**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

**LAB REPORT**

**on**

DATA STRUCTURES

***Submitted by***

**ADITYA S HUDDAR(1BM21CS007)**

***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 Oct 2022-Feb 2023**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by

**ADITYA S HUDDAR(1BM21CS007),** who is a bonafide student 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 Visvesvaraya Technological University, Belgaum during the year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab **- (22CS3PCDST)** work prescribed for the said degree.

|  |  |
| --- | --- |
| **Dr. Selva Kumar S** | **Dr. Jyothi S Nayak** |
| Assistant Professor | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |

Index Sheet :

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Program to simulate the working of stack using an array |  |

|  |  |  |
| --- | --- | --- |
| 2 | Program to convert a given infix arithmetic expression to postfix expression |  |
| 3 | Program to simulate the working of a queue of integers using an array |  |
| 4 | Program to simulate the working of a circular queue of integers using an array |  |
| 5 | Program to implement Singly Linked List (Create, Insert and Display functions) |  |
| 6 | Program to Implement Singly Linked List (Create, Delete and Display functions) |  |
| 7 | Program to Implement Single Link List  (Sort, Reverse and Concatenate list functions) |  |
| 8 | Program to implement Stack & Queues using Linked Representation |  |
| 9 | Program to Implement doubly link list |  |
| 10 | Program to Implement a Binary Search Tree  (Create, Traversal and Display functions) |  |

**Course Outcome :**

|  |  |
| --- | --- |
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyze data structure operations for a given problem. |
| CO3 | Design and develop solutions using Data Structure concepts. |

|  |  |
| --- | --- |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures. |

# LAB PROGRAM 1:

Program to simulate the working of stack using an array.

# Program code-C:

#include <stdio.h> #define Stack\_size 5 int top, item, st[10],i; top=-1;

void push()

{

if (top==Stack\_size-1) printf("STACK OVERFLOW\n\n\n"); else

{

top++; st[top]=item;

}

}

int pop()

{

int del\_item; if(top==-1)

printf("STACK UNDERFLOW\n");

else

{

del\_item = st[top]; top--;

return del\_item;

}

}

void display()

{ if(top==-1) printf("Stack empty. There is nothing to display\n");

for(i=0;i<=top;i++) printf(" %d ", st[i]);

}

int main()

{ int op; while(1)

{

printf("\nEnter the operation\n 1.PUSH 2. POP 3. DISPLAY\n"); scanf("%d", &op);

switch(op)

{

case 1: printf("Enter the number : "); scanf("%d", &item);

push();

break;

case 2: pop(); break;

case 3: display(); break;

default: printf("Invalid input\n\n"); break;

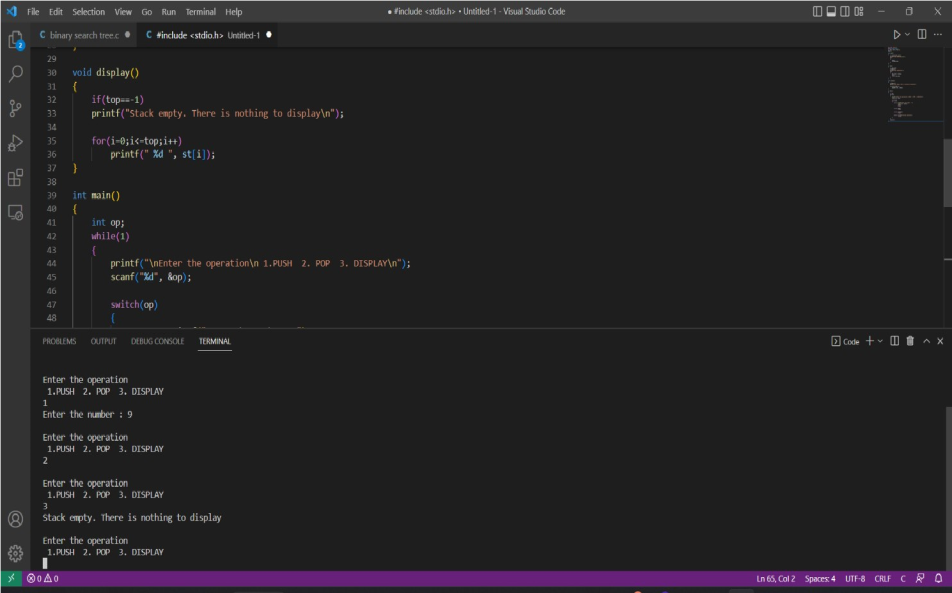
}

}

return 0;

}

# Outputs:



**LAB PROGRAM 2:**

Program to convert a given infix arithmetic expression to postfix expression.

# Program code-C:

#include<stdio.h> #include<string.h>

int top = -1; char s[20]; char infix[20]; char postfix[20];

void inf\_to\_post(); int sp(char); int ip(char); void push(char); char pop();

void main() { printf("enter a valid infix expression\n"); scanf("%s", infix); inf\_to\_post(); printf("The postfix expression is %s", postfix);

}

void push(char item) { s[++top] = item;

}

char pop() { return s[top--]; } int sp(char item) { switch (item) { case '+':

case '-': return 2; case '\*':

case '/': return 4; case '^':

case '$': return 5; case '(': return 0; case '#': return -1; default: return 8;

}

}

int ip(char item) { switch (item) { case '+': case '-': return 1; case '\*': case '/': return 3; case '^': case '$':

return 6; case '(':

return 9; case ')': return 0; default:

return 7;

}

}

void inf\_to\_post() { int i, j = 0;

char symbol;

push('#'); for (i = 0; i < strlen(infix); i++) { symbol = infix[i]; while (sp(s[top]) > ip(symbol)) { postfix[j] = pop();

j++;

}

if (sp(s[top] < ip(symbol))) { push(symbol);

}

if (sp(s[top]) == ip(symbol)) {

pop();

}

}

while (s[top] != '#') { postfix[j] = pop();

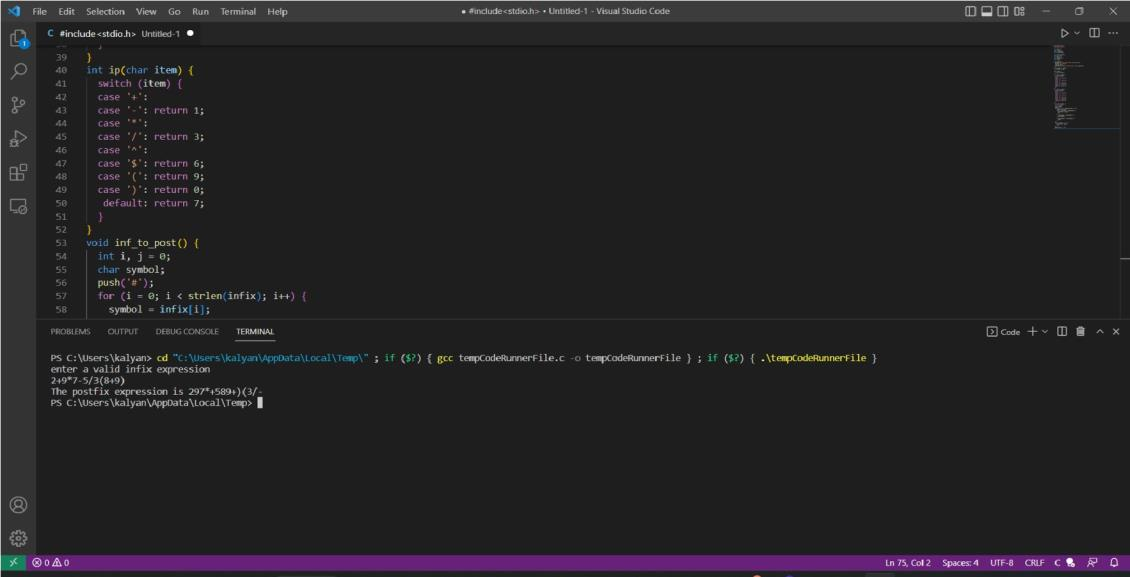
j++;

}

postfix[j] = '\0';

}

# Outputs:



**LAB PROGRAM 3:**

Program to simulate the working of a queue of integers using an array.

# Program code-C:

#include <stdio.h> #define QSIZE 5

void insert\_rear(int q[], int item, int \*r)

{

if(\*r==QSIZE-1)

printf("Queue Overflow\n"); else

{

(\*r)++;

q[\*r]=item;

}

}

int delete\_front(int q[], int \*f, int \*r)

{

if(\*f>\*r)z

printf("Queue Underflow\n"); else{

return q[(\*f)++]; //return(q[(\*f)++]);

}

}

void display(int q[], int \*f, int \*r)

{

int i; if(\*f>\*r)

printf("Queue is empty\n");

else

{

for(i=\*f;i<=\*r;i++) printf("%d",q[i]);

}

}

int main()

{

int op,item,st[10],val; int rear=-1; int front=0; while(1)

{

printf("\nEnter the operation\n 1.Insert 2.Delete 3. Display\n");

scanf("%d", &op);

switch(op)

{

case 1: printf("Enter the number : "); scanf("%d", &item);

insert\_rear(st, item , &rear); break;

case 2: val=delete\_front(st,&front,&rear); printf("The value deleted is :%d",val);

break;

case 3: display(st,&front,&rear); break;

default: printf("Invalid input\n\n"); break;

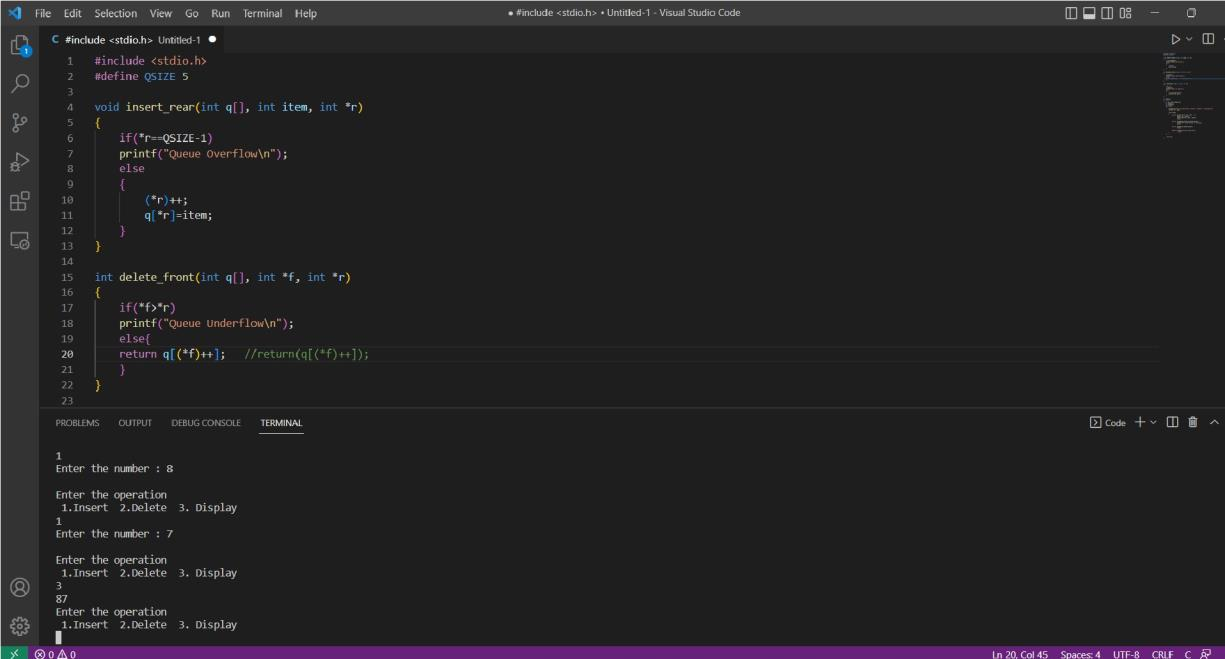
}

}

return 0;

}

# Outputs:



**LAB PROGRAM 4:**

Program to simulate the working of a circular queue of integers using an array.

# Program code-C:

#include<stdio.h>

#include<stdlib.h> #define QSIZE 3 int count=0; void insert\_rear(int q[3],int item,int \*r)

{

if(count==QSIZE) printf("Queue overflow\n");

else

{

\*r=\*r+1;

\*r=(\*r)% QSIZE;

q[\*r]=item; count++;

}

}

int delete\_front(int q[3],int \*f,int \*r)

{

int del\_item; if (count==0) printf("Queue underflow\n");

else

{

del\_item=q[\*f];

\*f=\*f+1;

\*f=((\*f)% QSIZE);

count--; return del\_item;

}

}

void display(int q[3],int \*f)

{ int temp,i; temp=\*f; for(i=0;i<count;i++)

{

printf("%d\t",q[temp]); temp=(temp+1)% QSIZE;

}

}

void main()

{

int q[QSIZE],item,r=-1,f=0,choice,val\_del; while(1)

{

printf("\n Enter your choice\n"); printf("\n1.insert 2.delete 3.display\n"); scanf("%d",&choice); switch(choice)

{

case 1:printf("enter the value to be inserted\n"); scanf("%d",&item); insert\_rear(q,item,&r);

break;

case 2:val\_del=delete\_front(q,&f,&r); printf("Item deleted=%d",val\_del);

break;

case 3:display(q,&f); break;

default:exit(0);

}

}

}

# Outputs:

**LAB PROGRAM 5:**

Program to implement Singly Linked List (Create, Insert and Display functions)

# Program code-C:

#include<stdio.h> #include<conio.h> #include<stdlib.h> struct node

{

int value;

struct node \*next;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE temp; temp=(NODE)malloc(sizeof(struct node)); if (temp==NULL)

{

printf("Memory not allocated\n"); return NULL;

}

return temp;

}

NODE insert\_beg(int item,NODE first)

{

NODE new; new=getnode(); new-

>value=item; new-

>next=NULL; if(first==NULL)

{

}

else

{

}

}

return new;

new->next=first; first=new; return first;

NODE insert\_end(int item,NODE first)

{

NODE new,last; new=getnode(); new->value=item; new->next=NULL;

if (first==NULL)

{

return new;

}

if(first->next==NULL)

{

first->next=new; return first;

}

last=first;

while(last->next!=NULL)

last=last->next; last-

>next=new; return first;

}

NODE insert\_pos(NODE first,int item,int pos)

{

int count=1; int val=item;

NODE new,curr,prev; new=getnode(); new-

>value=item; new->next=NULL;

if(first==NULL && pos==1) return new;

prev=NULL; curr=first;

while(count!=pos && curr!=NULL)

{

prev=curr; curr=curr-

>next; count++;

}

if(count==pos)

{

prev->next=new; new-

>next=curr; return first;

}

if(curr==NULL)

{

printf("position not fount\n"); return first;

}

if(first!=NULL && pos==1) first=insert\_beg(val,first); return first;

}

void display(NODE first)

{

NODE temp; temp=first; while(temp!=NULL)

{

printf("value stored in node=%d\n",temp->value); temp=temp->next;

}

}

void main()

{

NODE first=NULL; int choice,pos,item; while(1)

{

printf("\n1.Insert\_beg 2.Insert\_end 3.Insert\_pos 4.Display\n"); printf("\n enter your choice\n"); scanf("%d",&choice); switch(choice)

{

case 1:printf("\nEnter the value to be inserted at the beginning\n"); scanf("%d",&item); first=insert\_beg(item,first); break;

case 2:printf("\nEnter the value to be inserted at the end\n"); scanf("%d",&item);

first=insert\_end(item,first); break;

case 3: printf("\nEnter the value to be inserted\n");

scanf("%d",&item); printf("\nEnter the position at which item should be inserted \n"); scanf("%d",&pos); first=insert\_pos(first,item,pos); break;

case 4:display(first);

break;

default:exit(0);

}

}

}

# Outputs:

**LAB PROGRAM 6:**

Program to Implement Singly Linked List (Create, Delete and Display functions).

# Program code-C:

#include<stdio.h> #include<conio.h> #include<stdlib.h> struct node {

int value; struct node \* next;

};

typedef struct node \* NODE;

NODE getnode() { NODE temp; temp = (NODE) malloc(sizeof(struct node)); if (temp == NULL) { printf("Memory not allocated\n"); return NULL;

}

return temp;

}

NODE insert\_beg(int item, NODE first) { NODE new;

new = getnode(); new -> value = item; new -> next = NULL; if (first == NULL) { return new; } else {

new -> next = first; first = new; return first;

}

}

NODE insert\_end(int item, NODE first) { NODE new, last; new = getnode(); new -> value = item; new -> next = NULL;

if (first == NULL) { return new;

}

if (first -> next == NULL) { first -> next = new;

return first;

}

last = first;

while (last -> next != NULL) last = last -> next;

last -> next = new; return first;

}

NODE insert\_pos(NODE first, int item, int pos) { int count = 1;

int val = item;

NODE new, curr, prev; new = getnode(); new

-> value = item; new -

> next = NULL;

if (first == NULL && pos == 1) return new; prev = NULL;

curr = first;

while (count != pos && curr != NULL) { prev = curr;

curr = curr -> next; count++;

}

if (count == pos) { prev -> next = new; new -> next = curr;

return first;

}

if (curr == NULL) { printf("position not fount\n");

return first;

}

if (first != NULL && pos == 1) first = insert\_beg(val, first);

return first;

}

NODE delete\_beg(NODE first) { NODE temp;

if (first == NULL) { printf("Cannot delete\n"); return NULL;

}

temp = first;

temp = temp -> next; printf("Item deleted=%d", first -> value);

free(first); return temp;

}

NODE delete\_end(NODE first) { NODE prev, curr;

if (first == NULL) { printf("Cannot delete\n"); return NULL;

}

prev = NULL; curr = first;

while (curr -> next != NULL) { prev = curr;

curr = curr -> next;

}

prev -> next = NULL; printf("Item deleted=%d", curr -> value);

return first;

}

NODE delete\_specific\_value(NODE first, int key) { NODE prev, curr;

if (first == NULL) { printf("Cannot delete\n"); return NULL;

}

curr = first; if (curr -> value == key) { printf("Item deleted=%d", curr -> value);

first = first -> next; free(curr); return first;

}

prev = NULL; curr = first;

while (curr -> value != key && curr != NULL) { prev = curr;

curr = curr -> next;

}

if (curr -> value == key) { prev -> next

= curr -> next; printf("%d=Item deleted", curr -> value);

free(curr); return first;

}

if (curr == NULL) { printf("End of list reached and item not fount\n");

return first;

}

}

void display(NODE first) { NODE temp;

temp = first; while (temp != NULL) { printf("value stored in node=%d\n", temp -> value); temp = temp -> next;

}

}

void main() {

NODE first = NULL; int choice, pos, item;

while (1) {

printf("\n1.Insert\_beg \n2.Insert\_end \n3.Insert\_pos \n4.delete\_beg \n5.delete\_end \ n6.delete\_specific\_value \n7.Display\n");

printf("\n enter your choice\n"); scanf("%d", & choice); switch (choice) { case 1:

printf("\nEnter the value to be inserted at the begining\n"); scanf("%d", & item); first = insert\_beg(item, first);

break;

case 2:

printf("\nEnter the value to be inserted at the end\n"); scanf("%d", & item); first = insert\_end(item, first); break;

case 3:

printf("\nEnter the value to be inserted\n"); scanf("%d", & item); printf("\nEnter the position at which item should be inserted \n"); scanf("%d", & pos); first = insert\_pos(first, item, pos);

break;

case 4:

first = delete\_beg(first);

break;

case 5: first = delete\_end(first); break;

case 6:

printf("\nEnter the value to be deleted\n"); scanf("%d", & item); first = delete\_specific\_value(first, item);

break;

case 7: display(first); break;

default: exit(0);

}

}

}

# Outputs:

**LAB PROGRAM 7:**

Program to Implement Single Link List (Sort, Reverse and Concatenate list functions).

# Program code-C:

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

struct node

{

int value;

struct node \*next;

};

typedef struct node \*NODE; NODE getnode()

{

NODE temp; temp=(NODE)malloc(sizeof(struct node)); if(temp==NULL)

{

printf("Memory not allocated"); return NULL;

}

return temp;

}

int countfun(NODE first)

{

NODE temp=first;int c=0; while(temp!=NULL)

{

c++;

temp=temp->next;

}

return c;

}

NODE insert\_beg(NODE first,int item)

{

NODE new; new=getnode(); new-

>value=item; new-

>next=NULL; if(first==NULL)

{

}

else

{

}

}

return new;

new->next=first; first=new; return first;

void display(NODE first)

{

NODE temp; temp=first; if(first==NULL)

{

printf("List is empty\n");

}

while(temp!=NULL)

{

printf("Value stored in the node=%d\n",temp->value); temp=temp->next;

}

}

NODE sort(NODE first)

{

NODE curr=first; int count=countfun(first);

int temp,i,j; if(first-

>next==NULL)

return first; for(i=0;i<count-1;i++)

{

curr=first; for(j=0;j<count-i- 1;j++)

{

if(curr->value>curr->next->value)

{

temp=curr->value;

curr->value=curr->next->value; curr->next-

>value=temp;

}

curr=curr->next;

}

}

return first;

}

NODE concatenate(NODE first1, NODE first2)

{

NODE temp;

temp=first1; if(first1==NULL && first2==NULL)

{

return NULL;

}

if(first1==NULL)

{

return first2;

}

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=first2;

return first1;

}

NODE reverse(NODE first)

{

NODE prev=NULL;

NODE curr=first;

NODE next=NULL;

while(curr!=NULL)

{

next=curr->next; curr->next=prev; prev=curr; curr=next;

}

first=prev; return prev;

}

int main()

{

int item,c;

int count1=0,count2=0; NODE first1=NULL,first2=NULL; while(1)

{

printf("\n1.Insert at beginning for list1\n2.Insert at beginning for list2\n3.Sort list1\n3.Sort list2\n5.Concatenate(output is stored in list1)\n6.Reverse list1\n7.Reverse list2\ n8.Display list1\n9.Display list2\n\n");

printf("Enter your choice

:"); scanf("%d",&c); switch(c)

{

case 1:printf("Enter the item to be inserted :"); scanf("%d",&item); first1=insert\_beg(first1,item);

break;

case 2:printf("Enter the item to be inserted :"); scanf("%d",&item); first2=insert\_beg(first2,item);

break;

case 3:first1=sort(first1); break;

case 4:first2=sort(first2); break;

case 5:first1=concatenate(first1,first2); break;

case 6:first1=reverse(first1); break;

case 7:first2=reverse(first2); break;

case 8:display(first1);

break;

case 9:display(first2);

break;

default:printf("Invalid choice!!!");

exit(0);

}

}

}

# Outputs:

**LAB PROGRAM 8:**

Program to implement Stack & Queues using Linked Representation.

# Program code-C:

**Stacks:**

#include<stdio.h>

#include<conio.h> #include<stdlib.h> struct node

{

int value;

struct node \*next;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE temp; temp=(NODE)malloc(sizeof(struct node)); if (temp==NULL)

{

printf("Memory not allocated\n"); return NULL;

}

return temp;

}

NODE insert\_beg(int item,NODE first)

{

NODE new; new=getnode(); new-

>value=item; new-

>next=NULL; if(first==NULL)

{

}

else

{

}

}

return new;

new->next=first; first=new; return first;

NODE delete\_beg(NODE first)

{

NODE temp; if(first==NULL)

{

printf("Cannot delete\n"); return NULL;

}

temp=first;

temp=temp->next; printf("Item deleted=%d",first->value);

free(first); return temp;

}

void display(NODE first)

{

NODE temp; temp=first; while(temp!=NULL)

{

printf("value stored in node=%d\n",temp->value); temp=temp->next;

}

}

void main()

{

NODE first=NULL; int choice,pos,item; while(1)

{

printf("\n1.Push \n2.Pop \n3.Display\n"); printf("\n enter your choice\n"); scanf("%d",&choice);

switch(choice)

{

case 1:printf("\nEnter the value to be inserted\n");

scanf("%d",&item); first=insert\_beg(item,first);

break;

case 2:first=delete\_beg(first); break;

case 3:display(first);

break;

default:exit(0);

}

}

}

# Queues:

#include<stdio.h> #include<conio.h> #include<stdlib.h> struct node

{

int value;

struct node \*next;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE temp; temp=(NODE)malloc(sizeof(struct node)); if (temp==NULL)

{

printf("Memory not allocated\n"); return NULL;

}

return temp;

}

NODE insert\_beg(int item,NODE first)

{

NODE new; new=getnode(); new-

>value=item; new-

>next=NULL; if(first==NULL)

{

return new;

}

else

{

}

}

new->next=first; first=new; return first;

NODE delete\_end(NODE first)

{

NODE prev,curr; if(first==NULL)

{

printf("Cannot delete\n");

return NULL;

}

prev=NULL; curr=first;

while(curr->next!=NULL)

{

prev=curr; curr=curr-

>next;

}

prev->next=NULL; printf("Item deleted=%d",curr->value);

return first;

}

void display(NODE first)

{

NODE temp; temp=first; while(temp!=NULL)

{

printf("value stored in node=%d\n",temp->value); temp=temp->next;

}

}

void main()

{

NODE first=NULL; int choice,pos,item; while(1)

{

printf("\n1.Insert \n2.delete \n3.Display\n"); printf("\n enter your choice\n"); scanf("%d",&choice); switch(choice)

{

case 1:printf("\nEnter the value to be inserted\n");

scanf("%d",&item); first=insert\_beg(item,first);

break;

case 2:first=delete\_end(first);

break;

case 3:display(first);

break;

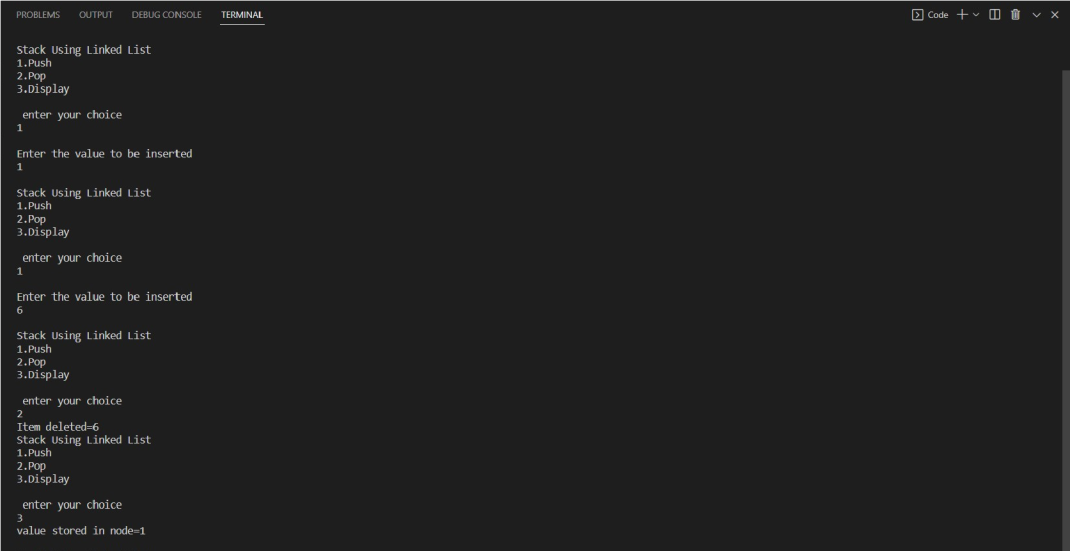
default:exit(0);

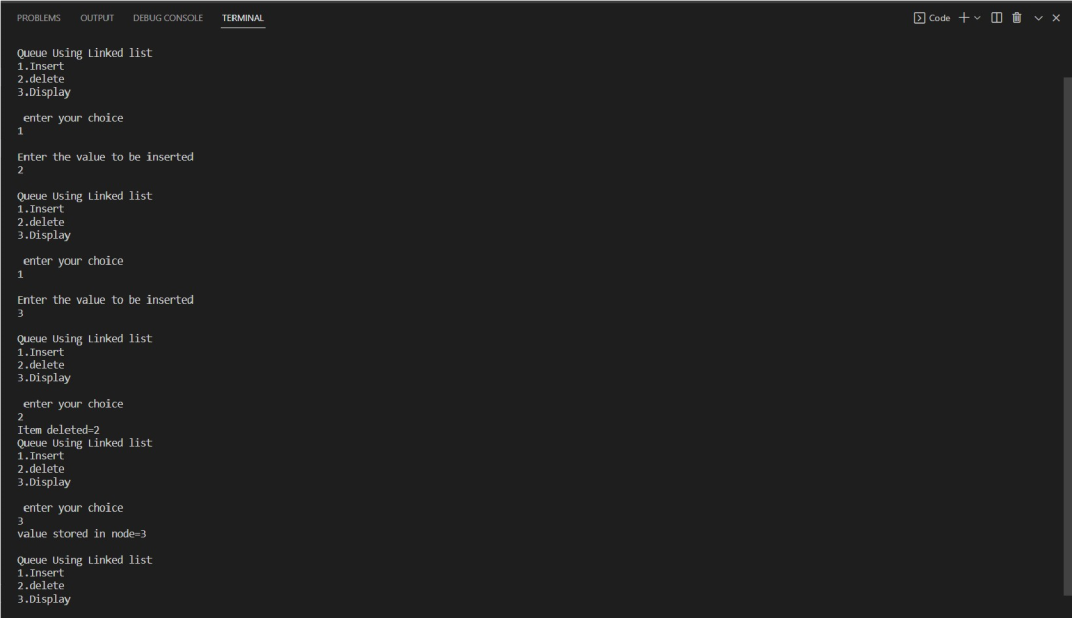
}

}

}

# Outputs:





**LAB PROGRAM 9:**

Program to Implement doubly link list.

# Program code-C:

#include<stdio.h>

#include<conio.h> #include<stdlib.h> struct node

{

int value; struct node \*next;

struct node \*prev;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE temp; temp=(NODE)malloc(sizeof(struct node)); if(temp==NULL)

{

printf("Memory not allocated\n");

}

return temp;

}

NODE insert\_beg(NODE first,int item)

{

NODE new; new=getnode(); new-

>value=item; new-

>prev=NULL; new-

>next=NULL; if(first==NULL)

{

return new;

}

new->next=first; first-

>prev=new; return new;

}

NODE insert\_left(NODE first,int key,int item)

{

NODE temp,new; new=getnode(); new-

>value=item; new->prev=NULL; new->next=NULL; if(first==NULL)

{

printf("List is empty"); return NULL;

}

if(first->next==NULL && first->value!=key)

{

printf("key not found cant insert!!!");

return first;

}

if(first->next==NULL && first->value==key)

{

first=insert\_beg(first,new->value);

}

temp=first; while(temp->value!=key && temp-

>next!=NULL)

{

temp=temp->next;

}

if(temp->value==key)

{

new->next=temp; new->prev=temp-

>prev; (temp->prev)->next=new; temp-

>prev=new; return first;

}

if(temp->value!=key)

{

printf("value not found\n"); return first;

}

}

NODE delete\_specific(NODE first,int key)

{

NODE curr,temp; if(first==NULL)

{

printf("Linkedlist is empty\n"); return NULL;

}

if(first->next==NULL && first->value==key)

{ free(first);

return NULL;

}

if(first->next==NULL && first->value!=key)

{

printf("element not found\n"); return first;

}

if(first->value==key)

{

(first->next)->prev=NULL; temp=first-

>next; free(first); return temp;

}

while(curr!=NULL)

{

if(curr->value==key) break;

curr=curr->next;

}

if(curr==NULL)

{

printf("Element not found\n");

}

(curr->prev)->next=curr->next; if(curr-

>next!=NULL)

{

(curr->next)->prev=curr->prev;

}

}

void display(NODE first)

{

NODE temp; if(first==NULL)

{

printf("List is empty\n");

}

temp=first; while(temp!=NULL)

{

printf("%d\n",temp->value); temp=temp-

>next;

}

}

void main()

{

NODE first=NULL; int choice,key,item; while(1)

{

printf("\n1.Insert\_beg 2.Insert\_left 3.Delete\_specific 4.Display\n"); printf("\n enter your choice"); scanf("%d",&choice); switch(choice)

{

case 1:printf("\nEnter the value to be inserted at the begining\n"); scanf("%d",&item); first=insert\_beg(first,item); break;

break;

case 2:printf("\nEnter the value to be inserted at the left\n"); scanf("%d",&item); printf("\nEnter the key\n"); scanf("%d",&key);

first=insert\_left(first,key,item);

case 3:printf("\nEnter the value to be deleted\n"); scanf("%d",&key); first=delete\_specific(first,key);

break;

case 4:display(first); break;

default:exit(0);

}

}

# Outputs:

**LAB PROGRAM 10:**

Program to Implement a Binary Search Tree (Create, Traversal and Display functions).

**Program code-C:** #include <stdio.h> #include <stdlib.h>

struct node

{ int data; struct node \*left; struct node \*right;

};

struct node \*insert(struct node \*node, int data)

{

if (node == NULL)

{

struct node \*temp = (struct node \*)malloc(sizeof(struct node)); temp->data = data; temp->left = temp->right = NULL;

return temp;

}

if (data < node->data)

node->left = insert(node->left, data); else if (data > node->data) node-

>right = insert(node->right, data);

return node;

}

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);

}

}

int main()

{

struct node \*root = NULL; int n, i, element;

printf("Enter the number of elements to be inserted: "); scanf("%d", &n); printf("Enter %d elements: ", n);

for (i = 0; i < n; i++)

{

scanf("%d", &element); root = insert(root, element);

}

printf("In-order traversal: "); inorder(root); printf("\nPre-

order traversal: "); preorder(root); printf("\nPost- order traversal: "); postorder(root);

return 0;

}

# Outputs:

P a g e | **61**