Britannia University
Lab Final Report
Sub: CSE 132L- Data Structure Lab

Submitted To

Mozammel Hoque
Chairman & exam controller
Department of CSE

Submitted by

Najmol Hasan
Sum-21
ID: 2102020002
Department of CSE

📋 TABLE OF CONTENTS

1. Array Operations
2. Linear Search
3. Bubble Sort
4. Stack Implementation
5. Queue Implementation
6. Linked List Traversal
7. Insert at Beginning
8. Binary Search
9. Factorial (Recursion)
10. Matrix Addition

1. ARRAY OPERATIONS

Description: Print elements of a fixed array

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   int i;
   for(i = 0; i < 5; i++) {
      printf("%d ", arr[i]);
   }
   return 0;
}
```

Output: 10 20 30 40 50

2. LINEAR SEARCH

Description: Find an element in an array

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int search = 30, i, found = 0;
    for(i = 0; i < 5; i++) {
        if(arr[i] == search) {
            printf("Found at position %d\n", i + 1);
            found = 1;
            break;
        }
    }
    if(found == 0) {
        printf("Not found!\n");
    }
    return 0;
}
```

Output: Found at position 3

3. BUBBLE SORT

Description: Sort array in ascending order

```
#include <stdio.h>
int main() {
    int arr[5] = {50, 40, 30, 20, 10};
    int i, j, temp;

    // Bubble Sort
    for(i = 0; i < 4; i++) {
```

```
        for(j = 0; j < 4 - i; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    printf("Sorted: ");
    for(i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output: Sorted: 10 20 30 40 50

4. STACK IMPLEMENTATION

Description: Basic push and pop operations

```
#include <stdio.h>
int stack[5], top = -1;
void push(int x) {
    stack[++top] = x;
    printf("Pushed: %d\n", x);
}
void pop() {
    printf("Popped: %d\n", stack[top--]);
```

```
}
int main() {
    push(10);
    push(20);
    push(30);
    pop();
    pop();
    return 0;
}
```

Output:

text

```
Pushed: 10
Pushed: 20
Pushed: 30
Popped: 30
Popped: 20
```

5. QUEUE IMPLEMENTATION

Description: Basic enqueue and dequeue operations

```
#include <stdio.h>
int queue[5], front = 0, rear = -1;

void enqueue(int x) {
    queue[++rear] = x;
    printf("Enqueued: %d\n", x);
}
```

```
void dequeue() {
    printf("Dequeued: %d\n", queue[front++]);
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    dequeue();
    dequeue();
    return 0;
}
```

Output:

text

Enqueued: 10

Enqueued: 20

Enqueued: 30

Dequeued: 10

Dequeued: 20

6. LINKED LIST TRAVERSAL

Description: Create and display 3 nodes

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
```

```
};

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    struct Node* third = (struct Node*)malloc(sizeof(struct Node));

    // Assign data
    head->data = 10;
    second->data = 20;
    third->data = 30;

    // Connect nodes
    head->next = second;
    second->next = third;
    third->next = NULL;

    // Print
    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    return 0;
}
```

Output: 10 20 30

7. INSERT AT BEGINNING

Description: Insert new node at start of list

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
int main() {
    // Create first node
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 20;
    head->next = NULL;

    // Insert at beginning
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = 10;
    newNode->next = head;
    head = newNode;

    // Print
    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    return 0;
```

```
}
```

Output: 10 20

8. BINARY SEARCH

Description: Search in sorted array

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   int search = 30, low = 0, high = 4, mid, found = 0;
   while(low <= high) {
      mid = (low + high) / 2;

      if(arr[mid] == search) {
         printf("Found at index %d\n", mid);
         found = 1;
         break;
      }
      else if(arr[mid] < search) {
         low = mid + 1;
      }
      else {
         high = mid - 1;
      }
   }

   if(found == 0) {
      printf("Not found!\n");
```

```
    }
    return 0;
}
```

Output: Found at index 2

9. FACTORIAL (RECURSION)

Description: Calculate 5! using recursion

```
#include <stdio.h>
int fact(int n) {
    if(n <= 1) return 1;
    return n * fact(n - 1);
}
int main() {
    printf("5! = %d\n", fact(5));
    return 0;
}
```

Output: 5! = 120

10. MATRIX ADDITION

Description: Add two 2x2 matrices

```
#include <stdio.h>
int main() {
    int mat1[2][2] = {{1, 2}, {3, 4}};
    int mat2[2][2] = {{5, 6}, {7, 8}};
    int result[2][2];
    int i, j;
```

```
    // Add
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            result[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
    // Print
    printf("Result:\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

Result:

6 8

10 12