Lab Assignment-14

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 30/11/21

QUES 1: [1] Write a menu driven program to create a GRAPH ADT and traverse it in breadth-first-search and depth-first-search using adjacency matrix.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
#define NULLpt -999
typedef int Node;
typedef struct Stack
{
    Node data;
    struct Stack *link;
} Stack;
typedef struct QueNode
{
    int data;
    struct QueNode *link;
} QueNode;
typedef struct Queue
{
    QueNode *front;
    QueNode *rear;
} Queue;
typedef struct Graph
{
    int **adjMatrix;
    int len;
} Graph;
int isEmpty(Queue *);
void enqueue(Queue *, int);
int dequeue(Queue *);
int peek(Queue *);
int isEmpty_stack(Stack *);
void push(Stack **, int);
int pop(Stack **);
int top(Stack *);
void initialise(Graph *, int);
void create_graph(Graph *);
void DFS(Graph *, int);
void BFS(Graph *, int);
void display(Graph *);
int main()
{
    int len;
    printf("Enter the number of nodes to work with: ");
    scanf("%d", &len);
```

```
    Graph graph;
    initialise(&graph, len);
    int choice, start;
    do
    {
        printf("\nMain Menu\n\n");
        printf("1) Create Graph\n2) Depth First Traversal\n");
        printf("3) Breadth First Traversal\n4) Display\n5) Exit\n->: ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
        case 1:
            create_graph(&graph);
            break;
        case 2:
            printf("Enter starting node: ");
            scanf("%d", &start);
            DFS(&graph, start);
            break;
        case 3:
            printf("Enter starting node: ");
            scanf("%d", &start);
            BFS(&graph, start);
            break;
        case 4:
            display(&graph);
            break;
        default:
            printf("Exiting...\n");
        }
        printf("---------------------------------\n");
    } while (choice >= 1 && choice <= 4);
    return 0;
}
void initialise(Graph *graph, int len)
{
    graph->len = len;
    graph->adjMatrix = (int **)calloc(len, sizeof(int *));
    for (int i = 0; i < len; i++)
        graph->adjMatrix[i] = (int *)calloc(len,
                                            sizeof(int));
}
void create_graph(Graph *graph)
{
    printf("--:Enter the nodes connected by adjMatrix edge:-- \n\n");
    char ch;
    int i, j;
    do
    {
```

```
        do
        {
            printf("Node from which the edge originates: ");
            scanf("%d", &i);
            if (i >= graph->len)
                printf("Invalid index entered!\n\n");
        } while (i >= graph->len);
        do
        {
            printf("Node at which the edge terminates: ");
            scanf("%d", &j);
            if (j >= graph->len)
                printf("Invalid index entered!\n\n");
        } while (j >= graph->len);
        graph->adjMatrix[i][j] = 1;
        printf("\nEnter \'Y\' to further create graph: ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
}
void DFS(Graph *graph, int start)
{
    if (start >= graph->len)
    {
        printf("Invalid Location!\n");
        return;
    }
    int *visited = (int *)calloc(graph->len, sizeof(int));
    int i;
    printf("%d->", start);
    visited[start] = 1;
    Stack *stack = NULL;
    push(&stack, start);
    while (!isEmpty_stack(stack))
    {
        start = top(stack);
        for (i = 0; i < graph->len; i++)
        {
            if (graph->adjMatrix[start][i] == 1 && visited[i] == 0)
            {
                push(&stack, i);
                printf("%d->", i);
                visited[i] = 1;
                break;
            }
        }
        if (i == graph->len)
            pop(&stack);
    }
    free(visited);
    printf("\b\b \n");
```

```c
}
void BFS(Graph *graph, int start)
{
    if (start >= graph->len)
    {
        printf("Invalid Location!\n");
        return;
    }
    int *visited = (int *)calloc(graph->len, sizeof(int));
    Queue queue = {NULL, NULL};
    enqueue(&queue, start);
    visited[start] = 1;
    while (!isEmpty(&queue))
    {
        start = dequeue(&queue);
        printf("%d->", start);
        for (int i = 0; i < graph->len; i++)
        {
            if (graph->adjMatrix[start][i] == 1 && visited[i] == 0)
            {
                enqueue(&queue, i);
                visited[i] = 1;
            }
        }
    }
    free(visited);
    printf("\b\b \n");
}
void display(Graph *graph)
{
    printf(" ");
    for (int i = 0; i < graph->len; i++)
        printf("%d ", i);
    printf("\n ");
    for (int i = 0; i < graph->len; i++)
        printf("- ");
    printf("\n");
    for (int i = 0; i < graph->len; i++)
    {
        printf("%d | ", i);
        for (int j = 0; j < graph->len; j++)
        {
            printf("%d ", graph->adjMatrix[i][j]);
        }
        printf("\n");
    }
}
int isEmpty(Queue *que)
{
    if (que->front == NULL)
```

```
        return 1;
    return 0;
}
void enqueue(Queue *que, int data)
{
    QueNode *temp = (QueNode *)malloc(sizeof(QueNode));
    temp->data = data;
    temp->link = NULL;

    if (isEmpty(que))
    {
        que->front = que->rear = temp;
        return;
    }

    que->rear->link = temp;
    que->rear = que->rear->link;
}
int dequeue(Queue *que)
{
    if (isEmpty(que))
        return NULLpt;

    QueNode *temp = que->front;
    que->front = que->front->link;

    if (que->front == NULL)
        que->rear = NULL;

    int n = temp->data;
    free(temp);
    return n;
}
int peek(Queue *que)
{
    if (isEmpty(que))
        return NULLpt;
    return que->front->data;
}
int isEmpty_stack(Stack *stack)
{
    if (!stack)
        return 1;
    return 0;
}
void push(Stack **stack, int data)
{
    Stack *temp = (Stack *)malloc(sizeof(Stack));
    temp->data = data;
    temp->link = *stack;
```

```
    *stack = temp;
}
int pop(Stack **stack)
{
    if (isEmpty_stack(*stack))
    {
        printf("\nUnderflow!");
        return -999;
    }

    Stack *temp = (*stack);
    *stack = (*stack)->link;

    int val = temp->data;
    free(temp);
    return val;
}
int top(Stack *stack)
{
    return stack->data;
}
```

OUTPUT:

```
Enter the number of nodes to work with: 7
Main Menu

1) Create Graph2) Depth First Traversal
3) Breadth First Traversal
4) Display
5) Exit->: 1

--:Enter the nodes connected by adjMatrix edge:--
Node from which the edge originates: 1
Node at which the edge terminates: 2

Enter 'Y' to further create graph: y
Node from which the edge originates: 1
Node at which the edge terminates: 3

Enter 'Y' to further create graph: y
Node from which the edge originates: 4Node at which the edge terminates: 1

Enter 'Y' to further create graph: y
Node from which the edge originates: 3
Node at which the edge terminates: 6

Enter 'Y' to further create graph: yNode from which the edge originates: 2
Node at which the edge terminates: 5
```

```
Enter 'Y' to further create graph: y
Node from which the edge originates: 2
Node at which the edge terminates: 6

Enter 'Y' to further create graph: y
Node from which the edge originates: 5
Node at which the edge terminates: 4

Enter 'Y' to further create graph: n
---------------------------------

Main Menu

1) Create Graph
2) Depth First Traversal
3) Breadth First Traversal
4) Display
5) Exit
->: 2

Enter starting node: 2
2->5->4->1->3->6 >
---------------------------------

Main Menu

1) Create Graph
2) Depth First Traversal
3) Breadth First Traversal
4) Display
5) Exit
->: 3

Enter starting node: 3
3->6 >
---------------------------------

Main Menu

1) Create Graph
2) Depth First Traversal
3) Breadth First Traversal
4) Display
5) Exit
->: 4

 0 1 2 3 4 5 6
 - - - - - - -
0 | 0 0 0 0 0 0 0
1 | 0 0 1 1 0 0 0
```

```
2 | 0 0 0 0 0 1 1
3 | 0 0 0 0 0 0 1
4 | 0 1 0 0 0 0 0
5 | 0 0 0 0 1 0 0
6 | 0 0 0 0 0 0 0
---------------------------------

Main Menu

1) Create Graph
2) Depth First Traversal
3) Breadth First Traversal
4) Display
5) Exit
->: 5

Exiting...
---------------------------------
```

[2] Write a menu driven program to create a GRAPH ADT and traverse it in breadth-first-search and depth-first-search using linked list.

SOLUTION:

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

struct Edge;
typedef struct Node
{
    int data;
    int visited_status;
    struct Edge *edges;
    struct Node *link;
} Node;
typedef struct Node Node;
typedef struct Stack
{
    Node *data;
    struct Stack *link;
} Stack;
typedef struct QueNode
{
    Node *data;
    struct QueNode *link;
} QueNode;
typedef struct Queue
{
    QueNode *front;
    QueNode *rear;
```

```c
} Queue;
typedef struct Edge
{
    struct Node *dest;
    struct Edge *link;
} Edge;
typedef struct Graph
{
    int size;
    struct Node *start;
} Graph;
int isEmpty(Queue *);
void enqueue(Queue *, Node *);
Node *dequeue(Queue *);
Node *peek(Queue *);
int isEmpty_stack(Stack *);
void push(Stack **, Node *);
Node *pop(Stack **);
void add_node(Graph *);
void insert_edges(Graph *, int, int);
void BFS(Graph *, int);
void DFS(Graph *, int);
void display(Graph);
int main()
{
    printf("Note: The graphs are directed and 1 based indexed!\n");
    printf("There can be more than one edge between a source and a destination.\n\n");
    Graph graph = {0, NULL};
    int choice;
    unsigned int vartex_1, vartex_2;
    do
    {
        printf("Current amount of nodes in graph: (%d)\n", graph.size);
        printf("1) Add a Node\n2) Add an edge\n3) Depth First Traversal\n");
        printf("4) Breadth First Traversal\n5) Display raw structure\n6) Exit\n->: ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
        case 1:
            add_node(&graph);
            break;
        case 2:
            printf("\nEnter the pair of vertices ");
            printf("(To and From, space separated): ");
            scanf("%d%d", &vartex_1, &vartex_2);
            insert_edges(&graph, vartex_1, vartex_2);
            break;
        case 3:
            printf("Enter a starting node: ");
```

```
            scanf("%d", &vartex_1);
            printf("BFS: ");
            BFS(&graph, vartex_1);
            break;
        case 4:
            printf("Enter a starting node: ");
            scanf("%d", &vartex_1);
            printf("DFS: ");
            DFS(&graph, vartex_1);
            break;
        case 5:
            display(graph);
        default:
            printf("Exiting...\n");
        }
        printf("---------------------------------\n");
    } while (choice >= 1 && choice <= 5);
    return 0;
}
void createGraph(Graph *graph, int nodes)
{
    int count = 1;
    Node *follow = NULL;
    graph->size = nodes;
    while (nodes--)
    {
        Node *temp = (Node *)malloc(sizeof(Node));
        temp->data = count++;
        temp->edges = NULL;
        temp->link = NULL;
        if (!follow)
        {
            graph->start = follow = temp;
            continue;
        }
        follow->link = temp;
        follow = temp;
    }
}
void add_node(Graph *graph)
{
    Node *lastNode = graph->start;
    if (!lastNode)
    {
        graph->size++;
        graph->start = (Node *)malloc(sizeof(Node));
        graph->start->data = graph->size;
        graph->start->edges = NULL;
        graph->start->link = NULL;
        return;
```

```
    }
    while (lastNode->link)
        lastNode = lastNode->link;
    lastNode->link = (Node *)malloc(sizeof(Node));
    lastNode->link->data = ++graph->size;
    lastNode->link->edges = NULL;
    lastNode->link->link = NULL;
}
void insert_edges_util(Graph *graph, Node **node, int vertex_2)
{
    if (vertex_2 < 1 || vertex_2 > graph->size)
        return;
    Node *start = graph->start;
    while (start)
    {
        if (start->data == vertex_2)
        {
            Node *temp_node = *node;
            Edge *temp = (Edge *)malloc(sizeof(Edge));
            temp->dest = start;
            temp->link = NULL;
            if (!temp_node->edges)
            {
                temp_node->edges = temp;
                return;
            }
            while (temp_node->edges->link)
                temp_node->edges = temp_node->edges->link;
            temp_node->edges->link = temp;
            return;
        }
        start = start->link;
    }
}
void insert_edges(Graph *graph, int vertex_1, int vertex_2)
{
    if (vertex_1 < 1 || vertex_1 > graph->size)
        return;
    else if (vertex_2 < 1 || vertex_2 > graph->size)
        return;
    Node *start = graph->start;
    while (start)
    {
        if (start->data == vertex_1)
        {
            insert_edges_util(graph, &start, vertex_2);
            return;
        }
        start = start->link;
    }
```

```
}
void BFS(Graph *graph, int val_start)
{
    if (!graph->start)
        return;
    Node *temp = graph->start;
    Node *start = NULL;
    while (temp)
    {
        if (val_start == temp->data)
            start = temp;
        temp->visited_status = 0;
        temp = temp->link;
    }
    Queue queue = {NULL, NULL};
    enqueue(&queue, start);
    start->visited_status = 1;
    while (!isEmpty(&queue))
    {
        temp = dequeue(&queue);
        printf("%d->", temp->data);
        Edge *temp_edges = temp->edges;
        while (temp_edges)
        {
            if (temp_edges->dest->visited_status == 0)
            {
                enqueue(&queue, temp_edges->dest);
                temp_edges->dest->visited_status = 1;
            }
            temp_edges = temp_edges->link;
        }
    }
    printf("\b\b \n");
}
void DFS(Graph *graph, int val_start)
{
    if (!graph->start)
        return;
    Node *temp = graph->start;
    Node *start = NULL;
    while (temp)
    {
        if (val_start == temp->data)
            start = temp;
        temp->visited_status = 0;
        temp = temp->link;
    }
    Stack *stack = NULL;
    push(&stack, start);
    start->visited_status = 1;
```

```
    while (!isEmpty_stack(stack))
    {
        temp = pop(&stack);
        printf("%d->", temp->data);
        Edge *temp_edges = temp->edges;
        while (temp_edges)
        {
            if (temp_edges->dest->visited_status == 0)
            {
                push(&stack, temp_edges->dest);
                temp_edges->dest->visited_status = 1;
            }
            temp_edges = temp_edges->link;
        }
    }
    printf("\b\b \n");
}
void display(Graph graph)
{
    while (graph.start)
    {
        printf("%d-> ", graph.start->data);
        Edge *edge = graph.start->edges;
        while (edge)
        {
            printf("%d, ", edge->dest->data);
            edge = edge->link;
        }
        printf("\b\b \n");
        graph.start = graph.start->link;
    }
    printf("\b\b\b \n");
}
int isEmpty_stack(Stack *stack)
{
    if (!stack)
        return 1;
    return 0;
}

void push(Stack **stack, Node *data)
{
    Stack *temp = (Stack *)malloc(sizeof(Stack));
    temp->data = data;
    temp->link = *stack;
    *stack = temp;
}

Node *pop(Stack **stack)
{
```

```
    if (isEmpty_stack(*stack))
    {
        printf("\nUnderflow!");
        return NULL;
    }

    Stack *temp = (*stack);
    *stack = (*stack)->link;

    Node *val = temp->data;
    free(temp);
    return val;
}
int isEmpty(Queue *que)
{
    if (que->front == NULL)
        return 1;
    return 0;
}

void enqueue(Queue *que, Node *data)
{
    QueNode *temp = (QueNode *)malloc(sizeof(QueNode));
    temp->data = data;
    temp->link = NULL;

    if (isEmpty(que))
    {
        que->front = que->rear = temp;
        return;
    }

    que->rear->link = temp;
    que->rear = que->rear->link;
}
Node *dequeue(Queue *que)
{
    if (isEmpty(que))
        return NULL;

    QueNode *temp = que->front;
    que->front = que->front->link;

    if (que->front == NULL)
        que->rear = NULL;

    Node *n = temp->data;
    free(temp);
    return n;
}
```

```
Node *peek(Queue *que)
{
    if (isEmpty(que))
        return NULL;
    return que->front->data;
}
```

OUTPUT:

```
Note: The graphs are directed and 1 based indexed!
There can be more than one edge between a source and a destination.

Current amount of nodes in graph: (0)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 1

---------------------------------
Current amount of nodes in graph: (1)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 1

---------------------------------
Current amount of nodes in graph: (2)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 1

---------------------------------
Current amount of nodes in graph: (3)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
```

```
->: 1

---------------------------------
Current amount of nodes in graph: (4)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 1

---------------------------------
Current amount of nodes in graph: (5)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 1

---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 1 3
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 1 2
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
```

```
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 4 1
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 2 5
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 2 6
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 3 6
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
```

```
6) Exit
->: 2


Enter the pair of vertices (To and From, space separated): 5 4
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 3

Enter a starting node: 1
BFS: 1->3->2->6->5->4 >
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 4

Enter a starting node: 1
DFS: 1->2->6->5->4->3 >
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 5

1-> 3, 2
2-> 5, 6
3-> 6
4-> 1
5-> 4
6-

Exiting...
---------------------------------
Current amount of nodes in graph: (6)
1) Add a Node
```

```
2) Add an edge
3) Depth First Traversal
4) Breadth First Traversal
5) Display raw structure
6) Exit
->: 6

Exiting...
---------------------------------
```

[3] Write a menu driven program to implement Linear Probing in hashing.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
#define hash(x) (x % 10)
int calc_index(int *, int);
int search_index(int *, int);
int delete (int *, int);
int main()
{
    int *hashTable = (int *)calloc(MAX, sizeof(int));
    int choice, val, index;
    do
    {
        printf("1) Insert Data\n2) Search for Data\n3) Display Table\n");
        printf("4) Delete Item\n5) Exit\n->: ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value: ");
            scanf("%d", &val);
            index = calc_index(hashTable, val);
            if (index == -1)
            {
                printf("\nThe hash table is full!\n");
                break;
            }
            else if (index == -2)
            {
                printf("\nYou cannot enter duplicate values!\n");
                break;
            }
            hashTable[index] = val;
            break;
        case 2:
```

```
            printf("Enter the value to look for: ");
            scanf("%d", &val);
            index = search_index(hashTable, val);
            if (index == -1)
            {
                printf("\nItem NOT found!\n");
                break;
            }
            printf("\nItem FOUND at index %d!\n", index);
            break;
        case 3:
            for (int i = 0; i < MAX; i++)
                printf("%d ", hashTable[i]);
            printf("\n");
            break;
        case 4:
            printf("Enter the item to delete: ");
            scanf("%d", &val);
            if (delete (hashTable, val))
                printf("\nItem deleted!\n");
            else
                printf("\nItem NOT found!\n");
            break;
        default:
            printf("Exiting...\n");
        }
        printf("---------------------------------\n");
    } while (choice >= 1 && choice <= 4);
    free(hashTable);
    return 0;
}
int calc_index(int *hashTable, int val)
{
    int offset = 0;
    int valueAt = hashTable[(hash(val) + offset) % MAX];
    while ((valueAt != 0 && valueAt != -1) && offset < MAX)
    {
        if (valueAt == val)
            return -2;
        offset++;
        valueAt = hashTable[(hash(val) + offset) % MAX];
    }
    if (offset >= MAX)
        return -1;
    return (hash(val) + offset) % MAX;
}
int search_index(int *hashTable, int val)
{
    int offset = 0;
    int valueAt = hashTable[(hash(val) + offset) % MAX];
```

```
    while (valueAt != 0 && offset < MAX)
    {
        if (valueAt == val)
            return (hash(val) + offset) % MAX;
        offset++;
        valueAt = hashTable[(hash(val) + offset) % MAX];
    }
    return -1;
}
int delete (int *hashTable, int val)
{
    int offset = 0;
    int valueAt = hashTable[(hash(val) + offset) % MAX];
    while (offset < MAX && valueAt != 0)
    {
        if (valueAt == val)
        {
            hashTable[(hash(val) + offset) % MAX] = -1;
            return 1;
        }
        offset++;
        valueAt = hashTable[(hash(val) + offset) % MAX];
    }
    return 0;
}
```

OUTPUT:

```
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 234
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 234

You cannot enter duplicate values!
---------------------------------
1) Insert Data
2) Search for Data
```

```
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 23423
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 312
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 233
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 34
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 32
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1
```

```
Enter value: 4
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 32

You cannot enter duplicate values!
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 55
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 5
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 3

5 0 312 23423 234 233 34 32 4 55
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 2

Enter the value to look for: 34

Item FOUND at index 6!
```

```
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 2
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 1

The hash table is full!
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 3

5 2 312 23423 234 233 34 32 4 55
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 4

Enter the item to delete: 34

Item deleted!
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 5

Exiting...
---------------------------------
```

[4] Write a menu driven program to implement chaining in hashing.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
#define hash(x) (x % 10)
typedef struct Node
{
    int data;
    struct Node *link;
} Node;
void initialise(Node *[]);
void insert(Node **, int);
int search(Node *, int);
void delete (Node **, int);
void display(Node *[]);
int main()
{
    Node *arr[10];
    initialise(arr);
    int choice, val;
    do
    {
        printf("1) Insert Data\n2) Search for Data\n");
        printf("3) Display Table\n4) Delete Item\n5) Exit\n->: ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
        case 1:
            printf("Enter value: ");
            scanf("%d", &val);
            insert(&arr[hash(val)], val);
            break;
        case 2:
            printf("Enter the value to look for: ");
            scanf("%d", &val);
            if (search(arr[hash(val)], val))
                printf("\nItem Found!\n");
            else
                printf("\nItem NOT found!\n");
            break;
        case 3:
            display(arr);
            break;
        case 4:
            printf("Enter the Item to delete: ");
            scanf("%d", &val);
            delete (&arr[hash(val)], val);
            break;
```

```
        default:
            printf("Exiting...\n");
        }
        printf("-----------------------------------\n");
    } while (choice >= 1 && choice <= 4);
    return 0;
}
void initialise(Node *arr[])
{
    for (int i = 0; i < 10; i++)
        arr[i] = NULL;
}
void insert(Node **start, int val)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = val;
    temp->link = NULL;
    if (!*start)
    {
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    while (tempStart->link)
        tempStart = tempStart->link;
    tempStart->link = temp;
}
int search(Node *start, int val)
{
    while (start)
    {
        if (start->data == val)
            return 1;
        start = start->link;
    }
    return 0;
}
void delete (Node **start, int val)
{
    if (!*start)
    {
        printf("The list is empty!\n");
        return;
    }
    else if ((*start)->data == val)
    {
        Node *ptr = *start;
        *start = (*start)->link;
        free(ptr);
        return;
```

```
    }
    Node *tempStart = *start;
    Node *tempPrev = *start;
    while (tempStart && tempStart->data != val)
    {
        tempPrev = tempStart;
        tempStart = tempStart->link;
    }
    if (!tempStart)
    {
        printf("\nItem not found!\n");
        return;
    }
    tempPrev->link = tempStart->link;
    free(tempStart);
}
void display(Node *arr[])
{
    Node *start;
    for (int i = 0; i < 10; i++)
    {
        start = arr[i];
        printf("%d | ", i);
        while (start)
        {
            printf("%d ", start->data);
            start = start->link;
        }
        printf("\n");
    }
}
```

OUTPUT:

```
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 33423
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1
```

```
Enter value: 22
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 543345
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 3242
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 234
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 45534
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 23457678
---------------------------------
1) Insert Data
2) Search for Data
```

```
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 5664
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 4564
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 5645
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 9798
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 1

Enter value: 0980
----------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 2
```

```
Enter the value to look for: 123

Item NOT found!
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 2

Enter the value to look for: 980

Item Found!
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 3

0 | 980
1 |
2 | 22 3242
3 | 33423
4 | 234 45534 5664 4564
5 | 543345 5645
6 |
7 |
8 | 23457678 9798
9 |
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 4

Enter the Item to delete: 22
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 3

0 | 980
```

```
1 |
2 | 3242
3 | 33423
4 | 234 45534 5664 4564
5 | 543345 5645
6 |
7 |
8 | 23457678 9798
9 |
---------------------------------
1) Insert Data
2) Search for Data
3) Display Table
4) Delete Item
5) Exit
->: 5

Exiting...
---------------------------------
```