# Data Structures in C

Data structures are fundamental concepts in computer science that help organize and store data efficiently. In C programming, understanding and implementing various data structures is essential for solving complex problems and optimizing performance. This document provides an overview of common data structures in C, including their implementation and usage.

## 1. Arrays

### Definition

An array is a collection of elements, all of the same type, stored in contiguous memory locations. Arrays provide fast access to elements but have a fixed size.

### Declaration

c

int arr[10]; // Declares an array of 10 integers

### Accessing Elements

c

arr[0] = 5; // Assigns 5 to the first element

int value = arr[2]; // Retrieves the third element

### Advantages

- Fast access to elements by index.

- Simple and easy to use.

### Disadvantages

- Fixed size.

- Insertion and deletion of elements can be costly.

## 2. Linked Lists

### Definition

A linked list is a collection of nodes where each node contains data and a pointer to the next node. Linked lists can be dynamically resized.

### Types

- \*Singly Linked List\*: Each node points to the next node.

- \*Doubly Linked List\*: Each node points to both the next and previous nodes.

### Implementation of Singly Linked List

c

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL; // Initialize head to NULL

// Function to insert a new node at the beginning

void insertAtBeginning(int newData) {

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

newNode->data = newData;

newNode->next = head;

head = newNode;

}

// Function to display the linked list

void printList() {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

### Advantages

- Dynamic size.

- Easy insertion and deletion.

### Disadvantages

- Slower access to elements.

- More memory overhead due to pointers.

## 3. Stacks

### Definition

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. It supports two primary operations: push (insert) and pop (delete).

### Implementation using Arrays

c

#define MAX 100

int stack[MAX];

int top = -1;

void push(int data) {

if (top == MAX - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = data;

}

int pop() {

if (top == -1) {

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

### Advantages

- Simple implementation.

- Useful for recursive algorithms and expression evaluation.

### Disadvantages

- Fixed size if implemented using arrays.

## 4. Queues

### Definition

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. It supports two primary operations: enqueue (insert) and dequeue (delete).

### Implementation using Arrays

c

#define MAX 100

int queue[MAX];

int front = -1, rear = -1;

void enqueue(int data) {

if (rear == MAX - 1) {

printf("Queue overflow\n");

return;

}

if (front == -1) front = 0;

queue[++rear] = data;

}

int dequeue() {

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

printf("Queue underflow\n");

return -1;

}

return queue[front++];

}

### Advantages

- Simple to implement.

- Useful in scheduling and buffering.

### Disadvantages

- Fixed size if implemented using arrays.

- Inefficient memory use if elements are dequeued frequently.

## 5. Trees

### Definition

A tree is a hierarchical data structure with a root node and child nodes forming a parent-child relationship. Trees are used to represent hierarchical data.

### Binary Tree Implementation

c

struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

};

struct TreeNode\* createNode(int data) {

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

newNode->data = data;

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

return newNode;

}

// Function to insert a node in a binary tree

struct TreeNode\* insertNode(struct TreeNode\* root, int data) {

if (root == NULL) return createNode(data);

if (data < root->data)

root->left = insertNode(root->left, data);

else

root->right = insertNode(root->right, data);

return root;

}

// In-order traversal

void inOrder(struct TreeNode\* root) {

if (root != NULL) {

inOrder(root->left);

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

inOrder(root->right);

}

}

### Advantages

- Reflects hierarchical structure.

- Efficient search, insert, and delete operations in balanced trees.

### Disadvantages

- Complexity in balancing trees.

- Inefficient if unbalanced.

## 6. Hash Tables

### Definition

A hash table is a data structure that maps keys to values for efficient lookup. It uses a hash function to compute an index into an array of buckets.

### Implementation

c

#define TABLE\_SIZE 100

struct HashItem {

int key;

int value;

};

struct HashItem\* hashTable[TABLE\_SIZE];

// Simple hash function

int hashFunction(int key) {

return key % TABLE\_SIZE;

}

void insert(int key, int value) {

int hashIndex = hashFunction(key);

struct HashItem\* newItem = (struct HashItem\*)malloc(sizeof(struct HashItem));

newItem->key = key;

newItem->value = value;

// Linear probing

while (hashTable[hashIndex] != NULL && hashTable[hashIndex]->key != -1) {

hashIndex++;

hashIndex %= TABLE\_SIZE;

}

hashTable[hashIndex] = newItem;

}

int search(int key) {

int hashIndex = hashFunction(key);

while (hashTable[hashIndex] != NULL) {

if (hashTable[hashIndex]->key == key)

return hashTable[hashIndex]->value;

hashIndex++;

hashIndex %= TABLE\_SIZE;

}

return -1;

}

### Advantages

- Fast lookups.

- Efficient for large datasets.

### Disadvantages

- Handling collisions.

- Requires a good hash function.

## Conclusion

Understanding and implementing these data structures in C is crucial for developing efficient algorithms and software. Each data structure has its unique strengths and weaknesses, making them suitable for different types of applications. Mastery of these concepts will enable you to choose the right data structure for the problem at hand and implement it effectively in C.