BITS edu campus, Varnama
Babaria Institute of Technology
Department of Computer Science and Engineering
Lab Manual - Faculty

Subject Name: Data Structure Semester: 3rd
Subject Code: 3130702 Academic Year:2020-2021

Practical-1

AIM: Implement a program to insert, delete and search an element from array.

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

void display(int *a);
void search(int *a,int num);
void insert(int *a,int num,int pos);
void delete(int *a,int pos);
int n;

void main()
{
        int a[10],i,ch,num,pos,n;
        printf("Enter the value of n:\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter value of a[%d]:",i);
                scanf("%d",&a[i]);
        }
        do
        {
        printf("Choose any option:\n1. Insert\n2. Display\n3. Search\n4.
Delete\n5. Exit\n");
        scanf("%d",&ch);
                switch(ch)
                {
                        case 1 :
                        printf("Enter number and position:\n");
                        scanf("%d %d",&num,&pos);
                        insert(a,num,pos);
                        display(a);
                        break;
                        case 2 :
                        display(a);
                        break;
                        case 3 :
                        printf("Enter number you want to search :\n");
                        scanf("%d",&num);
                        search(a,num);
                        break;
                        case 4 :
                        printf("Enter position");
                        scanf("%d",&pos);
                        delete(a,pos);
                        display(a);
                        break;
                        case 5 :
                        default :
                        printf("Exited successfully\n");
                        break;
                }
        }while(ch!=5);
}
void display(int *a)
{
        int i;
        for(i=0;i<n;i++)
```

```
	{
		printf("a[%d]=%d",i,a[i]);
	}
}
void search(int *a,int num)
{
	int i,flag=0;
	for(i=0;i<n;i++)
	{
		if(a[i]==num)
		{
			flag=1;
			break;
		}
	}

	if(flag==0)
	{
		printf("Your number is not in the list");
	}
	else
	{
		printf("Your number is at position %d",i);
	}
}
void insert(int *a,int num,int pos)
{
	int i;

	for(i=n-1;i>=pos;i--)
	{
```

```
                a[i+1]=a[i];
        }
        a[pos]=num;
        n++;
}
void delete(int *a,int pos)
{
        int i;
        for(i=pos;i<n-1;i++)
        {
                a[i]=a[i+1];
        }
        n=n-1;
}
```

Subject Name: Data Structure	Semester: 3rd
Subject Code: 3130702	Academic Year:2020-2021

Practical-2

AIM:

Implement a program for stack that performs following operations using array.

a. PUSH

b. POP

c. PEEP

d. CHANGE

e. DISPLAY

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

#define N 5
int top = -1;
int stack[N];

//Function prototypes
void push(int item);
int pop();
int peep();
void change(int,int);
void display();

void main()
{
        int item, choice, cont = 1,pos,val;
        clrscr();

        while(cont == 1)
```

```
    {
        printf("\n1.Push onto stack");
        printf("\n2.Pop from stack");
        printf("\n3.Peep into stack");
        printf("\n4.Change into stack");
        printf("\n5.Display stack\n");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                printf("\nEnter the value of item:");
                scanf("%d",&item);
                push(item);
                display();
                break;

            case 2:
                item = pop();
                if(item != NULL)
                {
                    printf("\nItem popped out: %d\n",item);
                }
                display();
                break;

            case 3:
                item = peep();
                if(item != NULL)
                {
                    printf("\nItem at top is: %d\n",item);
                }
                break;

            case 4:
                printf("Enter the Position");
                scanf("%d",&pos);
                printf("Enter the Value");
                scanf("%d",&val);
                change(pos,val);
                display();
                break;
            case 5:
                display();
                break;
            default:
                printf("\nInvalid choice.\n");
                break;
        }

        printf("\nDo you want to continue (1/0): ");
        scanf("%d",&cont);
    }

    getch();
}

//Function Definition for push operation
void push(int item)
{
    if(top >= N-1)
```

```
        {
                printf("\nStack Overflow. Push not possible.\n");
        }
        else
        {
                top = top+1;
                stack[top] = item;
        }
}

//Function for pop operation
int pop()
{

        int item = NULL;

        if(top == -1)
        {
                printf("\nStack Underflow. Pop not possible.\n");
        }
        else
        {
                item = stack[top];
                stack[top] = NULL;
                top = top-1;
        }
        return(item);
}

//Function for peep operation
int peep()
```

```
{
        int item = NULL;

        if(top == -1)
        {
                printf("\nStack Underflow. No element in stack.\n");
        }
        else
        {
                item = stack[top];
        }
        return(item);
}

void change(int pos,int val)
{
        if(top==-1)
        {
                printf("stack is underflow");
        }
        else
        {
                stack[pos]=val;
        }
}

void display()
{
        int i;

        for(i=top;i>=0;i--)
        {
                printf("\n%d",stack[i]);
        }
}
```

BITS edu campus, Varnama
Babaria Institute of Technology
Department of Computer Science and Engineering
Lab Manual - Faculty

Subject Name: Data Structure Semester: 3rd
Subject Code: 3130702 Academic Year:2020-2021

Practical-3

AIM: Implement a program to convert infix notation to postfix notation using stack.

```
#include<stdio.h>
#include<conio.h>
#define n 50
char stack[n];
int top=-1,j=0;
char postfix[50];

void push(char);
char pop();
int priority(char);

void main()
{

        int i;
        char element,ch;
        char infix[50];
        clrscr();
        printf("Enter infix expression\n");
        gets(infix);

        printf("\nSymbol\tStack content\tpostfix expression");

        for(i=0;infix[i]!=NULL;i++)
        {
                ch=infix[i];
```

```
if(ch>='a' && ch<='z')
{
        postfix[j]=ch;
        j++;
}
else if(ch=='(')
{
        push(ch);
}
else if(ch==')')
{

        while((element=pop())!='(')
        {
                postfix[j]=element;
                j++;
        }
}
else
{

        while(priority(ch)<=priority(stack[top]))
        {
                if(stack[top]=='(')
                break;
                element=pop();
                postfix[j]=element;
                j++;
        }
        push(ch);
}
```

```
        postfix[j]=NULL;

        printf("\n%c\t%s\t\t%s",ch,stack,postfix);
        getch();
}

        while((element=pop())!='(')
        {
                postfix[j]=element;
                j++;
        }

        getch();
}



void push(char ch)
{
        if(top>=n-1)
        {
                printf("overflow");
        }
        else
        {
                top=top+1;
                stack[top]=ch;
        }
}

char pop()
```

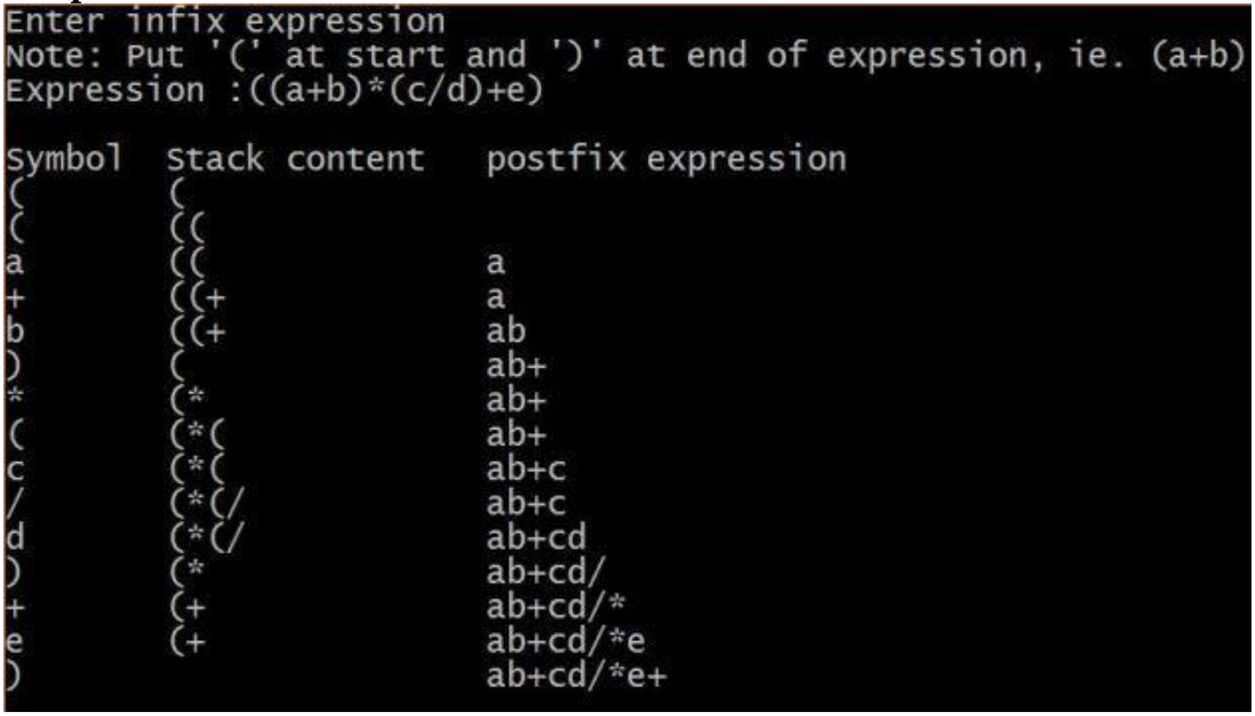
```
{
        char item;
        if(top==-1)
        {
                printf("stack is underflow");
                exit(0);
        }
        else
        {
                top=top-1;
                item=stack[top+1];
                stack[top+1]=NULL;
        }
        return item;
}

int priority(char ch)
{
        char operand[6]={'+','-','*','/','(','\0'};
        int prio[5]={1,1,2,2,3};
        int i,a;

        for(i=0;i<5;i++)
        {
                if(ch==operand[i])
                {
                        a=prio[i];
                        break;
                }
        }
        return a;
```

```
    }
```

Output:



```
Enter infix expression
Note: Put '(' at start and ')' at end of expression, ie. (a+b)
Expression :((a+b)*(c/d)+e)

Symbol  Stack content   postfix expression
(       (
(       ((
a       ((              a
+       ((+             a
b       ((+             ab
)       (               ab+
*       (*              ab+
(       (*(             ab+
c       (*(             ab+c
/       (*(/            ab+c
d       (*(/            ab+cd
)       (*              ab+cd/
+       (+              ab+cd/*
e       (+              ab+cd/*e
)                       ab+cd/*e+
```

Practical-4

AIM:

Write a program to implement QUEUE / CIRCULAR QUEUE using arrays that performs following operations:

a. INSERT

b. DELETE

c. DISPLAY

simple queue

```
#include <stdio.h>

#define N 5

void insert(int);
void delet();
void display();

int queue[N];
int rear = -1;
int front = -1;
main()
{
   int item,choice;
   while (1)
   {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
```

```
         printf("Enter your choice : ");
         scanf("%d", &choice);
         switch (choice)
         {
             case 1:
                      printf("Inset the element in queue : ");
                      scanf("%d", &item);
                      insert(item);
                      break;
             case 2:
                      delet();
                      break;
             case 3:
                      display();
                      break;
             case 4:
                      exit(1);
             default:
             printf("Wrong choice \n");
         } /*End of switch*/
    } /*End of while*/
} /*End of main()*/

void insert(int item)
{
    if (rear == N-1)
    {
         printf("Queue Overflow \n");
    }
    else
    {
         if(front==-1)
         {
                 front=rear=0;
         }
         else
         {
                 rear=rear+1;
         }
         queue[rear] = item;
    }
}

void delet()
{
    if (front == -1)
    {
         printf("Queue Underflow \n");
    }
    else
    {
         printf("Element deleted from queue is : %d\n", queue[front]);
         queue[front]=NULL;

         if(front==rear)
         {
                 front=rear=-1;
         }
         else
         {
                 front = front + 1;
         }
    }
}

void display()
{
    int i;
    if (front == -1)
         printf("Queue is empty \n");
    else
    {
         printf("Queue is : \n");
         for (i = front; i <= rear; i++)
             printf("\t%d ", queue[i]);
    }
}
```

Circular Queue

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

#define N 5
```

```
void insert(int);
void delet();
void display();

int cqueue[N];
int rear = -1;
int front = -1;
main()
{
   int item,choice;
   while (1)
   {
        printf("\n1.Insert element to circular queue \n");
        printf("2.Delete element from circular queue \n");
        printf("3.Display all elements of circular queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
           case 1:
                   printf("Inset the element in queue : ");
                   scanf("%d", &item);
                   insert(item);
                   break;
           case 2:
                   delet();
                   break;
           case 3:
                   display();
```

```
                break;
            case 4:
                exit(1);
            default:
            printf("Wrong choice \n");
        } /*End of switch*/
    } /*End of while*/
} /*End of main()*/

void insert(int item)
{
    if ((rear == N-1 && front==0) || (front==rear+1))
    {
        printf("Queue Overflow \n");
    }
    else
    {
        if(front==-1)
        {
            front=rear=0;
        }
        else if(rear==N-1)
        {
            rear=0;
        }
        else
        {
            rear=rear+1;
        }
        cqueue[rear] = item;
    }
```

```
}

void delet()
{
   if (front == -1)
   {
       printf("Queue Underflow \n");
   }
   else
   {
       printf("Element deleted from queue is : %d\n", cqueue[front]);
       cqueue[front]=NULL;

       if(front==rear)
       {
              front=rear=-1;
       }
       else if(front==N-1)
       {
              front=0;
       }
       else
       {
              front = front + 1;
       }
   }
}

void display()
{
   int i;
```

```
    if (front == -1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = 0; i < N; i++)
           printf("\t%d ", cqueue[i]);
    }
}
```

Practical-5

AIM:

Write a menu driven program to implement following operations on the singly linked list.

a. Insert a node at the front of the linked list.

b. Insert a node at the end of the linked list.

c. Insert a node such that linked list is in ascending order. (According to info. Field)

d. Delete a first node of the linked list.

e. Delete a node before specified position.

f. Delete a node after specified position.

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

struct node
{
	int data;
	struct node *next;
}*start=NULL;

void create_singly_LL();
void display_singly_LL();
void insert_begin(int);
void insert_end(int);
void insert_before(int,int);
void insert_after(int,int);
void insert_sorted(int);
void delet_begin();
void delet_end();
void delet_before(int);
void delet_after(int);
void delet_given(int);

void main()
{
        int val,num,ch;

        do
        {
        printf("\n1. Create LL\n2. Display LL\n3. Insert at Begin\n4. Insert at End\n5. Insert Before Given Node\n6. Insert After Given Node\n7. Insert in Sorted LL\n8. Delete from Begin\n9. Delete from End\n10. Delete Before Given Node\n11. Delete After Given Node\n12. Delete Given Node\n13. Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&ch);

        switch(ch)
        {
                case 1: create_singly_LL();
                                display_singly_LL();
                                break;
                case 2: display_singly_LL();
                                break;
                case 3: printf("\nEnter value of node : ");
                                scanf("%d",&val);
                                insert_begin(val);
                                display_singly_LL();
                                break;
```

```
case 4: printf("\nEnter value of node : ");
            scanf("%d",&val);
            insert_end(val);
            display_singly_LL();
            break;
case 5: printf("\nEnter value of node : ");
            scanf("%d",&val);
            printf("\nEnter Given node : ");
            scanf("%d",&num);
            insert_before(num,val);
            display_singly_LL();
            break;
case 6: printf("\nEnter value of node : ");
            scanf("%d",&val);
            printf("\nEnter Given node : ");
            scanf("%d",&num);
            insert_after(num,val);
            display_singly_LL();
            break;
case 7: printf("\nEnter value of node : ");
            scanf("%d",&val);
            insert_sorted(val);
            display_singly_LL();
            break;
case 8: delet_begin();
            display_singly_LL();
            break;
case 9: delet_end();
            display_singly_LL();
            break;
case 10: printf("\nEnter Given node : ");
```

```
                     scanf("%d",&num);
                     delet_before(num);
                     display_singly_LL();
                    break;
           case 11: printf("\nEnter Given node : ");
                     scanf("%d",&num);
                     delet_after(num);
                     display_singly_LL();
                    break;
           case 12: printf("\nEnter value of node : ");
                     scanf("%d",&val);
                     delet_given(val);
                     display_singly_LL();
                    break;
           case 13:break;
           default:printf("Wrong Choice");
      }
      }while(ch!=13);


   getch();
}

void create_singly_LL()
{
   struct node *new_node,*ptr;
   int val,choice;


   printf("Enter the Data Item (-1 for end): ");
   scanf("%d",&val);
```

```
while (val!=-1)
{
     new_node = (struct node *)malloc(sizeof(struct node));

     if(new_node == NULL)
     {
        printf("Overflow");
        return;
     }

     new_node->data = val;
     new_node->next = NULL;

     if (start == NULL) // first node
     {
        start=new_node;
     }
     else
     {
        ptr=start;
        while(ptr->next != NULL)
        {
             ptr = ptr->next;
        }
        ptr->next = new_node;
     }

printf("Enter the Data Item (-1 for end): ");
scanf("%d",&val);
}
```

```
}

void display_singly_LL()
{
   struct node *ptr;
   printf("\n status of the linked list is :\n");
   ptr = start;
   while (ptr != NULL)
   {
      printf("%d -> ", ptr->data);
      ptr = ptr -> next;
   }
   printf("NULL\n");
}

void insert_begin(int val)
{
        struct node *new_node;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;
                new_node->next=start;
                start=new_node;
```

```
	}
}

void insert_end(int val)
{
	struct node *new_node,*ptr;

	new_node = (struct node*)malloc(sizeof(struct node));

	if(new_node==NULL)
	{
		printf("Memory Full. Overflow");
	}
	else
	{
		new_node->data=val;
		new_node->next=NULL;

		ptr=start;
		while(ptr->next != NULL)
		{
			ptr=ptr->next;
		}
		ptr->next=new_node;
	}
}

void insert_before(int num,int val)
{
	struct node *new_node,*ptr,*preptr;
	new_node = (struct node*)malloc(sizeof(struct node));

	if(new_node==NULL)
	{
		printf("Memory Full. Overflow");
	}
	else
	{
		new_node->data=val;

		ptr=start;
		preptr=ptr;
		while(ptr->data != num)
		{
			preptr=ptr;
			ptr=ptr->next;
		}
		preptr->next=new_node;
		new_node->next=ptr;
	}
}

void insert_after(int num,int val)
{
	struct node *new_node,*ptr,*preptr;

	new_node = (struct node*)malloc(sizeof(struct node));

	if(new_node==NULL)
	{
		printf("Memory Full. Overflow");
```

```
    }
    else
    {
        new_node->data=val;

        ptr=start;
        preptr=ptr;
        while(preptr->data != num)
        {
            preptr=ptr;
            ptr=ptr->next;
        }
        preptr->next=new_node;
        new_node->next=ptr;
    }
}

void insert_sorted(int val)
{
        struct node *new_node,*ptr,*preptr;

    new_node = (struct node*)malloc(sizeof(struct node));

    if(new_node==NULL)
    {
        printf("Memory Full. Overflow");
    }
    else
    {
        new_node->data=val;
```

```
                ptr=start;
                preptr=ptr;
                while(ptr->data < val)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=new_node;
                new_node->next=ptr;
        }
}

void delet_begin()
{
        struct node *ptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                start=start->next;
                free(ptr);
        }
}

void delet_end()
{
        struct node *ptr,*preptr;
        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                preptr=ptr;
                while(ptr->next!=NULL)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=NULL;
                free(ptr);
        }
}

void delet_before(int num)
{
        struct node *ptr,*preptr,*prepreptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                prepreptr=preptr=ptr;
                while(ptr->data!=num)
                {
                        prepreptr=preptr;
                        preptr=ptr;
                        ptr=ptr->next;
                }
                prepreptr->next=ptr;
                free(preptr);
        }
}

void delet_after(int num)
{
        struct node *ptr,*preptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                preptr=ptr;
                while(ptr->data!=num)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=ptr->next;
                free(ptr);
        }
```

```
}

void delet_given(int val)
{
        struct node *ptr,*preptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                preptr=ptr;
                while(ptr->data!=val)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=ptr->next;
                free(ptr);
        }
}
```

Practical-6

AIM:

Write a program to implement following operations on the doubly linked list.

a. Insert a node at the front of the linked list.

b. Insert a node at the end of the linked list.

c. Delete a last node of the linked list.

d. Delete a node before specified position.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next,*prev;
};

void display(struct node *h)
{
        struct node *t;
        t=h;
        printf("\nThe List is as follows:\n");
        while(t!=NULL)
        {
                printf(" %d",t->data);
                t= t->next;
        }
        printf("\n================");
}
```

```
struct node* delFromK(struct node *h,int pos)
{
        struct node *t=h;
        if(t==NULL )
        {
                return h;
        }

        if(pos==1)
        {
                t = t->next;
                t->prev = NULL;
                return t;
        }

        while(--pos!=1 && t->next!=NULL)
        {
                t = t->next;
        }

        if(pos==1 && t->next!=NULL)
        {


                t->next = t->next->next;
                t->next->prev = t;
        }

        else
        {
                printf("\n Please enter correct position\n");
                }
        return h;
}

struct node* delFromEnd(struct node *h)
{
        struct node *t=h;
        if(t==NULL )
        {
                return h;
        }
        if(t->next==NULL)
        {
                return NULL;
        }
        while(t->next->next!=NULL)
        {
                t = t->next;
        }

        t->next = NULL;
        return h;
}
struct node* delFromHead(struct node *h)
{
        struct node *t=h;
        if(t==NULL )
        {
                return h;
        }
        if(t->next==NULL)
        {
                return NULL;
        }
        t = t->next;
        t->prev = NULL;
        return t;
}

struct node* insertatEnd(struct node *h,int ele)
{
        struct node *t,*new1;
        t=h;
        new1 = calloc(1,sizeof(struct node));
        new1->data=ele;
        new1->next = NULL;
        if(t==NULL)
        {
                return new1;
        }
        while(t->next!=NULL)
        {
                t = t->next;
        }
        new1->prev = t;
        t->next = new1;
        return h;
}
struct node* insertatHead(struct node *h,int ele)
{
        struct node *t,*new1;
        t=h;
        new1 = calloc(1,sizeof(struct node));
        new1->data=ele;

        if(t==NULL)
        {
                new1->next = NULL;
        //      h=new1;
        //      printf("\n%u",h);
        //      printf("\n%u",new1);
                return new1;
        }

        new1->next = h;
        h->prev = new1;
//      h = new1;
        return new1;

}



void main()
{
        struct node *h=NULL;
        int choice,e;


        while(1)
        {
```

```
		printf("\nplease select the choice:\n 1.InsertAtHead 2.InsertatEnd 
3.DeleteFromHead 4.DeletefromEnd 5.DeletefromK 6.Display 7.Exit\n");
		scanf("%d",&choice);

		switch(choice)
		{
			case 1: printf("\n Enter element you want to Insert at Head: 
");
					scanf("%d",&e);
					h = insertatHead(h,e);
					break;
			case 2: printf("\n Enter element you want to Insert at End: 
");
					scanf("%d",&e);
					h = insertatEnd(h,e);
					break;
			case 3: h = delFromHead(h);
					break;
			case 4: h = delFromEnd(h);
					break;
			case 5: printf("\n Enter position you want to delete: ");
					scanf("%d",&e);
					h = delFromK(h,e);
					break;
			case 6: display(h);
					break;
			case 7: goto ext;
					break;
		}
	}
}
```

Practical-7

AIM:

Write a program to implement following operations on the circular Singly linked list.

a. Insert a node at the end of the linked list.

b. Insert a node before specified position.

c. Delete a first node of the linked list.

d. Delete a node after specified position.

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

struct node
{
	int data;
	struct node *next;
}*start=NULL;

void create_circular_LL();
void display_circular_LL();
void insert_begin(int);
void insert_end(int);
void insert_before(int,int);
void insert_after(int,int);
void insert_sorted(int);
void delet_begin();
void delet_end();
void delet_before(int);
void delet_after(int);
void delet_given(int);

void main()
{
	int val,num,ch;

	do
	{
	printf("\n1. Create LL\n2. Display LL\n3. Insert at Begin\n4. Insert at End\n5. Insert Before Given Node\n6. Insert After Given Node\n7. Insert in Sorted LL\n8. Delete from Begin\n9. Delete from End\n10. Delete Before Given Node\n11. Delete After Given Node\n12. Delete Given Node\n13. Exit");
	printf("\nEnter your choice : ");
	scanf("%d",&ch);

	switch(ch)
	{
		case 1: create_circular_LL();
				display_circular_LL();
				break;
```

```
case 2: display_circular_LL();
            break;
case 3: printf("\nEnter value of node : ");
            scanf("%d",&val);
            insert_begin(val);
            display_circular_LL();
            break;
case 4: printf("\nEnter value of node : ");
            scanf("%d",&val);
            insert_end(val);
            display_circular_LL();
            break;
case 5: printf("\nEnter value of node : ");
            scanf("%d",&val);
            printf("\nEnter Given node : ");
            scanf("%d",&num);
            insert_before(num,val);
            display_circular_LL();
            break;
case 6: printf("\nEnter value of node : ");
            scanf("%d",&val);
            printf("\nEnter Given node : ");
            scanf("%d",&num);
            insert_after(num,val);
            display_circular_LL();
            break;
case 7: printf("\nEnter value of node : ");
            scanf("%d",&val);
            insert_sorted(val);
            display_circular_LL();
            break;
            case 8: delet_begin();
                        display_circular_LL();
                        break;
            case 9: delet_end();
                        display_circular_LL();
                        break;
            case 10: printf("\nEnter Given node : ");
                         scanf("%d",&num);
                         delet_before(num);
                         display_circular_LL();
                        break;
            case 11: printf("\nEnter Given node : ");
                         scanf("%d",&num);
                         delet_after(num);
                         display_circular_LL();
                        break;
            case 12: printf("\nEnter value of node : ");
                         scanf("%d",&val);
                         delet_given(val);
                         display_circular_LL();
                        break;
            case 13:break;
            default:printf("Wrong Choice");
      }
      }while(ch!=13);


   getch();
}

void create_circular_LL()
```

```
{
    struct node *new_node,*ptr;
    int val,choice;


    printf("Enter the Data Item (-1 for end): ");
    scanf("%d",&val);

    while (val!=-1)
    {
        new_node = (struct node *)malloc(sizeof(struct node));

        if(new_node == NULL)
        {
            printf("Overflow");
            return;
        }

        new_node->data = val;

        if (start == NULL) // first node
        {
            start=new_node;
            new_node->next = start;
        }
        else
        {
            new_node->next = start;
            ptr=start;
            while(ptr->next != start)
            {
                ptr = ptr->next;
            }
            ptr->next = new_node;
        }

    printf("Enter the Data Item (-1 for end): ");
    scanf("%d",&val);
    }
}

void display_circular_LL()
{
    struct node *ptr;
    printf("\n status of the linked list is :\n");
    ptr = start;
    do
    {
        printf("%d->", ptr->data);
        ptr = ptr -> next;
    }while (ptr != start);

    printf("START\n");
}

void insert_begin(int val)
{
        struct node *new_node,*ptr;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;

                ptr=start;
                while(ptr->next!=start)
                {
                    ptr=ptr->next;
                }
                new_node->next=start;
                ptr->next=new_node;
                start=new_node;
        }
}

void insert_end(int val)
{
        struct node *new_node,*ptr;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;
                new_node->next=start;

                ptr=start;
                while(ptr->next != start)
                {
                    ptr=ptr->next;
                }
                ptr->next=new_node;
        }
}

void insert_before(int num,int val)
{
        struct node *new_node,*ptr,*preptr;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;

                ptr=start;
                preptr=ptr;
                while(ptr->data != num)
                {
                    preptr=ptr;
                    ptr=ptr->next;
```

```
                }
                preptr->next=new_node;
                new_node->next=ptr;
        }
}

void insert_after(int num,int val)
{
        struct node *new_node,*ptr,*preptr;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;

                ptr=start;
                preptr=ptr;
                while(preptr->data != num)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=new_node;
                new_node->next=ptr;
        }
}
```

```
void insert_sorted(int val)
{
                struct node *new_node,*ptr,*preptr;

        new_node = (struct node*)malloc(sizeof(struct node));

        if(new_node==NULL)
        {
                printf("Memory Full. Overflow");
        }
        else
        {
                new_node->data=val;

                ptr=start;
                preptr=ptr;
                while(ptr->data < val)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=new_node;
                new_node->next=ptr;
        }
}

void delet_begin()
{
        struct node *ptr;
```

```
    if(start==NULL)
    {
        printf("Linked List is Empty. Underflow");
    }
    else
    {
        ptr=start;
        while(ptr->next!=start)
        {
            ptr=ptr->next;
        }
        ptr->next=start->next;
        free(start);
        start=ptr->next;
    }
}

void delet_end()
{
    struct node *ptr,*preptr;

    if(start==NULL)
    {
        printf("Linked List is Empty. Underflow");
    }
    else
    {
        ptr=start;
        preptr=ptr;
        while(ptr->next!=start)
        {
            preptr=ptr;
            ptr=ptr->next;
        }
        preptr->next=start;
        free(ptr);
    }
}

void delet_before(int num)
{
    struct node *ptr,*preptr,*prepreptr;

    if(start==NULL)
    {
        printf("Linked List is Empty. Underflow");
    }
    else
    {
        ptr=start;
        prepreptr=preptr=ptr;
        while(ptr->data!=num)
        {
            prepreptr=preptr;
            preptr=ptr;
            ptr=ptr->next;
        }
        prepreptr->next=ptr;
        free(preptr);
    }
}
```

```
void delet_after(int num)
{
        struct node *ptr,*preptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
        {
                ptr=start;
                preptr=ptr;
                while(preptr->data!=num)
                {
                        preptr=ptr;
                        ptr=ptr->next;
                }
                preptr->next=ptr->next;
                free(ptr);
        }
}

void delet_given(int val)
{
        struct node *ptr,*preptr;

        if(start==NULL)
        {
                printf("Linked List is Empty. Underflow");
        }
        else
```

```
    {
        ptr=start;
        preptr=ptr;
        while(ptr->data!=val)
        {
            preptr=ptr;
            ptr=ptr->next;
        }
        preptr->next=ptr->next;
        free(ptr);
    }
}
```

Practical-8

AIM:

Write a program which create binary search tree and Implement tree traversing methods inorder, preorder and post-order traversal.

```
// C program to demonstrate insert operation in binary search tree
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct node
{
   int data;
   struct node *left, *right;
}*root=NULL;

void inorder(struct node *root)
{
   if (root != NULL)
   {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
   }
```

```
}

void preorder(struct node *root)
{
   if (root != NULL)
   {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
   }
}

void postorder(struct node *root)
{
   if (root != NULL)
   {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
   }
}

void CreateBST(struct node* node, int val)
{
   if (root == NULL)
   {
           struct node *newnode =  (struct node *)malloc(sizeof(struct node));
           newnode->data = val;
           newnode->left = newnode->right = NULL;
           root=newnode;
           printf("%d is Root Node",val);
```

}

```
    }
    else if (val < node->data)
    {
        if(node->left==NULL)
        {
            struct node *newnode =  (struct node *)malloc(sizeof(struct node));
            newnode->data = val;
            newnode->left = newnode->right = NULL;
            node->left=newnode;
            printf("%d is Left Child of %d",val,node->data);
        }
        else
        {
            CreateBST(node->left,val);
        }
    }
    else if (val >= node->data)
    {
        if(node->right==NULL)
        {
            struct node *newnode =  (struct node *)malloc(sizeof(struct node));
            newnode->data = val;
            newnode->left = newnode->right = NULL;
            node->right=newnode;
            printf("%d is Right Child of %d",val,node->data);
        }
        else
        {
            CreateBST(node->right,val);
        }
    }
```

```
}

void main()
{
   int val;
   while(1)
   {
        printf("\nEnter Node : ");
        scanf("%d",&val);
        if(val==99)
                break;
        CreateBST(root, val);
   }

   printf("\nInorder :");
   inorder(root);
   printf("\nPreorder :");
   preorder(root);
   printf("\nPostorder :");
   postorder(root);

   getch();
}
```

Practical-9

AIM:

Write a program to implement following sorting algorithm

a. Bubble

b. Merge

c. Quick

a) Bubble Sort

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

void bubble_sort(int[], int);

void main() {
  int arr[30], num, i;

  printf("\nEnter no of elements :");
  scanf("%d", &num);

  printf("\nEnter array elements :");
  for (i = 0; i < num; i++)
    scanf("%d", &arr[i]);

  bubble_sort(arr, num);
  getch();
}

void bubble_sort(int iarr[], int num) {
  int i, j, k, temp;

  printf("\nUnsorted Data:");
  for (k = 0; k < num; k++) {
    printf("%5d", iarr[k]);
  }

  for (i = 1; i < num; i++) {
    for (j = 0; j < num - 1; j++) {
      if (iarr[j] > iarr[j + 1]) {
        temp = iarr[j];
        iarr[j] = iarr[j + 1];
        iarr[j + 1] = temp;
      }
    }

    printf("\nAfter pass %d : ", i);
    for (k = 0; k < num; k++) {
      printf("%5d", iarr[k]);
    }
  }
}
```

b) Merge Sort

```
#include <stdio.h>
#define ARRAY_SIZE 5

void MergeSort(int arr[],int p,int r);
void Merge(int arr[],int p,int q,int r);

int main(void)
{
   int array[ARRAY_SIZE];
   int i;
```

```
    printf("\nEnter the Array elements : \n");
    for(i=0;i<ARRAY_SIZE;i++)
    {
        scanf("%d",&array[i]);
    }

    /*calling MergeSort()*/
    MergeSort(array,0,ARRAY_SIZE-1);

    /*After sorting, printing the array elements*/
    printf("\nAfter sorting array elements :");
    for(i=0;i<ARRAY_SIZE;i++)
    {
        printf(" %d ",array[i]);
    }
    return 0;
}

/*It does the merge-sort on the array
 *p is the starting index of array
 *r is the ending index of array*/
void MergeSort(int arr[],int p,int r)
{
    int q;
    if(p<r)
    {
        q = (r+p)/2;
        MergeSort(arr,p,q);
        MergeSort(arr,q+1,r);
        Merge(arr,p,q,r);
    }
```

```
}

/*Merge() does the merging of two sorted subarrays.
 *arr[p..q] and arr[q+1..r] are the two sorted arrays.
 *It does the marging without using sentinales*/
void Merge(int arr[],int p,int q,int r)
{
   int  *L,*R;
   int i = 0,j=0,n1,n2,k=p;

   n1 = q-p+1;
   n2 = r - q;
   L = malloc(sizeof(int)*n1);
   R = malloc(sizeof(int)*n2);

   for(i=0;i<n1;i++)
      L[i] = arr[p+i];
   for(j=0; j<n2;j++)
      R[j] = arr[q+j+1];

   /*reset i and j*/
   i = 0;
   j = 0;
   /*merging the items in sorted order
    *till we find end of any array L or R*/
   while(i<n1 && j<n2)
   {
      if(L[i]<R[j])
      {
         arr[k] = L[i];
         i = i+1;
```

```
        }
        else
        {
            arr[k] = R[j];
            j = j+1;
        }
        k++;
    }

    /*check whether any array has some elements left
     *if some items left then put them to the final O/P array*/
    if(i!=n1)
        for(;i<n1;i++){
            arr[k] = L[i];
            k++;
        }
    else if(j!=n1)
        for(;j<n2;j++){
            arr[k] = R[j];
            k++;
        }
        free(L);
        free(R);
}
```

c) Quick Sort

```
#include<stdio.h>

void swap (int a[], int left, int right)
{
 int temp;
 temp=a[left];
 a[left]=a[right];
 a[right]=temp;
}//end swap

void quicksort( int a[], int low, int high )
{
 int pivot;
 // Termination condition!
 if ( high > low )
 {
  pivot = partition( a, low, high );
  quicksort( a, low, pivot-1 );
  quicksort( a, pivot+1, high );
 }
} //end quicksort

int partition( int a[], int low, int high )
{
 int left, right;
 int pivot_item;
 int pivot = left = low;
 pivot_item = a[low];
 right = high;
 while ( left < right )
 {
  // Move left while item < pivot
  while( a[left] <= pivot_item )
   left++;
  // Move right while item > pivot
  while( a[right] > pivot_item )
   right--;
  if ( left < right )
   swap(a,left,right);
 }
 // right is final position for the pivot
 a[low] = a[right];
 a[right] = pivot_item;
 return right;
}//end partition

// void quicksort(int a[], int, int);
void printarray(int a[], int);

int main()
{
 int a[50], i, n;
 printf("\nEnter no. of elements: ");
 scanf("%d", &n);
 printf("\nEnter the elements: \n");
 for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
 printf("\nUnsorted elements: \n");
 printarray(a,n);
 quicksort(a,0,n-1);
 printf("\nSorted elements: \n");
 printarray(a,n);

}//end main
```

```
void printarray(int a[], int n)
{
 int i;
 for (i=0; i<n; i++)
  printf(" %d ", a[i]);
 printf("\n");
}//end printarray
```

Practical-10

AIM: Write a program to implement Binary search algorithm.

```
void main( )
{
        int arr[10] = { 1, 2, 3, 9, 11, 13, 17, 25, 57, 90 } ;
        int mid, lower = 0 , upper = 9, num, flag = 1 ;

        clrscr( ) ;

        printf ( "Enter number to search: " ) ;
        scanf ( "%d", &num ) ;

        for ( mid = ( lower + upper ) / 2 ; lower <= upper ;
                mid = ( lower + upper ) / 2 )
        {
                if ( arr[mid] == num )
                {
                        printf ( "The number is at position %d in the array.", mid ) ;
                        flag = 0 ;
                        break ;
                }
                if ( arr[mid] > num )
                        upper = mid - 1 ;
                else
                        lower = mid + 1 ;
        }

        if ( flag )
                printf ( "Element is not present in the array." ) ;

        getch( ) ;
}
```