='\0';i++)
    {
        symbol=postfix[i];
        if(isdigit(symbol))
           push(s,symbol-'0');
        else
        {
            op2=pop(s);
            op1=pop(s);
            res=operate(op1,op2,symbol);
            push(s,res);
        }
```

```
    }
    return pop(s);
}

int main()
{
    char postfix[SIZE];
    STACK s;
    float ans;
    s.top=-1;
    printf("\n Read postfix expr\n");
    scanf("%s",postfix);
    ans=eval(&s,postfix);
    printf("\n The final answer is %f\n",ans);
    return 0;
}
```

OUTPUT :

```
Read postfix expr
234*+

The final answer is 14.000000
```

4. Develop recursive program in C to

i) To Find GCD of 2 numbers

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

int gcd(int a,int b)
{
    if(b!=0)
        return gcd(b,a % b);
    return a;
}
int main()
{
    int a,b;
    printf("\n Read two numbers:");
    scanf("%d%d",&a,&b);
    printf("\n GCD of %d and %d is %d\n",a,b,gcd(a,b));
    return 0;
}
```

OUTPUT :

```
Read two numbers:16 40

GCD of 16 and 40 is 8
```

ii) To Solve the Tower of Hanoi Problem

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

void towerofHanoi(int n,char source,char temp,char destination)
{
    if(n==1)
        printf("\n Move %d disc from %c to %c",n,source,destination);
    else
    {
        towerofHanoi(n-1,source,destination,temp);
        printf("\n Move %d disc from %c to %c",n,source,destination);
        towerofHanoi(n-1,temp,source,destination);
    }
}

int main()
{
    int n;
    printf("\n Read number of discs:");
    scanf("%d",&n);
    towerofHanoi(n,'S','T','D');
    return 0;
}
```

OUPUT :

Read number of discs:3

Move 1 disc from S to D
Move 2 disc from S to T
Move 1 disc from D to T
Move 3 disc from S to D
Move 1 disc from T to S
Move 2 disc from T to D

5. Develop a menu driven Program in C for the following operations on QUEUE of integers (Array Implementation of QUEUE with maximum size MAX)
a. Enqueue an Element on to Queue
b. Dequeue an Element from Queue
c. Demonstrate Overflow and Underflow situations on Queue
d. Display the status of Queue
f. Exit. Support the program with appropriate functions for each of the above operations

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
struct queue
{
    int front,rear;
    int data[SIZE];
};
typedef struct queue QUEUE;

void enqueue(QUEUE *q,int item)
{
    if(q->rear==SIZE-1)
        printf("\n Queue full");
    else
    {
        q->rear=q->rear+1;
        q->data[q->rear]=item;
        if(q->front==-1)
            q->front=0;
    }
}

void dequeue(QUEUE *q)
{
    if(q->front==-1)
    {
        printf("\n Queue empty");
    }
    else
    {
  printf("\n Deleted element is %d",q->data[q->front]);
        if(q->front==q->rear)
        {
            q->front=-1;
            q->rear=-1;
        }
```

```
else
            q->front=q->front+1;
    }
}

void display(QUEUE q)
{
    int i;
    if(q.front==-1)
        printf("\n Queue Empty");
    else
    {
        printf("]\ Queue content are\n");
        for(i=q.front;i<=q.rear;i++)
            printf("%d\t",q.data[i]);
    }
}
int main()
{
    int item,ch;
    QUEUE q;
    q.front=-1;
    q.rear=-1;
    for(;;)
    {
    printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4.Exit");
    printf("\nRead Choice :");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:printf("\n Read element to be inserted :");
               scanf("%d",&item);
               enqueue(&q,item);
               break;
        case 2:dequeue(&q);
               break;
        case 3:display(q);
                break;
        default:exit(0);
    }
    }
    return 0;

}
```

OUTPUT :

1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice :1

Read element to be inserted :10

1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice :1

Read element to be inserted :20

1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice :1

Read element to be inserted :30

1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice :3
] Queue content are
10 20 30
1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice :2

Deleted element is 10
1. Enqueue
2. Dequeue
3. Display
4.Exit
Read Choice : 4

6. Implement a program to multiply two polynomials using singly linked list.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
    int co,po;
    struct node *addr;
};
typedef struct node *NODE;
NODE insertend(NODE start,int co,int po)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->co=co;
    temp->po=po;
    temp->addr=NULL;
    if(start==NULL)
        return temp;
    cur=start;
    while(cur->addr!=NULL)
        cur=cur->addr;
    cur->addr=temp;
    return start;
}

void display(NODE start)
{
    NODE temp;
    if(start==NULL)
        printf("\n Polynomial Empty");
    else
    {
        temp=start;
        while(temp->addr!=NULL)
        {
            printf("%dx^%d+",temp->co,temp->po);
            temp=temp->addr;
        }
            printf("%dx^%d\n",temp->co,temp->po);
    }
}

NODE addterm(NODE res,int co,int po)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->co=co;
    temp->po=po;
    temp->addr=NULL;
    if(res==NULL)
        return temp;
    cur=res;
```

```
while(cur!=NULL)
    {
        if(cur->po==po)
        {
            cur->co=cur->co+co;
            return res;
        }
        cur=cur->addr;
    }
    if(cur==NULL)
        res=insertend(res,co,po);
    return res;
}
NODE multiply(NODE poly1,NODE poly2)
{
    NODE p1,p2,res=NULL;
    for(p1=poly1;p1!=NULL;p1=p1->addr)
        for(p2=poly2;p2!=NULL;p2=p2->addr)
             res=addterm(res,p1->co*p2->co,p1->po+p2->po);
    return res;
}
int main()
{
    NODE poly1=NULL,poly2=NULL,poly;
    int co,po;
    int i,n,m;
    printf("\nRead no of terms of first polynomial:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n Read CO and PO of %d term : ",i);
        scanf("%d%d",&co,&po);
        poly1=insertend(poly1,co,po);
    }
    printf("\n First polynomial is\n");
    display(poly1);
    printf("\nRead no of terms of second polynomial:");
    scanf("%d",&m);
    for(i=1;i<=m;i++)
    {
        printf("\n Read CO and PO of %d term : ",i);
        scanf("%d%d",&co,&po);
        poly2=insertend(poly2,co,po);
    }
    printf("\n Second polynomial is\n");
    display(poly2);
    poly=multiply(poly1,poly2);
    printf("\n Resultant polynomial is\n");
    display(poly);
    return 0;

}
```

OUTPUT :

Read no of terms of first polynomial:3

Read CO and PO of 1 term : 2 2

Read CO and PO of 2 term : 3 1

Read CO and PO of 3 term : 7 0

First polynomial is
2x^2+3x^1+7x^0

Read no of terms of second polynomial:2

Read CO and PO of 1 term : 3 3

Read CO and PO of 2 term : 4 1

Second polynomial is
3x^3+4x^1

Resultant polynomial is
6x^5+29x^3+9x^4+12x^2+28x^1

7. Design a doubly linked list to represent sparse matrix. Each node in the list can have the row and column index of the matrix element and the value of the element. Print the complete matrix as the output.

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

struct node
{
    int row,col,data;
    struct node *next;
    struct node *prev;
};
typedef struct node *NODE;


NODE insertend(NODE start,int row,int col,int item)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->row=row;
    temp->col=col;
    temp->data=item;
    temp->next=NULL;
    temp->prev=NULL;
    if(start == NULL)
        return temp;
    cur=start;
    while(cur->next!=NULL)
        cur = cur->next;
    cur->next=temp;
    temp->prev=cur;
    return start;
}


void display(NODE start)
{
    NODE temp;
    if(start==NULL)
        printf("\n list is empty");
    else
    {
        printf("\nROW\tCOL\tDATA\n");
        temp=start;
        while(temp!=NULL)
        {
            printf("%d\t%d\t%d\n",temp->row,temp->col,temp->data);
            temp=temp->next;
        }
    }
}
```

```
void displaymatrix(NODE start,int m,int n)
{
    NODE temp;
    int i,j;
    temp=start;
    printf("\n The Sparse matrix is\n");
    for(i=1;i<=m;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(temp!=NULL && temp->row == i && temp->col == j)
                {
                    printf("%d\t",temp->data);
                    temp=temp->next;
                }
                else
                    printf("0\t");
       }
     printf("\n");
    }

}


int main()
{
    NODE start = NULL;
    int i,j,m,n,item;
    printf("\n Read the order of the matrix\n");
    scanf("%d%d",&m,&n);
    printf("\n Read the matrix\n");
    for(i=1;i<=m;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&item);
            if(item!=0)
                start=insertend(start,i,j,item);
         }
    }
    display(start);
    displaymatrix(start,m,n);
    return 0;
}
```

OUTPUT :

Read the order of the matrix
4 4

Read the matrix
0 0 10 0
0 0 0 0
20 0 0 0
0 0 0 30

ROW	COL	DATA
1	3	10
3	1	20
4	4	30

The Sparse matrix is

0	0	10	0
0	0	0	0
20	0	0	0
0	0	0	30

8. Write a C program to create Binary Tree and to traverse the tree using In-order, Preorder and Post order (recursively).

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
typedef struct node *NODE;

NODE create_node(int item)
{
    NODE temp;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->left=NULL;
    temp->right=NULL;
    return temp;
}

NODE Insertbst(NODE root,int item)
{
    NODE temp;
    temp=create_node(item);
    if(root==NULL)
        return temp;
    else
    {
       if(item < root->data)
            root->left=Insertbst(root->left,item);
        else
            root->right=Insertbst(root->right,item);
    }
    return root;

}

void preorder(NODE root)
{
    if(root!=NULL)
    {
        printf("%d\t",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(NODE root)
{
    if(root!=NULL)
    {
        inorder(root->left);
```

```
printf("%d\t",root->data);
        inorder(root->right);
    }
}

void postorder(NODE root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d\t",root->data);
    }
}


int main()
{
    NODE root = NULL;
    int ch,item;
    for(;;)
    {
        printf("\n 1. Insert");
        printf("\n 2. Preorder");
        printf("\n 3. Inorder");
        printf("\n 4. Postorder");
        printf("\n 5. Exit");
        printf("\n Read ur choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read element to be inserted :");
                scanf("%d",&item);
                root=Insertbst(root,item);
                break;
            case 2:printf("\n The Preorder traversal is\n");
                 preorder(root);
                 break;
            case 3:printf("\n The Inorder traversal is\n");
                 inorder(root);
                 break;
            case 4:printf("\n The Postorder traversal is\n");
                 postorder(root);
                 break;
            default :exit(0);
        }
    }
    return 0;
}
```

OUTPUT :

1. **Insert**
2. **Preorder**
3. **Inorder**

4. Postorder
5. Exit
Read ur choice:1

Read element to be inserted :100

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Exit
Read ur choice:1

Read element to be inserted :50

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Exit
Read ur choice:1

Read element to be inserted :200

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Exit
Read ur choice:1

Read element to be inserted :150

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Exit
Read ur choice:3

The Inorder traversal is
50 100 150 200

9. Write a C program to implement priority queue using Heap.

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

void heapify(int a[10],int n)
{
    int i,k,v,j,flag=0;
    for(i=n/2;i>=1;i--)
    {
        k=i;
        v=a[k];
        while(!flag && 2*k <= n)
        {
            j=2*k;
            if(j<n)
            {
                if(a[j]<a[j+1])
                    j=j+1;
            }
            if(v>=a[j])
                flag=1;
            else
            {
                a[k]=a[j];
                k=j;
            }
        }
        a[k]=v;
        flag=0;
    }
}

int main()
{
    int n,i,a[10],ch;
    for(;;)
    {
        printf("\n 1. Create Heap");
        printf("\n 2. Extractmax");
        printf("\n 3. Exit");
        printf("\n Read Choice :");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read no of elements :");
                   scanf("%d",&n);
                   printf("\n Read Elements\n");
                   for(i=1;i<=n;i++)
                    scanf("%d",&a[i]);
                   heapify(a,n);
                   printf("\n Elements after heap\n");
                   for(i=1;i<=n;i++)
                    printf("%d\t",a[i]);
```

```
break;
            case 2:if(n>=1)
                    {
                        printf("\n Element deleted is %d\n",a[1]);
                        a[1]=a[n];
                        n=n-1;
                        heapify(a,n);
                        if(n!=0)
                        {
                            printf("\n Elements after reconstructing heap\n");
                            for(i=1;i<=n;i++)
                               printf("%d\t",a[i]);
                        }
                    }
                    else
                        printf("\n No element to delete");
                    break;
            default:exit(0);

        }
    }
    return 0;
}
```

OUTPUT :

1. Create Heap
2. Extractmax
3. Exit
Read Choice :1
Read no of elements :7
Read Elements
50
40
20
80
10
60
30

Elements after heap
80 50 60 40 10 20 30
1. Create Heap
2. Extractmax
3. Exit
Read Choice :2

Element deleted is 80

Elements after reconstructing heap
60 50 30 40 10 20

10. Write a C program to implement Hashing using Linear probing. Implement insertion, deletion, search and display.

```
#include <stdio.h>
#include <stdlib.h>
int tsize,hash[10],count;
int Cal_Hash(int key)
{
return key % tsize;
}
int Cal_ReHash(int key)
{
return (key+1)%tsize ;
}
void insert(int key)
{
int index;
if(count!=tsize)
{
index = Cal_Hash(key);
while (hash[index]!=-1)
{
index = Cal_ReHash(index);
}
hash[index]=key;
count++;
}
else
printf("\n Table is full");
}
int search(int key)
{
int i,index,loc=-1;
index = Cal_Hash(key);
for(i=0;i<tsize; i++)
{
loc=(index+i)%tsize;
if(hash[loc]==key)
{
return loc;
}
}
return -1;
}

void delete(int key)
{
int loc;
if(count == 0)
printf("\n Hash table empty,can't delete");
else
{
loc=search(key);
if(loc!=-1)
{
```

```
hash[lo
c]=-1;
count--;
printf("\n Key deleted");
}
else
printf("\n Key not found");
}
}
void display()
{
int i;
printf("\nThe elements in the hashtable are: ");
for (i=0;i<tsize;i++)
{
printf("\n Element at position %d: %d",i,hash[i]);
}
}
int main()
{
    int key,i,j,ch,index,loc;
    printf ("Enter the size of the hash table:  ");
    scanf ("%d",&tsize);
    for (i=0;i<tsize;i++)
        hash[i]=-1 ;
    for(;;)
    {
    printf("\n 1. Insert");
    printf("\n 2. Search");
    printf("\n 3. Delete");
    printf("\n 4. Display");
    printf("\n 5. Exit");
    printf("\n Read Choice :");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
    printf ("\nEnter the key to be inserted: ");
    scanf ("%d",&key);
    insert(key);
    break;
    case 2:
    printf("\nenter search key\n");
    scanf("%d",&key);
    loc=search(key);
    if(loc != -1)
        printf("\nkey is found at index %d",loc);
    else
        printf("\n Key not found");
    break;
    case 3:
    printf("\nenter key to be deleted\n");
    scanf("%d",&key);
    delete(key);
    break;
    case 4:
    display();
```

```
break;
     default :exit(0);
     }
     }
      return 0;
}
```

OUTPUT :

Enter the size of the hash table: 8

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Read Choice :1

Enter the key to be inserted: 10

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Read Choice :1

Enter the key to be inserted: 20

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Read Choice :1

Enter the key to be inserted: 30

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Read Choice :4

The elements in the hashtable are:
Element at position 0: -1
Element at position 1: -1
Element at position 2: 10
Element at position 3: -1
Element at position 4: 20
Element at position 5: -1
Element at position 6: 30
Element at position 7: -1
1. Insert
2. Search

```
 3.
Delete
 4. Display
 5. Exit
 Read Choice :1

Enter the key to be inserted: 40

 1. Insert
 2. Search
 3. Delete
 4. Display
 5. Exit
 Read Choice :3

enter key to be deleted
30

 Key deleted
 1. Insert
 2. Search
 3. Delete
 4. Display
 5. Exit
 Read Choice :4

The elements in the hashtable are:
 Element at position 0: 40
 Element at position 1: -1
 Element at position 2: 10
 Element at position 3: -1
 Element at position 4: 20
 Element at position 5: -1
 Element at position 6: -1
 Element at position 7: -1
```