LAB MANUAL
On
Data Structures using C Lab (16MCA16)
For MCA 1st Semester of VTU

Instructions to the Readers:

- This document is a useful material for the 1st Semester MCA students (2016 Scheme – 16MCA16) of Visvesvaraya Technological University, Belagavi, Karnatka.
- Though these programs are compiled and verified on Turbo C++ compiler, they can be executed on different C compilers with little/no modifications.
- It is advised to understand the theory concepts for respective programs by referring to the notes (available on the website).
- Clarifications/suggestions are welcome!!

1. Write a menu driven program in C for the following array operations:

a. Create an array of N integer elements
b. Display the array elements
c. Inserting an element at any valid position
d. Deleting an element from any valid position
e. Exit

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

int a[20], n;

void create()
{
        int i;
        printf("Enter the number of elements in an array:");
        scanf("%d", &n);
        printf("\nEnter array elements:");

        for(i=0;i<n;i++)
                scanf("%d", &a[i]);
}
```

```
void display()
{
	int i;
	printf("\nThe contents of the array are:\n");
	for(i=0;i<n;i++)
		printf("%d\t", a[i]);
}

void insert()
{
	int i, pos, item;

	printf("\nEnter the position for insertion:");
	scanf("%d", &pos);

	if(pos<0 || pos>n)
	{
		printf("\nInvalid position!!");
		getch();
	}
	else
	{
		printf("\nEnter item to be inserted:");
		scanf("%d", &item);
		for(i=n-1;i>=pos;i--)
			a[i+1]=a[i];

		a[pos]=item;
		printf("\nItem inserted successfully");
		n++;
	}
}

void del()
{
	int i, pos, item;

	printf("\nEnter the position from where to delete:");
	scanf("%d", &pos);

	if(pos<0 || pos>= n)
	{
		printf("\nInvalid postion");
		getch();
	}
```

```
        else
        {
                item = a[pos];
                printf("\nDeleted item is %d", item);

                for(i=pos;i<n;i++)
                        a[i]=a[i+1];

                n--;
        }
}

void main()
{
        int opt;
        clrscr();

        for(;;)
        {
                printf("\n ***** Array Operations *****");
                printf("\n 1.Create \n 2. Display \n 3. Insert \n 4. Delete \n5.Exit");
                printf("\nEnter your option:");
                scanf("%d", &opt);

                switch(opt)
                {
                        case 1: create();
                                break;
                        case 2: display();
                                break;
                        case 3: insert();
                                break;
                        case 4: del();
                                break;
                        case 5: exit(0);
                }
        }
}
```

2. Write a C program to implement following searching techniques:

a. Linear Search

b. Binary Search

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

int binary(int item, int a[],int low,int high)
{
	int mid;

	mid=(low+high)/2;

	if(low>high)
		return 0;

	if(a[mid]==item)
		return mid+1;
	else if(a[mid]>item)
		return binary(item,a,low,mid-1);
	else
		return binary(item,a,mid+1,high);
}

int linear(int item, int n, int a[])
{
	int i;
	for(i=0;i<n;i++)
		if(a[i]==item)
			return i+1;

	return 0;
}

void main()
{
	int a[20],n,pos,item, i, opt;
	clrscr();

	printf("Enter the array size:");
	scanf("%d",&n);

	for(;;)
	{
		printf("\n1. Binary Search \n2. Linear Search \n3. Exit");
```

```
                printf("\nEnter your option:");
                scanf("%d",&opt);
                switch(opt)
                {
                        case 1: printf("\nEnter the elements in ascending order:\n");
                                for(i=0;i<n;i++)
                                        scanf("%d", &a[i]);
                                printf("Enter the key element:");
                                scanf("%d",&item);
                                pos=binary(item,a,0,n-1);
                                break;
                        case 2: printf("\nEnter the elements:\n");
                                for(i=0;i<n;i++)
                                        scanf("%d", &a[i]);
                                printf ("Enter the key element:\n");
                                scanf ("%d",&item);
                                pos=linear(item,n,a);
                                break;
                        default : exit(0);
                }
                if(pos==0)
                        printf("Item not found\n");
                else
                        printf("Item found at the position %d",pos);

                getch();
        }
}
```

3. **Write a program to implement following sorting algorithms using user-defined functions:**
   a. **Bubble sort (Ascending order)**
   b. **Selection sort (Descending order)**

NOTE: Here, two functions are written respectively for bubble sort and selection sort. And main() function is used to call these functions.

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

void selection(int a[], int n)
{
        int i, j, temp;
```

```
	for(i=0;i<n-1;i++)
	{
		for(j=i+1;j<n;j++)
		{
			if(a[i]<a[j])
			{
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}
		}
	}
}

void bubble(int a[], int n)
{
	int i, j, temp;

	for(i=0;i<n;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;
			}
		}
	}
}

void main()
{
	int a[20],n,i, opt;
	clrscr();
	printf("Enter the size of the array:");
	scanf("%d",&n);

	for(;;)
	{
		printf("\n ***** Sorting *****\n");
		printf("\n 1. Selection Sort (Descending Order) \n 2. Bubble Sort
				(Ascending Order) \n 3. Exit ");
		printf("\n Enter your option:");
```

```
                scanf("%d",&opt);

                switch(opt)
                {
                        case 1: printf("\nEnter array elements:\n");
                                for(i=0;i<n;i++)
                                        scanf("%d",&a[i]);

                                selection(a, n);
                                printf("\nElements in Descending order:\n");
                                for(i=0;i<n;i++)
                                        printf("%d \t", a[i]);
                                break;
                        case 2: printf("\nEnter array elements:\n");
                                for(i=0;i<n;i++)
                                        scanf("%d",&a[i]);

                                bubble(a, n);
                                printf("\nElements in Descending order:\n");
                                for(i=0;i<n;i++)
                                        printf("%d \t", a[i]);
                                break;
                        case 3:
                        default: exit(0);
                }
        }
}
```

4. **Write a program in C for the following string operations (without using built-in functions)**
   a. **Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)**
   b. **Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.**

   Support the program with functions for each of the above operations.

(NOTE: In the below given program, it is assumed that size (or length) of the pattern and that of replacement string must be same.)

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

```
void read(char str[], char pat[], char rep[])
{
	fflush(stdin);
	printf("Enter string:");
	gets(str);
	printf("\nEnter Pattern:");
	gets(pat);
	printf("\nEnter replacement string:");
	gets(rep);
}

void match(char str[], char pat[], char rep[])
{
	int i, j=0, k=0, len_str=0, len_pat=0, len_rep=0;

	for(; str[len_str]!='\0';len_str++);

	for(; pat[len_pat]!='\0';len_pat++);

	for(; rep[len_rep]!='\0';len_rep++);


	if(len_pat>len_str)
	{
		printf("\n Pattern not found");
		getch();
	}
	else if(len_rep!=len_pat)
	{
		printf("\nPattern and replacement string should be of same size!!");
		getch();
	}
	else
	{
		for(i=0;i<str[i]!='\0';i++)
		{
			j=0;
			while(str[i+j]==pat[j] && j<len_pat)
				j++;

			if(j==len_pat)
			{
				printf("\npattern found at %d", i+1);
```

```
                        for(k=0;rep[k]!='\0';k++)
                            str[i+k]=rep[k];
                    }
                    else
                        continue;
                }
                printf("\n New string is: %s", str);
        }
}

void main()
{
        char str[100], pat[20], rep[20];
        int opt;
        clrscr();

        for(;;)
        {
                printf("\n ***** String Matching **** \n");
                printf("\n 1. Read \n 2. Match \n 3. Exit");
                printf("\nEnter your option:");
                scanf("%d", &opt);

                switch(opt)
                {
                        case 1: read(str, pat, rep);
                                break;
                        case 2: match(str, pat, rep);
                                break;
                        default: exit(0);
                }
        }
}
```

5. **Write a C Program to create a class called STACK to store Integers for the following operations (Array Implementation of Stack with maximum size MAX)**
   a. **Push an Element on to Stack**
   b. **Pop an Element from Stack**
   c. **Demonstrate Overflow and Underflow situations on Stack**
   d. **Display the status of Stack**
   e. **Exit**

Solution:
(NOTE: C is not an object oriented programming language. Hence, creating a class is not possible. It is a printing mistake. Hence, the program for stack operation done here by ignoring that term 'class'.)

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

#define MAX 3

void push(int st[], int item, int *t)
{
        if(*t==MAX-1)
        {
                printf("Stack full!!");
                return;
        }
        st[++(*t)]=item;
}

int pop(int st[], int *t)
{
        int item;

        if(*t== -1)
        {
            printf("\nStack Empty!!");
            return ;
        }
        item=st[(*t) - -];
        return item;
}

void disp(int st[], int *t)
{
        int i;

        if(*t== -1)
        {
                printf("\nStack Empty!!");
                return;
        }

        printf("\nStack contents:\n");
```

```
        for(i=*t ; i>=0; i- -)
                printf("\n%d",st[i]);
}

void main()
{
        int st[MAX], top= -1, opt, item;

        for(;;)
        {
                printf("\n*****Stack Operations****\n");
                printf("\n1. Push \n 2. Pop \n 3. Display \n 4. Exit\n");
                printf("Enter your option:");
                scanf("%d", &opt);

                switch(opt)
                {
                        case 1: printf("\nEnter item:");
                                scanf("%d",&item);
                                push(st, item, &top);
                                break;
                        case 2: item=pop(st, &top);
                                  printf("\n Deleted item is %d", item);
                                   break;
                        case 3: disp(st, &top);
                                   break;
                        case 4:
                        default:exit(0);
                }
        }
}
```

6. Implement a Program in C for converting an Infix Expression to Postfix Expression.

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

int inputpre(char sym)    //Function for input precedence
{
        switch(sym)
        {
                case '+' :
                case '-' : return 1;
                case '*' :
```

```
                case '/' : return 3;
                case '^' :
                case '$' : return 6;
                case '(' : return 9;
                case ')' : return 0;
                default : return 7;
        }
}

int stackpre(char sym)    //Function for stack precedence
{
        switch(sym)
        {
                case '+' :
                case '-' : return 2;
                case '*' :
                case '/' : return 4;
                case '^' :
                case '$' : return 5;
                case '(' : return 0;
                case '#' : return -1;
                default : return 8;
        }
}

void push (char item, int *top, char s[])
{
        s[++(*top)] = item;
}

char pop(int *top, char s[])
{
         return s[(*top)--];
}

void infix_to_postfix (char ifix[], char pfix[])
{
        int top = -1, i, j = 0;
        char s[30] , sym;

        push('#',&top,s);

        for(i=0;i < strlen(ifix);i++)
        {
                sym = ifix[i];
```

```
                while (stackpre(s[top]) > inputpre(sym))
                        pfix[j++] = pop(&top,s);

                if(stackpre(s[top]) != inputpre(sym))
                        push(sym,&top,s);
                else
                        pop(&top,s);
        }

        while(s[top] != '#')
                pfix[j++] = pop(&top,s);

        pfix[j] = '\0';
}

void main()
{
        char ifix[20], pfix[20];
        clrscr();

        printf("Enter valid infix expression\n");
        scanf("%s", ifix);
        infix_to_postfix (ifix,pfix);
        printf("The postfix expression is = %s", pfix);
}
```

7. Implement a Program in C for evaluating a Postfix Expression.

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

float oper(char sym, float op1, float op2)
{
        switch(sym)
        {
                case '+': return op1 + op2;
                case '-': return op1 - op2;
                case '*': return op1 * op2;
                case '/': if(op2== 0)
                          {
                                printf("Can't evalueate");
                                exit(0);
                          }
                          return op1 / op2;
```

```
                case '^':
                case '$': return pow(op1,op2);
        }
}

void push(float item, int *top, float s[ ])
{
        s[++(*top)] = item;
}

float pop(int *top, float s[ ])
{
        return s[(*top)--];
}

void main()
{
        float s[20], result, op1, op2, x;
        int top = -1, i;
        char postfix[20], sym;

        printf("Enter valid postfix expression\n");
        scanf("%s",postfix);

        for(i=0;i<strlen(postfix);i++)
        {
                sym = postfix[i];

                if(isdigit(sym))
                        push(sym-'0', &top, s);   // character to digit conversion
                else if (isalpha(sym))
                {
                        printf("Enter the value of %c: ", sym);
                        scanf("%f",&x);
                        push(x,&top,s);
                }
                else
                {
                        op2 = pop(&top,s);
                        op1 = pop(&top,s);
                        result = oper(sym,op1,op2);
                        push(result,&top,s);
                }
        }
```

```
        result = pop(&top,s);
        printf("Result =%.4f",result);
}
```

8. Write a C program to implement the following using recursion

a. Sum of n numbers
b. Generate Fibonacci sequence
c. Solve Towers of Hanoi Problem

a. Program to calculate the sum of n numbers

```
#include<stdio.h>

int sum(int n)
{
        if (n==1)
                return n;
        return n+sum(n-1);
}

void main()
{
        int n, s;

        printf("Enter n:");
        scanf("%d",&n);
        s=sum(n);
        printf("\nSum=%d",s);
}
```

b. Generate Fibonacci Sequence

```
#include<stdio.h>

int fibo(int n)
{
        if (n==1)
                return 0;
        else if (n==2)
                return 1;
        return fibo(n-1) + fibo (n-2);
}
```

```
void main()
{
        int n, i;

        printf("Enter value of n");
        scanf("%d", &n);

        if(n<=0)
                printf("invalid input");
        else
        {
                printf("Fibonacci Sequence is:\n");
                for(i=1;i<=n;i++)
                        printf("%d \t ", fibo(i));
        }
}
```

c. Program to solve Towers of Hanoi problem

```
#include<stdio.h>

int count=0;

void tower(int n, char s, char t, char d)
{
        if(n==1)
        {
                printf("Move disc 1 from %c to %c ", s, d);
                count ++;
                return;
        }
        tower(n-1, s, d, t);
        printf("Move disc %d from %c to %c", n, s,d);
        count ++;
        tower(n-1, t, s, d);
}

void main()
{
        int n;

        printf("Enter the number of discs");
        scanf("%d", &n);
```

```
        tower(n, 'A','B','C');
        printf("Total number of moves =%d", count);
}
```

9. Implement a menu driven Program in C for the following operations on Circular QUEUE of Integers (Array Implementation of Queue with maximum size MAX)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3

void insert(int q[], int *f, int *r, int item)
{
        if(*f==(*r+1)%MAX)
        {
                printf("\nQueue overflow");
                return;
        }

        *r=(*r+1)%MAX;
        q[*r]=item;

        if(*f==-1)
                (*f)++;
}

void del(int q[], int *f, int *r)
{
        if(*f==-1 )
        {
                printf("\nQueue underflow");
                return;
        }

        printf("\nDeleted element is %d", q[*f]);

        if(*f==*r)
                *f=*r=-1;
```

```
        else
                *f=(*f+1)%MAX;
}

void disp(int q[], int *f, int *r)
{
        int i;

        if(*f==-1)
        {
                printf("\nNo elements to display!!");
                return;
        }
        printf("\n Contents of queue:\n");

        if(*f>*r)
        {
                for(i=*f;i<MAX;i++)
                        printf("%d\t",q[i]);
                for(i=0;i<=*r;i++)
                        printf("%d\t", q[i]);
        }
        else
        {
                for(i=*f;i<=*r;i++)
                        printf("%d\t",q[i]);
        }
}

void main()
{       int q[10], f=-1, r=-1,item, opt;

        for(;;)
        {       printf("\n*****Circular Queue operations*****");
                printf("\n1.Insert\n 2.Delete\n 3.Display \n 4.Exit");
                printf("\nEnter your option: ");
                scanf("%d",&opt);

                switch(opt)
                {
                        case 1: printf("\nEnter item to be inserted:");
                                scanf("%d",&item);
                                insert(q,&f,&r,item);
                                break;
                        case 2: del(q, &f, &r);
```

```
                        break;
                case 3: disp(q, &f,&r);
                        break;
                case 4:
                default:exit(0);
                }
        }
}
```

10. Write a program to Simulate the working of a dequeue.

Solution: Dequeue or double-ended queue is a queue in which insertion/deletion are possible from both the ends. It is easy to implement a dequeue using linked list.

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

struct node
{
        int data;
        struct node *link;
};
typedef struct node *NODE;

NODE getnode()
{
        NODE x;
        x=(NODE) malloc(sizeof(struct node));
        if(x==NULL)
        {
                printf("no memory in heap");
                exit(0);
        }
        return x;
}

NODE insert_front(int item, NODE start)
{
        NODE temp;
        temp = getnode();
        temp->data=item;
        temp->link=start;
        return temp;
}
```

```
NODE delete_front(NODE start)
{
    NODE temp;

    if(start==NULL)
    {
      printf("no element to delete\n");
      return start;
    }
    temp=start;
    printf("Deleted item=%d", temp->data);
    start=start->link;
    free(temp);
    return start;
}

NODE insert_rear(int item, NODE start)
{
       NODE temp, cur;
       temp=getnode();
       temp->data=item;
       temp->link=NULL;

       if (start==NULL)
              return temp;

       cur=start;
       while(cur->link!=NULL)
              cur=cur->link;

       cur->link=temp;
       return start;
}

void display(NODE start)
{
       NODE temp;
       if(start==NULL)
       {
           printf("No element to display\n");
           return ;
       }
       printf("The contents of list:\n");

       temp=start;
```

```
        while(temp!=NULL)
        {
                printf("%d\n", temp->data);
                temp=temp->link;
        }
}

NODE delete_rear(NODE start)
{
        NODE prev, cur;

        if(start==NULL)
        {       printf("no element to delete\n");
                return start;
        }

        if(start->link==NULL)
        {       printf("\nDeleted element is%d", start->data);
                free(start);
                return NULL;
        }

        prev=NULL;
        cur=start;

        while(cur->link!=NULL)
        {       prev=cur;
                cur=cur->link;
        }
        printf("\nDeleted element is %d", cur->data);       .
        free(cur);
        prev->link=NULL;
        return start;
}

void main()
{       int opt, item;
        NODE start=NULL;

        for(;;)
        {
                printf("1.Insert Front\n 2.Insert Rear\n 3. Display\n");
                printf(" 4.Delete Front\n  5.Delete Rear\n");
                printf("enter your option:");
                scanf("%d",&opt);
```

```
                switch(opt)
                {
                        case 1: printf("\nenter item");
                                scanf("%d",&item);
                                  start=insert_front(item,start);
                                break;
                        case 2: printf("\nenter item");
                                scanf("%d",&item);
                                  start=insert_rear(item,start);
                                break;
                        case 3: display(start);
                                  break;
                        case 4: start=delete_front(start);
                                break;
                        case 5: start=delete_rear(start);
                                break;
                        default: exit(0);
                }
        }
}
```

11. Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem

a. Create a SLL of N Students Data.
b. Display the status of SLL and count the number of nodes.
c. Perform Insertion at the beginning /end of SLL.
d. Perform Deletion at the beginning /end of SLL.
e. Exit.

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

int count=0;
struct student
{
        int USN;
        char Name[20];
        char Branch[5];
        int Sem;
        struct student *link;
};
```

```
typedef struct student *NODE;

NODE getnode()
{
        NODE x;
        x = (NODE)malloc (sizeof (struct student));
        if (x == NULL)
        {
                printf("No memory space\n");
                exit(0);
        }
        return x;
}

NODE read()
{
        NODE temp;
        temp=getnode();
        printf("\nEnter new student details:");
        printf("\nEnter USN:");
        scanf("%d",&temp->USN);
        printf("\nEnter Name:");
        scanf("%s", temp->Name);
        printf("\nEnter Branch:");
        scanf("%s", temp->Branch);
        printf("\nEnter Semester:");
        scanf("%d", &temp->Sem);
        temp->link=NULL;
        return temp;
}

NODE insert_front(NODE start)
{
        NODE temp;

        temp=read();
        temp->link = start;
        count++;
        return temp;
}

NODE insert_rear(NODE start)
{
        NODE temp, cur;
        temp=read();
```

```
        count++;

        if (start==NULL)
                return temp;

        cur=start;
        while(cur->link!=NULL)
                cur=cur->link;

        cur->link=temp;
        return start;
}

void display (NODE start)
{
        NODE temp;
        if (start == NULL)
        {
                printf("List is empty\n");
                return;
        }
        printf("\nThe contents of the list are :\n");
        temp = start;
        printf("USN     NAME     BRANCH          SEM\n");
        while (temp != NULL)
        {
                printf("%d\t%s\t\t%s\t\t%d\n", temp->USN, temp->Name,
                                                temp->Branch, temp->Sem);
                temp = temp->link;
        }
        printf("\nNumber of nodes = %d", count);
}

NODE del_front(NODE start)
{
        NODE temp;

        if(start==NULL)
        {
                printf("\nNo element to delete!!");
                return start;
        }
        temp=start;
        printf("\nDeleted element is:");
```

```
        printf("\n%d \t %s \t %s \t %d", temp->USN, temp->Name,
                                                temp->Branch, temp->Sem);
        count--;

        if(start->link==NULL)
        {
                free(start);
                return NULL;
        }
        else
        {
                start=start->link;
                free(temp);
                return start;
        }
}

NODE del_rear(NODE start)
{
        NODE cur, prev;

        if(start==NULL)
        {
                printf("\nNo element to delete!!");
                return start;
        }

        if(start->link==NULL)
        {
                printf("\nDeleted element is:");
                printf("\n%d \t %s \t %s \t %d", start->USN, start->Name,
                                                start->Branch, start->Sem);
                free(start);
                return NULL;
        }
        prev=NULL;
        cur=start;

        while(cur->link!=NULL)
        {
                prev=cur;
                cur=cur->link;
        }

        printf("\nDeleted element is:");
```

```
        printf("\n%d \t %s \t %s \t %d", cur->USN, cur->Name, cur->Branch,
                                                        cur->Sem);

        free(cur);
        prev->link=NULL;
        count--;
        return start;
}

void main()
{
        NODE start=NULL;
        int opt;
        clrscr();

        for(;;)
        {
                printf("\n****Linked list operations****");
                printf("\n 1. Insert front \n 2. Inseart rear \n 3.Delete front ");
                printf("\n 4. Delete rear \n 5. Display \n 6. Exit");
                printf("\n Enter your option:");
                scanf("%d",&opt);

                switch(opt)
                {
                        case 1:start=insert_front(start);
                               break;
                        case 2: start=insert_rear(start);
                                break;
                        case 3: start=del_front(start);
                                break;
                        case 4: start=del_rear(start);
                                break;
                        case 5: display(start);
                                break;
                        case 6: exit(0);
                }
        }
}
```

12. Write a program to Simulate the working of a Singly circular linked list providing the following operations

a. Delete from the beginning/end

b. Delete a given element
c. Display &Insert is mandatory

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

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

typedef struct node *NODE;

NODE getnode()
{
        NODE x;
        x=(NODE) malloc(sizeof(struct node));

        if(x==NULL)
        {
                printf("no memory in heap");
                exit(0);
        }
        return x;
}

NODE insert_front(int item, NODE last)
{
        NODE temp;
        temp = getnode();

        temp->data=item;
        temp->link=temp;

        if(last==NULL)
                return temp;

        temp ->link = last->link;
        last ->link=temp;
        return last;
}
```

```
void display(NODE last)
{
        NODE temp;

        if(last==NULL)
        {
                printf("No element to display\n");
                return ;
        }

        temp=last->link;
        printf("The contents of list:\n");

        while(temp!=last)
        {
                printf("%d\n", temp->data);
                temp=temp->link;
        }

        printf("%d", temp->data);
}

NODE delete_front(NODE last)
{
        NODE temp;
        if(last==NULL)
        {
                printf("no element to delete\n");
                return NULL;
        }
        if(last->link==last)
        {
                printf("The item deleted is %d", last->data);
                free(last);
                return NULL;
        }
        temp=last->link;
        last->link=temp->link;
        printf("Item deleted is %d", temp->data);
        free(temp);
        return last;
}
```

```
NODE delete_rear(NODE last)
{
        NODE prev;

        if(last==NULL)
        {
                printf("no element to delete\n");
                return NULL;
        }

        if(last->link==last)
        {
                printf("The item deleted is %d", last->data);
                free(last);
                return NULL;
        }
        prev=last->link;

        while(prev->link!=last)
                prev=prev->link;

        prev->link=last->link;
        printf("\nDeleted element is %d", last->data);
        free(last);
        return prev;
}

NODE del_item(int key, NODE last)
{
        NODE prev, cur;

        if(last==NULL)
        {
                printf("\nList is empty!!\n");
                return last;
        }

        if(last==last->link &&key==last->data)
        {
                printf("\nDeleted item is %d", last->data);
                free(last);
                return NULL;
        }
        prev=last;
        cur=last->link;
```

```
        while(cur!=last && key!=cur->data)
        {
                prev=cur;
                cur=cur->link;
        }

        if(key==cur->data)
        {
                prev->link=cur->link;
                printf("\nDeleted item is %d", cur->data);

                if(cur==last)
                        last=prev;

                free(cur);
        }
        else
                printf("\nKey not found\n");

        return last;
}

void main()
{
        int opt, item, key;
        NODE last=NULL;
        clrscr();

        for(;;)
        {
                printf("\n****** Circular List Operations *******\n");
                printf("\n 1.Insert Front\n 2.Delete Front \n 3.Delete Rear");
                printf("\n 4.Delete a given element\n 5.Display \n 6.Exit \n");

                printf("Enter your option:");
                scanf("%d",&opt);

                switch(opt)
                {
                        case 1: printf("\nenter item:");
                                scanf("%d",&item);
                                last=insert_front(item, last);
                                break;
                        case 2: last=delete_front(last);
                                break;
```

```
                    case 3: last=delete_rear(last);
                            break;
                    case 4: printf("\nEnter key element to be deleted:");
                            scanf("%d",&key);
                            last=del_item(key, last);
                            break;
                    case 5: display(last);
                            break;
                    default: exit(0);
                }
        }
}
```

13. Write a C Program using Doubly Linked List to Implement Stack operations to store Integers

a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate Overflow and Underflow situations on Stack
d. Display the status of Stack
e. Exit

We know that to implement stack, we need following operations:

- Insert at front
- Delete from front
- Display

```
#include<stdio.h>
#include<alloc.h>
#include<stdlib.h>
#define MAX 3

int top=-1;

struct node
{
        struct node *llink;
        int data;
        struct node *rlink;
};
typedef struct node *NODE;

NODE getnode()
{
        NODE x;
```

```
        x=(NODE) malloc(sizeof(struct node));

        if(x==NULL)
        {
                printf("no memory in heap");
                exit(0);
        }
        return x;
}

NODE insert_front(int item, NODE start)
{
        NODE temp;

        if(top==MAX-1)
        {
                printf("\nStack Overflow!!");
                return start;
        }
        top++;
        temp = getnode();
        temp->llink=NULL;
        temp->data=item;
        temp->rlink=start;
        start->llink=temp;
        return temp;
}

void display(NODE start)
{
        NODE temp;

        if(top==-1)
        {
                printf("List is empty\n");
                return ;
        }
        temp=start;
        printf("\nThe contents of list:\n");
        while(temp!=NULL)
        {
                printf("%d\n", temp->data);
                temp=temp->rlink;
        }
}
```

```
NODE delete_front(NODE start)
{
        NODE temp;
        if(top==-1)
        {
                printf("\nStack Underflow!!");
                return start;
        }
        temp=start;
        printf("\nDeleted item %d", temp->data);
        start=start->rlink;
        start->llink=NULL;
        top--;
        free(temp);
        return start;
}

void main()
{
        int opt, item;
        NODE start=NULL;

        for(;;)
        {
                printf("\n***Stack using Double linked list****\n");
                printf("\n1. Push \n2.Pop \n3.Display \n4.Exit");
                printf("\nEnter you option:");
                scanf("%d",&opt);

                switch(opt)
                {
                        case 1: printf("\nEnter item to be inserted:");
                                scanf("%d",&item);
                                start=insert_front(item, start);
                                break;
                        case 2: start=delete_front(start);
                                break;
                        case 3: display(start);
                                break;
                        default: exit(0);
                }
        }
}
```

14. Implement a menu driven Program in C for the following operations on Binary Tree of Integers

a. Create a BST of N Integers
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element and report the appropriate message
d. Delete an element from BST
e. Exit

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

struct node
{
   struct node *llink;
   struct node *rlink;
   int data;
};
typedef struct node *NODE;

NODE getnode()
{
   NODE x=NULL;
   x = (NODE) malloc (sizeof (struct node));
   if (x == NULL)
   {
         printf("No memory space\n");
         exit(0);
   }
   return x;
}

NODE insert (int item, NODE root)
{
   NODE temp=NULL,prev,cur;

   temp = getnode();
   temp->rlink = NULL;
   temp->llink = NULL;
   temp->data = item;

   if(root == NULL)
         return temp;
```

```
    prev = NULL;
    cur = root;

    while (cur != NULL)
    {
          prev = cur;
          cur = (item < cur->data) ? cur->llink : cur->rlink;
    }
    if (item < prev->data)
          prev->llink = temp;
    else
          prev->rlink = temp;

    return root;
}

void search(int key, NODE root)
{
     NODE cur;

     if(root==NULL)
     {
           printf("empty tree");
           return;
     }

     cur=root;

     while(cur!=NULL && key!=cur->data)
          cur=(key<cur->data)? cur->llink : cur->rlink;

     if(cur==NULL)
          printf("key is not found");
     else
          printf("Key is found");
}

NODE  del_item(int key, NODE root)
{
     NODE cur, parent, suc, q;

     if(root==NULL)
     {
           printf("empty tree");
           return root;
```

```
    }

    parent=NULL;
    cur=root;

    while(cur!=NULL && key!=cur->data)
    {
          parent=cur;
          cur=(key<cur->data)? cur->llink : cur->rlink;
    }

    if(cur==NULL)
    {
          printf("key not found");
          return root;
    }

    if(cur->llink==NULL)
          q= cur ->rlink;
    else if(cur->rlink==NULL)
          q=cur->llink;
    else
    {
         suc=cur->rlink;
         while(suc->llink!=NULL)
               suc=suc->llink;

         suc->llink=cur->llink;
         q=cur->rlink;
    }

    if(parent==NULL)
         return q;

    if(cur==parent->llink)
         parent->llink=q;
    else
         parent->rlink=q;

    free(cur);
    return root;
}
```

```
void preorder(NODE root)          //Function to traverse tree in pre-order
{
   if(root != NULL)
   {
         printf("%d\t",root->data);
         preorder(root->llink);
         preorder(root->rlink);
   }
}

void inorder(NODE root)           //Function to traverse tree in in-order
{
   if(root != NULL)
   {
         inorder(root->llink);
         printf("%d\t",root->data);
         inorder(root->rlink);
   }
}

void postorder(NODE root)         //Function to traverse tree in post-order
{
   if(root != NULL)
   {
         postorder(root->llink);
         postorder(root->rlink);
         printf("%d\t",root->data);
   }
}

void main()
{
   NODE root = NULL;
   int opt,item;
   clrscr();

   for(; ;)
   {
         printf("\n ****** Binary Search Tree Operations *****\n");
         printf("1:Insert an element to tree\n");
         printf("2:Pre-Order Traversal\n");
         printf("3:In-Order Traversal\n");
         printf("4:Post-Order Traversal\n");
         printf("5:Search for a Key \n6:Delete an element \n");
         printf("7:Exit\n");
```

```
            printf("Enter your option:\n");
            scanf("%d",&opt);
            switch(opt)
            {
             case 1: printf("Enter the element to be inserted \n");
                    scanf("%d",&item);
                    root = insert(item,root);
                    break;
             case 2: printf("PREORDER TRAVERSAL\n");
                    preorder(root);
                    break;
             case 3: printf("INORDER TRAVERSAL\n");
                    inorder(root);
                    break;
             case 4: printf("POSTORDER TRAVERSAL\n");
                    postorder(root);
                    break;
             case 5:printf("\nEnter the item to be searched:");
                    scanf("%d",&item);
                    search(item, root);
                    break;
             case 6: printf("\nEnter the item to be deleted:");
                    scanf("%d",&item);
                    root=del_item(item, root);
                    break;
             default: exit(0);
            }
    }
}
```