ಡಾ|| ಅಂಬೇಡ್ಕರ್ ತಾಂತ್ರಿಕ ಮಹಾವಿದ್ಯಾಲಯ

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY

Outer Ring Road, Malathahalli, Bengaluru-560056, Karnataka, India

DEPARTMENT OF
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

"DATA STRUCTURES "
[SUBJECT CODE: 22AIL305]

LAB MANUAL

PREPARED BY:

Mrs.RAJESHWARI
ASST.PROFESSOR
DEPT.OF AI&ML

SEMESTER: III

Academic year 2025-2026

Dr Ambedkar Institute of Technology, Bengaluru-56

Department Of AI & ML

Scheme and Syllabus - CBCS-2023 -2024

Course Title	**Data Structures and Application**						
Course Code	**22AIL305**						
Category	Professional Core Course Laboratory (PCCL)						
Scheme and Credits	No. of Hours/Week					Total teaching hours	Credits
	L	T	P	SS	Total		
	00	**00**	**02**	**00**	**03**	**20**	**01**
CIE Marks: 50	**SEE Marks: 50**		**Total Max. marks=100**		**Duration of SEE: 03 Hours**		

Course Objectives:

1. Explain the fundamentals of data structures and their applications essential for implementing solutions to problems..
2. Illustrate representation of data structures: Stack, Queues, Linked Lists, Trees and Graphs.
3. Design and Develop Solutions to problems using Arrays, Structures, Stack, Queues, Linked Lists..
4. Explore usage of Trees and Graph for application development..
5. Apply the Hashing techniques in mapping key value pairs.

1. Design, Develop and Implement a menu driven Program in C for the following Array Operations
a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Exit.
Support the program with functions for each of the above operations.
2. Design, Develop and Implement a menu driven Program in C for the following Array operations
a. Inserting an Element (ELEM) at a given valid Position (POS)
b. Deleting an Element at a given valid Position POS)
c. Display of Array Elements
d. Exit.
Support the program with functions for each of the above operations.
https://ds2-iiith.vlabs.ac.in/exp/selection-sort/index.html

https://ds1-iiith.vlabs.ac.in/data-structures-1/List%20of%20experiments.html

3. Design, Develop and Implement a menu driven Program in C for the following operations on STA Integers (Array Implementation of Stack with maximum size MAX)

b. Pop an Element from Stack
c. Demonstrate Overflow and Underflow situations on Stack
d. Display the status of Stack
e. Exit
Support the program with appropriate functions for each of the above operations
4. Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks
https://ds1-iiith.vlabs.ac.in/exp/stacks-queues/index.html

5. Singly Linked List (SLL) of Integer Data
a. Create a SLL stack of N integer.
b. Display of SLL
c. Linear search. Create a SLL queue of N Students Data Concatenation of two SLL of integers.
6. Design, Develop and Implement a menu driven Program in C for the following operationson Doubly Linked List (DLL) of Professor Data with the fields: ID, Name, Branch, Area of specialization
a. Create a DLL stack of N Professor's Data.
b. Create a DLL queue of N Professor's Data
Display the status of DLL and count the number of nodes in it.
https://ds1-iiith.vlabs.ac.in/exp/linked-list/basics/overview.html
https://ds1-iiith.vlabs.ac.in/List%20of%20experiments.html
https://ds1-iiith.vlabs.ac.in/exp/linked-list/basics/overview.html

7. Given an array of elements, construct a complete binary tree from this array in level order fashion. T
is, elements from left in the array will be filled in the tree level wise starting from level 0. Ex: Input :
arr[] = {1, 2, 3, 4, 5, 6}
Output : Root of the following tree
1
/ \
2 3
/ \ /\
4 5 6
8. Design, Develop and Implement a menu driven Program in C for the following operations on Binar
Search Tree (BST) of Integers
a. Create a BST of N Integers
b. Traverse the BST in Inorder, Preorder and Post Order
https://ds1-iiith.vlabs.ac.in/exp/tree-traversal/index.html

https://ds1-iiith.vlabs.ac.in/exp/tree-traversal/depth-first-traversal/dft-practice.html

9. Design, Develop and implement a program in C for the following operations on Graph (G) of cities
a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a diagraph using DFS/BFS method.
10. Design and develop a program in C that uses Hash Function H:K->L as H(K)=K mod m(reminder method) and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

TEACHING LEARNING PROCESS: Chalk and Talk, power point presentation, animations, vide

COURSE OUTCOMES: On completion of the course, student should be able to:

CO 1. Identify different data structures and their applications.

CO 2. Apply stack and queues in solving problems.

CO 3. Demonstrate applications of linked list.

CO 4. Explore the applications of trees and graphs to model and solve the real-world problem.

CO 5. Make use of Hashing techniques and resolve collisions during mapping of key value pairs

SCHEME FOR EXAMINATIONS:

The theory part shall be evaluated both by CIE and SEE. The practical part shall be evaluated by only CIE (noSEE).

MAPPING of COs with POs and PSOs

	PO1	**PO**	**PO**	**PO4**	**PO5**	**PO6**	**PO**	**PO**	**PO9**	**PO10**	**PO1**	**PO1**	
CO			3	3			3		3	3			
CO		3	3	3					3			3	
CO		3	3	3					3			3	
CO		3	3	3					3			3	
CO		3	3	3					3			3	
Strength of correlation: Low-1, Medium- 2, High-3													

1. **Design, Develop and Implement a menu driven Program in C for the following Array Operations**

a. **Creating an Array of N Integer Elements**

b. **Display of Array Elements with Suitable Headings**

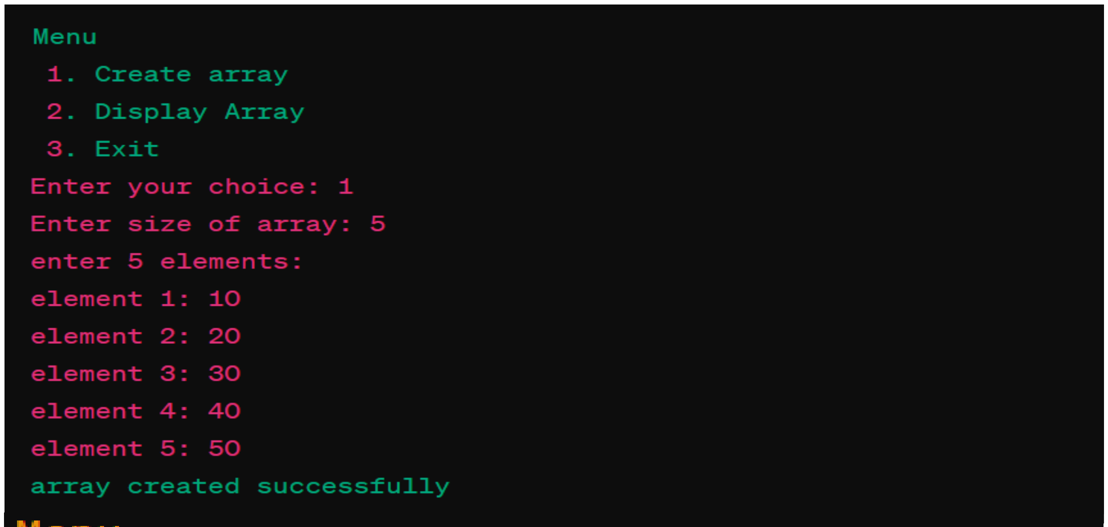
c. **Exit.**

Support the program with functions for each of the above operations.

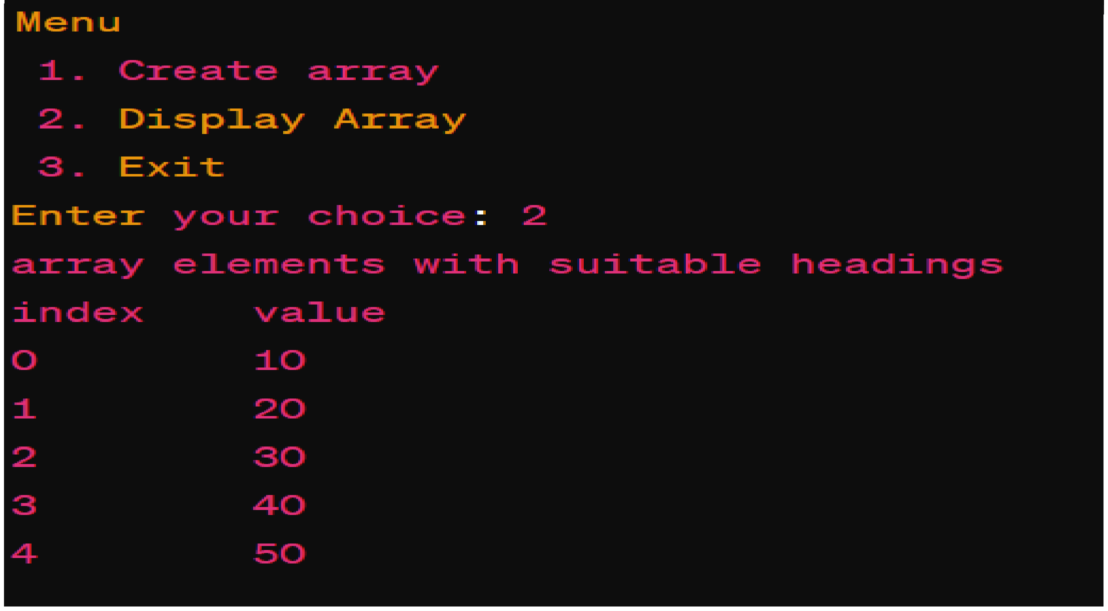
```
#include<stdio.h>


void createArray(int N,int array[])
{
    printf("enter %d elements:",N);
    for(int i=0;i<N;i++)
    { printf("element %d:",i+1);
      scanf("%d",&array[i]);
    }
  printf("array created successfully");
}

void displayArray(int N,int array[])
{  if(N==0)
   { printf("array is empty create an array first \n");
   }
   else
   { printf("array elements with suitable headings \n");
     printf("index \t value \n");
     for(int i=0;i<N;++i)
     {printf("%d \t %d \n",i,array[i]);}
   }
}

int main()
{int choice,N;
int array[25];
do{
    printf("\n Menu \n 1.Create array \n 2.Display Array \n 3.Exit\n");
    printf("Enter your choice");
    scanf("%d",&choice);

switch(choice)
  {
      case 1:
      printf("Enter size of array");
      scanf("%d",&N);
      createArray(N,array);
```

```
        break;
        case 2:
        displayArray(N,array);
        break;
        case 3:
        printf("Exiting the program");
        return 0;
        default:
        printf("Invalid choice");

    }
  }
  while(choice!=3);
  }
```

OUTPUT:



```
Menu
 1. Create array
 2. Display Array
 3. Exit
Enter your choice: 1
Enter size of array: 5
enter 5 elements:
element 1: 10
element 2: 20
element 3: 30
element 4: 40
element 5: 50
array created successfully
```



```
Menu
 1. Create array
 2. Display Array
 3. Exit
Enter your choice: 2
array elements with suitable headings
index     value
0         10
1         20
2         30
3         40
4         50
```

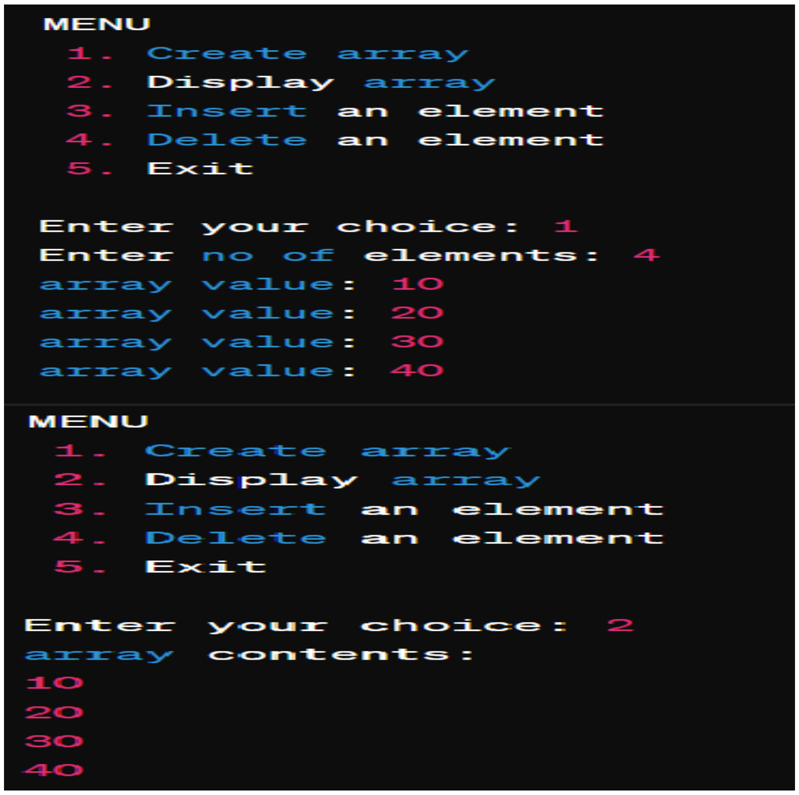
2. **Design, Develop and Implement a menu driven Program in C for the following Array operations**
   a. **Inserting an Element (ELEM) at a given valid Position (POS)**
   b. **Deleting an Element at a given valid Position POS)**
   c. **Display of Array Elements**
   d. **Exit.**

Support the program with functions for each of the above operations.

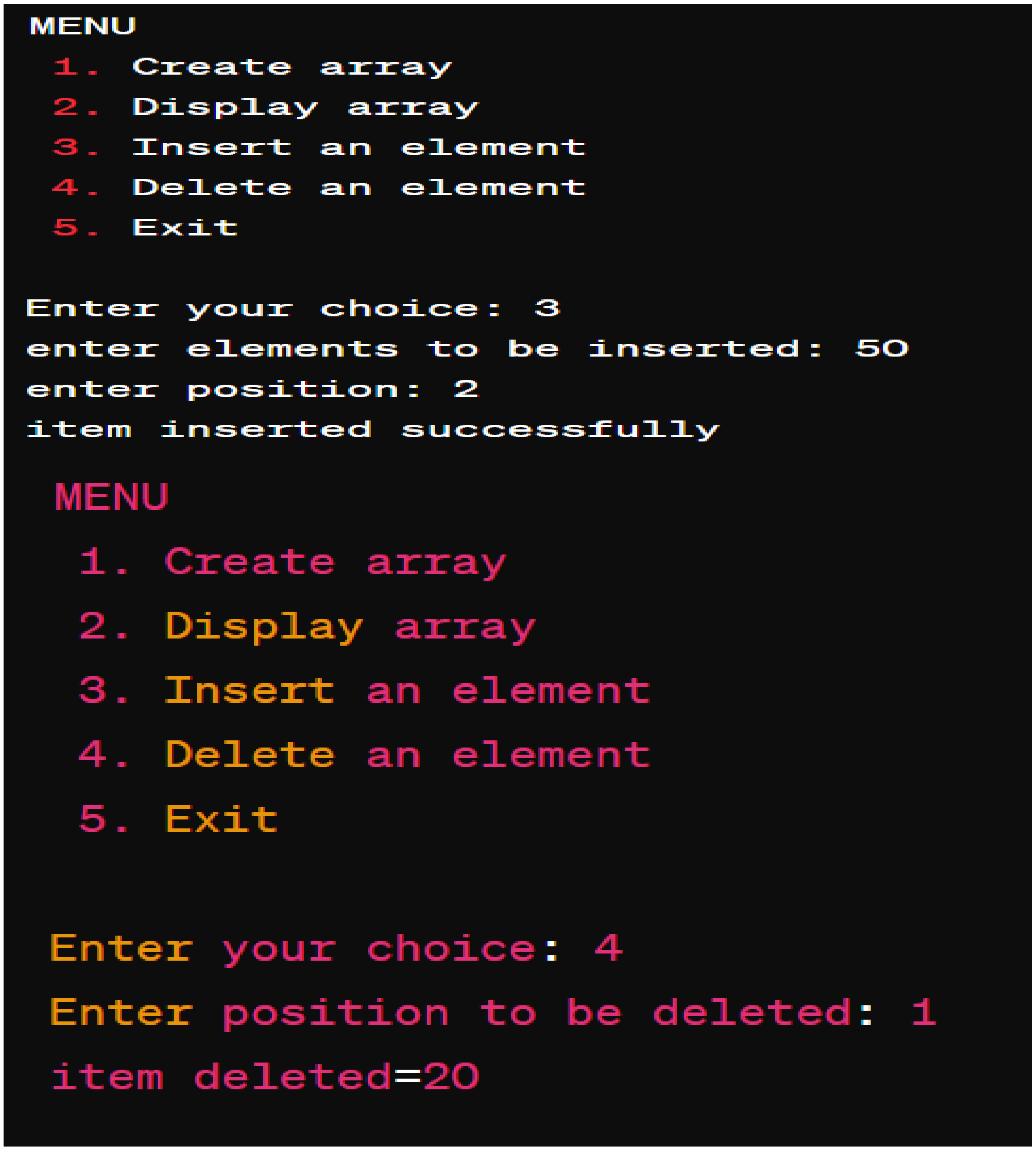
```
#include <stdio.h>


void create(int a[],int n)

{

    int i;

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

    {printf("array value");

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

}


void display(int a[],int n)

{

    int i;

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

    {printf("%d \n",a[i]);}

}

int insert(int ELEM,int a[],int n,int POS)

{

    int i;

    if(POS>n||POS<0)

    {printf("invalid position \n");
    return n;
    }

for(i=n-1;i>=POS;i--)

    a[i+1]=a[i];

    printf("item inserted successfully");
    a[POS]=ELEM;
    return n+1;
}

int delete(int a[],int n,int POS)
{
    int i;
    if(POS>=n||POS<0)
    {printf("invalid position \n");
    return n;
    }
    printf("item deleted=%d \n",a[POS]);
    for(i=POS+1;i<n;i++)
    {a[i-1]=a[i];}
    return n-1;
}
```

```
int main()
{
    int ch,a[25],n,ELEM,POS;
    do{
        printf("MENU \n 1.Create array \n 2.Dsplay array \n 3.Insert an element \n
4.Delete an element \n 5.Exit \n\n");
        printf("Enter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            printf("Enter no of elements");
            scanf("%d",&n);
            create(a,n);
            break;
            case 2:
            printf("array contents: \n");
            display(a,n);
            break;
            case 3:
            printf("enter elements to be inserted:");
            scanf("%d",&ELEM);
            printf("enter position");
            scanf("%d",&POS);
            n=insert(ELEM,a,n,POS);
            break;
            case 4:
            printf("Enter position to be deleted");
            scanf("%d",&POS);
            n=delete(a,n,POS);
            break;
            default:
            printf("Exiting program");
            return 0;
         }
      }

while(ch<5);
}
```

OUTPUT:



```
 MENU
  1. Create array
  2. Display array
  3. Insert an element
  4. Delete an element
  5. Exit

 Enter your choice: 1
 Enter no of elements: 4
 array value: 10
 array value: 20
 array value: 30
 array value: 40

MENU
  1. Create array
  2. Display array
  3. Insert an element
  4. Delete an element
  5. Exit

Enter your choice: 2
array contents:
10
20
30
40
```



```
MENU
 1.  Create array
 2.  Display array
 3.  Insert an element
 4.  Delete an element
 5.  Exit

Enter your choice: 3
enter elements to be inserted: 50
enter position: 2
item inserted successfully
 MENU
  1.  Create array
  2.  Display array
  3.  Insert an element
  4.  Delete an element
  5.  Exit

 Enter your choice: 4
 Enter position to be deleted: 1
 item deleted=20
```

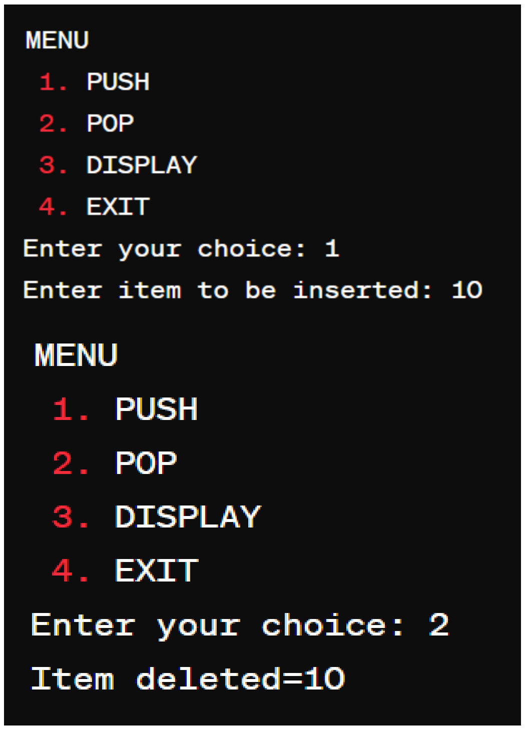
3. Design, Develop and Implement a menu driven Program in C for the following operations on STAIntegers (Array Implementation of Stack with maximum size MAX)

- **a. Pop an Element from Stack**
- **b. Demonstrate Overflow and Underflow situations on Stack**
- **c. Display the status of Stack**
- **d. Exit**

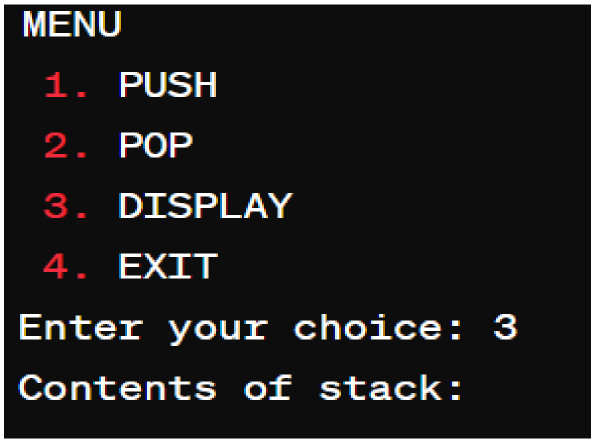
Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>
#define SIZE 10

void push(int item,int *top ,int s[])
{
   if(*top==SIZE-1)
   {
      printf("Stack Overflow \n");
      return;
   }
   *top=*top+1;
   s[*top]=item;
}

int pop(int*top,int s[])
{
   int del;
   if(*top==1)
   return 0;
   del=s[(*top)--];
   return del;
}

void display(int top , int s[])
{
   int i;
   if(top==-1)
   {printf("Stack Empty");
   return;
   }
   printf("contents of stack \n");
   for(i=top;i>=0;i--)
   printf("%d \n",s[i]);}
int main()
{int item,top,s[10],ch;
   top=-1;
while(1)
{printf("MENU \n 1.PUSH \n 2.POP \n 3.DISPLAY \n 4.EXIT");
   printf("enter your choice");
   scanf("%d",&ch);
   switch(ch)
   {
      case 1:
      printf("enter item to be inserted");
      scanf("%d",&item);
      push(item,&top,s);
      break;
      case 2:
      item=pop(&top,s);
      if(item==0)
      printf("Stack underflow");
      else
      printf("item deleted=%d",item);
      break;
      case 3:
      display(top,s);
      break;
      default:
      return(0);}}}
```

OUTPUT:



```
MENU
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
Enter your choice: 1
Enter item to be inserted: 10

MENU
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
Enter your choice: 2
Item deleted=10
```



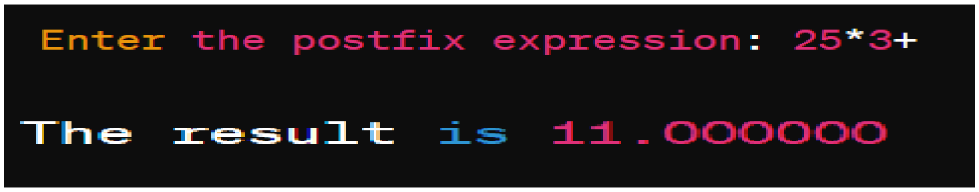
```
MENU
 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT
Enter your choice: 3
Contents of stack:
```

4. **Design, Develop and Implement a Program in C for the following Stack Applications**

a. **Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^**

```
#include  <stdio.h>

#include  <math.h>

#include <string.h>

1* Function to evaluate *1

double compute( char symbol, double op l, double op2)

{

switch(symbol)

{case '+': return op 1 + op2;

case '-':return opl - op2;

case '*': return opl * op2;

case 'I': return op 1 I op2;

case '$':

case "^":return pow(opl,op2);

}

}

void main()

{

double

[20],res;op1,op2;

int top;

int 1;

char postfix[20];

char symbol;
printf("Enter the postfix expression\n");
scanf("%s" ,postfix);
top = -1;
for(i = 0; i < strlen(postfix); i++)
 {
symbol = postfix[i];
if ( isdigit(symbol) )
s[top++] = symbol-'0';
else
{
op2 = s[top--];
op1 = s[top--];
res = compute(symbol,op1,op2);
s[++top] = res;
}
}
res = s[top--];
printf("The result is %f\n",res);
}
```

OUTPUT:

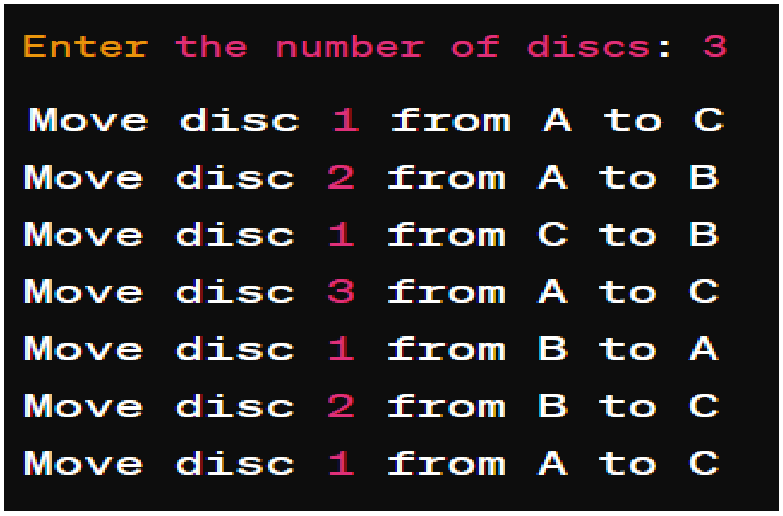


```
 Enter the postfix expression: 25*3+

The result is 11.000000
```

4 b.Solving Tower of Hanoi problem with n disks

```
#include<stdio.h>
void toh(int n, char s, char d,char a)
{
if (n ==0) return;
toh(n-1,s,a,d);
printf("Move disc %d from %c to %c\n",n, s,d);
toh(n-1, a, d, s);
}
void main()
{
int n;
printf("Enter the number of discs:\n");
scanf("%d" ,&n);
toh(n ,'A','C','B')·,
}
```

OUTPUT:



```
Enter the number of discs: 3

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C
```

5. A) Singly Linked List (SLL) of Integer Data

a. And menu driven Program in C for the following operations

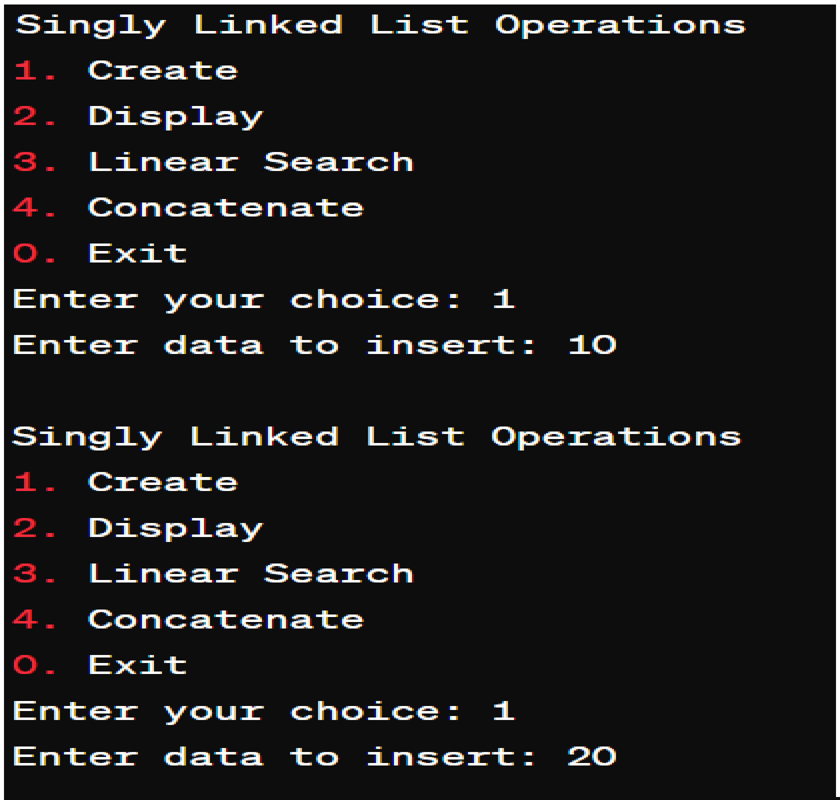
i. **Create**
ii. **Display**
iii. **Linear search**
iv. **Concatination**

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

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

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

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

// Function to perform linear search in the linked list
int linearSearch(struct Node* head, int key) {
    struct Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            return pos;
        }
        temp = temp->next;
        pos++;
    }
    return -1; // Key not found
}

// Function to concatenate two linked lists
void concatenate(struct Node* list1, struct Node* list2) {
    struct Node* temp = list1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = list2;
```

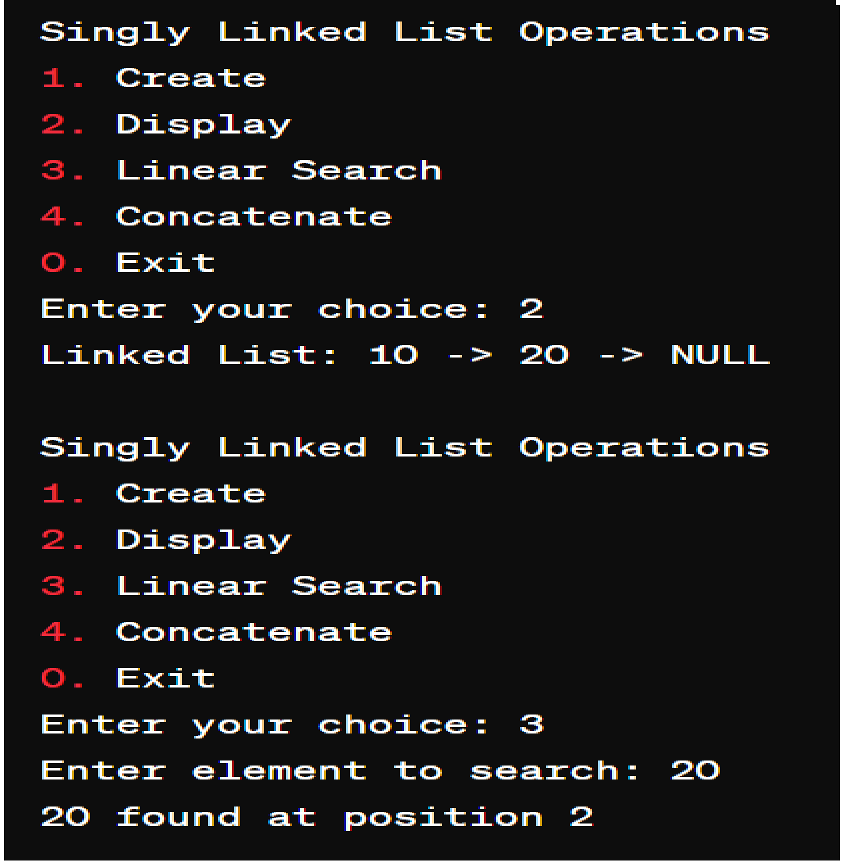
```
}

int main() {
    struct Node* head = NULL;
    struct Node* head2 = NULL;
    int choice, data, key;

    do {
        printf("\nSingly Linked List Operations\n");
        printf("1. Create\n");
        printf("2. Display\n");
        printf("3. Linear Search\n");
        printf("4. Concatenate\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                if (head == NULL) {
                    head = createNode(data);
                } else {
                    struct Node* temp = head;
                    while (temp->next != NULL) {
                        temp = temp->next;
                    }
                    temp->next = createNode(data);
                }
                break;
            case 2:
                if (head == NULL) {
                    printf("List is empty\n");
                } else {
                    printf("Linked List: ");
                    displayList(head);
                }
                break;
            case 3:
                printf("Enter element to search: ");
                scanf("%d", &key);
                if (head == NULL) {
                    printf("List is empty\n");
                } else {
                    int pos = linearSearch(head, key);
                    if (pos != -1) {
                        printf("%d found at position %d\n", key, pos);
                    } else {
                        printf("%d not found in the list\n", key);
                    }
                }
                break;
            case 4:
```

```
            printf("Creating second list to concatenate...\n");
            printf("Enter data to insert into second list: ");
            scanf("%d", &data);
            if (head2 == NULL) {
                head2 = createNode(data);
            } else {
                struct Node* temp = head2;
                while (temp->next != NULL) {
                    temp = temp->next;
                }
                temp->next = createNode(data);
            }
            concatenate(head, head2);
            printf("Lists concatenated successfully!\n");
            break;
        case 0:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
  } while (choice != 0);

  return 0;
}
```

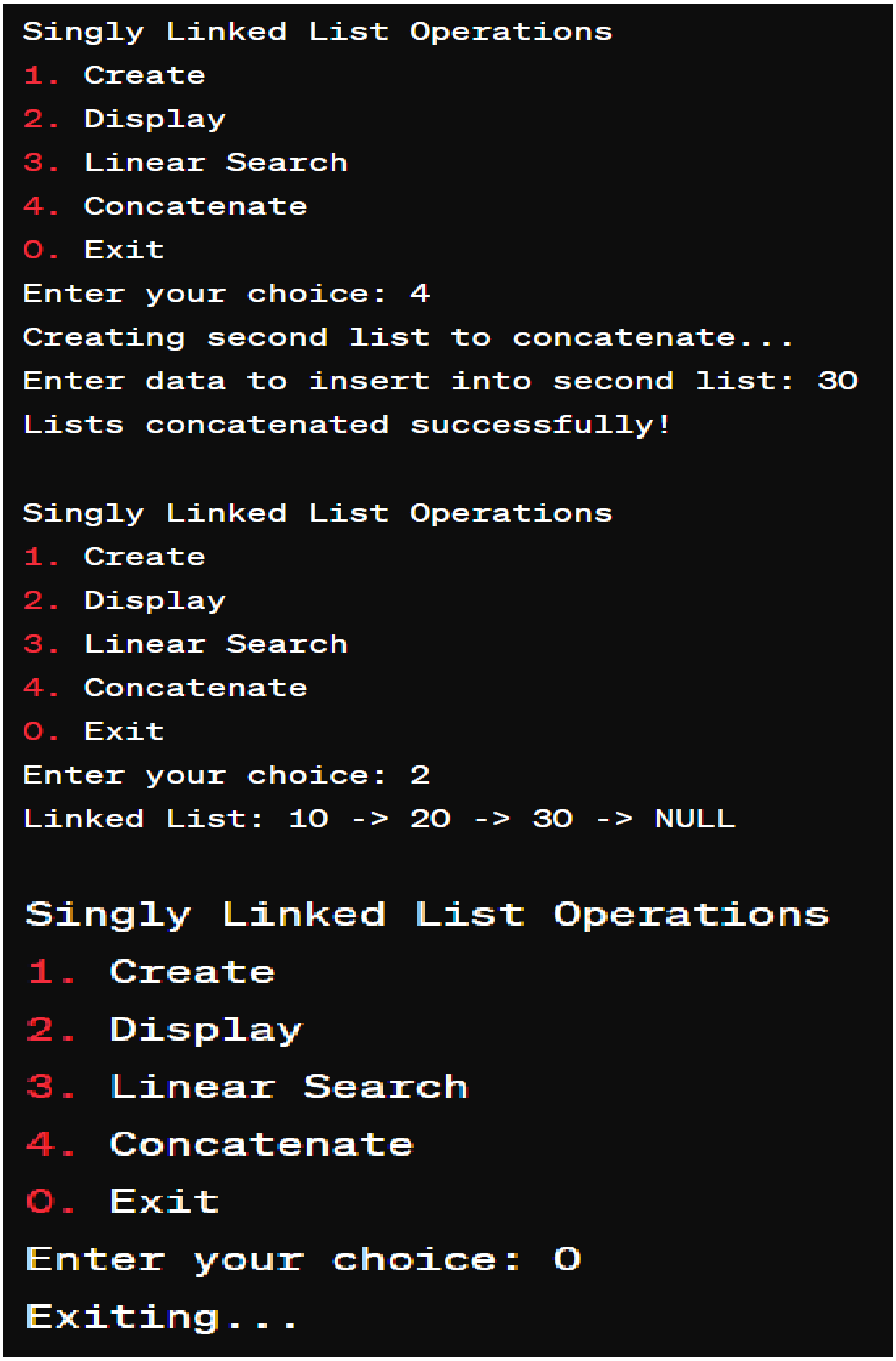
OUTPUT:



```
Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 1
Enter data to insert: 10

Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 1
Enter data to insert: 20
```



```
Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 2
Linked List: 10 -> 20 -> NULL

Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 3
Enter element to search: 20
20 found at position 2
```

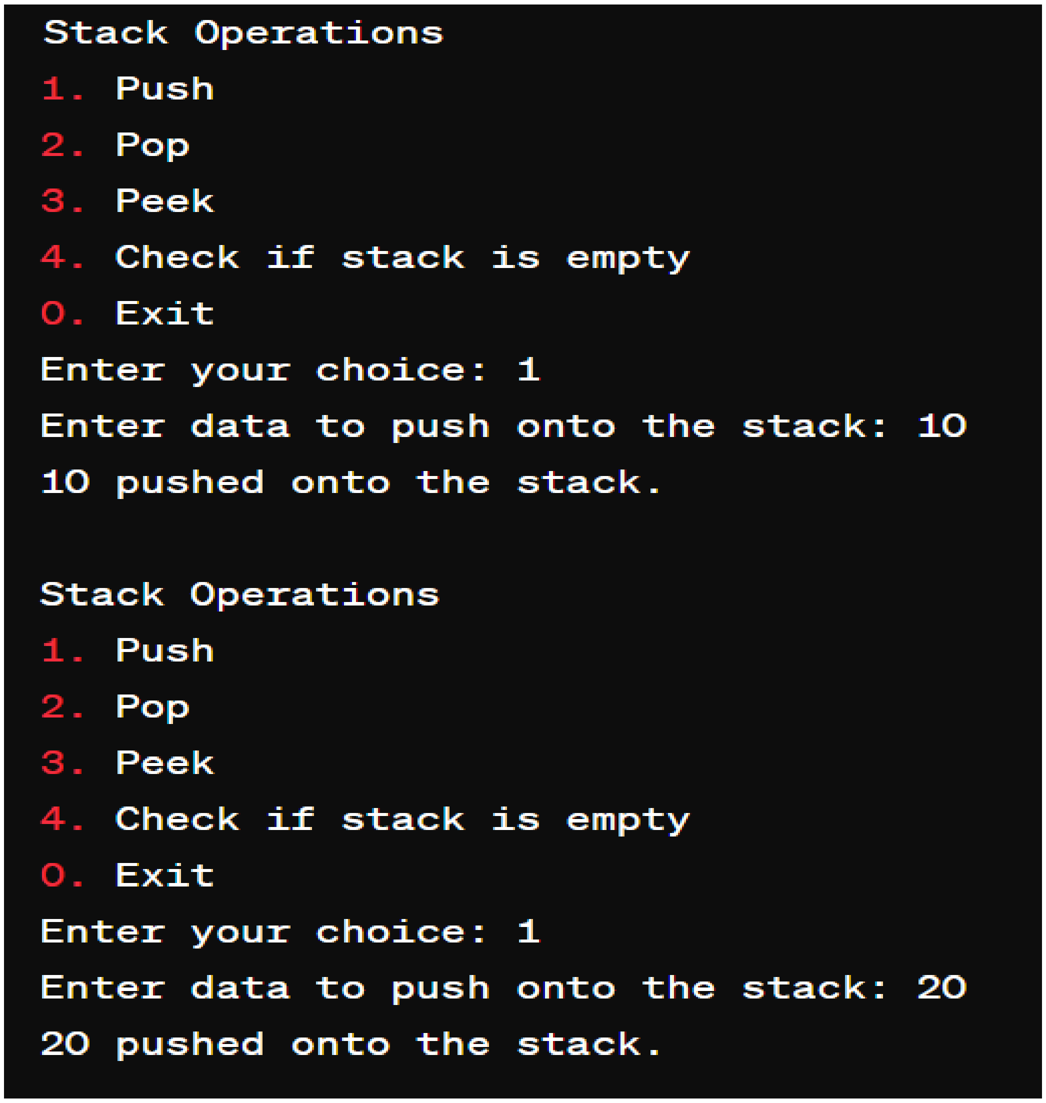

```
Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 4
Creating second list to concatenate...
Enter data to insert into second list: 30
Lists concatenated successfully!

Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 2
Linked List: 10 -> 20 -> 30 -> NULL

Singly Linked List Operations
1. Create
2. Display
3. Linear Search
4. Concatenate
0. Exit
Enter your choice: 0
Exiting...
```

5)b. Create a SLL stack of integers

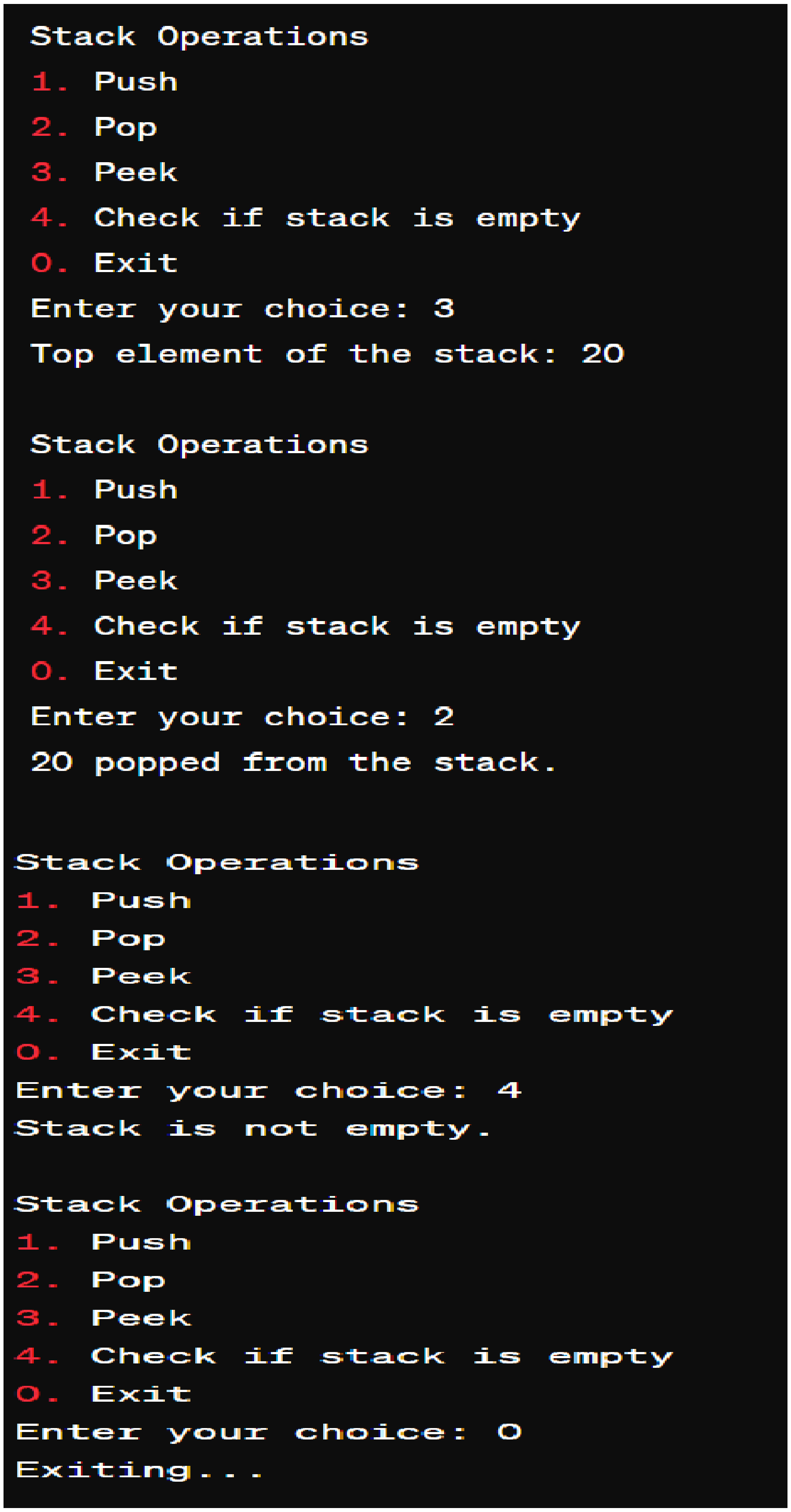
```
#include <stdio.h>
#include <stdlib.h>

// Define the structure of a node
struct Node {
   int data;
   struct Node* next;
};

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

// Function to push a new element onto the stack
void push(struct Node** top, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *top;
   *top = newNode;
   printf("%d pushed onto the stack.\n", data);
}

// Function to pop an element from the stack
int pop(struct Node** top) {
   if (*top == NULL) {
      printf("Stack underflow.\n");
      return -1;
   }
   struct Node* temp = *top;
   *top = (*top)->next;
   int popped = temp->data;
   free(temp);
   return popped;
}

// Function to check if the stack is empty
int isEmpty(struct Node* top) {
   return (top == NULL);
}

// Function to return the top element of the stack
```

```
int peek(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return -1;
    }
    return top->data;
}

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

    do {
        printf("\nStack Operations\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Check if stack is empty\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push onto the stack: ");
                scanf("%d", &data);
                push(&top, data);
                break;
            case 2:
                data = pop(&top);
                if (data != -1) {
                    printf("%d popped from the stack.\n", data);
                }
                break;
            case 3:
                data = peek(top);
                if (data != -1) {
                    printf("Top element of the stack: %d\n", data);
                }
                break;
            case 4:
                if (isEmpty(top)) {
                    printf("Stack is empty.\n");
                } else {
                    printf("Stack is not empty.\n");
                }
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 0);

    // Free memory before exiting
    while (top != NULL) {
        pop(&top);
    }

    return 0;
}
```

OUTPUT:



```
Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 1
Enter data to push onto the stack: 10
10 pushed onto the stack.

Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 1
Enter data to push onto the stack: 20
20 pushed onto the stack.
```



```
Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 3
Top element of the stack: 20

Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 2
20 popped from the stack.

Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 4
Stack is not empty.

Stack Operations
1. Push
2. Pop
3. Peek
4. Check if stack is empty
0. Exit
Enter your choice: 0
Exiting...
```

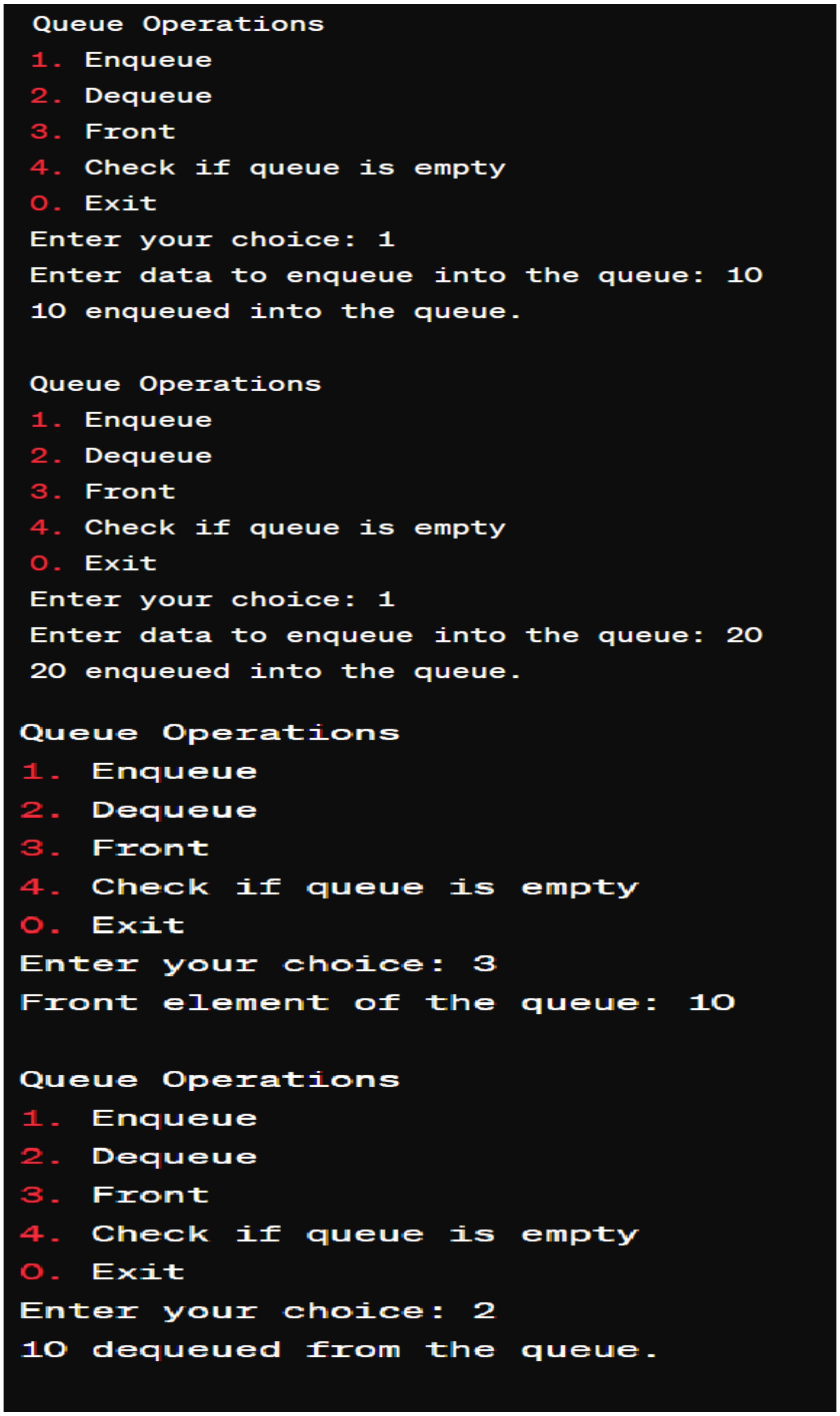
5) c. Create a SLL Oueue of integers

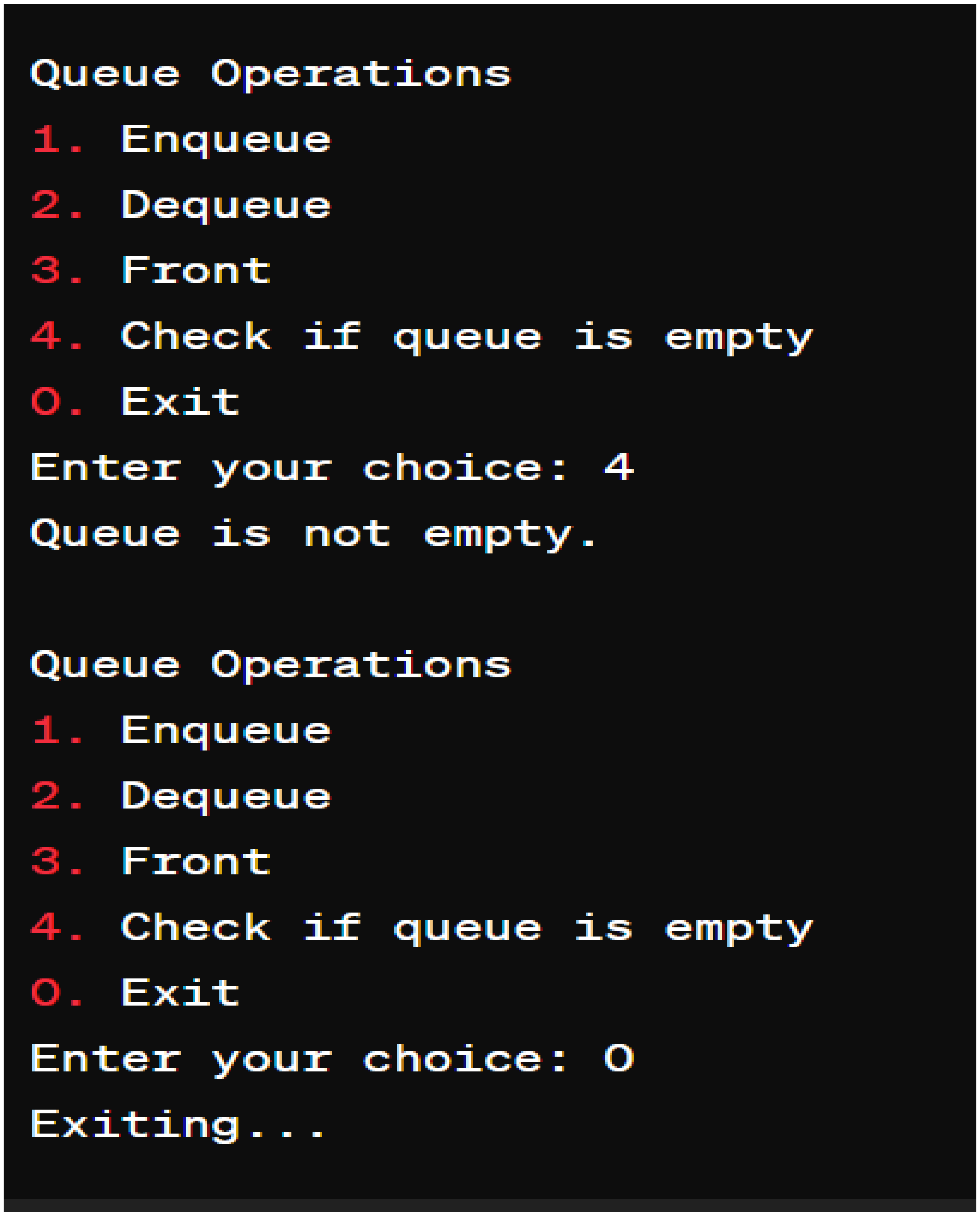
```
#include <stdio.h>
#include <stdlib.h>

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

// Define the structure of a queue
struct Queue {
    struct Node* front;
    struct Node* rear;
};

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

// Function to create an empty queue
```

```
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue (insert) an element into the queue
void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = createNode(data);
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    printf("%d enqueued into the queue.\n", data);
}

// Function to dequeue (remove) an element from the queue
int dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }
    struct Node* temp = queue->front;
    int dequeued = temp->data;
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    free(temp);
    return dequeued;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == NULL);
}

// Function to return the front element of the queue
int front(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->front->data;
}

int main() {
    struct Queue* queue = createQueue();
```

```
int choice, data;

do {
    printf("\nQueue Operations\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Front\n");
    printf("4. Check if queue is empty\n");
    printf("0. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to enqueue into the queue: ");
            scanf("%d", &data);
            enqueue(queue, data);
            break;
        case 2:
            data = dequeue(queue);
            if (data != -1) {
                printf("%d dequeued from the queue.\n", data);
            }
            break;
        case 3:
            data = front(queue);
            if (data != -1) {
                printf("Front element of the queue: %d\n", data);
            }
            break;
        case 4:
            if (isEmpty(queue)) {
                printf("Queue is empty.\n");
            } else {
                printf("Queue is not empty.\n");
            }
            break;
        case 0:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
} while (choice != 0);

// Free memory before exiting
while (!isEmpty(queue)) {
    dequeue(queue);}
free(queue);
return 0;}
```

OUTPUT:


```
Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 1
Enter data to enqueue into the queue: 10
10 enqueued into the queue.

Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 1
Enter data to enqueue into the queue: 20
20 enqueued into the queue.

Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 3
Front element of the queue: 10

Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 2
10 dequeued from the queue.
```



```
Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 4
Queue is not empty.

Queue Operations
1. Enqueue
2. Dequeue
3. Front
4. Check if queue is empty
0. Exit
Enter your choice: 0
Exiting...
```

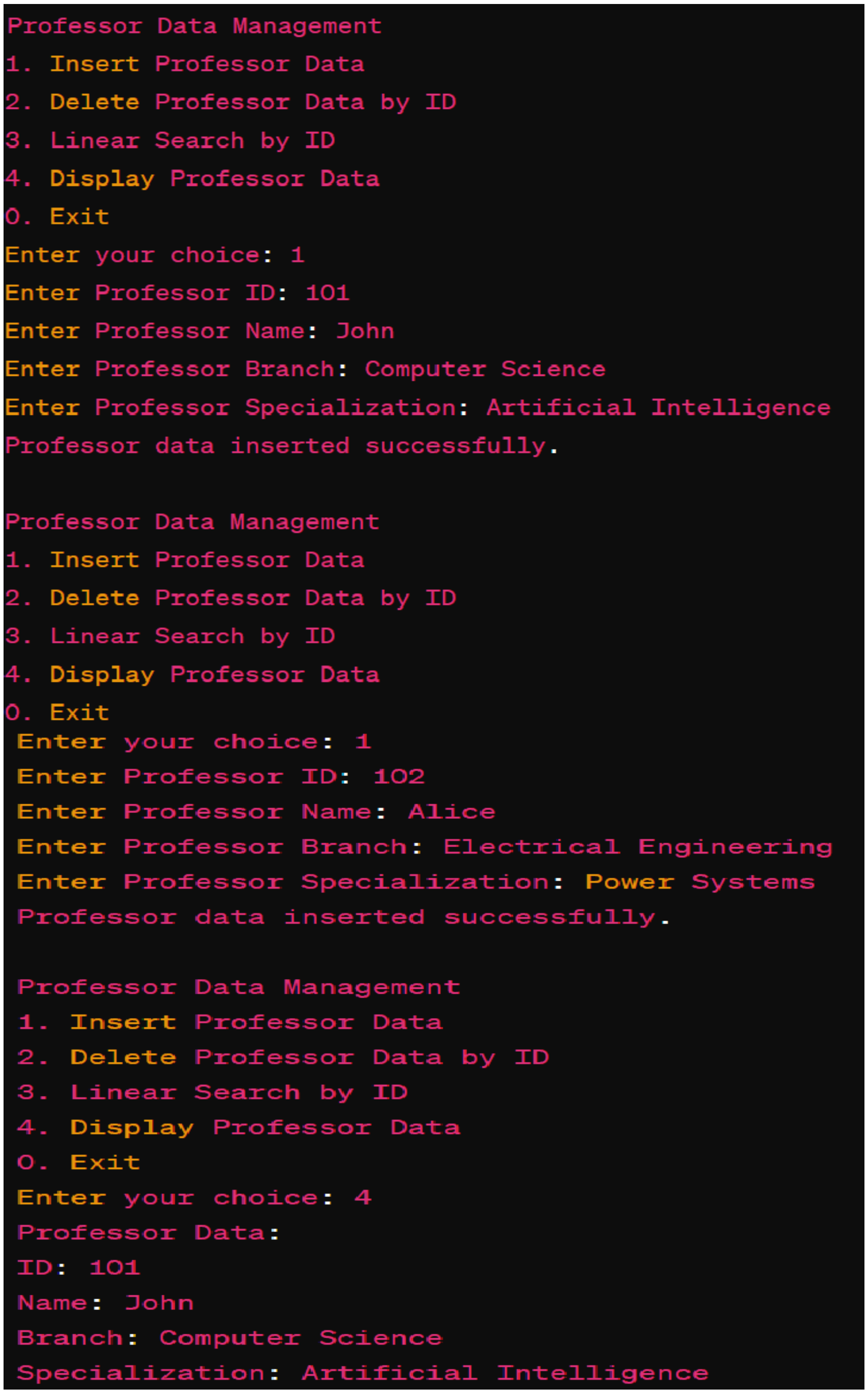
6. a) Design, Develop and Implement a menu driven Program in C for the following operationson DoublyLinked List (DLL) of Professor Data with the fields: ID, Name, Branch, Area of specialization

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

// Define the structure of a node
struct Node {
    int id;
    char name[50];
    char branch[50];
    char specialization[50];
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int id, char* name, char* branch, char* specialization) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
    strcpy(newNode->name, name);
    strcpy(newNode->branch, branch);
    strcpy(newNode->specialization, specialization);
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the end of the list
void insertEnd(struct Node** head, int id, char* name, char* branch, char*
specialization) {
    struct Node* newNode = createNode(id, name, branch, specialization);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Function to delete a node based on entered ID
void deleteById(struct Node** head, int id) {
    struct Node* current = *head;
    while (current != NULL) {
        if (current->id == id) {
            if (current->prev != NULL) {
                current->prev->next = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            }
            if (current == *head) {
                *head = current->next;
            }
            free(current);
            printf("Professor with ID %d deleted successfully.\n", id);
            return;
        }
        current = current->next;
    }
    printf("Professor with ID %d not found in the list.\n", id);
}

// Function to perform linear search based on entered ID
void linearSearch(struct Node* head, int id) {
    struct Node* current = head;
    while (current != NULL) {
        if (current->id == id) {
            printf("Professor found:\n");
            printf("ID: %d\nName: %s\nBranch: %s\nSpecialization: %s\n",
current->id, current->name, current->branch, current->specialization);
            return;
        }
        current = current->next;
    }
    printf("Professor with ID %d not found in the list.\n", id);
}

// Function to display the list
void displayList(struct Node* head) {
    printf("Professor Data:\n");
    while (head != NULL) {
        printf("ID: %d\nName: %s\nBranch: %s\nSpecialization: %s\n\n",
head->id, head->name, head->branch, head->specialization);
        head = head->next;
    }
}

int main() {
    struct Node* head = NULL;
    int choice, id;
    char name[50], branch[50], specialization[50];

    do {
        printf("\nProfessor Data Management\n");
        printf("1. Insert Professor Data\n");
        printf("2. Delete Professor Data by ID\n");
        printf("3. Linear Search by ID\n");
        printf("4. Display Professor Data\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

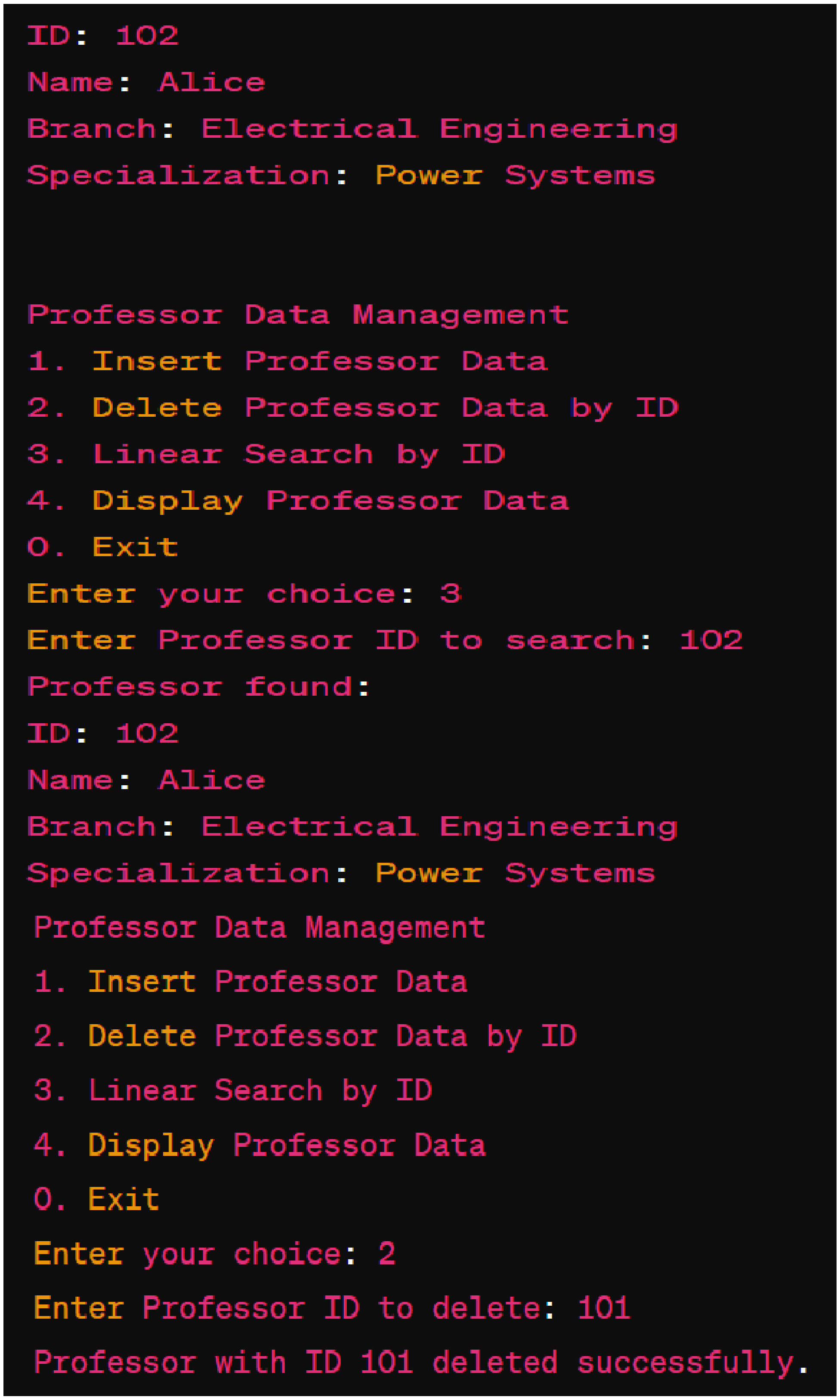
```
switch (choice) {
    case 1:
        printf("Enter Professor ID: ");
        scanf("%d", &id);
        printf("Enter Professor Name: ");
        scanf("%s", name);
        printf("Enter Professor Branch: ");
        scanf("%s", branch);
        printf("Enter Professor Specialization: ");
        scanf("%s", specialization);
        insertEnd(&head, id, name, branch, specialization);
        printf("Professor data inserted successfully.\n");
        break;
    case 2:
        printf("Enter Professor ID to delete: ");
        scanf("%d", &id);
        deleteById(&head, id);
        break;
    case 3:
        printf("Enter Professor ID to search: ");
        scanf("%d", &id);
        linearSearch(head, id);
        break;
    case 4:
        if (head == NULL) {
            printf("List is empty.\n");
```

```
                } else {
                    displayList(head);
                }
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 0);

    // Free memory before exiting
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

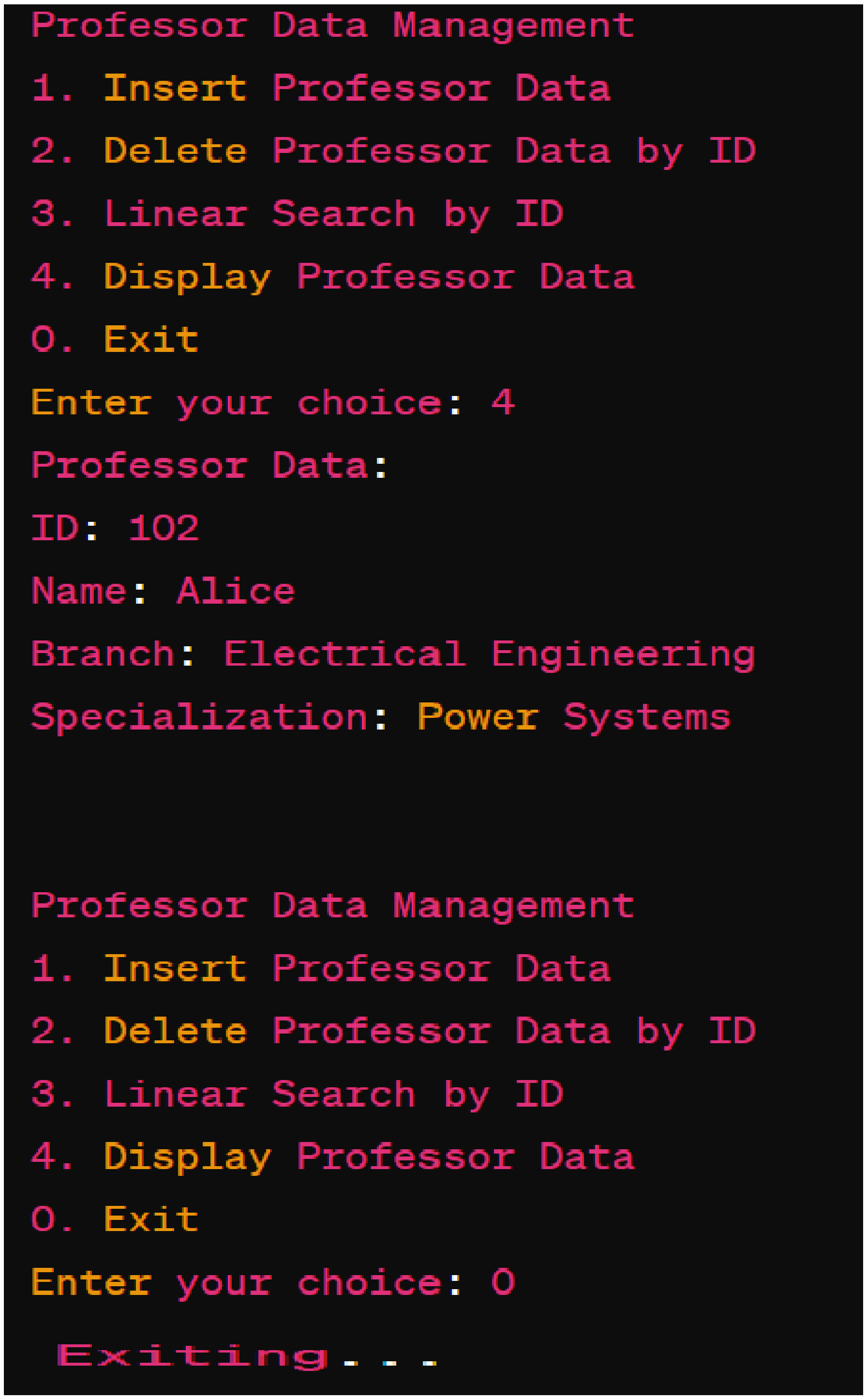
OUTPUT:



```
Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 1
Enter Professor ID: 101
Enter Professor Name: John
Enter Professor Branch: Computer Science
Enter Professor Specialization: Artificial Intelligence
Professor data inserted successfully.

Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 1
Enter Professor ID: 102
Enter Professor Name: Alice
Enter Professor Branch: Electrical Engineering
Enter Professor Specialization: Power Systems
Professor data inserted successfully.

Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 4
Professor Data:
ID: 101
Name: John
Branch: Computer Science
Specialization: Artificial Intelligence
```



```
ID: 102
Name: Alice
Branch: Electrical Engineering
Specialization: Power Systems


Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 3
Enter Professor ID to search: 102
Professor found:
ID: 102
Name: Alice
Branch: Electrical Engineering
Specialization: Power Systems
Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 2
Enter Professor ID to delete: 101
Professor with ID 101 deleted successfully.
```



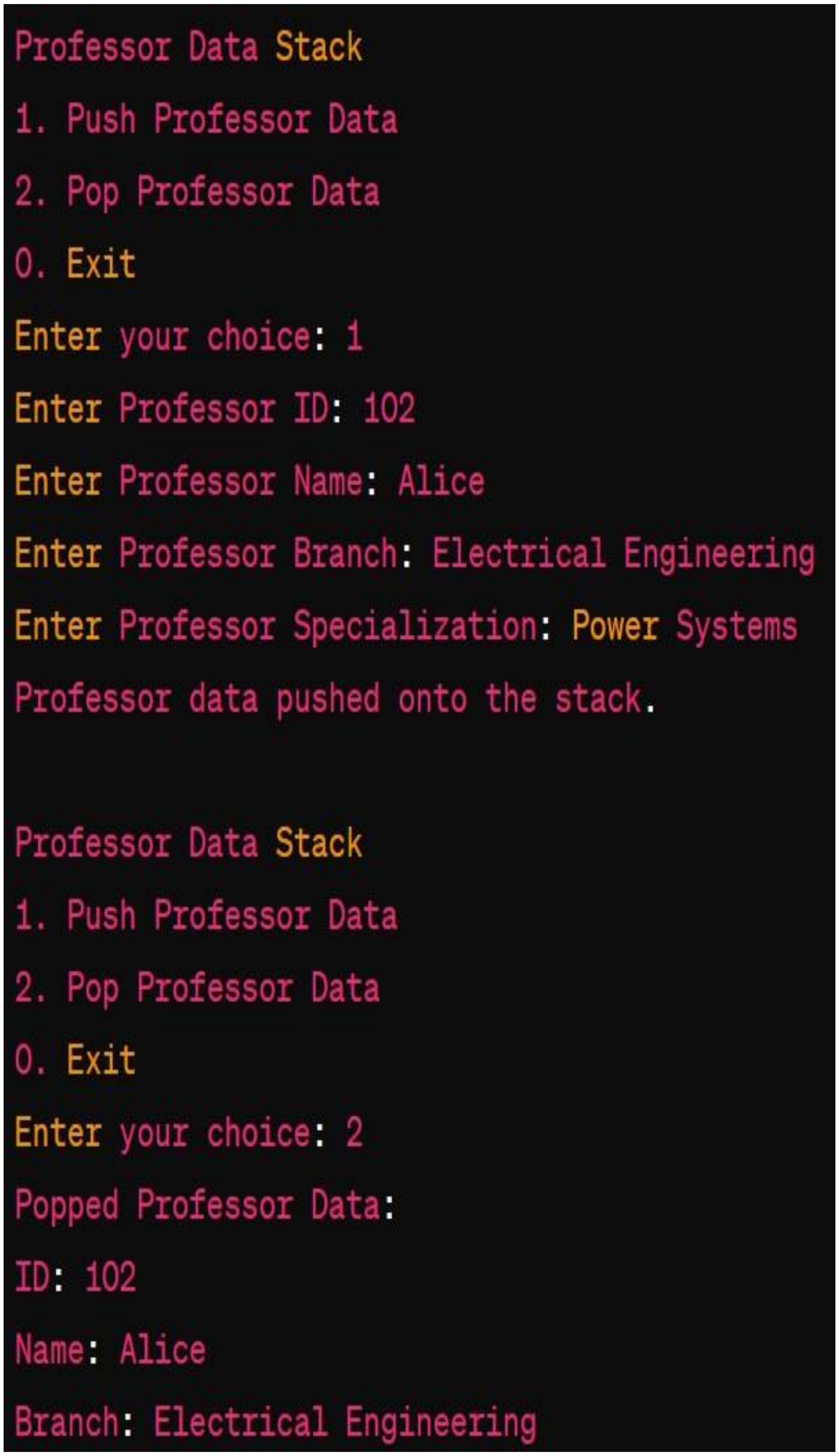
```
Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 4
Professor Data:
ID: 102
Name: Alice
Branch: Electrical Engineering
Specialization: Power Systems


Professor Data Management
1. Insert Professor Data
2. Delete Professor Data by ID
3. Linear Search by ID
4. Display Professor Data
0. Exit
Enter your choice: 0
 Exiting...
```

6) b. Create a DLL stack of N Professor's Data.

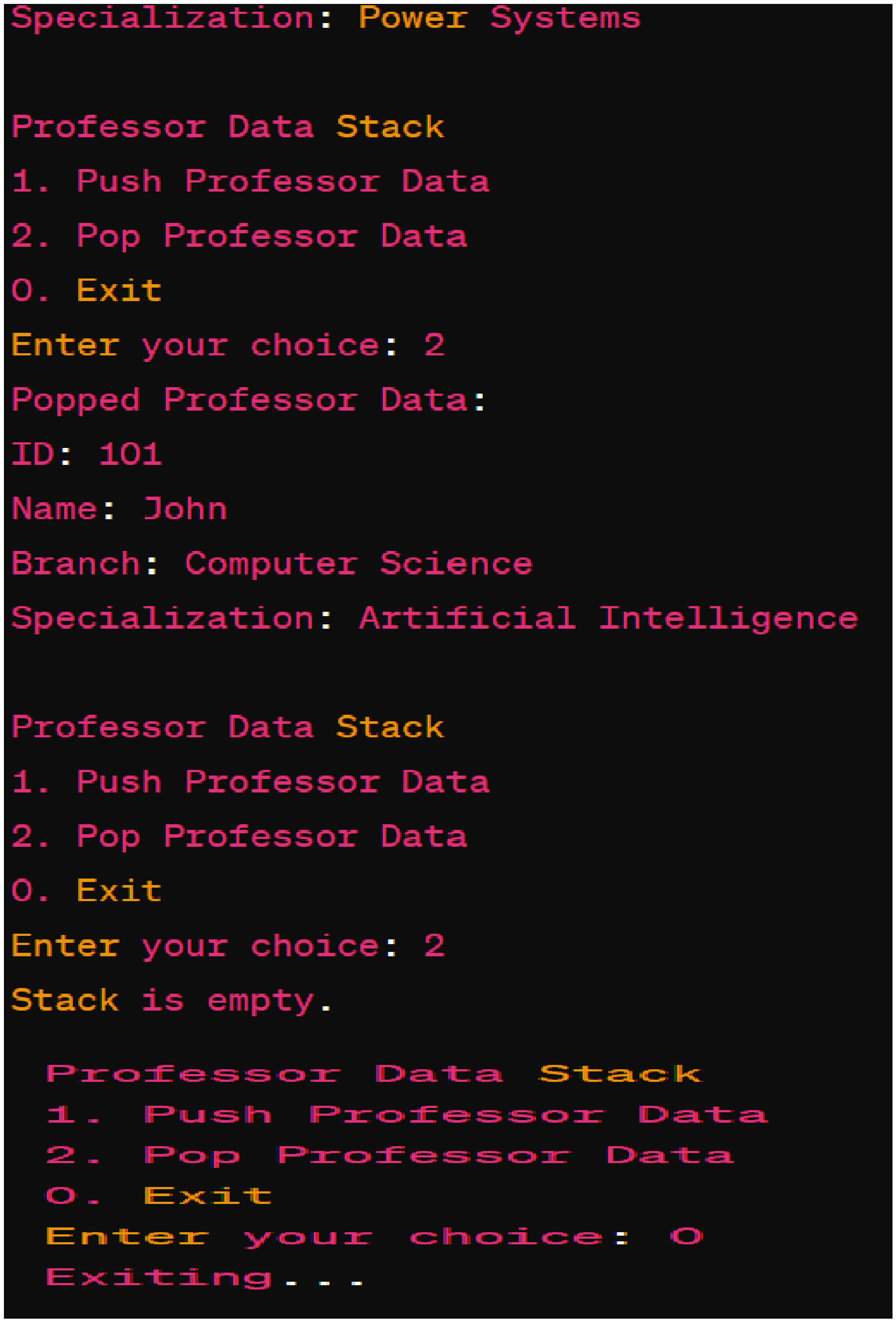
```
  #include <stdio.h>
#include  <stdlib.h>
#include <string.h>

// Define the structure of a node
struct Node {
    int id;
    char name[50];
    char branch[50];
    char specialization[50];
    struct Node* prev;
    struct Node* next;
};

// Define the structure of a stack
struct Stack {
    struct Node* top;
};

// Function to create a new node
struct Node* createNode(int id, char* name, char* branch, char* specialization) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
    strcpy(newNode->name, name);
    strcpy(newNode->branch, branch);
    strcpy(newNode->specialization, specialization);
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to create an empty stack
struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack;
}

// Function to push a new element onto the stack
void push(struct Stack* stack, int id, char* name, char* branch, char* specialization) {
    struct Node* newNode = createNode(id, name, branch, specialization);
    if (stack->top == NULL) {
        stack->top = newNode;
    } else {
        newNode->next = stack->top;
        stack->top->prev = newNode;
        stack->top = newNode;
    }
    printf("Professor data pushed onto the stack.\n");
}

// Function to pop an element from the stack
void pop(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct Node* temp = stack->top;
    printf("Popped Professor Data:\n");
    printf("ID: %d\nName: %s\nBranch: %s\nSpecialization: %s\n\n", temp->id, temp-
>name, temp->branch, temp->specialization);
    stack->top = stack->top->next;
    if (stack->top != NULL) {
        stack->top->prev = NULL;
    }
    free(temp);
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return (stack->top == NULL);
}

int main() {
```

```
struct Stack* stack = createStack();
int choice, id;
char name[50], branch[50], specialization[50];

do {
    printf("\nProfessor Data Stack\n");
    printf("1. Push Professor Data\n");
    printf("2. Pop Professor Data\n");
    printf("0. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter Professor ID: ");
            scanf("%d", &id);
            printf("Enter Professor Name: ");
            scanf("%s", name);
            printf("Enter Professor Branch: ");
            scanf("%s", branch);
            printf("Enter Professor Specialization: ");
            scanf("%s", specialization);
            push(stack, id, name, branch, specialization);
            break;
        case 2:
            pop(stack);
            break;
        case 0:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
        }
    } while (choice != 0);

    // Free memory before exiting
    while (!isEmpty(stack)) {
        pop(stack);
    }
    free(stack);

    return 0;
}
```

OUTPUT:



```
Professor Data Stack
1. Push Professor Data
2. Pop Professor Data
0. Exit
Enter your choice: 1
Enter Professor ID: 102
Enter Professor Name: Alice
Enter Professor Branch: Electrical Engineering
Enter Professor Specialization: Power Systems
Professor data pushed onto the stack.

Professor Data Stack
1. Push Professor Data
2. Pop Professor Data
0. Exit
Enter your choice: 2
Popped Professor Data:
ID: 102
Name: Alice
Branch: Electrical Engineering
```



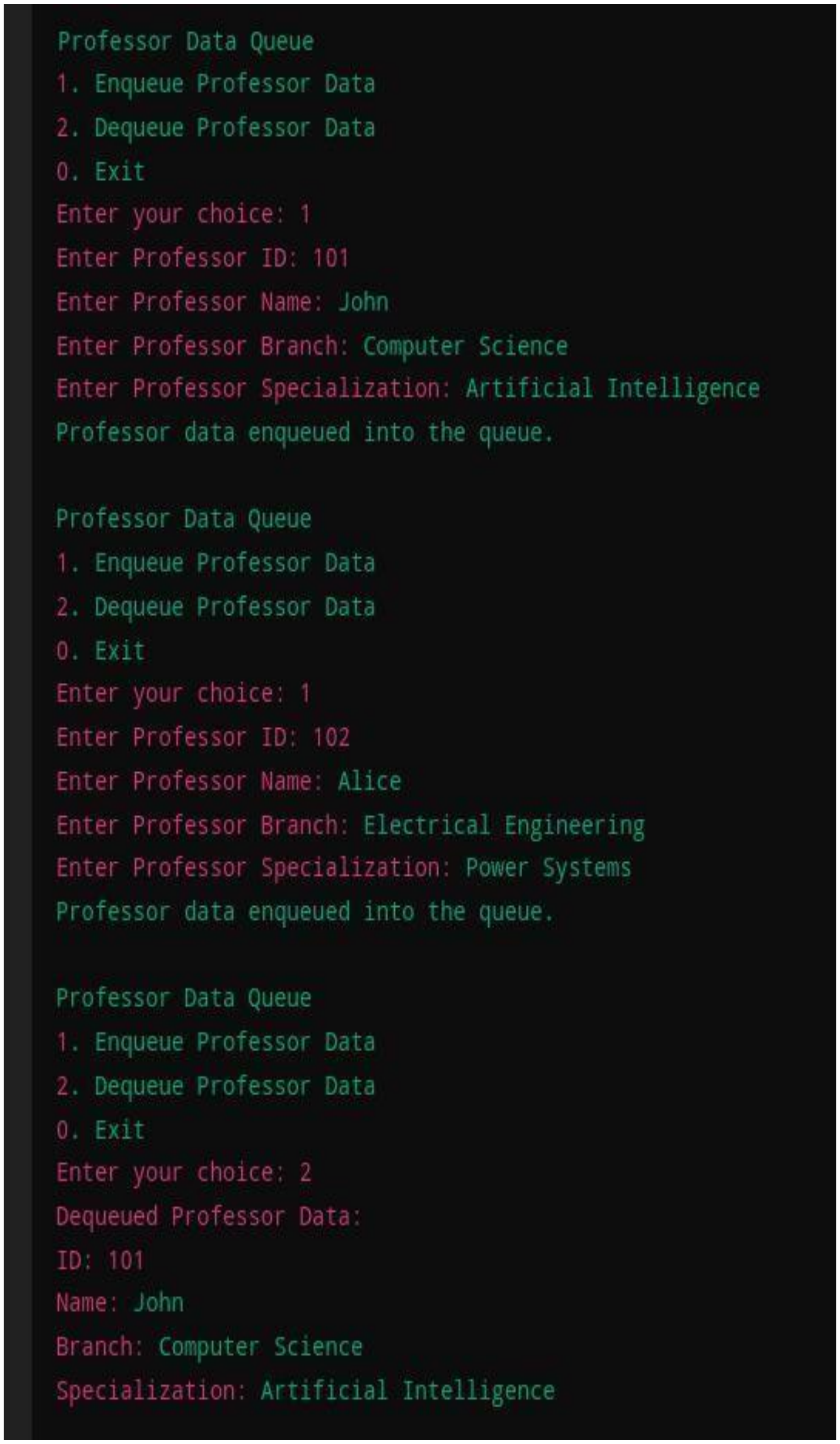
```
Specialization: Power Systems

Professor Data Stack
1. Push Professor Data
2. Pop Professor Data
0. Exit
Enter your choice: 2
Popped Professor Data:
ID: 101
Name: John
Branch: Computer Science
Specialization: Artificial Intelligence

Professor Data Stack
1. Push Professor Data
2. Pop Professor Data
0. Exit
Enter your choice: 2
Stack is empty.

 Professor Data Stack
 1. Push Professor Data
 2. Pop Professor Data
 0. Exit
 Enter your choice: 0
 Exiting...
```

6)c. Create a DLL queue of N Professor's Data

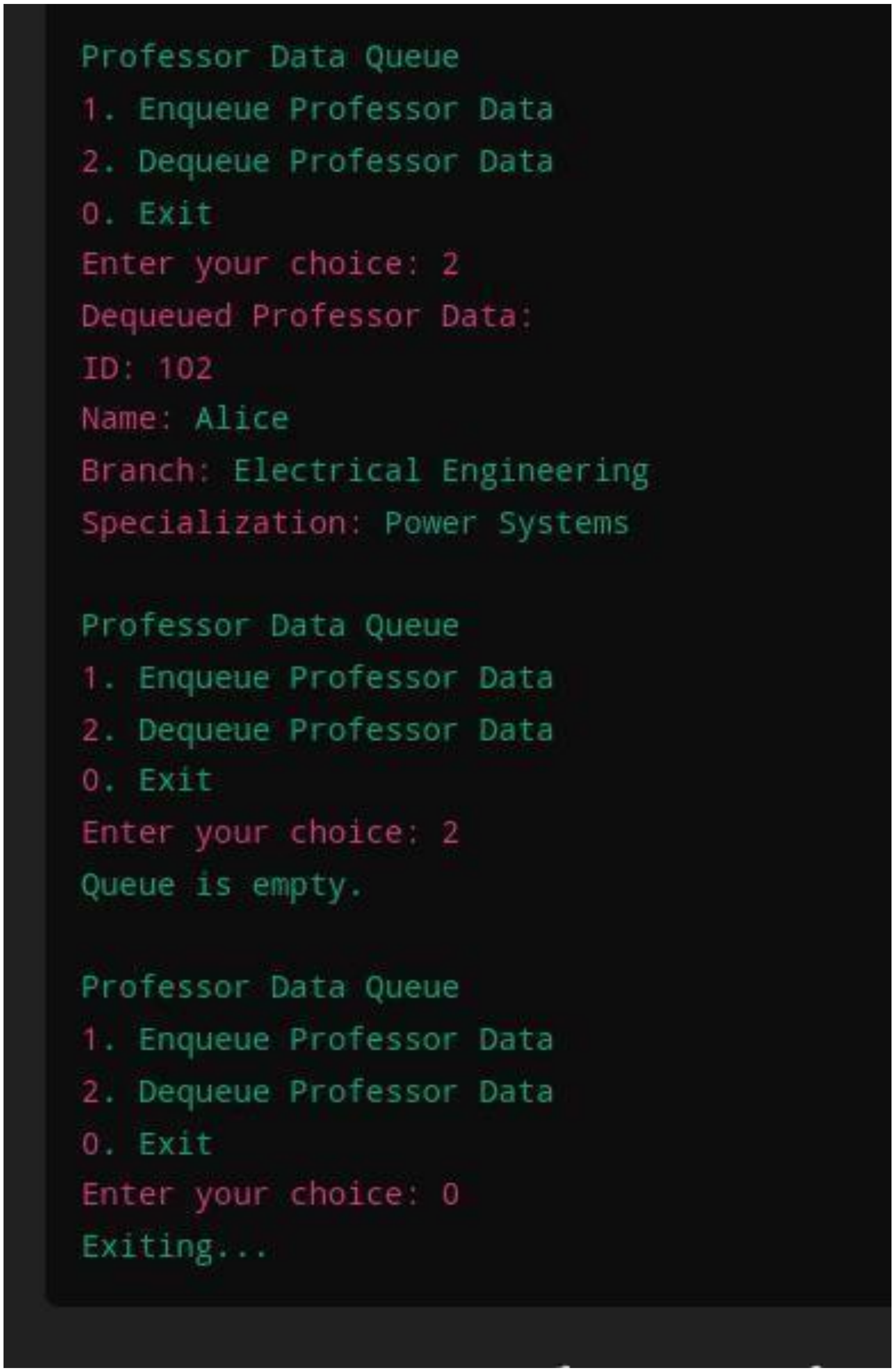
```
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define the structure of a node
struct Node {
    int id;
    char name[50];
    char branch[50];
    char specialization[50];
    struct Node* prev;
    struct Node* next;
};

// Define the structure of a queue
struct Queue {
    struct Node* front;
    struct Node* rear;
};

// Function to create a new node
struct Node* createNode(int id, char* name, char* branch, char* specialization) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
    strcpy(newNode->name, name);
    strcpy(newNode->branch, branch);
    strcpy(newNode->specialization, specialization);
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue (insert) an element into the queue
void enqueue(struct Queue* queue, int id, char* name, char* branch, char*
specialization) {
    struct Node* newNode = createNode(id, name, branch, specialization);
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        newNode->prev = queue->rear;
        queue->rear = newNode;
    }
    printf("Professor data enqueued into the queue.\n");
}

// Function to dequeue (remove) an element from the queue
void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = queue->front;
    printf("Dequeued Professor Data:\n");
    printf("ID: %d\nName: %s\nBranch: %s\nSpecialization: %s\n\n", temp->id,
temp->name, temp->branch, temp->specialization);
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    } else {
        queue->front->prev = NULL;
    }
    free(temp);
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == NULL);
```

```
}

int main() {
    struct Queue* queue = createQueue();
    int choice, id;
    char name[50], branch[50], specialization[50];

    do {
        printf("\nProfessor Data Queue\n");
        printf("1. Enqueue Professor Data\n");
        printf("2. Dequeue Professor Data\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Professor ID: ");
                scanf("%d", &id);
                printf("Enter Professor Name: ");
                scanf("%s", name);
                printf("Enter Professor Branch: ");
                scanf("%s", branch);
                printf("Enter Professor Specialization: ");
                scanf("%s", specialization);
                enqueue(queue, id, name, branch, specialization);
                break;
            case 2:
                dequeue(queue);
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 0);

    // Free memory before exiting
    while (!isEmpty(queue)) {
        dequeue(queue);
    }
    free(queue);

    return 0;
}
```

OUTPUT:



```
Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 1
Enter Professor ID: 101
Enter Professor Name: John
Enter Professor Branch: Computer Science
Enter Professor Specialization: Artificial Intelligence
Professor data enqueued into the queue.

Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 1
Enter Professor ID: 102
Enter Professor Name: Alice
Enter Professor Branch: Electrical Engineering
Enter Professor Specialization: Power Systems
Professor data enqueued into the queue.

Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 2
Dequeued Professor Data:
ID: 101
Name: John
Branch: Computer Science
Specialization: Artificial Intelligence
```



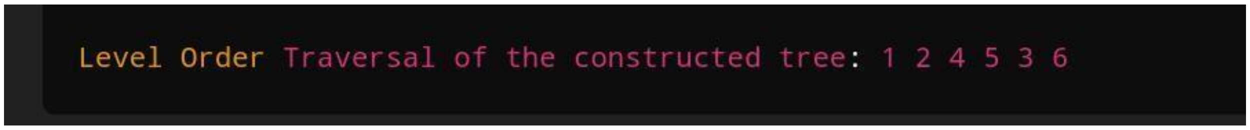
```
Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 2
Dequeued Professor Data:
ID: 102
Name: Alice
Branch: Electrical Engineering
Specialization: Power Systems

Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 2
Queue is empty.

Professor Data Queue
1. Enqueue Professor Data
2. Dequeue Professor Data
0. Exit
Enter your choice: 0
Exiting...
```

7. Given an array of elements, construct a complete binary tree from this array in level order fashion. That is, elements from left in the array will be filled in the tree level wise starting from level 0.

```
#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));

   newNode->data = value;

   newNode->left = NULL;

   newNode->right = NULL;

   return newNode;

}


void levelOrder(struct Node** root, int arr[], int size,int index)

{ if(index<size)

   {

*root=createNode(arr[index]);

levelOrder(&((*root)->left),arr,size,2*index+1);

levelOrder(&((*root)->right),arr,size,2*index+2);
```

```
    }
}

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

int main() {
    int arr = {1, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);

    struct Node* root = NULL;

    levelOrder(&root, arr, size,0);

    printf("Level Order Traversal of the constructed tree: ");
traverse(root);
    return 0;}
```

OUTPUT:



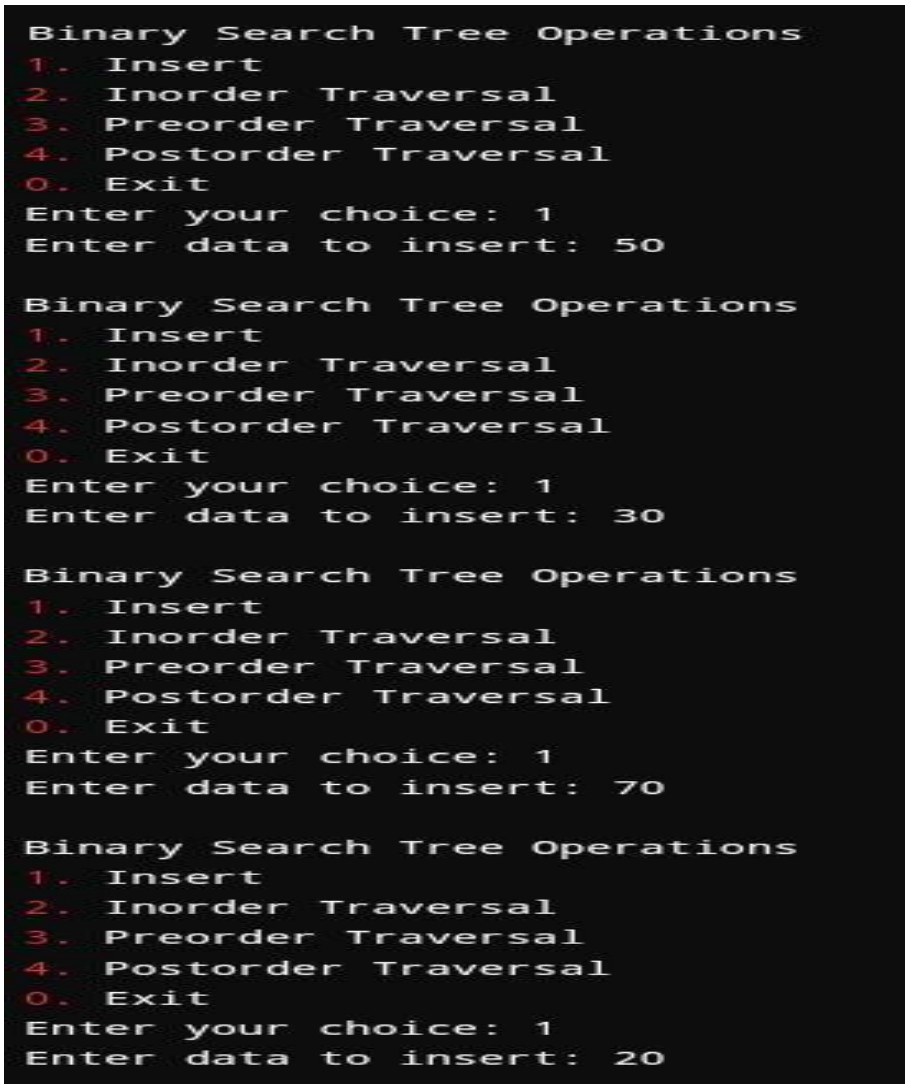
```
Level Order Traversal of the constructed tree: 1 2 4 5 3 6
```

8. Design, Develop and Implement a menu driven Program in C for the following operations on BinarSearch Tree (BST) of Integers
a. Create a BST of N Integers
b. Traverse the BST in Inorder, Preorder and Post Order

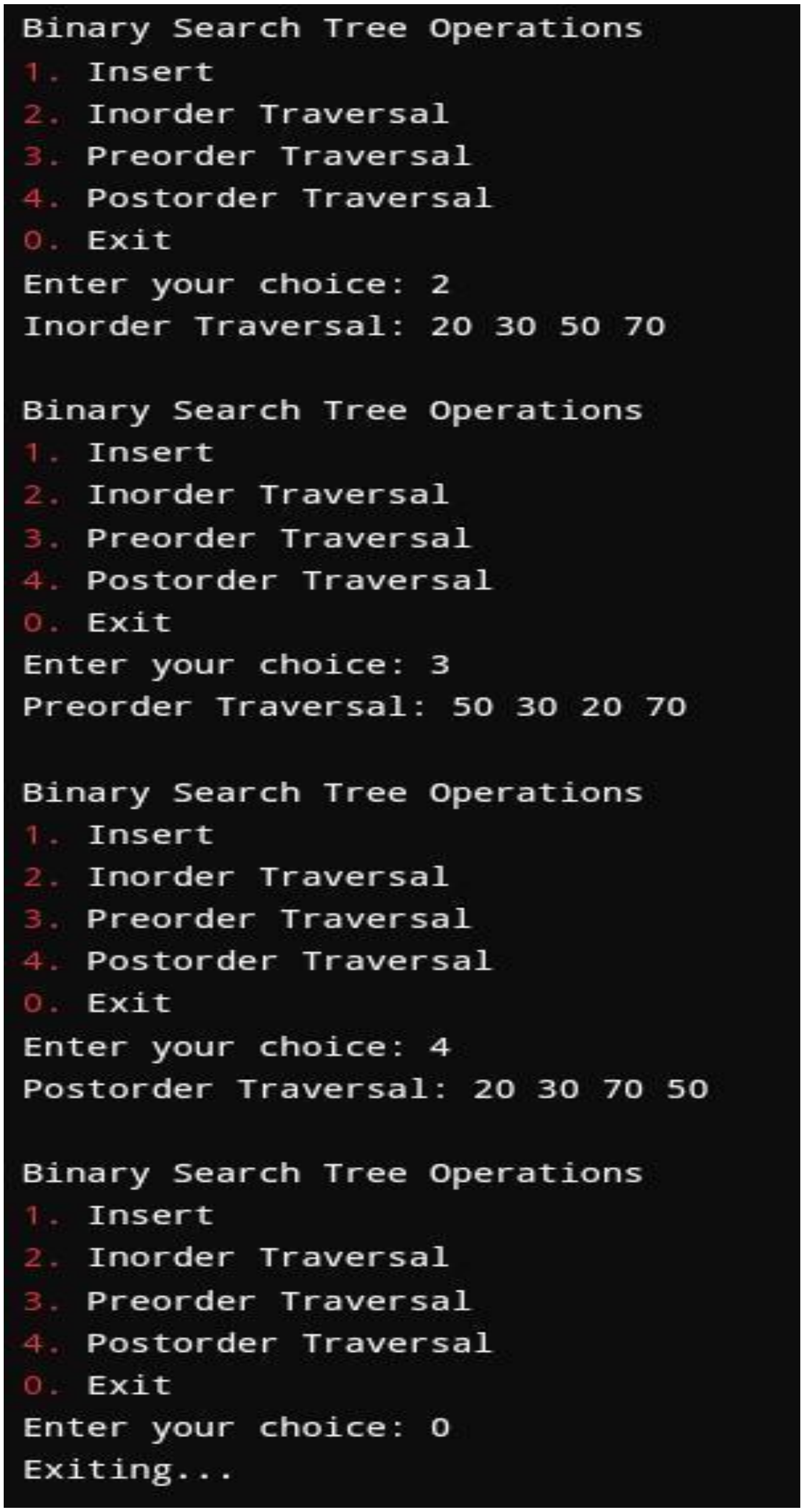
```
#include <stdio.h>
#include <stdlib.h>

// Define the structure of a node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

// Function to perform inorder traversal of the BST
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to perform preorder traversal of the BST
```

```c
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function to perform postorder traversal of the BST
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, data;

    do {
        printf("\nBinary Search Tree Operations\n");
        printf("1. Insert\n");
        printf("2. Inorder Traversal\n");
        printf("3. Preorder Traversal\n");
        printf("4. Postorder Traversal\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 0);

    return 0;}
```

OUTPUT:



```
Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 1
Enter data to insert: 50

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 1
Enter data to insert: 30

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 1
Enter data to insert: 70

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 1
Enter data to insert: 20
```



```
Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 2
Inorder Traversal: 20 30 50 70

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 3
Preorder Traversal: 50 30 20 70

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 4
Postorder Traversal: 20 30 70 50

Binary Search Tree Operations
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
0. Exit
Enter your choice: 0
Exiting...
```

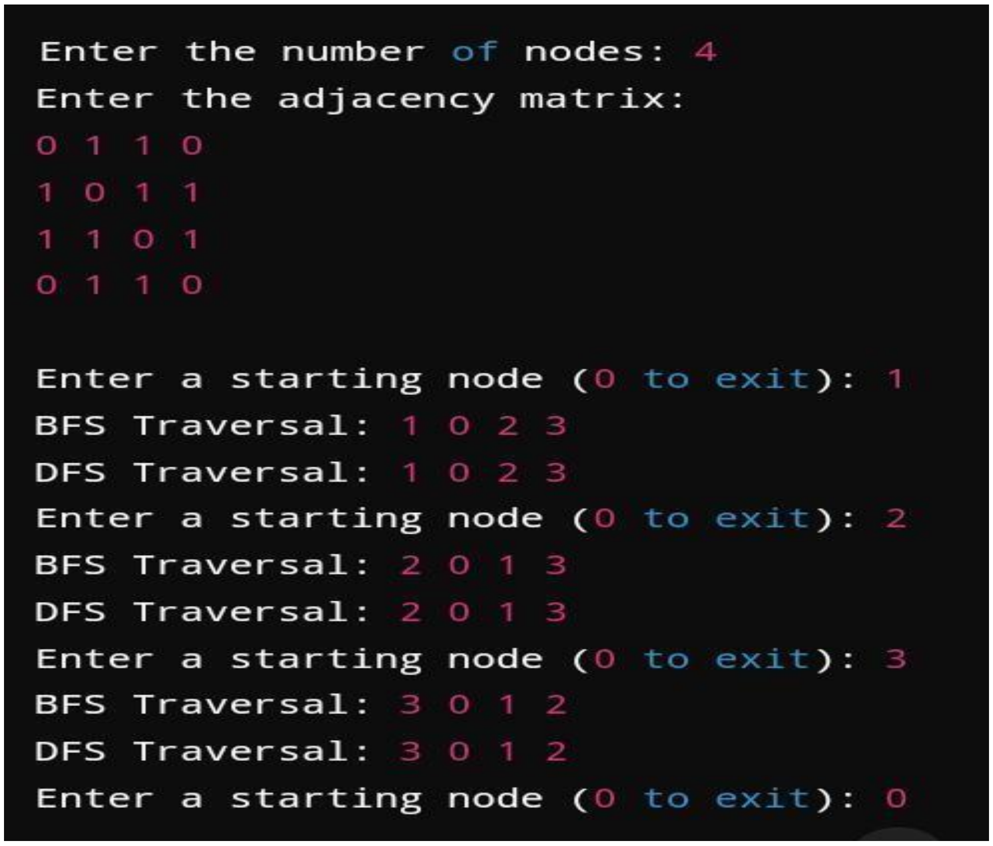
9. Design, Develop and implement a program in C for the following operations on Graph (G) of cities

a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a diagraph using DFS/BFS method.

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_NODES 100


// Function to initialize the adjacency matrix

void initializeGraph(int adjMatrix[][MAX_NODES], int numNodes) {

    int i, j;

    for (i = 0; i < numNodes; i++) {

        for (j = 0; j < numNodes; j++) {

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

        }

    }

}


// Depth-First Search (DFS) algorithm

void DFS(int adjMatrix[][MAX_NODES], int visited[], int currentNode, int
numNodes) {

    printf("%d ", currentNode);

    visited[currentNode] = 1;

    int i;
    for (i = 0; i < numNodes; i++) {
        if (adjMatrix[currentNode][i] && !visited[i]) {
            DFS(adjMatrix, visited, i, numNodes);
        }
    }
}

// Breadth-First Search (BFS) algorithm
void BFS(int adjMatrix[][MAX_NODES], int visited[], int startNode, int
numNodes) {
    int queue[MAX_NODES];
    int front = 0, rear = 0;
    printf("%d ", startNode);
    visited[startNode] = 1;
    queue[rear++] = startNode;
    while (front < rear) {
        int currentNode = queue[front++];
        int i;
        for (i = 0; i < numNodes; i++) {
            if (adjMatrix[currentNode][i] && !visited[i]) {
                printf("%d ", i);
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
```

```
}

int main() {
    int adjMatrix[MAX_NODES][MAX_NODES];
    int numNodes, startNode;

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

    printf("Enter the adjacency matrix:\n");
    initializeGraph(adjMatrix, numNodes);

    int choice;
    do {
        printf("\nEnter a starting node (0 to exit): ");
        scanf("%d", &startNode);

        if (startNode == 0)
            break;

        if (startNode >= numNodes) {
            printf("Invalid starting node. Please enter a valid node.\n");
            continue;
        }
        int visited[MAX_NODES] = {0};
        printf("BFS Traversal: ");
        BFS(adjMatrix, visited, startNode, numNodes);
        printf("\n");
        // Reset visited array for DFS
        for (int i = 0; i < numNodes; i++) {
            visited[i] = 0;
        }        printf("DFS Traversal: ");
        DFS(adjMatrix, visited, startNode, numNodes);
        printf("\n");
    } while (1);
    return 0;}
```

OUTPUT:



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

Enter a starting node (0 to exit): 1
BFS Traversal: 1 0 2 3
DFS Traversal: 1 0 2 3
Enter a starting node (0 to exit): 2
BFS Traversal: 2 0 1 3
DFS Traversal: 2 0 1 3
Enter a starting node (0 to exit): 3
BFS Traversal: 3 0 1 2
DFS Traversal: 3 0 1 2
Enter a starting node (0 to exit): 0
```

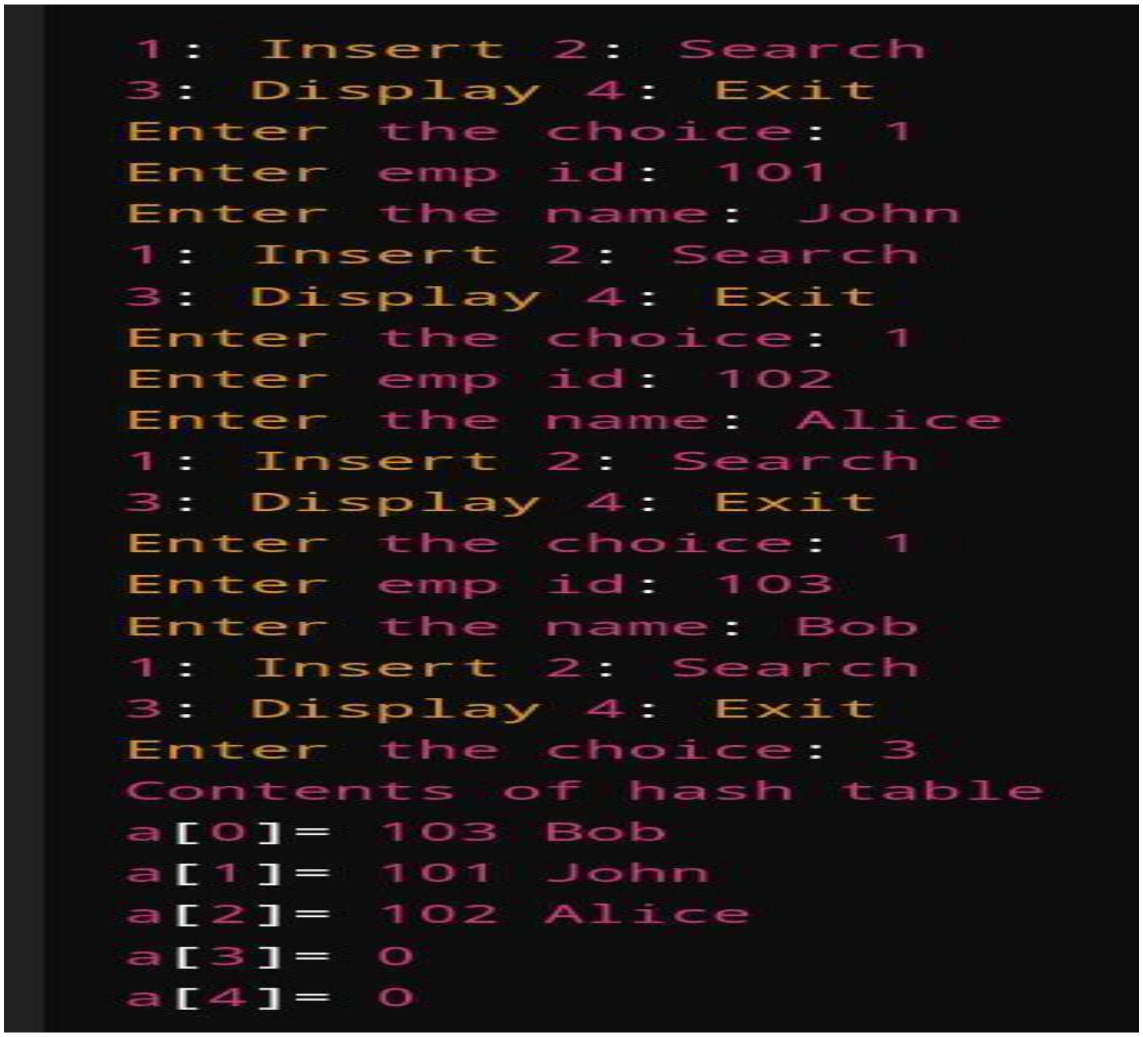
10. Design and develop a program in C that uses Hash Function H:K->L as H(K)=K mod m(reminder method) and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include  <stdio.h>

#include <stdlib.h>

#include <string.h>


#define HASH_SIZE 5


typedef struct employee {

   int id;

   char name[20];

} EMPLOYEE;


void initialize_hash_table(EMPLOYEE a[]) {

   int i;

   for (i = 0; i < HASH_SIZE; i++) {

      a[i].id = 0;

   }

}


int H(int k) {

   return k % HASH_SIZE;

}
```

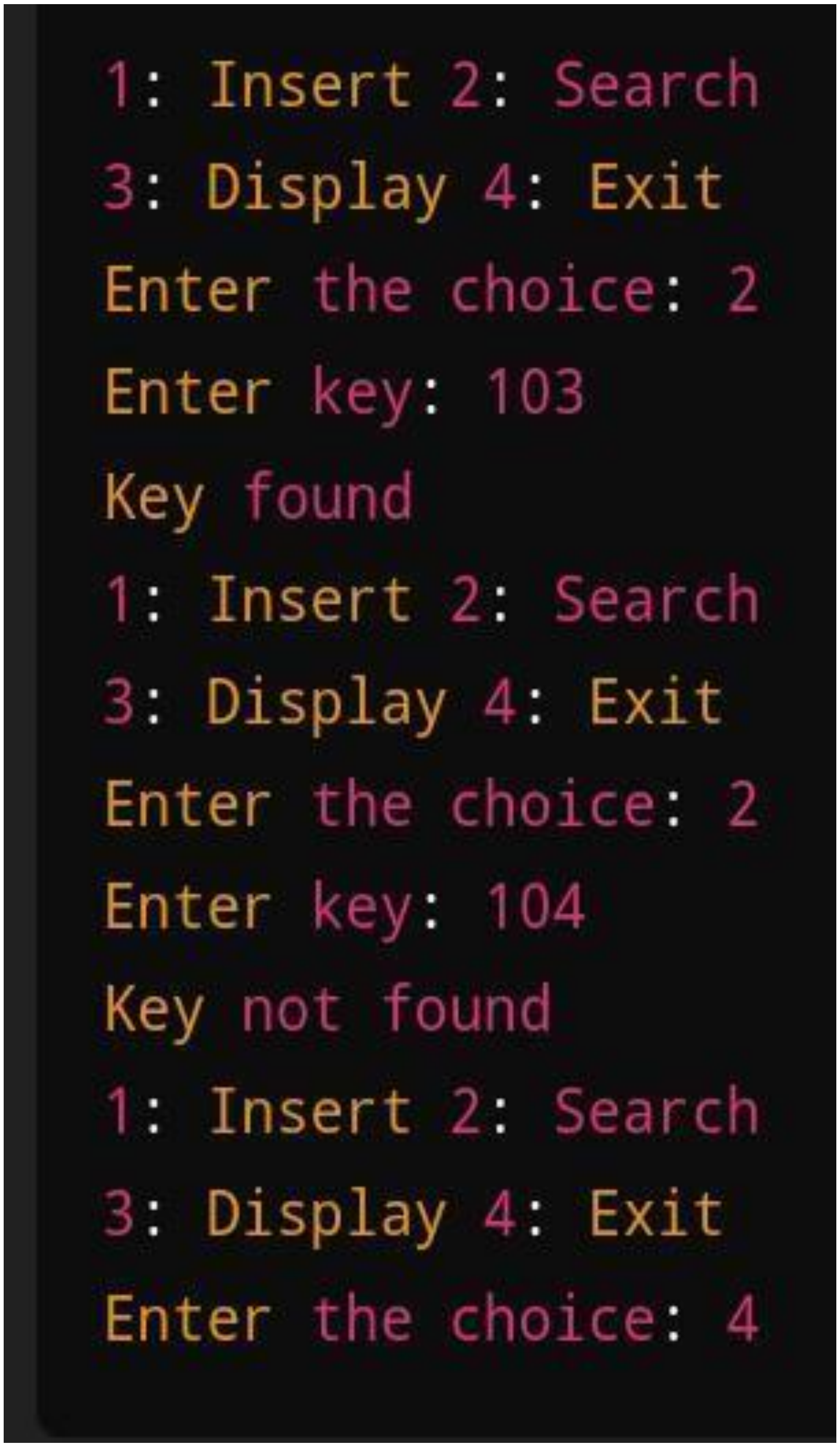
```
void insert_hash_table(int id, char name[], EMPLOYEE a[]) {
    int i, index, h_value;
    h_value = H(id);

    for (i = 0; i < HASH_SIZE; i++) {
        index = (h_value + i) % HASH_SIZE;
        if (a[index].id == 0) {
            a[index].id = id;
            strcpy(a[index].name, name);
            break;
        }
    }

    if (i == HASH_SIZE)
        printf("Hash table full\n");
}

int search_hash_table(int key, EMPLOYEE a[]) {
    int i, index, h_value;
    h_value = H(key);

    for (i = 0; i < HASH_SIZE; i++) {
        index = (h_value + i) % HASH_SIZE;
        if (key == a[index].id)
            return 1;
        if (a[index].id == 0)
            return 0;
    }

    if (i == HASH_SIZE)
        return 0;
}

void display_hash_table(EMPLOYEE a[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("a[%d]= %d %s\n", i, a[i].id, a[i].name);
    }
}

int main() {
    EMPLOYEE a[10];
    char name[20];
    int key, id, choice, flag;

    initialize_hash_table(a);

    for (;;) {
        printf("1: Insert 2: Search\n");
        printf("3: Display 4: Exit\n");
        printf("Enter the choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter emp id: ");
                scanf("%d", &id);
                printf("Enter the name: ");
                scanf("%s", name);
                insert_hash_table(id, name, a);
                break;

            case 2:
                printf("Enter key: ");
                scanf("%d", &key);
                flag = search_hash_table(key, a);
                if (flag == 0)
                    printf("Key not found\n");
                else
                    printf("Key found\n");
                break;

            case 3:
                printf("Contents of hash table\n");
                display_hash_table(a, HASH_SIZE);
```

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

            default:
                exit(0);
        }
    }
    return 0;
}
```

OUTPUT:



```
 1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 1
Enter emp id: 101
Enter the name: John
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 1
Enter emp id: 102
Enter the name: Alice
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 1
Enter emp id: 103
Enter the name: Bob
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 3
Contents of hash table
a[0]= 103 Bob
a[1]= 101 John
a[2]= 102 Alice
a[3]= 0
a[4]= 0
```



```
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 2
Enter key: 103
Key found
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 2
Enter key: 104
Key not found
1: Insert 2: Search
3: Display 4: Exit
Enter the choice: 4
```