VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

DATA STRUCTURE LAB RECORD

Submitted by

Rohan Siwach (1BM19CS132)

Under the Guidance of

Prof. SHEETAL VA
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2020 to Jan-2021

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the LAB RECORD carried out by **Rohan Siwach (1BM19CS132)** who is the bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide	Signature of the HOD
Prof. Prof. Sheelal VA	Dr. Umadevi V
Assistant Professor	Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

External Viva

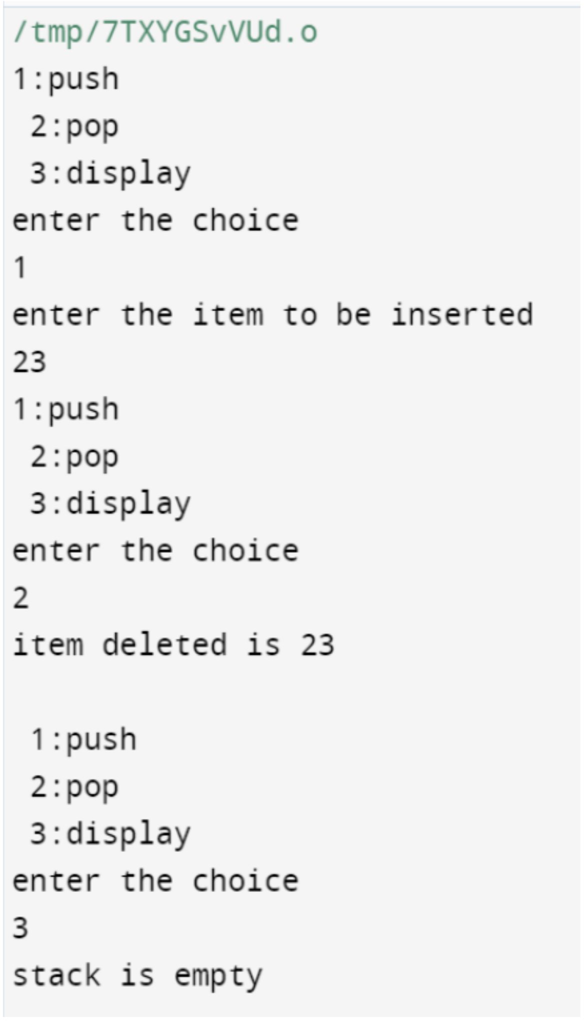
Name of the Examiner	Signature with date
1.______________________________	_________________________
2. ______________________________	__________________________

Lab1}

Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow

```
#include
<stdio.h>
            #define stack_size 5
            int top =-1;
            int s[10];
            int item;
            void push()
            {
                if(top==stack_size-1)
                {
                    printf ("stack overflow\n");
                    return;
                }
                top=top+1;
                s[top]=item;
            }
            int pop()
            {
                if (top==-1)
                return -1;
                else
                return s[top--];
            }
            void display()
            {
                int i;
                if (top==-1)
                {
                printf("stack is empty\n");
                return;
                }
                printf ("contents of the stack\n");
                for(i=top;i>=0;i--)
                {
                printf("%d\n", s[1]);
                }
            }
            int main ()
            {
```

```
    int item_deleted ;
    int choice;
    for(;;)
    {
        printf("\n 1:push\n 2:pop\n 3:display\n");
        printf ("enter the choice \n");
        scanf ("%d", &choice);
        switch (choice)
        {
            case 1: printf("enter the item to be inserted\n");
                   scanf("%d" ,&item);
                   push();
                   break;
            case 2: item_deleted=pop();
                   if(item_deleted==-1)
                    printf("stack is empty\n");
                   else
                   printf("item deleted is %d\n",item_deleted);
                   break;
            case 3: display();
                   break;
            default: exit(0);
        }
    }
    return 0;
}
```



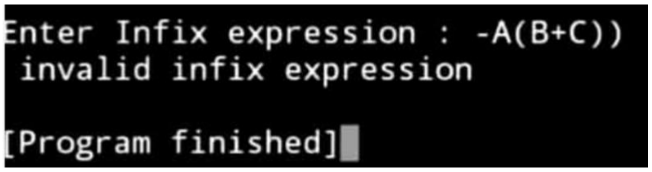
```
/tmp/7TXYGSvVUd.o
1:push
 2:pop
 3:display
enter the choice
1
enter the item to be inserted
23
1:push
 2:pop
 3:display
enter the choice
2
item deleted is 23

 1:push
 2:pop
 3:display
enter the choice
3
stack is empty
```

Lab2}

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
 #include<stdio.h>
                    #include<stdlib.h>
                    #include<ctype.h>
                    #include<string.h>
                    #define SIZE 100
                    char stack[SIZE];
                    int top = -1;
                    void push(char item)
                    {
                           if(top >= SIZE-1)
                           {
                                  printf("\nStack Overflow.");
	}
	else
	{
		top = top+1;
		stack[top] = item;
	}
}
char pop()
{
	char item ;

	if(top <0)
	{
		printf(" invalid infix expression");
		getchar();
		exit(1);
	}
	else
	{
		item = stack[top];
		top = top-1;
		return(item);
	}
}
int is_operator(char symbol)
{
	if(symbol == '^' || symbol == '*' || symbol == '/' || symbol ==
'+' || symbol =='-')
	{
		return 1;
	}
	else
	{
	return 0;
	}
}
int precedence(char symbol)
{
	if(symbol == '^')
	{
		return(3);
	}
	else if(symbol == '*' || symbol == '/')
	{
		return(2);
	}
	else if(symbol == '+' || symbol == '-')
	{
		return(1);
	}
	else
	{
		return(0);
	}
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
	int i, j;
	char item;
	char x;
	push('(');
	strcat(infix_exp,")");

	i=0;
	j=0;
	item=infix_exp[i];

	while(item != '\0')
	{
		if(item == '(')
		{
			push(item);
		}
		else if( isdigit(item) || isalpha(item))
		{
			postfix_exp[j] = item;
			j++;
		}
		else if(is_operator(item) == 1)
		{
			x=pop();
			while(is_operator(x) == 1 && precedence(x)>=
precedence(item))
			{
				postfix_exp[j] = x;
```

```
                                j++;
                                x = pop();
                        }
                        push(x);
        push(item);
                }
                else if(item == ')')
                {
                        x = pop();
                        while(x != '(')
                        {
                                postfix_exp[j] = x;
                                j++;
                                x = pop();
                        }
                }
                else
                {
                        printf("\nInvalid infix Expression.\n");
                        getchar();
                        exit(1);
                }
                i++;
       item = infix_exp[i];
        }
        if(top>0)
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        if(top>0)
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        postfix_exp[j] = '\0';
}
int main()
{
        char infix[SIZE], postfix[SIZE];
        printf("\nEnter Infix expression : ");
        gets(infix);
        InfixToPostfix(infix,postfix);
        printf("Postfix Expression: ");
        puts(postfix);
        return 0;
}
```

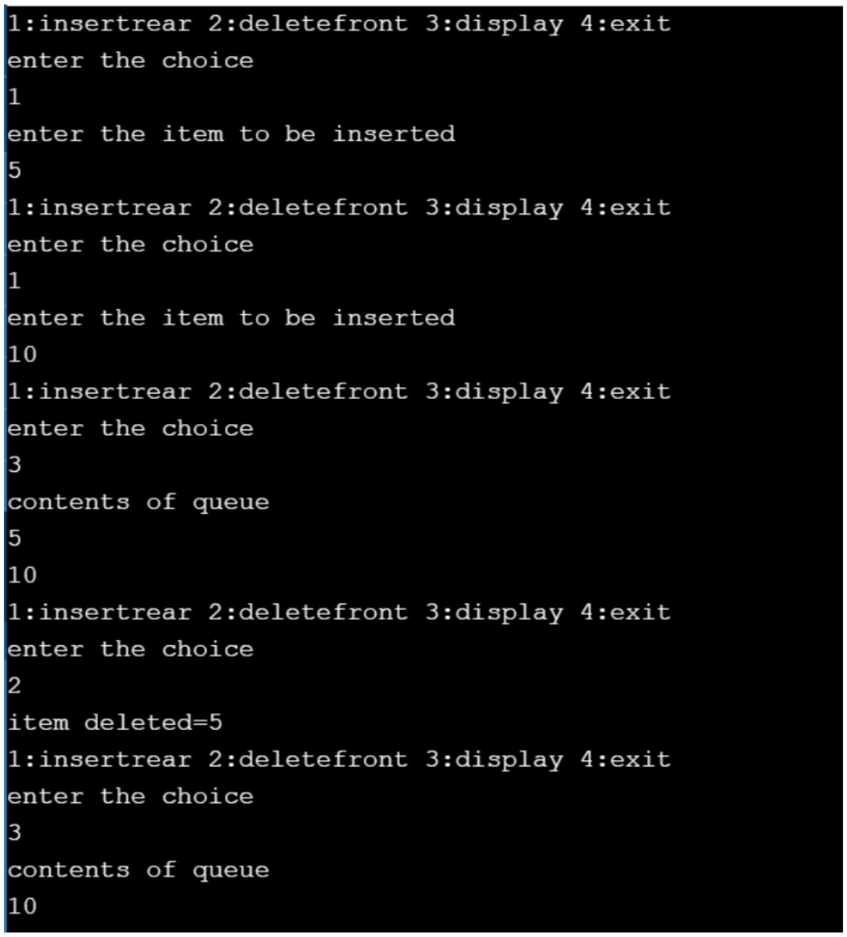


```
Enter Infix expression : -A(B+C))
 invalid infix expression

[Program finished]
```

Lab3}

WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include<stdio.h>
#include<stdlib.h>
#define QUE_SIZE 3
int item,front=0,rear=-1,q[10];
void insertrear()
{if(rear==QUE_SIZE-1)
{
        printf("queue overflow\n");
        return;
}
rear=rear+1;
q[rear]=item;
}int deletefront()
{if (front>rear)
{front=0;
rear=-1;
return -1;
}return q[front++];
}void displayQ()
{int i;
if (front>rear)
{
        printf("queue is empty\n");
        return;
}
printf("contents of queue\n");
for(i=front;i<=rear;i++)
{
       printf("%d\n",q[i]);
}}
int main()
{
       int choice;
       for(;;)
       {
              printf("1:insertrear 2:deletefront 3:display 4:exit\n");
              printf("enter the choice\n");
              scanf("%d",&choice);
              switch(choice)
              {
                     case 1:printf("enter the item to be inserted\n");
                     scanf("%d",&item);
                     insertrear ();
                     break;
                     case 2:item=deletefront();
                     if(item==-1)
                     printf("queue is empty\n");
                     else
                     printf("item deleted=%d\n",item);
                     break;
                     case 3:displayQ();
                     break;
                     default:exit (0);

              }

       }
       }
```



```
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
1
enter the item to be inserted
5
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
1
enter the item to be inserted
10
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
3
contents of queue
5
10
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
2
item deleted=5
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
3
contents of queue
10
```

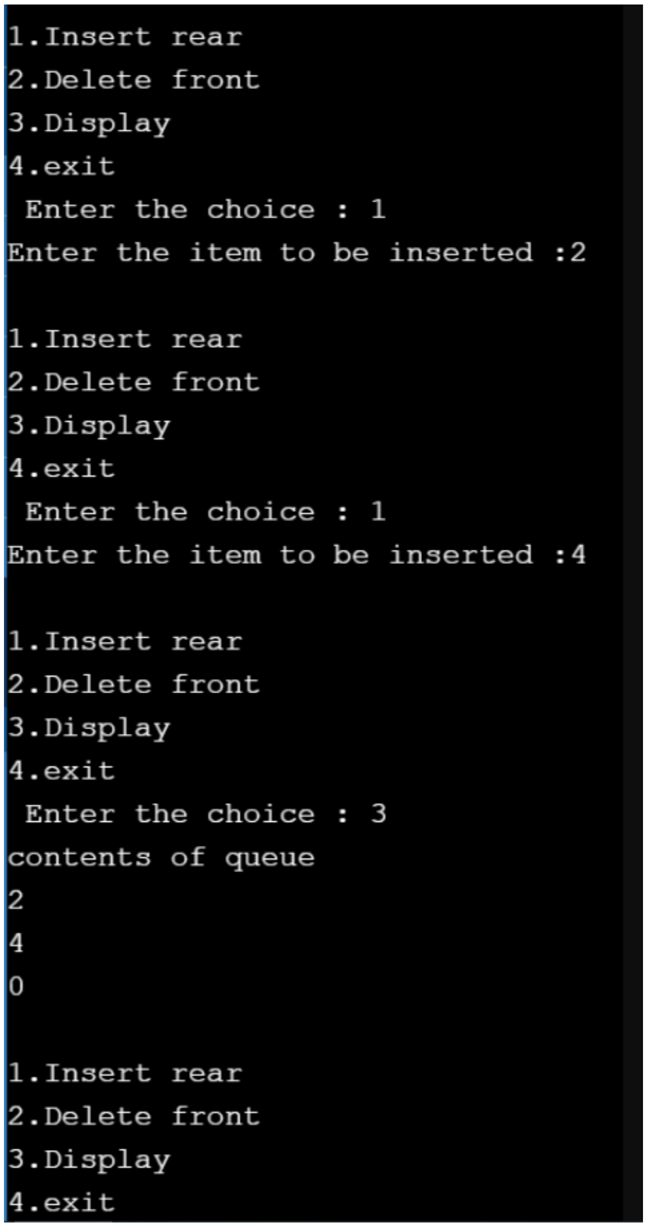
Lab4}

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations. a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

```
 #include<stdio.h>
                    #include<stdlib.h>
                    #define que_size 3
                    int item,front=0,rear=-1,q[que_size],count=0;
                    void insertrear()
                    {
                           if(count==que_size)
                           {
                                  printf("queue overflow");
                                  return;
                           }
                           rear=(rear+1)%que_size;
                           q[rear]=item;
                           count++;
```

```
}
int deletefront()
{
        if(count==0) return -1;
        item = q[front];
        front=(front+1)%que_size;
        count=count-1;
        return item;
}
void displayq()
{
        int i,f;
        if(count==0)
        {
                printf("queue is empty");
                return;
        }
        f=front;
        printf("contents of queue \n");
        for(i=0;i<=count;i++)
        {
                printf("%d\n",q[f]);
                f=(f+1)%que_size;
        }
}
void main()
{
        int choice;
        for(;;)
        {
                printf("\n1.Insert rear \n2.Delete front \n3.Display
\n4.exit \n ");
                printf("Enter the choice : ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter the item to be inserted :");
                               scanf("%d",&item);
                               insertrear();
                               break;
                        case 2:item=deletefront();
                                  if(item==-1)
                                  printf("queue is empty\n");
```

```
                                    else
                                    printf("item deleted is %d \n",item);
                                    break;
                            case 3:displayq();
                                    break;
                            default:exit(0);
                    }
            }
            getch();
    }
```



```
1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :2

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :4

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 3
contents of queue
2
4
0

1.Insert rear
2.Delete front
3.Display
4.exit
```

Lab5}

WAP to Implement Singly Linked List with following operations a) a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

Lab6}

WAP to Implement Singly Linked List with following operations a) a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

Merged program:

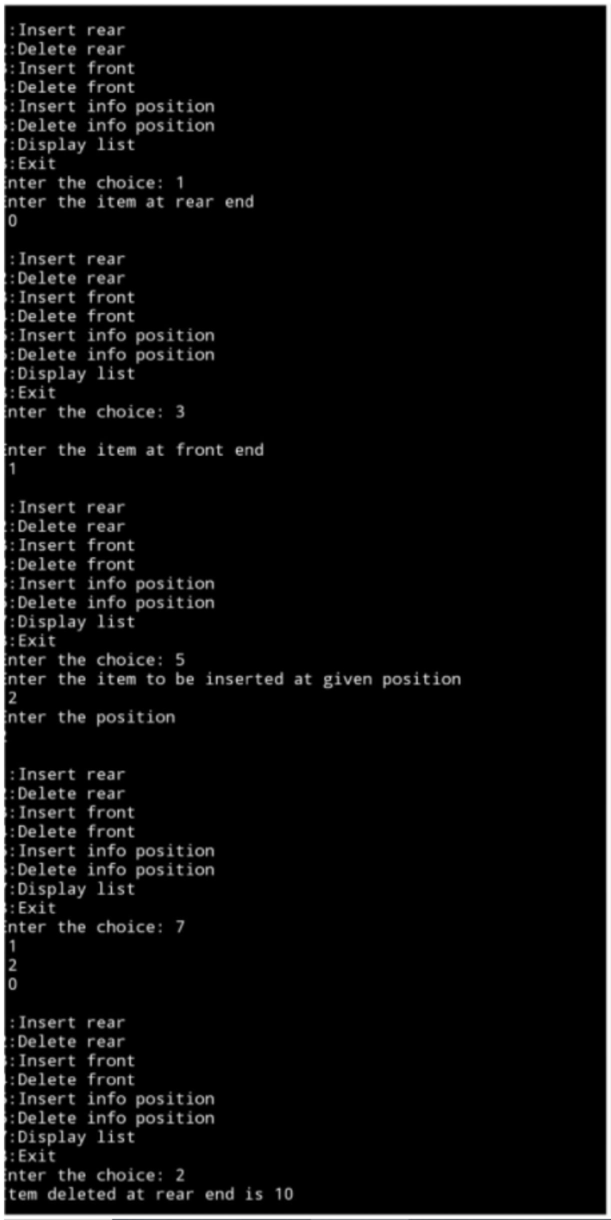
```
 #include<stdio.h>
                    #include<stdlib.h>
                    struct node{
                    int info;
                    struct node *link;
                    };
                    typedef struct node *NODE;
                    NODE getnode(){
                    NODE x;
                    x=(NODE)malloc(sizeof(struct node));
                    if(x==NULL){
                    printf("Memory full\n");
                    exit(0);
                    }
                    return x;
                    }
                    void freenode(NODE x){
                    free(x);
                    }
                    NODE insert_front(NODE first,int item){
                    NODE temp;
                    temp=getnode();
                    temp->info=item;
                    temp->link=NULL;
                    if(first==NULL)
                    return temp;
                    temp->link=first;
                    first=temp;
                    return first;
                    }
                    NODE delete_front(NODE first){
                    NODE temp;
                    if(first==NULL){
```

```
printf("List is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("Item deleted at front end is %d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item){
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
cur=first;
while(cur->link!=NULL)
cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first){
NODE cur,prev;
if(first==NULL){
printf("List is empty cannot delete\n");
return first;
}
if(first->link==NULL){
printf("Item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL){
prev=cur;
cur=cur->link;
}
printf("Item deleted at rear end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
```

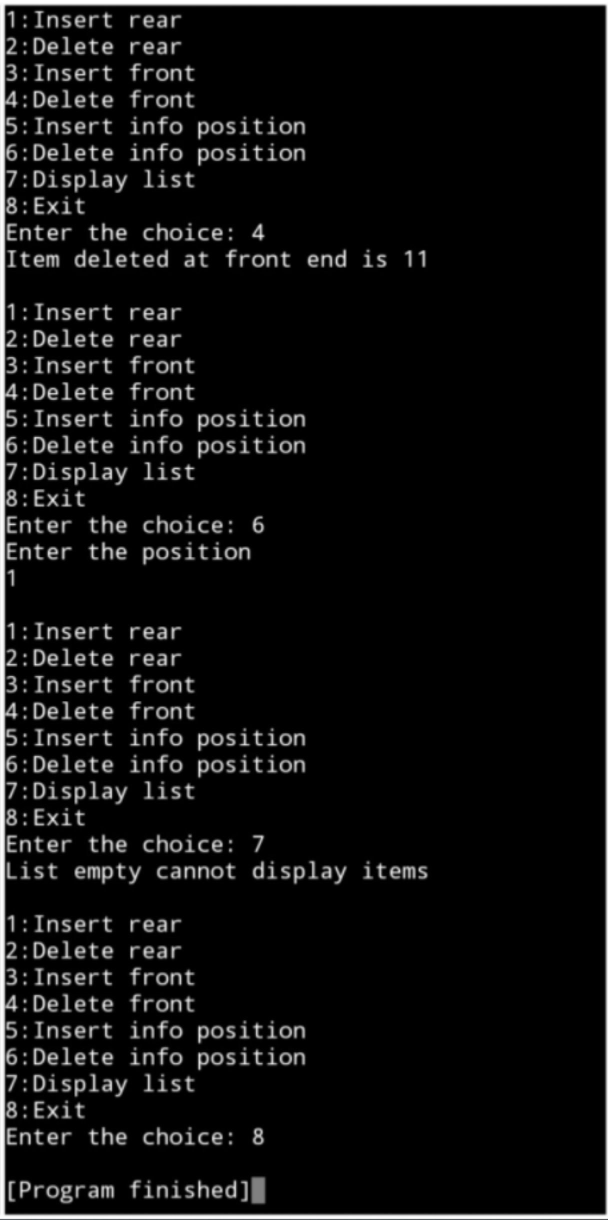
```
}
NODE insert_pos(int item,int pos,NODE first){
NODE temp,cur,prev;
int count;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL&&pos==1){
return temp;
}
if(first==NULL){
printf("Invalid position\n");
return first;
}
if(pos==1){
temp->link=first;
first=temp;
return temp;
}
count=1;
prev=NULL;
cur=first;
while(cur!=NULL&&count!=pos){
prev=cur;
cur=cur->link;
count++;
}
if(
count==pos){
prev->link=temp;
temp->link=cur;
return first;
}
printf("Invalid position\n");
return first;
}
NODE delete_pos(int pos,NODE first){
NODE cur;
NODE prev;
int count,flag=0;
if(first==NULL || pos<0){
printf("Invalid position\n");
return NULL;
```

```
}
if(pos==1){
cur=first;
first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
count=1;
while(cur!=NULL){
if(count==pos){
flag=1;
break;
}
count++;
prev=cur;
cur=cur->link;
}
if(flag==0){
printf("Invalid position\n");
return first;
}
printf("Item deleted at given position is %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}
void display(NODE first){
NODE temp;
if(first==NULL)
printf("List empty cannot display items\n");
for(temp=first;temp!=NULL;temp=temp->link){
printf("%d\n",temp->info);
}
}
void main()
{
int item,choice,key,pos;
int count=0;
NODE first=NULL;
for(;;){
```

```
printf("\n1:Insert rear\n2:Delete rear\n3:Insert front\n4:Delete
front\n5:Insert info position\n6:Delete info position\n7:Display
list\n8:Exit\n");
printf("Enter the choice: ");
scanf("%d",&choice);
switch(choice){
case 1:printf("Enter the item at rear end\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 2:first=delete_rear(first);
break;
case 3:printf("\nEnter the item at front end\n");
scanf("%d",&item);
first=insert_front(first,item);
break;
case 4:first=delete_front(first);
break;
case 5:printf("Enter the item to be inserted at given position\n");
scanf("%d",&item);
printf("Enter the position\n");
scanf("%d",&pos);
first=insert_pos(item,pos,first);
break;
case 6:printf("Enter the position\n");
scanf("%d",&pos);
first=delete_pos(pos,first);
break;
case 7:display(first);
break;
default:exit(0);
break;
}
}
}
```



```
:Insert rear
:Delete rear
:Insert front
:Delete front
:Insert info position
:Delete info position
:Display list
:Exit
nter the choice: 1
nter the item at rear end
0

:Insert rear
:Delete rear
:Insert front
:Delete front
:Insert info position
:Delete info position
:Display list
:Exit
nter the choice: 3

nter the item at front end
1

:Insert rear
:Delete rear
:Insert front
:Delete front
:Insert info position
:Delete info position
:Display list
:Exit
nter the choice: 5
nter the item to be inserted at given position
2
nter the position


:Insert rear
:Delete rear
:Insert front
:Delete front
:Insert info position
:Delete info position
:Display list
:Exit
nter the choice: 7
1
2
0

:Insert rear
:Delete rear
:Insert front
:Delete front
:Insert info position
:Delete info position
:Display list
:Exit
nter the choice: 2
tem deleted at rear end is 10
```



```
1:Insert rear
2:Delete rear
3:Insert front
4:Delete front
5:Insert info position
6:Delete info position
7:Display list
8:Exit
Enter the choice: 4
Item deleted at front end is 11

1:Insert rear
2:Delete rear
3:Insert front
4:Delete front
5:Insert info position
6:Delete info position
7:Display list
8:Exit
Enter the choice: 6
Enter the position
1

1:Insert rear
2:Delete rear
3:Insert front
4:Delete front
5:Insert info position
6:Delete info position
7:Display list
8:Exit
Enter the choice: 7
List empty cannot display items

1:Insert rear
2:Delete rear
3:Insert front
4:Delete front
5:Insert info position
6:Delete info position
7:Display list
8:Exit
Enter the choice: 8

[Program finished]
```

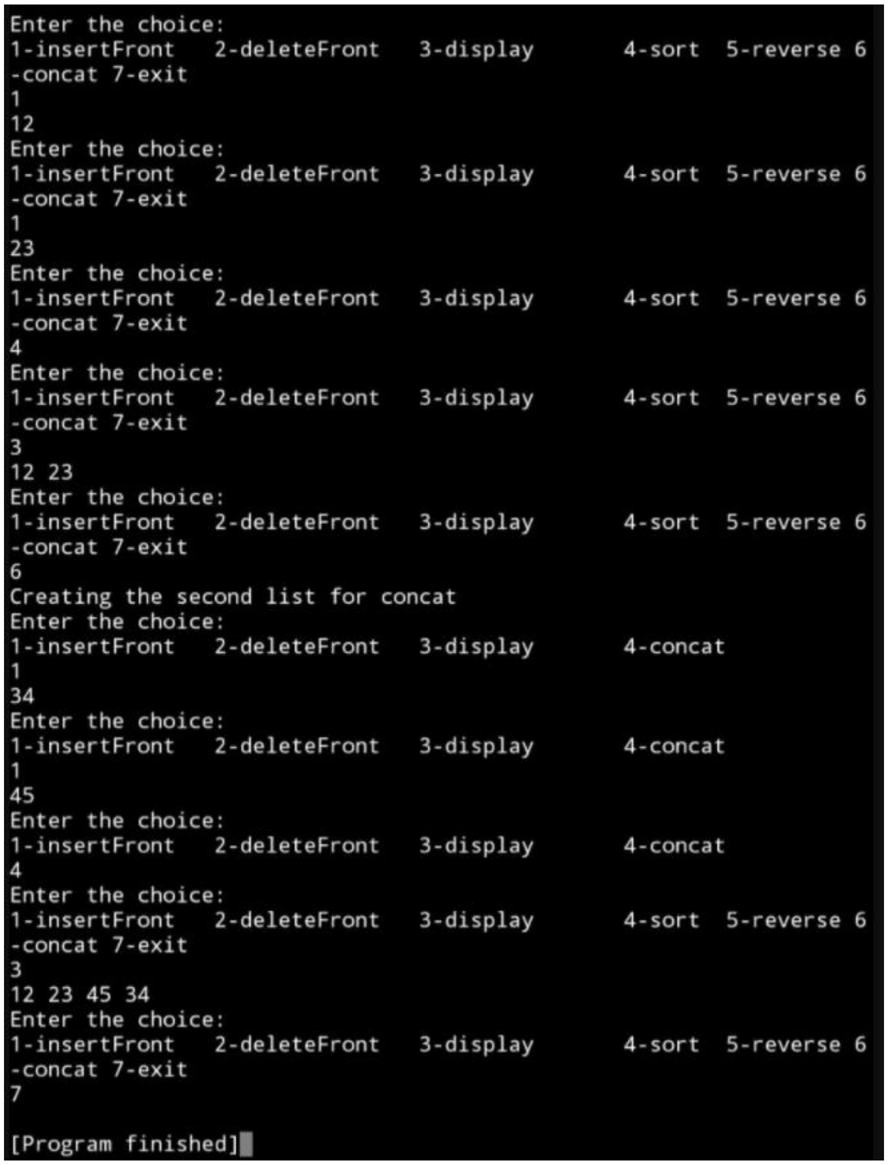
Lab7}

WAP Implement Single Link List with following operations a) a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

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

                    struct node{
                           int num;
                           struct node *next;
                    };
```

```c
typedef struct node *NODE;

NODE getNode(){
	NODE temp = (NODE)malloc(sizeof(struct node));
	if(temp == NULL){
		return NULL;
	}
	return temp;
}

void freeNode(NODE temp){
	free(temp);
}

NODE insertFront(NODE first){
	NODE temp;
	temp = getNode();
	int num;
	scanf("%d",&num);
	temp->num = num;
	temp->next = NULL;
	if(first==NULL){
		return temp;
	}
	temp->next = first;
	first = temp;
	return first;
}

NODE deleteFront(NODE first){
	NODE temp;
	if(first==NULL){
		printf("List is empty\n");
		return NULL;
	}
	if(first->next == NULL){
		printf("Deleted element = %d\n",first->num);
		freeNode(first);
		return NULL;
	}
	temp = first;
	temp = temp->next;
	printf("Deleted elements = %d\n",first->num);
```

```
        freeNode(first);
        return temp;
}

NODE sort(NODE first){
        NODE curr,temp;
        if(first==NULL){
                return NULL;
        }
        curr = first;
        while(curr!=NULL){
                temp = curr->next;
                while(temp!=NULL){
                        if(temp->num<curr->num){
                                int num = curr->num;
                                curr->num=temp->num;
                                temp->num = num;
                        }
                        temp = temp->next;
                }
                curr = curr->next;
        }
        return first;
}

void display(NODE first){
        NODE curr;
        if(first==NULL){
                printf("List is empty\n");
                return;
        }
        curr = first;
        while(curr!=NULL){
                printf("%d ",curr->num);
                curr=curr->next;
        }
        printf("\n");
}

NODE reverse(NODE first){
        NODE curr=NULL;
        NODE temp = getNode();
        while(first!=NULL){
                temp = first;
                first = first->next;
                temp->next = curr;
                curr = temp;
                //printf("%d ",first->num);
        }
        return temp;
}

NODE concat(NODE first){
        NODE sec = NULL;
        int chq;
        while(1){
                printf("Enter the choice:\n1-insertFront\t2-
deleteFront\t3-display\t4-concat\n");
                scanf("%d",&chq);
                if(chq==4){
                        break;
                }
                switch(chq){
                        case 1:
                                sec = insertFront(sec);
                                break;
                        case 2:
                                sec = deleteFront(sec);
                                break;
                        case 3:
                                display(sec);
                                break;
                }
        }
        NODE curr;
        if(first==NULL){
                return sec;
        }
        if(sec==NULL){
                return first;
        }
        curr = first;
        while(curr->next!=NULL){
                curr = curr->next;
        }
        curr->next = sec;
        return first;
}

int main(){
        int chq;
        NODE first = NULL;
        while(1){
                printf("Enter the choice:\n1-insertFront\t2-
deleteFront\t3-display\t4-sort\t5-reverse\t6-concat\t7-exit\n");
                scanf("%d",&chq);
                switch(chq){
                        case 1:
                                first = insertFront(first);
                                break;
                        case 2:
                                first = deleteFront(first);
                                break;
                        case 3:
                                display(first);
                                break;
                        case 4:
                                first = sort(first);
                                break;
                        case 5:
                                first = reverse(first);
                                break;
                        case 6:
                                printf("Creating the second list for
concat\n");
                                concat(first);
                                break;
                        case 7:
                                return 0;

                }
        }
}
```



```
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
1
12
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
1
23
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
4
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
3
12 23
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
6
Creating the second list for concat
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-concat
1
34
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-concat
1
45
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-concat
4
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
3
12 23 45 34
Enter the choice:
1-insertFront   2-deleteFront   3-display       4-sort  5-reverse 6
-concat 7-exit
7

[Program finished]
```

Lab8}

WAP to implement Stack & Queues using Linked Representation

Implement queue

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
```

```
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}

NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at rear-end\n");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
  case 2:first=delete_front(first);
            break;
  case 3:display(first);
            break;
 default:exit(0);
            break;
 }
}
getch();
}
```

Implement stack

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
```

```
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("stack is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("stack empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
```

```
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front-end\n");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
  case 2:first=delete_front(first);
            break;
  case 3:display(first);
            break;
 default:exit(0);
            break;
 }
}
```

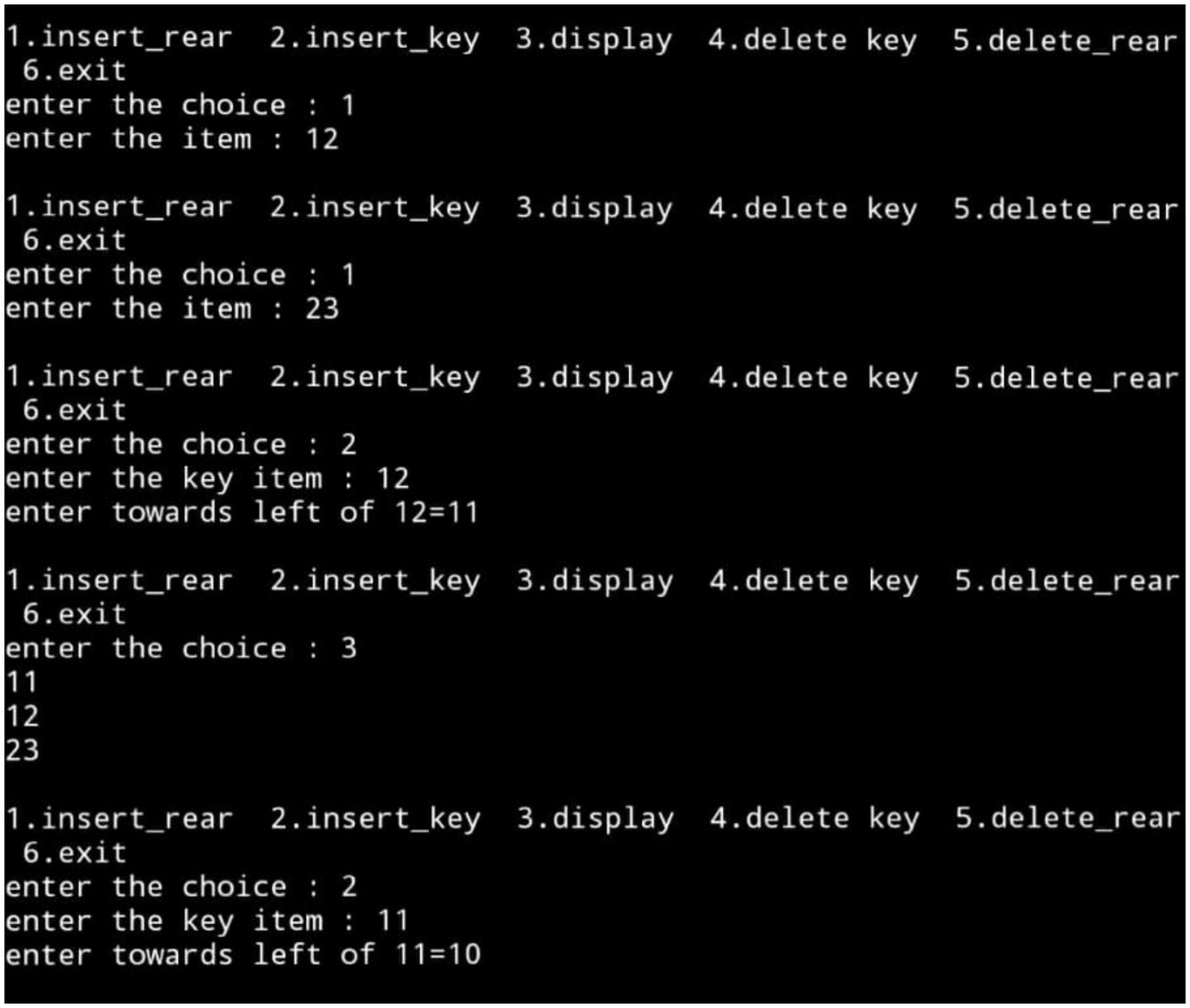
Lab9}

WAP Implement doubly link list with primitive operations a) a) Create a doubly linked list. b) Insert a new node to the left of the node. b) c) Delete the node based on a specific value. c) Display the contents of the list

```
 #include<stdio.h>
                    #include<stdlib.h>
                    struct node
                     {
                      int info;
                      struct node *rlink;
                      struct node *llink;
                     };
                    typedef struct node *NODE;
                    NODE getnode()
                    {
```

```
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE head,int item)
{
NODE temp,cur;
temp=getnode();
temp->rlink=NULL;
temp->llink=NULL;
temp->info=item;
cur=head->llink;
temp->llink=cur;
cur->rlink=temp;
head->llink=temp;
temp->rlink=head;
head->info=head->info+1;
return head;
}
NODE insert_leftpos(int item,NODE head)
{
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info)break;
cur=cur->rlink;
}
if(cur==head)
```

```
{
 printf("key not found\n");
 return head;
 }
 prev=cur->llink;
 printf("enter towards left of %d=",item);
 temp=getnode();
 scanf("%d",&temp->info);
 prev->rlink=temp;
 temp->llink=prev;
 cur->llink=temp;
 temp->rlink=cur;
 return head;
}
NODE delete_all_key(int item,NODE head)
{
NODE prev,cur,next;
int count;
   if(head->rlink==head)
    {
     printf("LE");
     return head;
     }
count=0;
cur=head->rlink;
while(cur!=head)
{
  if(item!=cur->info)
  cur=cur->rlink;
  else
 {
  count++;
  prev=cur->llink;
  next=cur->rlink;
  prev->rlink=next;
  next->llink=prev;
  freenode(cur);
  cur=next;
 }
}
if(count==0)
  printf("key not found");
  else
 printf("key found at %d positions and are deleted\n", count);

return head;
}
NODE ddelete_rear(NODE head)
{
NODE cur,prev;
if(head->rlink==head)
{
printf("list is empty\n");
return head;
}
cur=head->llink;
prev=cur->llink;
head->llink=prev;
prev->rlink=head;
printf("the node deleted is %d \n",cur->info);
freenode(cur);
return head;
}
void display(NODE head)
{
NODE temp;
if(head->rlink==head)
{
printf("list empty\n");
return;
}
for(temp=head->rlink;temp!=head;temp=temp->rlink)
printf("%d\n",temp->info);
}
void main()
{
int item,choice,key;
NODE head,tem;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
printf("\n1.insert_rear  2.insert_key  3.display  4.delete key
5.delete_rear 6.exit\n");
printf("enter the choice : ");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item : ");
                scanf("%d",&item);
                head=insert_rear(head,item);
                break;
  case 2:printf("enter the key item : ");
                scanf("%d",&item);
                head=insert_leftpos(item,head);
                break;
  case 3:display(head);
                break;
  case 4:printf("enter the key item : ");
                scanf("%d",&item);
                head=delete_all_key(item,head);
                break;
  case 5:head=ddelete_rear(head);
                      break;
  default:exit(0);
                 break;
  }
 }
}
```



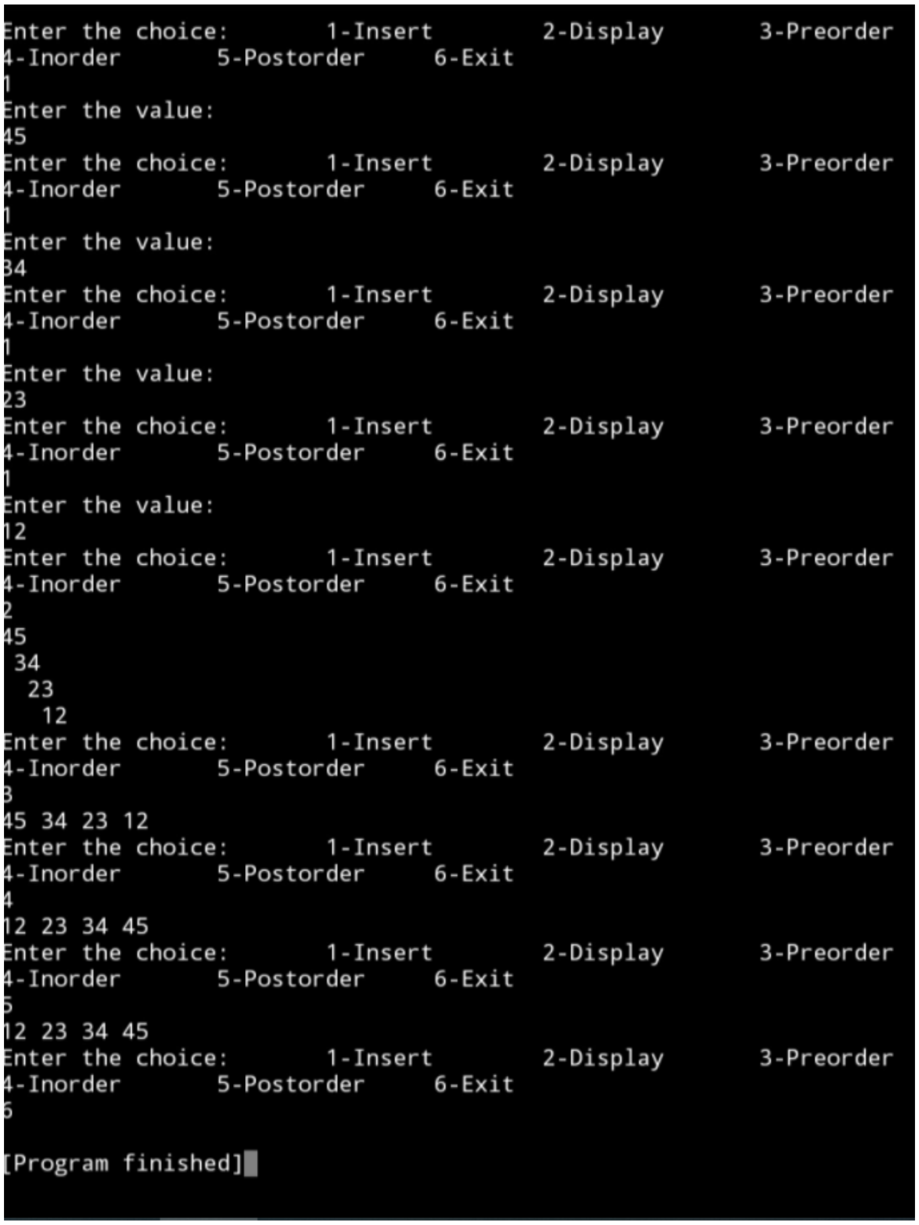
```
1.insert_rear  2.insert_key  3.display  4.delete key  5.delete_rear
 6.exit
enter the choice : 1
enter the item : 12

1.insert_rear  2.insert_key  3.display  4.delete key  5.delete_rear
 6.exit
enter the choice : 1
enter the item : 23

1.insert_rear  2.insert_key  3.display  4.delete key  5.delete_rear
 6.exit
enter the choice : 2
enter the key item : 12
enter towards left of 12=11

1.insert_rear  2.insert_key  3.display  4.delete key  5.delete_rear
 6.exit
enter the choice : 3
11
12
23

1.insert_rear  2.insert_key  3.display  4.delete key  5.delete_rear
 6.exit
enter the choice : 2
enter the key item : 11
enter towards left of 11=10
```

Lab10}

Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

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

                    struct node{
                           struct node *left;
                           int value;
                           struct node *right;
                    };

                    typedef struct node *NODE;

                    NODE getNode(){
                           NODE temp;
```

```
        temp = (NODE)malloc(sizeof(struct node));
        return temp;
}

NODE insert(NODE root){
        int value;
        NODE temp,curr,prev;
        temp = getNode();
        printf("Enter the value:\n");
        scanf("%d",&value);
        temp->value = value;
        temp->left = NULL;
        temp->right = NULL;
        if(root==NULL){
                return temp;
        }
        curr = root;
        prev = NULL;
        while(curr!=NULL){
                prev = curr;
                if(value<curr->value){
                        curr = curr->left;
                }else{
                        curr = curr->right;
                }
        }
        if(value<prev->value){
                prev->left = temp;
        }else{
                prev->right = temp;
        }
        return root;
}

void display(NODE root,int i){
        int j;
        if(root!=NULL){
                display(root->right,i+1);
                for(j=0;j<i;j++){
                        printf(" ");
                }
                printf("%d\n",root->value);
                display(root->left,i+1);
        }
}

void preOrder(NODE root){
        if(root==NULL){
                return;
        }
        printf("%d ",root->value);
        preOrder(root->left);
        preOrder(root->right);
}

void inOrder(NODE root){
        if(root == NULL){
                return;
        }
        inOrder(root->left);
        printf("%d ",root->value);
        inOrder(root->right);
}

void postOrder(NODE root){
        if(root == NULL){
                return;
        }
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ",root->value);
}

int main(){
        int chq;NODE root = NULL;
        while(1){
                printf("Enter the choice:\t1-Insert\t2-Display\t3-
Preorder\t 4-Inorder\t5-Postorder\t6-Exit\n");
                scanf("%d",&chq);
                switch(chq){
                        case 1:
                                root = insert(root);
                                break;
                        case 2:
                                if(root==NULL){
                                        printf("Tree is empty\n");
                                }else{
                                        display(root,0);
                                }
                                break;
                        case 3:
                                if(root==NULL){
                                        printf("Tree is empty\n");
                                }else{
                                        preOrder(root);
                                        printf("\n");
                                }
                                break;
                        case 4:
                                if(root==NULL){
                                        printf("Tree is empty\n");
                                }else{
                                        inOrder(root);
                                        printf("\n");
                                }
                                break;
                        case 5:
                                if(root==NULL){
                                        printf("Tree is empty\n");
                                }else{
                                        postOrder(root);
                                        printf("\n");
                                }
                                break;
                        case 6:
                                return 0;
                }
        }
}
```



```
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
1
Enter the value:
45
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
1
Enter the value:
34
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
1
Enter the value:
23
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
1
Enter the value:
12
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
2
45
 34
  23
   12
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
3
45 34 23 12
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
4
12 23 34 45
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
5
12 23 34 45
Enter the choice:       1-Insert        2-Display       3-Preorder
4-Inorder       5-Postorder     6-Exit
6

[Program finished]
```