VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

NITHYA LAKSHMI V (1BM22CS186)

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
Dec 2023- March 2024

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

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by **NITHYA LAKSHMI V (1BM22CS186)**, 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 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab **- (23CS3PCDST) work** prescribed for the said degree.

Prof. Sneha S Bagalkot	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

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 the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

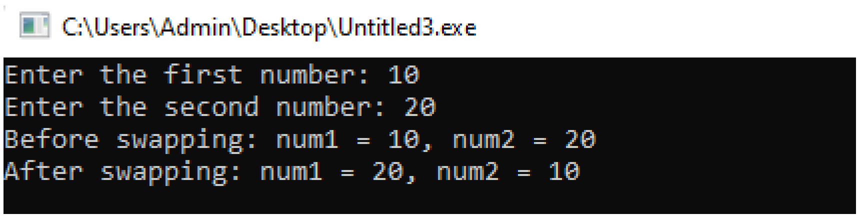
LAB -1

PROGRAM 1 : Swapping using Pointers

SOURCE CODE :

```
#include <stdio.h>
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
int main() {
   int num1, num2;
   printf("Enter the first number: ");
   scanf("%d", &num1);
   printf("Enter the second number: ");
   scanf("%d", &num2);
   printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
   swap(&num1, &num2);
   printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
   return 0;
}
```

OUTPUT:



```
Enter the first number: 10
Enter the second number: 20
Before swapping: num1 = 10, num2 = 20
After swapping: num1 = 20, num2 = 10
```

PROGRAM 2 : Dynamic memory allocation [Program should include malloc, free, calloc, realloc]

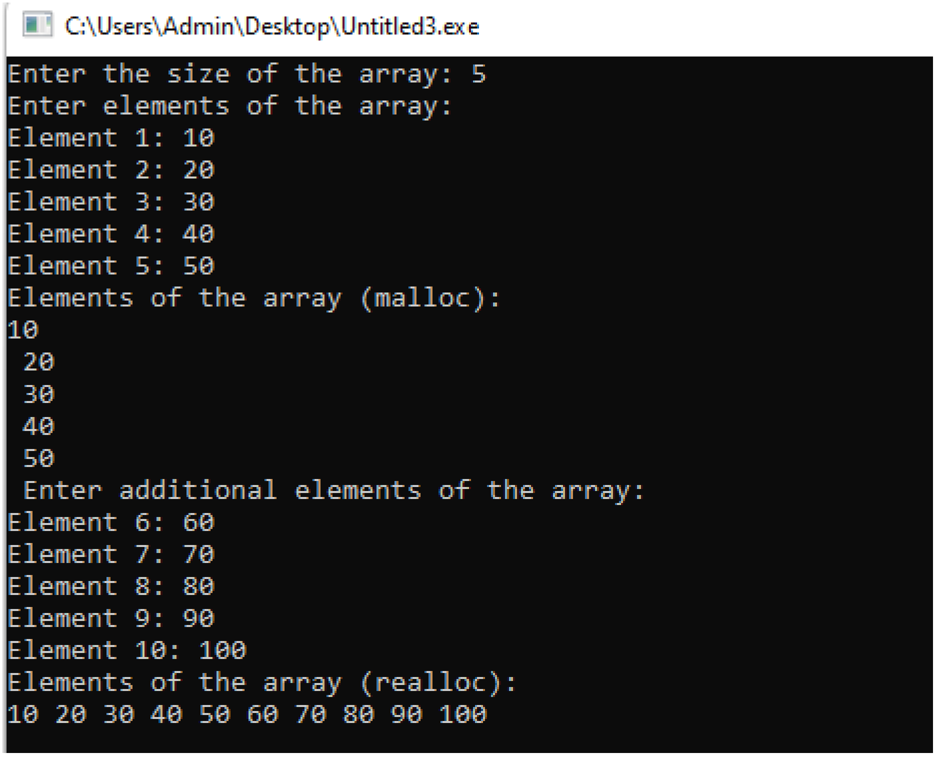
SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
void* Malloc(size_t size)
{
   return malloc(size);
}
void* Realloc(void* ptr, size_t size)
{
   return realloc(ptr, size);
```

```
}
void* Calloc(size_t num, size_t size)
{
   return calloc(num, size);
}
void Free(void* ptr)
{
   free(ptr);
}
int main()
{
   int *arr1, *arr2;
   size_t size;
   printf("Enter the size of the array: ");
   scanf("%d", &size);
   arr1 = (int*)Malloc(size * sizeof(int));
   if (arr1 == NULL)
   {
      printf("Memory allocation failed.\n");
      return 1;
   }
   printf("Enter elements of the array:\n");
   for (size_t i = 0; i < size; i++)
      {
      printf("Element %d: ", i + 1);
      scanf("%d", &arr1[i]);
   }
   printf("Elements of the array (malloc):\n");
   for (size_t i = 0; i < size; i++)
   {
      printf("%d\n ", arr1[i]);
   }
   size *= 2;
   arr2 = (int*)Realloc(arr1, size * sizeof(int));
   if (arr2 == NULL)
   {
      printf("Memory reallocation failed.\n");
      Free(arr1);
      return 1;
   }
   printf("Enter additional elements of the array:\n");
   for (size_t i = size / 2; i < size; i++)
   {
      printf("Element %d: ", i + 1);
      scanf("%d", &arr2[i]);
   }
   printf("Elements of the array (realloc):\n");
```

```
 for (size_t i = 0; i < size; i++)
    {
       printf("%d ", arr2[i]);
    }
    printf("\n");
    Free(arr2);
    return 0;
}
```

OUTPUT:



C:\Users\Admin\Desktop\Untitled3.exe

```
Enter the size of the array: 5
Enter elements of the array:
Element 1: 10
Element 2: 20
Element 3: 30
Element 4: 40
Element 5: 50
Elements of the array (malloc):
10
 20
 30
 40
 50
 Enter additional elements of the array:
Element 6: 60
Element 7: 70
Element 8: 80
Element 9: 90
Element 10: 100
Elements of the array (realloc):
10 20 30 40 50 60 70 80 90 100
```

PROGRAM 3 : Stack implementation [Lab Program: push, pop, display functions to be implemented]

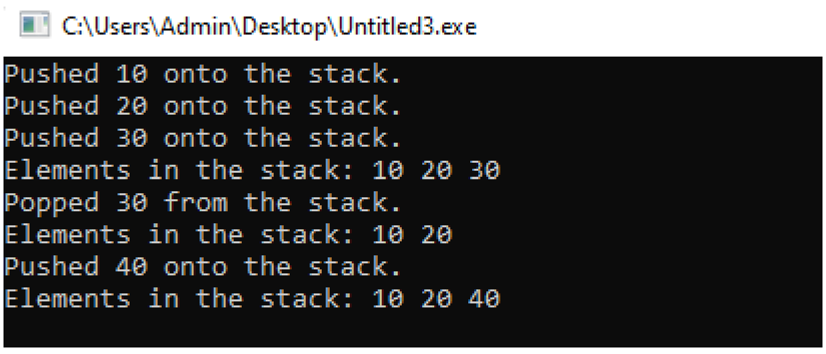
SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
struct Stack {
    int items[MAX_SIZE];
    int top;
};
```

```
void initialize(struct Stack *stack)
{
   stack->top = -1;
}
int isEmpty(struct Stack *stack)
{
   return stack->top == -1;
}
int isFull(struct Stack *stack)
{
   return stack->top == MAX_SIZE - 1;
}
void push(struct Stack *stack, int value)
{
   if (isFull(stack))
      {
      printf("Stack overflow. Cannot push %d.\n", value);
   } else
   {
      stack->top++;
      stack->items[stack->top] = value;
      printf("Pushed %d onto the stack.\n", value);
   }
}
int pop(struct Stack *stack)
{
   int poppedValue = -1;

   if (isEmpty(stack))
      {
      printf("Stack underflow. Cannot pop from an empty stack.\n");
   } else
   {
      poppedValue = stack->items[stack->top];
      stack->top--;
      printf("Popped %d from the stack.\n", poppedValue);
   }

   return poppedValue;
}
void display(struct Stack *stack)
{
   if (isEmpty(stack))
      {
      printf("Stack is empty.\n");
   } else
   {
```

```
  printf("Elements in the stack: ");
      for (int i = 0; i <= stack->top; i++)
      {
         printf("%d ", stack->items[i]);
      }
      printf("\n");
   }
}

int main()
{
   struct Stack stack;
   initialize(&stack);

   push(&stack, 10);
   push(&stack, 20);
   push(&stack, 30);
   display(&stack);

   pop(&stack);
   display(&stack);

   push(&stack, 40);
   display(&stack);

   return 0;
}
```

<u>OUTPUT:</u>



```
Pushed 10 onto the stack.
Pushed 20 onto the stack.
Pushed 30 onto the stack.
Elements in the stack: 10 20 30
Popped 30 from the stack.
Elements in the stack: 10 20
Pushed 40 onto the stack.
Elements in the stack: 10 20 40
```

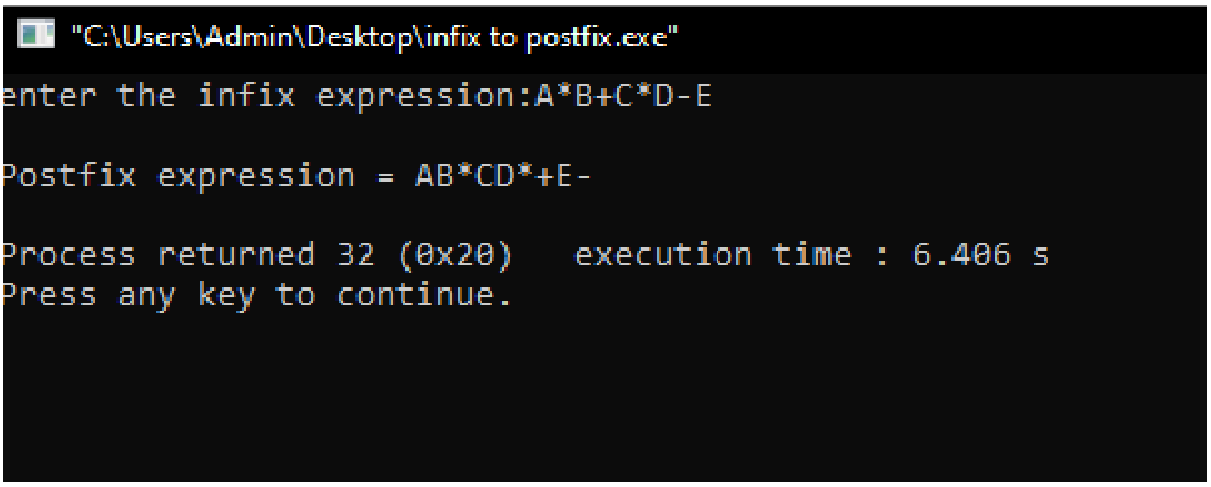
LAB -2

PROGRAM 2(a) : INFIX TO POSTFIX EXPRESSION

SOURCE CODE:

```
#include <stdio.h>
#include<ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;
push(char ele)
{
stack[++top]=ele;
}
char pop()
{
return(stack[top--]);
}
int pr(char symbol)
{
if (symbol == '^')
{
return(3);
}
else if (symbol == '*' || symbol == '/')
{
return(2);
}
else if (symbol == '+' || symbol == '-')
{
return(1);
}
else
{
return (0);
}
}
void main()
{
char infix[50],postfix[50],ch,ele;
int i=0,k=0;
printf("enter the infix expression:");
scanf("%s",infix);
push('#');
while( (ch=infix[i++]) != '\0')
{
if(ch=='(') push(ch);
else
if(isalnum(ch)) postfix[k++]=ch;
else
if(ch ==')')
{
while(stack[top]!='(')
postfix[k++]=pop();
ele=pop();
}
else
{
while(pr(stack[top])>=pr(ch))
postfix[k++]=pop();
push(ch);
}
}
while(stack[top]!='#')
postfix[k++]=pop();
postfix[k]='\0';
printf("\nPostfix expression = %s\n",postfix);
}
```

OUTPUT:



```
"C:\Users\Admin\Desktop\infix to postfix.exe"
enter the infix expression:A*B+C*D-E

Postfix expression = AB*CD*+E-

Process returned 32 (0x20)   execution time : 6.406 s
Press any key to continue.
```

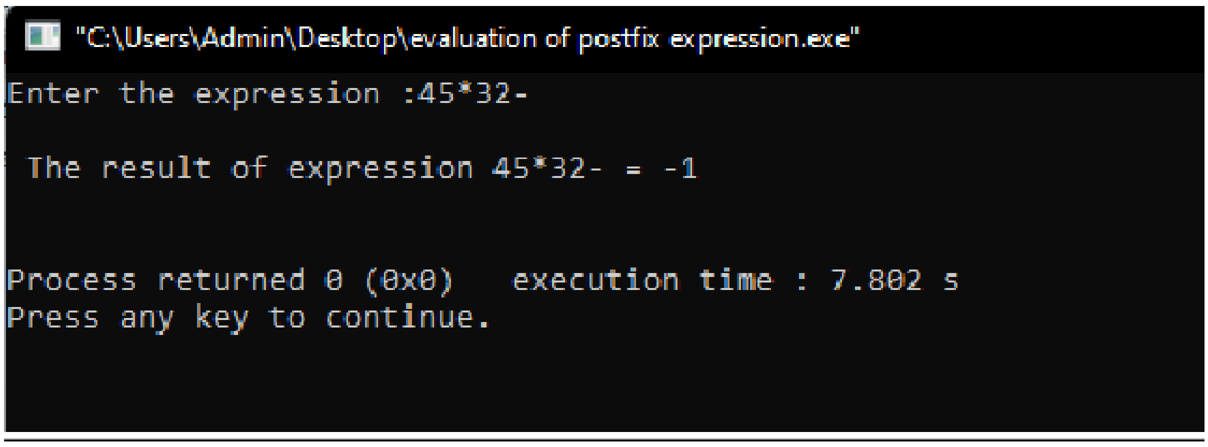
PROGRAM 2(b) : EVALUATION OF POSTFIX EXPRESSION

SOURCE CODE:

```
#include<stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
```

```
stack[++top]=x;
}
int pop()
{
return stack[top--];
}
int main()
{
char exp[20];
char *e;
int n1,n2,n3,num;
printf("Enter the expression :");
scanf("%s",exp);
e=exp;
while(*e!='\0')
{
if(isdigit(*e))
{
num=*e-48;
push(num);
}
else
{
n1=pop();
n2=pop();
switch(*e)
{
case '+':
{
n3=n1+n2;
break;
}
case '-':
{
n3=n1-n2;
break;
}
case '*':
{
n3=n1*n2;
break;
}
case '/':
{
n3=n1/n2;
break;
}
}
push(n3);
}
e++;
}
printf("\n The result of expression %s = %d\n\n",exp,pop());
}
```

OUTPUT:



```
"C:\Users\Admin\Desktop\evaluation of postfix expression.exe"
Enter the expression :45*32-

 The result of expression 45*32- = -1


Process returned 0 (0x0)   execution time : 7.802 s
Press any key to continue.
```

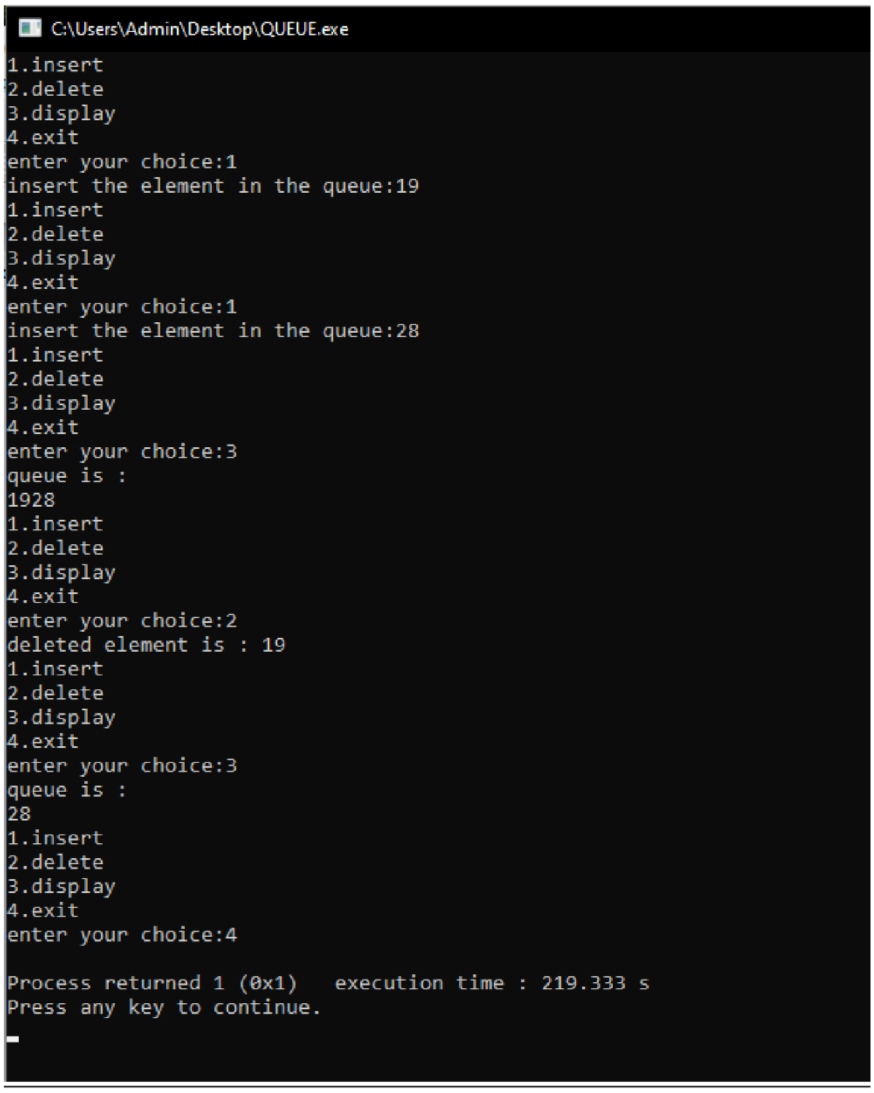
LAB -3

PROGRAM 3(a) : QUEUE IMPLEMENTATION

SOURCE CODE:

```
#include<stdio.h>
#define MAX 50
int queue_array[MAX];
int rear=-1;
int front=-1;
display()
{
  int i;
  if(front==-1)
    printf("queue is empty\n");
else
{
  printf("queue is :\n");
  for(i=front;i<=rear;i++)
    printf("%d",queue_array[i]);
  printf("\n");
}
}
main()
{
  int choice;
  while(1)
  {
    printf("1.insert\n");
    printf("2.delete\n");
    printf("3.display\n");
    printf("4.exit\n");
    printf("enter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      insert();
      break;
      case 2:
      delete();
      break;
      case 3:
      display();
      break;
      case 4:
      exit(1);
break;
      default:
      printf("invalid choice\n");
    }
  }
}
insert()
{
  int add_item;
  if(rear==MAX-1)
    printf("queue overflow\n");
  else
  {
    if(front==-1)
    front=0;
    printf("insert the element in the queue:");
    scanf("%d",&add_item);
    rear+=1;
    queue_array[rear]=add_item;
   }
}
delete()
{
  if(front==-1 || front>rear)
  {
    printf("queue underflow\n");
    return;
  }
  else
  {
    printf("deleted element is : %d\n",queue_array[front]);
    front+=1;
  }
}
```

OUTPUT:



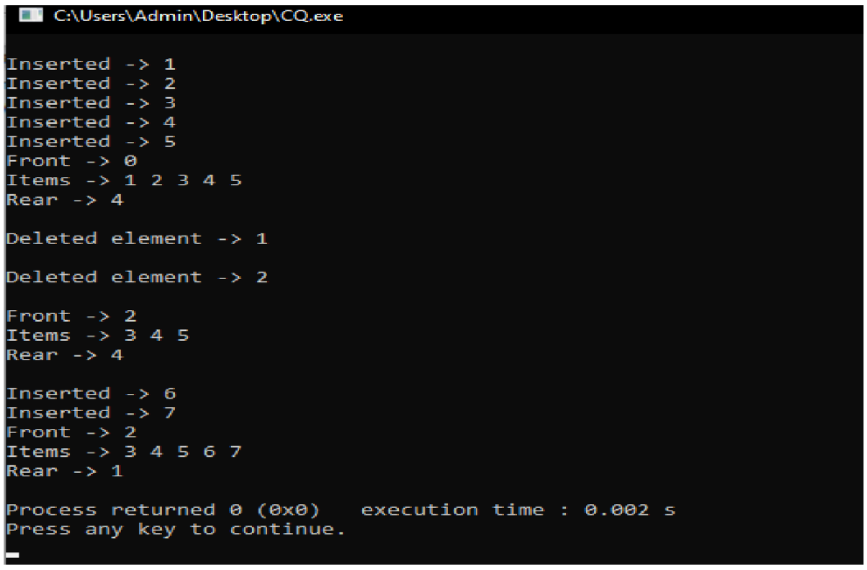
```
1.insert
2.delete
3.display
4.exit
enter your choice:1
insert the element in the queue:19
1.insert
2.delete
3.display
4.exit
enter your choice:1
insert the element in the queue:28
1.insert
2.delete
3.display
4.exit
enter your choice:3
queue is :
1928
1.insert
2.delete
3.display
4.exit
enter your choice:2
deleted element is : 19
1.insert
2.delete
3.display
4.exit
enter your choice:3
queue is :
28
1.insert
2.delete
3.display
4.exit
enter your choice:4

Process returned 1 (0x1)   execution time : 219.333 s
Press any key to continue.
```

PROGRAM 3(b) : CIRCULAR QUEUE

SOURCE CODE:

```
#include<stdio.h>

#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;

int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
    return 1;
  return 0;
}

int isEmpty() {
 if (front == -1)
       return 1;
   return 0;
}

void enQueue(int element) {
   if (isFull())
      printf("\nQueue is full");
   else {
      if (front == -1)
         front = 0;
      rear = (rear + 1) % SIZE;
      items[rear] = element;
      printf("\nInserted -> %d", element);
   }
}

int deQueue() {
   int element;
   if (isEmpty()) {
      printf("\nQueue is empty");
      return -1;
   } else {
      element = items[front];
      if (front == rear) {
         front = -1;
         rear = -1;
      } else {
         front = (front + 1) % SIZE;
      }
      printf("\nDeleted element -> %d\n", element);
      return element;
   }
}

void display() {
   int i;
   if (isEmpty())
      printf("\nEmpty queue\n");
   else {
      printf("\nFront -> %d", front);
      printf("\nItems -> ");
      for (i = front; i != rear; i = (i + 1) % SIZE) {
         printf("%d ", items[i]);
      }
      printf("%d", items[i]);
      printf("\nRear -> %d\n", rear);
}
}

int main() {
   enQueue(1);
   enQueue(2);
   enQueue(3);
   enQueue(4);
   enQueue(5);

   display();

   deQueue();
   deQueue();

   display();

   enQueue(6);
   enQueue(7);

   display();

   return 0;
}
```

OUTPUT:



```
C:\Users\Admin\Desktop\CQ.exe

Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Front -> 0
Items -> 1 2 3 4 5
Rear -> 4

Deleted element -> 1

Deleted element -> 2

Front -> 2
Items -> 3 4 5
Rear -> 4

Inserted -> 6
Inserted -> 7
Front -> 2
Items -> 3 4 5 6 7
Rear -> 1

Process returned 0 (0x0)   execution time : 0.002 s
Press any key to continue.
```

LAB -4

PROGRAM 1 : Singly Linked List Insert and display Implementation

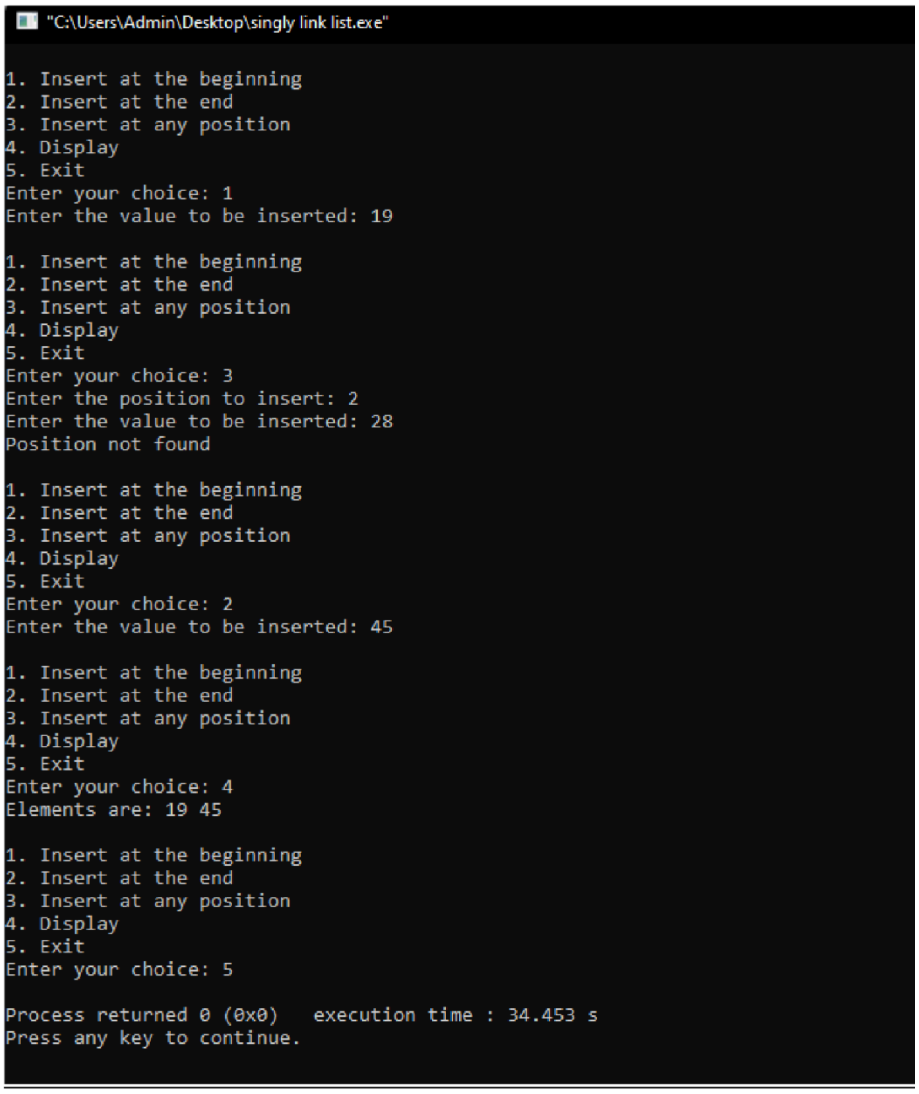
SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
void display() {
   struct node *ptr = head;
   if (ptr == NULL) {
      printf("List is empty\n");
      return;
   }
   printf("Elements are: ");
   while (ptr != NULL) {
      printf("%d ", ptr->data);
      ptr = ptr->next;
   }
   printf("\n");
}
void insert_begin() {
   struct node *temp;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter the value to be inserted: ");
   scanf("%d", &temp->data);
   temp->next = head;
   head = temp;
}
void insert_end() {
   struct node *temp, *ptr;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter the value to be inserted: ");
   scanf("%d", &temp->data);
   temp->next = NULL;
   if (head == NULL) {
      head = temp;
   } else {
      ptr = head;
      while (ptr->next != NULL) {
         ptr = ptr->next;
      }
```

```
ptr->next = temp;

  }
}
void insert_pos() {
   int pos, i;
   struct node *temp, *ptr;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter the position to insert: ");
   scanf("%d", &pos);
   printf("Enter the value to be inserted: ");
   scanf("%d", &temp->data);
   temp->next = NULL;

   if (pos == 0) {
      temp->next = head;
      head = temp;
   } else {
      ptr = head;
      for (i = 0; i < pos - 1; i++) {
         ptr = ptr->next;
         if (ptr == NULL) {
            printf("Position not found\n");
            return;
         }
      }
      temp->next = ptr->next;
      ptr->next = temp;
   }
}
int main() {
   int choice;
   while(1) {
      printf("\n1. Insert at the beginning\n2. Insert at the end\n3. Insert at any position\n4.
Display\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch(choice) {
         case 1:
            insert_begin();
            break;
         case 2:
            insert_end();
            break;
         case 3:
```

```
insert_pos();

 break;
           case 4:
               display();
               break;
           case 5:
               exit(0);
               break;
           default:
               printf("Enter the correct choice\n");
       }
   }
   return 0;
}
```

OUTPUT:



"C:\Users\Admin\Desktop\singly link list.exe"

```
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 1
Enter the value to be inserted: 19

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 3
Enter the position to insert: 2
Enter the value to be inserted: 28
Position not found

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 2
Enter the value to be inserted: 45

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 4
Elements are: 19 45

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Display
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 34.453 s
Press any key to continue.
```

LAB -5

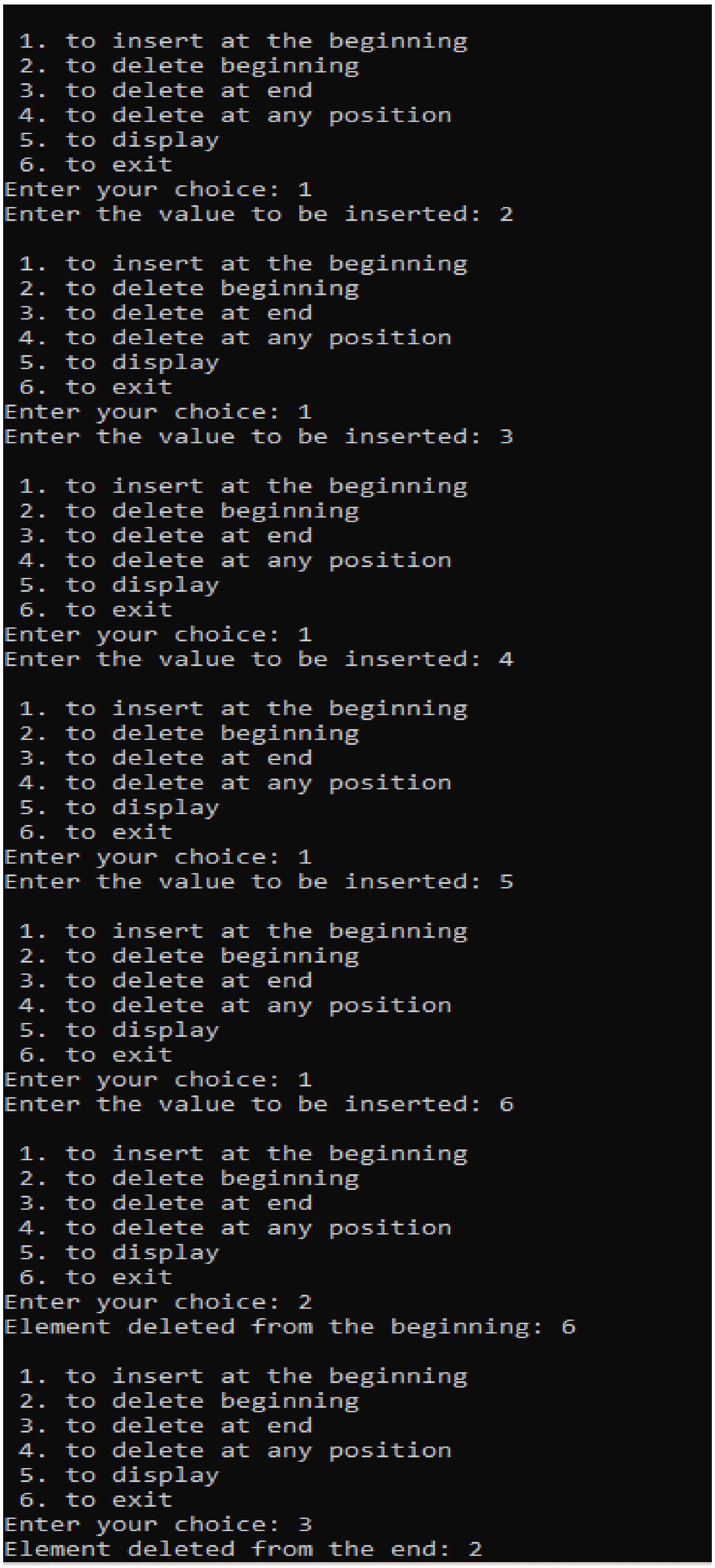
PROGRAM 1 : Singly Linked List Delete and display Implementation

SOURCE CODE:

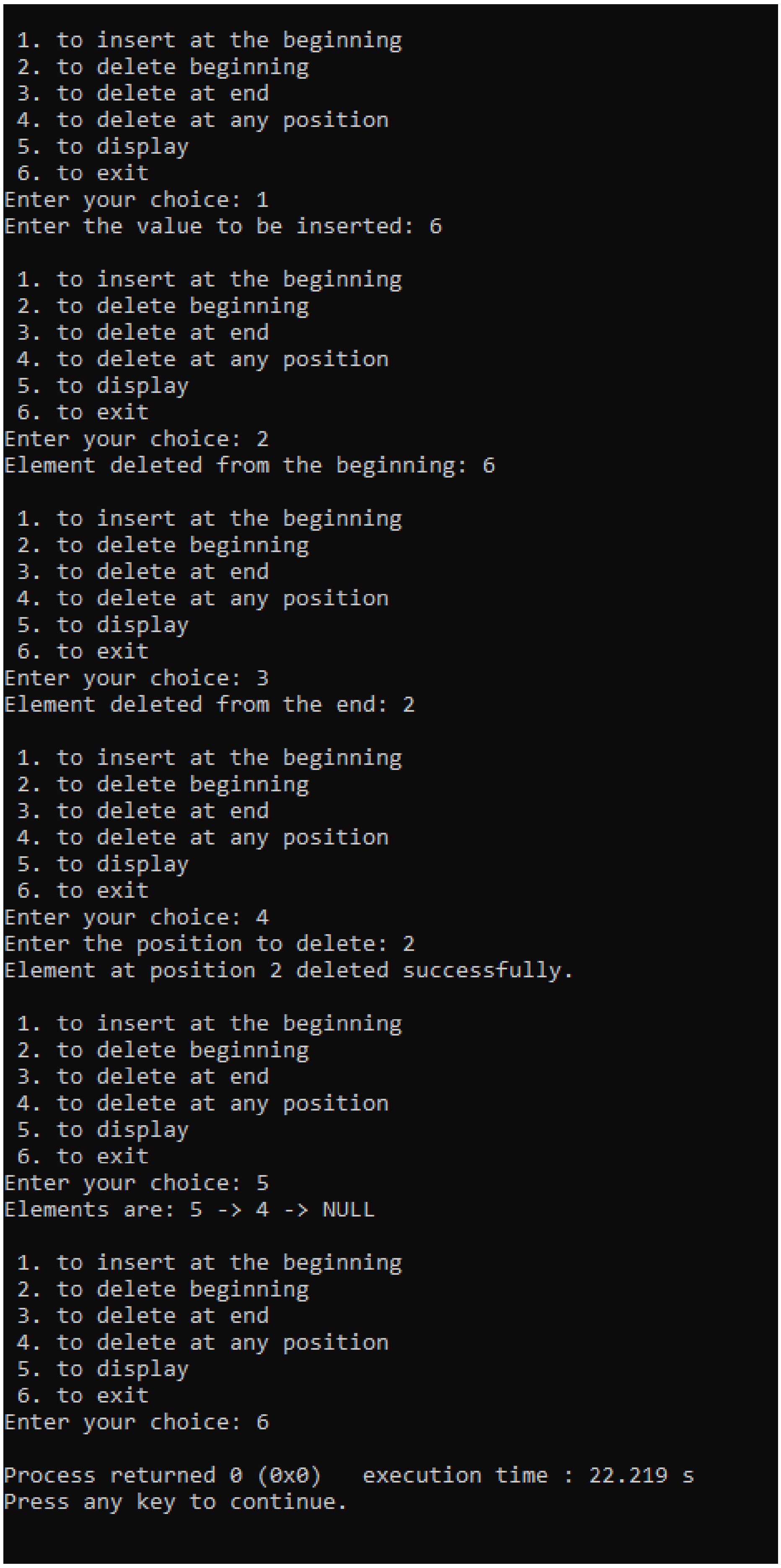
```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
void display() {
   printf("Elements are: ");
   struct node *ptr = head;
   while (ptr != NULL) {
       printf("%d -> ", ptr->data);
       ptr = ptr->next;
   }
   printf("NULL\n");
}
void insert_begin() {
   struct node *temp = (struct node*)malloc(sizeof(struct node));
   printf("Enter the value to be inserted: ");
   scanf("%d", &temp->data);
   temp->next = head;
   head = temp;
}
void delete_begin() {
   if (head == NULL) {
       printf("List is empty. Deletion not possible.\n");
       return;
}
   struct node *temp = head;
   head = head->next;
   printf("Element deleted from the beginning: %d\n", temp->data);
   free(temp);
}
void delete_end() {
   if (head == NULL) {
      printf("List is empty. Deletion not possible.\n");
      return;
   }
   struct node *temp, *prev;
   temp = head;
   while (temp->next != NULL) {
      prev = temp;
      temp = temp->next;
   }
   if (temp == head) {
      head = NULL;
   } else {
      prev->next = NULL;
   }
   printf("Element deleted from the end: %d\n", temp->data);
   free(temp);
}
void delete_at_position() {
   int position;
   printf("Enter the position to delete: ");
   scanf("%d", &position);
```

```
 if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    struct node *temp, *prev;
    temp = head;
    if (position == 0) {
        head = head->next;
        printf("Element at position %d deleted successfully.\n", position);
        free(temp);
        return;
    }
    for (int i = 0; temp != NULL && i < position; i++) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position %d is out of bounds.\n", position);
        return;
    }
    prev->next = temp->next;
    printf("Element at position %d deleted successfully.\n", position);
    free(temp);
}
int main() {
    int choice;
    while (1) {
        printf("\n 1. to insert at the beginning\n 2. to delete beginning\n 3. to delete at end\n 4. to
delete at any position\n 5. to display\n 6. to exit\n");
        printf("Enter your choice: ");
scanf("%d", &choice);
        switch (choice) {
            case 1:
                insert_begin();
                break;
            case 2:
                delete_begin();
                break;
            case 3:
                delete_end();
                break;
            case 4:
                delete_at_position();
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
                break;
            default:
                printf("Enter the correct choice\n");
                break;
        }
    }
    return 0;
}
```

OUTPUT:



```
 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 2

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 3

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 4

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 5

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 6

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 2
Element deleted from the beginning: 6

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 3
Element deleted from the end: 2
```



```
 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 1
Enter the value to be inserted: 6

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 2
Element deleted from the beginning: 6

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 3
Element deleted from the end: 2

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 4
Enter the position to delete: 2
Element at position 2 deleted successfully.

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 5
Elements are: 5 -> 4 -> NULL

 1. to insert at the beginning
 2. to delete beginning
 3. to delete at end
 4. to delete at any position
 5. to display
 6. to exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 22.219 s
Press any key to continue.
```

LAB -6

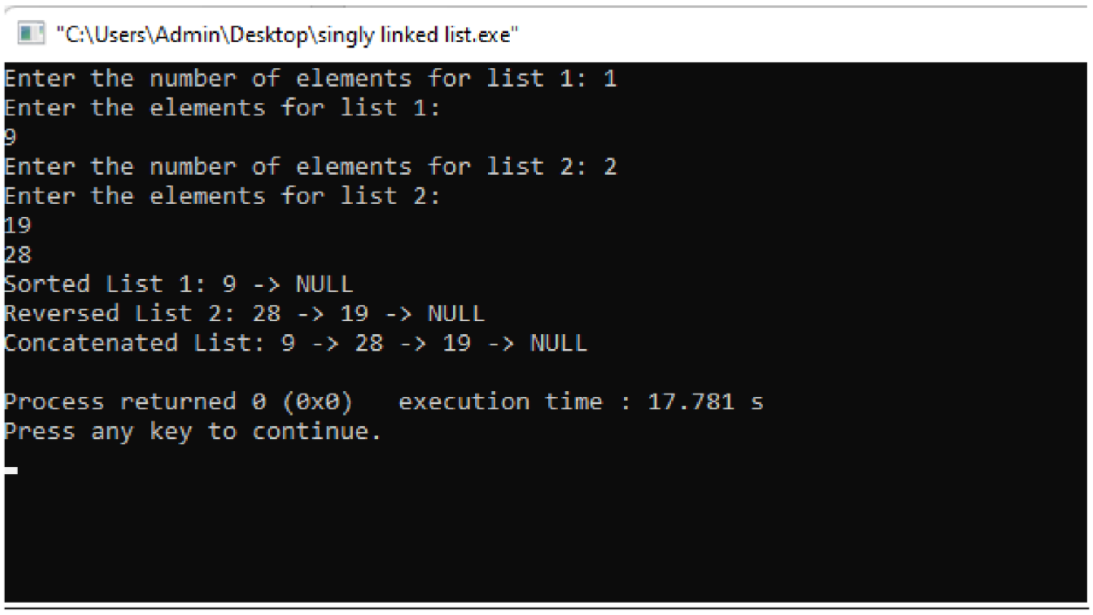
PROGRAM 1: WAP to Implement Single Linked List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* next;
};
struct Node* createNode(int value) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode->next = NULL;
  return newNode;
}
void insertEnd(struct Node** head, int value) {
  struct Node* newNode = createNode(value);
  if (*head == NULL) {
    *head = newNode;
  } else {
    struct Node* temp = *head;
    while (temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
  }
}
void display(struct Node* head) {
  struct Node* temp = head;
  while (temp != NULL) {
    printf("%d -> ", temp->data);
    temp = temp->next;
  }
  printf("NULL\n");
}
void sortLinkedList(struct Node* head) {
  int swapped, i;
  struct Node* ptr;
  struct Node* lptr = NULL;
  if (head == NULL)
    return;
  do {
    swapped = 0;
    ptr = head;
while (ptr->next != lptr) {
            if (ptr->data > ptr->next->data) {
                int temp = ptr->data;
                ptr->data = ptr->next->data;
                ptr->next->data = temp;
                swapped = 1;
            }
            ptr = ptr->next;
        }
        lptr = ptr;
    } while (swapped);
}
struct Node* reverseLinkedList(struct Node* head) {
    struct Node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void concatenateLinkedLists(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
    } else {
        struct Node* temp = *list1;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = list2;
    }
}
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n, value;
    printf("Enter the number of elements for list 1: ");
    scanf("%d", &n);
    printf("Enter the elements for list 1:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertEnd(&list1, value);
    }
    printf("Enter the number of elements for list 2: ");
    scanf("%d", &n);
    printf("Enter the elements for list 2:\n");
```

```
for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertEnd(&list2, value);
    }
    sortLinkedList(list1);
    printf("Sorted List 1: ");
    display(list1);
    list2 = reverseLinkedList(list2);
    printf("Reversed List 2: ");
    display(list2);
    concatenateLinkedLists(&list1, list2);
    printf("Concatenated List: ");
    display(list1);
    struct Node* temp;
    while (list1 != NULL) {
        temp = list1;
        list1 = list1->next;
        free(temp);
    }
    return 0;
}
```

<u>OUTPUT:</u>



"C:\Users\Admin\Desktop\singly linked list.exe"

```
Enter the number of elements for list 1: 1
Enter the elements for list 1:
9
Enter the number of elements for list 2: 2
Enter the elements for list 2:
19
28
Sorted List 1: 9 -> NULL
Reversed List 2: 28 -> 19 -> NULL
Concatenated List: 9 -> 28 -> 19 -> NULL

Process returned 0 (0x0)   execution time : 17.781 s
Press any key to continue.
```

<u>**LAB -7**</u>

<u>PROGRAM 1</u>: WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

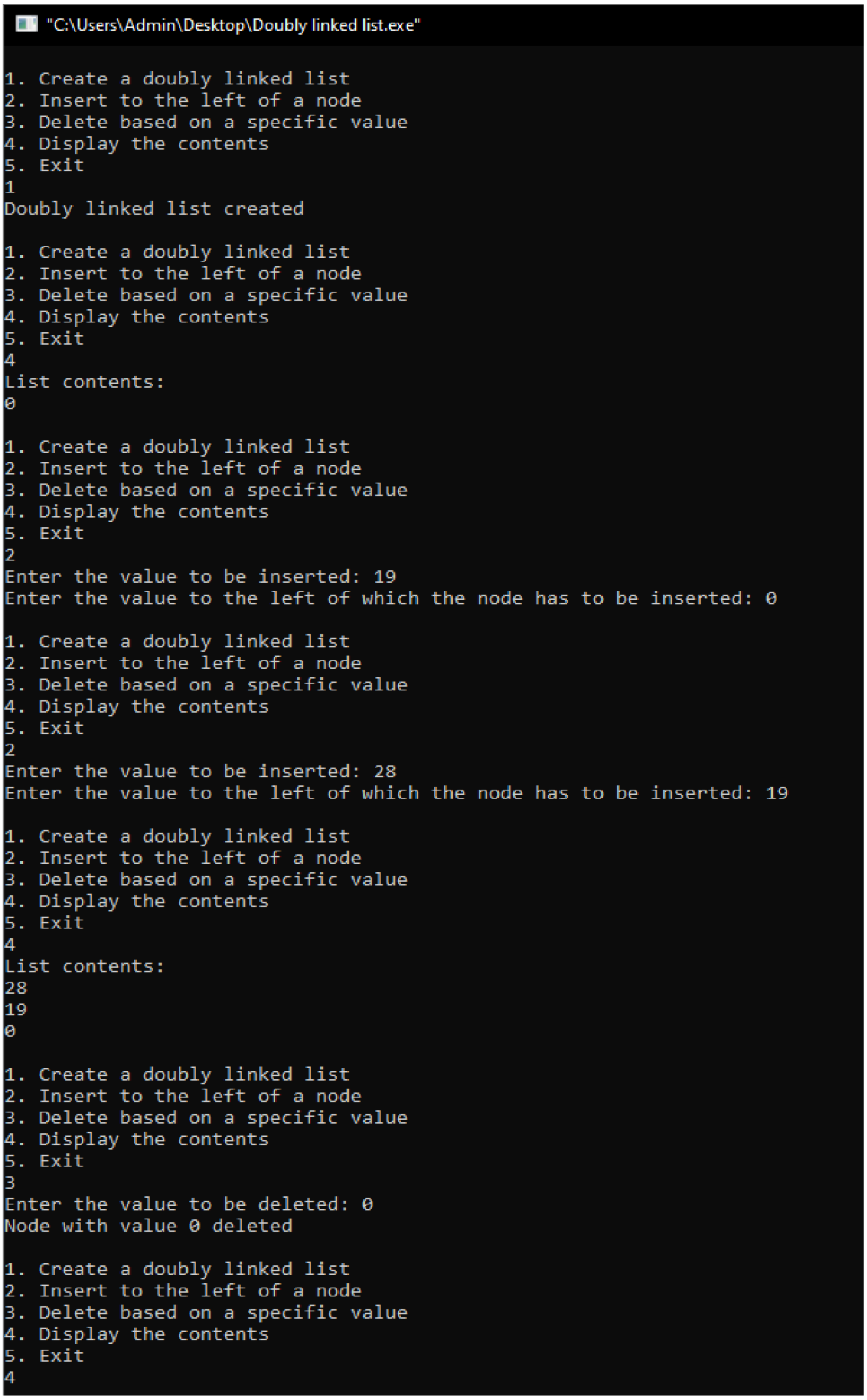
<u>**SOURCE CODE:**</u>

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *prev;
   struct node *next;
};
struct node *s1 = NULL;
struct node *createNode(int value) {
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   temp->data = value;
   temp->next = NULL;
   temp->prev = NULL;
   return temp;
}
struct node *insert_left(struct node *start) {
   int value, key;
   struct node *temp = createNode(0);
   printf("Enter the value to be inserted: ");
   scanf("%d", &temp->data);
   printf("Enter the value to the left of which the node has to be inserted: ");
   scanf("%d", &key);
   struct node *ptr = start;
   while (ptr != NULL && ptr->data != key) {
      ptr = ptr->next;
   }
   if (ptr == NULL) {
      printf("Node with value %d not found\n", key);
      free(temp);
   } else {
      temp->next = ptr;
      temp->prev = ptr->prev;
      if (ptr->prev != NULL) {
         ptr->prev->next = temp;
      }
      ptr->prev = temp;
      if (ptr == start) {
         start = temp;
```

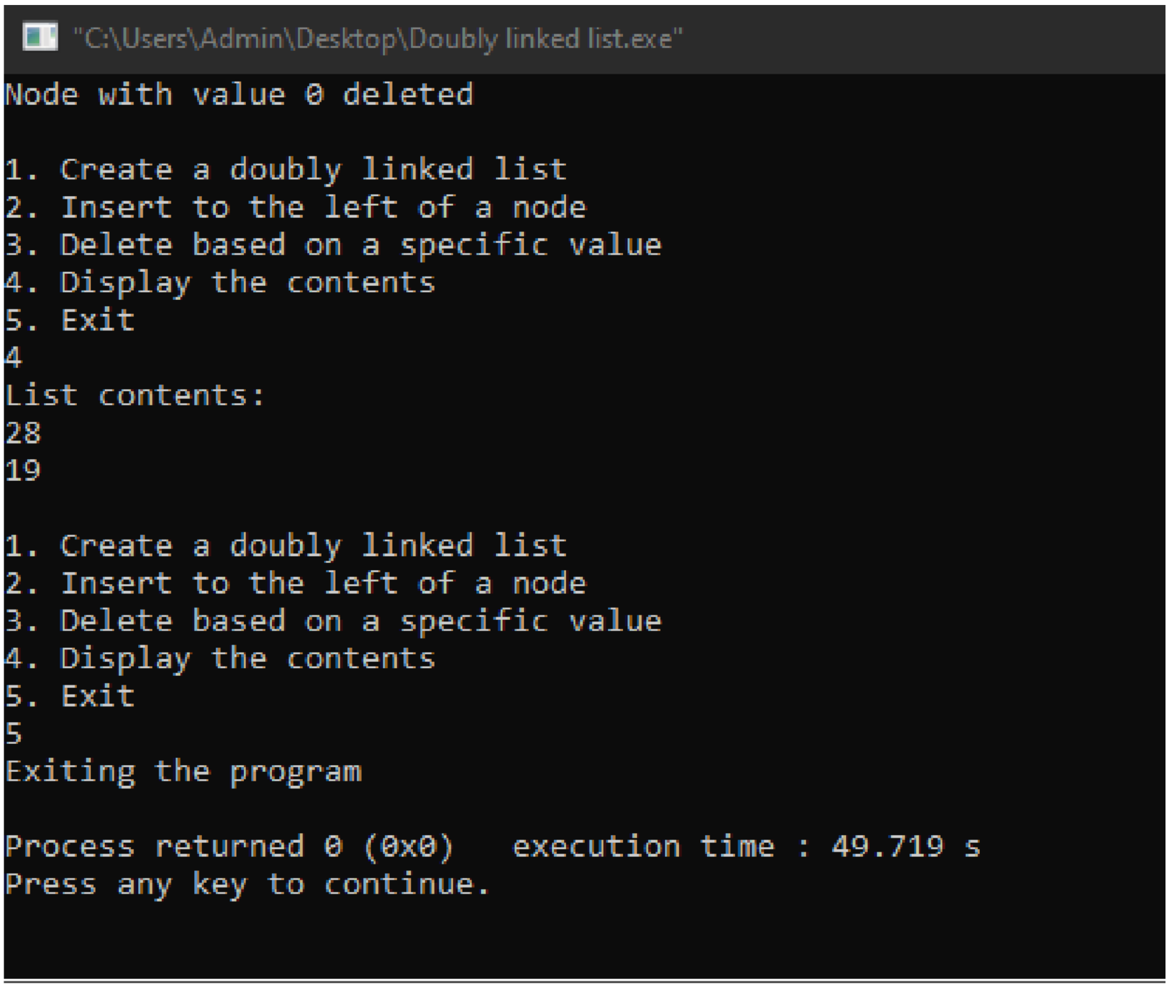
```
}
  }
  return start;
}
struct node *delete_value(struct node *start) {
  int value;
  printf("Enter the value to be deleted: ");
  scanf("%d", &value);
  struct node *ptr = start;
  while (ptr != NULL && ptr->data != value) {
    ptr = ptr->next;
  }
  if (ptr == NULL) {
    printf("Node with value %d not found\n", value);
  } else {
    if (ptr->prev != NULL) {
      ptr->prev->next = ptr->next;
    } else {
      start = ptr->next;
    }
    if (ptr->next != NULL) {
      ptr->next->prev = ptr->prev;
    }
    printf("Node with value %d deleted\n", value);
    free(ptr);
  }
  return start;
}
void display(struct node *start) {
  struct node *ptr = start;
  if (start == NULL) {
    printf("List is empty\n");
  } else {
    printf("List contents:\n");
    while (ptr != NULL) {
      printf("%d\n", ptr->data);
      ptr = ptr->next;
    }
  }
}
int main() {
  int choice;
  while (1) {
    printf("\n1. Create a doubly linked list\n2. Insert to the left of a node\n3. Delete based on a
specific value\n4. Display the contents\n5. Exit\n");
    scanf("%d", &choice);
    switch (choice) {
```

```
 case 1:
              s1 = createNode(0);
              printf("Doubly linked list created\n");
              break;
          case 2:
              s1 = insert_left(s1);
              break;
          case 3:
              s1 = delete_value(s1);
              break;
          case 4:
              display(s1);
              break;
          case 5:
              printf("Exiting the program\n");
              exit(0);
          default:
              printf("Invalid choice\n");
      }
   }
   return 0;
}
```

OUTPUT:



```
"C:\Users\Admin\Desktop\Doubly linked list.exe"

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
1
Doubly linked list created

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
4
List contents:
0

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
2
Enter the value to be inserted: 19
Enter the value to the left of which the node has to be inserted: 0

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
2
Enter the value to be inserted: 28
Enter the value to the left of which the node has to be inserted: 19

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
4
List contents:
28
19
0

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
3
Enter the value to be deleted: 0
Node with value 0 deleted

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
4
```



"C:\Users\Admin\Desktop\Doubly linked list.exe"

```
Node with value 0 deleted

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
4
List contents:
28
19

1. Create a doubly linked list
2. Insert to the left of a node
3. Delete based on a specific value
4. Display the contents
5. Exit
5
Exiting the program

Process returned 0 (0x0)   execution time : 49.719 s
Press any key to continue.
```

LAB -8

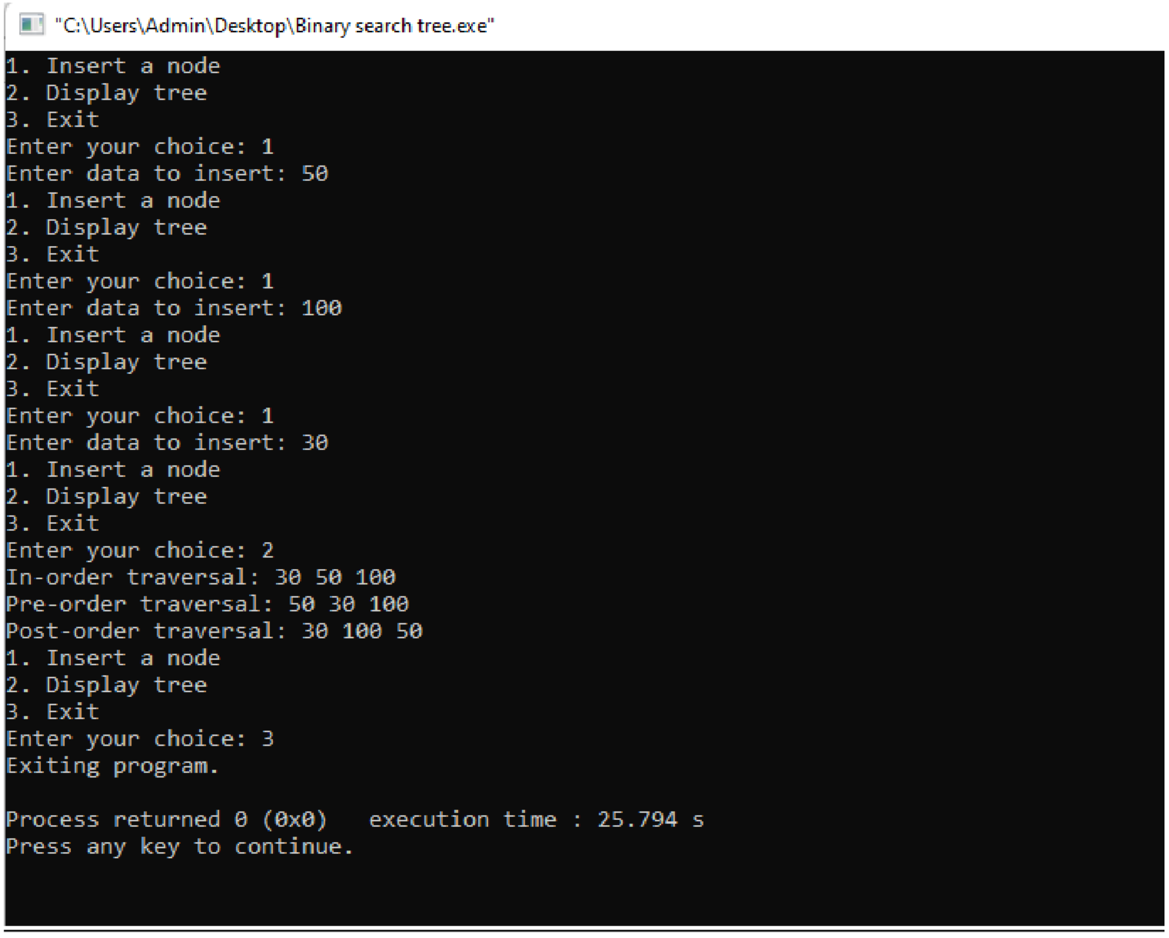
PROGRAM 1: Write a program
a. To construct a binary Search tree.
b. To traverse the tree using all the methods i.e., in-order, preorder and postorder
c. To display the elements in the tree.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
   int data;
   struct TreeNode* left;
   struct TreeNode* right;
};
struct TreeNode* createNode(int data) {
   struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}
struct TreeNode* insertNode(struct TreeNode* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insertNode(root->left, data);
   } else if (data > root->data) {
      root->right = insertNode(root->right, data);
   }
   return root;
}
void inOrderTraversal(struct TreeNode* root) {
   if (root != NULL) {
      inOrderTraversal(root->left);
      printf("%d ", root->data);
      inOrderTraversal(root->right);
   }
}
void preOrderTraversal(struct TreeNode* root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preOrderTraversal(root->left);
      preOrderTraversal(root->right);
   }
}
void postOrderTraversal(struct TreeNode* root) {
if (root != NULL) {
      postOrderTraversal(root->left);
      postOrderTraversal(root->right);
      printf("%d ", root->data);
   }
}
void displayTree(struct TreeNode* root) {
   printf("In-order traversal: ");
   inOrderTraversal(root);
   printf("\n");
   printf("Pre-order traversal: ");
   preOrderTraversal(root);
   printf("\n");
   printf("Post-order traversal: ");
   postOrderTraversal(root);
   printf("\n");
}
int main() {
   struct TreeNode* root = NULL;
   int choice, data;
   do {
      printf("1. Insert a node\n");
      printf("2. Display tree\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
          case 1:
              printf("Enter data to insert: ");
              scanf("%d", &data);
              root = insertNode(root, data);
              break;
          case 2:
              if (root == NULL) {
                  printf("Tree is empty.\n");
              } else {
                  displayTree(root);
              }
              break;
          case 3:
              printf("Exiting program.\n");
              break;
          default:
              printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 3);
   return 0;
```

```
}
```

OUTPUT:



"C:\Users\Admin\Desktop\Binary search tree.exe"

```
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 1
Enter data to insert: 50
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 1
Enter data to insert: 100
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 1
Enter data to insert: 30
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 2
In-order traversal: 30 50 100
Pre-order traversal: 50 30 100
Post-order traversal: 30 100 50
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 3
Exiting program.

Process returned 0 (0x0)   execution time : 25.794 s
Press any key to continue.
```

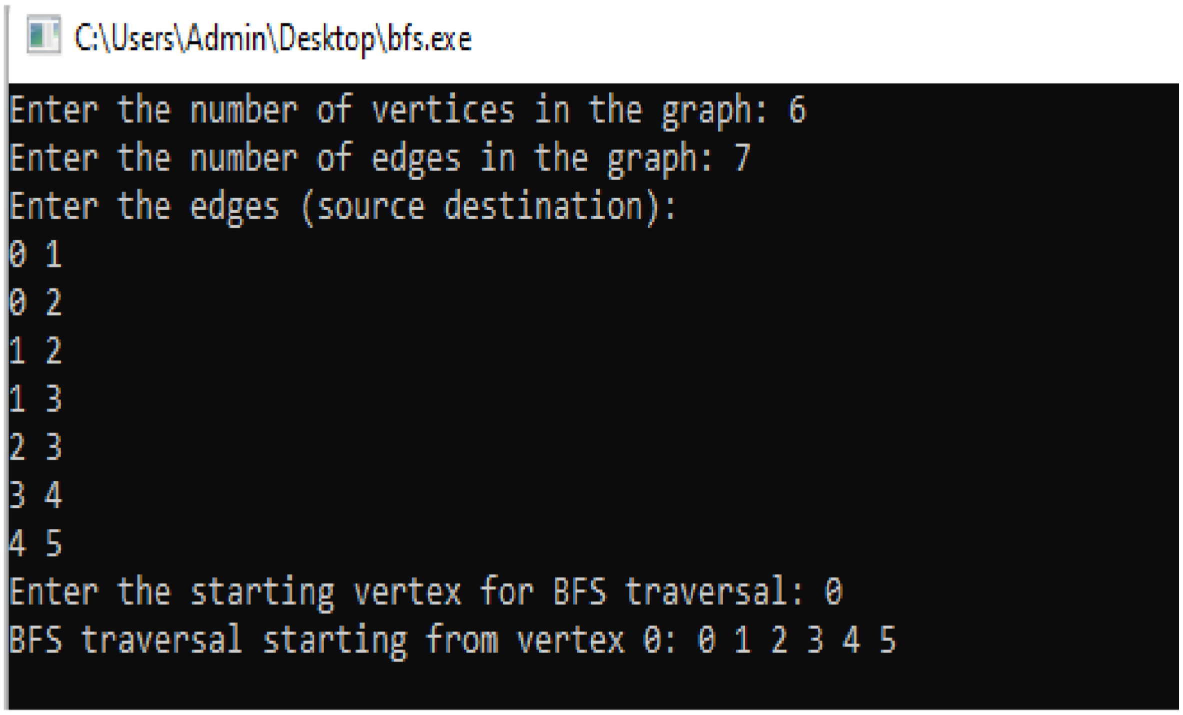
LAB -9

PROGRAM 1: Write a program to traverse a graph using BFS method.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void BFS(int adjacency_matrix[MAX][MAX], int visited[MAX], int n, int start) {
int queue[MAX], front = -1, rear = -1;
visited[start] = 1;
queue[++rear] = start;
while (front != rear) {
int current = queue[++front];
printf("%d ", current + 1);
for (int i = 0; i < n; i++) {
if (adjacency_matrix[current][i] == 1 && !visited[i]) {
visited[i] = 1;
queue[++rear] = i;
}
}
}
}
int main() {
int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;
printf("Enter the number of nodes: ");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for (i = 0; i < n; i++) {
visited[i] = 0; // Initialize visited array
for (j = 0; j < n; j++) {
scanf("%d", &adjacency_matrix[i][j]);
}
}
int start_node;
printf("Enter the starting node for BFS traversal (1 to %d): ", n);
scanf("%d", &start_node);
printf("BFS Traversal starting from node %d: ", start_node);
BFS(adjacency_matrix, visited, n, start_node - 1);
return 0;
}
```

OUTPUT:


C:\Users\Admin\Desktop\bfs.exe

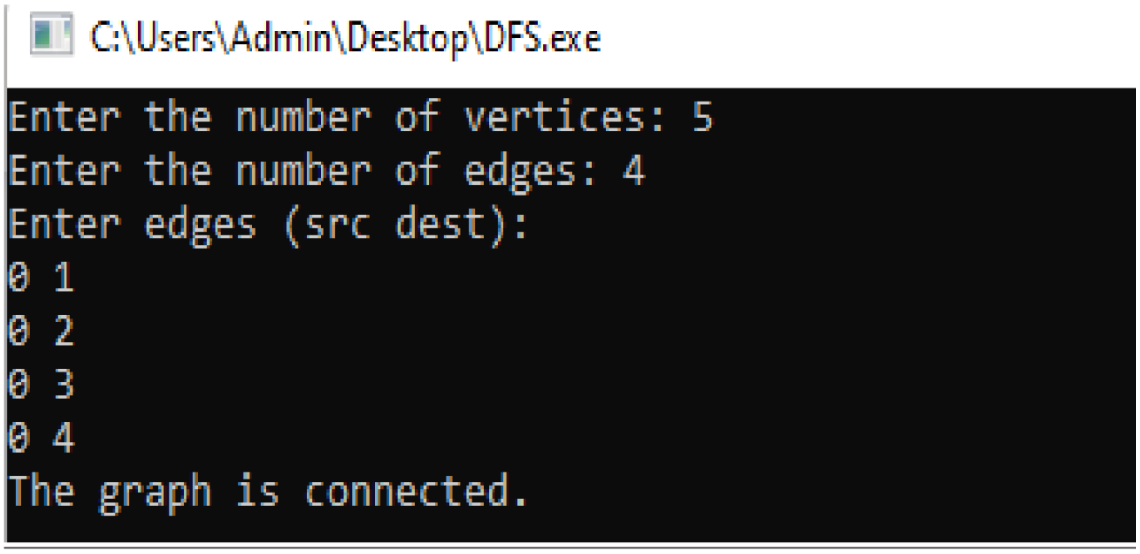
```
Enter the number of vertices in the graph: 6
Enter the number of edges in the graph: 7
Enter the edges (source destination):
0 1
0 2
1 2
1 3
2 3
3 4
4 5
Enter the starting vertex for BFS traversal: 0
BFS traversal starting from vertex 0: 0 1 2 3 4 5
```

PROGRAM 2: Write a program to check whether given graph is connected or not using DFS method.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void DFS(int adjacency_matrix[MAX][MAX], int visited[MAX], int n, int current) {
visited[current] = 1;
for (int i = 0; i < n; i++) {
if (adjacency_matrix[current][i] == 1 && !visited[i]) {
DFS(adjacency_matrix, visited, n, i);
}
}
}
int isConnected(int adjacency_matrix[MAX][MAX], int visited[MAX], int n) {
for (int i = 0; i < n; i++) {
visited[i] = 0;
}
DFS(adjacency_matrix, visited, n, 0);
for (int i = 0; i < n; i++) {
if (!visited[i]) {
return 0;
}
}
return 1;
}
int main() {
int adjacency_matrix[MAX][MAX], visited[MAX], n, i, j;
printf("Enter the number of nodes: ");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
scanf("%d", &adjacency_matrix[i][j]);
}
}
if (isConnected(adjacency_matrix, visited, n)) {
printf("The graph is connected.\n");
} else {
printf("The graph is not connected.\n");
}
return 0;
}
```

OUTPUT:



C:\Users\Admin\Desktop\DFS.exe

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter edges (src dest):
0 1
0 2
0 3
0 4
The graph is connected.
```

LAB -10

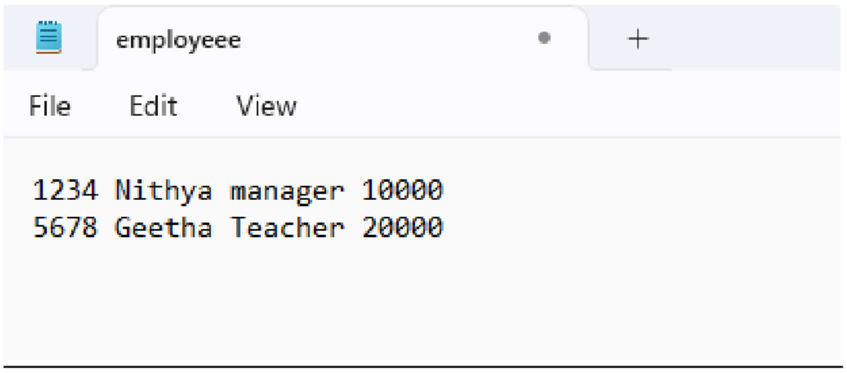
**PROGRAM 1: Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.
Resolve the collision (if any) using linear probing.**

SOURCE CODE:

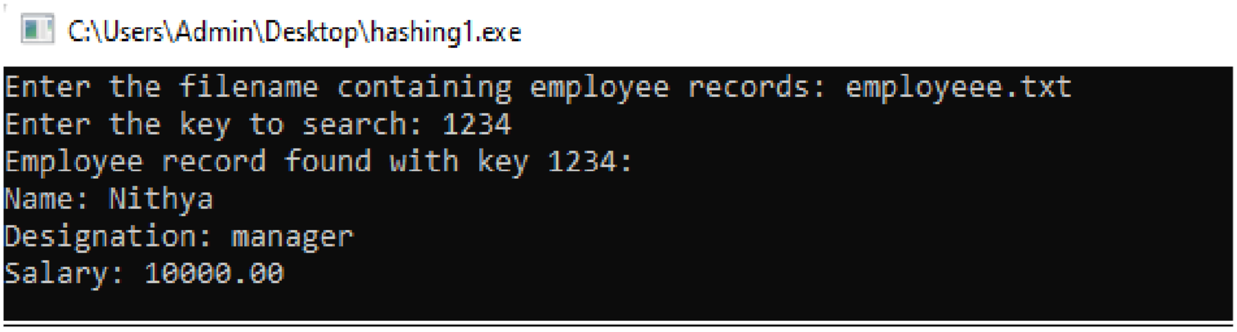
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 100
#define KEY_LENGTH 5
#define MAX_NAME_LENGTH 50
#define MAX_DESIGNATION_LENGTH 50
struct Employee {
   char key[KEY_LENGTH];
   char name[MAX_NAME_LENGTH];
   char designation[MAX_DESIGNATION_LENGTH];
   float salary;
};
struct HashTable {
   struct Employee* table[TABLE_SIZE];
};
int hash_function(const char* key, int m) {
   int sum = 0;
   for (int i = 0; key[i] != '\0'; i++) {
      sum += key[i];
   }
   return sum % m;
}
void insert(struct HashTable* ht, struct Employee* emp) {
   int index = hash_function(emp->key, TABLE_SIZE);

   while (ht->table[index] != NULL) {
      index = (index + 1) % TABLE_SIZE;
   }
   ht->table[index] = emp;
}
struct Employee* search(struct HashTable* ht, const char* key) {
   int index = hash_function(key, TABLE_SIZE);

   while (ht->table[index] != NULL) {
      if (strcmp(ht->table[index]->key, key) == 0) {
```

```
 return ht->table[index];
        }
        index = (index + 1) % TABLE_SIZE;
    }
    return NULL;
}
int main() {
    struct HashTable ht;
    struct Employee* emp;
    char key[KEY_LENGTH];
    FILE* file;
    char filename[100];
    char line[100];
    for (int i = 0; i < TABLE_SIZE; i++) {
        ht.table[i] = NULL;
    }
    printf("Enter the filename containing employee records: ");
    scanf("%s", filename);
    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    while (fgets(line, sizeof(line), file)) {
        emp = (struct Employee*)malloc(sizeof(struct Employee));
        sscanf(line, "%s %s %s %f", emp->key, emp->name, emp->designation, &emp->salary);
        insert(&ht, emp);
    }
    fclose(file);
    printf("Enter the key to search: ");
    scanf("%s", key);
    emp = search(&ht, key);
    if (emp != NULL) {
        printf("Employee record found with key %s:\n", emp->key);
        printf("Name: %s\n", emp->name);
        printf("Designation: %s\n", emp->designation);
        printf("Salary: %.2f\n", emp->salary);
    } else {
        printf("Employee record not found for key %s\n", key);
    }
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (ht.table[i] != NULL) {
            free(ht.table[i]);
        }
    }
    return 0;
}
```

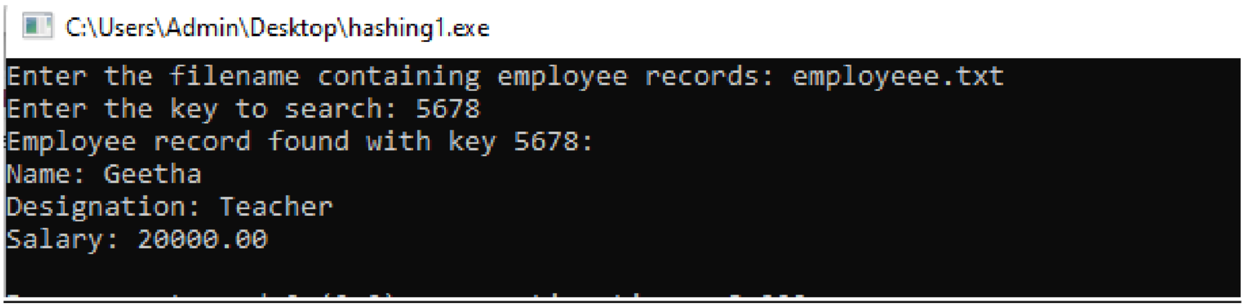
OUTPUT:



```
1234 Nithya manager 10000
5678 Geetha Teacher 20000
```



```
Enter the filename containing employee records: employeee.txt
Enter the key to search: 1234
Employee record found with key 1234:
Name: Nithya
Designation: manager
Salary: 10000.00
```



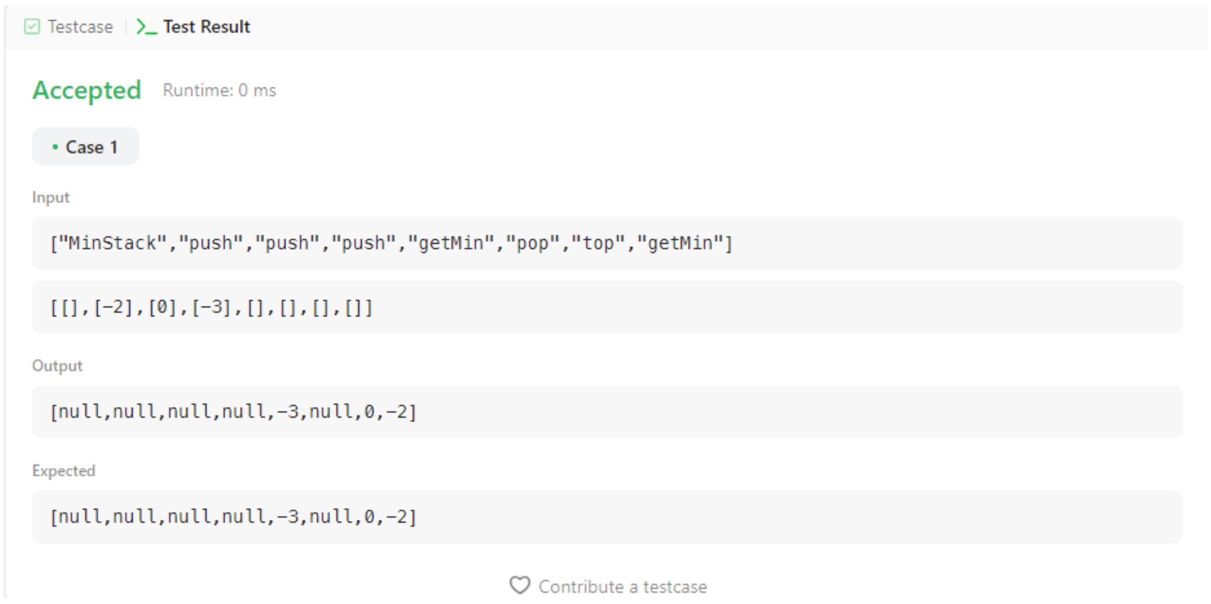
```
Enter the filename containing employee records: employeee.txt
Enter the key to search: 5678
Employee record found with key 5678:
Name: Geetha
Designation: Teacher
Salary: 20000.00
```

LEET CODE – 1 (STACK PROGRAM)

SOURCE CODE:

```
#include <stdlib.h>
typedef struct {
int *stack;
int *minStack;
int top
} MinStack;
MinStack* minStackCreate() {
MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
stack->stack = (int*)malloc(sizeof(int) * 50);
stack->minStack = (int*)malloc(sizeof(int) * 50);
stack->top = -1;
return stack;
}
void minStackPush(MinStack* obj, int val) {
obj->top++;
obj->stack[obj->top] = val;
if (obj->top == 0 || val <= obj->minStack[obj->top - 1]) {
obj->minStack[obj->top] = val;
} else {
obj->minStack[obj->top] = obj->minStack[obj->top - 1];
}
}
void minStackPop(MinStack* obj) {
obj->top--;
}
int minStackTop(MinStack* obj) {
return obj->stack[obj->top];
}
int minStackGetMin(MinStack* obj) {
return obj->minStack[obj->top];
}
void minStackFree(MinStack* obj) {
free(obj->stack);
free(obj->minStack);
free(obj);
}
/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
* minStackPush(obj, val);
 * minStackPop(obj);
 * int param_3 = minStackTop(obj);
 * int param_4 = minStackGetMin(obj);
 * minStackFree(obj);
*/
```

OUTPUT:



Accepted Runtime: 0 ms

Case 1

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

```
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

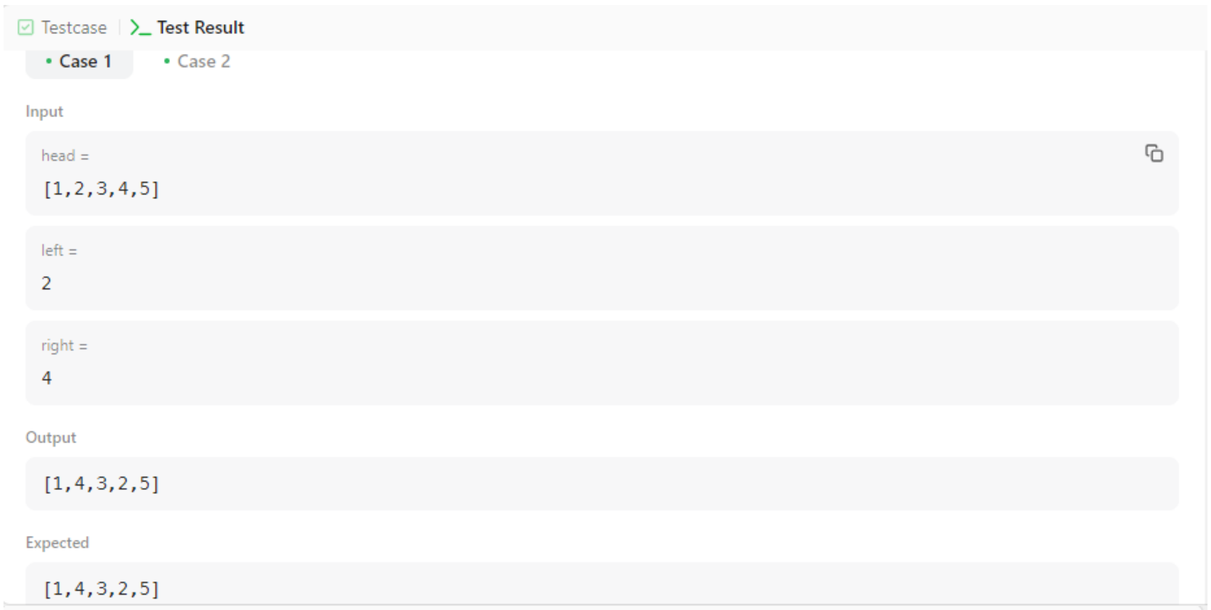
```
[null,null,null,null,-3,null,0,-2]
```

LEET CODE – 2 (REVERSE LINKED LIST ll)

SOURCE CODE:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* reverseBetween(struct ListNode* start, int a, int b)
{
a-=1;
b-=1;
struct ListNode *node1=NULL,*node2=NULL,*nodeb=NULL,*nodea=NULL,*ptr=start;
int c=0;
while(ptr!=NULL)
{
if(c==a-1)
nodeb=ptr;
else if(c==a)
node1=ptr;
else if(c==b)
node2=ptr;
else if(c==b+1)
{
nodea=ptr;
break;
}
c+=1;
ptr=ptr->next;
}
struct ListNode*pre=nodea,*temp;
ptr=start;
c=0;
while(ptr!=NULL)
{
if(c>=a && c<b)
{
temp=ptr->next;
ptr->next=pre;
pre=ptr;
ptr=temp;
}
else if(c==b)
{
ptr->next=pre;
if(a==0)
start=ptr;
else
nodeb->next=ptr;
break;
}
else
ptr=ptr->next;
c+=1;
}
return start;
}
```

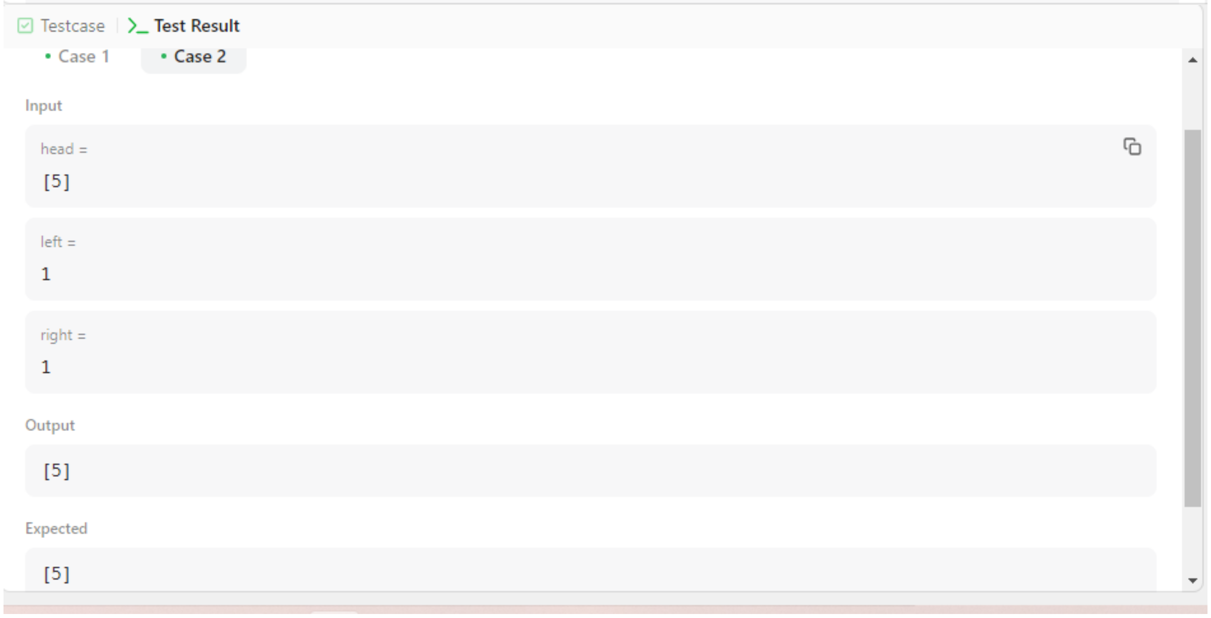
OUTPUT:



Testcase | Test Result

• Case 1 • Case 2

```
Input
head =
[1,2,3,4,5]
left =
2
right =
4
Output
[1,4,3,2,5]
Expected
[1,4,3,2,5]
```



Testcase | Test Result

• Case 1 • Case 2

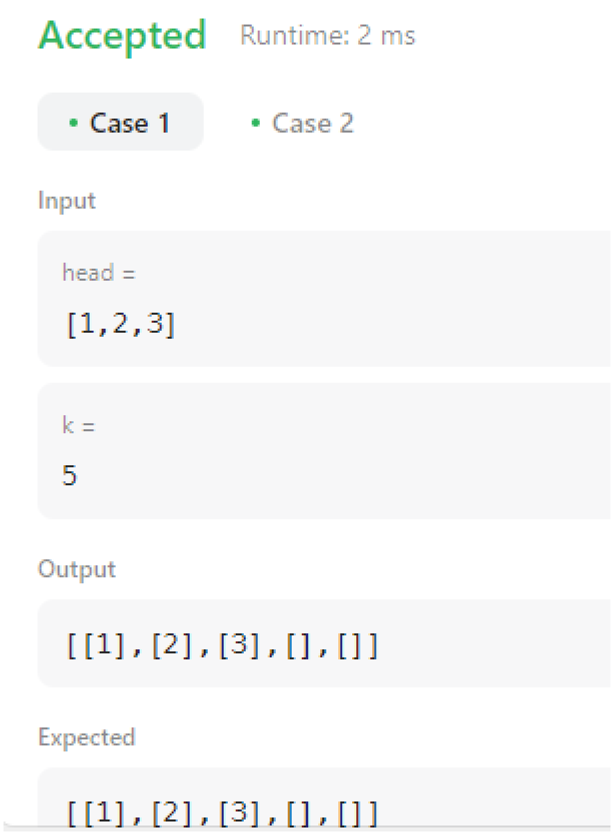
```
Input
head =
[5]
left =
1
right =
1
Output
[5]
Expected
[5]
```

LEET CODE – 3 (SPLIT LINKED LIST IN PARTS)

SOURCE CODE:

```
/** * Definition for singly-linked list.
* struct ListNode {
*    int val;
*    struct ListNode
*next;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
  struct ListNode* current = head;
  int length = 0;
  while (current) {
    length++;
    current = current->next;
  }
  int part_size = length / k;
  int extra_nodes = length % k;

  struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct ListNode*));
  current = head;
  for (int i = 0; i < k; i++) {
    struct ListNode* part_head = current;
    int part_length = part_size + (i < extra_nodes ? 1 : 0);
    for (int j = 0; j < part_length - 1 && current; j++) {
      current = current->next;
    }
    if (current) {
      struct ListNode* next_node = current->next;
      current->next = NULL;
      result[i] = part_head;
      current = next_node;
    } else {
      result[i] = NULL;
    }
  }
  *returnSize = k;
  return result;
}
```

OUTPUT:



Input

head =

```
[1,2,3]
```

k =

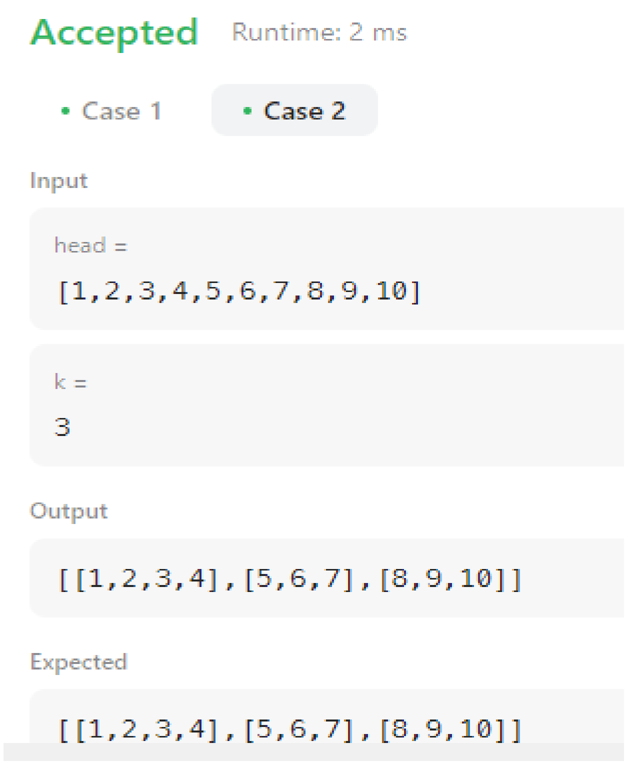
```
5
```

Output

```
[[1],[2],[3],[],[]]
```

Expected

```
[[1],[2],[3],[],[]]
```



Input

head =

```
[1,2,3,4,5,6,7,8,9,10]
```

k =

```
3
```

Output

```
[[1,2,3,4],[5,6,7],[8,9,10]]
```

Expected

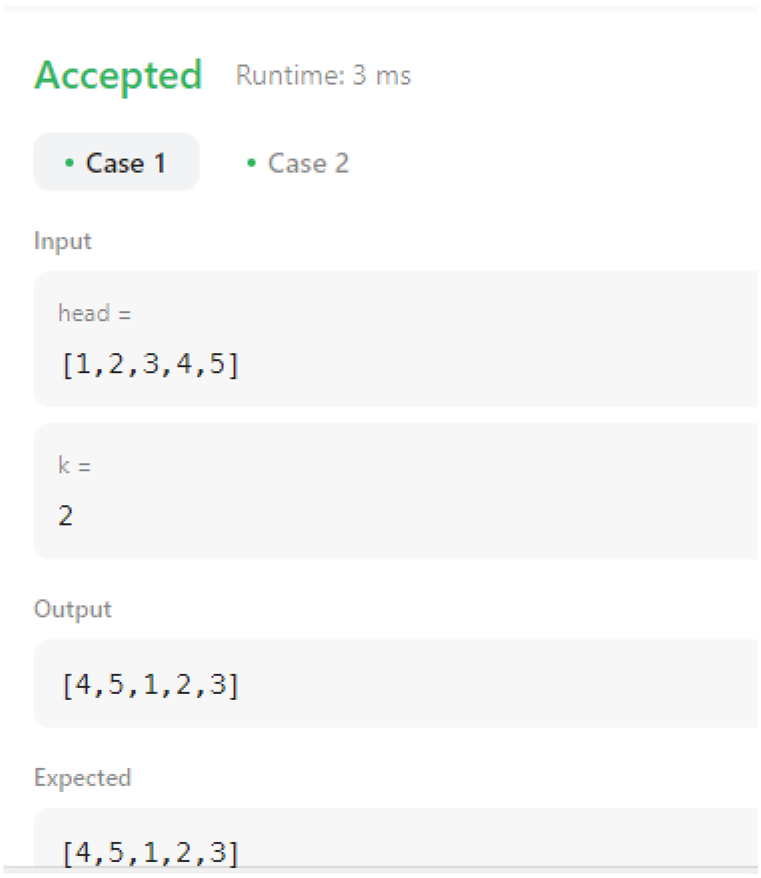
```
[[1,2,3,4],[5,6,7],[8,9,10]]
```

LEET CODE – 4 (ROTATE LIST)

SOURCE CODE:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/

struct ListNode* rotateRight(struct ListNode* head, int k) {
if (head == NULL || k == 0) {
return head;
}
struct ListNode* current = head;
int length = 1;
while (current->next != NULL) {
current = current->next;
length++;
}
k = k % length;
if (k == 0) {
return head;
}
current = head;
for (int i = 1; i < length - k; i++) {
current = current->next;
}
struct ListNode* newHead = current->next;
current->next = NULL;
current = newHead;
while (current->next != NULL) {
current = current->next;
}
current->next = head;
return newHead;
}
```

OUTPUT:


Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

head =

```
[1,2,3,4,5]
```

k =

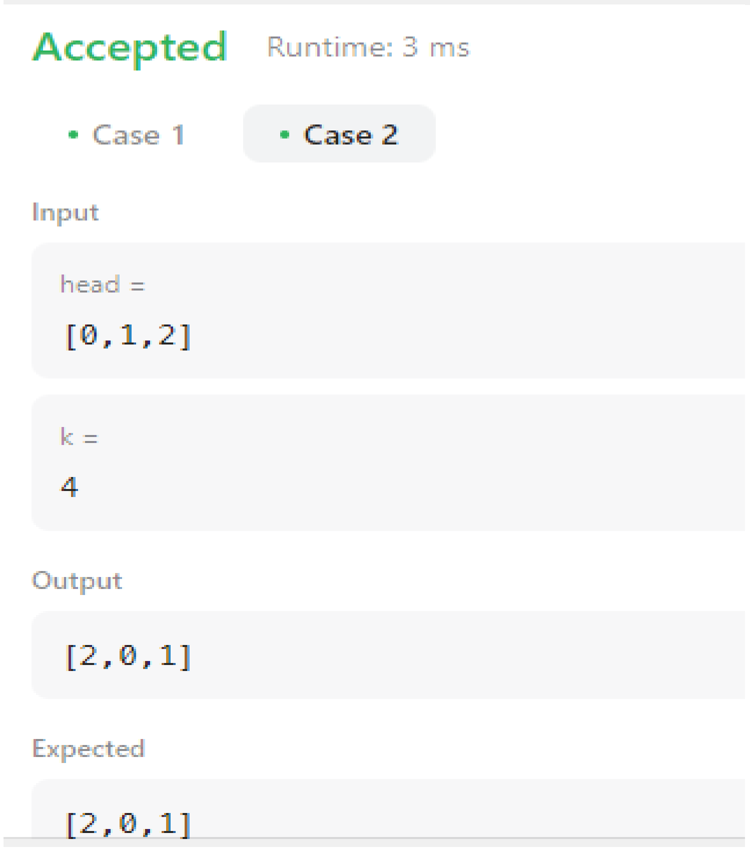
```
2
```

Output

```
[4,5,1,2,3]
```

Expected

```
[4,5,1,2,3]
```


Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

head =

```
[0,1,2]
```

k =

```
4
```

Output

```
[2,0,1]
```

Expected

```
[2,0,1]
```

HACKER RANK

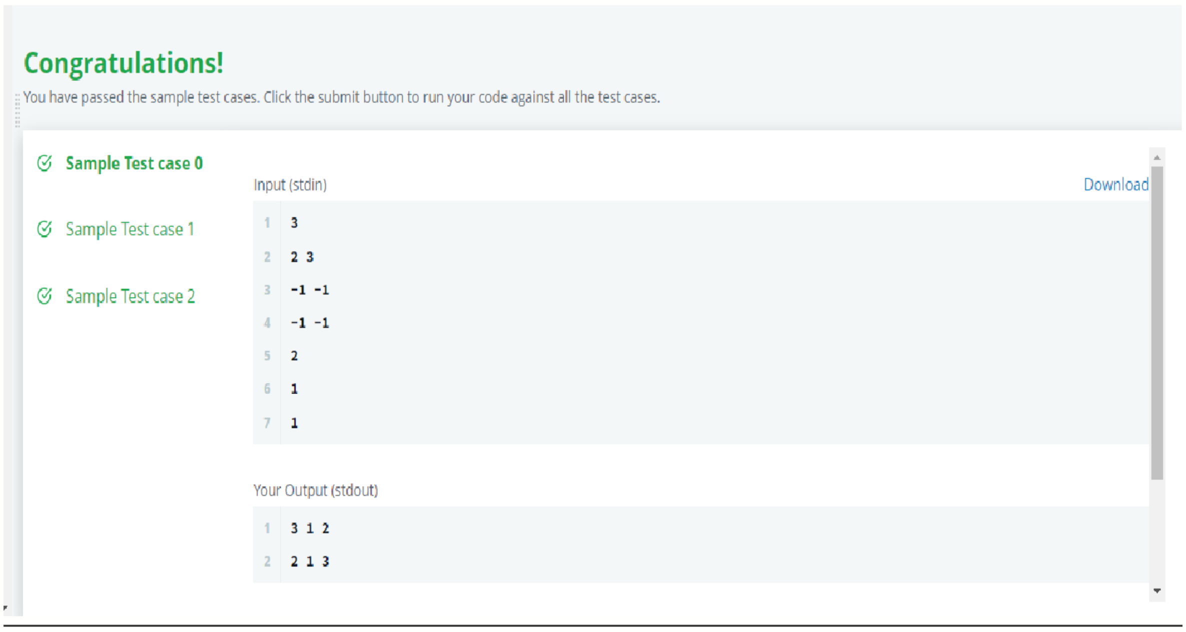
SOURCE CODE:

```
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
   int data;
   struct Node* left;
   struct Node* right;
} Node;
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
void inOrderTraversal(Node* root, int* result, int* index) {
   if (root == NULL) return;
   inOrderTraversal(root->left, result, index);
   result[(*index)++] = root->data;
   inOrderTraversal(root->right, result, index);
}
void swapAtLevel(Node* root, int k, int level) {
   if (root == NULL) return;
   if (level % k == 0) {
      Node* temp = root->left;
      root->left = root->right;
      root->right = temp;
   }
   swapAtLevel(root->left, k, level + 1);
   swapAtLevel(root->right, k, level + 1);
}
int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count, int* queri
es, int* result_rows, int* result_columns) {
   Node** nodes = (Node**)malloc((indexes_rows + 1) * sizeof(Node*));
   for (int i = 1; i <= indexes_rows; i++) {
      nodes[i] = createNode(i);
   }
   for (int i = 0; i < indexes_rows; i++) {
      int leftIndex = indexes[i][0];
      int rightIndex = indexes[i][1];
```

```
  if (leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];
      if (rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];
    }
    int** result = (int**)malloc(queries_count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;
    for (int i = 0; i < queries_count; i++) {
      swapAtLevel(nodes[1], queries[i], 1);
      int* traversalResult = (int*)malloc(indexes_rows * sizeof(int));
      int index = 0;
      inOrderTraversal(nodes[1], traversalResult, &index);
      result[i] = traversalResult;
    }
    free(nodes);
    return result;
}

int main() {
    int n;
    scanf("%d", &n);
    int** indexes = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
      indexes[i] = malloc(2 * sizeof(int));
      scanf("%d %d", &indexes[i][0], &indexes[i][1]);
    }
    int queries_count;
    scanf("%d", &queries_count);

    int* queries = malloc(queries_count * sizeof(int));
    for (int i = 0; i < queries_count; i++) {
      scanf("%d", &queries[i]);
    }
    int result_rows;
    int result_columns;
    int** result = swapNodes(n, 2, indexes, queries_count, queries, &result_rows, &result_columns);

    for (int i = 0; i < result_rows; i++) {
      for (int j = 0; j < result_columns; j++) {
        printf("%d ", result[i][j]);
      }
      printf("\n");
      free(result[i]);
    }
    free(result);
    for (int i = 0; i < n; i++) {
```

```
 free(indexes[i]);
    }
    free(indexes);
    free(queries);
    return 0;
}
```

Output:



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
3
2 3
-1 -1
-1 -1
2
1
1
```

Your Output (stdout)

```
3 1 2
2 1 3
```