Thadomal Shahani Engineering College
Bandra (W.), Mumbai – 400050
University of Mumbai
(A Y 2021-2022)

Division : C3

Batch : C31

Roll Number : 2003145

This is to certify that Mr. / ~~Miss~~ Ratlamwala Idris Ismail of Computers Department, Semester III with Roll No. 2003145 has completed a course of the necessary experiments in the subject **Data Structures** under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year **2021-2022 .**

Teacher In-Charge:
Prof. Anagha Durugkar

Index

Idris Ratlamwala C31-2003145

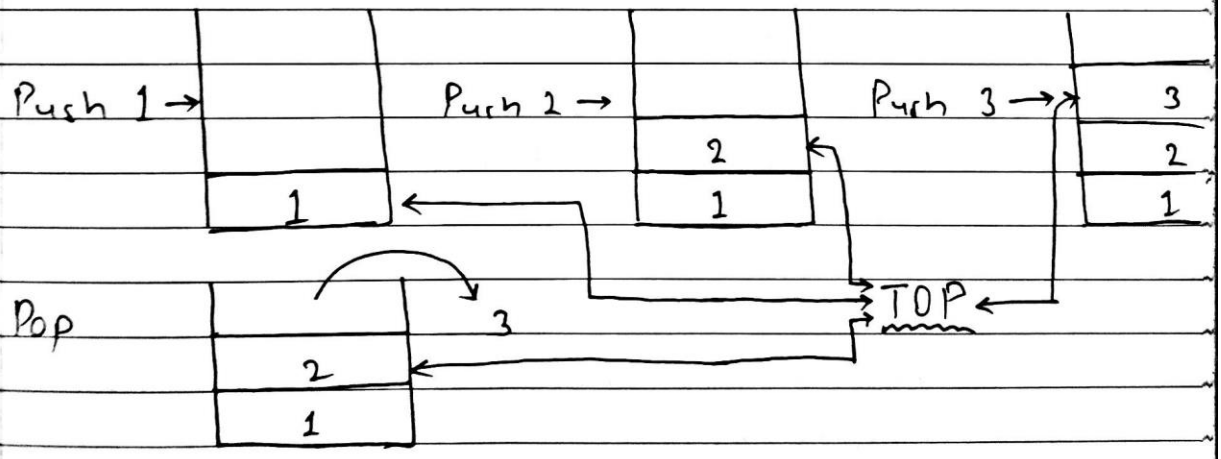
PAGE No.
DATE 1/9/17

Expt: 1

Aim: Implement stack ADT using array.

Theory: Stack is a linear data structure that follows LIFO (last in first out) principle. stack has one end unlike queue. It contains only one pointer "TOP" pointing towards the topmost element of the stack. Hence an element can only be added to the top of stack and deleted only from the top. Stack can therefore be defined as a container in which insertion and deletion can be done only from one end [TOP].

Example:



Stack operations: [ALGORITHMS]

Two main operations can be performed on the stack ie.

i) Push

ii) POp

i) Push :

Adding on element to the top of stack is referid to as push operation.

- It includes following 2 steps:
  1) Increment the value of top pointer so that it can now refer to next mem. location.
  2) Add an element at the position position of incremented top.
- Stack is overflown when we try to insert completely filed stack, hence the size of stack should be checked before incrementing the top.

Algorithm :

```
begin
    if top = n, then stack is full;
    top = top + 1
    Stack (top) = item;
end
```

ii) Pop :

Deletion of an element from the top of stack is called pop operation.

1) store the top value in another variable
2) Decrement the top by 1
- Stack is underflown when we try to delete empty stack.

Algorithm :

```
begin
    if top = 0, then stack is empty;
    item = Stack (top) ;
    top = top - 1;
end.
```

Program :

```
      #include <stdio.h>

   int main() {
       int a[20],elem,again,n,i,choice,j,temp;

       printf("Enter number of elements :");
       scanf("%d",&n);
       printf("Enter the elements :\n");
       for(i=0;i<n;i++) scanf("%d",&a[i]);

       do{
           printf("Enter your choice :\n1. search\t2. update\t3. insert\t4.
delete\t5. sort in ascending order\t6. sort in decending order\n");
           scanf("%d",&choice);
           switch (choice)
           {
           case 1:
               printf("Enter the element to be searched :");
               scanf("%d",&elem);
               for(i=0;i<n;i++)
               {
                   if(a[i]==elem)
                   {
                       printf("Index of required element is %d\n",i);
                       break;
                   }
               }
               if(i==n) printf("Not found\n");
               break;

           case 2:
               printf("Enter the index of element to be updated :");
               scanf("%d",&i);
               printf("update %d with :",a[i]);
               scanf("%d",&elem);
               a[i]=elem;
               break;

           case 3:
               printf("Enter the index where element to be inserted :");
               scanf("%d",&j);
               printf("Enter the element to be inserted :");
               scanf("%d",&elem);
               for ( i = ++n; i > j; i--)
               {
                   temp=a[i];
```

```
            a[i]=a[i-1];
            a[i-1]=temp;
        }
        a[j]=elem;
        break;

    case 4:
        printf("Enter the index of element to be deleted :");
        scanf("%d",&j);
        n--;
        for ( i = j; i < n; i++)
        {
            temp=a[i];
            a[i]=a[i+1];
            a[i+1]=temp;
        }
        break;

    case 5:
        printf("Sorting arrray in ascending order\n");
        for ( i = 0; i < n-1; i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(a[j]>a[j+1])
                {
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }
            }
        }
        break;

    case 6:
        printf("Sorting arrray in decending order\n");
        for ( i = 0; i < n-1; i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(a[j]<a[j+1])
                {
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }
            }
        }
        break;
```

```
            default:
                printf("Invalid Choice !\n");
            }
        printf("\nNew array : ");
        for(i=0;i<n;i++) printf("%d ",a[i]);

        printf("\nIf you want to traversal again enter 1 either 0 : ");
        scanf("%d",&again);
        }while(again==1);
    }
```

Output :

```
Enter the elements :
1
2
3
4
5
Enter your choice :
1. search       2. update       3. insert
4. delete       5. sort in ascending order      6. sort in decending order
1
Enter the element to be searched :3
Index of required element is 2

New array : 1 2 3 4 5
If you want to traversal again enter 1 either 0 : 1
Enter your choice :
1. search       2. update       3. insert
4. delete       5. sort in ascending order      6. sort in decending order
3
Enter the index where element to be inserted :2
Enter the element to be inserted :8

New array : 1 2 8 3 4 5
If you want to traversal again enter 1 either 0 : 1
Enter your choice :
1. search       2. update       3. insert
4. delete       5. sort in ascending order      6. sort in decending order
6
Sorting arrray in decending order

New array : 8 5 4 3 2 1
```

2003145

Expt no 2:

PAGE NO
DATE

Aim: Convert an infix expression to postfix expression.

Theory:

postfix expression: The expression of the form a b op ie. an operator is followed for every pair of operands.

Advantages of postfix over infix notation:

The compiler scans the expression either from left or right.

consider the following exp:

a + b * c + d.

The compiler first scans the exp to evaluate b * c, then again scans the expression to add a to it. The result is then added to d after another scan.

This recursive scans to perform an expression slow down the compilation and inefficient. Its better to convert infix expression to postfix form before evaluation. so that the expression is evaluated in single scan.

Algorithm:

The infix to postfix conversion takes an infix exp as input and returns a postfix expression

- When the priority of the operator is higher than the priority of operator at the top of stack then the operator is directly pushed onto the stack.

- Conversely, if the priority of infix operator is less than that of operator on top of stack, those operator are poped from the stack and then the infix operator is pushed.

- If a bracket "(" value is obtained then it is directly pushed onto the stack
- If a ")" is encounterd then all the operators above the "(" are poped to output.

Example.

Infix exp: 8 - 2 + (3 * 4) / 2 ^ 2

Character from Infix exp.	Stack	Postfix exp.
8		8
-	-	8
2	-	8 2
+	+	8 2 -
(	+ (	8 2 -
3	+ (	8 2 - 3
*	+ (*	8 2 - 3
4	+ (*	8 2 - 3 4
)	+	8 2 - 3 4 *
/	+ /	8 2 - 3 4 *
2	+ /	8 2 - 3 4 * 2
^	+ / ^	8 2 - 3 4 * 2
2	+ / ^	8 2 - 3 4 * 2 2
		8 2 - 3 4 * 2 2 ^ / +

Program :

```
   #include <stdio.h>
    #define MAX 100

    char stack[MAX];
    int top = -1,i,x;

    void push(char);
    char pop(void);
    int priority(char);
    void infixToPostfix(char exp[])
    {
        char postfix[100];
        int i=0,j=0;
        while(exp[i]!=0)
        {
            if((exp[i] >= 'a' && exp[i] <= 'z') || (exp[i] >= 'A' && exp[i] <=
'Z') || (exp[i] >= '0' && exp[i] <= '9'))
            {
                postfix[j]=exp[i];
                j++;
            }
            else if(exp[i]=='(')
            {
                push(exp[i]);

            }
            else if(exp[i]==')')
            {
                while((x = pop()) != '(')
                    {
                        postfix[j]=x;
                        j++;
                    }

            }
            else {
                while(priority(exp[i]) <= priority(stack[top])) {
                    postfix[j]=pop();
                    j++;
                }
                push(exp[i]);
            }
            i++;
        }

        while(top != -1) {
```

```
            postfix[j]=pop();
            j++;
        }
        postfix[j]=0;
        printf("The postfix of given expression is : %s\n",postfix);
    }


    int main()
    {
        char exp[100],postfix[100];
        printf("Enter the Infix Expression :");
        gets(exp);
        infixToPostfix(exp);
        return 0;
    }


    int priority(char c)
    {
        if(c=='^'||c=='$') return 3;
        else if(c=='*'||c=='/') return 2;
        else if(c=='+'||c=='-') return 1;
        else return 0;
    }

    void push(char x)
    {
        if (top == MAX - 1)
        {
            printf("\nSTACK is over flow");
            exit(0);
        }
        else
        {
            top++;
            stack[top] = x;
        }
    }

    char pop()
    {
        if (top == -1)
        {
            printf("\nStack is under flow");
            exit(0);
        }
        else return (stack[top--]);
    }
```

Output :

```
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs> & .\"InfixToPostfix.exe"
Enter the Infix Expression :a+b*c/(e+f)
The postfix of given expression is : abc*ef+/+
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs> cd "c:\Users\IsmailRatlamwala\Documents
\College prog\DS Labs"
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs> & .\"InfixToPostfix.exe"
Enter the Infix Expression :a-b*c
The postfix of given expression is : abc*-
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs>
```

2003145

PAGE NO. 1 chnf
DATE

Expt 3

Aim : Evaluate postfix expression using stack ADT.

Theory :

Compiler finds it convinient to evaluate exp. in postfix form as discussed in previous experiment.

As postfix expressions are without parentheses, they can be evaluated as 2 operands and 1 operator at a time, this becomes easier for the compiler to handle.

Algorithm:

1) Create a stack to store operands (or values)

2) Scan the given expression and do the following for each scanned element.
   i) If the element is a number, push it onto the stack.
   ii) If the element is an operand, pop 2 operands from the stack.

3) Evaluate the operation and push result back to the stack.

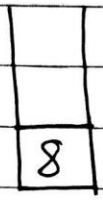
4) When the stack has ended, the number in stack is the final answer.

Example:

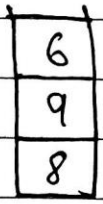
Evaluate: 2 6 + 9 6 - / ;

1) push 2
   push 6

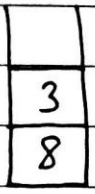
2) pop 6
   pop 2
   push (6+2)



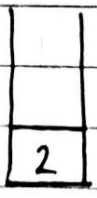
3) push 9
   push 6



4) pop 6
   pop 9
   push (9-6)



5) pop 3
   pop 8
   push (8/3)



6) pop: answer = 2

Program :

```c
#include<stdio.h>

#define size 100
int top=-1;
int arr[size];

void push(int data){
    if( top!= size-1){
        top++;
        arr[top]=data;
    }
    else{
        printf("Satck overflow!!");
    }
}
int pop(){
    if( top!=-1){
        int c = arr[ top];
        top--;
        return c;
    }
}
int postfixEvaluation(char exp[]){
    for(int i=0;i<strlen(exp);i++){
        if(exp[i]>='0'&& exp[i]<='9'){
            push(exp[i]-'0');
        }
        else{
            int op2=pop();
            int op1=pop();
            switch(exp[i]){
                case '+':
                    push(op1+op2);
                    break;
                case '-':
                    push(op1-op2);
                    break;
                case '*':
                    push(op1*op2);
                    break;
                case '/':
                    push(op1/op2);
                    break;
                case '^':
                    push(pow(op1,op2));
                    break;
```

```
            }
        }
    }
    return  arr[top];
}

int main(){
    char exp[size];
    printf("Enter the postfix expression: ");
    gets(exp);
    int result = postfixEvaluation(exp);
    printf("Evaluation :%d\n",result);
}
```

Output :

```
Enter the postfix expression: 54*
Evaluation :20
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs> cd "c:\Users\IsmailRatlamwala\Documents
\College prog\DS Labs"
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs> & .\"evaluationOfpostfix.exe"
Enter the postfix expression: 97-
Evaluation :2
```

2003145

PAGE No. 1dm>
DATE / /

Experiment 4.

Aim : Implement linear queue ~~using~~ ADT using array.

Theory :

Like stack queue is also a linear data structure which follows a particular order in which operations are performed. The order is (FIFO). A good example is a queue of ~~cust~~ consumers for a resoure, where the consumer come first is serve first.

The difference between queue is in removing elements, ~~the element from queue is removed from~~ In stacks we remove element most recently added whereas in queue we remove the element ~~e~~ least recently added.

Algorithm :

There are mainly 2 main operations performed on the queue ie.

i) Enqueue
ii) Dequeue.

i) Enqueue : Adding an element to the rear of queue is reffered to as enquing.

Algorithm:

1) Increment rear by 1.
2) Add element to the queue where the rear is pointing
3) If the queue is empty increment front from -1 to 0 :

ii) Dequeue: The deletion of data from front of queue is refered to as dequing.

Algorithm:

1) If the queue is not empty, access the data where front is pointing.
2) Increment front pointer to next available data element.
3) If enqueued element was the last element in the queue then reset front and rear to -1. in order to show underflow condition.

Program :

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

struct Queue
{
    int capcacity;
    int rear,front;
    int *arr;
};
typedef struct Queue *PtrToNode;
typedef PtrToNode queue;

queue createQueue(int max){
    queue q;
    q=(struct Queue*)malloc(sizeof(struct Queue));
    q->capcacity=max;
    q->front=-1,q->rear=-1;
    q->arr=(int*)malloc(max*sizeof(int));
    return q;
}

int isfull(queue q){
    return q->rear == q->capcacity-1;
}
int isempty(queue q){
    return q->rear ==-1 && q->front==-1;
}

queue enqueue(queue q){
    if(isfull(q)) printf("Overflow !\n");
    else{
        q->rear++;
        printf("Enter the element to be inserted : ");
        scanf("%d",&q->arr[q->rear]);
        if(q->front==-1) q->front=0;
    }
    return q;
}

queue dequeue(queue q){
    if(isempty(q)) printf("underflow !\n");
    else if(q->front==q->rear) {
        printf("Element dequeued : %d\n", q->arr[q->front]);
        q->front=q->rear=-1;
    }
    else{
```

```c
        printf("Element dequeued : %d\n", q->arr[q->front]);
        q->front++;
    }
    return q;
}

void display(queue q){
    if(isempty(q)) printf("Queue is Empty\n");
    else{
        printf("Elements in queue : ");
        for (int i = q->front; i <= q->rear; i++) printf("%d ",q->arr[i]);
        printf("\n");
    }
}

int main()
{
    int max = 10,choice;
    queue q;
    q=createQueue(max);
    do{
        printf("\n1.Enqueue  2.Dequeue  3.Display  4.Exit \n");
        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch (choice)
        {
            case 1:
                q=enqueue(q);
                break;
            case 2:
                q=dequeue(q);
                break;
            case 3:
                display(q);
                break;
            case 4:
                break;

            default:
                printf("Invalid choice !");
        }
    }while(choice!=4);
    return 0;
}
```

Output :

```
1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 2
underflow !

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 3
Queue is Empty

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 1

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 2

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 3
Elements in queue : 1 2

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 2
Element dequeued : 1

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 3
Elements in queue : 2

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice :
```

2003145

PAGE NO.
DATE

Experiment no. 5

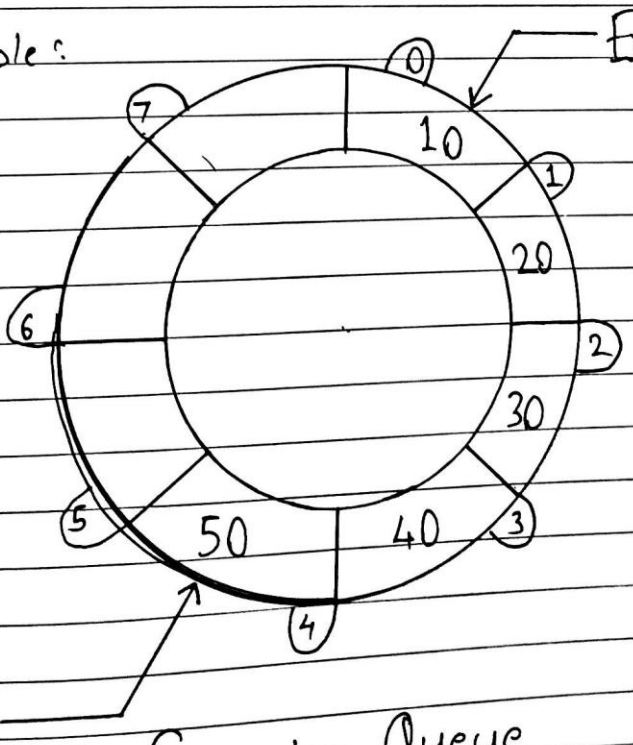
Aim : Implement circular queue ADT using array.

Theory :

In normal queue we cannot insert elements once queue is full. But, we cannot even insert element if the space is available in front of queue. If we simply increment front and the rear indices, then the front may reach the end of array and the queue once full, will be of no use even if there is empty space available at the start. The solution to this problem is to use Circular Queue.

The circular queue is same as linear queue Its just that the circular queue can grow circularly.

Example :



Circular Queue

Algorithm:

All the operations (ie Enque & Dequeue) of circular queue are same as that of linear queue (refer prev. expt.)

The only difference is the condition of overflow ~~and underflow~~.

Overflow:

```
if ((queue → rear +1) % queue → capacity == queue → front)
{
    printf("Overflow !");
}
else
enqueue(data);
```

Program :

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

struct Queue
{
    int capcacity;
    int rear,front;
    int *arr;
};
typedef struct Queue *PtrToNode;
typedef PtrToNode queue;

queue createQueue(int max){
    queue q;
    q=(struct Queue*)malloc(sizeof(struct Queue));
    q->capcacity=max;
    q->front=-1,q->rear=-1;
    q->arr=(int*)malloc(max*sizeof(int));
    return q;
}

int circularIncrement(int n,int max){
    n++;
    //printf("\ncirInc: %d\n",(n==max) ? 0 : n);
    return (n==max) ? 0 : n;
}

int isfull(queue q){
    return (q->rear+1)%q->capcacity==q->front;
}

int isempty(queue q){
    return q->rear ==-1 && q->front==-1;
}

queue enqueue(queue q){
    if(isfull(q)) printf("Overflow !\n");
    else{
        q->rear=circularIncrement(q->rear,q->capcacity);
        printf("Enter the element to be inserted : ");
        scanf("%d",&q->arr[q->rear]);
        if(q->front==-1) q->front=0;
    }
    return q;
}
```

```c
queue dequeue(queue q){
    if(isempty(q)) printf("underflow !\n");
    else if(q->front==q->rear) {
        printf("Element dequeued : %d\n", q->arr[q->front]);
        q->front=q->rear=-1;
    }
    else{
        printf("Element dequeued : %d\n", q->arr[q->front]);
        q->front=circularIncrement(q->front,q->capcacity);
    }
    return q;
}

void display(queue q){
    if(isempty(q)) printf("Queue is Empty\n");
    else{
        printf("Elements in queue : front =%d rear =%d\n",q->front,q->rear);

        int i=q->front;
        while (1)
        {
            printf("%d ",q->arr[i]);
            if(i==q->rear) break;
            i=circularIncrement(i,q->capcacity);
        }
        printf("\n");
    }
}

int main(int argc, char const *argv[])
{
    int max = 4,choice;
    queue q;
    q=createQueue(max);
    do{
        printf("\n1.Enqueue  2.Dequeue  3.Display  4.Exit \n");
        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch (choice)
        {
            case 1:
                q=enqueue(q);
                break;
            case 2:
                q=dequeue(q);
                break;
            case 3:
                display(q);
                break;
```

```
            case 4:
                break;

            default:
                printf("Invalid choice ^_^");
        }
    }while(choice!=4);

    return 0;
}
```

Output :

```
1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 1

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 2

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 3

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Overflow !

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 3
Elements in queue : front =0 rear =2
1 2 3

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 2
Element dequeued : 1

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 1
Enter the element to be inserted : 4

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice : 4
```

2003145

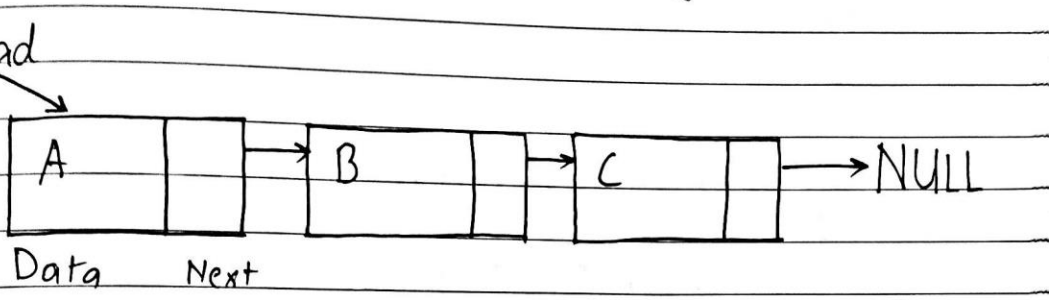
Experiment No. 6

PAGE No. / DATE — John

Aim : Implement Singly Linked List ADT.

Theory:

Like array, LL is a linear data structure, unlike array LL elements are not stored in contiguous mem location, the elements are linked using pointers.



Why Linked List ?

1) The size of array is fixed, so we must know the upper limit on the number of elements.
2) Inserting a new element in an array is expensive since whole array had to be shifted.
3) Deletion is also expensive in an array for the same reason.

Advantages over array:

1) Dynamic size
2) Ease of insertion / deletion.

Representation:

A Linked List is represented by a pointer to the first node known as head. If LL is empty then head pointer is null.

DATE / /

Each node in the list contains atleast 2 parts:
i) data
ii) pointer to the next node.

in C we can represent node using structure.

```
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
```

Program :

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

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

    struct node *head=NULL,*newnode,*ptr;

    struct node *createnode(){
        //creating a new node
        newnode = (struct node*)malloc(sizeof(struct node)); //allocates mem to
new node, newnode contains base add of new node

        printf("Enter the data : ");
        scanf("%d",&newnode->data);
        newnode->next=NULL;
        return newnode;
    }

    int createll(){
        int length=0;
        while (1)
        {
            newnode=createnode();

            if(newnode->data==-1) {
                printf("Linked List created /n");
                return length;
            };
            //new node is created
            length++;
            //Linking this node to the List
            if(head==NULL) head= ptr= newnode;
            else{
                ptr->next=newnode;   //linking previous node's next to this
(new)node
                ptr= newnode;        //updating our pointer to this
(new)node




            }
```

```
    }
}

void display(){
    ptr = head;
    printf("Elements in list are :");
    while (ptr!=NULL)
    {
        printf("%d ",ptr->data);
        ptr=ptr->next;
    }
    printf("\n");
}

void insertstart(){
    newnode=createnode();
    newnode->next = head;
    head = newnode;
}

void insertend(){
    newnode=createnode();
    ptr=head;
    while (ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=newnode;
}

void insertmidway(int after,int length,int beforeOrAfter){
    if(after>length) printf("Invalid index\n");
    else{
        ptr=head;
        for (int i = 1; i <= after-beforeOrAfter; i++)
        {
            ptr=ptr->next;
        }
        newnode=createnode();
        newnode->next= ptr->next;
        ptr->next=newnode;
    }
}

void deleteStart(){
    ptr = head->next;
    free(head);
    head = ptr;
}
```

```
void deleteEnd(){
    ptr = head;
    while (ptr->next->next!=NULL)
    {
        ptr=ptr->next;
    }
    free(ptr->next->next);
    ptr->next=NULL;
}

void delete(int at,int length){
    if(at>length) printf("Invalid index\n");
    else{
        struct node *temp;
        ptr=head;
        for (int i = 1; i <= at-2; i++)
        {
            ptr=ptr->next;
        }
        temp=ptr->next->next;
        free(ptr->next);
        ptr->next=temp;
    }
}

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

void sort(int n){
    for (int i = 0; i < n-1; i++){
        ptr=head;
        for(int j=0;j<n-i-1;j++){
            if(ptr->data > ptr->next->data){
                int temp= ptr->data;
                ptr->data=ptr->next->data;
                ptr->next->data=temp;
            }
            ptr=ptr->next;
        }
    }
    printf("After sorting in ascending order,\n");
```

```
    }

    int main()
    {
        int length,at,choice;

        do{
            printf("\nMAIN MENU :\n");
            printf("1. To Create a list \t\t\t2. To Add a Node at the beginning
\n3. To Add Node at the end \t\t4. To Add Node after the given node \n5. To Add
Node before the given node \t6. To delete a Node at the beginning \n7. To
delete a node from the end \t8. Delete a given Node\n9. Display the Linked
List\t\t10. Delete entire List\n11. Sort the list \t\t\t12. Exit\nEnter the
choice:");

            scanf("%d", &choice);

            switch (choice)
            {
            case 1:
                length= createll();
                break;
            case 2:
                insertstart();
                length++;
                break;
            case 3:
                insertend();
                length++;
                break;
            case 4:
                printf("add a new node before :\n");
                scanf("%d",&at);
                insertmidway(at,length,1);// 2 for before and 1 for after
                length++;
                break;
            case 5:
                printf("add a new node before :\n");
                scanf("%d",&at);
                insertmidway(at,length,2);// 2 for before and 1 for after
                length++;
                break;
            case 6:
                deleteStart();
                break;
            case 7:
                deleteEnd();
                break;
            case 8:
                printf("Delete node at index :\n");
```

```
            scanf("%d",&at);
            delete(at,length);
            length--;
            break;
        case 9:
            display();
            break;
        case 10:
            deletell();
            length=0;
            break;
        case 11:
            sort(length);
            display();
            break;

        }
    }while (choice != 12);



    deletell();

    display();

    return 0;
}
```

Output :

```
MAIN MENU :
1. To Create a list                          2. To Add a Node at the beginning
3. To Add Node at the end                    4. To Add Node after the given node
5. To Add Node before the given node         6. To delete a Node at the beginning
7. To delete a node from the end             8. Delete a given Node
9. Display the Linked List                   10. Delete entire List
11. Sort the list                            12. Exit
Enter the choice:1
Enter the data : 11
Enter the data : 22
Enter the data : 33
Enter the data : 44
Enter the data : -1
Linked List created

MAIN MENU :
1. To Create a list                          2. To Add a Node at the beginning
3. To Add Node at the end                    4. To Add Node after the given node
5. To Add Node before the given node         6. To delete a Node at the beginning
7. To delete a node from the end             8. Delete a given Node
9. Display the Linked List                   10. Delete entire List
11. Sort the list                            12. Exit
Enter the choice:4
add a new node before :
3
Enter the data : 88

MAIN MENU :
1. To Create a list                          2. To Add a Node at the beginning
3. To Add Node at the end                    4. To Add Node after the given node
5. To Add Node before the given node         6. To delete a Node at the beginning
7. To delete a node from the end             8. Delete a given Node
9. Display the Linked List                   10. Delete entire List
11. Sort the list                            12. Exit
Enter the choice:9
Elements in list are :11 22 33 88 44

MAIN MENU :
1. To Create a list                          2. To Add a Node at the beginning
3. To Add Node at the end                    4. To Add Node after the given node
5. To Add Node before the given node         6. To delete a Node at the beginning
7. To delete a node from the end             8. Delete a given Node
9. Display the Linked List                   10. Delete entire List
11. Sort the list                            12. Exit
Enter the choice:8
Delete node at index :
2
```

```
MAIN MENU :
1. To Create a list                         2. To Add a Node at the beginning
3. To Add Node at the end                   4. To Add Node after the given node
5. To Add Node before the given node        6. To delete a Node at the beginning
7. To delete a node from the end            8. Delete a given Node
9. Display the Linked List                  10. Delete entire List
11. Sort the list                           12. Exit
Enter the choice:11
After sorting in ascending order,
Elements in list are :11 33 44 88

MAIN MENU :
1. To Create a list                         2. To Add a Node at the beginning
3. To Add Node at the end                   4. To Add Node after the given node
5. To Add Node before the given node        6. To delete a Node at the beginning
7. To delete a node from the end            8. Delete a given Node
9. Display the Linked List                  10. Delete entire List
11. Sort the list                           12. Exit
Enter the choice:12
Elements in list are :
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Linked List>
```

2003145

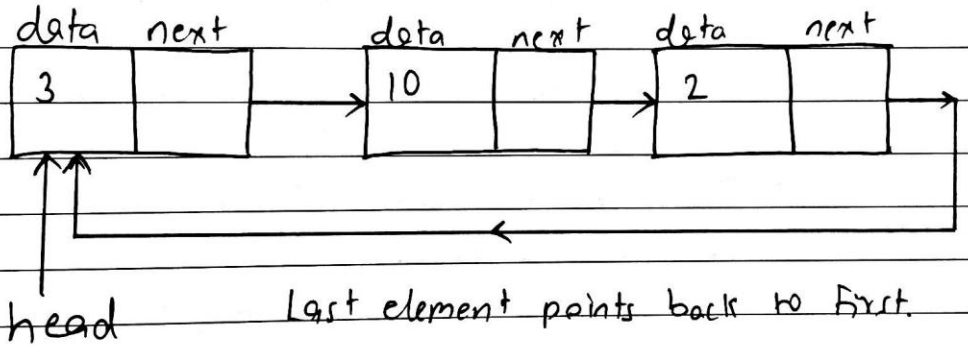
PAGE No. 10
DATE

Experiment no. 7

Aim : Implement Circular Linked List ADT.

Theory:

Circular Linked list is little more complicated Linked data structure. In the circular linked list, we can insert elements anywhere in the list because it is in contigous memory locations. In circular Linked list the previous node stores the address of next element and the last element stores the address of starting node. The elements points to each other forming a circular chain.



Algorithm:

1) Insertion:

We can use the following steps to insert a new node at the end of circular

1) Create a newNode with given value
2) check if head == NULL
3) if true then set head = newNode
   & newNode → next = head.

4) If [false] then define a node pointer temp and initilize with head.

5) Keep moving the temp to next node until it reaches the last node (ie temp → next == NULL)

6) Set temp → next = newNode
& newNode → next = head.

2) Delete :

We can follow the following steps to delete a node From beginning of circular linked list...

1) Define 2 pointers temp 1 & temp 2 and initilize both with head.

2) check if list is having only one node (temp 1 → next == head)

3) If it is true then delete temp 1 and set head = NULL.

4) If false then move the temp 1 to last node (as seen above)

5) Set head = temp 2 → next &
temp 1 → next = head.

Program :

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

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

    struct node *head=NULL,*newnode,*ptr;

    struct node *createnode(){
        //creating a new node
        newnode = (struct node*)malloc(sizeof(struct node)); //allocates mem to
new node, newnode contains base add of new node

        printf("Enter the data : ");
        scanf("%d",&newnode->data);
        newnode->next=head;
        return newnode;
    }

    int createll(){
        int length=0;
        while (1)
        {
            newnode=createnode();
            if(newnode->data==-1) {
                printf("Linked List created \n");
                return length;
            };
            length++;

            if(head==NULL) head= ptr= newnode;
            else{
                ptr->next=newnode;
                ptr=
newnode;




            }
        }
    }
```

```
void display(){
    if(head==NULL) printf("Empty List\n");
    else{
        ptr = head;
    printf("Elements in list are : ");
    while (ptr->next!=head)
    {
        printf("%d ",ptr->data);
        ptr=ptr->next;
    }
    printf("%d ",ptr->data);
    printf("\n");
    }
}

void insertstart(){
    newnode=createnode();
    newnode->next = head;

    ptr = head;
    while (ptr->next!=head) ptr=ptr->next;

    ptr->next=newnode;  //updates the value of head in last node
    head = newnode;
}

void insertend(){
    newnode=createnode();
    ptr=head;
    while (ptr->next!=head)
    {
        ptr=ptr->next;
    }
    ptr->next=newnode;
}

void insertmidway(int at,int length,int beforeOrAfter){
    if(at>length) printf("Invalid index\n");
    else{
        ptr=head;
        for (int i = 1; i <= at-beforeOrAfter; i++)
        {
            ptr=ptr->next;
        }
        newnode=createnode();
        newnode->next= ptr->next;
        ptr->next=newnode;
    }
}
```

```
void deleteStart(){
    struct node *temp = head->next;
    free(head);
    ptr = temp;
    while (ptr->next!=head) ptr=ptr->next;

    ptr->next=temp;  //updates the value of head in last node
    head = temp;
}

void deleteEnd(){
    ptr = head;
    while (ptr->next->next!=head)
    {
        ptr=ptr->next;
    }
    free(ptr->next);
    ptr->next=head;
}

void delete(int at,int length){
    if(at>length) printf("Invalid index\n");
    else{
        struct node *temp;
        ptr=head;
        for (int i = 1; i <= at-2; i++)
        {
            ptr=ptr->next;
        }
        temp=ptr->next->next;
        free(ptr->next);
        ptr->next=temp;
    }
}

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

void sort(int n){
    for (int i = 0; i < n-1; i++){
        ptr=head;
```

```
            for(int j=0;j<n-i-1;j++){
                if(ptr->data > ptr->next->data){
                    int temp= ptr->data;
                    ptr->data=ptr->next->data;
                    ptr->next->data=temp;
                }
                ptr=ptr->next;
            }
        }
        printf("After sorting in ascending order,\n");
    }

    int main()
    {
        int length,at,choice;

        do{
            printf("\n-----------------------------MAIN MENU-----------------
------------------\n");
            printf("1. To Create a list \t\t\t2. To Add a Node at the beginning
\n3. To Add Node at the end \t\t4. To Add Node after the given node \n5. To Add
Node before the given node \t6. To delete a Node at the beginning \n7. To
delete a node from the end \t8. Delete a given Node\n9. Display the Linked
List\t\t10. Delete entire List\n11. Sort the list \t\t\t12. Exit\nEnter the
choice:");

            scanf("%d", &choice);

            switch (choice)
            {
            case 1:
                length= createll();
                break;
            case 2:
                insertstart();
                length++;
                break;
            case 3:
                insertend();
                length++;
                break;
            case 4:
                printf("add a new node before :\n");
                scanf("%d",&at);
                insertmidway(at,length,1);// 2 for before and 1 for after
                length++;
                break;
            case 5:
                printf("add a new node before :\n");
                scanf("%d",&at);
```

```
            insertmidway(at,length,2);// 2 for before and 1 for after
            length++;
            break;
        case 6:
            deleteStart();
            break;
        case 7:
            deleteEnd();
            break;
        case 8:
            printf("Delete node at index :\n");
            scanf("%d",&at);
            delete(at,length);
            length--;
            break;
        case 9:
            display();
            break;
        case 10:
            deletell();
            length=0;
            break;
        case 11:
            sort(length);
            break;
        case 12:
            exit(0);
        }
        display();

    }while (1);

    return 0;
}
```

Output :

```
                    ________
----------------------------MAIN MENU-------------------------------------
1. To Create a list                     2. To Add a Node at the beginning
3. To Add Node at the end               4. To Add Node after the given node
5. To Add Node before the given node    6. To delete a Node at the beginning
7. To delete a node from the end        8. Delete a given Node
9. Display the Linked List              10. Delete entire List
11. Sort the list                       12. Exit
Enter the choice:1
Enter the data : 10
Enter the data : 20
Enter the data : 30
Enter the data : 40
Enter the data : -1
Linked List created
Elements in list are : 10 20 30 40

----------------------------MAIN MENU-------------------------------------
1. To Create a list                     2. To Add a Node at the beginning
3. To Add Node at the end               4. To Add Node after the given node
5. To Add Node before the given node    6. To delete a Node at the beginning
7. To delete a node from the end        8. Delete a given Node
9. Display the Linked List              10. Delete entire List
11. Sort the list                       12. Exit
Enter the choice:4
add a new node before :
2
Enter the data : 88
Elements in list are : 10 20 88 30 40

----------------------------MAIN MENU-------------------------------------
1. To Create a list                     2. To Add a Node at the beginning
```

```
3. To Add Node at the end              4. To Add Node after the given node
5. To Add Node before the given node   6. To delete a Node at the beginning
7. To delete a node from the end       8. Delete a given Node
9. Display the Linked List             10. Delete entire List
11. Sort the list                      12. Exit
Enter the choice:6
Elements in list are : 20 88 30 40

----------------------------MAIN MENU-------------------------------------
1. To Create a list                    2. To Add a Node at the beginning
3. To Add Node at the end              4. To Add Node after the given node
5. To Add Node before the given node   6. To delete a Node at the beginning
7. To delete a node from the end       8. Delete a given Node
9. Display the Linked List             10. Delete entire List
11. Sort the list                      12. Exit
Enter the choice:11
After sorting in ascending order,
Elements in list are : 20 30 40 88

----------------------------MAIN MENU-------------------------------------
1. To Create a list                    2. To Add a Node at the beginning
3. To Add Node at the end              4. To Add Node after the given node
5. To Add Node before the given node   6. To delete a Node at the beginning
7. To delete a node from the end       8. Delete a given Node
9. Display the Linked List             10. Delete entire List
11. Sort the list                      12. Exit
Enter the choice:12
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Linked List>
```

2003145

PAGE No. 10m3
DATE / /

Experiment no. 8

Aim: To implement stack and Linear queue ADT using Linked List.

Theory: Instead of using array, we can also use Linked List to implement stack / queue.

Linked list allocates the memory dynamically however the time complexity is same for both the implementations.

In Linked List implementation of stack, the nodes are maintained non contigously in the memory. Each node contains a ~~node~~ pointer to immediate next node in the stack

Stack Algorithm:

1) Push operation:
   i) Create a newNode with given value.
   ii) Check if stack is empty (top == NULL)
   iii) If it is empty then set newNode → next = NULL
   iv) Else set newNode → next = top.
   v) Finally set top = newNode.

2) Pop operation:
   i) Check if the stack is empty (top == NULL)
   ii) If empty display "Underflow" and terminate the function
   iii) Else, define a node pointer temp and initialize it with top.
   iv) Set top = top → next
   v) finally delete temp (ie free(temp))

Algorithm:

- Insert operation:

The algorithm depends on if the property that if each node of bst left subtree of bst has a value less than root node and each left subtree has a value greater than root node.

1) If root == NULL
   return createNode(data);
2) If (data < node → data){
   node → left = insert(node → left, data);
3) else if (data > node → data)
   node → right = insert(node → right, data);

we keep going either to right subtree or left subtree depending upon the value until we reach a point, left and right subtree is null, we put new Node there.

- Deletion operation:

There are 3 case possible for deleting a node:

1) The node to be deleted is leaf node,
   in such case delete the node directly from the tree.
2) The node to be deleted has a single child node,
   in such case i) Replace that node with its child
   ii) Delete the child.
3) The node to be deleted has 2 child nodes,
   i) Get the inorder successor.
   ii) Replace the node with inorder successor.
   iii) Repeat this process of deletion for the inorder successor.

Stack using LL:

Program :

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

struct stack
{
    int data;
    struct stack *next;
};

struct stack *top=NULL,*ptr;

struct stack *createnode(){
    struct stack *newnode;
    newnode = (struct stack*)malloc(sizeof(struct stack));
    printf("Enter the number to be pushed onto the stack : ");
    scanf("%d",&newnode->data);
    newnode->next=NULL;

    return newnode;
}

void push(){
    struct stack *newnode=createnode();

    newnode->next=top;
    top=newnode;
}

void pop(){
    if(top==NULL) printf("Underflow\n");
    else{
        printf("Poped element : %d\n",top->data);
        ptr=top;
        top=top->next;
        free(ptr);
    }
}

void display(){
    if(top==NULL) printf("No elements in the stack\n");
    else{
        ptr=top;
        printf("Elements in stack : ");
        while (ptr!=NULL)
        {
```

```
                printf("%d ",ptr->data);
                ptr=ptr->next;
            }
            printf("\n");
        }
    }

    int main()
    {
        do{
            int choice;
            printf("\n1. PUSH   2. POP   3. DISPLAY   4. EXIT\nEnter your
choice : ");
            scanf("%d",&choice);
            switch (choice)
            {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;

            default:
                printf("Invalid choice !\n");
                break;
            }
        }while(1);

        return 0;
    }
```

Output :

```
1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 1
Enter the number to be pushed onto the stack : 10

1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 1
Enter the number to be pushed onto the stack : 20

1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 3
Elements in stack : 20 10

1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 2
Poped element : 20

1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 2
Poped element : 10

1. PUSH   2. POP   3. DISPLAY   4. EXIT
Enter your choice : 2
Underflow
```

Queue using linked list :

Program :

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

struct queue
{
    int data;
    struct queue *next;
};

struct queue *front=NULL,*ptr,*temp;

struct queue *createnode(){
    struct queue *newnode;
    newnode = (struct queue*)malloc(sizeof(struct queue));
    printf("Enter the number to enqueued : ");
    scanf("%d",&newnode->data);
    newnode->next=NULL;

    return newnode;
}

void enqueue(){
    struct queue *newnode=createnode();

    if(front==NULL) front=ptr=newnode;
    else{
        ptr->next=newnode;
        ptr=newnode;
    }
}

void dequeue(){

    if(front==NULL) printf("Underflow\n");
    else{
        printf("Dequeued element : %d\n",front->data);
        front=front->next;
    }
}

void display(){
    struct queue ;
    if(front==NULL) printf("No elements in the queue\n");
    else{
        temp=front;
```

```
            printf("Elements in queue : ");
            while (temp!=NULL)
            {
                printf("%d ",temp->data);
                temp=temp->next;
            }
            printf("\n");
        }
    }

    int main()
    {
        do{
            int choice;
            printf("\n1. ENQUEUE   2. DEQUEUE   3. DISPLAY   4. EXIT\nEnter
your choice : ");
            scanf("%d",&choice);
            switch (choice)
            {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;

            default:
                printf("Invalid choice !\n");
                break;
            }
        }while(1);

        return 0;
    }
```

Output :

2003145

PAGE No.
DATE

Experiment no. 9:

Aim: Implement Binary Search Tree ADT using Linked List.

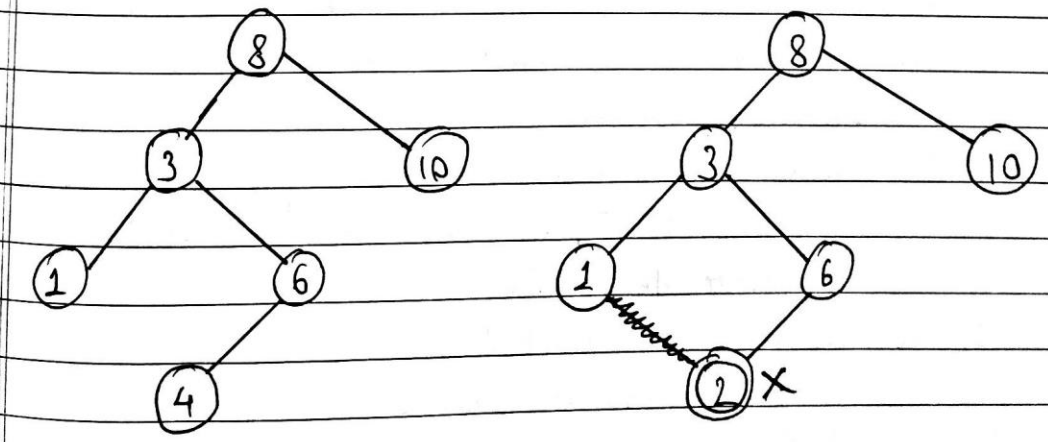
Theory:

Binary search tree is a data structure that quickly allows us to maintain a sorted list of elements.

- It is called binary tree because each node can have atmost 2 childerens.
- It is called a search tree because it can be used to find presence of a number in O (log(n)) time.

The properties that seperate bst from a regular binary tree is:

1) All ~~the~~ nodes of the left subtree are less than root node.
2) All nodes of the right subtree are greater than root node.
3) Both subtrees of each node are also BST ie they have above 2 properties.



BST ✗ Not a BST

Algorithm:

- Insert operation:

The algorithm depends on if the property that if each node of bst left subtree of bst has a value less than root node and each left subtree has a value greater than root node.

1) If root == NULL
   return createNode(data);
2) If (data < node → data){
   node → left = insert(node → left, data);
3) else if (data > node → data)
   node → right = insert(node → right, data);

we keep going either to right subtree or left subtree depending upon the value until we reach a point, left and right subtree is null, we put new Node there.

- Deletion operation:

There are 3 case possible for deleting a node:

1) The node to be deleted is leaf node,
   in such case delete the node directly from the tree.
2) The node to be deleted has a single child node,
   in such case i) Replace that node with its child
   ii) Delete the child.
3) The node to be deleted has 2 child nodes,
   i) Get the inorder successor.
   ii) Replace the node with inorder successor.
   iii) Repeat this process of deletion for the inorder successor.

Program :

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

struct node{
    int data ;
    struct node *right,*left;
};

struct node *createNode(int data){
    struct node* tree;
    tree=(struct node*)malloc(sizeof(struct node));
    tree->left=tree->right=NULL;
    tree->data=data;
    return tree;
}

struct node *createBST(int data){
        return createNode(data);
}

void insert(struct node *root,int data){
    if(data < root->data){
        if(root->left==NULL){
            root->left=createNode(data);
            return NULL;
        }
        else{
            insert(root->left,data);
        }
    }
    else if(data > root->data){
        if(root->right==NULL){
            root->right=createNode(data);
            return NULL;
        }
        else{
            insert(root->right,data);
        }
    }
}

struct node *preorder(struct node *root){
    if(root==NULL) return NULL;

    printf("%d ",root->data);
    preorder(root->left);
```

```
    preorder(root->right);
}

struct node *inorder(struct node *root){
    if(root==NULL) return NULL;

    inorder(root->left);
    printf("%d ",root->data);
    inorder(root->right);
}

struct node *postorder(struct node *root){
    if(root==NULL) return NULL;

    postorder(root->left);
    postorder(root->right);
    printf("%d ",root->data);
}

int smallest(struct node *root){
    if(root->left==NULL) {       //go to left extreme
        return root->data;
    }
    smallest(root->left);
}

int largest(struct node *root){
    if(root->right==NULL) {       //go to right extreme
        return root->data;
    }
    largest(root->right);
}

void mirror(struct node *root){
    if(root==NULL) return NULL;
    mirror(root->left);
    mirror(root->right);

    struct node *temp = root->left;
    root->left=root->right;
    root->right=temp;
}

int countNodes(struct node*root){
    if(root==NULL) return 0;

    return(countNodes(root->left)+countNodes(root->right)+1);
}

int internalNodeCount(struct node*root){
```

```
    if(root==NULL) return 0;
    if((root->right==NULL && root->left==NULL)) return 0;

    return(internalNodeCount(root->left)+internalNodeCount(root->right)+1);
}

int LeafNodeCount(struct node*root){
    if(root==NULL) return 0;
    if(root->right==NULL && root->left==NULL) return 1;

    return(LeafNodeCount(root->right) + LeafNodeCount(root->left));
}

int height(struct node *root){
    if(root==NULL) return 0;

    int leftHeight = height(root->left);
    int rightHeight = height(root->right);
    return(__max(leftHeight,rightHeight)+1);
}

struct node* deleteNode(struct node*root,int data){
    if(data<root->data)
        root->left= deleteNode(root->left,data);
    else if(data>root->data)
        root->right= deleteNode(root->right,data);

    else{
        if(root->right==NULL && root->left==NULL){
            free(root);
            return NULL;
        }
        else if(root->right==NULL || root->left==NULL){
            struct node* temp;

            if(root->right!=NULL) temp =root->right;
            else temp =root->left;

            free(root);
            return temp;
        }
        else{
            int inorderSucc= smallest(root->right);
            root->data=inorderSucc;
            root->right= deleteNode(root->right,inorderSucc);
        }
    }
}

struct node* deleteTree(struct node*root){
```

```
        if(root==NULL) return NULL;
        if(root->right==NULL && root->left==NULL){
            printf("%d ",root->data);
            free(root);
            return NULL;
        }
        root->left =deleteTree(root->left);
        root->right= deleteTree(root->right);
        deleteTree(root);
    }

    int main()
    {
        struct node *root=NULL;
        int option,val;

        do{
            printf("\n\n----------------------------- MAIN MENU ---------------
-------------------------");
            printf("\n 1. Insert Element");
            printf("\t\t\t 2. Preorder Traversal");
            printf("\n 3. Inorder Traversal");
            printf("\t\t\t 4. Postorder Traversal");
            printf("\n 5. Find the smallest element");
            printf("\t\t 6. Find the largest element");
            printf("\n 7. Delete an element");
            printf("\t\t\t 8. Count total number of nodes");
            printf("\n 9. Count total number of leaf nodes");
            printf("\t 10. Count total number of internal nodes");
            printf("\n 11. Determine the height of the tree");
            printf("\t 12. Find the mirror image of the tree");
            printf("\n 13. Delete the tree");
            printf("\t\t\t 14. Exit\n");

            printf("\n Enter your option : ");
            scanf("%d", &option);
            switch (option)
            {
                case 1:
                    printf("\n Enter the value of the new node : ");
                    scanf("%d", &val);
                    if(root==NULL) {
                    root=createBST(val);
                    }
                    else insert(root,val);
                    break;
                case 2:
                    printf("\n The elements of the tree in preorder are : ");
                    preorder(root);
                    break;
```

```
                case 3:
                    printf("\n The elements of the tree in inorder are : ");
                    inorder(root);
                    break;
                case 4:
                    printf("\n The elements of the tree in postorder are : ");
                    postorder(root);
                    break;
                case 5:
                    printf("\n Smallest element is :%d",smallest(root));
                    break;
                case 6:
                    printf("\n Largest element is : %d",largest(root));
                    break;
                case 7:
                    printf("\n Enter the element to be deleted : ");
                    scanf("%d", &val);
                    deleteNode(root, val);
                    break;
                case 8:
                    printf("\n Total no. of nodes = %d", countNodes(root));
                    break;
                case 9:
                    printf("\n Total no. of leaf nodes =
%d",LeafNodeCount(root));
                    break;
                case 10:
                    printf("\n Total no. of internal nodes =
%d",internalNodeCount(root));
                    break;
                case 11:
                    printf("\n The height of the tree = %d", height(root)-1);
                    break;
                case 12:
                    mirror(root);
                    break;
                case 13:
                    root=deleteTree(root);
                    break;
            }
        } while(option != 14);

        return 0;
    }
```

Output :

```
---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                       2. Preorder Traversal
 3. Inorder Traversal                    4. Postorder Traversal
 5. Find the smallest element            6. Find the largest element
 7. Delete an element                    8. Count total number of nodes
 9. Count total number of leaf nodes     10. Count total number of internal nodes
 11. Determine the height of the tree    12. Find the mirror image of the tree
 13. Delete the tree                     14. Exit

 Enter your option : 1

 Enter the value of the new node : 20


---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                       2. Preorder Traversal
 3. Inorder Traversal                    4. Postorder Traversal
 5. Find the smallest element            6. Find the largest element
 7. Delete an element                    8. Count total number of nodes
 9. Count total number of leaf nodes     10. Count total number of internal nodes
 11. Determine the height of the tree    12. Find the mirror image of the tree
 13. Delete the tree                     14. Exit

 Enter your option : 3

 The elements of the tree in inorder are : 5 10 12 15 20

---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                       2. Preorder Traversal
 3. Inorder Traversal                    4. Postorder Traversal
```

```
 3. Inorder Traversal                    4. Postorder Traversal
 5. Find the smallest element            6. Find the largest element
 7. Delete an element                    8. Count total number of nodes
 9. Count total number of leaf nodes     10. Count total number of internal nodes
 11. Determine the height of the tree    12. Find the mirror image of the tree
 13. Delete the tree                     14. Exit

 Enter your option : 6

 Largest element is : 20

---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                       2. Preorder Traversal
 3. Inorder Traversal                    4. Postorder Traversal
 5. Find the smallest element            6. Find the largest element
 7. Delete an element                    8. Count total number of nodes
 9. Count total number of leaf nodes     10. Count total number of internal nodes
 11. Determine the height of the tree    12. Find the mirror image of the tree
 13. Delete the tree                     14. Exit

 Enter your option : 9

 Total no. of leaf nodes = 3

---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                       2. Preorder Traversal
 3. Inorder Traversal                    4. Postorder Traversal
 5. Find the smallest element            6. Find the largest element
 7. Delete an element                    8. Count total number of nodes
 9. Count total number of leaf nodes     10. Count total number of internal nodes
 11. Determine the height of the tree    12. Find the mirror image of the tree
```

```
 13. Delete the tree                    14. Exit

 Enter your option : 10

 Total no. of internal nodes = 2

---------------------------- MAIN MENU ----------------------------------------
 1. Insert Element                      2. Preorder Traversal
 3. Inorder Traversal                   4. Postorder Traversal
 5. Find the smallest element           6. Find the largest element
 7. Delete an element                   8. Count total number of nodes
 9. Count total number of leaf nodes    10. Count total number of internal nodes
 11. Determine the height of the tree   12. Find the mirror image of the tree
 13. Delete the tree                    14. Exit

 Enter your option : 11

 The height of the tree = 2

---------------------------- MAIN MENU ----------------------------------------
 1. Insert Element                      2. Preorder Traversal
 3. Inorder Traversal                   4. Postorder Traversal
 5. Find the smallest element           6. Find the largest element
 7. Delete an element                   8. Count total number of nodes
 9. Count total number of leaf nodes    10. Count total number of internal nodes
 11. Determine the height of the tree   12. Find the mirror image of the tree
 13. Delete the tree                    14. Exit

 Enter your option : 12
```

```
---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                        2. Preorder Traversal
 3. Inorder Traversal                     4. Postorder Traversal
 5. Find the smallest element             6. Find the largest element
 7. Delete an element                     8. Count total number of nodes
 9. Count total number of leaf nodes      10. Count total number of internal nodes
 11. Determine the height of the tree     12. Find the mirror image of the tree
 13. Delete the tree                      14. Exit

 Enter your option : 3

 The elements of the tree in inorder are : 20 15 12 10 5

---------------------------- MAIN MENU -----------------------------------------
 1. Insert Element                        2. Preorder Traversal
 3. Inorder Traversal                     4. Postorder Traversal
 5. Find the smallest element             6. Find the largest element
 7. Delete an element                     8. Count total number of nodes
 9. Count total number of leaf nodes      10. Count total number of internal nodes
 11. Determine the height of the tree     12. Find the mirror image of the tree
 13. Delete the tree                      14. Exit

 Enter your option : 7

 Enter the element to be deleted : 12
```

2903145

PAGE No.
DATE 10/11/25

Experiment. 10

Aim: Implement Graph traversal techniques
a. Depth first search
b. Breadth first search.

Theory:

Depth first seach or depth first traversal is a recursive algorithm for searching all the vertices of a graph or a tree data structure.

oDepth first search Algorithm:

We need to put all the vertices of graph into one of two: a) Visited.
b) Not visited.

The purpose of this algorithm is to mark each vertex so as to avoid cycles.

1. Start by putting any one of the graph vertice on the top of the stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list, to the top of stack.
4. Keep repeating step 2 & 3 until the stack is empty.

- BFS algorithm:

We need to put each vertex of the graph into one of 2 categories.

i) Visited

ii) Not visited.

The purpose of the algorithm is to mark each vertex as visited while avoiding ~~that~~ clashes.

1) Start by putting any one of the graph's vertice at the back of queue

2) Take the front of the queue and add it to the ~~front~~ item of visited list.

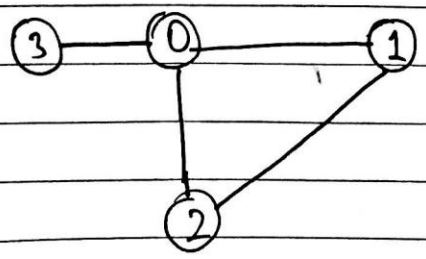
3) Create a list of that vertex's adjacent nodes. Add the ones which are not in the visited list to the back of queue.

4) Keep repeating step 2 & 3 until the queue is empty.

for example : 0 1 1 1 is the adjacency matrix
1 0 1 0
1 1 0 0
1 0 0 0

BFS:

ie



(starting with 0)

Queue :

Queue : ~~0~~ ~~1~~ ~~2~~ ~~3~~

Output : 0 1 2 3

BFS :

Program :

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

struct Queue{
    int size;
    int front;
    int rear;
    int* arr;
};
int isFull(struct Queue* q){
    if(q->rear==q->size-1){
        return 1;
    }
    return 0;
}
int isEmpty(struct Queue* q){
    if(q->front>=q->rear){
        return 1;
    }
    return 0;
}
struct Queue* enqueue(struct Queue* q,int data){
    if(q->rear!=q->size-1){
        q->rear++;
        q->arr[q->rear]=data;
        return q;
    }
}
int dequeue(struct Queue *q){
    int b = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->front++;
        b = q->arr[q->front];
    }
    return b;
}


int main(){
    struct Queue q ;
    q.size=20;
    q.front=-1;
```

```
    q.rear=-1;
    q.arr=(int*)malloc(q.size*sizeof(int));

    int a[20][20],visited[20],n,start;

        printf("Enter the number of vertices in graph :");
        scanf("%d",&n);
        printf("Enter the adjacency matrix :\n");
        for(int i=0;i<n;i++){
            visited[i]=0;
            for(int j=0;j<n;j++){
                scanf("%d",&a[i][j]);
            }
        }
        printf("Enter the starting vertex :");
        scanf("%d",&start);

    printf("BFS traversal : %d ",start);
    visited[start]=1;
    enqueue(&q,start);

    while(!isEmpty(&q)){
        int node = dequeue(&q);
        for(int j=0;j<n;j++){
            if(a[node][j]==1 && visited[j]==0){
                printf("%d ",j);
                visited[j] = 1;
                enqueue(&q,j);
            }
        }
    }

}
```

Output :

```
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Graph> & .\"BFS.exe"
Enter the number of vertices in graph :4
Enter the adjacency matrix :
0 1 1 1
1 0 1 0
1 1 0 0
1 0 0 0
Enter the starting vertex :0
BFS traversal : 0 1 2 3 PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Graph>
```

DFS :

Program :

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

int visited[20];
int a[20][20],n;

void DFS(int i){
    int j;
    printf("%d ",i);
    visited[i] = 1;
    for(int j=0;j<n;j++){
        if(a[i][j]==1 && visited[j]==0){
            DFS(j);
        }
    }
}


int main(){

    int start;

        printf("Enter the number of vertices in graph :");
        scanf("%d",&n);
        printf("Enter the adjacency matrix :\n");
        for(int i=0;i<n;i++){
            visited[i]=0;
            for(int j=0;j<n;j++){
                scanf("%d",&a[i][j]);
            }
        }
        printf("Enter the starting vertex :");
        scanf("%d",&start);

    printf("DFS traversal :");
    DFS(start);
}
```

Output :

```
PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Graph> & .\"DFS.exe"
Enter the number of vertices in graph :4
Enter the adjacency matrix :
0 1 1 1
1 0 1 0
1 1 0 0
1 0 0 0
Enter the starting vertex :0
DFS traversal :0 1 2 3 PS C:\Users\IsmailRatlamwala\Documents\College prog\DS Labs\Graph>
```

V.2.

Theory Assingment - 1.

Idris Ratlamwala
2003145

1.

Linear data structures: A linear ds have it's elements arranged in sequential manner and each element is connected to it's previous and next element. Such ds are easy to implement as comp. mem is also sequential.
eg. Linked list, Array, Queue, Stack.

Non-linear data structures: A non linear ds has no sequence of connecting all its elements, ~~in~~ and each element can have multiple paths to connect to ~~each~~ other elements. Such data structure are not easy to implement but are more efficient in utilizing comp. memory.

eg. Tree, BST, Graphs, etc.

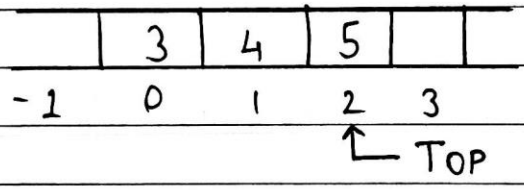
Linear data Structures	Non linear data structures.
1. All data elements are present at a single level.	1. Data elements are present at multiple levels.
2. Linear ds are easier to implement.	2. Non linear ds are difficult to understand and implement.
3. Can be traversed completely in single run.	3. Not easy to traverse and needs multiple runs to be traversed completely.
4. Linear ds are not very memory friendly and are not utilizing it efficiently.	4. Non linear ds utilizes memory very efficiently.
5. Time complexity often increases with increase in size.	5. Time complexity often remains constant ~~for~~ with increase in size.
6. Example: Array, Queue, LL, Stack.	6. Examples: Tree, graphs, BST.

2.

A stack is an Abstract Data Type (ADT), commonly used in most of programming languages. It is named STACK as it behaves like real world stack.
For example, a deck of cards, pile of plates, etc.
Stack uses LIFO data structure ie [Last In First Out].
Here the elements which is placed (inserted/added) last is accessed first. In stack terminology, insertion operation is termed as PUSH and deletion operation is termed as POP.

Stack representation:

A stacks can be implemented by means of array, structure, pointer and Linked List



Stack using array

Stack using L.L.

Basic operations:

i) PUSH() - ~~Pushing~~ Storing an element on the top of stack.
ii) POP() - Removing (accessing) an element from the top of the stacks.

3. Implementation of priority queue using array in C:

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

struct Queue {
    int capacity, rear, front;
    int *arr;
};
typedef struct Queue *pointerToNode;
typedef pointerToNode queue;

queue createQueue(int maxSize) {
    queue q = (struct queue*) malloc (sizeof (struct Queue));
    q -> capacity = maxSize;
    q -> rear = q -> front = -1;
    q -> arr = (int*) malloc (maxSize * sizeof (int));
    return q;
}
int isempty (queue q) { return q -> rear == -1 && q -> front == -1; }

int isfull (queue q) { return q -> rear == q -> capacity - 1; }

queue enqueue (queue q, int data) {
    if (isfull (q)) printf ("overflow");
    else {
        int pos = q -> rear;
        q -> rear = q -> rear + 1;
        while (pos >= 0 && q -> arr[pos] > data)
        {
            q -> arr[pos + 1] = q -> arr[pos];
    }   pos --
```

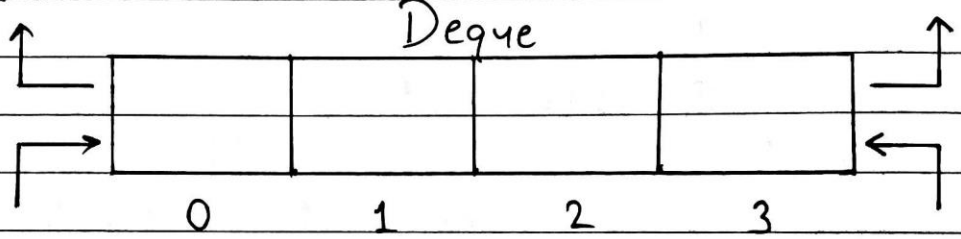
```
        q->arr[pos+1] = data;
        if (q->front == -1) q->front = 0;
    }
    return q;
}
void dequeue(queue q){
    if(isempty(q)) printf("Underflow");
    else if (q->front == q->rear){
        printf("Dequeed element:%d", q->arr[q->front]);
        q->front = q->rear = -1;
    }
    else {
        printf("Dequeued element: %d", q->arr[q->front]);
        q->front++;
    }
}
void display(queue q) {
    if(isempty(q)) printf("Queue is empty");
    else {
        printf("Elements in queue: ");
        for(int i = q->front; i<=q->rear; i++)
            printf("%d", q->arr[i]);
        printf("\n");
    }
void search(queue q, int data) {
    int i;
    for(i = q->front; i<=q->rear; i++) {
        if (q->arr[i] == data) {
            printf("Index of given element is %d", i);
            break;
        }
```

```
        q->arr[pos+1] = data;
        if (q->front == -1) q->front = 0;
    }
    return q;
}
void dequeue(queue q){
    if(isempty(q)) printf("Underflow");
    else if (q->front == q->rear){
        printf("Dequeed element:%d", q->arr[q->front]);
        q->front = q->rear = -1;
    }
    else {
        printf("Dequeued element: %d", q->arr[q->front]);
        q->front++;
    }
}
void display(queue q) {
    if(isempty(q)) printf("Queue is empty");
    else {
        printf("Elements in queue: ");
        for(int i = q->front; i<=q->rear; i++)
            printf("%d", q->arr[i]);
        printf("\n");
    }
void search(queue q, int data) {
    int i;
    for(i = q->front; i<=q->rear; i++) {
        if (q->arr[i] == data) {
            printf("Index of given element is %d", i);
            break;
        }
```

```
    if(i==q->rear+1) printf("Not found");
}
int main(){
    int maxSize=10, choice, data;
    queue q;
    q=createQueue(maxSize);
    do{
        printf("\n1.Enqueue 2.Dequeue 3.Display 4.Search
        5.Exit");
        scanf("%d",&choice);
        switch(choice){
            case 1:
                printf("Enter the element to be inserted:");
                scanf("%d",&data);
                enqueue(q,data);
                break;
            case 2:
                dequeue(q);
                break;
            case 3:
                display(q);
                break;
            case 4:
                printf("Enter the element to be searched:");
                scanf("%d",&data);
            case 5:
                break;
        }
    }while(choice!=5);
    return 0;
}
```

4. Double Ended Queue:

A double ended queue also known as deque, is an ordered collection of items similar to queue. It has 2 ends, a front end and, a rear end and the items remained positioned between them. What make deque different from normal queue is unrestrictive nature of adding and removing elements from queue. New items can be added from either ends and likewise for removing items. In a sense, this hybrid linear structure provides all the capabilities of stack and queue in a single data structure.



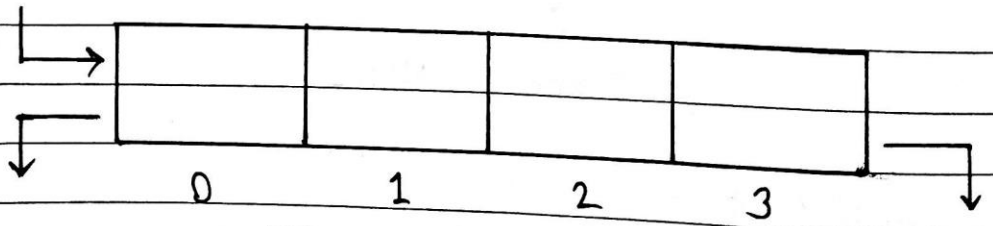
There are 2 types of queues:

i) Input restricted queue

ii) Output restricted queue

i) Input restricted queue:

The input restricted queue means that some of the restrictions are applied to the insertion. In input restricted queue, the insertion is applied on one end and deletion can be applied from both the ends.

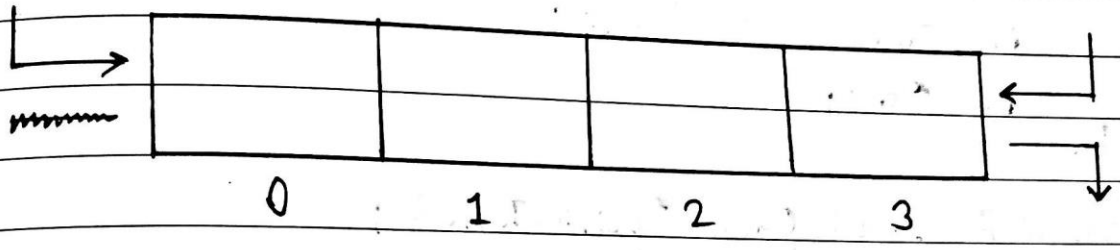


Input-Restricted Queue.

DATE / /

ii) Output restricted queue:

The output restricted queue means that some restrictions are applied to the deletion of elements operations. In output restricted queue the deletion can only be applied from one end, where as the insertion is possible from both the ends



Operations on deque:

The following are the operations performed on deque:

i) Insert at front

ii) Insert at rear

iii) Delete from front

iv) Delete from rear.

5. Polynomial representation and addition:

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

struct node {
    int coeff, pow;
    struct node *next;
};
typedef struct node *PtrToNode;
typedef PtrToNode term;

term *head1 = NULL, *head2 = NULL, *head3 = NULL, *ptr1, *ptr2, *ptr3,
     *newnode;

term createTerm(){
    newNode = (struct node*) malloc(sizeof(struct node));
    printf("Enter the exponent: ");
    scanf("%d", &newnode->pow);
    printf("Enter the coefficient: ");
    scanf("%d", &newnode->coeff);
    return newNode;
}

void getPolynomial(){
    int n;
    printf("Enter the number of terms in 1st polynomial");
    scanf("%d", &n); head1 = ptr1 = createTerm();
    for(int i=0; i<n; i++){
    {   newNode = createTerm();
        ptr1->next = newNode;
        ptr1 = newNode;
    }
```

```
    ptr1->next = NULL;
    printf("Enter the number of terms in 2nd polynomial");
    scanf("%d", &n);
    head2 = ptr2 = createTerm();
    for(int i=1; i<n; i++){
        newNode = createTerm();
        ptr2->next = newNode;
        ptr2 = newNode;
    }

    ptr2->next = NULL;
}
void display(){
    ptr1 = head3;
    while(ptr1 != NULL){
        if(ptr1 != head3)
        if(ptr1->coeff >= 0) printf(" + "); else printf(" - ");
        printf("%dx^", ptr1->ceff);
        printf("%d", ptr1->pow);
        ptr1 = ptr1->next;
    }
}
void addPolynomial(){
    ptr1 = head1; ptr2 = head2;
    while(ptr1 != NULL && ptr2 != NULL){
        if(ptr1->pow == ptr2->pow){
            newNode = createNode();
            newNode->coeff = ptr1->coeff + ptr2->coeff;
            newNode->pow = ptr1->pow;
            ptr1 = ptr1->next;
            ptr2 = ptr2->next;
        }
```

```
else if (ptr1 → pow > ptr2 → pow) {
        newNode = createNode();
        newNode → coeff = ptr1 → coeff;
        newNode → pow = ptr1 → pow;
        ptr1 = ptr1 → next;
    }
    else {
        newNode = createNode();
        newNode → coeff = ptr2 → coeff;
        newNode → pow = ptr2 → pow;
        ptr2 = ptr2 → next;
    }
}
if (ptr1 != NULL) {
    while (ptr1 != NULL) {
        newNode = createNode();
        newNode → coeff = ptr1 → coeff;
        newNode → pow = ptr1 → pow;
        ptr1 = ptr1 → next;
    }
if (ptr2 != NULL)
    while (ptr2 != NULL) {
        newNode = createNode();
        newNode → coeff = ptr2 → coeff;
        newNode → pow = ptr2 → pow;
        ptr2 = ptr2 → next;
    }
ptr3 → next = NULL
}
void main() {
    getPolynomial();
    addPolynomial();
    displayPolyno();
}
```

Idris Ratlamwala

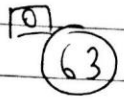
2003145 - C31

DS. Assingment 2.

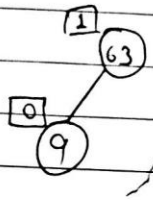
PAGE No.
DATE

1. Data : 63, 9, 19, 18, 108, 99, 81, 45, 12, 106

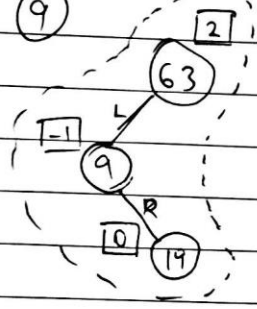
•) 63 :



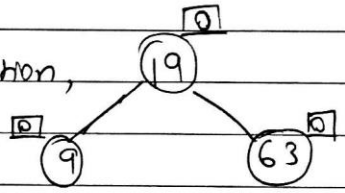
•) 9 :



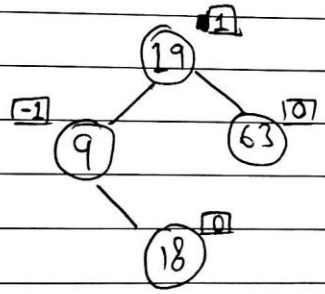
•) 19 :



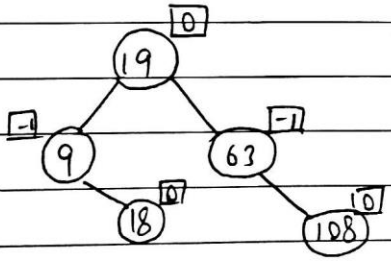
Applying LR rotation,



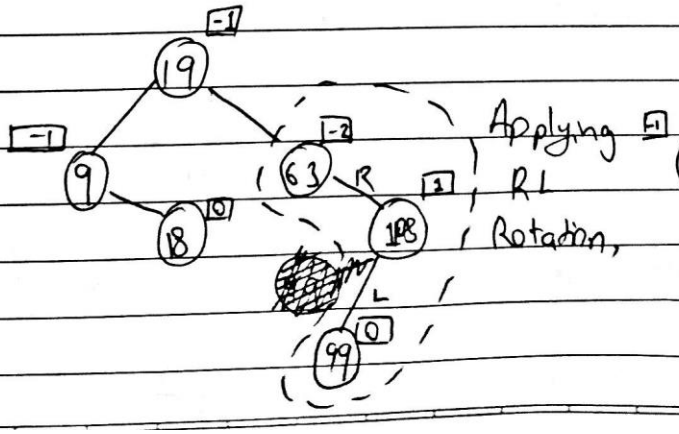
•) 18 :

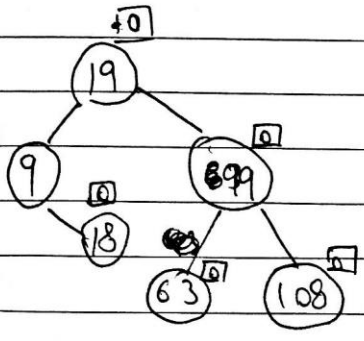


•) 108 :

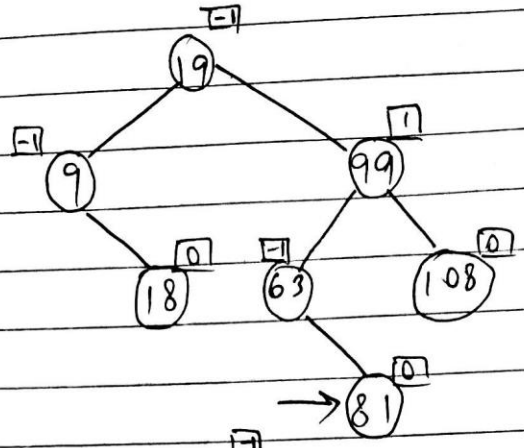


•) 99

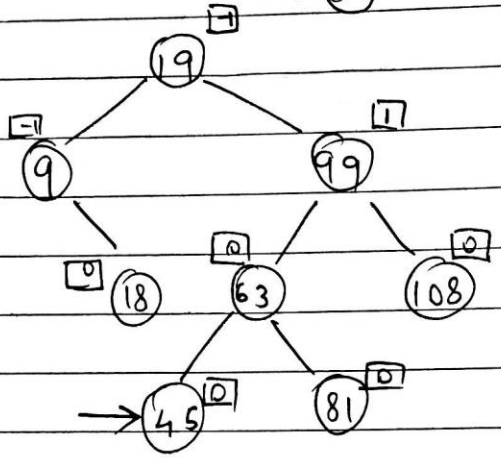




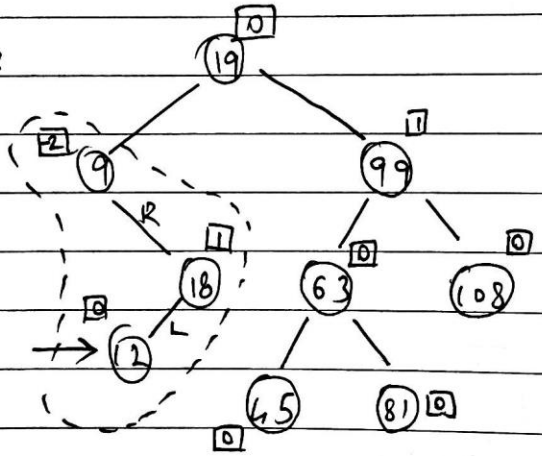
•) 81 :



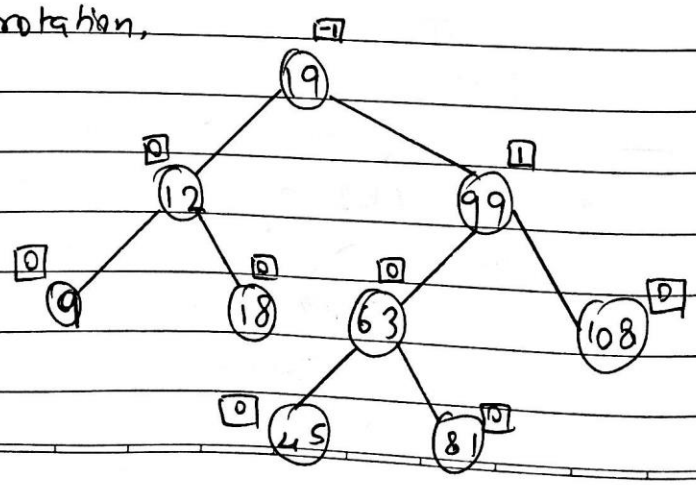
•) 45 :



•) 12 :



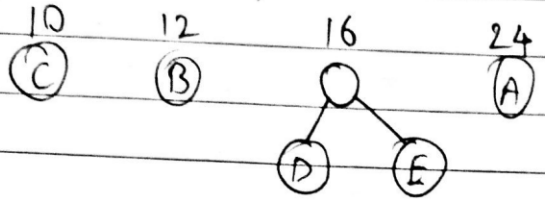
Applying RL rotation,



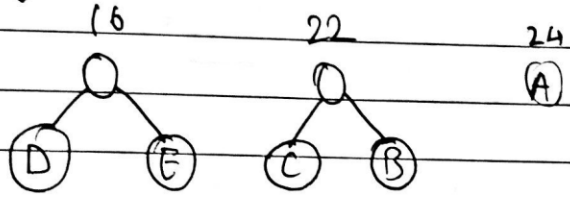
2.

1) Each node is represented as tree:

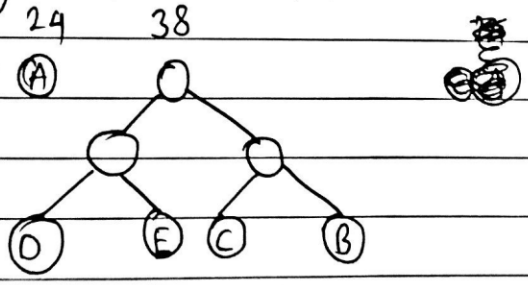
2) Merge D and E,



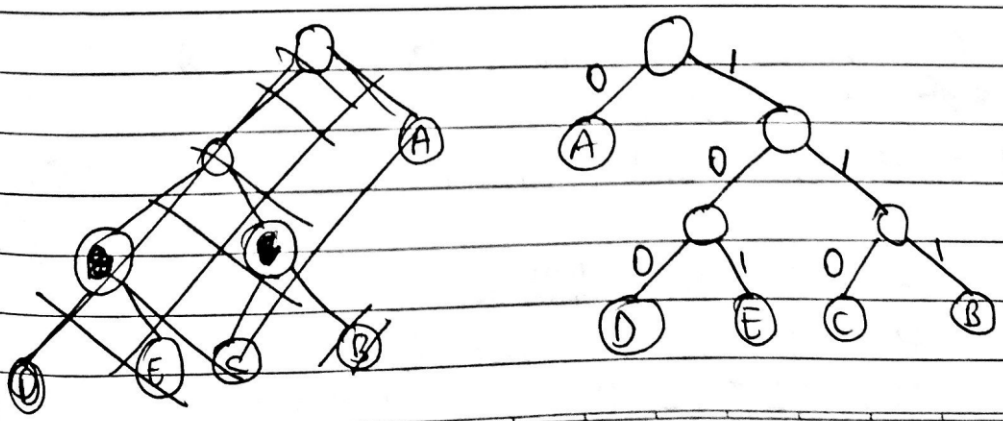
3) Merge C and B,



4) Merge (D, E) and (C, B),



5) Merge (D, E, C, B) and A,

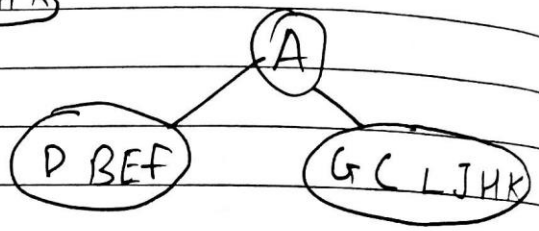


Symbols	Frequency	Huffman code
A	24	0
B	12	111
C	10	110
D	8	100
E	8	101

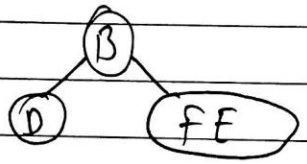
4.

Postorder : D E f B G L J K H C A Root

Inorder : D B f E A G C L J H K

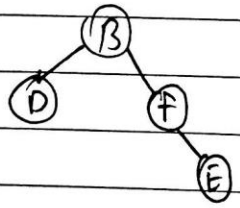


• For left subtree postorder is D E f B, ∴ B is the root node



∵ B divides D and f E in inorder.

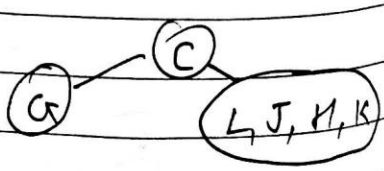
and the postorder for right subtree (f E) is E f ∴ f is root



• for Right subtree, postorder is G L J K H C.

∴ C is the root node.

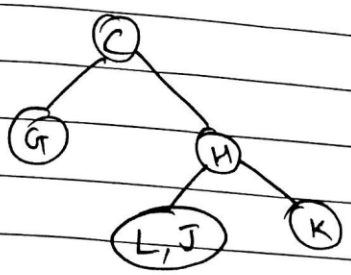
And C divides G and L, J, H, K in inorder



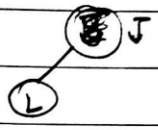
For Right subtree, Post order is

L J K H, ∴ H is the root node

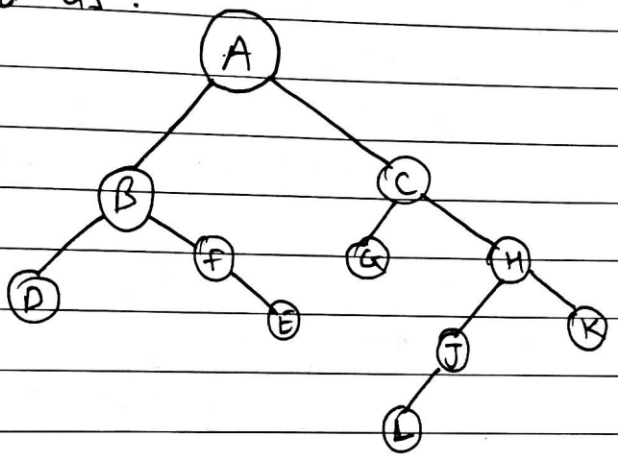
And it divides L, J & K in inorder.



post order of (L, J) is L, J; hence J is root node and L comes before J in pinorder.



∴ The whole tree can be constructed as:



3

Hashing:

- Hashing is the process of mapping large number of data items to smaller table with the help of hashing function.
- This technique converts a range of key values into a range of indexes of an array.
- Hashing allows to update and retrive data entries in a constant time O(1), which could never have been possible with linear or binary search.
- Hashing is used in the encryption and decryption of digital signatures.

63, 82, 94, 77, 53, 87, 23, 55, 10, 44

I] Linear probing

here, m = 10, and let $h'(k) = k \% m$

$h(k) = [h'(k) + i] \% m$

Initial hash table ⇒

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

1] k = 63

h(k) = 63 % 10 = 3 -1 -1 -1 63 -1 -1 -1 -1 -1 -1

• ... ($h(k) = h'(k) \because i = 0$)

2) k = 82

h(k) = 82 % 10 = 2 -1 -1 82 63 -1 -1 -1 -1 -1 -1

3) k = 94

h(k) = 94 % 10 = 4 -1 -1 82 63 94 -1 -1 -1 -1 -1

4) k = 77

h(k) = 77 % 10 = 7 -1 -1 82 63 94 -1 -1 77 -1 -1

5) k = 53

h(k) = (53 % 10) + 0) % 10 = 3

∵ index 3 is already occupied, try again for next location

∴ h(k) = ((53 % 10) + 1) % 10 = 4

h(k) = (3 + 2) % 10 = 5 -1 -1 82 63 94 53 -1 77 -1 -1

[2 collisions]

6) k = 87

h(k) = ((87 % 10) + 0) % 10 = 7

index 7 is already occupied

h(k) = (7 + 1) % 10 = 8 -1 -1 82 63 94 53 -1 77 87 -1

[1 collision]

7) K = 23

h(K) = ((23 % 10) + 0) % 10 = 3

3 is already occupied

h(K) = (3 + 1) % 10 = 4

h(K) = (3 + 2) % 10 = 5

h(K) = (3 + 3) % 10 = 6 -1 -1 82 63 94 53 23 77 87 -1

[3 coll]

8) K = 55

h(K) = ((55 % 10) + 0) % 10 = 5

5 is already occupied.

h(K) = (5 + 1) % 10 = 6

h(K) = (5 + 4) % 10 = 9 -1 -1 82 63 94 53 23 77 87 55

[4 coll]

9) K = 10

h(K) = ((10 % 10) + 0) % 10 = 0 10 -1 82 63 94 53 23 77 87 55

10) K = 44

h(K) = ((44 % 10) + 0) % 10 = 4

4 is already occupied.

h(K) = (4 + 1) % 10 = 5

h(K) = (4 + 7) % 10 = 1

[7 coll]

10	44	82	63	94	53	23	77	87	55

Total collisions : 17

II] Quadratic probing :

$h(K,i) = [h'(K) + c_1 i + c_2 i^2] \% m$