# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**Nihal Reddy S(1BM22CS179)**

**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 **Nihal Reddy S(1BM22CS179)**,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**
Assistant Professor
Department of CSE
BMSCE, Bengaluru

**Dr. Jyothi S Nayak**
Professor and Head
Department of CSE
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. |

# Week-1

## 1. Swapping using pointers

```
#include <stdio.h>
void swap(int *a, int *b)

{

      int temp;

      temp=*a;

      *a=*b;

      *b=temp;

}

void main()

{

      int a,b;

      printf("Enter two numbers to swap ");

      scanf("%d %d", &a, &b);

      printf("The numbers before swapping are %d and %d", a, b);

      swap(&a, &b);

      printf("\nThe numbers after swapping are %d and %d", a, b);

}
```

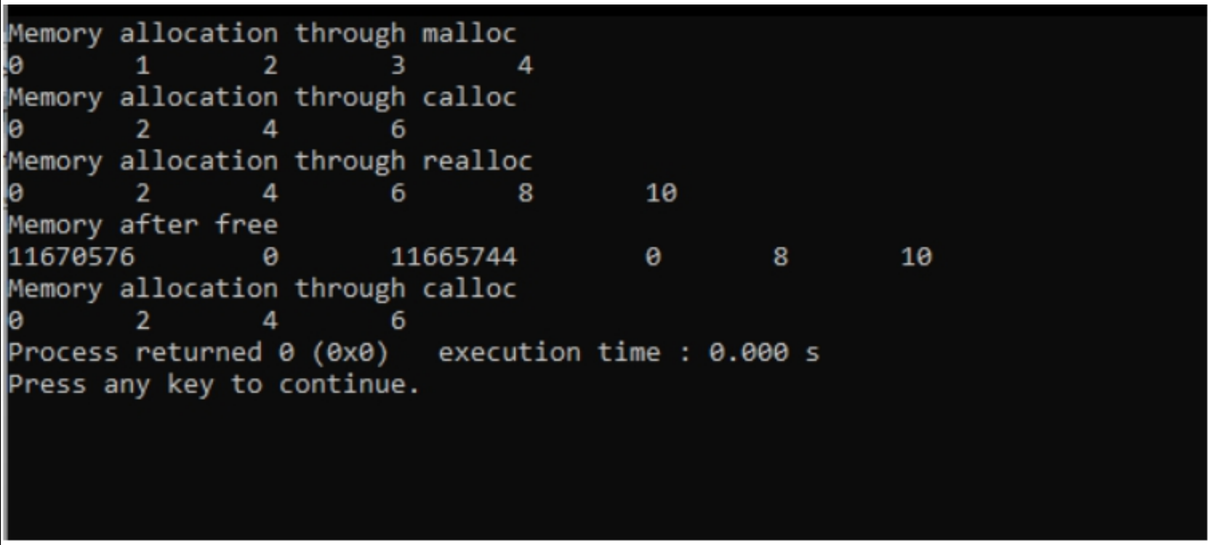
```
Enter two numbers to swap 23 76
The numbers before swapping are 23 and 76
The numbers after swapping are 76 and 23
Process returned 41 (0x29)   execution time : 6.088 s
Press any key to continue.
```

## 2. Dynamic memory allocation

```
#include <stdio.h>

#include <stdlib.h>


int main() {

    int *ptr_malloc, *ptr_calloc, *ptr_realloc;

    int n = 5;

    ptr_malloc = (int *)malloc(n * sizeof(int));

    if (ptr_malloc == NULL) {

        printf("Memory allocation using malloc failed\n");

        return 1;

    }
```

```
printf("Memory allocation through malloc: ");
 for (int i = 0; i < n; i++) {
     ptr_malloc[i] = i + 1;
     printf("%d ", ptr_malloc[i]);
 }

 ptr_calloc = (int *)calloc(n, sizeof(int));
 if (ptr_calloc == NULL) {
     printf("\nMemory allocation using calloc failed\n");
     free(ptr_malloc);
     return 1;
 }

 printf("\nMemory allocation through calloc: ");
 for (int i = 0; i < n; i++) {
     ptr_calloc[i] = (i + 1) * 2;
     printf("%d ", ptr_calloc[i]);
 }

   ptr_realloc = (int *)realloc(ptr_calloc, 2 * n * sizeof(int));
 if (ptr_realloc == NULL) {
     printf("\nMemory reallocation using realloc failed\n");
     free(ptr_malloc);
     free(ptr_calloc);
     return 1;
 }

   printf("\nMemory reallocation through realloc: ");
 for (int i = n; i < 2 * n; i++) {
     ptr_realloc[i] = (i + 1) * 2;
     printf("%d ", ptr_realloc[i]);
 }
```

```
   free(ptr_malloc);

   free(ptr_realloc);

   free(ptr_calloc);


    return 0;

}
```



```
Memory allocation through malloc
0       1       2       3       4
Memory allocation through calloc
0       2       4       6
Memory allocation through realloc
0       2       4       6       8       10
Memory after free
11670576        0       11665744        0       8       10
Memory allocation through calloc
0       2       4       6
Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

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

```
#include <stdio.h>

#include <stdlib.h>

#define max 100

int top = -1;

int stack[max];

void push(int a);

int pop();

void display();

int main() {

int arr[100], size;
printf("Enter array size: “);

 scanf("%d", &size);

printf("Enter values of stack:\n");

for (int i = 0; i < size; i++) {

scanf("%d", &arr[i]);
```

```
push(arr[i]); }

printf("Stack before popping:\n");

display();


for (int i = size - 1; i >= 0; i--) {

pop();

}
printf("Stack after popping:\n"); display();
return 0;

}
void push(int a) {

if (top == max - 1) { printf("Stack overflow\n"); return;

}
top = top + 1; stack[top] = a;

}
int pop() {

if (top == -1) {
printf("Stack underflow\n"); return -1;

}
top--;
return stack[top];

}
void display() {

if (top == -1) {
printf("Stack is empty\n"); return;

}
printf("Stack elements:\n"); for (int i = 0; i <= top; i++) {

printf("%d\t", stack[i]); }

printf("\n"); }
```

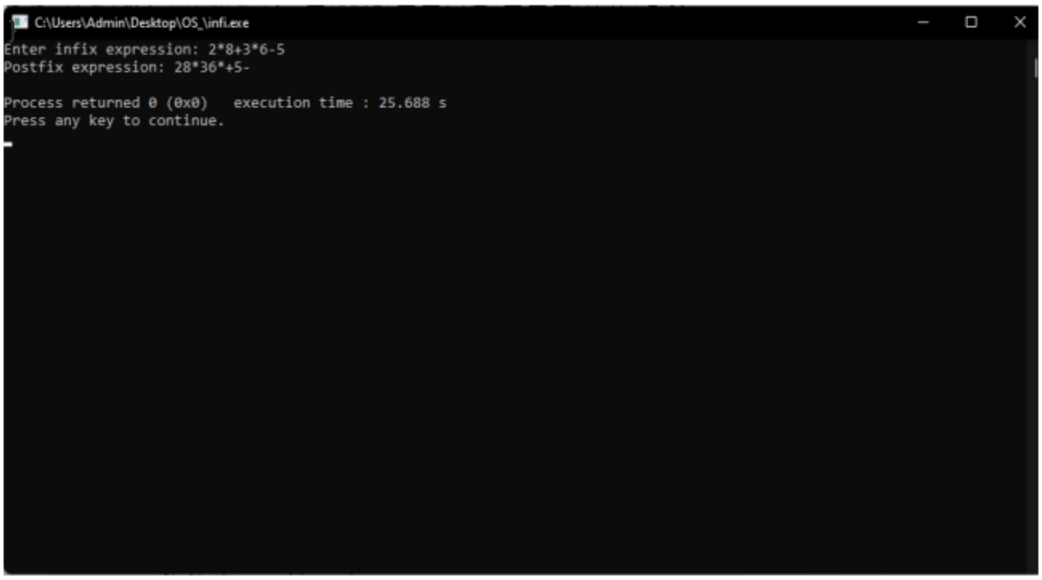
```
Enter array size: 7
Enter values of stack:
1
2
3
4
5
6
7
Stack before popping:
Stack elements:
1       2       3       4       5       6       7
Stack after popping:
Stack is empty

Process returned 0 (0x0)   execution time : 6.063 s
Press any key to continue.
```

## (WEEK 2)

1. **Write a program 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), / (divide) and ^ (power).**

```
#include <stdio.h>

#include

<stdlib.h>

#define MAX_SIZE

100 int

isOperator(char

ch) {
  return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch ==
  '%');
}

int precedence(char operator) {
  if (operator == '+' || operator == '-')
    return 1;
  if (operator == '*' || operator == '/' || operator == '%')
    return 2;
  return 0;
}

void infixToPostfix(char infix[], char postfix[]) {
  char
  stack[MAX_SIZE];
  int top = -1;
  int i, j;

  for (i = 0, j = 0; infix[i] != '\0'; i++) {
    if (infix[i] >= '0' && infix[i] <= '9') {
      postfix[j++] = infix[i];
    } else if (isOperator(infix[i])) {
      while (top >= 0 && precedence(stack[top]) >=
precedence(infix[i])) {
        postfix[j++] = stack[top--];
      }
```

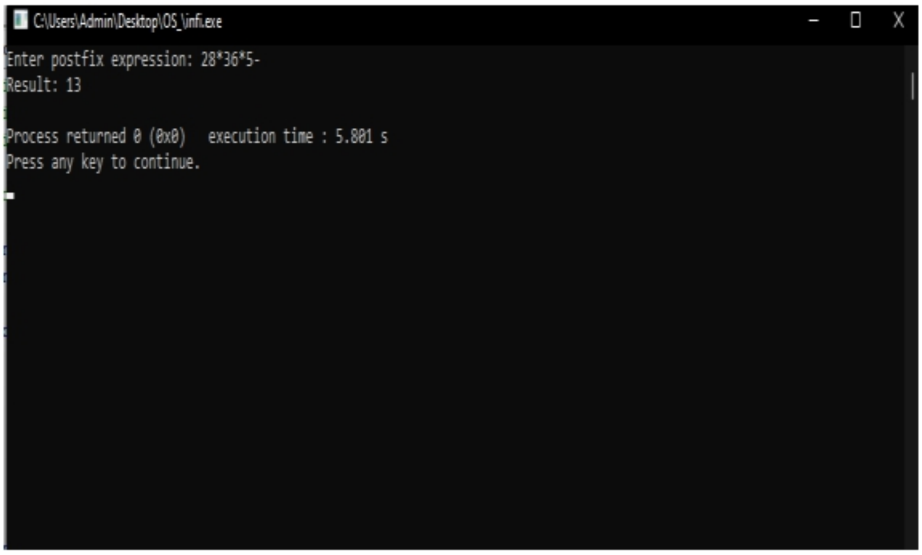
```
            postfix[j++] = stack[top--];
          }
          if (top >= 0 && stack[top] == '(') {
            top--;
          }
        }
      }

      while (top >= 0) {
        postfix[j++] = stack[top--];
      }

      postfix[j] = '\0';
    }
    int main() {
      char infix[MAX_SIZE], postfix[MAX_SIZE];

      printf("Enter infix
      expression: "); scanf("%s",
      infix);

      infixToPostfix(infix, postfix);

      printf("Postfix expression: %s\n",

      postfix); return 0;
```



## 2.Program 2:Postfix Evaluation

```
 #include
<stdio.h>
#include
<stdlib.h>
#include
<ctype.h>
#define MAX_STACK_SIZE 100
int stack[MAX_STACK_SIZE];
int top = -1;
void push(int item) {
  if (top == MAX_STACK_SIZE - 1) {
    printf("Stack Overflow\n");
    exit(EXIT_FAILURE);
  }
  stack[++top] = item;
}
int pop() {
  if (top == -1) {
    printf("Stack
    Underflow\n");
    exit(EXIT_FAILURE);
  }
  return stack[top--];
}
int isOperator(char ch) {
  return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch ==
  '%');
}
int evaluatePostfix(char postfix[]) {
  int i = 0;
  while (postfix[i] != '\0') {
    char currentSymbol = postfix[i];
    if (isdigit(currentSymbol)) {
      push(currentSymbol - '0');
    } else if (isOperator(currentSymbol)) {
      int operand2 = pop();
      int operand1 = pop();
      switch (currentSymbol) {
        case '+':
          push(operand1 +
```

```
            break;
          case '*':
            push(operand1 *
            operand2); break;
          case '/':
            push(operand1 / operand2);
            break;
          case '%':
            push(operand1 % operand2);
           break;
        }
      }
      i++;
    }
    return pop();
  }

  int main() {
    char postfixExpression[100];
    printf("Enter postfix
    expression: "); scanf("%s",
    postfixExpression);
    int result =
    evaluatePostfix(postfixExpression);
    printf("Result: %d\n", result);
```



**(WEEK 3)**

**1.WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display. The program should print appropriate messages for queue empty and queue overflow conditions.**

```
#include <stdio.h>
#define SIZE 5
int queue[SIZE];
int front=-1;
int rear=-1;
void insert(int element)
{
   if((rear+1)%SIZE==front)
      printf("Queue overflow");
   else
   {
      rear=(rear+1)%SIZE;
      if(front==-1)
         front=front+1;
      queue[rear]=element;
   }
}
void delete()
{
   if(front==-1 && rear==-1)
      printf("Queue underflow");
   else
   {
      printf("The element popped is %d",queue[front]);
```

```
        if(front==rear)
            front=rear=-1;
        else
            front=(front+1)%SIZE;
    }
}
void display()
{
    int i;
    if(front==-1 && rear==-1)
        printf("Queue underflow");
    else
    {
        i=front;
        while(1)
        {
            printf("%d ",queue[i]);
            if(i==rear)
                break;
            i=(i+1)%SIZE;
        }
    }
}
void main()
{   while(1)
    {
        int ch,element;
```

```
        printf("Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit ");
        scanf("%d",&ch);
        if(ch==1)
        {
           printf("Enter the element to insert into the queue ");
           scanf("%d",&element);
           insert(element);
        }
        else if(ch==2)
           delete();
        else if(ch==3)
           display();
        else if(ch==4)
           break;
        else
           printf("Invalid input");
        printf("\n\n");
   }
}
```

```
Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 0


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 1


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 2


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 3


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 4


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
0 1 2 3 4

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 2
The element popped is 0

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 2
The element popped is 1

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
2 3 4

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 100


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 200


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
2 3 4 100 200

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 4
```

**(WEEK 4)**

**1. WAP to Implement Singly Linked List with following operations:**

**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.**

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head=NULL;
void push()
{
   struct Node *new_node=malloc(sizeof(struct Node));
   int data;
   printf("Enter the data to be entered ");
   scanf("%d",&data);
   (*new_node).data=data;
   (*new_node).next=head;
   head=new_node;


}
void append()
{
```

```
    struct Node *new_node=malloc(sizeof(struct Node));
    int data;
    struct Node *last=head;
    printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
    (*new_node).next=NULL;
    if(head==NULL)
        head=new_node;
    else
    {
        while((*last).next!=NULL)
        {
            last=(*last).next;
        }

        (*last).next=new_node;
    }
}
void insert_at_pos(int pos)
{
    struct Node *new_node=malloc(sizeof(struct Node));
    struct Node *temp=head;
    int data;
    printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
```

```
    if(pos==1)
    {
        (*new_node).next=head;
        head=new_node;
        return;
    }
    int position=1;
    while(1)
    {
        if(position==pos-1)
            break;
        else
        {
            temp=(*temp).next;
            position=position+1;
        }
    }
    (*new_node).next=(*temp).next;
    (*temp).next=new_node;
}
void display()
{
    struct Node *node=head;
    while(1)
    {
        printf("%d ",(*node).data);
        if((*node).next==NULL)
```

```
            break;
        node=(*node).next;
    }
}
void main()
{
    int choice;
    while(1)
    {
        printf("Enter 1 to insert at the beginning, 2 to append at the
end, 3 to insert in the middle, 4 to display the contents and 5 to
exit. ");
        scanf("%d",&choice);
        if(choice==1)
            push();
        else if(choice==2)
            append();
        else if(choice==3)
        {
            int position;
            printf("Enter the position to insert the node. ");
            scanf("%d",&position);
            insert_at_pos(position);
        }
        else if(choice==4)
            display();
        else if(choice==5)
            break;
```

```
else
    printf("Invalid input entered.");
printf("\n\n");
}
}
```

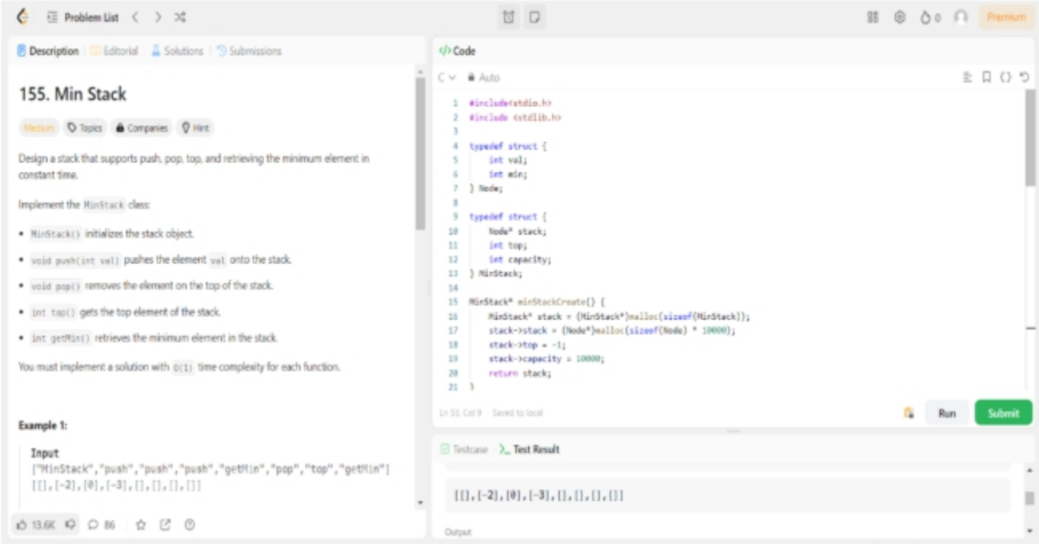
```
Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 1
Enter the data to be entered 23


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 1
Enter the data to be entered 45


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 2
Enter the data to be entered 77


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 4
45 23 77

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 3
Enter the position to insert the node. 3
Enter the data to be entered 100


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 4
45 23 100 77

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 5

Process returned 5 (0x5)   execution time : 114.974 s
Press any key to continue.
```

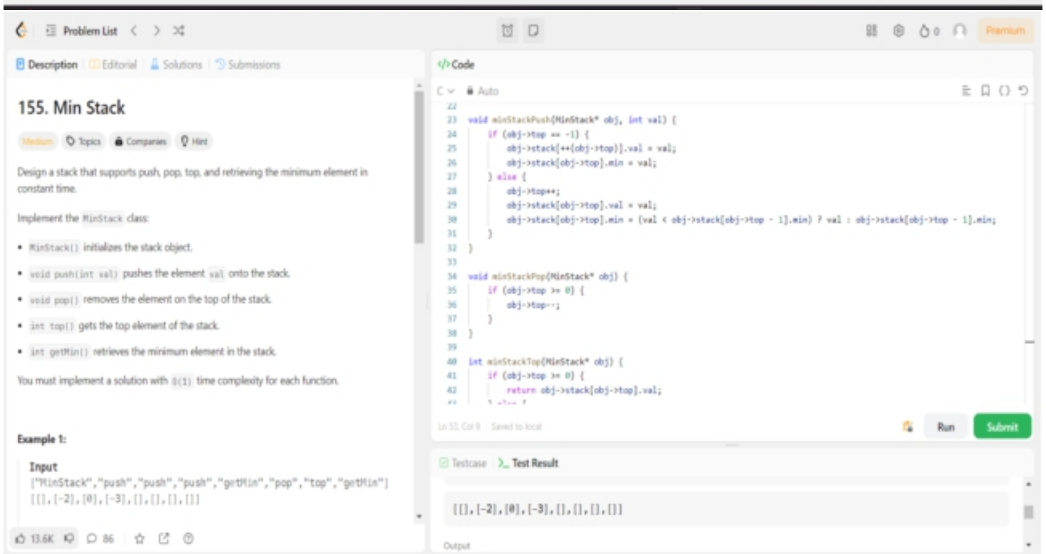
2. **(LEET CODE-1)**
   **Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.**

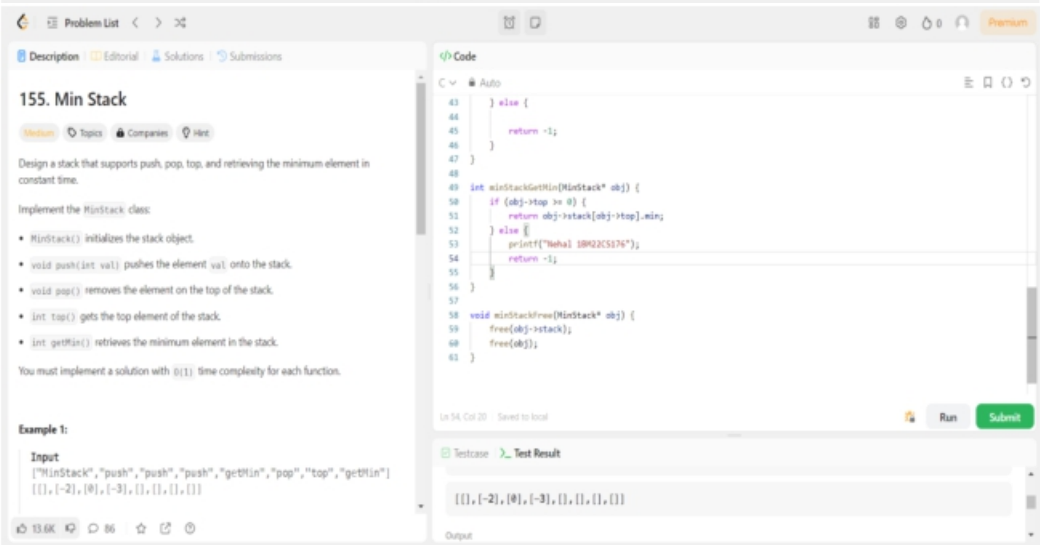
```
#include<stdio.h>
#include <stdlib.h>

typedef struct {
    int val;
    int min;
} Node;

typedef struct
    { Node*
    stack; int
    top;
    int capacity;
} MinStack;

MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->stack = (Node*)malloc(sizeof(Node) * 10000);
    stack->top = -1;
```

```
    stack->capacity = 10000;
    return stack;
}

void minStackPush(MinStack* obj, int val) {
    if (obj->top == -1) { obj->stack[++
        (obj->top)].val = val; obj-
        >stack[obj->top].min = val;
    } else {
        obj->top++; obj->stack[obj-
        >top].val = val;
        obj->stack[obj->top].min = (val < obj->stack[obj->top - 1].min) ?
val : obj->stack[obj->top - 1].min;
    }
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj) {
    if (obj->top >= 0) {
        return obj->stack[obj->top].val;
    } else {

        return -1;
    }
}

int minStackGetMin(MinStack* obj) {
    if (obj->top >= 0) {
        return obj->stack[obj->top].min;
    } else {

        return -1;
    }
}
```

```
void minStackFree(MinStack* obj) {
    free(obj->stack);
    free(obj);
}
```



**(WEEK 5)**

**WAP to Implement Singly Linked List with following operations**

**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.**

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
   int data;
   struct Node* next;
};

void addAtIndex(struct Node** head, int index, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;

   if (index == 0) { newNode-
      >next = *head; *head =
      newNode;
   } else {
      struct Node* temp = *head;
      for (int i = 0; i < index - 1 && temp != NULL; i++) {
         temp = temp->next;
      }

      if (temp == NULL)
         { printf("Invalid index!
         \n"); free(newNode);
         return;
      }

      newNode->next = temp->next;
      temp->next = newNode;
   }

   printf("Element added at index %d\n", index);
}
```

```
void deleteAtStart(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }

    struct Node* temp = *head;
    *head = temp->next;
    free(temp);

    printf("Element deleted at the start\n");
}
void deleteAtIndex(struct Node** head, int index) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }

    struct Node* temp = *head;
    if (index == 0) {
        *head = temp->next;
        free(temp);
        printf("Element deleted at index
    0\n"); } else {
        for (int i = 0; i < index - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp == NULL || temp->next == NULL) {
            printf("Invalid index!\n");
            return;
        }

        struct Node* toDelete = temp->next;
        temp->next = toDelete->next;
        free(toDelete);

        printf("Element deleted at index %d\n", index);
    }
```

```
}
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }

    struct Node* temp = *head;
    struct Node* prev = NULL;

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

    if (prev == NULL) {
        *head = NULL;
    } else {
        prev->next = NULL;
    }

    free(temp);

    printf("Element deleted at the end\n");
}
void displayList(struct Node* head) {
    printf("Linked List: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, index, data;

    while (1) {
        printf("\nNihal 1BM22CS179");
```

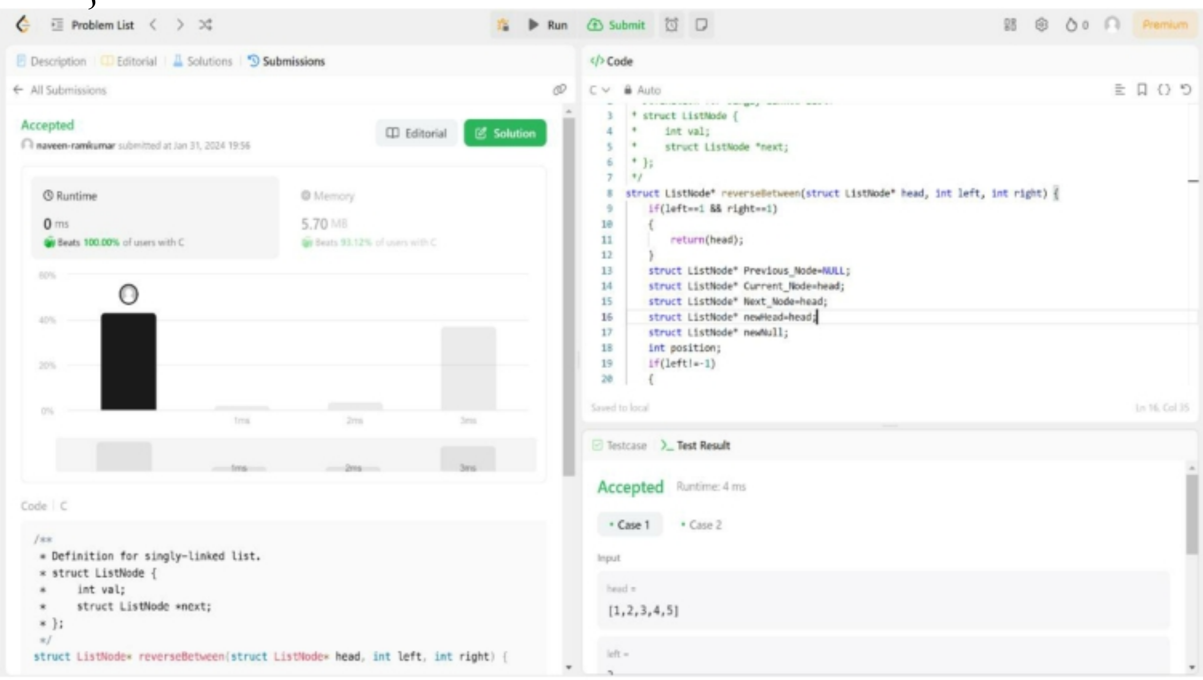
```
        printf("\n1. Add element at a given index\n");
        printf("2. Delete at start\n");
        printf("3. Delete at index\n");
        printf("4. Delete at end\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter index and data to add: ");
                scanf("%d %d", &index, &data);
                addAtIndex(&head, index, data);
                break;
            case 2:
                deleteAtStart(&head);
                break;
            case 3:
                printf("Enter index to delete: ");
                scanf("%d", &index);
                deleteAtIndex(&head, index);
                break;
            case 4:
                deleteAtEnd(&head);
                break;
            case 5:
                displayList(head);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

```
1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 1
Enter index and data to add: 0 1
Element added at index 0

1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 1
Enter index and data to add: 1 2
Element added at index 1

1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 1
Enter index and data to add: 2 3
Element added at index 2

1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 3
Enter index to delete: 2
Element deleted at index 2

1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 5
Linked List: 1 2

1. Add element at a given index
2. Delete at start
3. Delete at index
4. Delete at end
5. Display
6. Exit
Enter your choice: 1
Enter index and data to add: 2 3
Element added at index 2
```

**2.(LEET CODE-2) Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.**

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
   if(left==1 && right==1)
   {
      return(head);
   }
   struct ListNode* Previous_Node=NULL;
   struct ListNode* Current_Node=head;
   struct ListNode* Next_Node=head;
   struct ListNode* newHead=head;
   struct ListNode* newNull;
   int position;
   if(left!=-1)
   {
      for(position=1; position<left; position++)
```

```
            {
                if(position==left-1)
                {
                    newHead=Current_Node;
                }
                Current_Node=(*Current_Node).next;
            }
        }
        Previous_Node=Current_Node;
        newNull=Current_Node;
        Current_Node=(*Current_Node).next;
        for(position=left+1; position<=right; position++)
        {
            Next_Node=(*Current_Node).next;
            (*Current_Node).next=Previous_Node;
            Previous_Node=Current_Node;
            if(position==right)
            {
                if(left==1)
                {
                    head=Current_Node;
                }
                else
                    (*newHead).next=Current_Node;
            }
            Current_Node=Next_Node;
        }
        (*newNull).next=Current_Node;
        return(head);
    }
```



(WEEK 6)

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

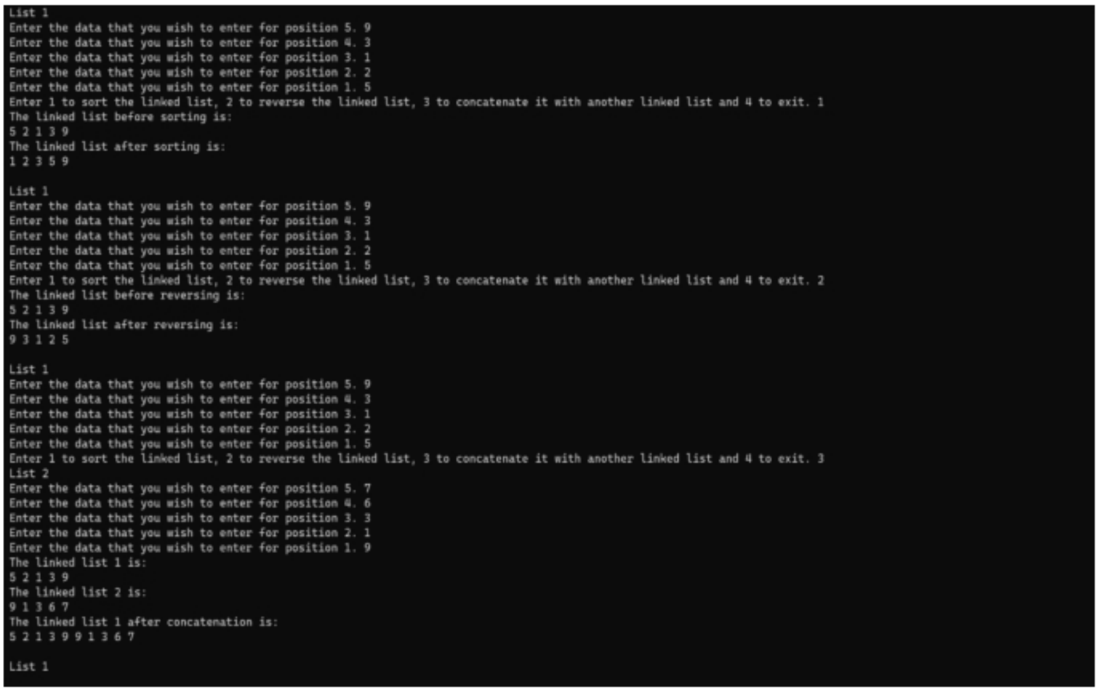
```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* next;
};
struct Node* head=NULL;
struct Node* head2=NULL;
void sort(struct Node* head)
{
    struct Node* i;
    struct Node* j;
    int temp;
    printf("The linked list before sorting is:\n");
    display(head);
    for(i=head; (*i).next!=NULL; i=(*i).next)
    {
        for(j=(*i).next; (*j).next!=NULL; j=(*j).next)
        {
            if((*j).data<(*i).data)
            {
                temp=(*i).data;
                (*i).data=(*j).data;
```

```
            (*j).data=temp;
         }
      }
   }
   printf("\nThe linked list after sorting is:\n");
   display(head);
}
void reverse(struct Node* head)
{
   struct Node* previous_Node=NULL;
   struct Node* current_Node=head;
   struct Node* next_Node;
   printf("The linked list before reversing is:\n");
   display(head);
   while(current_Node!=NULL)
   {
      next_Node=(*current_Node).next;
      if(next_Node==NULL)
      {
         head=current_Node;
      }
      (*current_Node).next=previous_Node;
      previous_Node=current_Node;
      current_Node=next_Node;
   }
   printf("\nThe linked list after reversing is:\n");
   display(head);
```

```
}
void concatenate(struct Node* head1, struct Node* head2)
{
   printf("The linked list 1 is:\n");
   display(head);
   printf("\nThe linked list 2 is:\n");
   display(head2);
   struct Node* last;
   for(last=head; (*last).next!=NULL; last=(*last).next);
   (*last).next=head2;
   printf("\nThe linked list 1 after concatenation is:\n");
   display(head);
}
void display(struct Node* head)
{
   struct Node* temp;
   for(temp=head; temp!=NULL; temp=(*temp).next)
   {
      printf("%d ", (*temp).data);
   }
}
void main()
{
   struct Node* New_Node;
   int position;
   int data;
   int choice;
```

```
while(1)
{
    head=NULL;
    head2=NULL;
    printf("List 1\n");
    for(position=1; position<=5; position++)
    {
        printf("Enter the data that you wish to enter for position %d. ", 6-position);
        scanf("%d",&data);
        struct Node* New_Node=malloc(sizeof(struct Node));
        (*New_Node).data=data;
        (*New_Node).next=head;
        head=New_Node;
    }
    printf("Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. ");
    scanf("%d", &choice);
    if(choice==1)
        sort(head);
    else if(choice==2)
        reverse(head);
    else if(choice==3)
    {
        printf("List 2\n");
        for(position=1; position<=5; position++)
        {
```

```
            printf("Enter the data that you wish to enter for position %d. ", 6-position);
            scanf("%d",&data);
            struct Node* New_Node=malloc(sizeof(struct Node));
            (*New_Node).data=data;
            (*New_Node).next=head2;
            head2=New_Node;
         }
         concatenate(head, head2);
      }
      else if(choice==4)
         break;
      else
         printf("Invalid input character.");
      printf("\n\n");
   }
}
```



```
List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 1
The linked list before sorting is:
5 2 1 3 9
The linked list after sorting is:
1 2 3 5 9

List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 2
The linked list before reversing is:
5 2 1 3 9
The linked list after reversing is:
9 3 1 2 5

List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 3
List 2
Enter the data that you wish to enter for position 5. 7
Enter the data that you wish to enter for position 4. 6
Enter the data that you wish to enter for position 3. 3
Enter the data that you wish to enter for position 2. 1
Enter the data that you wish to enter for position 1. 9
The linked list 1 is:
5 2 1 3 9
The linked list 2 is:
9 1 3 6 7
The linked list 1 after concatenation is:
5 2 1 3 9 9 1 3 6 7

List 1
```

**2.WAP to Implement Single Linked List to simulate Stack Operations.**

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
int top=0;
struct Node
{
    int data;
    struct Node *next;
};
struct Node *head=NULL;
void push()
{
    if(top==SIZE)
    {
        printf("Stack overflow. Cannot insert more elements into the
stack.");
    }
    else
    {
        struct Node *new_node=malloc(sizeof(struct Node));
        int data;
        struct Node *last=head;
        printf("Enter the data to be entered ");
        scanf("%d",&data);
        (*new_node).data=data;
```

```
        (*new_node).next=NULL;
        if(head==NULL)
        {
            head=new_node;
        }
        else
        {
            while((*last).next!=NULL)
            {
                last=(*last).next;
            }
            (*last).next=new_node;
        }
        top=top+1;
    }
}
void pop()
{
    if(top==0)
        printf("Stack underflow. You cannot delete from an empty
list.");
    else
    {
        int deleted_node;
        if((*head).next==NULL)
        {
            deleted_node=(*head).data;
            free(head);
```

```
            head=NULL;
        }
        else
        {
            struct Node *ptr1=head;
            struct Node *ptr=(*ptr1).next;
            while((*ptr).next!=NULL)
            {
                ptr1=(*ptr1).next;
                ptr=(*ptr1).next;
            }
            (*ptr1).next=NULL;
            deleted_node=(*ptr).data;
            free(ptr);
        }
        top=top-1;
        printf("The deleted element is %d", deleted_node);
    }
}
void display()
{
    if(top==0)
    {
        printf("Stack undeflow. Cannot display the contents of an
empty stack.");
    }
    else
    {
```

```
        struct Node *node=head;
        while(node!=NULL)
        {
          printf("%d ",(*node).data);
          node=(*node).next;
        }
    }
}
void main()
{
    while(1)
    {
        printf("Enter 1 to push into the stack, 2 to pop from the stack,
3 to display the contents and 4 to exit. ");
        int ch;
        scanf("%d",&ch);
        if(ch==1)
        {
          push();
        }
        else if(ch==2)
        {
          pop();
        }
        else if(ch==3)
        {
          display();
        }
```

```
        else if(ch==4)
        {
            break;
        }
        else
        {
            printf("Invalid character.");
        }
        printf("\n\n");
    }
}
```

2. **WAP to Implement Single Linked List to simulate Queue Operations.**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
struct Node
{
    int data;
    struct Node *next;
};
struct Node *head=NULL;
int rear=-1;
void append()
{
    if(rear==MAX-1)
    {
        printf("Queue overflow");
```

```
    }   else
    {
        rear=rear+1;
        struct Node *new_node=malloc(sizeof(struct Node));
        int data;
        struct Node *last=head;
        printf("Enter the data to be entered ");
        scanf("%d",&data);
        (*new_node).data=data;
        (*new_node).next=NULL;
        if(head==NULL)
            head=new_node;
        else
        {
            while((*last).next!=NULL)
            {
                last=(*last).next;
            }
            (*last).next=new_node;
        }
    }
}
void Pop()
{
    if(head==NULL)
        printf("The queue is empty. You cannot delete from an empty
queue");
    else
```

```
    {
        struct Node *ptr=head;
        head=(*ptr).next;
        free(ptr);
    }
}
void display()
{
    if(head==NULL)
        printf("The queue is empty. You cannot display the elements
from an empty queue");
    else
    {
        struct Node *node=head;
        while(node!=NULL)
        {   printf("%d ",(*node).data);
            node=(*node).next;
        }   } }
void main()
{   while(1)   {
        printf("Enter 1 to append elements to the queue, 2 to delete
elements from the queue, 3 to display the elements of the queue
and 4 to exit. ");
        int ch;
        scanf("%d",&ch);
        if(ch==1)
        {
            append();
```

```
    }
    else if(ch==2)
    {
        Pop();
    }
    else if(ch==3)
    {
        display();
    }
    else if(ch==4)
    {
        break;
    }
    else {
        printf("Invalid character");
    }
    printf("\n\n");
}}
```

```
Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 1
Enter the data to be entered 25


Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 1
Enter the data to be entered 67


Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 3
25 67

Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 2


Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 3
67

Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 2


Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 3
The queue is empty. You cannot display the elements from an empty queue

Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. 4

Process returned 4 (0x4)   execution time : 47.482 s
Press any key to continue.
```

(WEEK 7)

1. **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. Display the contents of the list**

```
#include <stdio.h>

#include <stdlib.h>

struct Node

{

   int data;

   struct Node *next;

   struct Node *previous;

};

struct Node *head=NULL;
void insert(int position)

{

   int pos;

   struct Node *node=head;

   for(pos=1; pos<=position; pos++)

   {

      if(node==NULL && !(head==NULL && position==1))

      {

         printf("The given position is longer than the linked list.
Please enter another position.");

         return;

      }
```

```
    if(pos==position)
    {
        break;
    }
    node=(*node).next;
}
int data;
printf("Enter the data to be entered in the new node ");
scanf("%d", &data);
struct Node *newNode;
newNode=malloc(sizeof(struct Node));
(*newNode).data=data;
(*newNode).next=node;
if(head==NULL)
{
    (*newNode).previous=NULL;
    head=newNode;
}
else{ (*newNode).previous=(*node).previo
    us; struct Node *previous;
    previous=(*node).previous;
    (*node).previous=newNode;
    if(previous==NULL)
    {
        head=newNode;
    }
```

```
        else
        {
            (*previous).next=newNode;
        }
    }

}
void delete_based_on_a_value(int value)
{
    struct Node *node=head;
    int first_time=1;
    while(1)
    {
        if(node==NULL)
        {
            printf("Cannot delete from an empty list.");
            return;
        }
        for(node=head; node!=NULL; node=(*node).next)
        {
            if((*node).data==value)
            {
                break;
            }
        }
        if(node==NULL)
        {
```

```
            if(first_time==1)
            {
               printf("The node with the given value is not found in the
linked list.");
            }
            return;
        }
        else
        {
            if((*node).previous==NULL)
            {
               head=(*node).next;
            }
            else
            {
               (*(*node).previous).next=(*node).next;
            } if((*node).next!
            =NULL) {
               (*(*node).next).previous=(*node).previous;
            }
            free(node);
        }
        first_time=0;
    }

}
void display()
```

```
{
   if(head==NULL)
   {
      printf("The linked list is empty.");
   }
   else
   {
      struct Node *node;
      for(node=head; node!=NULL; node=(*node).next)
      {
         printf("%d ", (*node).data);
      }
   }
}
void main()
{
   while(1)
   {
      int ch;
      printf("Enter 1 to insert, 2 to delete an element based on its
value, 3 to display the elements of the linked list and 4 to exit. ");
      scanf("%d", &ch);
      if(ch==1)
      {
         int data, position;
         printf("Enter the position to the left of which you want to
enter the data. ");
         scanf("%d", &position);
```

```
            insert(position);
        }
        else if(ch==2)
        {
            int value;
            printf("Enter the value for which you want to delete from
the linked list. ");
            scanf("%d", &value);
            delete_based_on_a_value(value);
        }
        else if(ch==3)
            display();
        else if(ch==4)
        {
            break;
        }
        else
        {
            printf("Invalid character");
        }
        printf("\n\n");
    }
}
```

```
Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 3
The given position is longer than the linked list. Please enter another position.

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 45


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 34


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
23 34 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 3
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
23 23 34 23 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 2
Enter the value for which you want to delete from the linked list. 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
34 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 4

Process returned 4 (0x4)   execution time : 92.693 s
```

**2. (LEET CODE-3)Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.**

```
/**
 * 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;
}
```

Split Linked List in Parts

## Submission Detail

43 / 43 test cases passed.

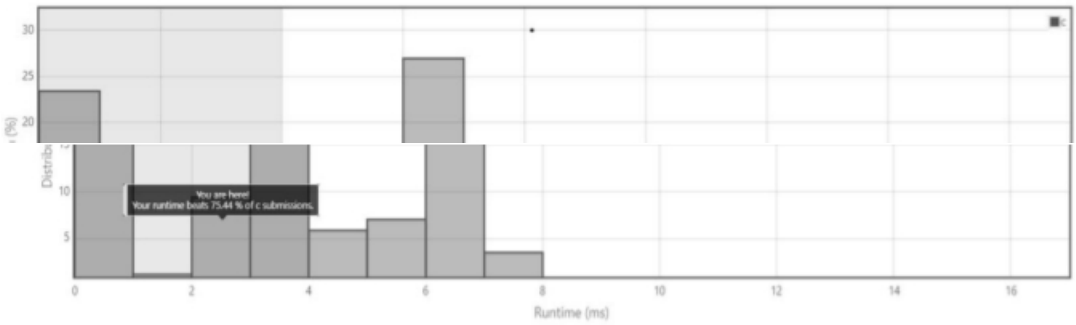
Status: Accepted

Runtime: 2 ms

Memory Usage: 6.2 MB

Submitted: 1 month ago

### Accepted Solutions Runtime Distribution



**(WEEK 8)**

**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.**

```
#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) {
```

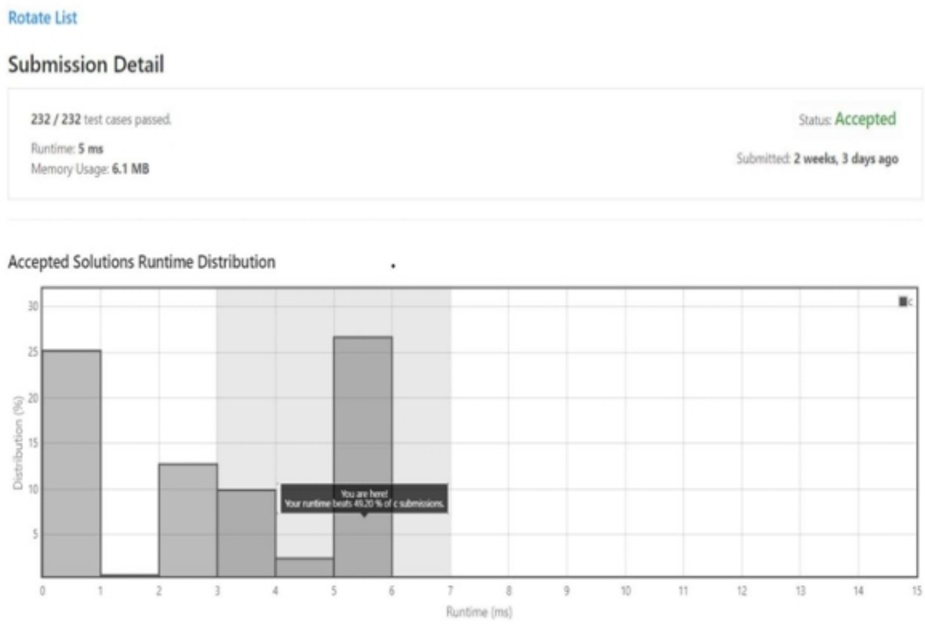
```
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
void postOrderTraversal(struct TreeNode* root) {
    if (root != NULL)
        { postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", 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;
    printf("Nihal1BM22CS179\n");
    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;
```

```
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: 20
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 1
Enter data to insert: 70
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 1
Enter data to insert: 80
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 2
In-order traversal: 20 50 70 80
Pre-order traversal: 50 20 70 80
Post-order traversal: 20 80 70 50
1. Insert a node
2. Display tree
3. Exit
Enter your choice: 3
Exiting program.

Process returned 0 (0x0)   execution time : 20.159 s
Press any key to continue.
```

**2.(LEET CODE-4) Given the head of a linked list, rotate the list to the right by k places**

```
/**
 * 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;
```


Rotate List
Submission Detail
232 / 232 test cases passed.
Status: Accepted
Runtime: 5 ms
Memory Usage: 6.1 MB
Submitted: 2 weeks, 3 days ago
Accepted Solutions Runtime Distribution
You are here!
Your runtime beats 49.20 % of c submissions.
Distribution (%)
Runtime (ms)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
5
10
15
20
25
30

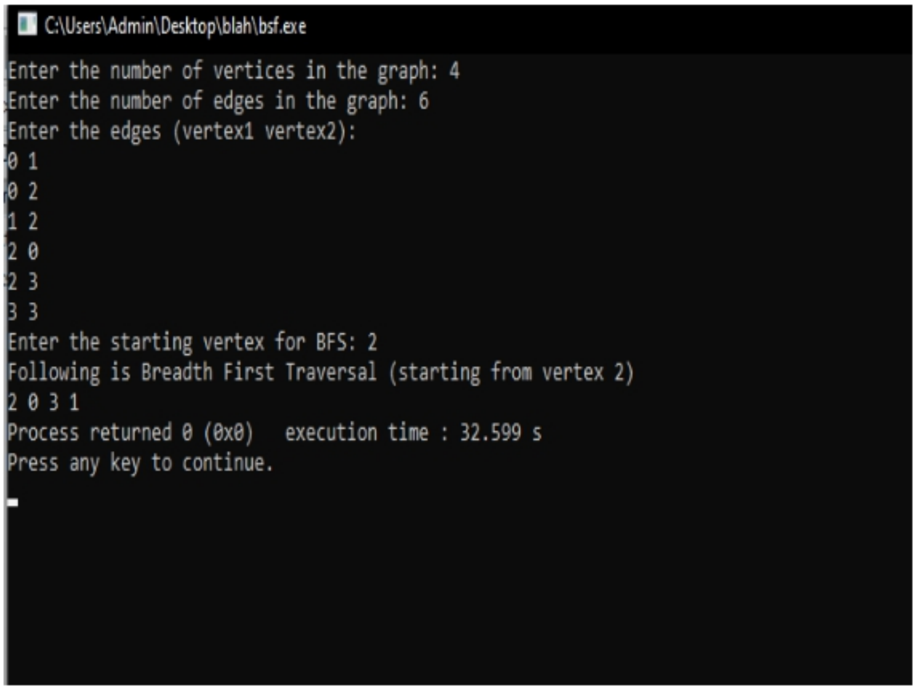
**(WEEK 9)**

**1.Write a program to traverse a graph using BFS method.**

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

#define MAX_VERTICES 50

typedef struct Graph_t {
   int V;
   bool adj[MAX_VERTICES]
[MAX_VERTICES]; } Graph;

Graph* Graph_create(int V) {
   Graph* g = malloc(sizeof(Graph));
   g->V = V;

   for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++)
         { g->adj[i][j] = false;
      }
   }

   return g;
}

void Graph_destroy(Graph* g) {
   free(g);
}

void Graph_addEdge(Graph* g, int v, int w)
   { g->adj[v][w] = true;
}

void Graph_BFS(Graph* g, int s)
   { bool
   visited[MAX_VERTICES]; for
   (int i = 0; i < g->V; i++) {
      visited[i] = false;
   }
```

```
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[s] = true;
    queue[rear++] = s;

    while (front != rear)
        { s = queue[front+
        +]; printf("%d ", s);

        for (int adjacent = 0; adjacent < g->V; adjacent++) {
            if (g->adj[s][adjacent] && !visited[adjacent]) {
                visited[adjacent] = true;
                queue[rear++] = adjacent;
            }
        }
    }
}

int main() {
    int numVertices;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &numVertices);

    Graph* g = Graph_create(numVertices);

    int numEdges;
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &numEdges);

    printf("Enter the edges (vertex1 vertex2):\n");
    for (int i = 0; i < numEdges; i++) {
        int v, w;
        scanf("%d %d", &v, &w);
        Graph_addEdge(g, v, w);
    }

    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);
```

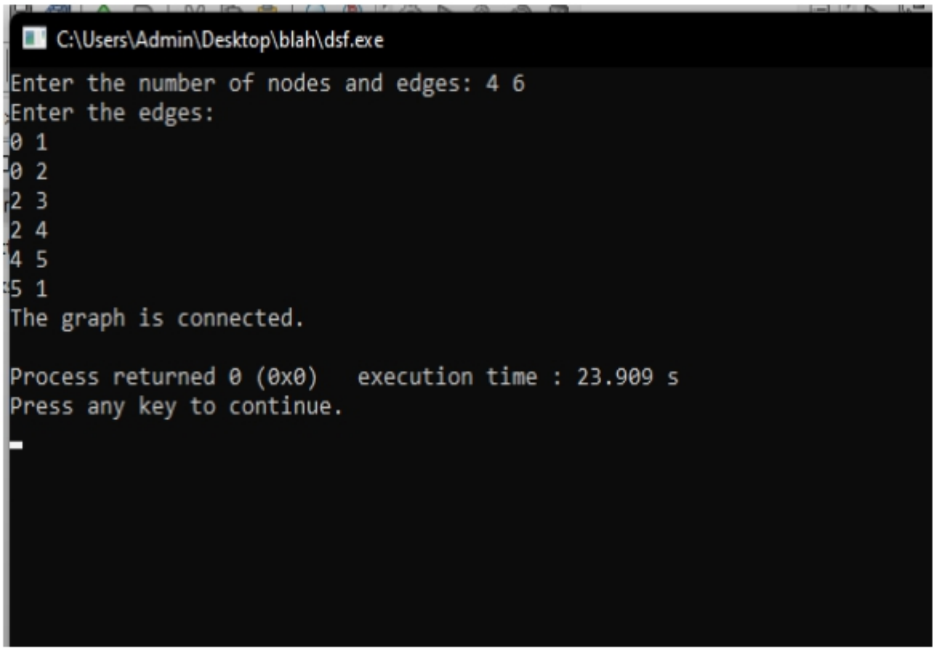
```
    printf("Following is Breadth First Traversal (starting from vertex
%d)\n", startVertex);

    Graph_BFS(g, startVertex);

    Graph_destroy(g);
    return 0;
}
```




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

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

#define MAX_NODES 100
#define MAX_EDGES 100

int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];

void DFS(int start, int n) {
    visited[start] = 1;
```

```
    for(int i = 0; i < n; i++) {
        if(graph[start][i] == 1 && !visited[i]) {
            DFS(i, n);
        }
    }
}

int isConnected(int n) {
    DFS(0, n);

    for(int i = 0; i < n; i++) {
        if(!visited[i]) {
            return 0;
        }
    }

    return 1;
}

int main() {
    int n, m;
    printf("Enter the number of nodes and edges: ");
    scanf("%d %d", &n, &m);

    printf("Enter the edges:\n");
    for(int i = 0; i < m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        graph[a][b] = 1; graph[b]
        [a] = 1;
    }

    if(isConnected(n)) {
        printf("The graph is connected.
    \n"); } else {
        printf("The graph is not connected.\n");
    }
    return 0;

}
```



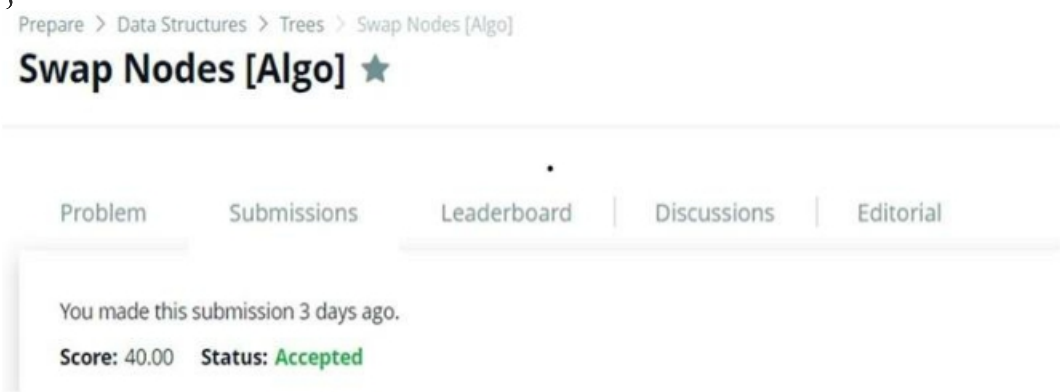

**3.(Hacker Rank-1) Complete the *swapNodes* function in the editor below. It should return a two-dimensional array where each element is an array of integers representing the node indices of an in-order traversal after a swap operation.**

```
#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* queries, int* result_rows, int*
result_columns) {
    // Build the tree
    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];
    }

    // Perform swaps and store results
    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 memory allocated for each row
    }
```

```
    free(result); // Free memory allocated for the result array

    // Free memory allocated for indexes and queries arrays
    for (int i = 0; i < n; i++) {
        free(indexes[i]);
    }
    free(indexes);
    free(queries);

    return 0;
}
```



## (WEEK 10)

**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.**

```
#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);
        // Print other details as
    needed } 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;
}
```

```
Inserted key 1234 at index 4
Inserted key 5678 at index 8
Inserted key 9876 at index 6
Key 5678 found at index 8
Key 1111 not found in hash table.

Process returned 0 (0x0)   execution time : 0.074 s
Press any key to continue.
```