B.M.S. COLLEGE OF ENGINEERING
Basavanagudi, Bengaluru- 560019
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB REPORT

On

Data Structures
(23CS3PCDST)

Submitted By :
NAVNEETH KS
1BM22CS174

In partial fulfilment of
BACHELOR OF ENGINEERING
In
COMPUTER SCIENCE AND ENGINEERING
B.M.S. COLLEGE OF ENGINEERING
Basavanagudi, Bengaluru- 560019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the Lab work entitled “Data Structures (22CS3PCDST)” conducted by **NAVNEETH KS (1BM22CS174),** who is Bonafide student at **B.M.S.College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** during the academic year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of a Data Structures (23CS3PCDST) work prescribed for the said degree.

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

INDEX

Course Outcomes:

CO1	**Apply the concept of linear and nonlinear data structures.**
CO2	**Analyse data structure operations for a given problem**

CO3	**Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.**
CO4	**Conduct practical experiments for demonstrating the operations of different data structures.**

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int top = -1;
int stack[SIZE];

void push(int element);
int pop();
void display();

int main() {
    int choice, element;

    do {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to push: ");
                scanf("%d", &element);
                push(element);
                break;
            case 2:
                element = pop();
```

```
            if (element != -1) {
                printf("Popped element: %d\n", element);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }

  } while (choice != 4);

  return 0;
}

void push(int element) {
  if (top == SIZE - 1) {
    printf("Stack Overflow. Cannot push element %d.\n", element);
  } else {
    top++;
    stack[top] = element;
    printf("Element %d pushed onto the stack.\n", element);
  }
}

int pop() {
  if (top == -1) {
    printf("Stack Underflow. Cannot pop from an empty stack.\n");
    return -1; // indicating failure
  } else {
    int element = stack[top];
    top--;
    return element;
  }
}

void display() {
  if (top == -1) {
    printf("Stack is empty.\n");
  } else {
    printf("Stack elements: ");
    for (int i = 0; i <= top; i++) {
        printf("%d ", stack[i]);
    }
    printf("\n");
  }
}
```

OUTPUT

```
Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter element to push: 23

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter element to push: -45

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 12
Invalid choice. Please enter a valid option.

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 3
Stack elements: 23 -45

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter element to push: 12

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 3
Stack elements: 23 -45 12

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 2
Popped element: 12

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 4
Exiting program.

Process returned 0 (0x0)   execution time : 78.017 s
Press any key to continue.
```

Lab program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and /

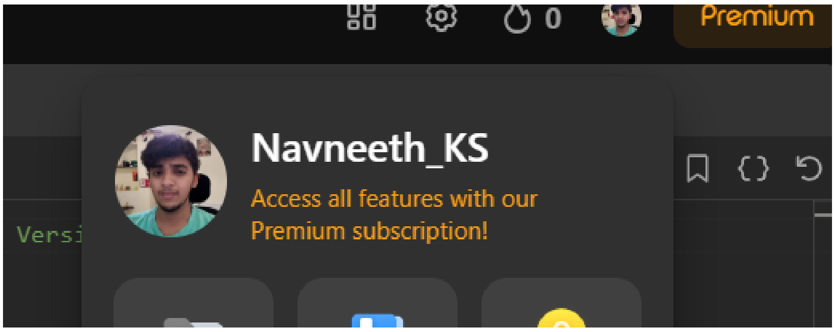
(divide)

```
#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--];
            }
            stack[++top] = infix[i];
        } else if (infix[i] == '(') {
            stack[++top] = infix[i];
        } else if (infix[i] == ')') {
            while (top >= 0 && 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;
}
```

```
Enter infix expression: 2+7*(5%4)-6
Postfix expression: 2754%*+6-

Process returned 0 (0x0)   execution time : 11.875 s
Press any key to continue.
```

Demonstration of account creation on LeetCode platform
Program - Leetcode platform



Lab program 3:

WAP to simulate the working of a 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>
#include <stdlib.h>

#define MAX 3
```

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

void insert(int item){
    if (rear == MAX - 1) {
        printf("Overflow\n");
        exit(EXIT_FAILURE);
    } else {
        if (rear == -1 && front == -1) {
            front = rear = 0;
            queue[rear] = item;
        } else {
            rear = rear + 1;
            queue[rear] = item;
        }
    }
}
int delete () {
    if (front == -1 || front > rear) {
        return -1;
    } else {
        return queue[front++];
    }
}
int main() {
    int n;
    do {
        printf("1. Insert element\n2. Delete element\n3. Exit\n");
        scanf("%d", &n);

        switch (n) {
            case 1:
                int ele;
                printf("Enter the element: ");
                scanf("%d", &ele);
                insert(ele);
                break;

            case 2:
                int d = delete();
                if (d == -1) {
                    printf("Underflow\n");
                    exit(EXIT_FAILURE);
                }
                printf("The element deleted is: %d\n", d);
                break;
            case 3:
                printf("Exiting the program\n");
                break;
            default:
```

```
            printf("Please enter the right choice\n");
        }
    } while (n != 3);
    return 0;
}
```

```
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 23
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 54
1. Insert element
2. Delete element
3. Exit
2
The element deleted is: 23
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 65
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 45
Overflow
```

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

#define MAX 6

int cq[MAX];
int front = -1, rear = -1;

bool is_full() {
    return (rear + 1) % MAX == front;
}
```

```
bool is_empty() {
    return front == -1 && rear == -1;
}

void insert(int item) {
    if (is_full()) {
        printf("Overflow: Circular queue is full.\n");
        // Handle overflow appropriately, e.g., return without enqueueing
        return;
    }

    if (is_empty()) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % MAX;
    }

    cq[rear] = item;
    printf("Enqueued: %d\n", item);
}

int dequeue() {
    if (is_empty()) {
        printf("Underflow: Circular queue is empty.\n");
        return -1;
    }

    int deletedItem = cq[front];

    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front + 1) % MAX;
    }

    printf("Dequeued: %d\n", deletedItem);
    return deletedItem;
}

int main() {
    int n, ele;
    do {
        printf("\n1. Insert\n2. Delete\n3. Exit\n");
        scanf("%d", &n);
        switch (n) {
            case 1:
                printf("Enter the element to be inserted: ");
                scanf("%d", &ele);
                insert(ele);
                break;
```

```
        case 2:
          {
            int deletedItem = dequeue();
            if (deletedItem != -1) {
              printf("The element %d is removed.\n", deletedItem);
            }
          }
          break;
        case 3:
          printf("Thanks\n");
          break;
        default:
          printf("Please enter the right option.\n");
      }
    } while (n != 3);

    return 0;
}
```

```
1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 23
Enqueued: 23

1. Insert
2. Delete
3. Exit
2
Dequeued: 23
The element 23 is removed.

1. Insert
2. Delete
3. Exit
2
Underflow: Circular queue is empty.

1. Insert
2. Delete
3. Exit
```

Lab program 4:

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.
Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* newNode(int data) {
    struct Node* ptr = (struct Node*)malloc(sizeof(struct Node));
    ptr->data = data;
    ptr->next = NULL;
    return ptr;
}
struct Node* insertBeg(struct Node* first, int data) {
    struct Node* ptr = newNode(data);
    ptr->next = first;
    first = ptr;
    return first;
}
struct Node* insertMid(struct Node* first, int data, int pos) {
    struct Node* ptr = newNode(data);
    if (pos == 0) {
        return insertBeg(first, data);
    }
    struct Node* p1 = first;
```

```
    struct Node* p2 = first;
    int i = 0;
    while (i != pos && p2!= NULL) {
        p1 = p2;
        p2 = p2->next;
        i++;
    }
    if (i != pos) {
        printf("Position %d is invalid\n", pos);
        return first;
    }
    p1->next = ptr;
    ptr->next = p2;
    return first;
}
struct Node* insertEnd(struct Node* first, int data) {
    struct Node* ptr = newNode(data);
    struct Node* p = first;
    while (p->next != NULL) {
        p = p->next;
    }
    p->next = ptr;
    ptr->next = NULL;
    return first;
}
void display(struct Node* first) {
    if(first==NULL){
        printf("Empty List\n");
    }
    struct Node* p = first;
    while (p != NULL) {
        printf("Element %d\n", p->data);
        p = p->next;
    }
}
int main() {
    struct Node* head = NULL;
    int choice, data, pos;
    while (1) {
        printf("\nOptions:\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at a specific position\n");
        printf("3. Insert at the end\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
```

```
            head = insertBeg(head, data);
            break;
        case 2:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
            head = insertMid(head, data, pos);
            break;
        case 3:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            head = insertEnd(head, data);
            break;
        case 4:
            display(head);
            break;
        case 5:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
  }
  return 0;
```

```
Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 1
Enter the new element to insert at the beginning: 1

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 3
Enter the new element to insert: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 2
Enter the new element to insert: 3
Enter the index to insert at: 1

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 4
Linked List:
Element: 1
Element: 3
Element: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 5
Quitting the program.

Process returned 0 (0x0)   execution time : 53.094 s
Press any key to continue.
```

Program - Leetcode platform

Min Stack

```
typedef struct {
  int *array;
  int top_index;
  int *min_array;
} MinStack;


MinStack* minStackCreate() {
    MinStack* m = malloc(sizeof(MinStack));
    m->array = (int*)calloc(300001,sizeof(int));
    m->min_array = (int*)calloc(300001,sizeof(int));
    m->top_index = 0;
    return m;
1
```

```
}

void minStackPush(MinStack* m, int val) {
   m->array[m->top_index] = val;
   if(m->top_index ==0 || m->min_array[m->top_index -1] > val){
     m->min_array[m->top_index] = val;
   }
   else{
     m->min_array[m->top_index] = m->min_array[m->top_index-1];
   }
   m->top_index++;
}

void minStackPop(MinStack* m) {
   m->top_index--;
}

int minStackTop(MinStack* m) {
   return m->array[m->top_index-1];
}

int minStackGetMin(MinStack* m) {
   return m->min_array[m->top_index-1];
}

void minStackFree(MinStack* m) {
   free(m->array);
   free(m->min_array);
   free(m);
}
```

Lab program 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;
};
struct Node* newNode(int data) {
    struct Node* ptr = (struct Node*)malloc(sizeof(struct Node));
```

```
    ptr->data = data;
    ptr->next = NULL;
    return ptr;
}
struct Node* insertMid(struct Node* first, int data, int pos) {
    struct Node* ptr = newNode(data);
    if (pos == 0) {
        ptr->next = first;
        first = ptr;
        return first;
    }
    struct Node* p1 = first;
    struct Node* p2 = first;
    int i = 0;
    while (i != pos && p2!= NULL) {
        p1 = p2;
        p2 = p2->next;
        i++;
    }
    if (i != pos) {
        printf("Position %d is invalid\n", pos);
        return first;
    }
    p1->next = ptr;
    ptr->next = p2;
    return first;
}
struct Node* DeleteBeg(struct Node* first) {
    if(first==NULL){
        printf("Linked List Empty\n");
        return first;
    }
    struct Node* ptr = first;
    first = first->next;
    free(ptr);
    return first;
}
struct Node* DeleteMid(struct Node* first,int pos) {
    if (pos == 0) {
        return DeleteBeg(first);
    }
    struct Node* p1 = first;
    struct Node* p2 = first;
    int i = 0;
    while (i != pos && p2->next != NULL) {
        p1 = p2;
        p2 = p2->next;
        i++;
    }
    if (i != pos) {
        printf("Position %d is invalid\n", pos);
```

```
        return first;
    }
    p1->next = p2->next;
    free(p2);
    return first;
}
struct Node* DeleteEnd(struct Node* first) {
    struct Node* p = first;
    while (p->next->next != NULL) {
        p = p->next;
    }
    free(p->next);
    p->next = NULL;
    return first;
}
void display(struct Node* first) {
    if(first==NULL){
        printf("Empty List\n");
    }
    struct Node* p = first;
    while (p != NULL) {
        printf("Element %d\n", p->data);
        p = p->next;
    }
}
int main(){
    struct Node* head = NULL;
    int choice, data, pos;
    while (1) {
        printf("\nOptions:\n");
        printf("1. Insert at a specific position\n");
        printf("2. Delete at start\n");
        printf("3. Delete at specified 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 data to insert: ");
                scanf("%d", &data);
                printf("Enter position: ");
                scanf("%d", &pos);
                head = insertMid(head, data, pos);
                break;
            case 2:
                head = DeleteBeg(head);
                break;
            case 3:
                printf("Enter position: ");
```

```
            scanf("%d", &pos);
            head = DeleteMid(head,pos);
            break;
        case 4:
            head = DeleteEnd(head);
            break;
        case 5:
            display(head);
            break;
        case 6:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please enter a valid option.\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
```

Program - Leetcode platform

REVERSING A LINKED LIST:

1

```
class Solution {
public:

    ListNode* reverseLL(ListNode* head){

        ListNode* prev = NULL;

        ListNode* curr = head;

        while(curr!=NULL){
            ListNode* currkanext = curr->next;
            curr->next = prev;
            prev = curr;
            curr = currkanext;
        }

        return prev;
    }

    ListNode* reverseBetween(ListNode* head, int left, int right) {

        if(!head || head->next==NULL){
            return head;
        }

        left--;
        right--;

        ListNode* dummy1 = new ListNode(-1);
        ListNode* dummy2 = new ListNode(-1);
        ListNode* dummy3 = new ListNode(-1);

        ListNode* head1 = dummy1;

        ListNode* head2 = dummy2;

        ListNode* head3 = dummy3;

        ListNode* temp = head;
        for(int i=0;i<left;i++){
            dummy1->next = new ListNode(temp->val);
            temp = temp->next;
            dummy1 = dummy1->next;
        }

        for(int i=left;i<=right;i++){
            dummy2->next = new ListNode(temp->val);
            temp = temp->next;
            dummy2 = dummy2->next;
        }
```

```
        while(temp!=NULL){
            dummy3->next = new ListNode(temp->val);
            temp = temp->next;
            dummy3 = dummy3->next;
        }

        ListNode* dummy = head2->next;

        ListNode* node = reverseLL(head2->next);

        dummy1->next = node;

        cout<<dummy2->val<<endl;

        dummy->next = head3->next;

        return head1->next;

    }
};
```

Lab program 6:

WAP to Implement Single Link 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* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

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

struct Node* concatenateLists(struct Node* list1, struct Node* list2) {
    if (list1 == NULL) {
        return list2;
    }

    struct Node* current = list1;
    while (current->next != NULL) {
        current = current->next;
    }

    current->next = list2;
    return list1;
}

int main() {
    struct Node* list1 = createNode(1);
    list1->next = createNode(2);
    list1->next->next = createNode(3);

    struct Node* list2 = createNode(4);
    list2->next = createNode(5);
    list2->next->next = createNode(6);

    printf("Original List 1: ");
    printList(list1);
    printf("Original List 2: ");
    printList(list2);

    list1 = concatenateLists(list1, list2);
```

```
    printf("Concatenated List: ");
    printList(list1);

    free(list1);
    free(list2);

    return 0;
}
```

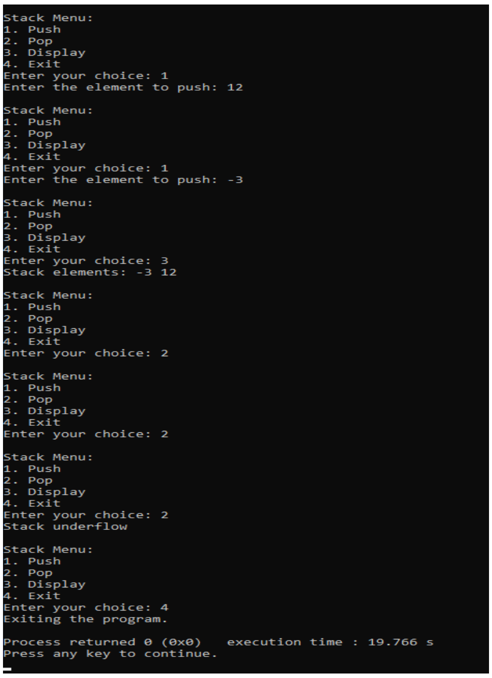
```
Original List 1: 1 -> -2 -> 3 -> NULL
Original List 2: 4 -> 10 -> 6 -> NULL
Concatenated List: 1 -> -2 -> 3 -> 4 -> 10 -> 6 -> NULL

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

WAP to Implement Single Link List to simulate Stack and Queue Operations.

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

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

struct Node* push(struct Node* top, int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    if(top==NULL){
        newNode->next=NULL;
        top = newNode;
        return top;
    }
    newNode->next = top;
    top = newNode;
    return top;
}

struct Node* pop(struct Node* top) {
    if(top==NULL){
        printf("Under-flow");
        exit(1);
    }
    struct Node *ptr = top;
```

```
    top = ptr->next;
    free(ptr);
    return top;

}

void display(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    struct Node* current = top;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Node* top = NULL;
    int choice, element;

    do {
        printf("\nStack Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\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 push: ");
                scanf("%d", &element);
                top = push(top, element);
                break;
            case 2:
                top = pop(top);
                break;
            case 3:
                display(top);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
```

```
    }
  } while (choice != 4);

  while (top != NULL) {
    struct Node* temp = top;
    top = top->next;
    free(temp);
  }

  return 0;
}
```



Lab program 7:

WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

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

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

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

struct Node* Insert(struct Node* first, int data, int pos) {
    struct Node* ptr = newNode(data);
    if (pos == 0) {
        ptr->next = first;
        if (first != NULL) {
            first->prev = ptr;
        }
        return ptr;  // Update to return the new head of the list
    }
    struct Node* p2 = first;
    struct Node* p1 = first;
    int i = 0;
    while (i != pos + 1 && p2 != NULL) {
        p1 = p2;
        p2 = p2->next;
        i++;
    }
    if (i < pos) {
        printf("Invalid position\n");
        free(ptr);  // Free the allocated memory before returning
        return first;
    }
    p1->next = ptr;
    ptr->prev = p1;
    ptr->next = p2;
    if (p2 != NULL) {
        p2->prev = ptr;  // Corrected line
    }
    return first;
}


struct Node* Delete(struct Node* first, int pos) {
    if (pos == 0) {
        struct Node* ptr = first;
        if (first->next != NULL) {
            first = first->next;
            first->prev = NULL;
        } else {
            first = NULL;
        }
        free(ptr);
```

```
        return first;  // Update to return the new head of the list
    }
    struct Node* p2 = first;
    struct Node* p1 = first;
    int i = 0;
    while (i != pos && p2 != NULL) {
        p1 = p2;
        p2 = p2->next;
        i++;
    }
    if (i != pos) {
        printf("Invalid position\n");
        return first;
    }
    p1->next = p2->next;
    if (p2->next != NULL) {
        p2->next->prev = p1;
    }
    free(p2);
    return first;
}



void display(struct Node* first) {
    if(first==NULL){
        printf("Empty List\n");
    }
    struct Node* p = first;
    while (p != NULL) {
        printf("Element %d\n", p->data);
        p = p->next;
    }
}

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

    while (1) {
        printf("\nOptions:\n");
        printf("1. Insert at a specific position\n");
        printf("2. Delete at specified index\n");
        printf("3. display\n");
        printf("4. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
            head = Insert(head, data, pos);
            break;

        case 2:
            printf("Enter position: ");
            scanf("%d", &pos);
            head = Delete(head,pos);
            break;

        case 3:
            display(head);
            break;
        case 4:
            printf("Exiting the program.\n");
            exit(0);

        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
   }
   return 0;
}
```

```
Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 1
Enter data to insert: 1
Enter position: 0

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 1
Enter data to insert: 2
Enter position: 1

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 1
Enter data to insert: -1
Enter position: 0

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 2
Enter position: 2

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 2
Enter position: 5
Invalid position

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 3
Element -1
Element 1

Options:
1. Insert at a specific position
2. Delete at specified index
3. display
4. Exit
Enter your choice: 4
Exiting the program.

Process returned 0 (0x0)   execution time : 68.237 s
```

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

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

struct Node* enqueue(struct Node* front, struct Node* rear, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

```
    newNode->next = NULL;

    if (front == NULL) {
        // Queue is empty
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }

    return rear;  // Return the updated rear
}

struct Node* dequeue(struct Node* front, struct Node* rear) {
    if (front == NULL && rear == NULL) {
        printf("UNDER-FLOW\n");
        exit(1);
    }

    struct Node* ptr = front;
    if (front == rear) {
        // Only one element in the queue
        front = rear = NULL;
    } else {
        front = ptr->next;
    }
    free(ptr);
    return front;  // Return the updated front
}

void display(struct Node* front, struct Node* rear) {
    if (rear == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    struct Node* ptr = front;
    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}

int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;
    int choice, element;

    do {
        printf("\nQueue Menu:\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 enqueue: ");
                scanf("%d", &element);
                rear = enqueue(front, rear, element);
                if (front == NULL) {
                    // Update front when the queue was empty
                    front = rear;
                }
                break;
            case 2:
                front = dequeue(front, rear);
                break;
            case 3:
                display(front, rear);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);
    while (front != NULL) {
        struct Node* temp = front;
        front = front->next;
        free(temp);
    }

    return 0;
}
```

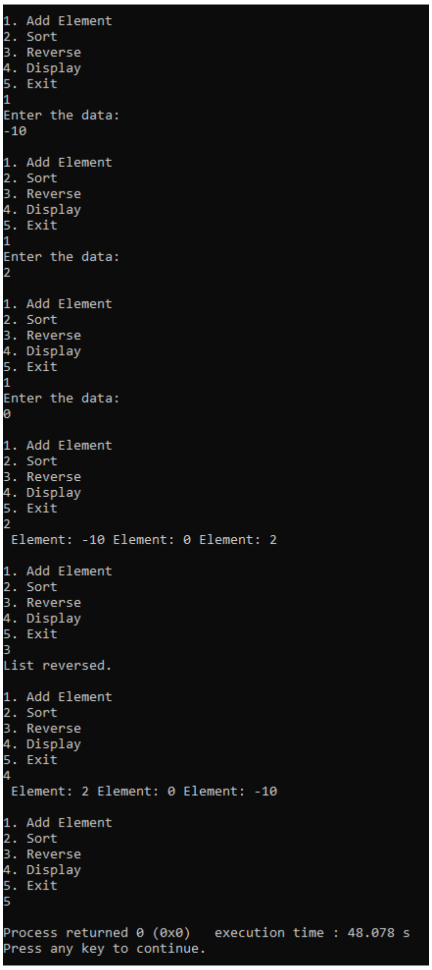
```
Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 12

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 34

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is empty

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue underflow

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting the program.

Process returned 0 (0x0)   execution time : 18.094 s
Press any key to continue.
```

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

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

struct Node* insert(struct Node* first, int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (first == NULL) {
        return newNode;
    }

    struct Node* ptr = first;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }

    ptr->next = newNode;
    return first;
}

void sort(struct Node* first) {
    if (first == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* ptr;
    struct Node* p;

    for (ptr = first; ptr != NULL; ptr = ptr->next) {
        bool swapped = false;
        for (p = first; p->next != NULL; p = p->next) {
            if (p->data > (p->next)->data) {
                int temp = p->data;
                p->data = (p->next)->data;
                (p->next)->data = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }
    struct Node* p1 = first;
    while (p1 != NULL) {
        printf(" Element: %d", p1->data);
        p1 = p1->next;
    }
    printf("\n");

}

void display(struct Node* first){
    struct Node* p1 = first;
```

```
    while (p1 != NULL) {
        printf(" Element: %d", p1->data);
        p1 = p1->next;
    }
    printf("\n");

}

struct Node* reverse(struct Node* first) {
    struct Node* prev = NULL;
    struct Node* current = first;
    struct Node* next = NULL;

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

    return prev;
}

int main() {
    struct Node* first = NULL;
    int n, ele;

    do {
        printf("\n1. Add Element\n2. Sort\n3. Reverse\n4. Display\n5. Exit\n");
        scanf("%d", &n);

        switch (n) {
            case 1:
                printf("Enter the data:\n");
                scanf("%d", &ele);
                first = insert(first, ele);
                break;
            case 2:
                sort(first);
                break;
            case 3:
                first = reverse(first);
                printf("List reversed.\n");
                break;
            case 4:
                display(first);
                break;
            case 5:
                exit(0);
            default:
                printf("Enter correct choice\n");
```

```
        }
    } while (5);

    return 0;
}
```



```
1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
-10

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
2

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
0

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
2
 Element: -10 Element: 0 Element: 2

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
3
List reversed.

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
4
 Element: 2 Element: 0 Element: -10

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
5

Process returned 0 (0x0)   execution time : 48.078 s
Press any key to continue.
```

Program - Leetcode platform

Split Linked List in Parts

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize){
    struct ListNode **ans = (struct ListNode **)calloc(1, sizeof(struct ListNode *) * k);
    struct ListNode *prev;
    int base, len = 0, part = 0;

    for (struct ListNode *tmp = head; tmp; tmp = tmp->next) {
        len++;
    }

    base = len / k;

    for (int i = len % k; i > 0; i--) {
```

```
        ans[part] = head;
        part++;

        for (int i = 0; i < (base + 1); i++) {
            prev = head;
            head = head->next;
        }

        prev->next = NULL;

    }

    if (base) {
        for (int i = part; i < k; i++) {

            ans[part] = head;
            part++;

            for (int j = 0; j < base; j++) {
                prev = head;
                head = head->next;
            }

            prev->next = NULL;
        }
    }

    *returnSize = k;

    return ans;
}
```

Lab program 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 post order
c) To display the elements in the tree.

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

struct Node {
    struct Node* left;
    int data;
```

```
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* ptr = (struct Node*)malloc(sizeof(struct Node));
    ptr->left = ptr->right = NULL;
    ptr->data = data;
    return ptr;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data > root->data) root->right = insert(root->right, data);
    else root->left = insert(root->left, data);
    return root;
}

void preOrder(struct Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preOrder(root->left);
    preOrder(root->right);
}

void inOrder(struct Node* root) {
    if (root == NULL) return;
    inOrder(root->left);
    printf("%d ", root->data);
    inOrder(root->right);
}
void postOrder(struct Node* root) {
    if (root == NULL) return;
    postOrder(root->left);
    postOrder(root->right);
    printf("%d ", root->data);
}

void display(struct Node* root, int level) {
    if (root != NULL) {
        display(root->right, level + 1);
        for (int i = 0; i < level; i++)
            printf("\t");
        printf("%d\n", root->data);
        display(root->left, level + 1);
    }
}
int main() {
    struct Node* root = NULL;

    root = insert(root, 50);
    insert(root, 30);
```

```
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printf("Preorder traversal: ");
    preOrder(root);

    printf("\nInorder traversal: ");
    inOrder(root);

    printf("\nPostorder traversal: ");
    postOrder(root);

    printf("\n\nBinary Tree Structure:\n");
    display(root, 0);

    return 0;
}
```

```
Preorder traversal: 50 30 20 40 70 60 80
Inorder traversal: 20 30 40 50 60 70 80
Postorder traversal: 20 40 30 60 80 70 50

Binary Tree Structure:
                80
        70
                60
50
                40
        30
                20
```

Program - Leetcode platform

Rotate List

```
int GetLength(struct ListNode* head)
{
    if (head == NULL)
        return 0;

    return 1 + GetLength(head->next);
}
struct ListNode* rotateRight(struct ListNode* head, int k){
        if (head == NULL || k == 0)
        return head;
```

```
    int length = GetLength(head);

    if (length == 1)
        return head;
    for(int i=0;i<k%length;i++)
    {
        struct ListNode *p=head;
        while(p->next->next!=NULL)
        {
            p=p->next;
        }
        struct ListNode *a=(struct ListNode *)malloc(sizeof(struct ListNode));
        a->val=p->next->val;
        a->next=head;
        head=a;
        p->next=NULL;
    }
    return head;

}
```

Lab program 9:

Write a program to traverse a graph using BFS method.

```
#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)

{

for(i=1; i<=n; i++)

if(a[v][i] && !visited[i])

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}
```

```
int main()

{

int v;

printf("\n Enter the number of vertices:");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

q[i]=0;

visited[i]=0;

}

printf("Enter graph data in matrix form:\n");

for(i=1; i<=n; i++)

for(j=1; j<=n; j++)

scanf("%d", &a[i][j]);

printf("\n Enter the starting vertex:");

scanf("%d", &v);

bfs(v);

printf("The nodes which are reachable are:\n");

for(i=1; i<=n; i++)

if(visited[i])

printf("%d\t", i);

return 0;

}
```

```
 Enter the number of vertices:6
Enter graph data in matrix form:
0 1 0 0 0 1
1 0 1 1 1 0
1 1 0 0 0 0
0 1 0 0 1 0
0 1 0 1 0 1
1 0 0 0 1 0

 Enter the starting vertex:4
The nodes which are reachable are:
1       2       3       4       5       6

...Program finished with exit code 0
Press ENTER to exit console.
```

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

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

int a[20][20], s[20], n;

void dfs(int v) {
   int i;
   s[v] = 1;

   for (i = 1; i <= n; i++)
      if (a[v][i] && !s[i]) {
         printf("\n %d->%d", v, i);
         dfs(i);
      }
}

int main() {
   int i, j, count = 0;

   printf("\nEnter number of vertices:");
   scanf("%d", &n);

   for (i = 1; i <= n; i++) {
      s[i] = 0;
      for (j = 1; j <= n; j++)
         a[i][j] = 0;
   }

   printf("Enter the adjacency matrix:\n");
   for (i = 1; i <= n; i++)
```

```
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);

    dfs(1);

    printf("\n");

    for (i = 1; i <= n; i++) {
        if (s[i])
            count++;
    }

    if (count == n)
        printf("Graph is connected");
    else
        printf("Graph is not connected");

    return 0;
}
```

```
Enter number of vertices:4
Enter the adjacency matrix:
0 1 1 0
1 0 1 0
1 1 0 0
1 0 1 0

 1->2
 2->3
Graph is not connected

...Program finished with exit code 0
Press ENTER to exit console.
```

HACKER RANK

Complete the *swap Nodes* 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 <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*);
int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count, int* queries, int*
result_rows, int* result_columns) {

}

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

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

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

    for (int i = 0; i < n; i++) {
        *(indexes + i) = malloc(2 * (sizeof(int)));

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

        for (int j = 0; j < 2; j++) {
            int indexes_item = parse_int(*(indexes_item_temp + j));

            *(*(indexes + i) + j) = indexes_item;
        }
    }

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

    int* queries = malloc(queries_count * sizeof(int));

    for (int i = 0; i < queries_count; i++) {
        int queries_item = parse_int(ltrim(rtrim(readline())));

        *(queries + i) = queries_item;
    }
```

```
    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++) {
            fprintf(fptr, "%d", *(*(result + i) + j));

            if (j != result_columns - 1) {
                fprintf(fptr, " ");
            }
        }

        if (i != result_rows - 1) {
            fprintf(fptr, "\n");
        }
    }

    fprintf(fptr, "\n");

    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;
}
```

Lab program 10:

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>

#define MAX_EMPLOYEES 100 // Maximum number of employee records
#define MAX_HASH_SLOTS 50 // Maximum number of hash table slots

// Structure to represent an employee record
struct Employee {
    int key;
    // Add other fields as needed (e.g., name, age, salary)
};

// Structure to represent the hash table entry
struct HashEntry {
    int occupied; // Flag to check if the slot is occupied
    struct Employee employee;
};

// Function to calculate the hash value using remainder method
int hashFunction(int key, int m) {
```

```
    return key % m;
}

// Function to insert an employee record into the hash table
void insert(struct HashEntry hashTable[], int m, struct Employee emp) {
    int key = emp.key;
    int index = hashFunction(key, m);

    // Linear probing to resolve collisions
    while (hashTable[index].occupied) {
        index = (index + 1) % m; // Move to the next slot
    }

    // Insert the employee record into the hash table
    hashTable[index].employee = emp;
    hashTable[index].occupied = 1;
}

// Function to search for an employee record by key
int search(struct HashEntry hashTable[], int m, int key, struct Employee *result) {
    int index = hashFunction(key, m);

    // Linear probing to find the record
    while (hashTable[index].occupied) {
        if (hashTable[index].employee.key == key) {
            *result = hashTable[index].employee;
            return 1; // Employee found
        }
        index = (index + 1) % m; // Move to the next slot
    }

    return 0; // Employee not found
}

int main() {
    // Initialize the hash table
    struct HashEntry hashTable[MAX_HASH_SLOTS] = {0}; // Initialize to zero
    int m = MAX_HASH_SLOTS;

    // Example: Inserting employee records
    struct Employee emp1 = {1234}; // Example record with key 1234
    struct Employee emp2 = {5678}; // Example record with key 5678

    insert(hashTable, m, emp1);
    insert(hashTable, m, emp2);

    // Example: Searching for an employee record
    int searchKey = 5678;
    struct Employee result;

    if (search(hashTable, m, searchKey, &result)) {
```

```
        printf("Employee with key %d found!\n", searchKey);
        // Access other fields in 'result' as needed
    } else {
        printf("Employee with key %d not found!\n", searchKey);
    }

    return 0;
}
```

```
Employee with key 5678 found!
```