LAB REPORT

NAME : MITHUN R K

USN : 1BM19CS087

SUBJECT : DATA STRUCTURES

ACADEMIC YEAR : 2020-21 BATCH : 2

# LAB PROGRAM 1

**\*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.**

CODE:

#include <stdio.h> #include <stdlib.h> int stack[50];

int ch;

void push(void); void pop(void); void display(void); int n, top, no, i; int main()

{

top = -1;

printf("\n Enter the size of stack:"); scanf("%d", &n);

printf("\n Please enter the stack operation which you want to perform:"); printf("\n 1.Push\n 2.Pop\n 3.display\n 4.exit");

while (ch != '0')

{

printf("\n Enter the Choice:"); scanf("%d", &ch);

switch (ch)

{

case 1:

push(); break;

case 2:

pop(); break;

case 3:

display(); break;

case 4:

exit(0); break;

default:

{

printf("\nINVALID CHOICE!");

}

}

}

return 0;

}

void push()

{

if (top >= n - 1)

{

printf("\nSTACK OVERFLOW");

}

else

{

printf(" Enter a value to be inserted/pushed:"); scanf("%d", &no);

top++; stack[top] = no;

}

}

void pop()

{

if (top <= -1)

{

printf("\n UNDERFLOW");

}

else

{

printf("\n The popped element is %d", stack[top]); top--;

}

}

void display()

{

if (top >= 0)

{

printf("\n The elements in stack are as follows: \n"); for (i = top; i >= 0; i--)

printf("\n%d\,", stack[i]); printf("\n Press Next Choice");

}

else

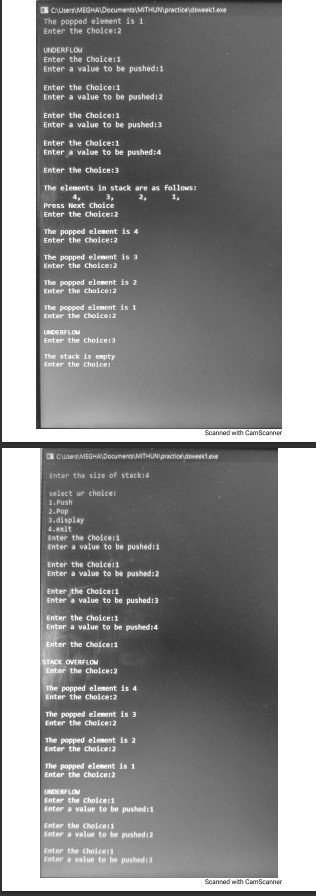
{

printf("\n The stack is empty");

}

}

OUTPUT:



# LAB PROGRAM 2

**\*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).**

SOURCE CODE:

#include <stdio.h> #include <string.h> #include<process.h> int F(char symbol)

{

switch (symbol)

{

case '+':

case '-': return 2;

case '\*':

case '/': return 4;

case '^':

case '$': return 5;

case '(': return 0;

case '#': return -1;

default: return 8;

}

}

int G(char symbol)

{

switch (symbol)

{

case '+':

case '-': return 1;

case '\*':

case '/': return 3;

case '^':

case '$': return 6;

case '(': return 9;

case ')': return 0;

default: return 7;

}

}

void infix\_postfix(char infix[], char postfix[])

{

int top, j, i;

char s[30], symbol; top = -1;

s[++top] = '#'; j = 0;

for (i = 0; i < strlen(infix); i++)

{

symbol = infix[i];

while (F(s[top]) > G(symbol))

{

postfix[j] = s[top--]; j++;

}

if (F(s[top]) != G(symbol))

{

s[++top] = symbol;

}

else

{

top--;

}

}

while (s[top] != '#')

{

postfix[j++] = s[top--];

}

postfix[j] = '\0';

}

void main()

{

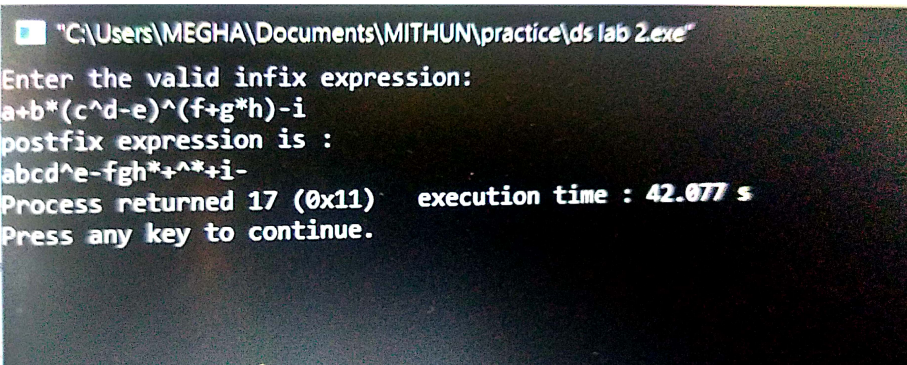
char infix[30], postfix[30];

printf("Enter the valid infix expression:\n"); scanf("%s",infix);

infix\_postfix(infix, postfix); printf("The postfix expression is:\n"); printf("%s\n",postfix);

}

OUTPUT:



# LAB PROGRAM 3

**\*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.**

SOURCE CODE:

#include <stdio.h> #include <conio.h> #include <process.h> #define QUE\_SIZE 5

int item, front = 0, rear = -1, q[10];

void insertRear()

{

if (rear == QUE\_SIZE - 1)

{

printf("Queue Overflow, Cannot add this element.\n"); return;

}

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 the queue:\n"); for (i = front; i <= rear; i++)

{

printf("%d\n", q[i]);

}

}

void main()

{

int choice; for (;;)

{

printf("\n1:insertRear\n2:deleteFront\n3:display\n4:exit\n"); printf("Enter your choice:\n");

scanf("%d", &choice); switch (choice)

{

case 1:

printf("Enter 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;

case 4:

exit(0); default:

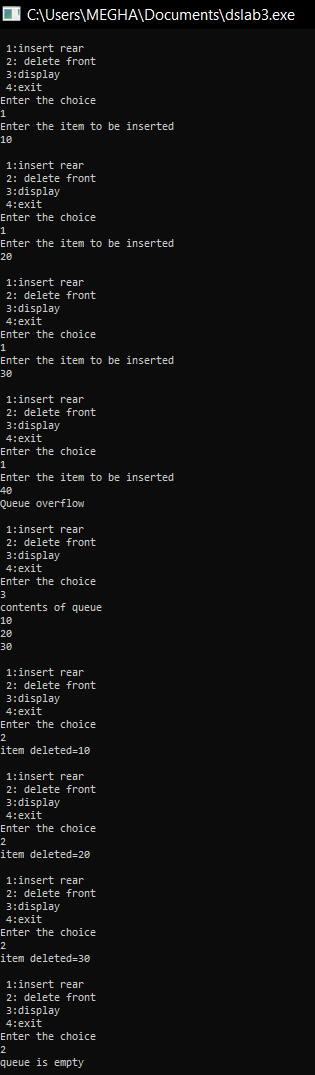
printf("Invalid Choice\n");

}

}

}

OUTPUT:



# LAB PROGRAM 4

**\*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.**

SOURCE CODE:

#include<stdio.h> #include<conio.h> #include<process.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\n"); 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 underflow\n"); return;

}

f=front;

printf("Contents of queue: \n"); for(i=1;i<=count;i++)

{

printf("%d\n",q[f]); f=(f+1)%QUE\_SIZE;

}

}

void main()

{

int choice;

for(;;)

{

printf("\n1:insertrear\n2:deletefront\n3:display\n4: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;

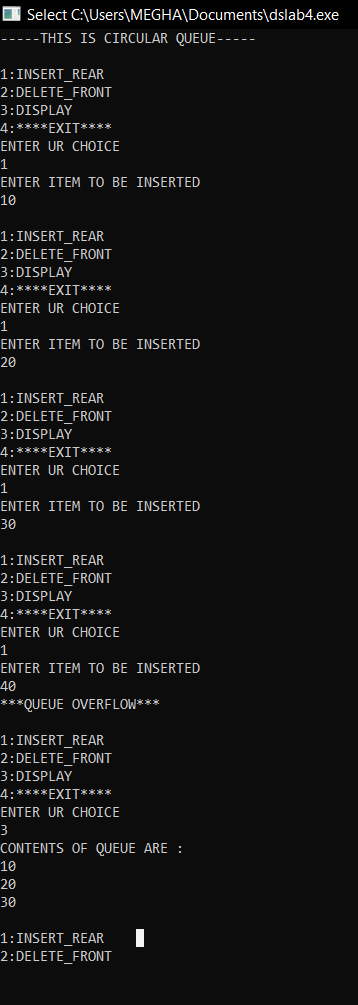
default:exit(0);

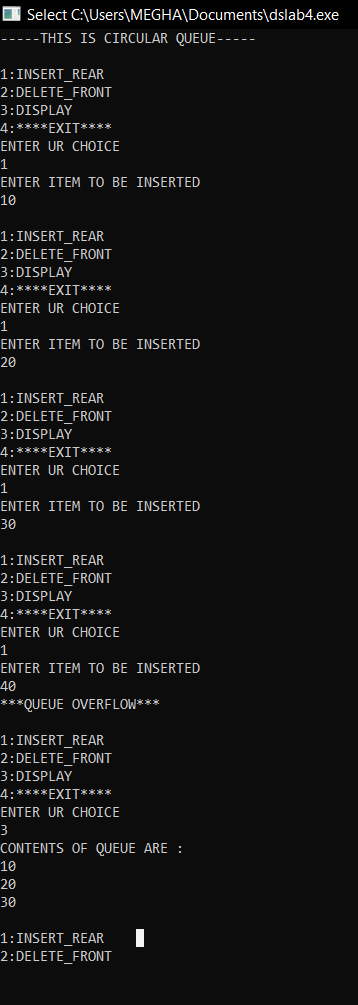
}

}

}

OUTPUT:





# LAB PROGRAM 5

**\*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.**

SOURCE CODE:

#include <stdio.h> #include <conio.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 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 insert\_pos(int item, int pos, NODE first)

{

NODE temp; NODE prev, cur; int count;

temp = getnode(); temp->info = item; temp->link = NULL;

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

if (first == NULL)

{

printf("invalid pos\n"); return first;

}

if (pos == 1)

{

temp->link = first; 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("IP\n");

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, pos;

NODE first = NULL;

for (;;)

{

printf("\n 1:Insert\_front\n 2:Insert\_rear\n3:insert\_pos\n4:display\_list\nDEFAULT: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:

printf("enter the item at rear-end\n"); scanf("%d", &item);

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

case 3:

printf("enter the position and item:\n"); scanf("%d", &pos);

scanf("%d",&item);

first = insert\_pos(item, pos, first); break;

case 4:

display(first); break;

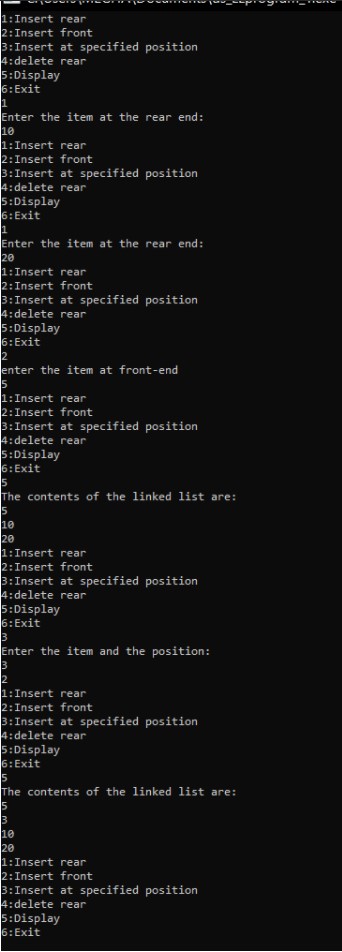
default: exit(0);

}

}

}

OUTPUT:



# LAB PROGRAM 6

**\*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.**

SOURCE CODE:

#include <stdio.h> #include <conio.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("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("iten 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; NODE prev, cur; int count;

temp = getnode(); temp->info = item; temp->link = NULL;

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

if (first == NULL)

{

printf("invalid pos\n"); return first;

}

if (pos == 1)

{

temp->link = first; 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("IP\n"); return first;

}

NODE delete\_pos(int pos, NODE first)

{

NODE prev, cur; int count;

if (first == NULL || pos <= 0)

{

printf("Invalid position\n"); return NULL;

}

if (pos == 1)

{

cur = first;

first = first->link;

printf("iten deleted is %d", cur->info); freenode(cur);

return first;

}

prev = NULL; cur = first; count = 1;

while (cur != NULL)

{

if (count == pos)

{

break;

}

prev = cur;

cur = cur->link; count++;

}

if (count != pos)

{

printf("Invalid position\n"); return first;

}

prev->link = cur->link;

printf("iten deleted is %d", cur->info); 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, pos;

NODE first = NULL;

for (;;)

{

printf("\n 1:Insert\_front\n 2:Delete\_front\n 3:Insert\_rear\n 4:Delete\_rear\n 5:insert\_pos\n 6:delete\_pos\n 7:display\_list\ndefault: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:

printf("enter the item at rear-end\n"); scanf("%d", &item);

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

case 4:

first = delete\_rear(first); break;

case 5:

printf("enter the position and item:\n");

scanf("%d", &pos);

scanf("%d", &item);

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;

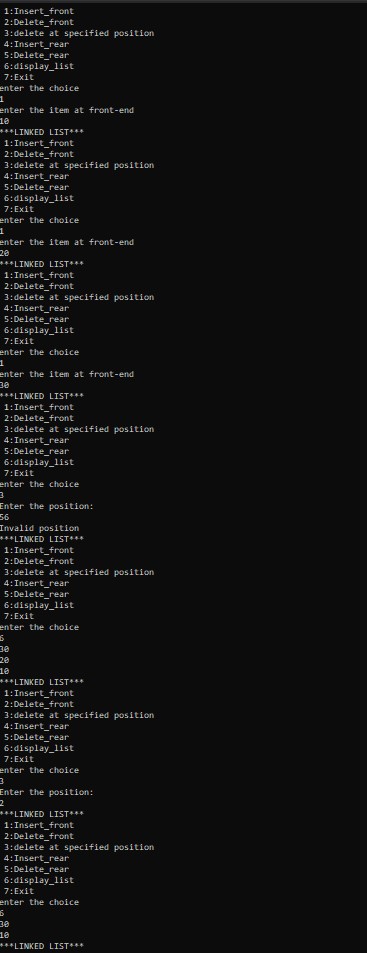
default: exit(0); break;

}

}

}

OUTPUT:



# LAB PROGRAM 7

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

SOURCE CODE:

#include <stdio.h> #include <conio.h> #include <process.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("mem full\n"); exit(0);

}

return 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;

}

void display(NODE first)

{

NODE temp;

if (first == NULL) printf("list empty");

for (temp = first; temp != NULL; temp = temp->link)

{

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

}

}

NODE concat(NODE first, NODE second)

{

NODE cur;

if (first == NULL) return second;

if (second == NULL) return first;

cur = first;

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

cur->link = second; return first;

}

NODE reverse(NODE first)

{

NODE cur, temp; cur = NULL;

while (first != NULL)

{

temp = first; first = first->link;

temp->link = cur; cur = temp;

}

return cur;

}

NODE sort(NODE first)

{

int swapped; NODE ptr1;

NODE lptr = NULL;

if (first == NULL) return NULL;

do

{

swapped = 0; ptr1 = first;

while (ptr1->link != lptr)

{

if (ptr1->info > ptr1->link->info)

{

int tem = ptr1->info;

ptr1->info = ptr1->link->info; ptr1->link->info = tem;

swapped = 1;

}

ptr1 = ptr1->link;

}

lptr = ptr1;

} while (swapped); 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 is %d", cur->info); free(cur);

prev->link = NULL; return first;

}

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 is=%d\n", first->info); free(first);

return temp;

}

NODE order\_list(int item, NODE first)

{

NODE temp, prev, cur; temp = getnode(); temp->info = item; temp->link = NULL;

if (first == NULL) return temp;

if (item < first->info)

{

temp->link = first; return temp;

}

prev = NULL; cur = first;

while (cur != NULL && item > cur->info)

{

prev = cur;

cur = cur->link;

}

prev->link = temp; temp->link = cur; return first;

}

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

{

NODE temp; NODE prev, cur; int count;

temp = getnode(); temp->info = item; temp->link = NULL;

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

if (first == NULL)

{

printf("invalid pos\n"); return first;

}

if (pos == 1)

{

temp->link = first; 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("IP\n"); return first;

}

NODE delete\_pos(int pos, NODE first)

{

NODE prev, cur; int count;

if (first == NULL || pos <= 0)

{

printf("Invalid position\n"); return NULL;

}

if (pos == 1)

{

cur = first;

first = first->link;

printf("iten deleted is %d", cur->info); free(cur);

return first;

}

prev = NULL; cur = first; count = 1;

while (cur != NULL)

{

if (count == pos)

{

break;

}

prev = cur;

cur = cur->link; count++;

}

if (count != pos)

{

printf("Invalid position\n"); return first;

}

prev->link = cur->link;

printf("iten deleted is %d", cur->info); free(cur);

return first;

}

void main()

{

int item, choice, pos, i, n; NODE first = NULL, a, b;

for (;;)

{

printf("1.insert\_rear\n2.concat\n3.reverse\n4.display\n5.sort\n6.delete\_rear\n7.insert\_fro nt\n8.delete\_front\n9.order\_list\n10.insert\_pos\n11.delete\_pos\ndefault:exit\n");

printf("enter the choice\n"); scanf("%d", &choice);

switch (choice)

{

case 1:

printf("enter the item\n"); scanf("%d", &item);

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

case 2:

printf("enter the no of nodes in 1\n"); scanf("%d", &n);

a = NULL;

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

{

printf("enter the item\n"); scanf("%d", &item);

a = insert\_rear(a, item);

}

printf("enter the no of nodes in 2\n"); scanf("%d", &n);

b = NULL;

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

{

printf("enter the item\n"); scanf("%d", &item);

b = insert\_rear(b, item);

}

a = concat(a, b); display(a); break;

case 3:

first = reverse(first);

display(first); break;

case 4:

display(first); break;

case 5:

first = sort(first); break;

case 6:

first = delete\_rear(first); break;

case 7:

printf("Enter the item\n"); scanf("%d", &item);

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

case 8:

first = delete\_front(first); break;

case 9:

printf("Enter the item to be inserted in ordered\_list\n"); scanf("%d", &item);

first = order\_list(item, first); break;

case 10:

printf("enter the position and item:\n"); scanf("%d", &pos);

scanf("%d", &item);

first = insert\_pos(item, pos, first); break;

case 11:

printf("Enter the position:\n"); scanf("%d", &pos);

first = delete\_pos(pos, first); break;

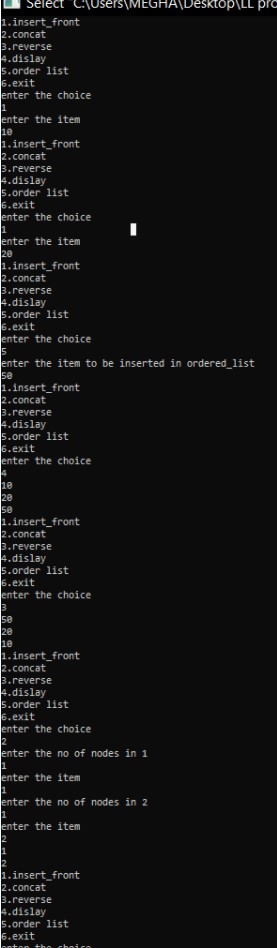
default: exit(0);

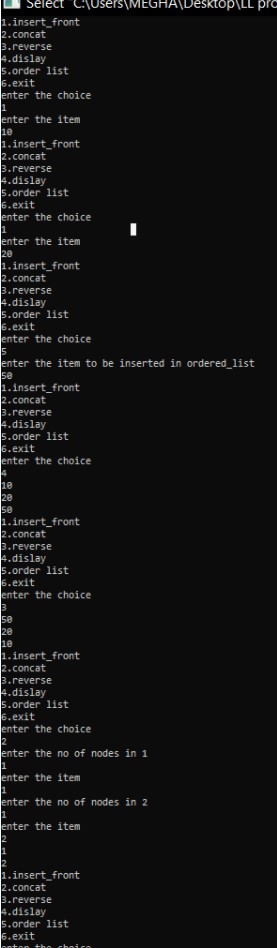
}

}

}

OUTPUT:





# LAB PROGRAM 8

**\*WAP to implement Stack & Queues using Linked Representation.**

SOURCE CODE:

#include<stdio.h> #include<conio.h> #include<process.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)

{

}

else

printf("MEMORY IS FULL\n"); exit(0);

return x;

}

NODE insertfront(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 insertrear(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 deletefront(NODE first)

{

NODE cur; if(first==NULL)

{

printf("THE LIST IS EMPTY\n");

return first;

}

cur=first; cur=cur->link;

printf("THE DELETED ITEM FROM FRONT IS=%d\n",first->info); free(first);

return cur;

}

NODE deleterear(NODE first)

{

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

{

printf("THE LIST IS EMPTY\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("THE DELETED ITEM FROM REAR IS=%d\n",cur->info); free(cur);

prev->link=NULL; return first;

}

void display(NODE first)

{

NODE temp; if(first==NULL){

printf("THE LIST IS EMPTY\n");

}

printf("THE ELEMENTS ARE=");

for(temp=first;temp!=NULL;temp=temp->link)

{

printf("%d\t",temp->info);

}

printf("\n");

}

int main()

{

int c,item,pos; int n,i;

int choice;

NODE first=NULL,sec,fir; for(;;)

{

printf(" 1-STACK \n 2-QUEUE \n 3-EXIT\n"); printf("ENTER THE CHOICE\n");

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

{

case 1: printf("STACK\n");

for(;;)

{

printf("\n 1:Insert\_rear\n 2:Delete\_rear\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=insertrear(first,item);

break;

case 2:first=deleterear(first); break;

case 3:display(first); break;

default:exit(0);

}

}

break; case 2:

printf("QUEUE\n"); 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=insertrear(first,item); break;

case 2:first=deletefront(first); break;

case 3:display(first); break;

default:exit(0); break;

}

}

break;

case 3:

exit(0); default:

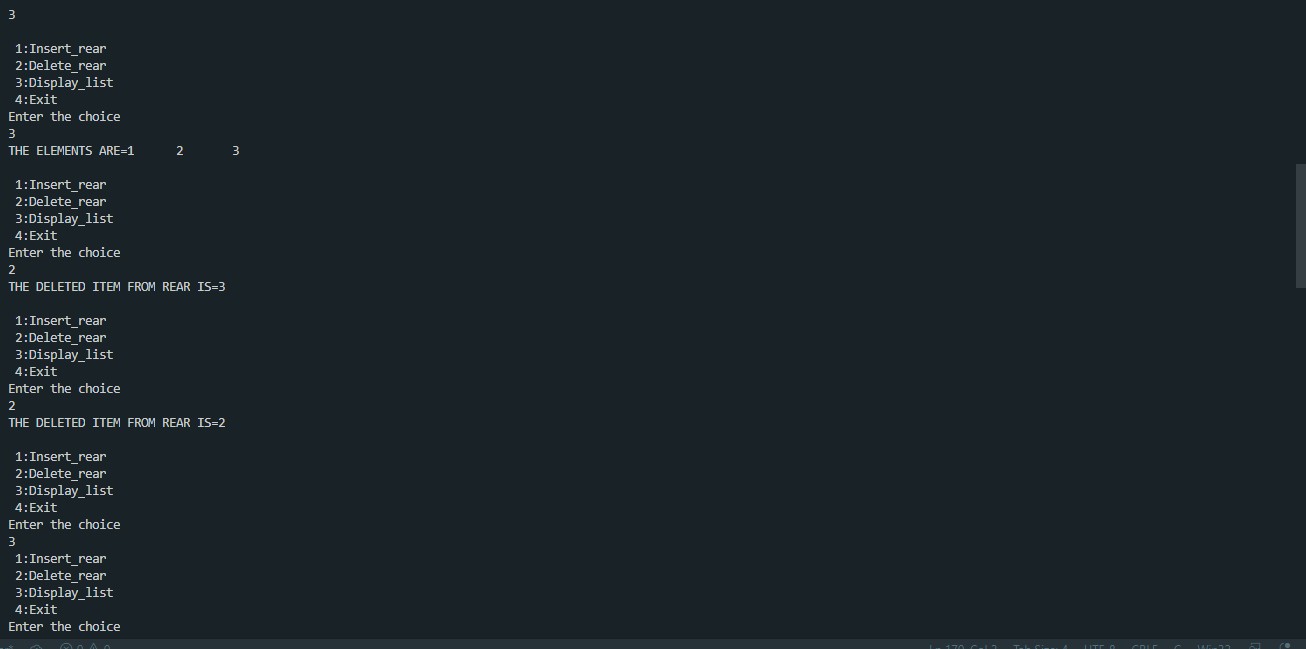
printf("INVALID CHOICE\n");

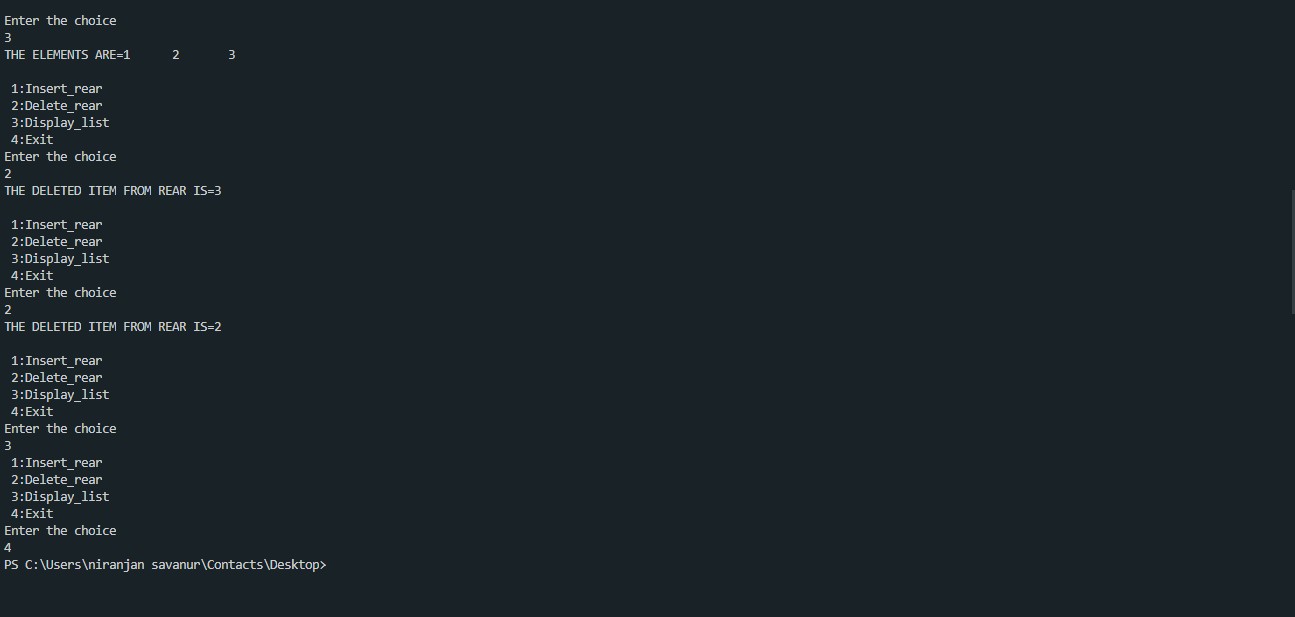
}

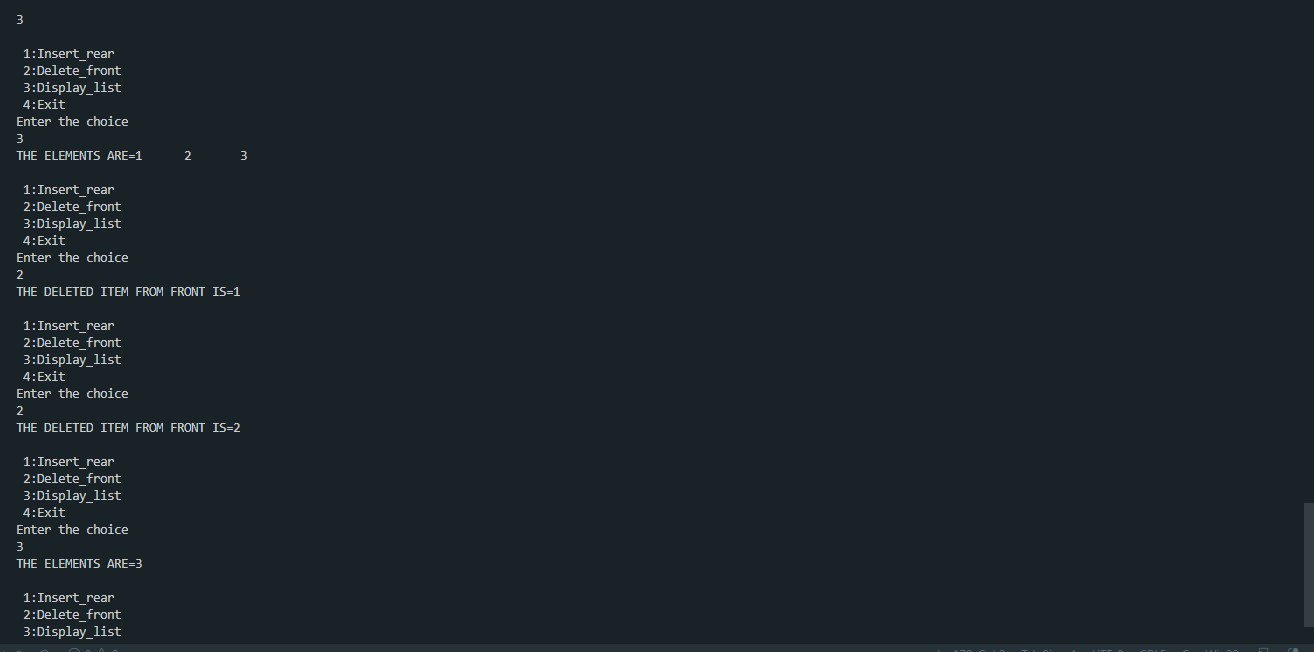
}

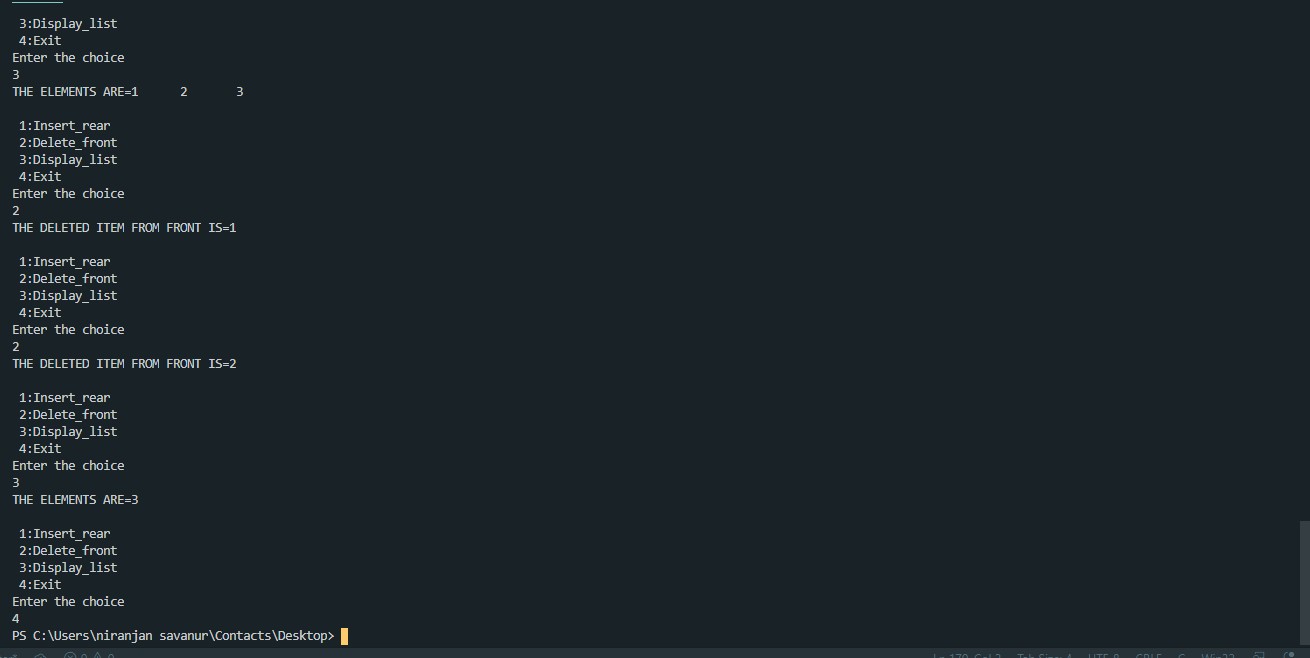
}

OUTPUT:









# LAB PROGRAM 9

**\*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.**

SOURCE CODE:

#include <stdio.h> #include <stdlib.h> #include <process.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 dinsert\_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 dinsert\_front(int item, NODE head)

{

NODE temp, cur; temp = getnode(); temp->info = item; cur = head->rlink; head->rlink = temp; temp->llink = head; temp->rlink = cur; cur->llink = temp; return head;

}

NODE ddelete\_front(NODE head)

{

NODE cur, next;

if (head->rlink == head)

{

printf("dq empty\n"); return head;

}

cur = head->rlink; next = cur->rlink; head->rlink = next; next->llink = head;

printf("the node deleted is %d", cur->info); freenode(cur);

return head;

}

NODE ddelete\_rear(NODE head)

{

NODE cur, prev;

if (head->rlink == head)

{

printf("dq empty\n"); return head;

}

cur = head->llink; prev = cur->llink; head->llink = prev; prev->rlink = head;

printf("the node deleted is %d", cur->info); freenode(cur);

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;

}

void Search\_info(int item, NODE head)

{

NODE cur;

if (head->rlink == head)

{

printf("list empty\n");

}

cur = head->rlink; while (cur != head)

{

if (item == cur->info)

{

printf("Search Successfull\n"); break;

}

cur = cur->rlink;

}

if (cur == head)

{

printf("Info not found\n");

}

}

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;

head = getnode(); head->rlink = head; head->llink = head; for (;;)

{

printf("\n1.insert\_rear\n2.insert\_front\n3.delete\_front\n4.delete\_rear\n5.insert\_key\_left\ n6.delete\_duplicates\n7.Searh\_info\n8.display\ndefault: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);

head = dinsert\_rear(head, item); break;

case 2:

printf("enter the item at front end\n"); scanf("%d", &item);

head = dinsert\_front(item, head); break;

case 3:

head = ddelete\_front(head); break;

case 4:

head = ddelete\_rear(head); break;

case 5:

printf("enter the key item\n"); scanf("%d", &item);

head = insert\_leftpos(item, head); break;

case 6:

printf("enter the key item\n"); scanf("%d", &item);

head = delete\_all\_key(item, head); break;

case 7:

printf("enter the key item\n"); scanf("%d", &item); Search\_info(item, head); break;

case 8:

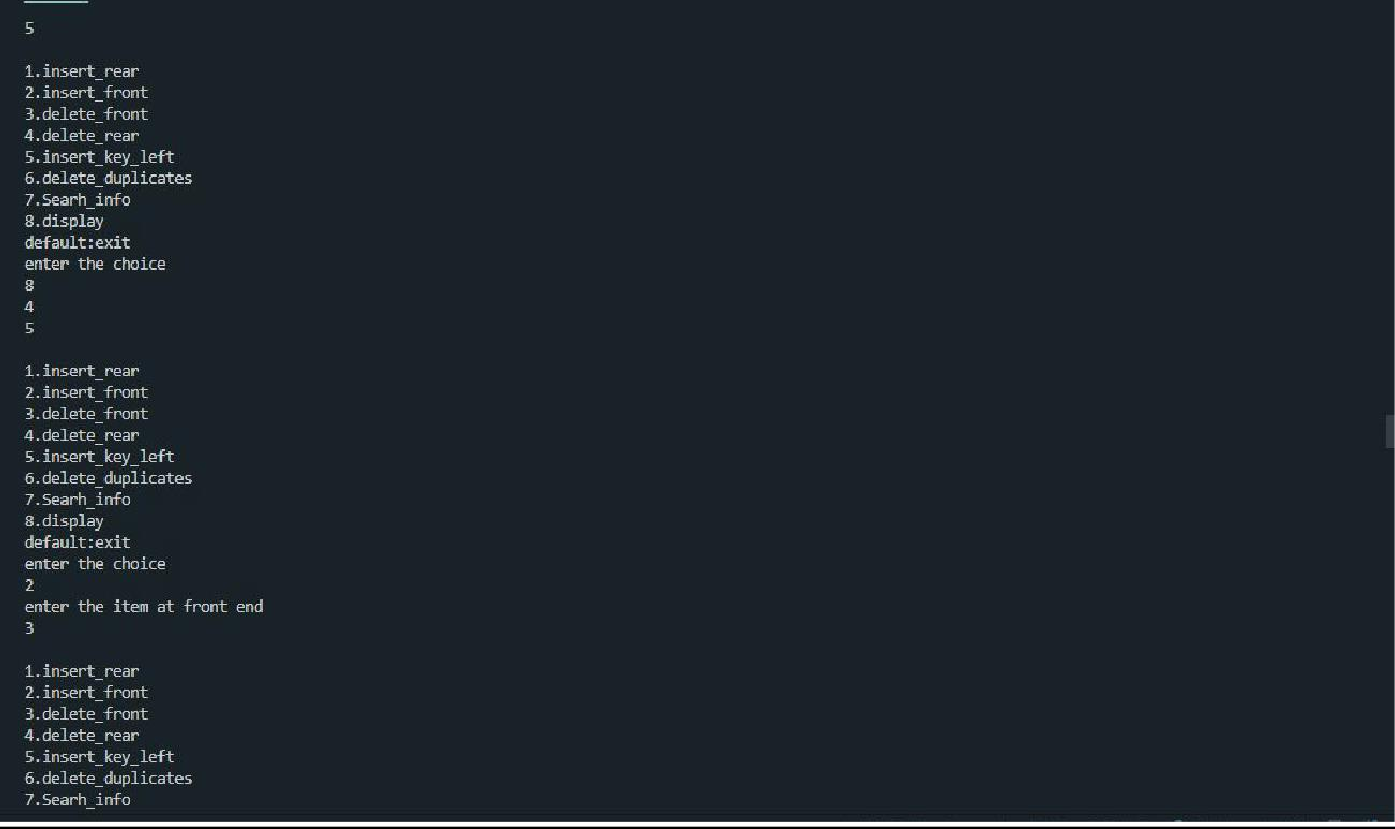
display(head); break;

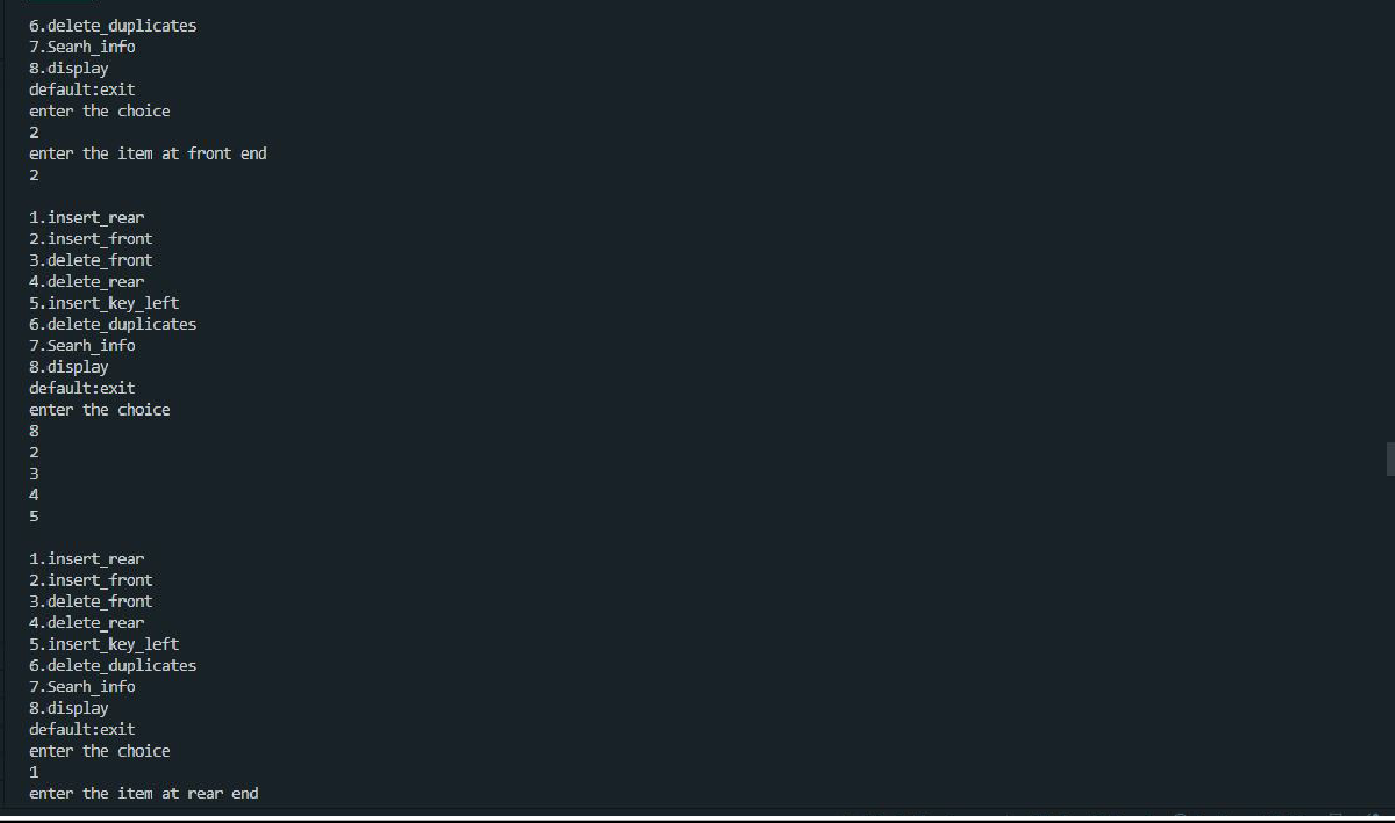
default: exit(0); break;

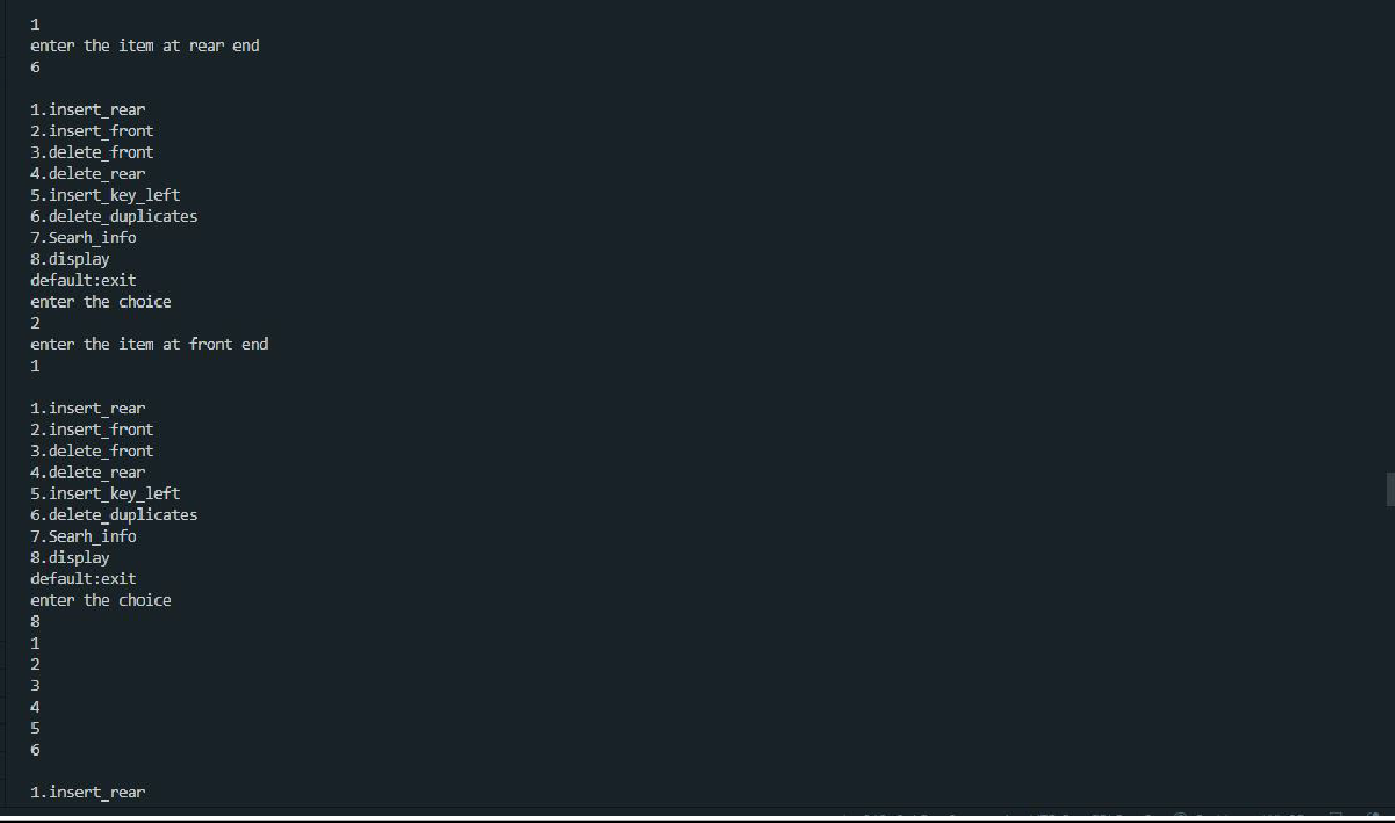
}

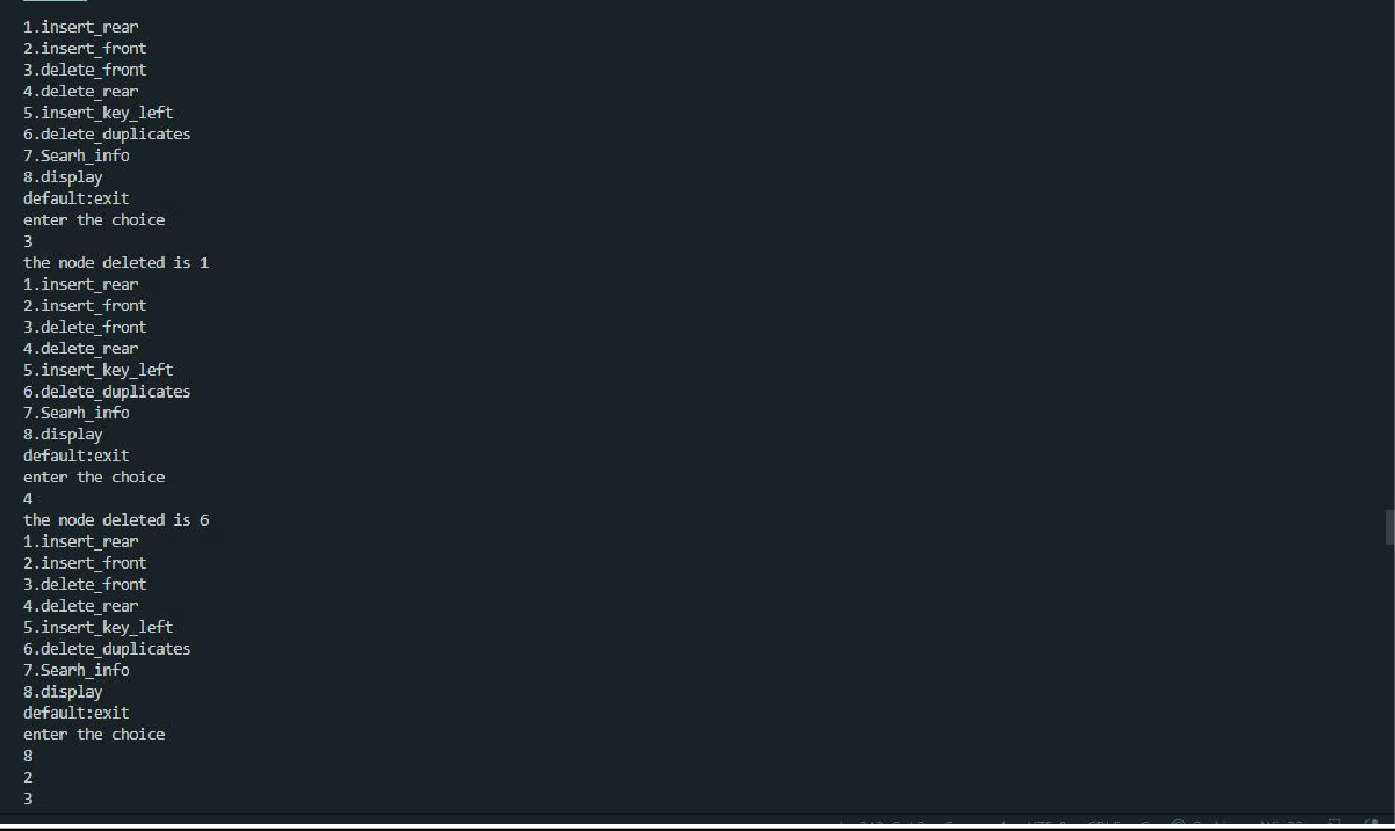
}

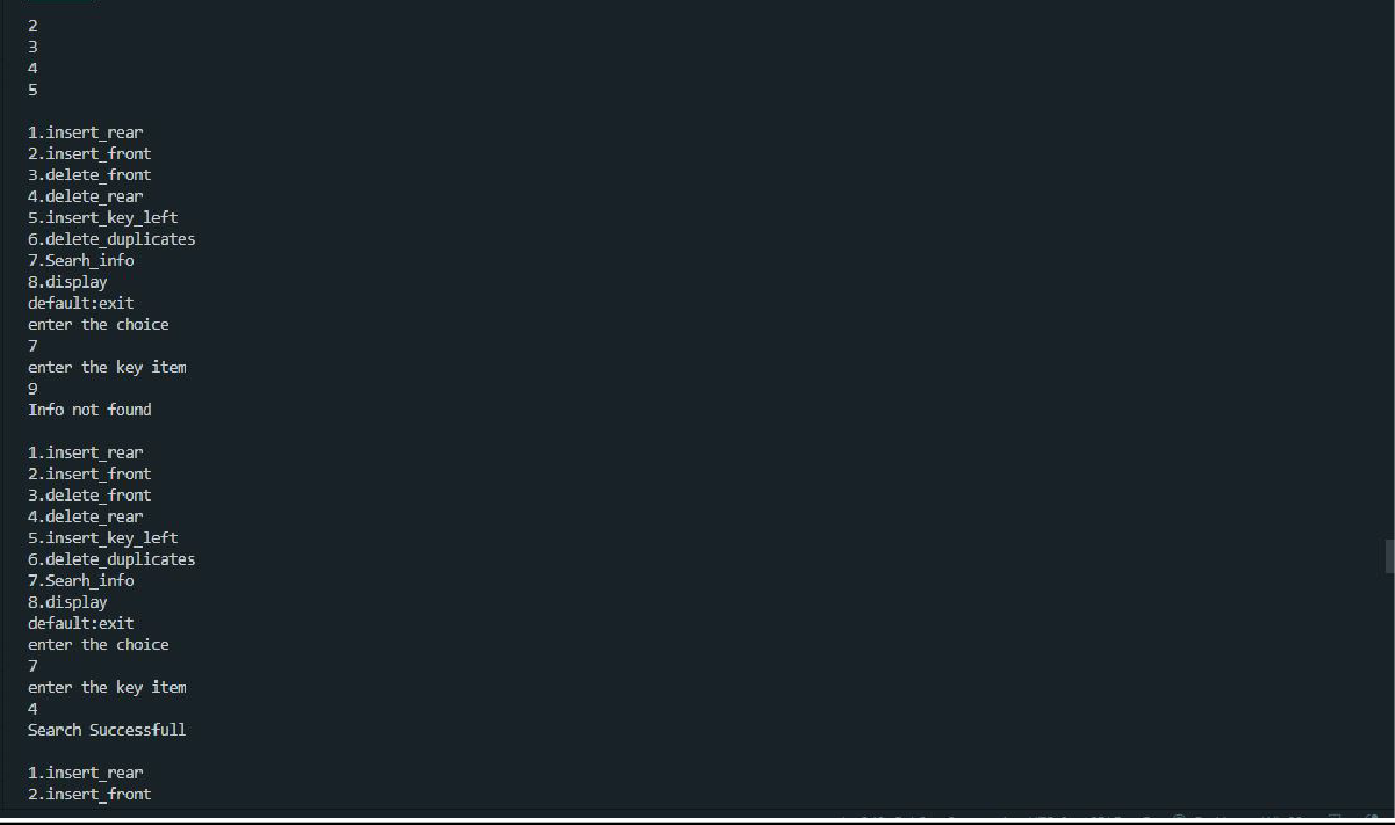
}

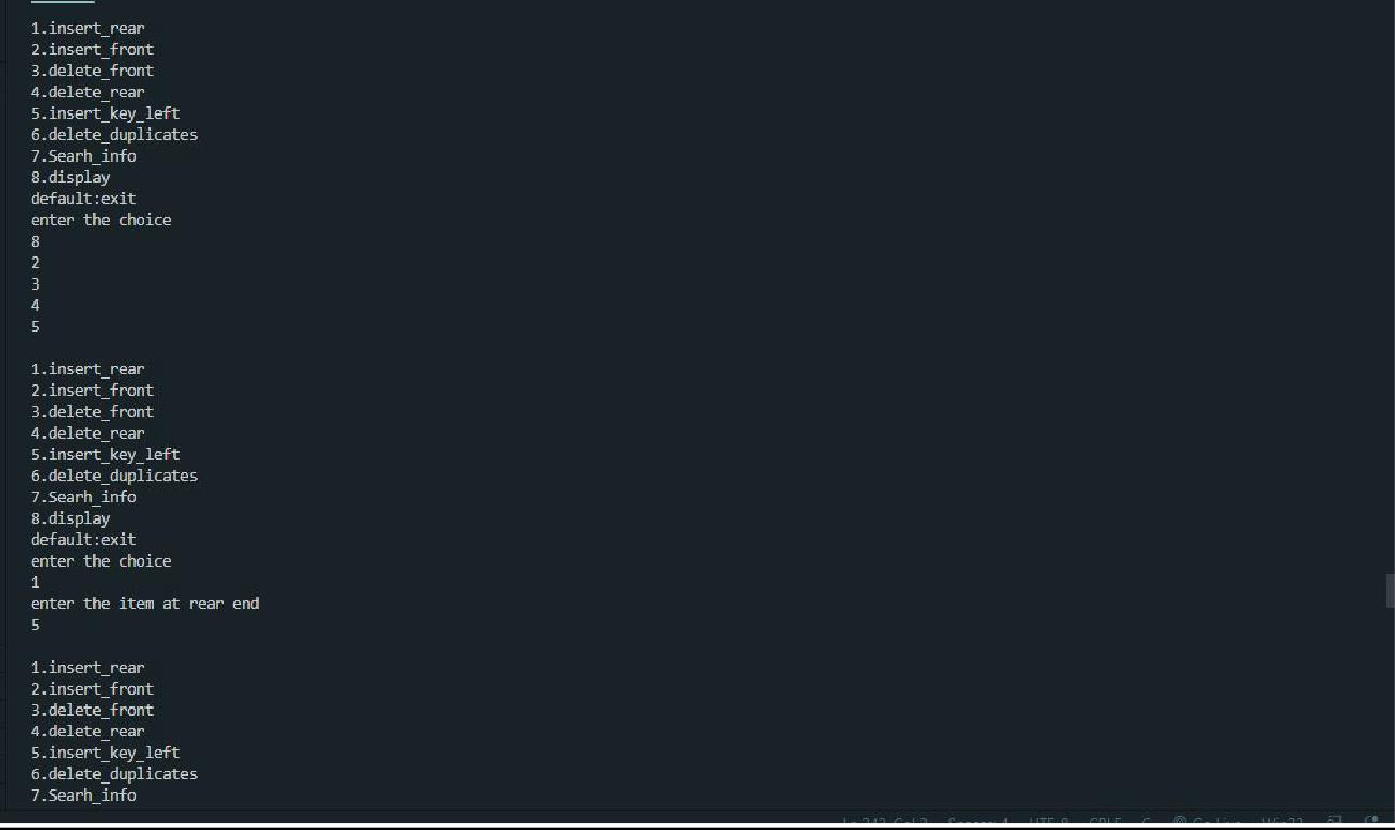
OUTPUT:



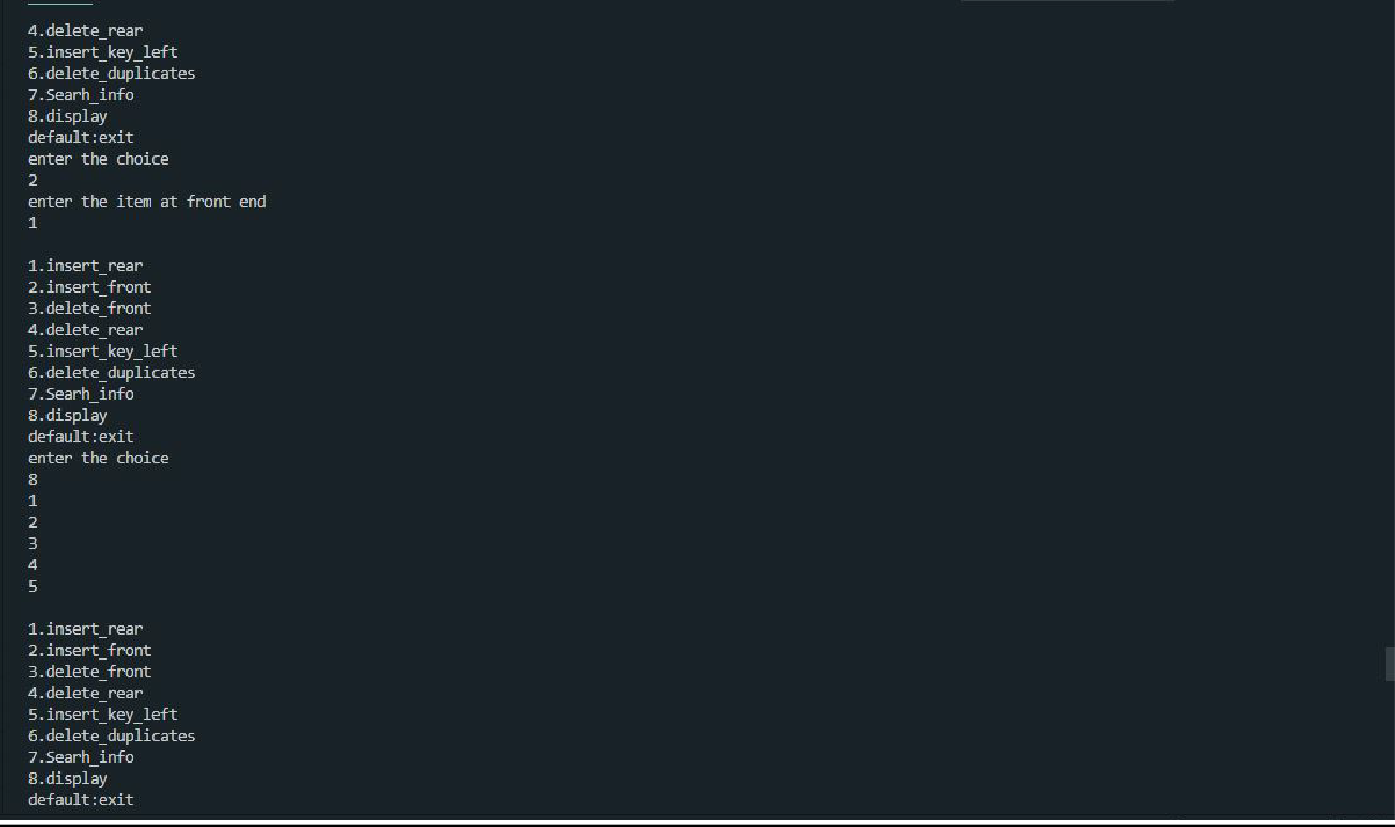




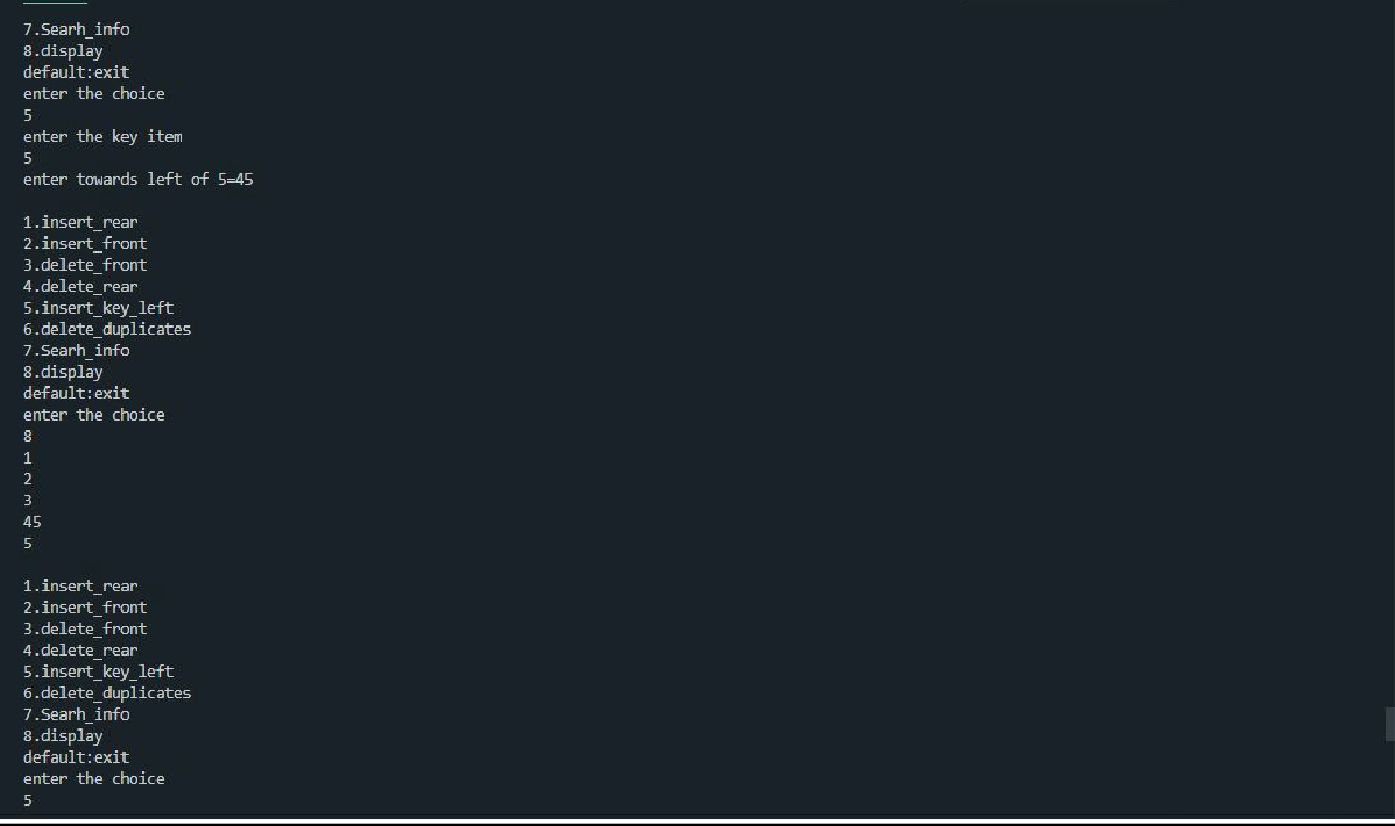


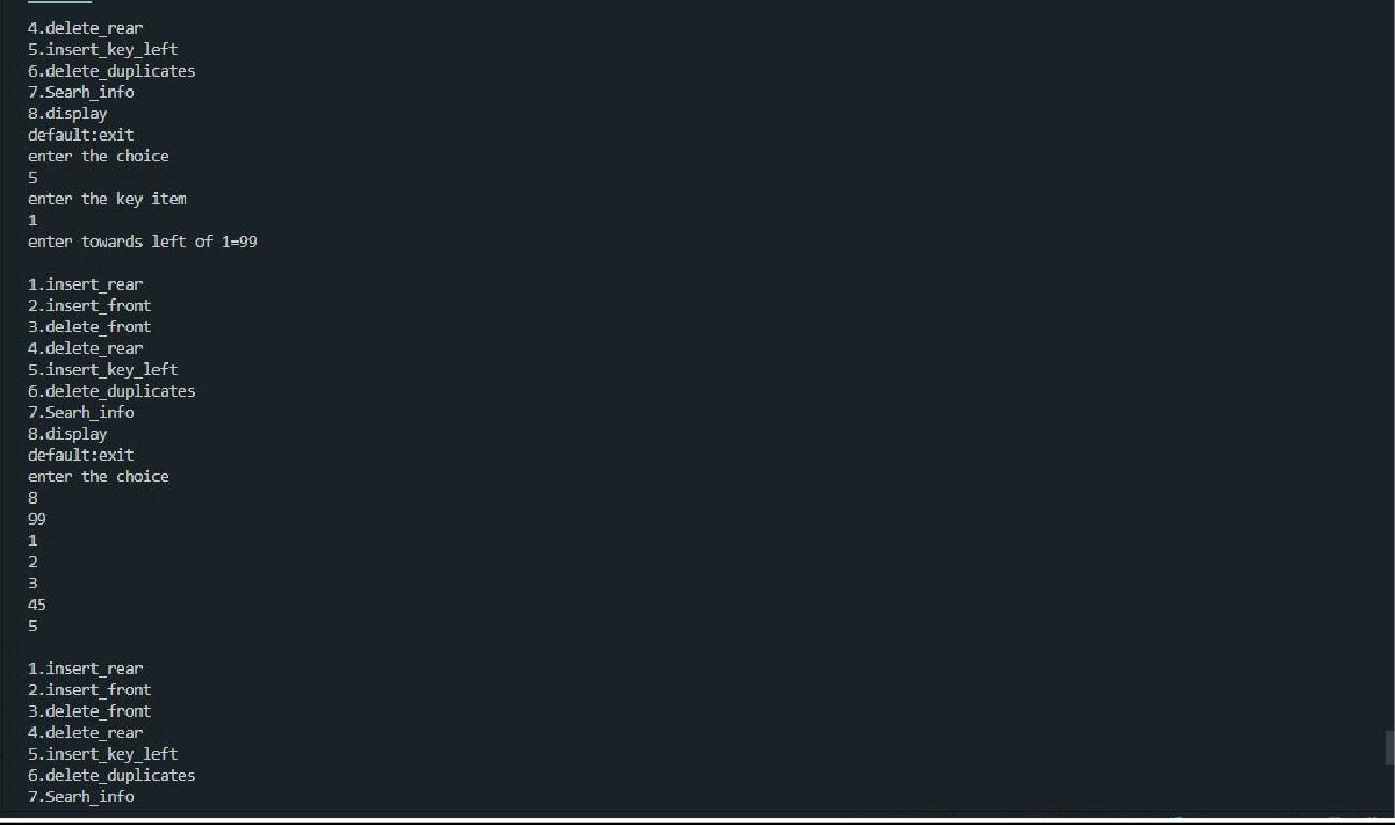


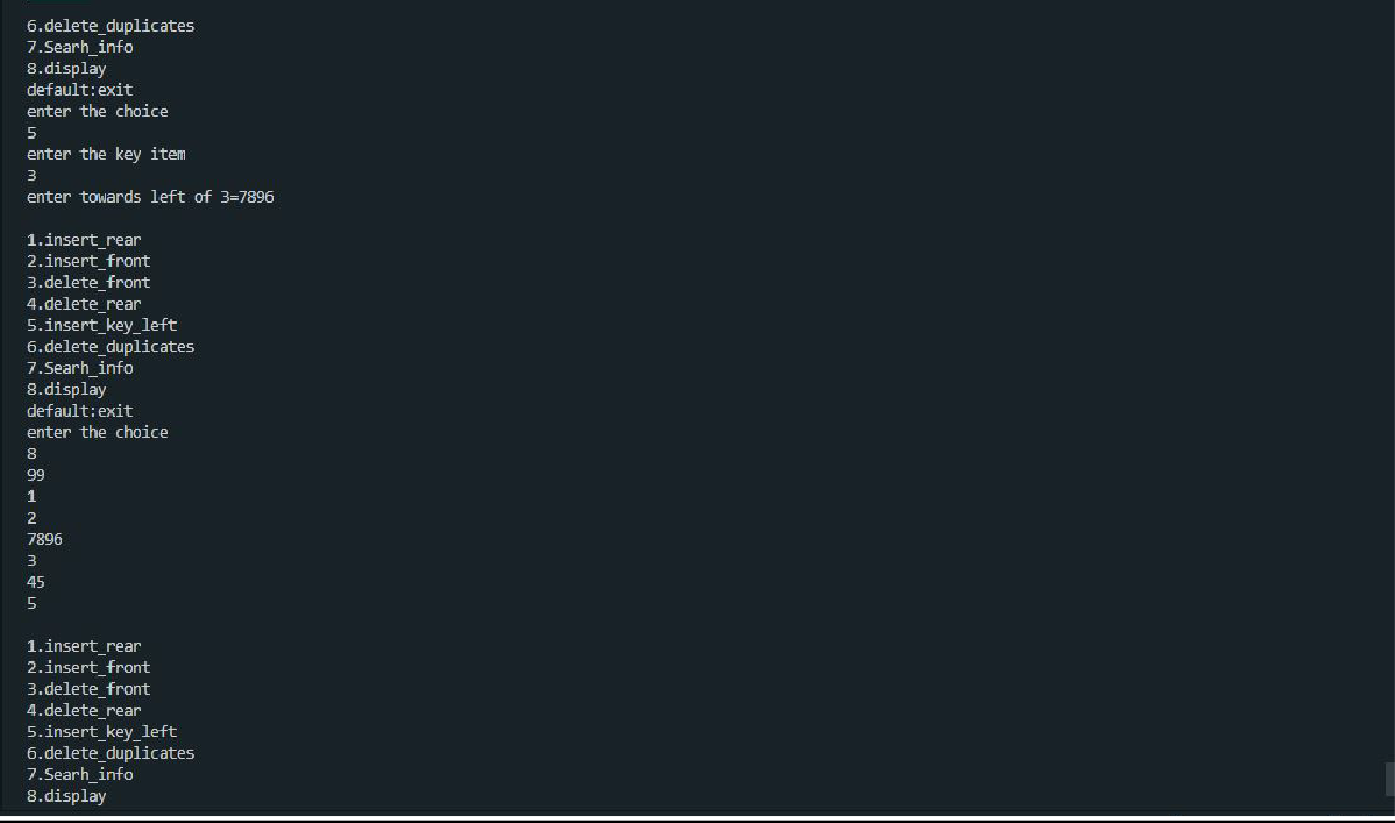


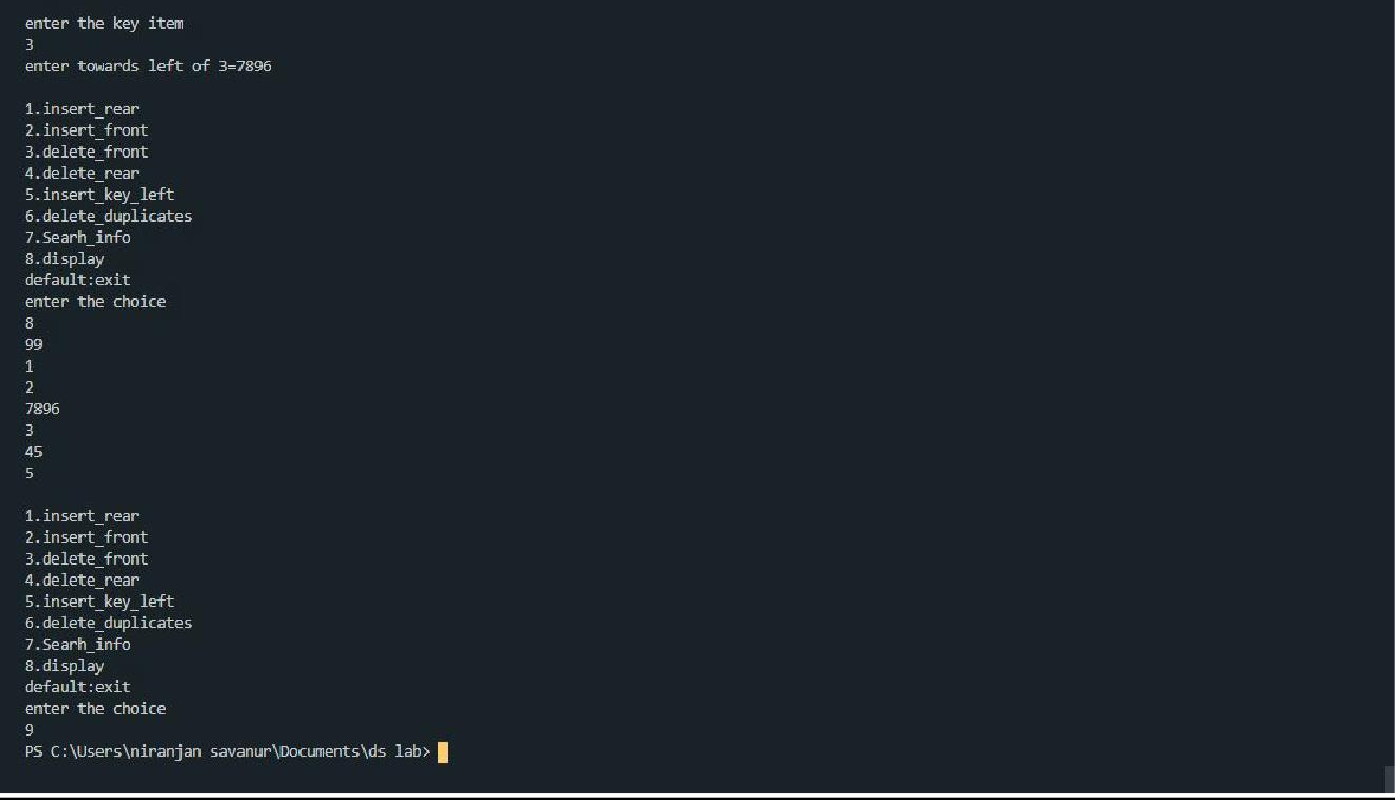












# LAB PROGRAM 10

**\*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.**

SOURCE CODE:

#include <stdio.h> #include <process.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(NODE root, int item)

{

NODE temp, cur, prev; temp = getnode();

temp->rlink = NULL; temp->llink = NULL; temp->info = item; if (root == NULL)

return temp; prev = NULL; cur = root;

while (cur != NULL)

{

prev = cur;

cur = (item < cur->info) ? cur->llink : cur->rlink;

}

if (item < prev->info) prev->llink = temp;

else

prev->rlink = temp; return root;

}

void display(NODE root, int i)

{

int j;

if (root != NULL)

{

display(root->rlink, i + 1); for (j = 0; j < i; j++)

printf(" "); printf("%d\n", root->info); display(root->llink, i + 1);

}

}

NODE delete (NODE root, int item)

{

NODE cur, parent, q, suc; if (root == NULL)

{

printf("empty\n"); return root;

}

parent = NULL; cur = root;

while (cur != NULL && item != cur->info)

{

parent = cur;

cur = (item < cur->info) ? cur->llink : cur->rlink;

}

if (cur == NULL)

{

printf("not found\n"); return root;

}

if (cur->llink == NULL) q = cur->rlink;

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

else

{

suc = cur->rlink;

while (suc->llink != NULL) suc = suc->llink;

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

}

if (parent == NULL) return q;

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

else

parent->rlink = q; freenode(cur); return root;

}

void preorder(NODE root)

{

if (root != NULL)

{

printf("%d\n", root->info); preorder(root->llink); preorder(root->rlink);

}

}

void postorder(NODE root)

{

if (root != NULL)

{

postorder(root->llink); postorder(root->rlink); printf("%d\n", root->info);

}

}

void inorder(NODE root)

{

if (root != NULL)

{

inorder(root->llink); printf("%d\n", root->info); inorder(root->rlink);

}

}

void main()

{

int item, choice;

NODE root = NULL;

for (;;)

{

printf("\n1.insert\n2.display\n3.pre\n4.post\n5.in\n6.delete\n7.exit\n"); printf("enter the choice\n");

scanf("%d", &choice); switch (choice)

{

case 1:

printf("enter the item\n"); scanf("%d", &item);

root = insert(root, item); break;

case 2:

display(root, 0); break;

case 3:

preorder(root); break;

case 4:

postorder(root); break;

case 5:

inorder(root); break;

case 6:

printf("enter the item\n"); scanf("%d", &item);

root = delete (root, item); break;

default: exit(0); break;

}

}

}

OUTPUT:

