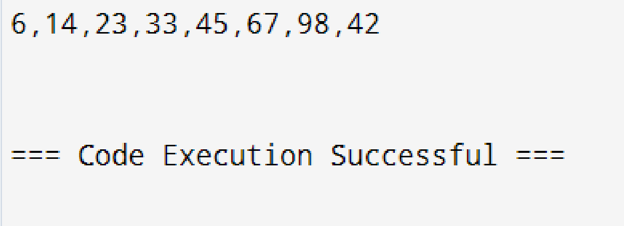
20 LAB PROGRAMS

1.

```
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int key = arr[i], j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
        if (i == 6)
        {
            for (int k = 0; k < n; k++)
                printf("%d%s", arr[k], k == n - 1 ? "" : ",");
            printf("\n");
        }
    }
}
int main() {
    int arr[] = {98, 23, 45, 14, 6, 67, 33, 42};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    return 0;
}
```

OUTPUT:

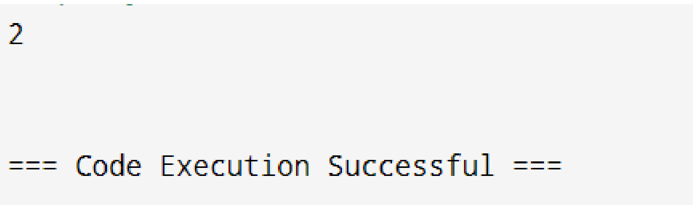


```
6,14,23,33,45,67,98,42


=== Code Execution Successful ===
```

2.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 100
typedef struct {
    int edges[MAX][MAX];
    int vertices;
} Graph;
void bfs(Graph *g, int start, int end) {
    int queue[MAX], front = 0, rear = 0, distance[MAX] = {0};
    bool visited[MAX] = {false};
    queue[rear++] = start;
    visited[start] = true;
    while (front < rear) {
        int current = queue[front++];
        for (int i = 0; i < g->vertices; i++) {
            if (g->edges[current][i] && !visited[i]) {
                queue[rear++] = i;
                visited[i] = true;
                distance[i] = distance[current] + 1;
                if (i == end) {
                    printf("%d\n", distance[i]);
                    return;
                }
            }
        }
    }
}

int main() {
    Graph g = {0};
    g.vertices = 6;
    g.edges[1][2] = g.edges[2][1] = 1;
    g.edges[2][5] = g.edges[5][2] = 1;
    bfs(&g, 1, 5);
    return 0;
}
```

OUTPUT:

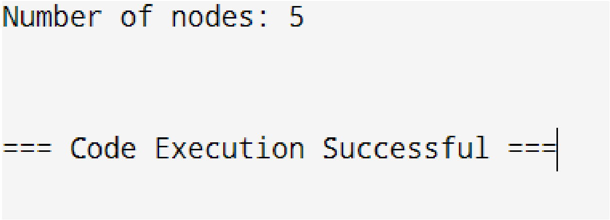


```
2


=== Code Execution Successful ===
```

3.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
int countNodes(struct Node* head) {
    int count = 0;
    while (head) {
        count++;
        head = head->next;
    }
    return count;
}
int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 3;
    head->next->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->next->data = 5;
    head->next->next->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->next->next->data = 8;

    head->next->next->next->next->next = NULL;

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

    return 0;

}
```

OUTPUT:

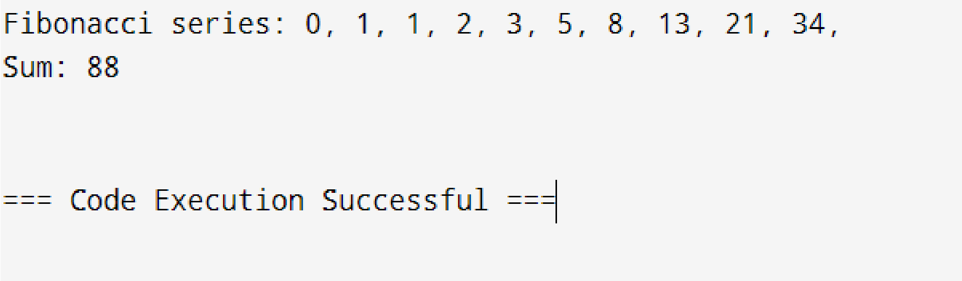


```
Number of nodes: 5


=== Code Execution Successful ===
```

4.

```
#include <stdio.h>

int fib(int n) {

    return (n <= 1) ? n : fib(n - 1) + fib(n - 2);

}

int sumFibonacci(int n) {

    return (n < 0) ? 0 : sumFibonacci(n - 1) + fib(n);

}

int main() {

    int n = 10, sum = 0;

    printf("Fibonacci series: ");

    for (int i = 0; i < n; i++) {

        printf("%d, ", fib(i));

    }
    sum = sumFibonacci(n - 1);

    printf("\nSum: %d\n", sum);

    return 0;

}
```

OUTPUT:



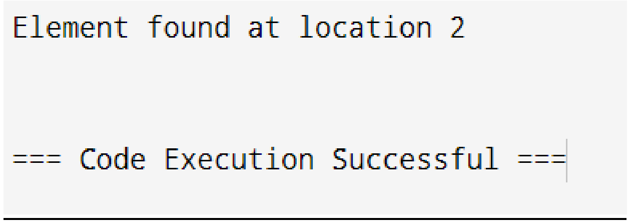
```
Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
Sum: 88


=== Code Execution Successful ===
```

5.

```
#include <stdio.h>

int binarySearch(int arr[], int size, int x) {

    int left = 0, right = size - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (arr[mid] == x) return mid;

        if (arr[mid] < x) left = mid + 1;

        else right = mid - 1;

    }

    return -1;

}

int main() {

    int arr[] = {1, 5, 6, 7, 9, 10}, x = 6;

    int result = binarySearch(arr, sizeof(arr)/sizeof(arr[0]), x);

    if (result != -1)
```

```
            printf("Element found at location %d\n", result);
        else
            printf("Element not found\n");
        return 0;
}
```

OUTPUT:

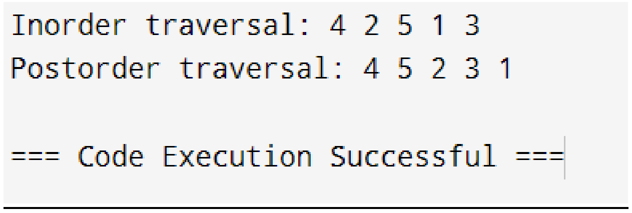


```
Element found at location 2


=== Code Execution Successful ===
```

6.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void postorder(struct Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
int main() {
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("Inorder traversal: ");
    inorder(root);
    printf("\nPostorder traversal: ");
    postorder(root);
    return 0;
}
```

OUTPUT:

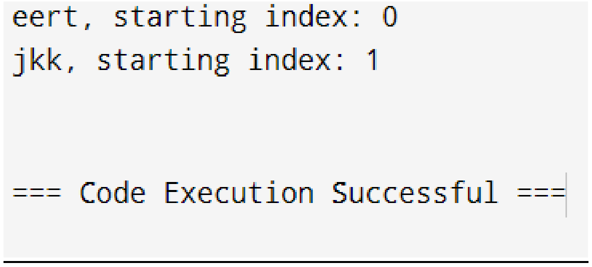


```
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1

=== Code Execution Successful ===
```

7.

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int cmp(const void *a, const void *b) { return *(char *)a - *(char *)b; }

void sortAndFindIndex(char *s) {

    int len = strlen(s), index = -1;

    qsort(s, len, sizeof(char), cmp);

    for (int i = 1; i < len; i++) {

        if (s[i] == s[i - 1]) {

            index = i - 1;

            break;

        }

    }

    printf("%s, starting index: %d\n", s, index);

}

int main() {

    char s1[] = "tree";

    char s2[] = "kkj";

    sortAndFindIndex(s1);

    sortAndFindIndex(s2);
    return 0;
}
```

OUTPUT:



```
eert, starting index: 0
jkk, starting index: 1


=== Code Execution Successful ===
```

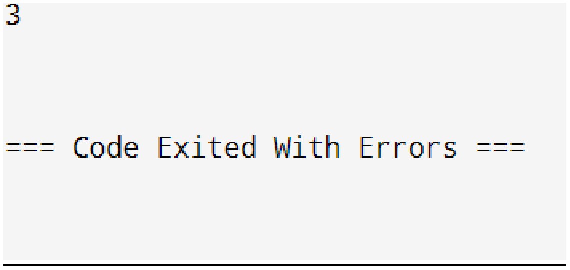
8.

```
#include <stdbool.h>
#include <stddef.h>
struct ListNode {
    int val;
    struct ListNode *next;
};
bool isPalindrome(struct ListNode* head) {
    if (!head || !head->next) return true;
    struct ListNode *slow = head, *fast = head, *prev = NULL;
    while (fast && fast->next) {
        fast = fast->next->next;
        struct ListNode *temp = slow->next;
        slow->next = prev;
        prev = slow;
        slow = temp;
    }
    struct ListNode *secondHalf = fast ? slow->next : slow;
    struct ListNode *firstHalf = prev;
    while (firstHalf && secondHalf) {
        if (firstHalf->val != secondHalf->val) return false;
        firstHalf = firstHalf->next;
        secondHalf = secondHalf->next;
    }
    return true;
}
```

9.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
int count = 0, result = 0;
void inorder(struct TreeNode* root, int k) {
    if (!root) return;
    inorder(root->left, k);
    if (++count == k) result = root->val;
    inorder(root->right, k);
}
int kthSmallest(struct TreeNode* root, int k) {
    inorder(root, k);
    return result;
}
```

OUTPUT:

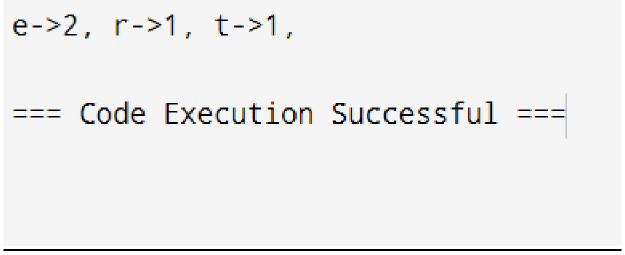


```
3


=== Code Exited With Errors ===
```

10.

```
#include <stdio.h>
#include <string.h>
void charFrequency(const char *s) {
    int freq[256] = {0};
    for (int i = 0; s[i]; i++) freq[(unsigned char)s[i]]++;
    for (int i = 0; i < 256; i++) if (freq[i]) printf("%c->%d, ", i, freq[i]);
}
int main() {
    char s[] = "tree";
    charFrequency(s);
    return 0;
}
```

OUTPUT:

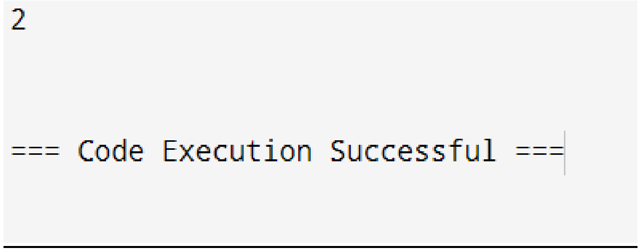


```
e->2, r->1, t->1,

=== Code Execution Successful ===
```

11.

```
#include <stdio.h>

#include <stdlib.h>

int findMissing(int arr[], int n) {

    for (int i = 0; i < n; i++) if (arr[i] <= 0) arr[i] = n + 1;

    for (int i = 0; i < n; i++) if (abs(arr[i]) <= n) arr[abs(arr[i]) - 1] = -abs(arr[abs(arr[i]) - 1]);

    for (int i = 0; i < n; i++) if (arr[i] > 0) return i + 1;

    return n + 1;

}

int main() {

    int arr[] = {1,3,4,5};

    printf("%d\n", findMissing(arr, sizeof(arr) / sizeof(arr[0])));

    return 0;

}
```

OUTPUT:



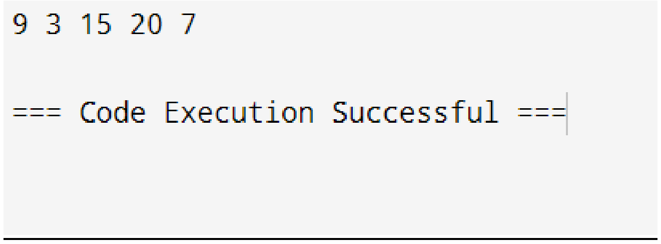
```
2


=== Code Execution Successful ===
```

12.

```
#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

    int val;

    struct TreeNode *left, *right;
```

```
};
struct TreeNode* buildTree(int* preorder, int preorderSize, int* inorder, int inorderSize) {
    if (preorderSize == 0 || inorderSize == 0) return NULL;
    struct TreeNode* root = malloc(sizeof(struct TreeNode));
    root->val = preorder[0];
    int rootIndex;
    for (rootIndex = 0; rootIndex < inorderSize; rootIndex++)
        if (inorder[rootIndex] == root->val) break;
    root->left = buildTree(preorder + 1, rootIndex, inorder, rootIndex);
    root->right = buildTree(preorder + rootIndex + 1, preorderSize - rootIndex - 1, inorder + rootIndex + 1, inorderSize - rootIndex - 1);
    return root;
}

void printInOrder(struct TreeNode* root) {
    if (root) {
        printInOrder(root->left);
        printf("%d ", root->val);
        printInOrder(root->right);
    }
}
int main() {
    int preorder[] = {3, 9, 20, 15, 7};
    int inorder[] = {9, 3, 15, 20, 7};
    struct TreeNode* root = buildTree(preorder, 5, inorder, 5);
    printInOrder(root); // Output: 9 3 15 20 7
    return 0;
}
```

OUTPUT:

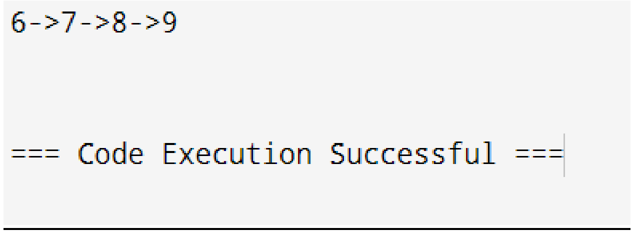


```
9 3 15 20 7

=== Code Execution Successful ===
```

13.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void displayList(struct Node* head) {
    while (head) {
        printf("%d", head->data);
        if (head->next) printf("->");
        head = head->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 6;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 7;
    head->next->next = malloc(sizeof(struct Node));
    head->next->next->data = 8;
    head->next->next->next = malloc(sizeof(struct Node));
    head->next->next->next->data = 9;
    head->next->next->next->next = NULL;
    displayList(head);
    struct Node* temp;
    while (head) {
        temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
```

OUTPUT:

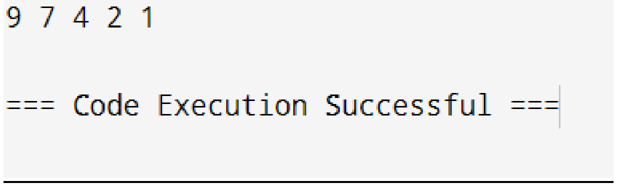


```
6->7->8->9


=== Code Execution Successful ===
```

14.

```
#include <stdio.h>
int main() {
    int arr[] = {4, 7, 9, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] < arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

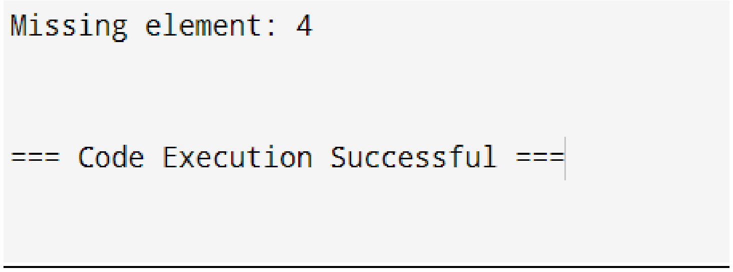


```
9 7 4 2 1

=== Code Execution Successful ===
```

15.

```
#include <stdio.h>
int findMissing(int arr[], int n) {
    int total = n * (n + 1) / 2;
    int sum = 0;
    for (int i = 0; i < n - 1; i++)
        sum += arr[i];
    return total - sum;
}
int main() {
    int arr[] = {1, 2, 3, 5};
    int n = 5;
    printf("Missing element: %d\n", findMissing(arr, n));
    return 0;
}
```

OUTPUT:

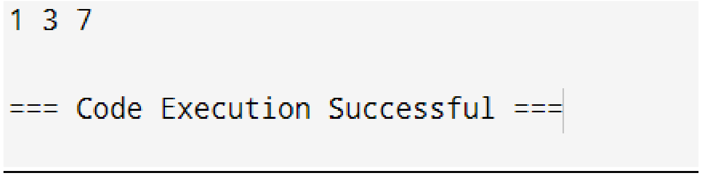


```
Missing element: 4


=== Code Execution Successful ===
```

16.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void printOddNumbers(struct Node* head) {
    while (head) {
        if (head->data % 2 != 0) {
            printf("%d ", head->data);
        }
        head = head->next;
    }
}
int main() {
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 1;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = malloc(sizeof(struct Node));
    head->next->next->data = 3;
    head->next->next->next = malloc(sizeof(struct Node));
    head->next->next->next->data = 7;
    head->next->next->next->next = NULL;
    printOddNumbers(head);
    struct Node* temp;
    while (head) {
        temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
```

OUTPUT:

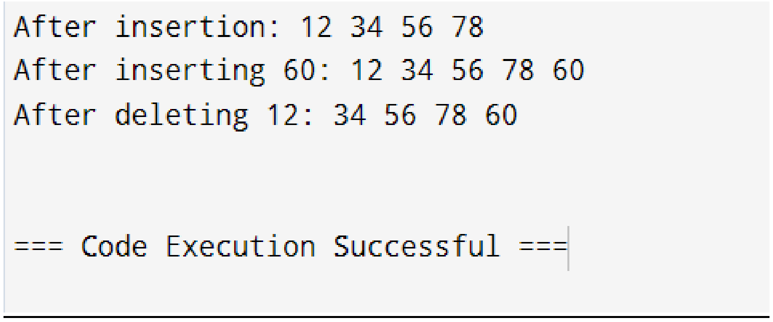


```
1 3 7

=== Code Execution Successful ===
```

17.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct Queue {

    int items[MAX];

    int front, rear;

} Queue;

void initQueue(Queue* q) {

    q->front = -1;

    q->rear = -1;

}

int isFull(Queue* q) {

    return q->rear == MAX - 1;

}

int isEmpty(Queue* q) {

    return q->front == -1 || q->front > q->rear;

}

void enqueue(Queue* q, int value) {

    if (isFull(q)) {

        printf("Queue is full!\n");

        return;

    }

    if (isEmpty(q)) q->front = 0;

    q->items[++q->rear] = value;

}

int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return -1;
    }
    return q->items[q->front++];
}
void displayQueue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    }
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}
int main() {
    Queue q;
    initQueue(&q);

    int values[] = {12, 34, 56, 78};
    for (int i = 0; i < 4; i++) {
        enqueue(&q, values[i]);
    }
    printf("After insertion: ");
    displayQueue(&q);
    enqueue(&q, 60);
    printf("After inserting 60: ");
    displayQueue(&q);
    dequeue(&q);
    printf("After deleting 12: ");
    displayQueue(&q);
    return 0;
}
```

OUTPUT:

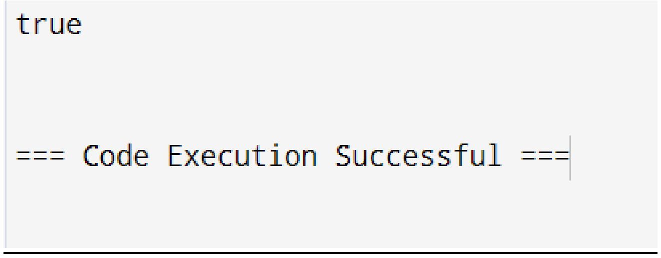


```
After insertion: 12 34 56 78
After inserting 60: 12 34 56 78 60
After deleting 12: 34 56 78 60


=== Code Execution Successful ===
```

18.

```
#include <stdio.h>
#include <stdlib.h>
int isValid(char *s) {
    char stack[100];
    int top = -1;
    for (int i = 0; s[i]; i++) {
        if (s[i] == '(' || s[i] == '{' || s[i] == '[') {
            stack[++top] = s[i];
        } else {
            if (top == -1 || (s[i] == ')' && stack[top] != '(') ||
                (s[i] == '}' && stack[top] != '{') ||
                (s[i] == ']' && stack[top] != '[')) {
                return 0;
            }
            top--;
        }
    }
    return top == -1;
}
int main() {
    char *s = "()";
    printf("%s\n", isValid(s) ? "true" : "false");
    return 0;
}
```

OUTPUT:

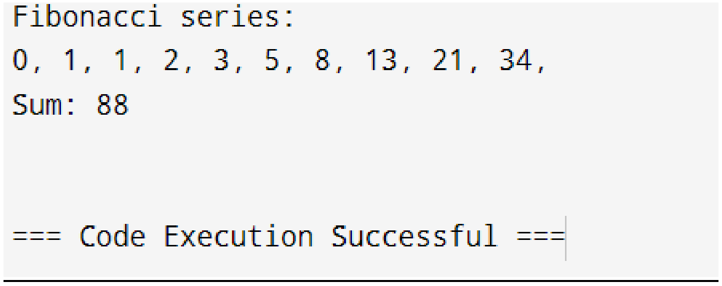


```
true


=== Code Execution Successful ===
```

19.

```
#include <stdio.h>
int main() {
    int n = 10, a = 0, b = 1, sum = 0;
    printf("Fibonacci series:\n");
    for (int i = 0; i < n; i++) {
        printf("%d, ", a);
        sum += a;

        int next = a + b;

        a = b;

        b = next;

    }

    printf("\nSum: %d\n", sum);

    return 0;

}
```

OUTPUT:

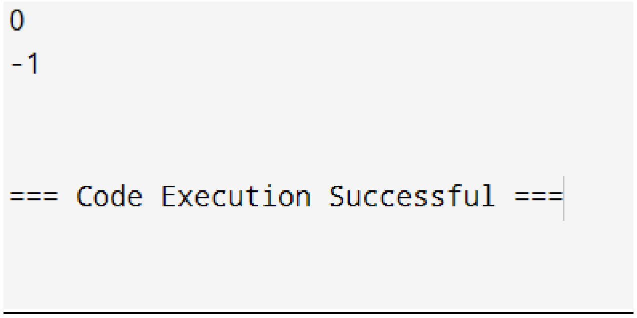


```
Fibonacci series:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
Sum: 88


=== Code Execution Successful ===
```

20.

```
#include <stdio.h>


int strStr(char *h, char *n) {

    if (!*n) return 0;

    for (char *p1 = h; *p1; p1++) {

        char *p1Begin = p1, *p2 = n;

        while (*p1 && *p2 && *p1 == *p2) {

            p1++;

            p2++;

        }

        if (!*p2) return p1Begin - h;

        p1 = p1Begin;
    }
    return -1;
}

int main() {
    printf("%d\n", strStr("sadbutsad", "sad")); // Output: 0
    printf("%d\n", strStr("leetcode", "leeto")); // Output: -1
    return 0;
}
```

OUTPUT:



```
0
-1


=== Code Execution Successful ===
```