Data Structures and Algorithms

Mohammed Fahad

July 4, 2025

Syllabus

Basic Concepts of Data Structures

Definitions; Data Abstraction; Performance Analysis - Time & Space Complexity, Asymptotic Notations; Polynomial representation using Arrays, Sparse matrix (Tuple representation); Stacks and Queues - Stacks, Multi-Stacks, Queues, Circular Queues, Double Ended Queues; Evaluation of Expressions- Infix to Postfix, Evaluating Postfix Expressions.

1 Definitions

- **Data Structures:** ways of organizing and storing data in a computer so that it can be accessed and modified efficiently. Types:
  1. Linear: Arrays, Linked Lists, Stacks, Queues
  2. Non-linear: Trees, Graphs
- **Data Abstraction:** concept of hiding the internal details of how data is stored or maintained and only showing the essential features or operations that can be performed on the data.

2 Performance Analysis

1. **Time Complexity**
2. **Space Complexity**

3 Stack

It follow FILO (First In Last Out) scheme

- pop - Removes from top
- push - Adds to top
- peek/top - See topmost element

3.1 Implementation of Stack

```c
#include <stdio.h>
#define MAX 2

int stack[MAX];
int top = -1;

void push(int a) {
    if (top + 1 >= MAX) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = a;
    }
}

int pop() {
    if (top == -1) {
        printf("Stack underflow \n");
        return -1;
    } else {
        return stack[top--];
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty!\n");
    } else {
        for (int i = top; i > -1; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    pop();

    push(5);
    push(8);

    display();

    pop();

    display();

    return 0;
}
```

Listing 1: Implementation of Stack

4 Queue

It follow FIFO (First In First Out) scheme

- pop - Removes from front
- push - Adds to rear
- peek/top - See frontmost element

4.1 Implementation of Queue

```
#include <stdio.h>
#define MAX 3

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

void enqueue(int a) {
    if (rear + 1 >= MAX) {
        printf("Queue Overflow\n");
    } else {
        if (front == -1) front = 0;
        queue[++rear] = a;
    }
}

int dequeue() {
    if (front > rear) {
        printf("Queue underflow \n");
        return -1;
    } else {
        return queue[front++];
    }
}

void display() {
    if (rear == -1 || front > rear) {
        printf("Queue is empty!\n");
    } else {
        for (int i = rear; i >= front; i--) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
```

```
}

int main() {
    dequeue();

    enqueue(5);
    enqueue(8);

    display();

    dequeue();

    display();

    enqueue(3);

    display();

    return 0;
}
```

Listing 2: Implementation of Queue

5 Multi-Stacks

2 or more stacks in a single array.

5.1 2 stacks in one array

Stack 1 grows from left to right. Stack2 grows frome right to left. **Condition:**

- To push into stack1:
$$top1 + 1 < top2$$
- To push into stack2:
$$top2 - 1 > top1$$

6 Addition of sparse polynomial

All the polynomials are stored inside an array of structures:

```
    // Structure to represent a term
    typedef struct {
        int coeff;
        int expo;
    } Term;

```

```
    Term polynomial[] = {{2, 3},{4, 0}} // 2x^3 + 4
```

Listing 3: Sparse Polynomial Addition Outline

6.1 Logic when adding 2 polynomials

let it be poly1 (with i as indexing), poly2 (with j as indexing) & result (with k as indexing)

- If poly1[i].exp == poly2[j].exp: add coefficients
- If poly1[i].exp is greater than poly2[j].exp: Copy over poly1[i]
- If poly1[i].exp is less than poly2[j].exp: Copy over poly2[j]