**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

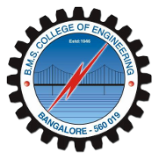
**Chethan T M (1BM23CS076)**

**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 Chethan T M **(1BM23CS076)**, 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.

|  |  |
| --- | --- |
| **Dr. Selva kumar S** | **Dr. Kavitha Sooda** |
| Associate Professor | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Stacks-Push,Pop,Display  Leetcode- Move Zeroes | 4-8 |
| 2 | Infix To Postfix | 9-11 |
| 3 | A) Queue Using Array B) Circular Queue Using Array  Leetcode- First Unique Character in a String | 12-19 |
| 4 | Singly Linked List – Insertion  Leetcode- Backspace String Compare  Leetcode- Remove Digit from Number to Maximize Result | 20-28 |
| 5 | Singly Linked List – Deletion  Leetcode- Remove Digit from Number to Maximize Result  Leetcode- Linked List Cycle  Leetcode- Palindrome Linked List | 29-43 |
| 6 | A) Singly Linked List -Sort,Reverse,Concatenation  B) Singly Linked List -Stacks and Queues | 44-59 |
| 7 | Doubly Linked List - Insertion and Deletion | 60-64 |
| 8 | Binary Search Tree | 65-70 |
| 9 | A) Traverse a Graph using BFS method  B) Graph is Connected or not Using DFS method | 71-76 |
| 10 | Hash Table | 77-82 |

**Lab program 1:**

**Write a program to simulate the working of stack using an array with the following: a) Push**

1. **Pop**
2. **Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

Code:

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_SIZE 5

void insert(int queue[], int \*front, int \*rear) {

    int item;

    if (\*rear == QUEUE\_SIZE - 1) {

        printf("Queue overflow\n");

    } else {

        printf("\nEnter an item: ");

       scanf("%d", &item);

        if (\*front == -1) {

            \*front = 0;

        }

        queue[++(\*rear)] = item;

        printf("%d inserted into the queue\n", item);

    }

}

void delete(int queue[], int \*front, int \*rear) {

    if (\*front == -1 || \*front > \*rear) {

        printf("Queue is empty\n");

    } else {

        printf("\n%d deleted from the queue\n", queue[(\*front)++]);

        if (\*front > \*rear) { // Reset queue if empty

            \*front = \*rear = -1;

        }

    }

}

void display(int queue[], int \*front, int \*rear) {

    if (\*front == -1 || \*front > \*rear) {

        printf("Queue is empty\n");

    } else {

        printf("Queue elements: ");

        for (int i = \*front; i <= \*rear; i++) {

            printf("%d\t", queue[i]);

        }

        printf("\n");

    }

}

int main() {

    int queue[QUEUE\_SIZE], front = -1, rear = -1, choice;

    while (1) {

        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1: insert(queue, &front, &rear); break;

            case 2: delete(queue, &front, &rear); break;

            case 3: display(queue, &front, &rear); break;

            case 4: exit(0);

            default: printf("Invalid choice!!!\n");

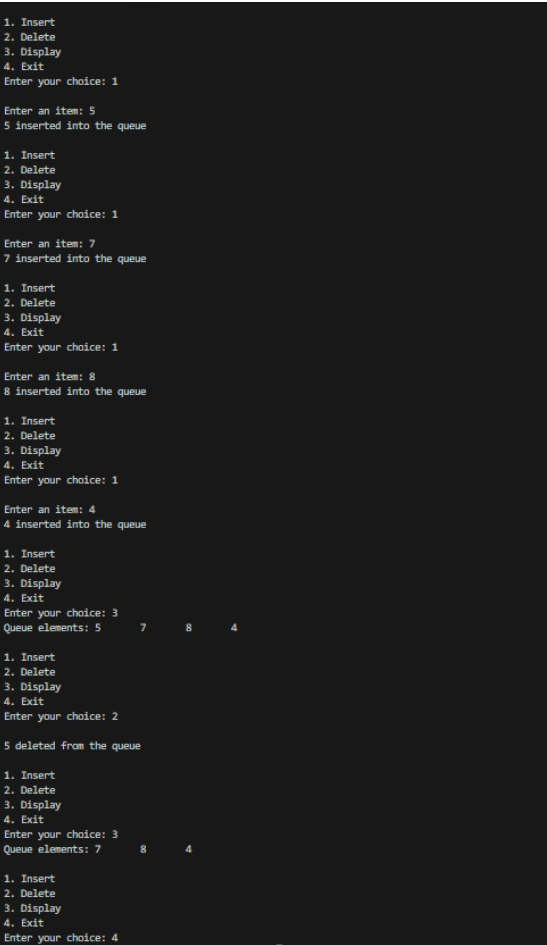
return 0;

        }

    }

}

**Output:**



**Leetcode**

**Move Zeroes**

Code:

#include <stdio.h>

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

int j = 0;

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

if (nums[i] != 0) {

int temp = nums[j];

nums[j] = nums[i];

nums[i] = temp;

j++;

}

}

}

int main() {

int nums[] = {0, 1, 0, 3, 12};

int numsSize = sizeof(nums) / sizeof(nums[0]);

printf("Original array: ");

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

printf("%d ", nums[i]);

}

moveZeroes(nums, numsSize);

printf("\nModified array: ");

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

printf("%d ", nums[i]);

}

return 0;

}

**Output:**

A black and grey striped background

Description automatically generated

**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).**

Code:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define STACK\_SIZE 50

char stack[STACK\_SIZE];

int top = -1;

void push(char c) {

if (top == STACK\_SIZE - 1) {

printf("Stack overflow\n");

exit(1);

}

stack[++top] = c;

}

char pop() {

if (top == -1) {

printf("Stack underflow\n");

exit(1);

}

return stack[top--];

}

int precedence(char op) {

switch (op) {

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

default: return 0;

}

}

void infixToPostfix(char infix[], char postfix[]) {

int i = 0, j = 0;

char c;

while (infix[i] != '\0') {

if (isalnum(infix[i])) { // If operand, add to postfix

postfix[j++] = infix[i];

} else if (infix[i] == '(') { // Push '(' onto stack

push(infix[i]);

} else if (infix[i] == ')') { // Pop until '(' is found

while (top != -1 && (c = pop()) != '(') {

postfix[j++] = c;

}

} else {

while (top != -1 && precedence(stack[top]) >= precedence(infix[i])) {

postfix[j++] = pop();

}

push(infix[i]);

}

i++;

}

while (top != -1) {

postfix[j++] = pop();

}

postfix[j] = '\0'; // Null-terminate the postfix string

}

int main() {

char infix[50], postfix[50];

printf("Enter a valid infix expression: ");

scanf("%s", infix);

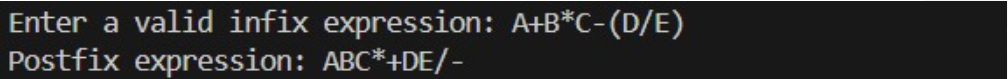
infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

**Output:**



**Lab program: 3**

**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).**

Code:

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_SIZE 5

void insert(int queue[], int \*front, int \*rear) {

    int item;

    if (\*rear == QUEUE\_SIZE - 1) {

        printf("Queue overflow\n");

    } else {

        printf("\nEnter an item: ");

        scanf("%d", &item);

        if (\*front == -1) {

            \*front = 0;

        }

        queue[++(\*rear)] = item;

        printf("%d inserted into the queue\n", item);

    }

}

void delete(int queue[], int \*front, int \*rear) {

    if (\*front == -1 || \*front > \*rear) {

        printf("Queue is empty\n");

    } else {

        printf("\n%d deleted from the queue\n", queue[(\*front)++]);

        if (\*front > \*rear) { // Reset queue if empty

            \*front = \*rear = -1;

        }

    }

}

void display(int queue[], int \*front, int \*rear) {

    if (\*front == -1 || \*front > \*rear) {

        printf("Queue is empty\n");

    } else {

        printf("Queue elements: ");

        for (int i = \*front; i <= \*rear; i++) {

            printf("%d\t", queue[i]);

        }

        printf("\n");

    }

}

int main() {

int queue[QUEUE\_SIZE], front = -1, rear = -1, choice;

while (1) {

printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: insert(queue, &front, &rear); break;

case 2: delete(queue, &front, &rear); break;

case 3: display(queue, &front, &rear); break;

case 4: exit(0);

default: printf("Invalid choice!!!\n");

}

}

return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated

**b) 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.**

Code:

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_SIZE 5

void insert(int queue[], int \*front, int \*rear) {

    int item;

    if ((\*rear + 1) % QUEUE\_SIZE == \*front) {

        printf("Queue overflow\n");

    } else {

        printf("\nEnter an item: ");

        scanf("%d", &item);

        if (\*front == -1) {

            \*front = 0;

        }

        \*rear = (\*rear + 1) % QUEUE\_SIZE;

        queue[\*rear] = item;

        printf("%d inserted into the queue\n", item);

    }

}

void delete(int queue[], int \*front, int \*rear) {

    if (\*front == -1) {

        printf("Queue is empty\n");

    } else {

        printf("\n%d deleted from the queue\n", queue[\*front]);

        if (\*front == \*rear) {

            \*front = \*rear = -1; // Reset queue if empty

        } else {

            \*front = (\*front + 1) % QUEUE\_SIZE;

        }

    }

}

void display(int queue[], int \*front, int \*rear) {

    if (\*front == -1) {

        printf("Queue is empty\n");

    } else {

        printf("Queue elements: ");

        int i = \*front;

        while (1) {

            printf("%d\t", queue[i]);

            if (i == \*rear) {

                break;

            }

            i = (i + 1) % QUEUE\_SIZE;

        }

        printf("\n");

    }

}

int main() {

int queue[QUEUE\_SIZE], front = -1, rear = -1, choice;

while (1) {

printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: insert(queue, &front, &rear); break;

case 2: delete(queue, &front, &rear); break;

case 3: display(queue, &front, &rear); break;

case 4: exit(0); break;

default: printf("Invalid choice!!!\n"); break;

}

}

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Leetcode**

**First Unique Character in a String**

Code:

#include <stdio.h>

#include <string.h>

int firstUniqChar(char\* s) {

int count[26] = {0};

int n = strlen(s);

int queue[n], front = 0, rear = -1;

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

count[s[i] - 'a']++;

queue[++rear] = i;

}

while (front <= rear) {

int index = queue[front++];

if (count[s[index] - 'a'] == 1) {

return index;

}

}

return -1;

}

int main() {

char s[] = "leetcode";

int result = firstUniqChar(s);

printf("%d\n", result);

return 0;

}

**Output:**

A black and grey striped background

Description automatically generated

**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.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

newNode->next = \*head;

\*head = newNode;

printf("%d inserted at the beginning\n", value);

}

void insertAtEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

printf("%d inserted at the end\n", value);

}

void insertAtPosition(struct Node\*\* head, int value, int position) {

struct Node\* newNode = createNode(value);

if (position == 1) {

newNode->next = \*head;

\*head = newNode;

printf("%d inserted at position %d\n", value, position);

return;

}

struct Node\* temp = \*head;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position\n");

free(newNode);

} else {

newNode->next = temp->next;

temp->next = newNode;

printf("%d inserted at position %d\n", value, position);

}

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("Linked list is empty\n");

} else {

printf("Linked list contents: ");

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

}

int main() {

struct Node\* head = NULL;

int choice, value, position;

while (1) {

printf("\n1. Insert at beginning\n2. Insert at end\n3. Insert at position\n4. Display list\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

insertAtBeginning(&head, value);

break;

case 2:

printf("Enter value to insert: ");

scanf("%d", &value);

insertAtEnd(&head, value);

break;

case 3:

printf("Enter value to insert: ");

scanf("%d", &value);

printf("Enter position: ");

scanf("%d", &position);

insertAtPosition(&head, value, position);

break;

case 4:

displayList(head);

break;

case 5:

return 0;

default:

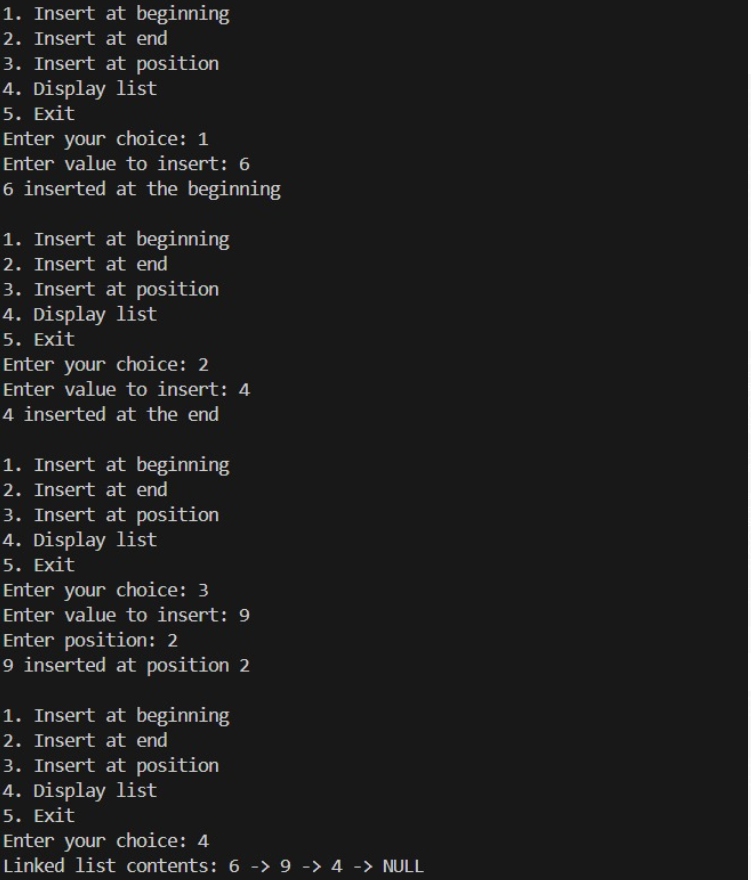
printf("Invalid choice!!!\n");

}

}

}

**Output:**



**Leetcode**

**Backspace String Compare**

Code:

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

bool backspaceCompare(char\* s, char\* t) {

int i = strlen(s) - 1, j = strlen(t) - 1;

while (i >= 0 || j >= 0) {

int skipS = 0, skipT = 0;

while (i >= 0 && (s[i] == '#' || skipS > 0)) {

if (s[i] == '#') skipS++;

else skipS--;

i--;

}

while (j >= 0 && (t[j] == '#' || skipT > 0)) {

if (t[j] == '#') skipT++;

else skipT--;

j--;

}

if (i >= 0 && j >= 0 && s[i] != t[j]) return false;

if ((i >= 0) != (j >= 0)) return false;

i--;

j--;

}

return true;

}

int main() {

char s1[] = "ab#c";

char t1[] = "ad#c";

printf("Compare '%s' and '%s': %s\n", s1, t1, backspaceCompare(s1, t1) ? "true" : "false");

char s2[] = "ab##";

char t2[] = "c#d#";

printf("Compare '%s' and '%s': %s\n", s2, t2, backspaceCompare(s2, t2) ? "true" : "false");

char s3[] = "a#c";

char t3[] = "b";

printf("Compare '%s' and '%s': %s\n", s3, t3, backspaceCompare(s3, t3) ? "true" : "false");

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Leetcode**

**Remove Digit from Number to Maximize Result**

Code:

#include <stdio.h>

#include <string.h>

char\* removeDigit(char\* number, char digit) {

int n = strlen(number);

for (int i = 0; i < n - 1; i++) {

if (number[i] == digit && number[i] < number[i + 1]) {

memmove(&number[i], &number[i + 1], n - i);

return number;

}

}

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

if (number[i] == digit) {

memmove(&number[i], &number[i + 1], n - i);

break;

}

}

return number;

}

int main() {

char number1[] = "1231";

char digit1 = '1';

printf("Result after removing '%c' from \"%s\": %s\n", digit1, number1, removeDigit(number1, digit1));

char number2[] = "551";

char digit2 = '5';

printf("Result after removing '%c' from \"%s\": %s\n", digit2, number2, removeDigit(number2, digit2));

char number3[] = "7654321";

char digit3 = '6';

printf("Result after removing '%c' from \"%s\": %s\n", digit3, number3, removeDigit(number3, digit3));

return 0;

}

**Output:**

A black and grey striped background

Description automatically generated

**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.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* createNode(int value) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->data = value;

    newNode->next = NULL;

    return newNode;

}

void insertAtEnd(struct Node\*\* head, int value) {

    struct Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        struct Node\* temp = \*head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    printf("%d inserted into the list\n", value);

}

void deleteFirst(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty\n");

    } else {

        struct Node\* temp = \*head;

        \*head = (\*head)->next;

        printf("%d deleted from the list (first element)\n", temp->data);

        free(temp);

    }

}

void deleteLast(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("List is empty\n");

    } else if ((\*head)->next == NULL) { // Only one element

        printf("%d deleted from the list (last element)\n", (\*head)->data);

        free(\*head);

        \*head = NULL;

    } else {

        struct Node\* temp = \*head;

        while (temp->next->next != NULL) {

            temp = temp->next;

        }

        printf("%d deleted from the list (last element)\n", temp->next->data);

        free(temp->next);

        temp->next = NULL;

    }

}

void deleteElement(struct Node\*\* head, int value) {

    if (\*head == NULL) {

        printf("List is empty\n");

    } else if ((\*head)->data == value) {

        deleteFirst(head);

    } else {

        struct Node\* temp = \*head;

        struct Node\* prev = NULL;

        while (temp != NULL && temp->data != value) {

            prev = temp;

            temp = temp->next;

        }

        if (temp == NULL) {

            printf("%d not found in the list\n", value);

        } else {

            prev->next = temp->next;

            printf("%d deleted from the list (specified element)\n", temp->data);

            free(temp);

        }

    }

}

void displayList(struct Node\* head) {

    if (head == NULL) {

        printf("Linked list is empty\n");

    } else {

        printf("Linked list contents: ");

        while (head != NULL) {

            printf("%d -> ", head->data);

            head = head->next;

        }

        printf("NULL\n");

    }

}

int main() {

    struct Node\* head = NULL;

    int choice, value;

    while (1) {

        printf("\n1. Create (Insert at end)\n2. Delete first element\n3. Delete last element\n4. Delete specified element\n5. Display list\n6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                insertAtEnd(&head, value);

                break;

            case 2:

                deleteFirst(&head);

                break;

            case 3:

                deleteLast(&head);

                break;

            case 4:

                printf("Enter value to delete: ");

                scanf("%d", &value);

                deleteElement(&head, value);

                break;

            case 5:

                displayList(head);

                break;

            case 6:

                exit(0);

            default:

                printf("Invalid choice!!!\n");

        }

    }

return 0;

}

**Output:**



A screenshot of a computer program

Description automatically generated

**Leetcode**

**Remove Digit from Number to Maximize Result**

Code:

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode\* next;

};

struct ListNode\* deleteDuplicates(struct ListNode\* head) {

struct ListNode\* current = head;

while (current != NULL && current->next != NULL) {

if (current->val == current->next->val) {

struct ListNode\* temp = current->next;

current->next = current->next->next;

free(temp);

} else {

current = current->next;

}

}

return head;

}

struct ListNode\* createNode(int value) {

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = value;

newNode->next = NULL;

return newNode;

}

void printList(struct ListNode\* head) {

while (head != NULL) {

printf("%d ", head->val);

head = head->next;

}

printf("\n");

}

void freeList(struct ListNode\* head) {

while (head != NULL) {

struct ListNode\* temp = head;

head = head->next;

free(temp);

}

}

int main() {

struct ListNode\* head = createNode(1);

head->next = createNode(1);

head->next->next = createNode(2);

head->next->next->next = createNode(3);

head->next->next->next->next = createNode(3);

printList(head);

head = deleteDuplicates(head);

printList(head);

freeList(head);

return 0;

}

**Output:**

A screenshot of a video game

Description automatically generated

**Leetcode**

**Linked List Cycle**

Code:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct ListNode {

int val;

struct ListNode\* next;

};

bool hasCycle(struct ListNode\* head) {

if (head == NULL) return false;

struct ListNode \*slow = head, \*fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}

struct ListNode\* createNode(int value) {

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = value;

newNode->next = NULL;

return newNode;

}

void freeList(struct ListNode\* head) {

struct ListNode\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

int main() {

struct ListNode\* head = createNode(3);

head->next = createNode(2);

head->next->next = createNode(0);

head->next->next->next = createNode(-4);

head->next->next->next->next = head->next;

printf("%s\n", hasCycle(head) ? "true" : "false");

freeList(head);

return 0;

}

**Output:**

A screenshot of a test results

Description automatically generated

**Leetcode**

**Palindrome Linked List**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct ListNode {

int val;

struct ListNode\* next;

};

bool isPalindrome(struct ListNode\* head) {

if (head == NULL || head->next == NULL) return true;

struct ListNode \*slow = head, \*fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

struct ListNode\* second\_half = slow;

struct ListNode\* prev = NULL;

while (second\_half != NULL) {

struct ListNode\* next = second\_half->next;

second\_half->next = prev;

prev = second\_half;

second\_half = next;

}

struct ListNode\* first\_half = head;

second\_half = prev;

while (second\_half != NULL) {

if (first\_half->val != second\_half->val) return false;

first\_half = first\_half->next;

second\_half = second\_half->next;

}

return true;

}

struct ListNode\* createNode(int value) {

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = value;

newNode->next = NULL;

return newNode;

}

void freeList(struct ListNode\* head) {

while (head != NULL) {

struct ListNode\* temp = head;

head = head->next;

free(temp);

}

}

int main() {

struct ListNode\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(2);

head->next->next->next = createNode(1);

printf("%s\n", isPalindrome(head) ? "true" : "false");

freeList(head);

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Program 6**

1. **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 value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->next = NULL;

return newNode;

}

void insertAtEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

printf("%d inserted into the list\n", value);

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("Linked list is empty\n");

} else {

printf("Linked list contents: ");

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

}

void sortList(struct Node\*\* head) {

if (\*head == NULL || (\*head)->next == NULL) {

printf("List is already sorted or empty\n");

return;

}

struct Node \*i, \*j;

int temp;

for (i = \*head; i != NULL; i = i->next) {

for (j = i->next; j != NULL; j = j->next) {

if (i->data > j->data) {

temp = i->data;

i->data = j->data;

j->data = temp;

}

}

}

printf("List sorted\n");

}

void reverseList(struct Node\*\* head) {

struct Node \*prev = NULL, \*current = \*head, \*next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

printf("List reversed\n");

}

void concatenateLists(struct Node\*\* head1, struct Node\*\* head2) {

if (\*head1 == NULL) {

\*head1 = \*head2;

} else {

struct Node\* temp = \*head1;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = \*head2;

}

\*head2 = NULL;

printf("Lists concatenated\n");

}

int main() {

struct Node \*list1 = NULL, \*list2 = NULL;

int choice, value;

while (1) {

printf("\n1. Insert into List 1\n2. Insert into List 2\n3. Sort List 1\n4. Reverse List 1\n5. Concatenate Lists\n6. Display List 1\n7. Display List 2\n8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert into List 1: ");

scanf("%d", &value);

insertAtEnd(&list1, value);

break;

case 2:

printf("Enter value to insert into List 2: ");

scanf("%d", &value);

insertAtEnd(&list2, value);

break;

case 3:

sortList(&list1);

break;

case 4:

reverseList(&list1);

break;

case 5:

concatenateLists(&list1, &list2);

break;

case 6:

displayList(list1);

break;

case 7:

displayList(list2);

break;

case 8:

exit(0);

default:

printf("Invalid choice!!!\n");

}

}

return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**b) WAP to Implement Single Link List to simulate Stack & Queue Operations.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* createNode(int value) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->data = value;

    newNode->next = NULL;

    return newNode;

}

void push(struct Node\*\* top, int value) {

    struct Node\* newNode = createNode(value);

    newNode->next = \*top;

    \*top = newNode;

    printf("%d pushed onto the stack\n", value);

}

void pop(struct Node\*\* top) {

    if (\*top == NULL) {

        printf("Stack underflow\n");

    } else {

        struct Node\* temp = \*top;

        \*top = (\*top)->next;

        printf("%d popped from the stack\n", temp->data);

        free(temp);

    }

}

void enqueue(struct Node\*\* front, struct Node\*\* rear, int value) {

    struct Node\* newNode = createNode(value);

    if (\*rear == NULL) {

        \*front = \*rear = newNode;

    } else {

        (\*rear)->next = newNode;

        \*rear = newNode;

    }

    printf("%d enqueued into the queue\n", value);

}

void dequeue(struct Node\*\* front, struct Node\*\* rear) {

    if (\*front == NULL) {

        printf("Queue underflow\n");

    } else {

        struct Node\* temp = \*front;

        \*front = (\*front)->next;

        if (\*front == NULL) {

            \*rear = NULL;

        }

        printf("%d dequeued from the queue\n", temp->data);

        free(temp);

    }

}

void displayStack(struct Node\* top) {

    if (top == NULL) {

        printf("Stack is empty\n");

    } else {

        printf("Stack contents: ");

        while (top != NULL) {

            printf("%d -> ", top->data);

            top = top->next;

        }

        printf("NULL\n");

    }

}

void displayQueue(struct Node\* front) {

    if (front == NULL) {

        printf("Queue is empty\n");

    } else {

        printf("Queue contents: ");

        while (front != NULL) {

            printf("%d -> ", front->data);

            front = front->next;

        }

        printf("NULL\n");

    }

}

int main() {

    struct Node\* stackTop = NULL;

    struct Node \*queueFront = NULL, \*queueRear = NULL;

    int choice, value;

    while (1) {

        printf("\n1. Push (Stack)\n2. Pop (Stack)\n3. Display Stack\n4. Enqueue (Queue)\n5. Dequeue (Queue)\n6. Display Queue\n7. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to push onto the stack: ");

                scanf("%d", &value);

                push(&stackTop, value);

                break;

            case 2:

                pop(&stackTop);

                break;

            case 3:

                displayStack(stackTop);

                break;

            case 4:

                printf("Enter value to enqueue into the queue: ");

                scanf("%d", &value);

                enqueue(&queueFront, &queueRear, value);

                break;

            case 5:

                dequeue(&queueFront, &queueRear);

                break;

            case 6:

                displayQueue(queueFront);

                break;

            case 7:

                exit(0);

            default:

                printf("Invalid choice!!!\n");

        }

    }

return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Program 7**

**WAP to Implement doubly link list with primitive operations**

**a) Create a doubly linked list.**

**b) Insert a new node at the beginning.**

**c) Insert the node based on a specific location.**

**d) Insert a new node at the end.**

**e) Display the contents of the list.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

};

struct Node\* createNode(int value) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->data = value;

    newNode->next = newNode->prev = NULL;

    return newNode;

}

void insertAtBeginning(struct Node\*\* head, int value) {

    struct Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        newNode->next = \*head;

        (\*head)->prev = newNode;

        \*head = newNode;

    }

    printf("%d inserted at the beginning\n", value);

}

void insertAtPosition(struct Node\*\* head, int value, int position) {

    struct Node\* newNode = createNode(value);

    if (position == 1) {

        insertAtBeginning(head, value);

        return;

    }

    struct Node\* temp = \*head;

    for (int i = 1; temp != NULL && i < position - 1; i++) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Position out of range\n");

    } else {

        newNode->next = temp->next;

        if (temp->next != NULL) {

            temp->next->prev = newNode;

        }

        temp->next = newNode;

        newNode->prev = temp;

        printf("%d inserted at position %d\n", value, position);

    }

}

void insertAtEnd(struct Node\*\* head, int value) {

    struct Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

    } else {

        struct Node\* temp = \*head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

    printf("%d inserted at the end\n", value);

}

void displayList(struct Node\* head) {

    if (head == NULL) {

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

        return;

    }

    struct Node\* temp = head;

    printf("Doubly Linked List: ");

    while (temp != NULL) {

        printf("%d <-> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

int main() {

    struct Node\* head = NULL;

    int choice, value, position;

    while (1) {

        printf("\n1. Insert at Beginning\n2. Insert at Position\n3. Insert at End\n4. Display List\n5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert at the beginning: ");

                scanf("%d", &value);

                insertAtBeginning(&head, value);

                break;

            case 2:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                printf("Enter position: ");

                scanf("%d", &position);

                insertAtPosition(&head, value, position);

                break;

            case 3:

                printf("Enter value to insert at the end: ");

                scanf("%d", &value);

                insertAtEnd(&head, value);

                break;

            case 4:

                displayList(head);

                break;

            case 5:

                exit(0);

            default:

                printf("Invalid choice!!!\n");

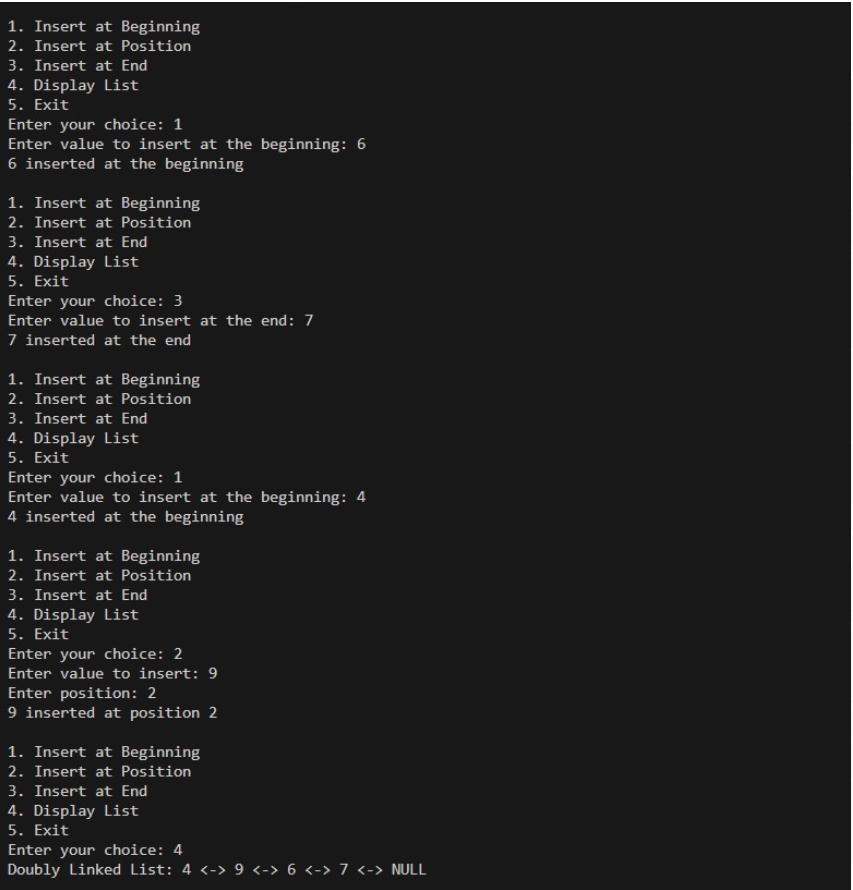
        }

    }

return 0;

}

**Output:**



**Program 8**

**Write a program**

**a) To construct a binary Search tree.**

**b) To traverse the tree using all the methods i.e., inorder, preorder and post order c) To display the elements in the tree.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

struct Node\* insert(struct Node\* root, int value) {

    if (root == NULL) {

        return createNode(value);

    }

    if (value < root->data) {

        root->left = insert(root->left, value);

    } else if (value > root->data) {

        root->right = insert(root->right, value);

    }

    return root;

}

void inorderTraversal(struct Node\* root) {

    if (root != NULL) {

        inorderTraversal(root->left);

        printf("%d ", root->data);

        inorderTraversal(root->right);

    }

}

void preorderTraversal(struct Node\* root) {

    if (root != NULL) {

        printf("%d ", root->data);

        preorderTraversal(root->left);

        preorderTraversal(root->right);

    }

}

void postorderTraversal(struct Node\* root) {

    if (root != NULL) {

        postorderTraversal(root->left);

        postorderTraversal(root->right);

        printf("%d ", root->data);

    }

}

int main() {

    struct Node\* root = NULL;

    int choice, value;

    while (1) {

        printf("\n1. Insert node into BST\n");

        printf("2. In-order Traversal\n");

        printf("3. Pre-order Traversal\n");

        printf("4. Post-order Traversal\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert into BST: ");

                scanf("%d", &value);

                root = insert(root, value);

                printf("%d inserted into the BST\n", value);

                break;

            case 2:

                printf("In-order traversal: ");

                inorderTraversal(root);

                printf("\n");

                break;

            case 3:

                printf("Pre-order traversal: ");

                preorderTraversal(root);

                printf("\n");

                break;

            case 4:

                printf("Post-order traversal: ");

                postorderTraversal(root);

                printf("\n");

                break;

            case 5:

                exit(0);

            default:

                printf("Invalid choice!!!\n");

        }

    }

return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Program 9**

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

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 20

struct Queue {

    int items[MAX];

    int front, rear;

};

void initQueue(struct Queue\* q) {

    q->front = -1;

    q->rear = -1;

}

int isEmpty(struct Queue\* q) {

    return q->front == -1;

}

void enqueue(struct Queue\* q, int value) {

    if (q->rear == MAX - 1) {

        printf("Queue overflow\n");

    } else {

        if (q->front == -1) {

            q->front = 0;

        }

        q->rear++;

        q->items[q->rear] = value;

    }

}

int dequeue(struct Queue\* q) {

    if (isEmpty(q)) {

        printf("Queue underflow\n");

        return -1;

    } else {

        int value = q->items[q->front];

        if (q->front == q->rear) {

            q->front = q->rear = -1;

        } else {

            q->front++;

        }

        return value;

    }

}

void bfs(int graph[MAX][MAX], int startVertex, int n) {

    struct Queue q;

    initQueue(&q);

    int visited[MAX] = {0};

    visited[startVertex] = 1;

    enqueue(&q, startVertex);

    printf("BFS traversal starting from vertex %d: ", startVertex);

    while (!isEmpty(&q)) {

        int currentVertex = dequeue(&q);

        printf("%d ", currentVertex);

        for (int i = 0; i < n; i++) {

            if (graph[currentVertex][i] == 1 && !visited[i]) {

                enqueue(&q, i);

                visited[i] = 1;

            }

        }

    }

    printf("\n");

}

int main() {

    int graph[MAX][MAX], n, startVertex;

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

    scanf("%d", &n);

    printf("Enter the adjacency matrix (0 for no edge, 1 for edge):\n");

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

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

        }

    }

    printf("Enter the starting vertex (0 to %d): ", n - 1);

    scanf("%d", &startVertex);

    bfs(graph, startVertex, n);

return 0;

}

**Output:**

A computer screen with white text

Description automatically generated

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

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 20

void dfs(int graph[MAX][MAX], int visited[MAX], int vertex, int n) {

    // Mark the current vertex as visited

    visited[vertex] = 1;

    printf("%d ", vertex);

    for (int i = 0; i < n; i++) {

        if (graph[vertex][i] == 1 && !visited[i]) {

            dfs(graph, visited, i, n);

        }

    }

}

int main() {

    int graph[MAX][MAX], visited[MAX], n, startVertex;

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

    scanf("%d", &n);

    printf("Enter the adjacency matrix (0 for no edge, 1 for edge):\n");

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

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

        }

    }

    for (int i = 0; i < n; i++) {

        visited[i] = 0;

    }

    printf("Enter the starting vertex (0 to %d): ", n - 1);

    scanf("%d", &startVertex);

    printf("DFS traversal starting from vertex %d: ", startVertex);

    dfs(graph, visited, startVertex, n);

    printf("\n");

return 0;

}

**Output:**

A black screen with white text

Description automatically generated

**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 -&gt; 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.**

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_RECORDS 100

#define MAX\_KEYS 10

struct Employee {

int key;

char name[50];

};

int linearProbing(int hashTable[], int m, int index) {

int i = index;

while (hashTable[i] != -1) {

i = (i + 1) % m; // Linear probing

if (i == index) {

return -1; // No space available

}

}

return i;

}

int hashFunction(int key, int m) {

return key % m;

}

void insert(int hashTable[], struct Employee employees[], int key, int m, int \*size) {

int index = hashFunction(key, m); // Get the index from the hash function

if (hashTable[index] == -1) {

hashTable[index] = key;

employees[index].key = key;

printf("Enter employee name: ");

scanf("%s", employees[index].name);

(\*size)++;

printf("Employee with key %d inserted at index %d.\n", key, index);

} else {

int newIndex = linearProbing(hashTable, m, index);

if (newIndex != -1) {

hashTable[newIndex] = key;

employees[newIndex].key = key;

printf("Enter employee name: ");

scanf("%s", employees[newIndex].name);

(\*size)++;

printf("Employee with key %d inserted at index %d.\n", key, newIndex);

} else {

printf("No available space to insert the record.\n");

}

}

}

int search(int hashTable[], struct Employee employees[], int key, int m) {

int index = hashFunction(key, m); // Get the index from the hash function

if (hashTable[index] == key) {

return index; // Found at the index

} else {

int i = (index + 1) % m;

while (i != index) {

if (hashTable[i] == key) {

return i; // Found at the index

}

i = (i + 1) % m;

}

}

return -1; // Key not found

}

// Function to display all employee records in the hash table

void display(int hashTable[], struct Employee employees[], int m) {

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

if (hashTable[i] != -1) {

printf("Employee Key: %d, Name: %s at index %d\n", employees[i].key, employees[i].name, i);

}

}

}

int main() {

int m, size = 0;

printf("Enter the size of the hash table (m): ");

scanf("%d", &m);

int hashTable[m]; // Hash table to store employee keys

struct Employee employees[m]; // Array of employee records

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

hashTable[i] = -1;

}

while (1) {

int choice, key;

printf("\n1. Insert Employee\n2. Search Employee\n3. Display Employees\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter employee key (4-digit): ");

scanf("%d", &key);

insert(hashTable, employees, key, m, &size);

break;

case 2:

printf("Enter employee key to search: ");

scanf("%d", &key);

int index = search(hashTable, employees, key, m);

if (index != -1) {

printf("Employee with key %d found at index %d. Name: %s\n", key, index, employees[index].name);

} else {

printf("Employee with key %d not found.\n", key);

}

break;

case 3:

display(hashTable, employees, m);

break;

case 4:

exit(0);

default:

printf("Invalid choice!\n");

}

}

return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated