VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Rukith Nayak(1BM23CS277)

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
September 2024-January 2025

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 NAME **(USN)**, who is 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 2024-25. 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. Lakshmi Neelima M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

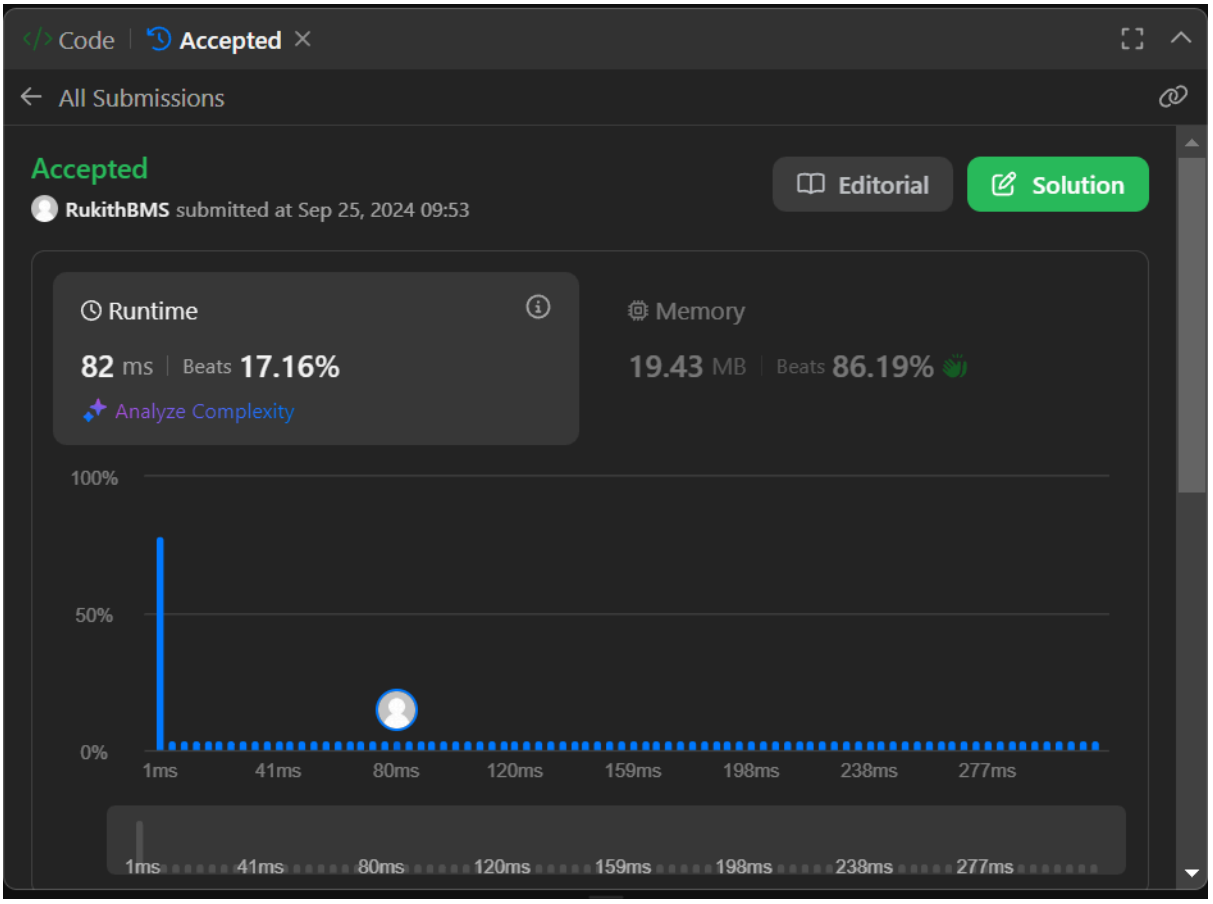
Index Sheet

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.

Leetcode 283: Move Zeroes

```
void moveZeroes(int* nums, int numsSize) {

    int count = 0;

    int j = 0;

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

        if(nums[i] == 0) {

            count++;

        }

        else{

            nums[j] = nums[i];

            j++;

        }

    }

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

        nums[j] = 0;

        j++;

    }

}
```


Code
Accepted
All Submissions
Accepted
RukithBMS submitted at Sep 25, 2024 09:53
Editorial
Solution
Runtime
82 ms | Beats 17.16%
Analyze Complexity
Memory
19.43 MB | Beats 86.19%
100%
50%
0%
1ms
41ms
80ms
120ms
159ms
198ms
238ms
277ms

Stack Operations

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int STACK[SIZE];
int top = -1;
void push(int value) {
   if (top == SIZE - 1){
      printf("Stack overflow!");
   }
   else {
      top++;
      STACK[top]=value;
      printf("%d pushed onto stack\n",value);
   }
}
void pop() {
   if(top==-1) {
      printf("Stack underflow!!");
   }
   else{
      int value = STACK[top];
      top--;
      printf("%d popped successfully!\n",value);
   }
}
void display(){
   if(top==-1){
      printf("stack is empty\n");
```

```
    }
    else{
        printf("stack elements :\n");
        for(int i=top;i>=0;i--){
            printf("%d",STACK[i]);
            printf("\n");
        }

    }
}
int main() {
    int choice,a;
    while(1) {
        printf(" 1.push\n 2.pop\n 3.display\n 4-exit\n");
        printf("CHOOSE OPERATION:\n");
        scanf("%d",&choice);
        switch(choice) {
            case 1:
                printf("Enter scan elements:");
                scanf("%d",&a);
                push(a);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
```

```
            printf("Exiting.....");
            exit(0);
            break;
        default :
            printf("Invalid choice! enter again");
      }
   }
   return 0;
}
```

OUTPUT:

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 10

10 pushed onto stack

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 20

20 pushed onto stack

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 30

30 pushed onto stack

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 40

40 pushed onto stack

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 50

50 pushed onto stack

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 1

Enter scan elements: 60

Stack overflow!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 3

stack elements:

50

40

30

20

10

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

50 popped successfully!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

40 popped successfully!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

30 popped successfully!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

20 popped successfully!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

10 popped successfully!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 2

Stack underflow!

1.push

2.pop

3.display

4.exit

CHOOSE OPERATION: 4

Exiting.....

Press any key to continue

Infix to Postfix

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

int index1 = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];

void infixToPostfix();
void push(char symbol);
char pop();
int pred(char symbol);

int main() {
   printf("Enter infix expression:\n");
   scanf("%s", infix);
   infixToPostfix();
   printf("\nInfix expression: %s", infix);
   printf("\nPostfix expression: %s\n", postfix);
   return 0;
}

void infixToPostfix() {
   length = strlen(infix);
   push('#'); // Push an initial dummy character to the stack
   while (index1 < length) {
      symbol = infix[index1];
      switch (symbol) {
         case '(':
            push(symbol);
```

```
            break;
        case ')':
            temp = pop();
            while (temp != '(') {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (pred(stack[top]) >= pred(symbol)) {
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
    }
    index1++;
}
while (top > 0) {
    temp = pop();
    postfix[pos++] = temp;
}
postfix[pos] = '\0';
```

```
}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

char pop() {
    char symb;
    symb = stack[top];
    top = top - 1;
    return symb;
}

int pred(char symbol) {
    int p;
    switch (symbol) {
        case '^':
            p = 3;
            break;
        case '*':
        case '/':
            p = 2;
            break;
        case '+':
        case '-':
            p = 1;
            break;
        case '(':
```

```
            p = 0;
            break;
        case '#':
            p = -1;
            break;
        default:
            p = -1;
    }
    return p;
}
```

OUTPUT:

Enter infix expression:

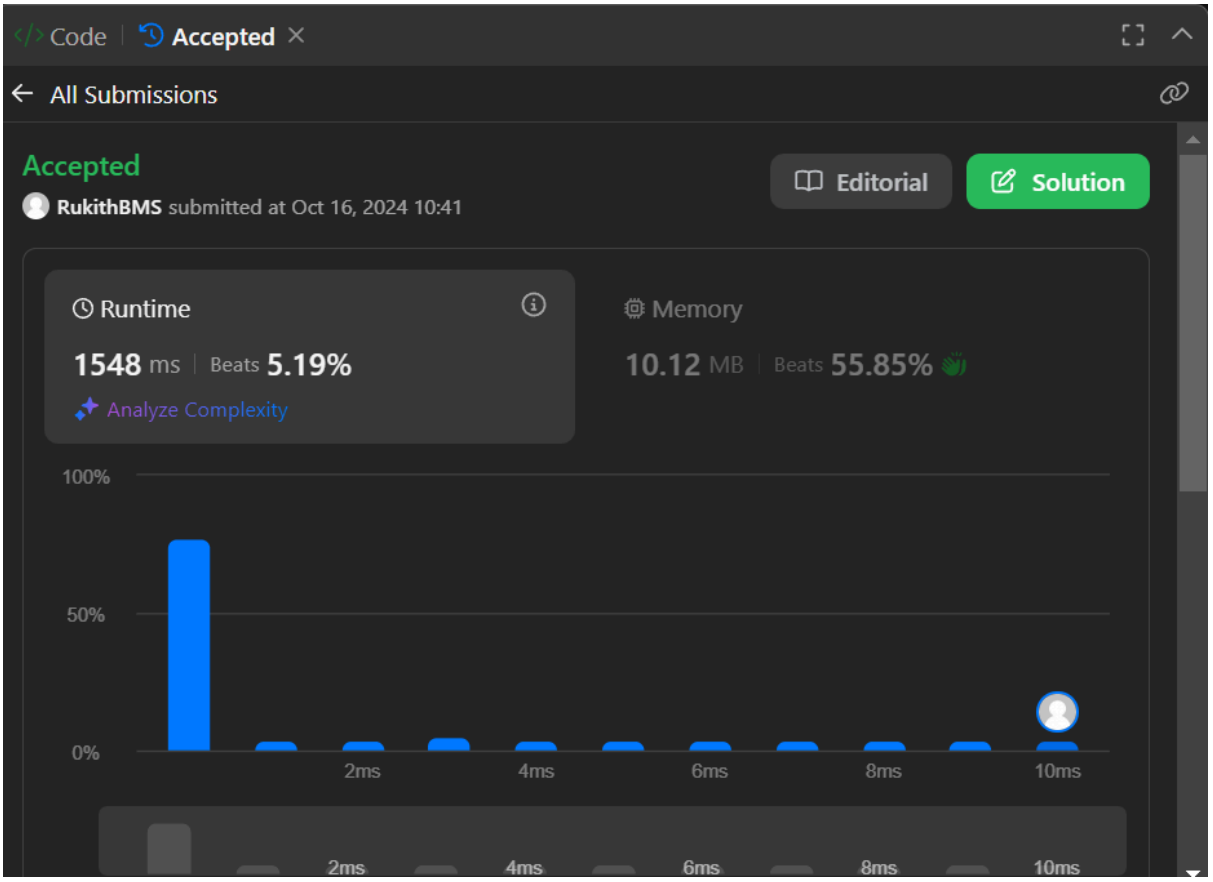
A+(B*C-(D/E^F)*G)*H

Infix expression: A+(B*C-(D/E^F)*G)*H

Postfix expression: ABC*DEF^/G*-H*+

Leetcode 169: Majority Element

```
int majorityElement(int* nums, int numsSize) {

    int num;

    int count = 0;

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

        if(nums[i] == num) {

            continue;

        }

        num = nums[i];



        for(int j = 0; j < numsSize; j++) {

            if(nums[j] == num) {

                count++;

            }

        }

        if(count > (numsSize / 2)) {

            break;

        }

        count = 0;

    }

    return num;

}
```


Code
Accepted
All Submissions
Accepted
RukithBMS submitted at Oct 16, 2024 10:41
Editorial
Solution
Runtime
1548 ms | Beats 5.19%
Analyze Complexity
Memory
10.12 MB | Beats 55.85%
100%
50%
0%
2ms
4ms
6ms
8ms
10ms

Linear Queue

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1;
int rear = -1;


void enqueue(int value) {
    if (rear == MAX - 1) {
        printf("Queue is full! Overflow !\n");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = value;
        printf("Inserted %d\n", value);
    }
}


int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty! Underflow!\n");
        return -1;
    } else {
        int item = queue[front];
```

```
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        printf("Deleted %d\n", item);
        return item;
    }
}


void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements are: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}


int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
```

```
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            enqueue(value);
            break;

        case 2:
            dequeue();
            break;

        case 3:
            display();
            break;

        case 4:
            printf("Exiting...\n");
            return 0;

        default:
            printf("Invalid choice! Please try again.\n");
    }
}
```

}

OUPUT:

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 2

Inserted 2

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 4

Inserted 4

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 5

Inserted 5

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice:

1

Enter the value to insert: 7

Inserted 7

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 6

Inserted 6

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 5

Queue is full! Overflow !

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 3

Queue elements are: 2 4 5 7 6

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Deleted 2

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Deleted 4

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Deleted 5

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Deleted 7

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Deleted 6

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Queue is empty! Underflow!

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 3

Queue is empty!

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 4

Exiting...

Circular Queue

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

#define SIZE 5
int items[SIZE], 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!!\n");
   } else {
      if (front == -1)
         front = 0;
      rear = (rear + 1) % SIZE;
      items[rear] = element;
      printf("\n%d is inserted into the queue.\n", element);
```

```
    }
}


int dequeue() {
    int element;
    if (isEmpty()) {
        printf("\nQueue is empty!!\n");
        return -1;
    } else {
        element = items[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % SIZE;
        }
        printf("\n%d is deleted from the queue.\n", element);
        return element;
    }
}


void display() {
    int i;
    if (isEmpty()) {
        printf("\nQueue is empty!!\n");
    } else {
        printf("\nFront position: %d\n", front);
```

```
        printf("Queue elements: ");
        for (i = front; i != rear; i = (i + 1) % SIZE) {
            printf("%d ", items[i]);
        }
        printf("%d\n", items[i]);
    }
}


int main() {
    int choice, element;
    while (1) {
        printf("\n****** Circular Queue Operations ******\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &element);
                enqueue(element);
                break;
            case 2:
                element = dequeue();
                if (element != -1)
```

```
                printf("%d element is deleted.\n", element);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid choice! Please try again.\n");
      }
   }
   return 0;
}
```

OUTPUT:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 2

2 is inserted into the queue.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 5

5 is inserted into the queue.

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 7

7 is inserted into the queue.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 6

6 is inserted into the queue.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 8

8 is inserted into the queue.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the element to insert: 6

Queue is full!!

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 3

Front position: 0

Queue elements: 2 5 7 6 8

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

2 is deleted from the queue.

2 element is deleted.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

5 is deleted from the queue.

5 element is deleted.

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

7 is deleted from the queue.

7 element is deleted.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

6 is deleted from the queue.

6 element is deleted.

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

8 is deleted from the queue.

8 element is deleted.

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 2

Queue is empty!!

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 3

Queue is empty!!

2. Dequeue

3. Display

4. Exit

Enter your choice: 4

Game of Two Stacks

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'twoStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER maxSum
 *  2. INTEGER_ARRAY a
```

```
 *  3. INTEGER_ARRAY b
 */

int twoStacks(int maxSum, int a_count, int* a, int b_count, int*
b) {
    int sumStackA[a_count + 1];
    int sumStackB[b_count + 1];
    int max = 0;
    sumStackA[0] = 0;
    for(int i = 0; i < a_count; i++) {
        sumStackA[i + 1] = sumStackA[i] + a[i];
    }
    sumStackB[0] = 0;
    for(int i = 0; i < b_count; i++) {
        sumStackB[i + 1] = sumStackB[i] + b[i];
    }
    int j = b_count;
    for(int i = 0; i <= a_count; i++) {
        if(sumStackA[i] > maxSum) {
            break;
        }
        while(j > 0 && sumStackA[i] + sumStackB[j] > maxSum) {
            j--;
        }
        max = (max > i + j) ? max : (i + j);
    }
```

```
    return max;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int g = parse_int(ltrim(rtrim(readline())));

    for (int g_itr = 0; g_itr < g; g_itr++) {
        char** first_multiple_input =
split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));

        int m = parse_int(*(first_multiple_input + 1));

        int maxSum = parse_int(*(first_multiple_input + 2));

        char** a_temp = split_string(rtrim(readline()));

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

        for (int i = 0; i < n; i++) {
            int a_item = parse_int(*(a_temp + i));
```

```
            *(a + i) = a_item;
        }

        char** b_temp = split_string(rtrim(readline()));

        int* b = malloc(m * sizeof(int));

        for (int i = 0; i < m; i++) {
            int b_item = parse_int(*(b_temp + i));

            *(b + i) = b_item;
        }

        int result = twoStacks(maxSum, n, a, m, b);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
```

```
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length,
stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length -
1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
```

```
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
```

```
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
```

```
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }
```

```
    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Linked List

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

struct node {
   int data;
   struct node *next;
};

struct node *start = NULL;

struct node *create_ll(struct node *start) {
   struct node *new_node, *ptr;
   int num;
   printf("Enter -1 to end\n");
   printf("Enter the data: ");
   scanf("%d", &num);
   while (num != -1) {
      new_node = (struct node *)malloc(sizeof(struct node));
      new_node->data = num;
      new_node->next = NULL;
      if (start == NULL) {
         start = new_node;
      } else {
         ptr = start;
         while (ptr->next != NULL) {
            ptr = ptr->next;
         }
         ptr->next = new_node;
```

```
        }
        printf("Enter the data: ");
        scanf("%d", &num);
    }
    return start;
}

struct node *display(struct node *start) {
    struct node *ptr;
    ptr = start;
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
    return start;
}

struct node *insert_beg(struct node *start) {
    struct node *new_node;
    int num;
    printf("Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
    return start;
}
```

```
struct node *insert_end(struct node *start) {
    struct node *ptr, *new_node;
    int num;
    printf("Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    ptr = start;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->next = new_node;
    return start;
}

struct node *insert_before(struct node *start) {
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("Enter the data: ");
    scanf("%d", &num);
    printf("Enter the value before which the data has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->data != val) {
        preptr = ptr;
```

```
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}

struct node *insert_after(struct node *start) {
    struct node *new_node, *ptr;
    int num, val;
    printf("Enter the data: ");
    scanf("%d", &num);
    printf("Enter the value after which the data has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->data != val) {
        ptr = ptr->next;
    }
    new_node->next = ptr->next;
    ptr->next = new_node;
    return start;
}

struct node *delete_beg(struct node *start) {
    struct node *ptr;
    ptr = start;
    start = start->next;
```

```
    free(ptr);
    return start;
}

struct node *delete_end(struct node *start) {
    struct node *ptr, *preptr;
    ptr = start;
    while (ptr->next != NULL) {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = NULL;
    free(ptr);
    return start;
}

struct node *delete_node(struct node *start) {
    struct node *ptr, *preptr;
    int val;
    printf("Enter the value of the node to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    if (ptr->data == val) {
        start = delete_beg(start);
        return start;
    } else {
        while (ptr->data != val) {
            preptr = ptr;
            ptr = ptr->next;
```

```
        }
        preptr->next = ptr->next;
        free(ptr);
        return start;
    }
}

struct node *delete_after(struct node *start) {
    struct node *ptr, *preptr;
    int val;
    printf("Enter the value after which the node has to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    while (preptr->data != val) {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = ptr->next;
    free(ptr);
    return start;
}

int main() {
    int option;
    do {
        printf("\n\n *****MAIN MENU*****");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Add a node at the beginning");
```

```
printf("\n 4: Add a node at the end");
printf("\n 5: Add a node before a given node");
printf("\n 6: Add a node after a given node");
printf("\n 7: Delete a node from the beginning");
printf("\n 8: Delete a node from the end");
printf("\n 9: Delete a given node");
printf("\n 10: Delete a node after a given node");
printf("\n 11: EXIT");
printf("\n\n Enter your option: ");
scanf("%d", &option);
switch (option) {
   case 1: start = create_ll(start);
           printf("LINKED LIST CREATED");
           break;
   case 2: start = display(start);
           break;
   case 3: start = insert_beg(start);
           break;
   case 4: start = insert_end(start);
           break;
   case 5: start = insert_before(start);
           break;
   case 6: start = insert_after(start);
           break;
   case 7: start = delete_beg(start);
           break;
   case 8: start = delete_end(start);
           break;
   case 9: start = delete_node(start);
```

```
                break;
            case 10: start = delete_after(start);
                break;
        }
    } while (option != 11);
    return 0;
}
```

==========

```
*****MAIN MENU*****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node before a given node
6: Add a node after a given node
7: Delete a node from the beginning
8: Delete a node from the end
9: Delete a given node
10: Delete a node after a given node
11: EXIT

Enter your option: 1
Enter -1 to end
Enter the data: 1
Enter the data: 2
Enter the data: 3
Enter the data: -1
```

LINKED LIST CREATED

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 3

Enter the data: 0

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

0 -> 1 -> 2 -> 3 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 4

Enter the data: 4

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

0 -> 1 -> 2 -> 3 -> 4 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 5

Enter the data: 2

Enter the value before which the data has to be inserted: 2

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

0 -> 1 -> 2 -> 2 -> 3 -> 4 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 6

Enter the data: 6

Enter the value after which the data has to be inserted: 4

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

0 -> 1 -> 2 -> 2 -> 3 -> 4 -> 6 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 7

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 8

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

1 -> 2 -> 2 -> 3 -> 4 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 9

Enter the value of the node to be deleted: 2

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 10

Enter the value after which the node has to be deleted: 2

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 2

1 -> 2 -> 4 -> NULL

*****MAIN MENU*****

1: Create a list

2: Display the list

3: Add a node at the beginning

4: Add a node at the end

5: Add a node before a given node

6: Add a node after a given node

7: Delete a node from the beginning

8: Delete a node from the end

9: Delete a given node

10: Delete a node after a given node

11: EXIT

Enter your option: 11

Process returned 0 (0x0) execution time : 231.454 s

Press any key to continue.

Linked List Implementation

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

// Node structure for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Insert a node at the end of the linked list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next) {
        temp = temp->next;
    }
```

```
    temp->next = newNode;
}

// Display the linked list
void displayList(struct Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Sort the linked list
void sortList(struct Node** head) {
    if (*head == NULL || (*head)->next == NULL) return;

    struct Node* i;
    struct Node* j;
    for (i = *head; i->next; i = i->next) {
        for (j = i->next; j; j = j->next) {
            if (i->data > j->data) {
                int temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}
```

```
// Reverse the linked list
void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

// Concatenate two linked lists
void concatenateLists(struct Node** head1, struct Node** head2) {
    if (*head1 == NULL) {
        *head1 = *head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = *head2;
}

// Stack
void push(struct Node** stack, int data) {
```

```
    struct Node* newNode = createNode(data);
    newNode->next = *stack;
    *stack = newNode;
}

int pop(struct Node** stack) {
    if (*stack == NULL) {
        printf("Stack underflow\n");
        return -1;
    }
    int data = (*stack)->data;
    struct Node* temp = *stack;
    *stack = (*stack)->next;
    free(temp);
    return data;
}

// Queue
void enqueue(struct Node** queue, int data) {
    insertEnd(queue, data);
}

int dequeue(struct Node** queue) {
    if (*queue == NULL) {
        printf("Queue underflow\n");
        return -1;
    }
    int data = (*queue)->data;
    struct Node* temp = *queue;
```

```
    *queue = (*queue)->next;
    free(temp);
    return data;
}

// display the stack
void displayStack(struct Node* stack) {
    if (stack == NULL) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack: ");
    while (stack) {
        printf("\n%d ", stack->data);
        stack = stack->next;
    }
    printf("\n");
}

// display the queue
void displayQueue(struct Node* queue) {
    if (queue == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    while (queue) {
        printf("%d ", queue->data);
        queue = queue->next;
```

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

// Main function
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    struct Node* stack = NULL;
    struct Node* queue = NULL;

    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1) Insert into Linked List\n");
        printf("2) Display Linked List\n");
        printf("3) Sort Linked List\n");
        printf("4) Reverse Linked List\n");
        printf("5) Concatenate Two Linked Lists\n");
        printf("6) Push to Stack\n");
        printf("7) Pop from Stack\n");
        printf("8) Enqueue to Queue\n");
        printf("9) Dequeue from Queue\n");
        printf("10) Display Stack\n");
        printf("11) Display Queue\n");
        printf("12) Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("Enter value to insert into Linked List: ");
        scanf("%d", &value);
        insertEnd(&list1, value);
        break;
    case 2:
        printf("Linked List: ");
        displayList(list1);
        break;
    case 3:
        sortList(&list1);
        printf("Linked List sorted.\n");
        break;
    case 4:
        reverseList(&list1);
        printf("Linked List reversed.\n");
        break;
    case 5:
        printf("Enter values for the second list (terminate with -1):\n");
        while (1) {
            scanf("%d", &value);
            if (value == -1) break;
            insertEnd(&list2, value);
        }
        concatenateLists(&list1, &list2);
        printf("Lists concatenated.\n");
        break;
```

```
case 6:
    printf("Enter value to push onto Stack: ");
    scanf("%d", &value);
    push(&stack, value);
    break;
case 7:
    value = pop(&stack);
    if (value != -1) {
        printf("Popped from Stack: %d\n", value);
    }
    break;
case 8:
    printf("Enter value to enqueue to Queue: ");
    scanf("%d", &value);
    enqueue(&queue, value);
    break;
case 9:
    value = dequeue(&queue);
    if (value != -1) {
        printf("Dequeued from Queue: %d\n", value);
    }
    break;
case 10:
    displayStack(stack);
    break;
case 11:
    displayQueue(queue);
    break;
case 12:
```

```
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}
```

```
Menu:
1) Insert into Linked List
2) Display Linked List
3) Sort Linked List
4) Reverse Linked List
5) Concatenate Two Linked Lists
6) Push to Stack
7) Pop from Stack
8) Enqueue to Queue
9) Dequeue from Queue
10) Display Stack
11) Display Queue
12) Exit
Enter your choice: 1
Enter value to insert into Linked List: 1

Menu:
1) Insert into Linked List
```

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 1

Enter value to insert into Linked List: 7

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 1

Enter value to insert into Linked List: 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 1

Enter value to insert into Linked List: 0

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 1

Enter value to insert into Linked List: 8

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 2

Linked List: 1 -> 7 -> 3 -> 0 -> 8 -> NULL

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 3

Linked List sorted.

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 2

Linked List: 0 -> 1 -> 3 -> 7 -> 8 -> NULL

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 4

Linked List reversed.

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 2

Linked List: 8 -> 7 -> 3 -> 1 -> 0 -> NULL

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 5

Enter values for the second list (terminate with -1):

1

2

3

4

5

-1

Lists concatenated.

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 2

Linked List: 8 -> 7 -> 3 -> 1 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> NULL

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 6

Enter value to push onto Stack: 1

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 6

Enter value to push onto Stack: 2

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 6

Enter value to push onto Stack: 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 10

Stack:

3

2

1

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 8

Enter value to enqueue to Queue: 1

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 8

Enter value to enqueue to Queue: 2

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 8

Enter value to enqueue to Queue: 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 11

Queue: 1 2 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 7

Popped from Stack: 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 10

Stack:

2

1

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 9

Dequeued from Queue: 1

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice: 11

Queue: 2 3

Menu:

1) Insert into Linked List

2) Display Linked List

3) Sort Linked List

4) Reverse Linked List

5) Concatenate Two Linked Lists

6) Push to Stack

7) Pop from Stack

8) Enqueue to Queue

9) Dequeue from Queue

10) Display Stack

11) Display Queue

12) Exit

Enter your choice:12

Exiting program.

Process returned 0 (0x0) execution time : 475.729 s

Press any key to continue.

Double Linked List

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


struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}


void append(struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);
    struct Node* temp = *head_ref;

    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }
```

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

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


void insertLeft(struct Node** head_ref, int target, int data) {
    struct Node* temp = *head_ref;

    while (temp != NULL && temp->data != target) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Node with value %d not found.\n", target);
        return;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp;
    newNode->prev = temp->prev;

    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    } else {
```

```
        *head_ref = newNode;
    }
    temp->prev = newNode;
}


void deleteNode(struct Node** head_ref, int target) {
    struct Node* temp = *head_ref;

    while (temp != NULL && temp->data != target) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Node with value %d not found.\n", target);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head_ref = temp->next;
    }

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

    free(temp);
```

```
}


void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" <-> ");
        }
        temp = temp->next;
    }
    printf("\n");
}


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

    do {
        printf("\nMenu:\n");
        printf("1. Append a node\n");
        printf("2. Insert a node to the left of a given node\n");
        printf("3. Delete a node by value\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("Enter the value to append: ");
        scanf("%d", &data);
        append(&head, data);
        break;

    case 2:
        printf("Enter the target value: ");
        scanf("%d", &target);
        printf("Enter the value to insert: ");
        scanf("%d", &data);
        insertLeft(&head, target, data);
        break;

    case 3:
        printf("Enter the value to delete: ");
        scanf("%d", &target);
        deleteNode(&head, target);
        break;

    case 4:
        printf("List contents: ");
        display(head);
        break;

    case 5:
        printf("Exiting program.\n");
```

```
            break;

        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 5);

return 0;
```

OUTPUT:

-- Menu --

1. Append a node
2. Insert a node to the left of a given node
3. Delete a node by value
4. Display the list
5. Exit

Enter your choice: 1

Enter the value to append: 1

-- Menu --

1. Append a node
2. Insert a node to the left of a given node
3. Delete a node by value
4. Display the list
5. Exit

Enter your choice: 1

Enter the value to append: 5

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 1

Enter the value to append: 7

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 1

Enter the value to append: 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 3

Enter the value to delete: 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 1

Enter the value to append: 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 4

List contents: 1 <-> 5 <-> 7 <-> 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 3

Enter the value to delete: 7

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 4

List contents: 1 <-> 5 <-> 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 2

Enter the target value: 1

Enter the value to insert: 8

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 4

List contents: 8 <-> 1 <-> 5 <-> 9

-- Menu --

1. Append a node

2. Insert a node to the left of a given node

3. Delete a node by value

4. Display the list

5. Exit

Enter your choice: 5

Exiting program.

Process returned 0 (0x0)   execution time : 73.135 s

Press any key to continue.

Leetcode 234: Palindrome Linked List

```
bool isPalindrome(struct ListNode* head) {
    if (!head || !head->next) {
        return true;
    }

    struct ListNode* slow = head;
    struct ListNode* fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }

    struct ListNode* prev = NULL;
    struct ListNode* current = slow;
    while (current) {
        struct ListNode* nextTemp = current->next;
        current->next = prev;
        prev = current;
        current = nextTemp;
    }

    struct ListNode* firstHalf = head;
    struct ListNode* secondHalf = prev;
    while (secondHalf) {
        if (firstHalf->val != secondHalf->val) {
            return false;
        }
        firstHalf = firstHalf->next;
```

```
        secondHalf = secondHalf->next;
    }

    return true;
}
```

Binary Search Tree

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

typedef struct BST {
    int data;
    struct BST *left;
    struct BST *right;
} node;

node *create() {
    node *temp;
    printf("Enter data: ");
    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}

void insert(node *root, node *temp) {
    if (temp->data < root->data) {
        if (root->left != NULL)
            insert(root->left, temp);
        else
            root->left = temp;
    } else if (temp->data > root->data) {
        if (root->right != NULL)
            insert(root->right, temp);
        else
```

```
            root->right = temp;
    }
}

void preorder(node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void postorder(node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main() {
```

```
    char ch;
    node *root = NULL, *temp;

    printf("Do you want to enter data? ");
    printf("\Y/N:");
    scanf("%c", &ch);  // Initial check to start the loop

    while (ch == 'y' || ch == 'Y') {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);

        printf("\nEnter more data?\nY/N:");
        getchar(); // To consume the newline character left by scanf
        scanf("%c", &ch);  // Read the user's input
    }

    printf("\nPreorder Traversal: ");
    preorder(root);
    printf("\nInorder Traversal: ");
    inorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);

    return 0;
}
```

OUPUT:

Do you want to enter data? Y/N:y

Enter data: 5

Enter more data?

Y/N:y

Enter data: 3

Enter more data?

Y/N:y

Enter data: 7

Enter more data?

Y/N:y

Enter data: 2

Enter more data?

Y/N:y

Enter data: 4

Enter more data?

Y/N:n

Preorder Traversal: 5 3 2 4 7

Inorder Traversal: 2 3 4 5 7

Postorder Traversal: 2 4 3 7 5

Process returned 0 (0x0) execution time : 25.126 s

Press any key to continue.

Breadth First Search

```
#include <stdio.h>

void bfs(int adj[10][10], int n, int source){
    int que[10];
    int front=0,rear=-1;
    int visited[10]={0};
    int node;
    printf("The nodes visited from %d: ", source);
    que[++rear]=source;
    visited[source]=1;
    printf("%d",source);
    while(front<=rear){
        int u= que[front++];
        for(int v=0; v<n; v++){
            if(adj[u][v]==1){
                if(visited[v]==0){
                    printf("%d",v);
                    visited[v]=1;
                    que[++rear]=v;
                }
            }
        }
    }
    printf("\n");
}

int main() {
    int n;
```

```
    int adj[10][10];
    int source;
    printf("enter number of nodes \n");
    scanf("%d",&n);
    printf("Enter Adjacency Matrix \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d",&adj[i][j]);
        }
    }
    for(source=0; source<n; source++){
        bfs(adj,n,source);
    }

    return 0;
}
```

OUPUT:

```
enter number of nodes
4
Enter Adjacency Matrix
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The nodes visited from 0: 0123
The nodes visited from 1: 1023
The nodes visited from 2: 2013
The nodes visited from 3: 3120
```

Process returned 0 (0x0)   execution time : 51.967 s

Press any key to continue.

Leetcode 112: Path Sum

```
bool hasPathSum(struct TreeNode* root, int targetSum) {
    if (root == NULL) {
        return false;
    }

    targetSum -= root->val;

    if (root->left == NULL && root->right == NULL) {
        return targetSum == 0;
    }

    return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);
}
```

Depth First Search

```
#include <stdio.h>

int adjacencyMatrix[10][10];
int visited[10];
int nodes;


void DFS(int vertex) {
   visited[vertex] = 1;
   printf("%d ", vertex);

   for (int i = 0; i < nodes; i++) {
      if (adjacencyMatrix[vertex][i] == 1 && !visited[i]) {
         DFS(i);
      }
   }
}


int isConnected() {

   for (int i = 0; i < nodes; i++) {
      visited[i] = 0;
   }


   DFS(0);


   for (int i = 0; i < nodes; i++) {
      if (!visited[i]) {
         return 0;
      }
   }
   return 1;
}

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

   printf("Enter the adjacency matrix:\n");
   for (int i = 0; i < nodes; i++) {
      for (int j = 0; j < nodes; j++) {
         scanf("%d", &adjacencyMatrix[i][j]);
      }
```

```
    if (isConnected()) {
        printf("\nThe graph is connected.\n");
    } else {
        printf("\nThe graph is not connected.\n");
    }

    return 0;
}
```

OUTPUT:

Enter the number of nodes: 3

Enter the adjacency matrix:

0

1

0

1

0

1

0

1

0

0 1 2

The graph is connected.

Linear Probing

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

#define MAX_SIZE 100

int hashTable[MAX_SIZE];
int m;

void initializeHashTable() {
   for (int i = 0; i < m; i++) {
      hashTable[i] = -1;
   }
}


int hashFunction(int key) {
   return key % m;
}


void insertKey(int key) {
   int address = hashFunction(key);

   int originalAddress = address;
   while (hashTable[address] != -1) {
      printf("Collision detected at address %d for key %d. Probing...\n", address, key);
      address = (address + 1) % m;
      if (address == originalAddress) {
         printf("Hash table is full! Cannot insert key %d.\n", key);
         return;
      }
   }
   hashTable[address] = key;
   printf("Key %d inserted at address %d.\n", key, address);
}


void displayHashTable() {
   printf("\nHash Table:\n");
   for (int i = 0; i < m; i++) {
      if (hashTable[i] == -1)
         printf("Address %d: EMPTY\n", i);
      else
         printf("Address %d: %d\n", i, hashTable[i]);
   }
}

int main() {
```

```
    int n;
    int key;

    printf("Enter the number of memory locations in the hash table (m): ");
    scanf("%d", &m);
    if (m > MAX_SIZE) {
        printf("Error: m exceeds the maximum size %d.\n", MAX_SIZE);
        return 1;
    }

    initializeHashTable();

    printf("Enter the number of employee keys (n): ");
    scanf("%d", &n);

    printf("Enter %d employee keys (4-digit integers):\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &key);
        insertKey(key);
    }

    displayHashTable();

    return 0;
}
```

OUTPUT:

```
Enter the number of memory locations in the hash table (m): 5
Enter the number of employee keys (n): 4
Enter 4 employee keys (4-digit integers):
0001
Key 1 inserted at address 1.
0002
Key 2 inserted at address 2.
0001
Collision detected at address 1 for key 1. Probing...
Collision detected at address 2 for key 1. Probing...
Key 1 inserted at address 3.
0003
Collision detected at address 3 for key 3. Probing...
Key 3 inserted at address 4.

Hash Table:
Address 0: EMPTY
Address 1: 1
Address 2: 2
Address 3: 1
Address 4: 3
```