DSA Practical Programs in C

1) Right Angle Triangle with Numbers

Program:

```
#include <stdio.h>

int main() {
   int n, i, j;
   printf("Enter number of rows: ");
   scanf("%d", &n);

   for(i = 1; i <= n; i++) {
      for(j = 1; j <= i; j++) {
         printf("%d ", j);
      }
      printf("\n");
   }
   return 0;
}
```

Output:

Output (for n=5):

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

---

2) Pyramid with Asterisks

Program:

```
#include <stdio.h>

int main() {
   int n, i, j, space;
   printf("Enter number of rows: ");
   scanf("%d", &n);
```

```
   for(i = 1; i <= n; i++) {
      for(space = i; space < n; space++)
         printf(" ");
      for(j = 1; j <= (2*i-1); j++)
         printf("*");
      printf("\n");
   }
   return 0;
}
```

Output:

Output (for n=5):

```
   *
  ***
 *****
*******
*********
```

---

3) Binary Search

Program:

```
#include <stdio.h>

int main() {
   int arr[10], n, i, key, low, high, mid;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter sorted array elements: ");
   for(i = 0; i < n; i++)
      scanf("%d", &arr[i]);

   printf("Enter element to search: ");
   scanf("%d", &key);

   low = 0;
   high = n - 1;
   while(low <= high) {
      mid = (low + high) / 2;
      if(arr[mid] == key) {
         printf("Element found at position %d\n", mid + 1);
         return 0;
```

```
    } else if(arr[mid] < key)
        low = mid + 1;
    else
        high = mid - 1;
  }
  printf("Element not found.\n");
  return 0;
}
```

---

4) Sequential Search

Program:

```
#include <stdio.h>

int main() {
  int arr[10], n, key, i, found = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  printf("Enter array elements: ");
  for(i = 0; i < n; i++)
    scanf("%d", &arr[i]);

  printf("Enter element to search: ");
  scanf("%d", &key);

  for(i = 0; i < n; i++) {
    if(arr[i] == key) {
      printf("Element found at position %d\n", i + 1);
      found = 1;
      break;
    }
  }
  if(!found)
    printf("Element not found.\n");
  return 0;
}
```

Output:

Output Example:
Array: 3 7 9 12 15

Key: 9
Element found at position 3

---

5) Bubble Sort

Program:

```
#include <stdio.h>

int main() {
   int arr[10], n, i, j, temp;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter array elements: ");
   for(i = 0; i < n; i++)
      scanf("%d", &arr[i]);

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

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

Output:

Output Example:
Input: 5 3 2 4 1
Sorted: 1 2 3 4 5

---

6) Stack Operations (Push, Pop, Display)

Program:

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

int stack[MAX], top = -1;

void push(int x) {
   if(top == MAX-1)
      printf("Stack Overflow\n");
   else {
      stack[++top] = x;
      printf("%d pushed\n", x);
   }
}

void pop() {
   if(top == -1)
      printf("Stack Underflow\n");
   else
      printf("%d popped\n", stack[top--]);
}

void display() {
   if(top == -1)
      printf("Stack is empty\n");
   else {
      printf("Stack elements: ");
      for(int i = top; i >= 0; i--)
         printf("%d ", stack[i]);
      printf("\n");
   }
}

int main() {
   int ch, val;
   do {
      printf("\n1.PUSH 2.POP 3.DISPLAY 4.EXIT\nEnter choice: ");
      scanf("%d", &ch);
      switch(ch) {
         case 1: printf("Enter value: ");
               scanf("%d", &val);
```

```
            push(val);
            break;
        case 2: pop(); break;
        case 3: display(); break;
      }
   } while(ch != 4);
   return 0;
}
```

Output:

Output Example:
1.PUSH 2.POP 3.DISPLAY 4.EXIT
Enter value: 10
10 pushed
Stack elements: 10

---

7) Queue Operations (Enqueue, Dequeue, Display)

Program:

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

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

void enqueue(int x) {
   if(rear == MAX-1)
      printf("Queue Overflow\n");
   else {
      if(front == -1) front = 0;
      queue[++rear] = x;
      printf("%d inserted\n", x);
   }
}

void dequeue() {
   if(front == -1 || front > rear)
      printf("Queue Underflow\n");
   else
      printf("%d deleted\n", queue[front++]);
}
```

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

int main() {
   int ch, val;
   do {
      printf("\n1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT\nEnter choice: ");
      scanf("%d", &ch);
      switch(ch) {
         case 1: printf("Enter value: ");
               scanf("%d", &val);
               enqueue(val);
               break;
         case 2: dequeue(); break;
         case 3: display(); break;
      }
   } while(ch != 4);
   return 0;
}
```

Output:

Output Example:
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter value: 5
5 inserted
Queue elements: 5

---

8) Singly Linked List (Create and Display)

Program:

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

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

int main() {
   struct Node *head = NULL, *temp, *newNode;
   int n, i;
   printf("Enter number of nodes: ");
   scanf("%d", &n);

   for(i = 0; i < n; i++) {
      newNode = (struct Node*)malloc(sizeof(struct Node));
      printf("Enter data for node %d: ", i+1);
      scanf("%d", &newNode->data);
      newNode->next = NULL;

      if(head == NULL)
         head = newNode;
      else {
         temp = head;
         while(temp->next != NULL)
            temp = temp->next;
         temp->next = newNode;
      }
   }

   printf("\nLinked List: ");
   temp = head;
   while(temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
   return 0;
}
```

Output:

Output Example:
Nodes: 3
Data: 10 20 30
Linked List: 10 -> 20 -> 30 -> NULL

---

9) Inorder Tree Traversal (Recursion)

Program:

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

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

struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}

void inorder(struct Node* root) {
   if(root == NULL)
      return;
   inorder(root->left);
   printf("%d ", root->data);
   inorder(root->right);
}

int main() {
   struct Node* root = createNode(1);
   root->left = createNode(2);
   root->right = createNode(3);
   root->left->left = createNode(4);
   root->left->right = createNode(5);

   printf("Inorder Traversal: ");
   inorder(root);
   return 0;
}
```

Output:

Output:
Inorder Traversal: 4 2 5 1 3

---

10) Count Number of Nodes in a Graph

Program:

```
#include <stdio.h>

int main() {
   int n, i, j;
   printf("Enter number of nodes: ");
   scanf("%d", &n);

   int graph[n][n];
   printf("Enter adjacency matrix:\n");
   for(i = 0; i < n; i++)
      for(j = 0; j < n; j++)
         scanf("%d", &graph[i][j]);=+

   printf("Number of nodes in graph = %d\n", n);
   return 0;
}
```

Output:

Output Example:
Enter number of nodes: 4
Number of nodes in graph = 4

-1 Algorithm: Right Angle Triangle with Numbers

Step 1: Start

Step 2: Take the number of rows as input (n).

Step 3: Use two nested loops — outer for rows, inner for printing numbers.

Step 4: Print numbers from 1 to the current row number.

Step 5: Move to the next line after each row.

Step 6: Stop.

2 Algorithm: Pyramid with Asterisks

Step 1: Start

Step 2: Take number of rows as input.

Step 3: Use loop to print spaces then asterisks for each row.

Step 4: After each row, print newline.

Step 5: Stop.

3 Algorithm: Binary Search

Step 1: Start

Step 2: Take sorted array and element to search.

Step 3: Initialize low = 0, high = n-1.

Step 4: Repeat until low ≤ high.

Step 5: Find mid = (low + high) / 2.

Step 6: If arr[mid] == key, element found.

Step 7: If key < arr[mid], set high = mid - 1 else low = mid + 1.

Step 8: If not found, display message.

Step 9: Stop.

4 Algorithm: Sequential Search

Step 1: Start

Step 2: Take array and key as input.

Step 3: Loop through each element.

Step 4: If arr[i] == key, return position.

Step 5: If end of array, element not found.

Step 6: Stop.

5 Algorithm: Bubble Sort

Step 1: Start

Step 2: Take array input.

Step 3: Use two nested loops to compare adjacent elements.

Step 4: Swap if left element > right element.

Step 5: Repeat for all passes.

Step 6: Display sorted array.

Step 7: Stop.

6 Algorithm: Stack Operations

Step 1: Start

Step 2: Initialize stack and top = -1.

Step 3: For PUSH — insert element at top if not full.

Step 4: For POP — remove element from top if not empty.

Step 5: For DISPLAY — print all elements from top to bottom.

Step 6: Stop.

7 Algorithm: Queue Operations

Step 1: Start

Step 2: Initialize queue, front = 0, rear = -1.

Step 3: For ENQUEUE — add element to rear if not full.

Step 4: For DEQUEUE — remove element from front if not empty.

Step 5: For DISPLAY — print elements from front to rear.

Step 6: Stop.

8 Algorithm: Singly Linked List

Step 1: Start

Step 2: Define structure node (data, next).

Step 3: Create new node and link it at the end.

Step 4: Repeat for all insertions.

Step 5: Traverse and display list.

Step 6: Stop.

9 Algorithm: Tree Traversal (Inorder)

Step 1: Start

Step 2: Traverse left subtree recursively.

Step 3: Visit root node.

Step 4: Traverse right subtree recursively.

Step 5: Stop.

10 Algorithm: Count Nodes in Graph

Step 1: Start

Step 2: Take adjacency matrix as input.

Step 3: Initialize count = 0.

Step 4: For each node, increment count by 1.

Step 5: Display total count.

Step 6: Stop.