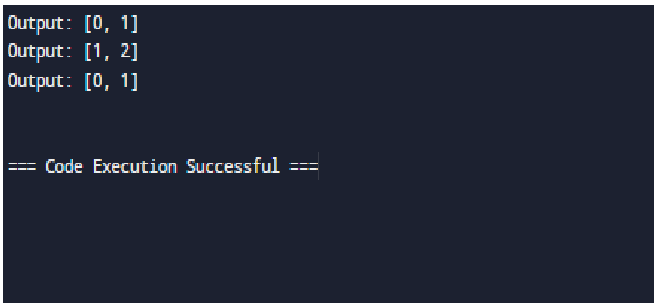
1. Two Sum

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

typedef struct {
    int key;
    int value;
} HashItem;

typedef struct {
    HashItem* items;
    int size;
    int capacity;
} HashMap;

HashMap* createHashMap(int capacity) {
    HashMap* map = (HashMap*)malloc(sizeof(HashMap));
    map->items = (HashItem*)calloc(capacity, sizeof(HashItem));
    map->size = 0;
    map->capacity = capacity;
    return map;
}
int hash(int key, int capacity) {
    return abs(key) % capacity;
}

void insert(HashMap* map, int key, int value) {
    int idx = hash(key, map->capacity);
    while (map->items[idx].key != 0) {
        idx = (idx + 1) % map->capacity;
    }
    map->items[idx].key = key;
    map->items[idx].value = value;
    map->size++;
}
```

```
int search(HashMap* map, int key) {
    int idx = hash(key, map->capacity);
    while (map->items[idx].key != 0) {
        if (map->items[idx].key == key) {
            return map->items[idx].value;
        }
        idx = (idx + 1) % map->capacity;
    }
    return -1;
}

int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    HashMap* map = createHashMap(numsSize);
    int* result = (int*)malloc(2 * sizeof(int));
    *returnSize = 2;

    for (int i = 0; i < numsSize; i++) {
        int complement = target - nums[i];
        int complementIndex = search(map, complement);
        if (complementIndex != -1) {
            result[0] = complementIndex;
            result[1] = i;
            return result;
        }
        insert(map, nums[i], i);
    }

    *returnSize = 0;
    return NULL;
}

int main() {
    int nums1[] = {2, 7, 11, 15};
    int target1 = 9;
    int returnSize;
    int* result1 = twoSum(nums1, 4, target1, &returnSize);
    printf("Output: [%d, %d]\n", result1[0], result1[1]);
    free(result1);

    int nums2[] = {3, 2, 4};
    int target2 = 6;
    int* result2 = twoSum(nums2, 3, target2, &returnSize);
    printf("Output: [%d, %d]\n", result2[0], result2[1]);
    free(result2);

    int nums3[] = {3, 3};
    int target3 = 6;
    int* result3 = twoSum(nums3, 2, target3, &returnSize);
    printf("Output: [%d, %d]\n", result3[0], result3[1]);
    free(result3);

    return 0;
}
```

OUTPUT

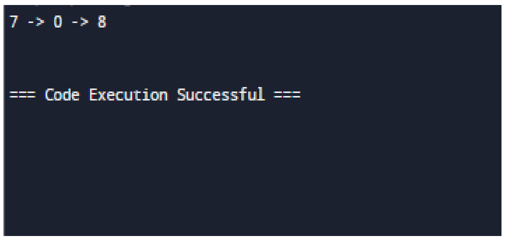


```
Output: [0, 1]
Output: [1, 2]
Output: [0, 1]


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

2. Add Two Numbers

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

struct ListNode {
    int val;
    struct ListNode* next;
};
struct ListNode* createNode(int val) {
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}

struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2) {
    struct ListNode dummy;
    struct ListNode* current = &dummy;
    dummy.next = NULL;
    int carry = 0;

    while (l1 != NULL || l2 != NULL) {
        int x = (l1 != NULL) ? l1->val : 0;
        int y = (l2 != NULL) ? l2->val : 0;
        int sum = carry + x + y;
        carry = sum / 10;
        current->next = createNode(sum % 10);
        current = current->next;
        if (l1 != NULL) l1 = l1->next;
        if (l2 != NULL) l2 = l2->next;
    }

    if (carry > 0) {
        current->next = createNode(carry);
    }

    return dummy.next;
}

void printList(struct ListNode* node) {
    while (node != NULL) {
        printf("%d", node->val);
        if (node->next != NULL) printf(" -> ");
        node = node->next;
    }
    printf("\n");
}
struct ListNode* createList(int arr[], int size) {
    if (size == 0) return NULL;
    struct ListNode* head = createNode(arr[0]);
    struct ListNode* current = head;
    for (int i = 1; i < size; i++) {
        current->next = createNode(arr[i]);
        current = current->next;
    }
    return head;
}

int main() {
    int arr1[] = {2, 4, 3};
    int arr2[] = {5, 6, 4};
    struct ListNode* l1 = createList(arr1, 3);
    struct ListNode* l2 = createList(arr2, 3);

    struct ListNode* result = addTwoNumbers(l1, l2);
    printList(result);
    while (l1 != NULL) {
        struct ListNode* temp = l1;
        l1 = l1->next;
        free(temp);
    }
    while (l2 != NULL) {
        struct ListNode* temp = l2;
        l2 = l2->next;
        free(temp);
    }
    while (result != NULL) {
        struct ListNode* temp = result;
        result = result->next;
        free(temp);
    }

    return 0;
}
```

OUTPUT

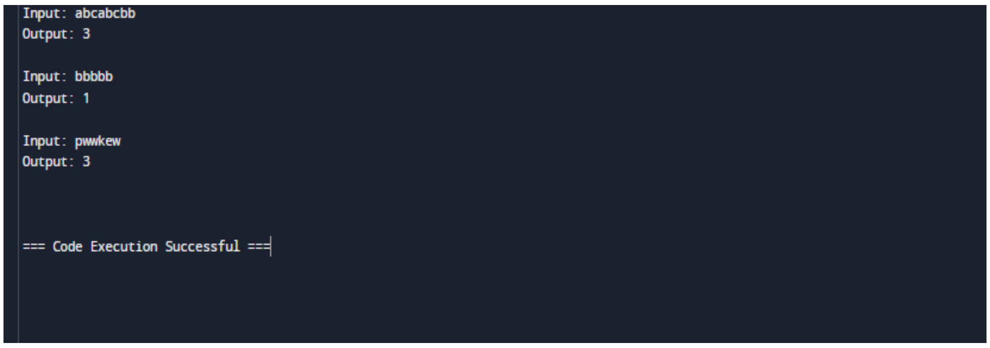


```
7 -> 0 -> 8


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

3. Longest Substring without Repeating Characters

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

int lengthOfLongestSubstring(char *s) {
    int n = strlen(s);
    if (n == 0) return 0;

    int lastIndex[256];
    for (int i = 0; i < 256; i++) {
        lastIndex[i] = -1;
    }

    int maxLength = 0;
    int start = 0;

    for (int end = 0; end < n; end++) {
        if (lastIndex[s[end]] != -1) {
            start = (start > lastIndex[s[end]] + 1) ? start : lastIndex[s[end]] + 1;
        }

        lastIndex[s[end]] = end;
        maxLength = (maxLength > (end - start + 1)) ? maxLength : (end - start + 1);
    }

    return maxLength;
}

int main() {
    char s1[] = "abcabcbb";
    printf("Input: %s\nOutput: %d\n\n", s1, lengthOfLongestSubstring(s1));

    char s2[] = "bbbbb";
    printf("Input: %s\nOutput: %d\n\n", s2, lengthOfLongestSubstring(s2));

    char s3[] = "pwwkew";
    printf("Input: %s\nOutput: %d\n\n", s3, lengthOfLongestSubstring(s3));

    return 0;
}
```

OUTPUT



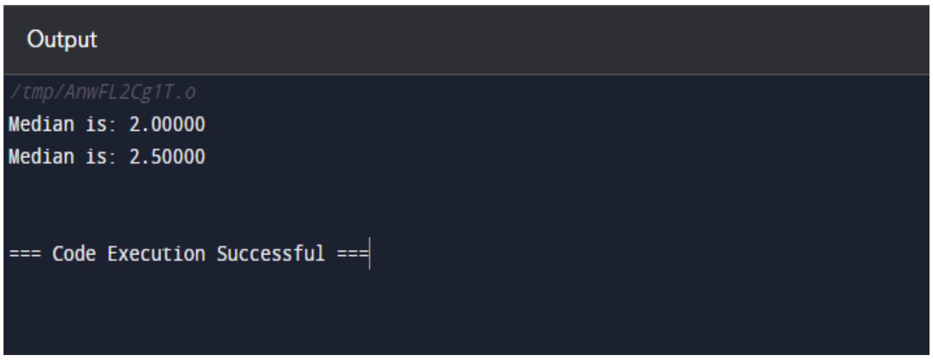
```
Input: abcabcbb
Output: 3

Input: bbbbb
Output: 1

Input: pwwkew
Output: 3


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

4. Median of Two Sorted Arrays

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

double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int nums2Size) {
    if (nums1Size > nums2Size) {

        return findMedianSortedArrays(nums2, nums2Size, nums1, nums1Size);
    }

    int x = nums1Size;
    int y = nums2Size;
    int low = 0;
    int high = x;
```

```c
    while (low <= high) {
        int partitionX = (low + high) / 2;
        int partitionY = (x + y + 1) / 2 - partitionX;

        int maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
        int maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];

        int minX = (partitionX == x) ? INT_MAX : nums1[partitionX];
        int minY = (partitionY == y) ? INT_MAX : nums2[partitionY];

        if (maxX <= minY && maxY <= minX) {

            if ((x + y) % 2 == 0) {
                return ((double)fmax(maxX, maxY) + fmin(minX, minY)) / 2;
            } else {
                return (double)fmax(maxX, maxY);
            }
        } else if (maxX > minY) {
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }



    return -1.0;
}

int main() {
    int nums1[] = {1, 3};
    int nums2[] = {2};
    printf("Median is: %.5f\n", findMedianSortedArrays(nums1, 2, nums2, 1));

    int nums3[] = {1, 2};
    int nums4[] = {3, 4};
    printf("Median is: %.5f\n", findMedianSortedArrays(nums3, 2, nums4, 2));
    return 0;
}
```

OUTPUT



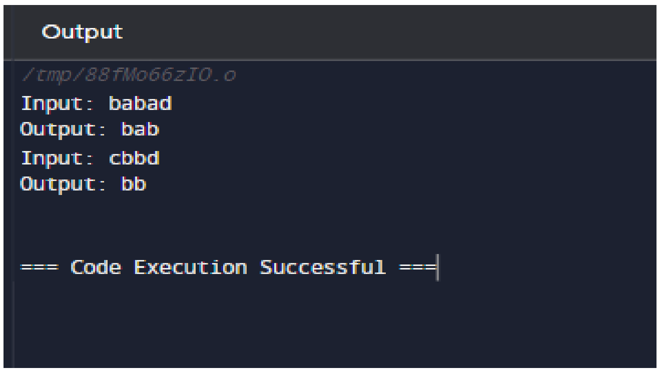
Output

```
/tmp/AnwFL2Cg1T.o
Median is: 2.00000
Median is: 2.50000


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

5. Longest Palindromic Substring

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

void expandAroundCenter(char* s, int left, int right, int* start, int* maxLength) {
    while (left >= 0 && right < strlen(s) && s[left] == s[right]) {
        left--;
        right++;
    }
    left++;
    right--;
    int len = right - left + 1;
    if (len > *maxLength) {
        *start = left;
        *maxLength = len;
    }
}

char* longestPalindrome(char* s) {
    int n = strlen(s);
    if (n == 0) return "";

    int start = 0, maxLength = 1;
    for (int i = 0; i < n; i++) {
        expandAroundCenter(s, i, i, &start, &maxLength);
        expandAroundCenter(s, i, i + 1, &start, &maxLength);
    }

    char* result = (char*)malloc((maxLength + 1) * sizeof(char));
    strncpy(result, s + start, maxLength);
    result[maxLength] = '\0';
    return result;
}

int main() {
    char s1[] = "babad";
    printf("Input: %s\nOutput: %s\n", s1, longestPalindrome(s1));

    char s2[] = "cbbd";
    printf("Input: %s\nOutput: %s\n", s2, longestPalindrome(s2));

    return 0;
}
```

OUTPUT



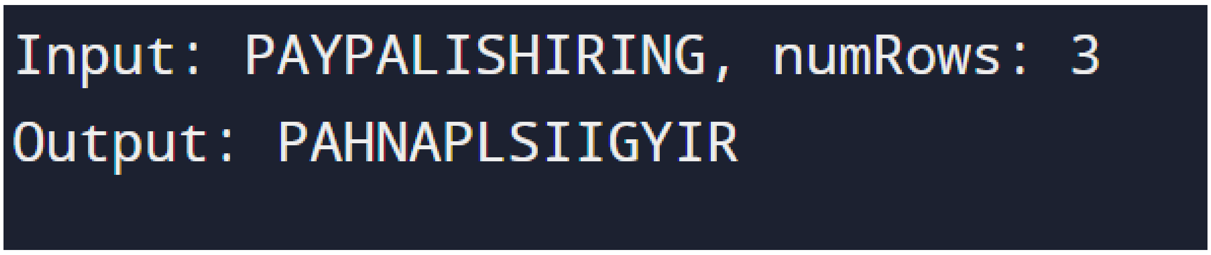
Output

```
/tmp/88fMo66zIO.o
Input: babad
Output: bab
Input: cbbd
Output: bb


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

6. Zigzag Conversion

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

char* convert(char* s, int numRows) {
    if (numRows == 1) return s;
    int len = strlen(s);
    char** rows = (char*)malloc(numRows * sizeof(char));
    for (int i = 0; i < numRows; i++) {
        rows[i] = (char*)malloc((len + 1) * sizeof(char));
        rows[i][0] = '\0';
    }

    int currRow = 0;
    int goingDown = 0;

    for (int i = 0; i < len; i++) {
        strncat(rows[currRow], &s[i], 1);

        if (currRow == 0 || currRow == numRows - 1) {
            goingDown = !goingDown;
        }

        currRow += goingDown ? 1 : -1;
    }

    char* result = (char*)malloc((len + 1) * sizeof(char));
    result[0] = '\0';

    for (int i = 0; i < numRows; i++) {
        strcat(result, rows[i]);
        free(rows[i]);
    }

    free(rows);

    return result;
}

int main() {
    char s1[] = "PAYPALISHIRING";
    int numRows1 = 3;
    printf("Input: %s, numRows: %d\nOutput: %s\n\n", s1, numRows1, convert(s1, numRows1));

    char s2[] = "PAYPALISHIRING";
    int numRows2 = 4;
    printf("Input: %s, numRows: %d\nOutput: %s\n\n", s2, numRows2, convert(s2, numRows2));

    char s3[] = "A";
    int numRows3 = 1;
    printf("Input: %s, numRows: %d\nOutput: %s\n", s3, numRows3, convert(s3, numRows3));

    return 0;
}
```

OUTPUT

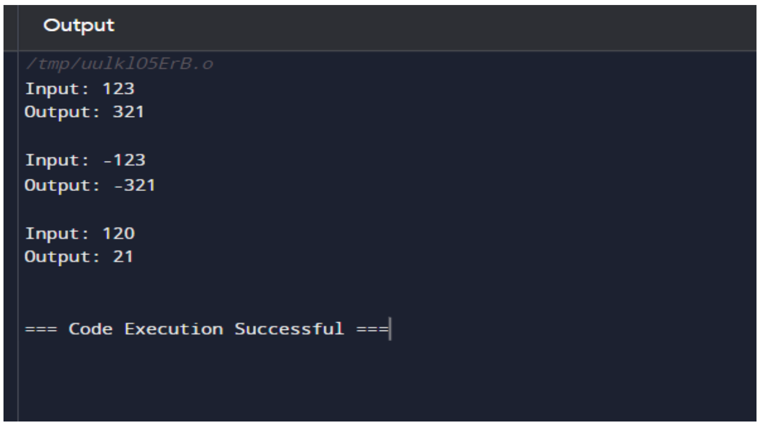


```
Input: PAYPALISHIRING, numRows: 3
Output: PAHNAPLSIIGYIR
```

7. Reverse Integer

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

int reverse(int x) {
    int reversed = 0;

    while (x != 0) {
        int digit = x % 10;
        x /= 10;

        if (reversed > INT_MAX / 10 || (reversed == INT_MAX / 10 && digit > 7)) {
            return 0;
        }
        if (reversed < INT_MIN / 10 || (reversed == INT_MIN / 10 && digit < -8)) {
            return 0;
        }
        reversed = reversed * 10 + digit;
    }

    return reversed;
}

int main() {
    int x1 = 123;
    printf("Input: %d\nOutput: %d\n\n", x1, reverse(x1));

    int x2 = -123;
    printf("Input: %d\nOutput: %d\n\n", x2, reverse(x2));

    int x3 = 120;
    printf("Input: %d\nOutput: %d\n", x3, reverse(x3));

    return 0;
}
```

OUTPUT



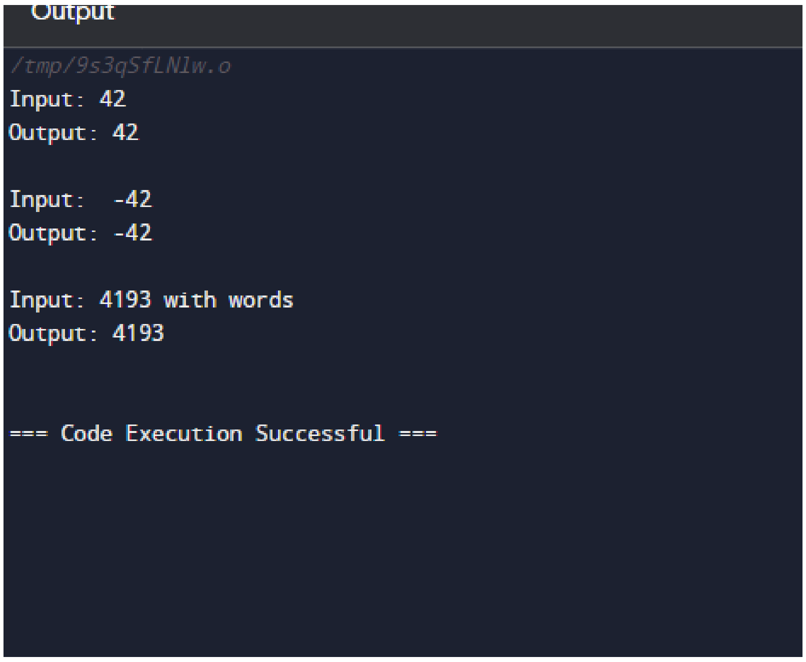
Output

```
/tmp/uulkl05ErB.o
Input: 123
Output: 321

Input: -123
Output: -321

Input: 120
Output: 21


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

8. String to Integer (atoi)

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

int myAtoi(char *s) {
    int sign = 1;
    long long result = 0;
    int i = 0;
    while (s[i] == ' ')
        i++;

    if (s[i] == '-') {
        sign = -1;
        i++;
    } else if (s[i] == '+') {
        i++;
    }

    while (s[i] >= '0' && s[i] <= '9') {
        result = result * 10 + (s[i] - '0');
        if (sign == 1 && result > INT_MAX)
            return INT_MAX;
        else if (sign == -1 && -result < INT_MIN)
            return INT_MIN;
        i++;
    }

    return sign * result;
}

int main() {
    char s1[] = "42";
    printf("Input: %s\nOutput: %d\n\n", s1, myAtoi(s1));

    char s2[] = " -42";
    printf("Input: %s\nOutput: %d\n\n", s2, myAtoi(s2));

    char s3[] = "4193 with words";
    printf("Input: %s\nOutput: %d\n", s3, myAtoi(s3));

    return 0;
}
```

OUTPUT

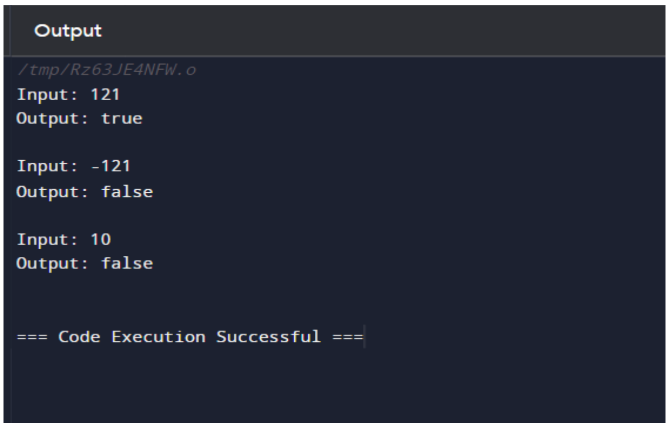


```
Output
/tmp/9s3qSfLNlw.o
Input: 42
Output: 42

Input:  -42
Output: -42

Input: 4193 with words
Output: 4193


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

9. Palindrome Number

```
#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>


bool isPalindrome(int x) {

    if (x < 0)

        return false;

    char str[12];

    sprintf(str, "%d", x);

    int len = strlen(str);

    for (int i = 0; i < len / 2; i++) {

        if (str[i] != str[len - i - 1]) {

            return false;

        }

    }

    return true;

}

int main() {

    int x1 = 121;

    printf("Input: %d\nOutput: %s\n\n", x1, isPalindrome(x1) ? "true" : "false");
    int x2 = -121;
    printf("Input: %d\nOutput: %s\n\n", x2, isPalindrome(x2) ? "true" : "false");

    int x3 = 10;
    printf("Input: %d\nOutput: %s\n", x3, isPalindrome(x3) ? "true" : "false");

    return 0;
}
```

OUTPUT

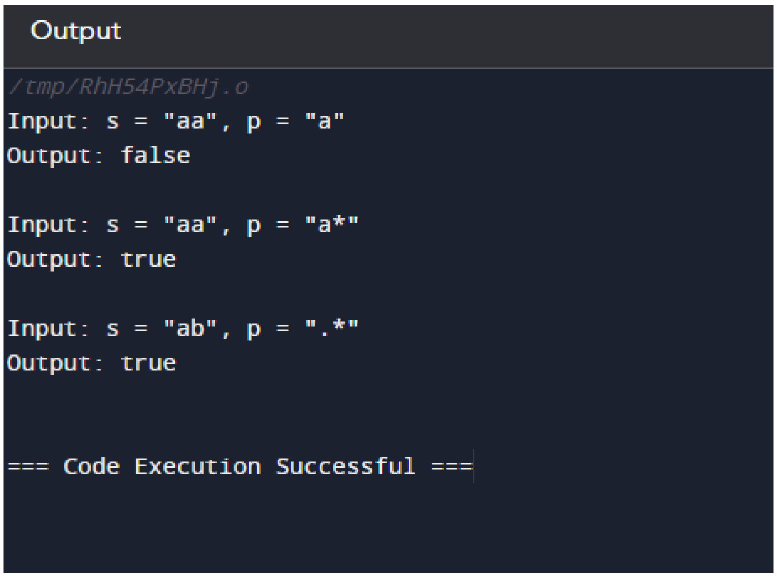


```
Output
/tmp/Rz63JE4NFW.o
Input: 121
Output: true

Input: -121
Output: false

Input: 10
Output: false


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

10. Regular Expression Matching

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

bool isMatch(char *s, char *p) {
    int m = strlen(s);
    int n = strlen(p);
    bool dp[m + 1][n + 1];
    memset(dp, false, sizeof(dp));

    dp[0][0] = true;
    for (int j = 1; j <= n; j++) {
        if (p[j - 1] == '*' && dp[0][j - 2]) {
            dp[0][j] = true;
        }
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (s[i - 1] == p[j - 1] || p[j - 1] == '.') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                if (s[i - 1] == p[j - 2] || p[j - 2] == '.') {
                    dp[i][j] = dp[i][j - 2] || dp[i - 1][j];
                } else {
                    dp[i][j] = dp[i][j - 2];
                }
            }
        }
    }

    return dp[m][n];
}

int main() {
    char s1[] = "aa";
    char p1[] = "a";
    printf("Input: s = \"%s\", p = \"%s\"\nOutput: %s\n\n", s1, p1, isMatch(s1, p1) ? "true" : "false");

    char s2[] = "aa";
    char p2[] = "a*";
    printf("Input: s = \"%s\", p = \"%s\"\nOutput: %s\n\n", s2, p2, isMatch(s2, p2) ? "true" : "false");

    char s3[] = "ab";
    char p3[] = ".*";
    printf("Input: s = \"%s\", p = \"%s\"\nOutput: %s\n", s3, p3, isMatch(s3, p3) ? "true" : "false");

    return 0;
}
```

OUTPUT



```
/tmp/RhH54PxBHj.o
Input: s = "aa", p = "a"
Output: false

Input: s = "aa", p = "a*"
Output: true

Input: s = "ab", p = ".*"
Output: true


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