Additional Data Structures and Programs

1. Circular Queue

- **Definition:** A queue where the last position connects back to the first position, forming a circle.

Program

```
#include <stdio.h>
#define SIZE 5

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

void enqueue(int value) {
   if ((rear + 1) % SIZE == front) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1) front = 0;
   rear = (rear + 1) % SIZE;
   queue[rear] = value;
   printf("%d enqueued\n", value);
}

void dequeue() {
   if (front == -1) {
      printf("Queue Underflow\n");
      return;
   }
```

```
    printf("%d dequeued\n", queue[front]);
    if (front == rear) front = rear = -1;
    else front = (front + 1) % SIZE;
}

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

---

2. Heapify in Max-Heap

- **Definition:** Converts an array into a max-heap where the largest element is at the root.

Program

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;
    if (right < n && arr[right] > arr[largest]) largest = right;
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

int main() {
    int arr[] = {1, 3, 5, 7, 9};
    int n = 5;
    heapify(arr, n, 0);
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    return 0;
}
```

---

3. Trie Insertion and Search

- **Definition:** A tree-like structure used to store strings efficiently.

Program

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

struct TrieNode {
    struct TrieNode* children[26];
    int isEndOfWord;
};

struct TrieNode* createNode() {
    struct TrieNode* node = (struct TrieNode*)malloc(sizeof(struct TrieNode));
    node->isEndOfWord = 0;
    for (int i = 0; i < 26; i++) node->children[i] = NULL;
    return node;
}

void insert(struct TrieNode* root, const char* key) {
    struct TrieNode* current = root;
    for (int i = 0; i < strlen(key); i++) {
        int index = key[i] - 'a';
        if (!current->children[index]) current->children[index] = createNode();
        current = current->children[index];
    }
    current->isEndOfWord = 1;
}

int search(struct TrieNode* root, const char* key) {
    struct TrieNode* current = root;
    for (int i = 0; i < strlen(key); i++) {
        int index = key[i] - 'a';
        if (!current->children[index]) return 0;
        current = current->children[index];
    }
    return current->isEndOfWord;
}

int main() {
    struct TrieNode* root = createNode();
    insert(root, "hello");
    printf("Search 'hello': %s\n", search(root, "hello") ? "Found" : "Not Found");
    return 0;
}
```

---

4. Reverse an Array

- **Definition:** Reverses the elements of an array in-place.

Program

```
#include <stdio.h>

void reverseArray(int arr[], int n) {
    int start = 0, end = n - 1;
    while (start < end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

```
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = 5;
    reverseArray(arr, n);
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    return 0;
}
```

---

5. Count Nodes in a Linked List

- **Definition:** Counts the number of nodes in a singly linked list.

Program

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

struct Node {
    int data;
    struct Node* next;
};

int countNodes(struct Node* head) {
    int count = 0;
    while (head != NULL) {
        count++;
        head = head->next;
    }
    return count;
}

int main() {
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 10;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 20;
    head->next->next = NULL;

    printf("Number of nodes: %d\n", countNodes(head));
    return 0;
}
```

---

6. Check Palindrome Using Stack

- **Definition:** Checks if a string is a palindrome using a stack.

Program

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

#define MAX 100
char stack[MAX];
int top = -1;

void push(char c) {
   stack[++top] = c;
}
```

```
char pop() {
    return stack[top--];
}

int isPalindrome(char str[]) {
    int n = strlen(str);
    for (int i = 0; i < n; i++) push(str[i]);
    for (int i = 0; i < n; i++)
        if (str[i] != pop()) return 0;
    return 1;
}

int main() {
    char str[] = "radar";
    printf("%s is %s\n", str, isPalindrome(str) ? "a Palindrome" : "not a Palindrome");
    return 0;
}
```

---

7. Calculate Depth of a Binary Tree

- **Definition:** Finds the maximum depth of a binary tree.

Program

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

int depth(struct Node* root) {
    if (root == NULL) return 0;
    int leftDepth = depth(root->left);
    int rightDepth = depth(root->right);
    return (leftDepth > rightDepth ? leftDepth : rightDepth) + 1;
}

int main() {
    struct Node* root = malloc(sizeof(struct Node));
    root->left = malloc(sizeof(struct Node));
    root->right = malloc(sizeof(struct Node));
    root->left->left = NULL;
    root->right->right = NULL;
    printf("Depth: %d\n", depth(root));
    return 0;
}
```