EXPERIMENT NUMBER: 1.1 **DATE:**

1.1) Recursive and non-recursive functions to perform Linear search for a Key value in a given list.

Non-recursive Program:

```
#include<stdio.h>
int linearsearch(int n,int k){
    if(n==k)
        return 1;
    return 0;
}
int main(){
    int n,c=0,f=0;
    scanf("%d",&n);
    int i,arr[n],k;
    for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
    scanf("%d",&k);
    for(i=0;i<n;i++){
        if(linearsearch(arr[i],k)){
            printf("Element found at index %d",i);
            f=1;
            break;}
    }
    if(f==0)
        printf("Element not found");
}
```


Sample Input:

```
7
1 2 3 4 5 6 7
3
```

Sample Output:

```
Element found at index 2
```

Recursive Program:

```
#include<stdio.h>
int linearsearch(int *arr,int n,int i,int k){
    if(i<n){
        if(arr[i]==k)
            return i;
```

```
            else
                return linearsearch(arr,n,i+1,k);
        }
        return -1;}
    int main(){
        int n,c=0,f;
        scanf("%d",&n);
        int i,arr[n],k;
        for(i=0;i<n;i++)
            scanf("%d",&arr[i]);
        scanf("%d",&k);
        f=linearsearch(arr,n,0,k);
        if(f==-1)
            printf("Element not found");
        else
            printf("Element found at index %d",f);
    }
```

Sample Input:

7

1 2 3 4 5 6 7

8

Sample Output:

Element not found

EXPERIMENT NUMBER: 1.2 **DATE:**

1.2) Recursive and non recursive functions to perform Binary search for a Key value in a given list

Non-recursive Program:

```
#include<stdio.h>
int binarysearch(int *arr,int e,int l,int r){
        int m;
        while(l<=r){
                m=(l+r)/2;
                if(arr[m]==e)
                        return m;
                if(arr[m]<e)
                        l=m+1;
                if(arr[m]>e)
                        r=m-1;
        }
        return -1;}
int main(){
        int n,e,i,j,k,t;
        scanf("%d",&n);
        int l=0,r=n-1,arr[n];
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        scanf("%d",&e);
        k= binarysearch(arr,e,l,r);
        if(k!=-1)
                printf("Element found at index %d",k);
        else
                printf("Element not found");
}
```


Sample Input:

7

1 2 3 4 5 6 7

0

Sample Output:

Element not found

EXPERIMENT NUMBER: **DATE:**

Recursive Program:

```
#include<stdio.h>
int binarysearch(int *arr,int e,int l,int r){
        int m;
        if(l>r)
                return -1;
        else{
                m=(l+r)/2;
                if(arr[m]==e)
                        return m;
                if(arr[m]<e)
                        l=m+1;
                if(arr[m]>e)
                        r=m-1;
        }
        return binarysearch(arr,e,l,r);}
int main(){
        int n,e,i,j,k,t;
        scanf("%d",&n);
        int l=0,r=n-1,arr[n];
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        scanf("%d",&e);
        k= binarysearch(arr,e,l,r);
        if(k!=-1)
                printf("Element found at index %d",k);
        else
                printf("Element not found");
}
```


Sample Input:

```
7
1 2 3 4 5 6 7
7
```

Sample Output:

```
Element found at index 6
```

EXPERIMENT NUMBER: 2.1 **DATE:**

2.1) Bubble sort, to sort a given list of integers.

Program:

```
#include<stdio.h>
void bubblesort(int *arr,int n){
	int i,j,temp,flag;
	for(i=0;i<n-1;i++){
		flag=0;
		for(j=0;j<n-i-1;j++){
			if(arr[j]>arr[j+1]){
				temp=arr[j];
				arr[j]=arr[j+1];
				arr[j+1]=temp;
				flag=1;
			}
		}
		if(flag==0)
			break;
	}
}
int main(){
	int n;
	scanf("%d",&n);
	int i,arr[n];
	for(i=0;i<n;i++)
		scanf("%d",&arr[i]);
	bubblesort(arr,n);
	for(i=0;i<n;i++)
		printf("%d ",arr[i]);
}
```


Sample Input:

10

85 20 66 33 54 35 22 11 9 67

Sample Output:

9 11 20 22 33 35 54 66 67 85

2.2) Insertion sort, to sort a given list of integers.

Program:

```
#include<stdio.h>
int *insertsort(int *arr,int n){
        int i,temp,j;
        for(i=1;i<n;i++){
                temp=arr[i];
                j=i-1;
                while(j>=0&&arr[j]>temp){
                        arr[j+1]=arr[j];
                        j--;
                }
                arr[j+1]=temp;
        }
        return arr;}
int main(){
        int n;
        scanf("%d",&n);
        int arr[n],i,*array;
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        array=insertsort(arr,n);
        for(i=0;i<n;i++)
                printf("%d ",array[i]);
}
```


Sample Input:

10

9 8 7 6 5 4 3 2 1 10

Sample Output:

1 2 3 4 5 6 7 8 9 10

EXPERIMENT NUMBER: 2.3 **DATE:**

2.3) Selection sort, to sort a given list of integers.

Program:

```
#include<stdio.h>
int max(int *arr,int i){
	int j,maxx=0,k;
	for(j=i;j>=0;j--){
		if(arr[j]>maxx){
			maxx=arr[j];
			k=j;
		}
	}
	return k;
}
void selection_sort(int *arr,int n){
	int i,l,k;
	for(i=n-1;i>=0;i--){
		l=max(arr,i);
		k=arr[l];
		arr[l]=arr[i];
		arr[i]=k;
	}
}
int main(){
	int n,i,j,k,l;
	scanf("%d",&n);
	int arr[n];
	for(i=0;i<n;i++)
		scanf("%d",&arr[i]);
	selection_sort(arr,n);
	for(i=0;i<n;i++)
		printf("%d ",arr[i]);
}
```


Sample Input:

10

85 20 66 33 54 35 22 11 9 67

Sample Output:

9 11 20 22 33 35 54 66 67 85

EXPERIMENT NUMBER: 3 **DATE:**

3) Quick sort, to sort a given list of integers.

Program:

```
#include<stdio.h>
int fun(int *a,int lb,int ub){
        int pivot=a[lb],start=lb,end=ub,temp;
        while(start<end){
                while(a[start]<=pivot)
                        start++;
                while(a[end]>pivot)
                        end--;
                if(start<end){
                        temp=a[start];
                        a[start]=a[end];
                        a[end]=temp;}}
        temp=a[lb];
        a[lb]=a[end];
        a[end]=temp;
        return end;}
void quicksort(int *arr,int lb,int ub,int n){
        int loc,i;
        if(lb<ub){
                loc=fun(arr,lb,ub);
                quicksort(arr,lb,loc-1,n);
                quicksort(arr,loc+1,ub,n);}
}
int main(){
        int i,n,arr[10000];
        scanf("%d",&n);
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        quicksort(arr,0,n-1,n);
        for(i=0;i<n;i++)
                printf("%d ",arr[i]);}
```


Sample Input:

8

88 66 22 11 33 55 77 44

Sample Output: 11 22 33 44 55 66 77 88

EXPERIMENT NUMBER: 4.1 **DATE:**

4.1) Merge sort, to sort a given list of integers

Program:

```
#include<stdio.h>
void merge(int *arr,int l,int m,int h){
        int i,a[100],b[100],n1=m-l+1,n2=h-m,k=l,newarr[1000],j;
        for(i=0;i<n1;i++)
                a[i]=arr[k++];
        k=m+1;
        for(i=0;i<n2;i++)
                b[i]=arr[k++];
        i=j=k=0;
        while(i<n1&&j<n2)
        {
                if(a[i]<=b[j])
                        newarr[k]=a[i++];
                else
                        newarr[k]=b[j++];
                k++;}
        while(i<n1)
                newarr[k++]=a[i++];
        while(j<n2)
                newarr[k++]=b[j++];
   k=l;
   for(i=0;i<n1+n2;i++)
        arr[k++]=newarr[i];
}
void mergesort(int *arr,int l,int h){
        int m;
        if(l<h)
        {
                m=(l+h)/2;
                mergesort(arr,l,m);
                mergesort(arr,m+1,h);
                merge(arr,l,m,h);
        }
}
int main()
```

```
{
        int n;
        scanf("%d",&n);
        int i,arr[n];
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        mergesort(arr,n);
        for(i=0;i<n;i++)
                printf("%d ",arr[i]);
}
```

Sample Input:

8

8 1 7 2 6 3 5 4

Sample Output:

1 2 3 4 5 6 7 8

EXPERIMENT NUMBER: 4.2 **DATE:**

4.2) Radix sort, to sort a given list of integers.

Program:

```
#include<stdio.h>
void countsort(int *arr,int n,int pos){
        int count[10]={0},b[n],i;
        for(i=0;i<n;i++)
                ++count[(arr[i]/pos)%10];
        for(i=1;i<10;i++)
                count[i]+=count[i-1];
        for(i=n-1;i>=0;i--)
                b[--count[(arr[i]/pos)%10]]=arr[i];
        for(i=0;i<n;i++)
                arr[i]=b[i];}
void radixsort(int *arr,int n){
        int max=-100,pos;
        for(pos=0;pos<n;pos++){
                if(max<arr[pos])
                        max=arr[pos];}
        for(pos=1;(max/pos)>0;pos*=10)
                countsort(arr,n,pos);
}
int main(){
        int n;
        scanf("%d",&n);
        int arr[n],i;
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        radixsort(arr,n);
        for(i=0;i<n;i++)
                printf("%d ",arr[i]);
}
```


Sample Input:

6

22 10 999 8765 12345 7

Sample Output:

7 10 22 999 8765 12345

5) Stack operations using arrays.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int Top=-1;
int n
int *stack;
void push(int val){
        if(Top==n-1)
                printf("Stack is full\n");
        else
                stack[++Top]=val;}
int pop(){
        int val;
        if(Top==-1)
                return -1;
        else{
                val=stack[Top];
                stack[Top--]=0;
                return val;}
}
void display(){
        int i;
        if(Top==-1)
                printf("Stack is empty\n");
        else{
                for(i=Top;i>=0;i--)
                        printf("%d ",stack[i]);
                printf("\n");}
}
int main(){
        int i,ch,val;
        scanf("%d",&n);
        stack=(int *)calloc(n,sizeof(int));
        while(1){
                printf("1.Push\t2.Pop\t3.Display\t4.Exit\n");
                scanf("%d",&ch);
```

```
                if(ch==1){
                        scanf("%d",&val);
                        push(val);}
                else if(ch==2){
                        val=pop();
                        if(val==-1)
                                printf("Stack is empty\n");
                        else
                                printf("%d\n",val);
                }
                else if(ch==3)
                        display();
                else
                        break;
        }
}
```

Sample Input and Sample Output:

```
5
1.Push  2.Pop   3.Display     4.Exit
2
Stack is empty
1.Push  2.Pop   3.Display     4.Exit
3
Stack is empty
1.Push  2.Pop   3.Display     4.Exit
1
10
1.Push  2.Pop   3.Display     4.Exit
3
10
1.Push  2.Pop   3.Display     4.Exit
1
20
1.Push  2.Pop   3.Display     4.Exit
1
30
1.Push  2.Pop   3.Display     4.Exit
```

1
40
1.Push 2.Pop 3.Display 4.Exit
1
50
1.Push 2.Pop 3.Display 4.Exit
3
50 40 30 20 10
1.Push 2.Pop 3.Display 4.Exit
1
60
Stack is full
1.Push 2.Pop 3.Display 4.Exit
3
50 40 30 20 10
1.Push 2.Pop 3.Display 4.Exit
2
50
1.Push 2.Pop 3.Display 4.Exit
3
40 30 20 10
1.Push 2.Pop 3.Display 4.Exit
1
100
1.Push 2.Pop 3.Display 4.Exit
3
100 40 30 20 10
1.Push 2.Pop 3.Display 4.Exit
4

6) Stack operations to evaluate the postfix expression.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char pe[10000];
int stack[10000],top=-1;
int is_op(char ch){
        char op[5]={'+','-','*','/','%'};
        if(ch==42||ch==43||ch==45||ch==47||ch==37)
                return 1;
        return 0;
}
void postevaluation(char *s)
{
        int i,k=0,val=0,op1,op2;
        char temp[100];
        for(i=0;s[i]!='\0';i++){
                if(is_op(s[i])){
                        op1=stack[top--];
                        op2=stack[top--];
                        if(s[i]=='+')
                                stack[++top]=op2+op1;
                        else if(s[i]=='-')
                                stack[++top]=op2-op1;
                        else if(s[i]=='*')
                                stack[++top]=op2*op1;
                        else if(s[i]=='/')
                                stack[++top]=op2/op1;
                        else
                                stack[++top]=op2%op1;
                }
                else{
                        if(s[i]!=' ')
                                temp[k++]=s[i];
                        else if(s[i]==' ' && !is_op(s[i-1]))
                        {
```

```
                                temp[k]='\0';
                                stack[++top]=atoi(temp);
                                k=0;
                        }
                }
        }
}
int main(){
        scanf("%[^\n]s",pe);
        postevaluation(pe);
        printf("%d",stack[0]);
}
```

Sample Input:

2 3 2 * + 1 2 / 3 * +

Sample Output:

8

7) Queue operations using arrays.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int f=-1,r=-1,n,*queue;
void enqueue(int val){
        if(r==n-1)
                printf("Overflow\n");
        else if(f==-1&&r==-1){
                queue[++r]=val;
                f=0;}
        else
                queue[++r]=val;}
int dequeue(){
        int val;
        if(f==-1&&r==-1)
                return -1;
        else if(f==r){
                val=queue[f];
                f=-1;
                r=-1;
                return val;}
        else{
                val=queue[f];
                f++;
                return val;}
}
void display(){
        int i;
        if(f==-1&&r==-1)
                printf("Queue is empty\n");
        else{
                for(i=f;i<=r;i++)
                        printf("%d ",queue[i]);
                printf("\n");}
}
```

```
int main(){
        int i,ch,val;
        scanf("%d",&n);
        queue=(int *)calloc(n,sizeof(int));
        while(1){
                printf("1.Enqueue\t2.Dequeue\t3.Display\t4.Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&val);
                        enqueue(val);
                }
                else if(ch==2){
                        val=dequeue();
                        if(val==-1)
                                printf("Underflow\n");
                        else
                                printf("%d\n",val);}
                else if(ch==3)
                        display();
                else
                        break;}
}
```


Sample Input and Sample Output:

```
5
1.Enqueue    2.Dequeue    3.Display    4.Exit
1
10
1.Enqueue    2.Dequeue    3.Display    4.Exit
1
20
1.Enqueue    2.Dequeue    3.Display    4.Exit
1
30
1.Enqueue    2.Dequeue    3.Display    4.Exit
1
40
1.Enqueue    2.Dequeue    3.Display    4.Exit
```

3
10 20 30 40
1.Enqueue 2.Dequeue 3.Display 4.Exit
2
10
1.Enqueue 2.Dequeue 3.Display 4.Exit
3
20 30 40
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
50
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
60
Overflow
1.Enqueue 2.Dequeue 3.Display 4.Exit
3
20 30 40 50
1.Enqueue 2.Dequeue 3.Display 4.Exit
2
20
1.Enqueue 2.Dequeue 3.Display 4.Exit
4

EXPERIMENT NUMBER: 8 **DATE:**

8) Singly linked list and its operations

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;};
typedef struct node node;
node *head=NULL;
void insert_at_end(int val){
        node *nn,*temp=head;
        nn=(node *)malloc(sizeof(node));
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=nn;}
}
int delete_at_end(){
        int val;
        node *temp;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
                temp=head;
                while(temp->next->next!=NULL)
                        temp=temp->next;
                val=temp->next->data;
                temp->next=NULL;
                return val;}
```

```
}
void display(){
        node *temp=head;
        if(head==NULL)
                printf("NO NODES\n");
        else{
                while(temp!=NULL){
                        printf("%d %d %d\n",temp,temp->data,temp->next);
                        temp=temp->next;}
        }
}
void insert_at_head(int val){
        node *nn;
        nn=(node *)malloc(sizeof(node));
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                nn->next=head;
                head=nn;}
}
int delete_at_head(){
        int val;
        node *temp;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
                temp=head;
                temp=temp->next;
                head->next=NULL;
                val=head->data;
                head=temp;
```

```
                return val;}
}
void insert_at_pos(int pos,int val){
        node *temp,*nn;
        int c=1,i;
        nn=(node *)malloc(sizeof(node));
        nn->data=val;
        nn->next=NULL;
        temp=head;
        while(temp!=NULL){
                temp=temp->next;
                c++;}
        if(head==NULL)
                head=nn;
        else if(pos==1)
                insert_at_head(val);
        else if(c<=pos)
                insert_at_end(val);
        else{
                temp=head;
                for(i=1;i<pos-1;i++)
                        temp=temp->next;
                nn->next=temp->next;
                temp->next=nn;}
}
int delete_at_pos(int pos){
        node *temp=head;
        int c=1,val,i;
        if(head==NULL)
                return -1;
        else if(pos==1)
                return delete_at_head();
        while(temp!=NULL){
                temp=temp->next;
                c++;}
        if(c<=pos)
                return delete_at_end();
```

```
        else{
                temp=head;
                for(i=1;i<pos-1;i++)
                        temp=temp->next;
                val=temp->next->data;
                temp->next=temp->next->next;
                return val;}
}
int main(){
        int ch,val,pos;
        while(1){
                printf("1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at
head 6)Insert at position 7)Delete at position 8)Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&val);
                        insert_at_end(val);}
                else if(ch==2){
                        val=delete_at_end();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
                else if(ch==3)
                        display();
                else if(ch==4){
                        scanf("%d",&val);
                        insert_at_head(val);}
                else if(ch==5){
                        val=delete_at_head();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
                else if(ch==6){
                        scanf("%d%d",&pos,&val);
                        insert_at_pos(pos,val);}
                else if(ch==7){
```

```
                        scanf("%d",&pos);
                        val=delete_at_pos(pos);
                        if(val==-1)
                                printf("NO NODES TO DELETE AT THAT POSITION\n");
                        else
                                printf("%d\n",val);}
                else
                        break;
        }
}
```

Sample Input and Sample Output:

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

1

10

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

7607392 10 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

4

20

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

7607424 20 7607392

7607392 10 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

2

10

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

7607424 20 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

6

1 10

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

7607488 10 7607424

7607424 20 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

7

1

10

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

7607424 20 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

5

20

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

NO NODES

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

8

9) Doubly linked list and its operations

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        struct node *prev;
        int data;
        struct node *next;};
typedef struct node node;
node *head=NULL;
void insert_at_end(int val){
        node *nn,*temp=head;
        nn=(node *)malloc(sizeof(node));
        nn->prev=NULL;
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=nn;
                nn->prev=temp;}
}
int delete_at_end(){
        int val;
        node *temp=head,*temp1;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
                while(temp->next->next!=NULL)
                        temp=temp->next;
                temp1=temp->next;
```

```
                temp->next=NULL;
                temp1->prev=NULL;
                return temp1->data;}
}
void display(){
        node *temp=head;
        if(temp==NULL)
                printf("NO NODES\n");
        else{
                while(temp!=NULL){
                        printf("(%d) %d %d %d\n",temp,temp->prev,temp->data,temp->next);
                        temp=temp->next;}
        }
}
void insert_at_head(int val){
        node *nn;
        nn=(node *)malloc(sizeof(node));
        nn->prev=NULL;
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                nn->next=head;
                head->prev=nn;
                head=nn;}
}
int delete_at_head(){
        int val;
        node *temp;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
```

```
                temp=head->next;
                val=head->data;
                temp->prev=NULL;
                head->next=NULL;
                head=temp;
                return val;}
}
void insert_at_pos(int pos,int val){
        node *temp,*nn,*temp1;
        int nc=1,i;
        nn=(node *)malloc(sizeof(node));
        nn->prev=NULL;
        nn->data=val;
        nn->next=NULL;
        temp=head;
        while(temp!=NULL){
                temp=temp->next;
                nc++;}
        if(head==NULL)
                head=nn;
        else if(pos==1)
                insert_at_head(val);
        else if(nc<=pos)
                insert_at_end(val);
        else{
                temp=head;
                for(i=1;i<pos-1;i++){
                        temp=temp->next;
                temp1=temp->next;
                temp->next=nn;
                nn->prev=temp;
                temp1->prev=nn;
                nn->next=temp1;}
}
int delete_at_pos(int pos){
        int val,nc=0,i;
        node *temp=head,*temp1;
```

```
        while(temp!=NULL){
                temp=temp->next;
                nc++;}
        if(head==NULL||pos>nc)
                return -1;
        if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        if(pos==1)
                return delete_at_head();
        if(pos==nc)
                return delete_at_end();
        temp=head;
        for(i=1;i<pos-1;i++)
                temp=temp->next;
        temp1=temp->next;
        temp->next=temp1->next;
        temp->next->prev=temp;
        temp1->next=NULL;
        temp1->prev=NULL;
        return temp1->data;
}
int main(){
        int ch,val,pos;
        while(1){
                printf("1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at
head 6)Insert at position 7)Delete at position 8)Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&val);
                        insert_at_end(val);}
                else if(ch==2){
                        val=delete_at_end();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
```

```
                else if(ch==3)
                        display();
                else if(ch==4){
                        scanf("%d",&val);
                        insert_at_head(val);}
                else if(ch==5){
                        val=delete_at_head();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
                else if(ch==6){
                        scanf("%d%d",&pos,&val);
                        insert_at_pos(pos,val);}
                else if(ch==7){
                        scanf("%d",&pos);
                        val=delete_at_pos(pos);
                        if(val==-1)
                                printf("NO NODES TO DELETE AT THAT POSITION\n");
                        else
                                printf("%d\n",val);}
                else
                        break;
        }
}
```


Sample Input and Sample Output:

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

1

10

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

1

20

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

1

30

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

1

40

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

(2036768) 0 10 2036800

(2036800) 2036768 20 2036832

(2036832) 2036800 30 2036864

(2036864) 2036832 40 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

2

40

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

4

50

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

5

50

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

6

2 60

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

3

(2036768) 0 10 2036928

(2036928) 2036768 60 2036800

(2036800) 2036928 20 2036832

(2036832) 2036800 30 0

1)Insert at end 2)Delete at end 3)Display 4)Insert at head 5)Delete at head 6)Insert at position 7)Delete at position 8)Exit

7

2

60

10.1) Stack operations using Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;};
typedef struct node node;
node *head=NULL;
void insert(int val){
        node *nn,*temp;
        nn=(node *)malloc(sizeof(node));
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                temp=head;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=nn;}
}
int deletee(){
        int val;
        node *temp;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
                temp=head;
                while(temp->next->next!=NULL)
                        temp=temp->next;
                val=temp->next->data;
                temp->next=NULL;
                return val;}
```

```
}
void display(){
        node *temp;
        if(head==NULL)
                printf("NO NODES\n");
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d %d %d\n",temp,temp->data,temp->next);
                        temp=temp->next;}
        }
}
int main(){
        int ch,val;
        while(1){
                printf("1)Insert\t2)Delete\t3)Display\t4)Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&val);
                        insert(val);}
                else if(ch==2){
                        val=deletee();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
                else if(ch==3)
                        display();
                else
                        break;}
}
```


Sample Input and Sample Output:

```
1)Insert      2)Delete      3)Display      4)Exit
1
10
1)Insert      2)Delete      3)Display      4)Exit
```

1
20
1)Insert 2)Delete 3)Display 4)Exit
1
30
1)Insert 2)Delete 3)Display 4)Exit
1
40
1)Insert 2)Delete 3)Display 4)Exit
1
50
1)Insert 2)Delete 3)Display 4)Exit
3
10818624 10 10818656
10818656 20 10818688
10818688 30 10818720
10818720 40 10818752
10818752 50 0
1)Insert 2)Delete 3)Display 4)Exit
2
50
1)Insert 2)Delete 3)Display 4)Exit
2
40
1)Insert 2)Delete 3)Display 4)Exit
2
30
1)Insert 2)Delete 3)Display 4)Exit
3
10818624 10 10818656
10818656 20 0

EXPERIMENT NUMBER: 10.2 **DATE:**

10.2) Queue operations using Linked List

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;};
typedef struct node node;
node *head=NULL;
void insert_at_end(int val){
        node *nn,*temp;
        nn=(node *)malloc(sizeof(node));
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                temp=head;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=nn;}
}
int delete_at_head(){
        int val;
        node *temp;
        if(head==NULL)
                return -1;
        else if(head->next==NULL){
                val=head->data;
                head=NULL;
                return val;}
        else{
                temp=head;
                temp=temp->next;
                head->next=NULL;
                val=head->data;
                head=temp;
```

```
                return val;}
}
void display(){
        node *temp;
        if(head==NULL)
                printf("NO NODES\n");
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d %d %d\n",temp,temp->data,temp->next);
                        temp=temp->next;}
        }
}
int main(){
        int ch,val;
        while(1){
                printf("1)Insert at rear\t2)Delete at front\t3)Display\t4)Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&val);
                        insert_at_end(val);}
                else if(ch==2){
                        val=delete_at_head();
                        if(val==-1)
                                printf("NO NODES TO DELETE\n");
                        else
                                printf("%d\n",val);}
                else if(ch==3)
                        display();
                else
                        break;}
}
```


Sample Input and Sample Output:

```
1)Insert at rear      2)Delete at front      3)Display      4)Exit
1
10
1)Insert at rear      2)Delete at front      3)Display      4)Exit
```

1
20
1)Insert at rear 2)Delete at front 3)Display 4)Exit
1
30
1)Insert at rear 2)Delete at front 3)Display 4)Exit
3
12850208 10 12850240
12850240 20 12850272
12850272 30 0
1)Insert at rear 2)Delete at front 3)Display 4)Exit
2
10
1)Insert at rear 2)Delete at front 3)Display 4)Exit
3
12850240 20 12850272
12850272 30 0
1)Insert at rear 2)Delete at front 3)Display 4)Exit
1
40
1)Insert at rear 2)Delete at front 3)Display 4)Exit
3
12850240 20 12850272
12850272 30 12850304
12850304 40 0
1)Insert at rear 2)Delete at front 3)Display 4)Exit
4

11) Binary tree traversals : inorder, preorder and postorder

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
        struct Node *left;
        int data;
        struct Node *right;};
typedef struct Node node;
node *root=NULL,*adrs[100];
void preorder(node *root){
        if(root!=NULL){
                printf("%d ",root->data);
                preorder(root->left);
                preorder(root->right);}
}
void postorder(node *root){
        if(root!=NULL){
                postorder(root->left);
                postorder(root->right);
                printf("%d ",root->data);}
}
void inorder(node *root){
        if(root!=NULL){
                inorder(root->left);
                printf("%d ",root->data);
                inorder(root->right);}
}
void create_tree(int *arr,int n)
{
        int i,val=arr[0];
        node *nn=(node *)malloc(sizeof(node));
        node *nn1,*nn2;
        nn->data=val;
        nn->left=NULL;
        nn->right=NULL;
        adrs[0]=nn;
```

```
        root=nn;
        for(i=0;2*i+1<n;i++){
                nn1=(node *)malloc(sizeof(node));
                nn1->data=arr[2*i+1];
                nn1->left=NULL;
                nn1->right=NULL;
                nn2=(node *)malloc(sizeof(node));
                nn2->data=arr[2*i+2];
                nn2->left=NULL;
                nn2->right=NULL;
                adrs[2*i+1]=nn1;
                adrs[2*i+2]=nn2;
                adrs[i]->left=nn1;
                adrs[i]->right=nn2;}
}
int main(){
        int n;
        scanf("%d",&n);
        int i,arr[n];
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        create_tree(arr,n);
        printf("Inorder Travesal: ");
        inorder(root);
        printf("\nPostorder Traversal: ");
        postorder(root);
        printf("\nPreorder Traversal: ");
        preorder(root);
}
```


Sample Input:

7

10 20 30 40 50 60 70

Sample Output:

Inorder Travesal: 40 20 50 10 60 30 70

Postorder Traversal: 40 50 20 60 70 30 10

Preorder Traversal: 10 20 40 50 30 60 70

12) Binary Search Tree and its operations

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
        struct Node *left;
        int data;
        struct Node *right;};
typedef struct Node node;
node *root=NULL;
int search(int key){
        if(root==NULL)
                return 0;
        node *temp=root;
        while(temp!=NULL){
                if(temp->data==key)
                        return 1;
                else if(temp->data>key)
                        temp=temp->left;
                else
                        temp=temp->right;}
        return 0;}
void insert(int val){
        node *nn=(node *)malloc(sizeof(node)),*temp=root,*temp1;
        nn->data=val;
        nn->left=NULL;
        nn->right=NULL;
        if(root==NULL)
                root=nn;
        else{
                while(temp!=NULL){
                        temp1=temp;
                        if(temp->data==val)
                                return;
                        else if(temp->data>val)
                                temp=temp->left;
                        else
```

```
                                temp=temp->right;}
            if(temp1->data>val)
                        temp1->left=nn;
            else
                        temp1->right=nn;}
}
void delete_case1(node *temp,node *temp1,int key){
        if(temp==root){
                root=NULL;
                return;}
        if(temp1->left!=NULL && temp1->left->data==key)
                temp1->left=NULL;
        else
                temp1->right=NULL;}
void delete_case2(node *temp,node *temp1,int key){
        if(temp1==NULL){
                if(temp->left!=NULL){
                        temp=temp->left;
                        root=temp;}
                else{
                        temp=temp->right;
                        root=temp;}
                return;}
        if(temp->right!=NULL){
                if(temp1->right->data==key)
                        temp1->right=temp->right;
                else if(temp1->left->data==key)
                        temp1->left=temp->right;}
        else{
                if(temp1->right!=NULL && temp1->right->data==key)
                        temp1->right=temp->left;
                else if(temp1->left->data==key)
                        temp1->left=temp->left;}
}
int deletee(int key){
        if(root==NULL)
                return 0;
```

```
        node *temp=root,*temp1=NULL,*t1,*t2=NULL;
        while(temp!=NULL){
                if(temp->data==key){
                        if(temp->left==NULL && temp->right==NULL)
                                delete_case1(temp,temp1,key);
                        else if(temp->left==NULL || temp->right==NULL)
                                delete_case2(temp,temp1,key);
                        else{
                                t1=temp->right;
                                while(t1->left!=NULL{
                                        t2=t1;
                                        t1=t1->left;}
                                int val=t1->data;
                                t1->data=temp->data;
                                temp->data=val;
                                if(t2==NULL)
                                        t2=temp;
                                if(t1->left==NULL && t1->right==NULL)
                                        delete_case1(t1,t2,key);
                                else
                                        delete_case2(t1,t2,key);
                        }
                        return 1;}
                temp1=temp;
                else if(temp->data>key)
                        temp=temp->left;
                else
                        temp=temp->right;}
}
void inorder(node *root){
        if(root!=NULL){
                inorder(root->left);
                printf("%d ",root->data);
                inorder(root->right);}
}
void preorder(node *root){
        if(root!=NULL){
                printf("%d ",root->data);
```

```
                preorder(root->left);
                preorder(root->right);}
}
void postorder(node *root){
        if(root!=NULL){
                postorder(root->left);
                postorder(root->right);
                printf("%d ",root->data);}
}
int main(){
        int val,ch,key;
        while(1){
printf("1)Search\t2)Insert\t3)Delete\t4)Inorder\t5)Preorder\t6)Postorder\t7)Exit\n");
                scanf("%d",&ch);
                if(ch==1){
                        scanf("%d",&key);
                        if(search(key))
                                printf("Element found\n");
                        else
                                printf("Element not found\n");}
                else if(ch==2){
                        scanf("%d",&val);
                        insert(val);}
                else if(ch==3){
                        scanf("%d",&key);
                        if(deletee(key))
                                printf("%d is deleted\n",key);
                        else
                                printf("Element not found\n");}
                else if(ch==4){
                        inorder(root);
                        printf("\n");}
                else if(ch==5){
                        preorder(root);
                        printf("\n");}
                else if(ch==6){
                        postorder(root);
                        printf("\n");}
```

```
                else
                        break;}
}
```

Sample Input and Sample Output:

```
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 50
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 40
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 30
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 45
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 35
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
2 48
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
5
50 40 30 35 45 48
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
6
35 30 48 45 40 50
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
3
40
40 is deleted
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
3
45
45 is deleted
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
3
48
48 is deleted
1)Search 2)Insert 3)Delete 4)Inorder 5)Preorder 6)Postorder 7)Exit
4
30 35 50
```

List of Augmented Experiments:
(Any 2 of the following experiments can be performed)
13) Balanced brackets problem using stack. A bracket is considered to be any one of the following characters: (,), {, }, [, or]. Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int top=-1;
int *stack;
void push(char val)
        stack[++top]=val;
void pop()
        stack[top--]=0;
int balanced(char *str)
{
        int i;
        for(i=0;str[i];i++)
        {
                if(str[i]=='('||str[i]=='{'||str[i]=='[')
                        push(str[i]);
                else
                {
                        if(top==-1)
                                return 0;
                        else
                        {
                                if((str[i-1]=='('&&str[i]==')')||
                                   (str[i-1]=='{'&&str[i]=='}')||
                                   (str[i-1]=='['&&str[i]==']'))
                                        pop();
                                else
                                {
                                        push(str[i]);
                                }
                        }
                }
        }
```

```
        if(top!=-1)
                return 0;
        return 1;
}
int main()
{
        char str[10000];
        stack=(int *)calloc(10000,sizeof(int));
        scanf("%[^\n]s",str);
        if(balanced(str))
        {
                printf("String is balanced");
        }
        else
        {
                printf("Not a balanced string");
        }
}
```


Sample Input:

([]){}

Sample Output:

String is balanced

15) Represent Sparse Matrices using Linked Lists

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int row;
        int col;
        int data;
        struct node *next;};
typedef struct node node;
node *head=NULL;
void insert(int r,int c,int val){
        node *nn,*temp;
        nn=(node *)malloc(sizeof(node));
        nn->row=r;
        nn->col=c;
        nn->data=val;
        nn->next=NULL;
        if(head==NULL)
                head=nn;
        else{
                temp=head;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=nn;}
}
void display(){
        node *temp=head;
        if(head==NULL)
                printf("No nodes to display\n");
        else{
                while(temp!=NULL){
                        printf("%d %d %d %d %d\n",temp,temp->row,temp->col,temp->data,temp->next);
                        temp=temp->next;}
        }
}
int main(){
```

```
int r,c,arr[100][100],i,j;
scanf("%d%d",&r,&c);
for(i=0;i<r;i++){
        for(j=0;j<c;j++)
                scanf("%d",&arr[i][j]);
}
for(i=0;i<r;i++){
        for(j=0;j<c;j++){
                if(arr[i][j]!=0)
                        insert(i,j,arr[i][j]);}
}
display();}
```

Sample Input:

3 3

0 0 9

1 5 0

2 0 7

Sample Output:

10163264 0 2 9 10163296

10163296 1 0 1 10163328

10163328 1 1 5 10163360

10163360 2 0 2 10163392

10163392 2 2 7 0