

LAB 1

1. Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward.

Example 1:

Input: words = ["abc","car","ada","racecar","cool"]

Output: "ada"

Explanation: The first string that is palindromic is "ada".

Note that "racecar" is also palindromic, but it is not the first.

Example 2:

Input: words = ["notapalindrome","racecar"]

Output: "racecar"

Explanation: The first and only string that is palindromic is "racecar".

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

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

```
int isPalindrome(char str[]) {  
    int start = 0;  
    int end = strlen(str) - 1;  
  
    while (start < end) {  
        if (str[start] != str[end]) {  
            return 0; // Not a palindrome  
        }  
        start++;  
        end--;  
    }  
    return 1; // It is a palindrome  
}
```

```

int main() {

    int n, i;

    printf("Enter number of words: ");

    scanf("%d", &n);

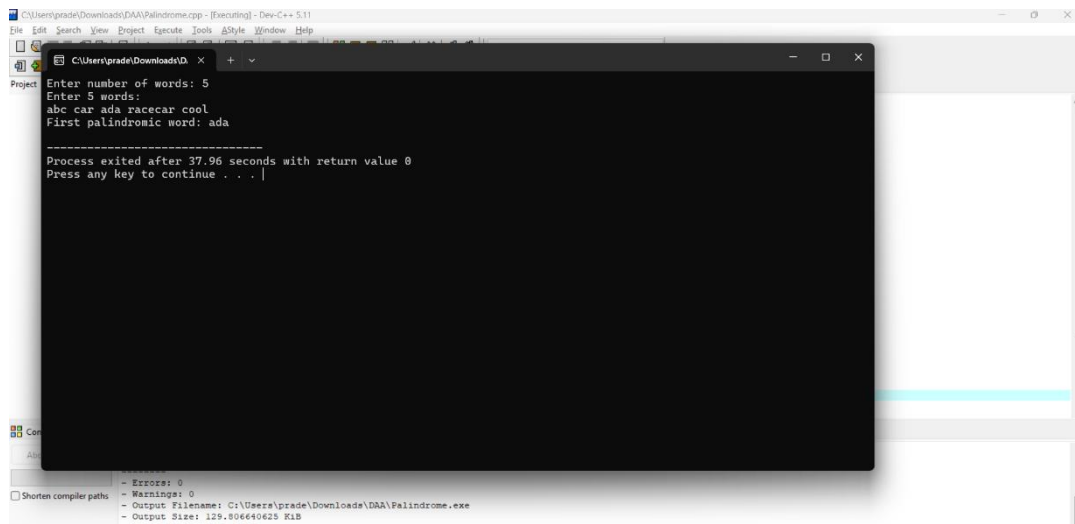
    char words[n][100];

    printf("Enter %d words:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%s", words[i]);
    }

    // Check for first palindrome
    for (i = 0; i < n; i++) {
        if (isPalindrome(words[i])) {
            printf("First palindromic word: %s\n", words[i]);
            return 0;
        }
    }

    printf("No palindromic word found.\n");
    return 0;
}

```



```

C:\Users\prade\Downloads\DAA\Palindrome.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools Style Window Help
Project
Enter number of words: 5
Enter 5 words:
abc car ada racecar cool
First palindromic word: ada

-----
Process exited after 37.96 seconds with return value 0
Press any key to continue . . .

```

- Errors: 0
 - Warnings: 0
 - Output Filename: C:\Users\prade\Downloads\DAA\Palindrome.exe
 - Output Size: 129.806640625 KiB

2. You are given two integer arrays `nums1` and `nums2` of sizes `n` and `m`, respectively. Calculate the following values: `answer1` : the number of indices `i` such that `nums1[i]` exists in `nums2`. `answer2` : the number of indices `i` such that `nums2[i]` exists in `nums1`. Return `[answer1,answer2]`.

Example 1:

Input: `nums1 = [2,3,2]`, `nums2 = [1,2]`

Output: `[2,1]`

Explanation:

Example 2:

Input: `nums1 = [4,3,2,3,1]`, `nums2 = [2,2,5,2,3,6]`

Output: `[3,4]`

Explanation:

The elements at indices 1, 2, and 3 in `nums1` exist in `nums2` as well. So `answer1` is 3.

The elements at indices 0, 1, 3, and 4 in `nums2` exist in `nums1`. So `answer2` is 4.

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

```
int main() {
```

```
    int n, m, i, j, answer1 = 0, answer2 = 0;
```

```
    printf("Enter size of first array: ");
```

```
    scanf("%d", &n);
```

```
    int nums1[n];
```

```
printf("Enter %d elements of first array:\n", n);  
for (i = 0; i < n; i++) {  
    scanf("%d", &nums1[i]);  
}
```

```
printf("Enter size of second array: ");  
scanf("%d", &m);  
int nums2[m];  
printf("Enter %d elements of second array:\n", m);  
for (i = 0; i < m; i++) {  
    scanf("%d", &nums2[i]);  
}
```

```
for (i = 0; i < n; i++) {  
    for (j = 0; j < m; j++) {  
        if (nums1[i] == nums2[j]) {  
            answer1++;  
            break;  
        }  
    }  
}
```

```
// Find how many nums2[i] exist in nums1  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        if (nums2[i] == nums1[j]) {  
            answer2++;  
            break;  
        }  
    }  
}
```

```

    }

}

}

printf("Result: [%d, %d]\n", answer1, answer2);

return 0;

}

```

The screenshot shows a C++ IDE with a file named DAA.cpp. The code in the IDE is as follows:

```

17     int nums2[*];
18     printf("Enter %d elements of second array:\n", m);

```

The output window shows the following execution flow:

```

Enter size of first array: 5
Enter 5 elements of first array:
4 3 2 3 1
Enter size of second array: 6
Enter 6 elements of second array:
2 2 5 2 3 6
Result: [3, 4]

-----
Process exited after 43.45 seconds with return value 0
Press any key to continue . . .

```

3. You are given a 0-indexed integer array `nums`. The distinct count of a subarray of `nums` is defined as: Let `nums[i..j]` be a subarray of `nums` consisting of all the indices from `i` to `j` such that $0 \leq i \leq j < \text{nums.length}$. Then the number of distinct values in `nums[i..j]` is called the distinct count of `nums[i..j]`. Return the sum of the squares of distinct counts of all subarrays of `nums`. A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

Input: `nums = [1,2,1]`

Output: 15

Explanation: Six possible subarrays are:

[1]: 1 distinct value

[2]: 1 distinct value

[1]: 1 distinct value

[1,2]: 2 distinct values

[2,1]: 2 distinct values

[1,2,1]: 2 distinct values

The sum of the squares of the distinct counts in all subarrays is equal to $1^2 + 1^2 + 1^2 + 2^2 + 2^2 + 2^2 = 15$.

Example 2:

Input: nums = [1,1]

Output: 3

Explanation: Three possible subarrays are:

[1]: 1 distinct value

[1]: 1 distinct value

[1,1]: 1 distinct value

The sum of the squares of the distinct counts in all subarrays is equal to $1^2 + 1^2 + 1^2 = 3$.

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

```
int main() {
```

```
    int n, i, j, k;
```

```
    printf("Enter number of elements: ");
```

```
    scanf("%d", &n);
```

```
    int nums[n];
```

```
    printf("Enter %d elements:\n", n);
```

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

```
        scanf("%d", &nums[i]);
```

```
    }
```

```
    int sum = 0;
```

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

```
        for (j = i; j < n; j++) {
```

```
            int distinct = 0;
```

```
            int seen[100] = {0};
```

```

        for (k = i; k <= j; k++) {
            int val = nums[k];
            if (seen[val] == 0) {
                seen[val] = 1;
                distinct++;
            }
        }

        sum += distinct * distinct;
    }
}

printf("Sum of squares of distinct counts: %d\n", sum);

return 0;
}

```

The screenshot shows a Dev-C++ IDE window titled "C:\Users\prade\Downloads\DAA\DAApp - [Executing] - Dev-C++ 5.11". The console window is open, displaying the following text:

```

Enter number of elements: 3
Enter 3 elements:
1 2 1
Sum of squares of distinct counts: 15

-----
Process exited after 12.86 seconds with return value 0
Press any key to continue . . .

```

Below the console window, the status bar shows the following information:

```

Line: 7 Col: 1 Sel: 0 Lines: 38 Length: 954 Insert Done parsing in 0.031 seconds

```

4. Given a 0-indexed integer array `nums` of length `n` and an integer `k`, return *the number of pairs* (i, j) *where* $0 \leq i < j < n$, *such that* `nums[i] == nums[j]` *and* $(i * j)$ *is divisible by* `k`.

Example 1:

Input: `nums = [3,1,2,2,2,1,3]`, `k = 2`

Output: 4

Explanation:

There are 4 pairs that meet all the requirements:

- `nums[0] == nums[6]`, and $0 * 6 == 0$, which is divisible by 2.
- `nums[2] == nums[3]`, and $2 * 3 == 6$, which is divisible by 2.
- `nums[2] == nums[4]`, and $2 * 4 == 8$, which is divisible by 2.
- `nums[3] == nums[4]`, and $3 * 4 == 12$, which is divisible by 2.

Example 2:

Input: `nums = [1,2,3,4]`, `k = 1`

Output: 0

Explanation: Since no value in `nums` is repeated, there are no pairs (i, j) that meet all the requirements.

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

```
int main() {  
    int n, k;  
  
    printf("Enter number of elements: ");  
    scanf("%d", &n);  
  
    int nums[n];  
    printf("Enter %d elements:\n", n);  
    for (int i = 0; i < n; i++) {  
        scanf("%d", &nums[i]);  
    }  
}
```



```

printf("Enter k (positive integer): ");
scanf("%d", &k);

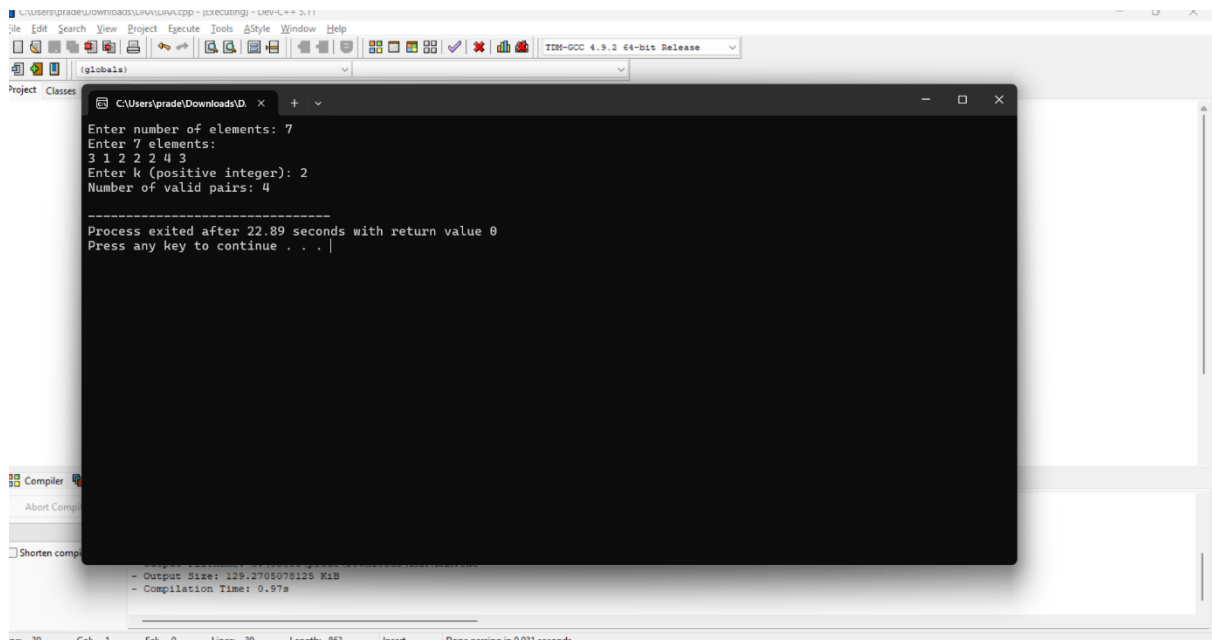
if (k <= 0) {
    printf("k must be a positive integer. Exiting.\n");
    return 0;
}

long long count = 0;

// Check every pair i < j
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        if (nums[i] == nums[j]) {
            long long prod = (long long)i * (long long)j;
            if (prod % k == 0) {
                count++;
            }
        }
    }
}

printf("Number of valid pairs: %lld\n", count);
return 0;
}

```



5. Write a program FOR THE BELOW TEST CASES with least time complexity
Test Cases: -

- 1) Input: {1, 2, 3, 4, 5} Expected Output: 5
- 2) Input: {7, 7, 7, 7, 7} Expected Output: 7
- 3) Input: {-10, 2, 3, -4, 5} Expected Output: 5

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

```
int main() {
```

```
    int n;
```

```
    printf("Enter number of elements: ");
```

```
    scanf("%d", &n);
```

```
    int arr[n];
```

```
    printf("Enter %d elements:\n", n);
```

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

```
        scanf("%d", &arr[i]);
```

```
    }
```

```
    int max = arr[0]; // Assume first element is max
```

```
    for (int i = 1; i < n; i++) {
```

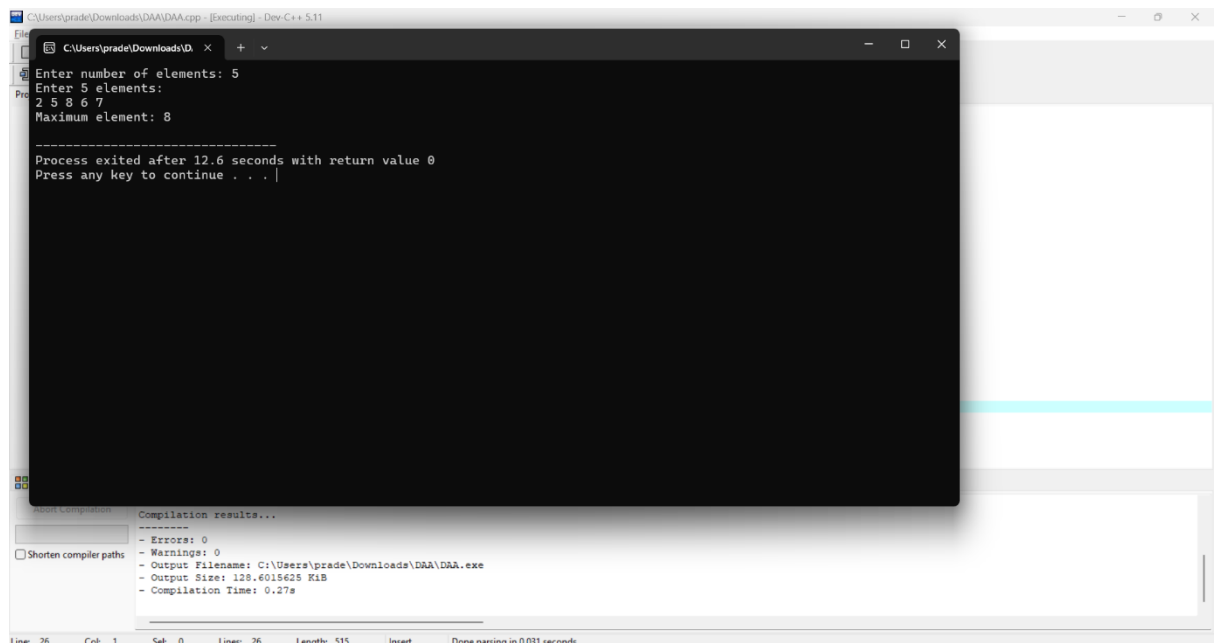
```

        if (arr[i] > max) {
            max = arr[i];
        }
    }

    printf("Maximum element: %d\n", max);

    return 0;
}

```



6. You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.

Test Cases

1. Empty List

1. Input: []
2. Expected Output: None or an appropriate message indicating that the list is empty.

2. Single Element List

1. Input: [5]

2. Expected Output: 5
3. All Elements are the Same
 1. Input: [3, 3, 3, 3, 3]
 2. Expected Output: 3

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

```
int main() {
```

```
    int n, i, j, temp;
```

```
    printf("Enter number of elements: ");
```

```
    scanf("%d", &n);
```

```
    // If list is empty
```

```
    if (n == 0) {
```

```
        printf("List is empty. No maximum element.\n");
```

```
        return 0;
```

```
    }
```

```
    int arr[n];
```

```
    printf("Enter the elements:\n");
```

```
    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;
        }
    }
}

int max = arr[n - 1];

printf("Maximum element: %d\n", max);

return 0;
}

```

The screenshot shows a C++ IDE with a project named 'DAA'. The main window displays the output of the program. The user has entered 4 elements: 1, 2, 6, and 7. The program correctly identifies the maximum element as 7. The console also shows the process exit time and a prompt to press any key to continue.

```

C:\Users\prade\Downloads\DAA\DAA.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
C:\Users\prade\Downloads\DAA\
Project
Enter number of elements: 4
Enter the elements:
1 2 6 7
Maximum element: 7
-----
Process exited after 5.381 seconds with return value 0
Press any key to continue . . . |
Console
Shorten compiler paths
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 129.2705078125 KiB
- Compilation Time: 0.56s
Line: 17 Col: 1 Sel: 0 Lines: 41 Length: 745 Insert Done parsing in 0 seconds

```

7. Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?

Test Cases

Some Duplicate Elements

- Input: [3, 7, 3, 5, 2, 5, 9, 2]
- Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)

Negative and Positive Numbers

- Input: [-1, 2, -1, 3, 2, -2]
- Expected Output: [-1, 2, 3, -2] (Order may vary)

List with Large Numbers

- Input: [1000000, 999999, 1000000]
- Expected Output: [1000000, 999999]

```
#include <stdio.h>

int main() {
    int n, i, j, isUnique;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    int unique[n];
    int uniqueCount = 0;

    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    for (i = 0; i < n; i++) {
        isUnique = 1;

        for (j = 0; j < uniqueCount; j++) {
            if (arr[i] == unique[j]) {
                isUnique = 0;
                break;
            }
        }

        if (isUnique) {
            unique[uniqueCount] = arr[i];
            uniqueCount++;
        }
    }

    printf("Unique Elements: ");
    for (i = 0; i < uniqueCount; i++) {
        printf("%d ", unique[i]);
    }
    printf("\n");

    return 0;
}
```

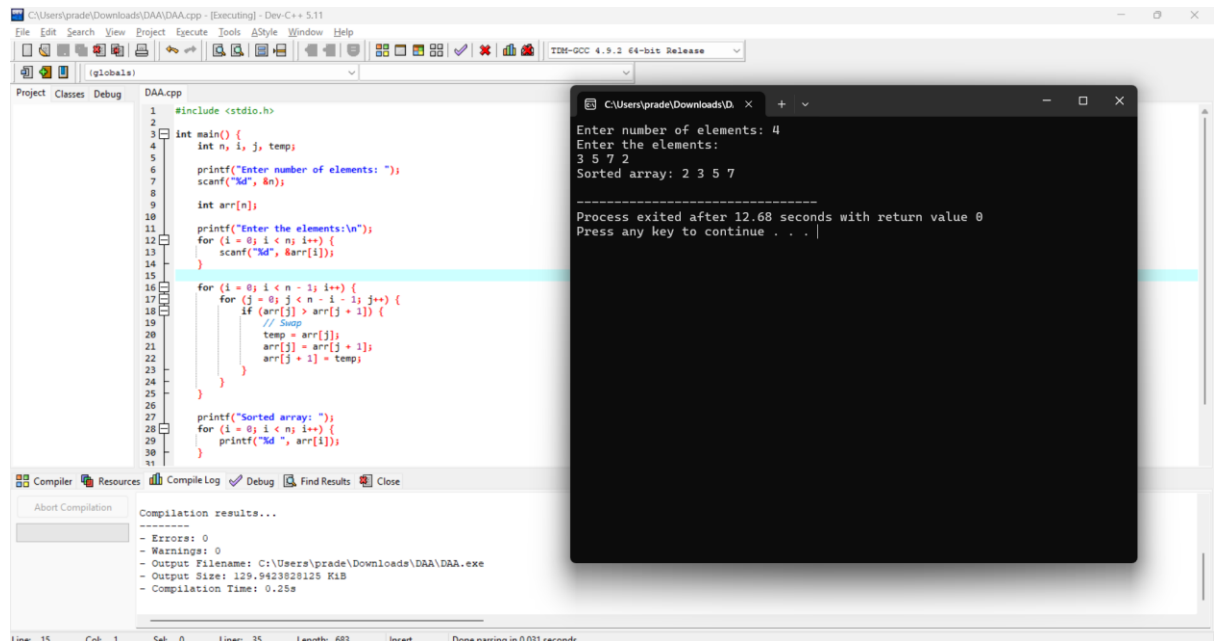
Enter number of elements: 3 7 3 5 2 5 9 2
Enter the elements:
Unique Elements: 7 3 5

Process exited after 21.75 seconds with return value 0
Press any key to continue . . .

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 130.4423828125 KiB
- Compilation Time: 0.70s

8. Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code.



The screenshot shows a C++ IDE with a file named DAA.cpp. The code implements a bubble sort algorithm. It prompts the user to enter the number of elements and the elements themselves. The output shows the sorted array and the execution time.

```
#include <stdio.h>

int main() {
    int n, i, j, temp;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];

    printf("Enter the elements:\n");
    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]) {
                // Swap
                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]);
    }
}
```

Execution output:

```
Enter number of elements: 4
Enter the elements:
3 5 7 2
Sorted array: 2 3 5 7

Process exited after 12.68 seconds with return value 0
Press any key to continue . . .
```

9. Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.

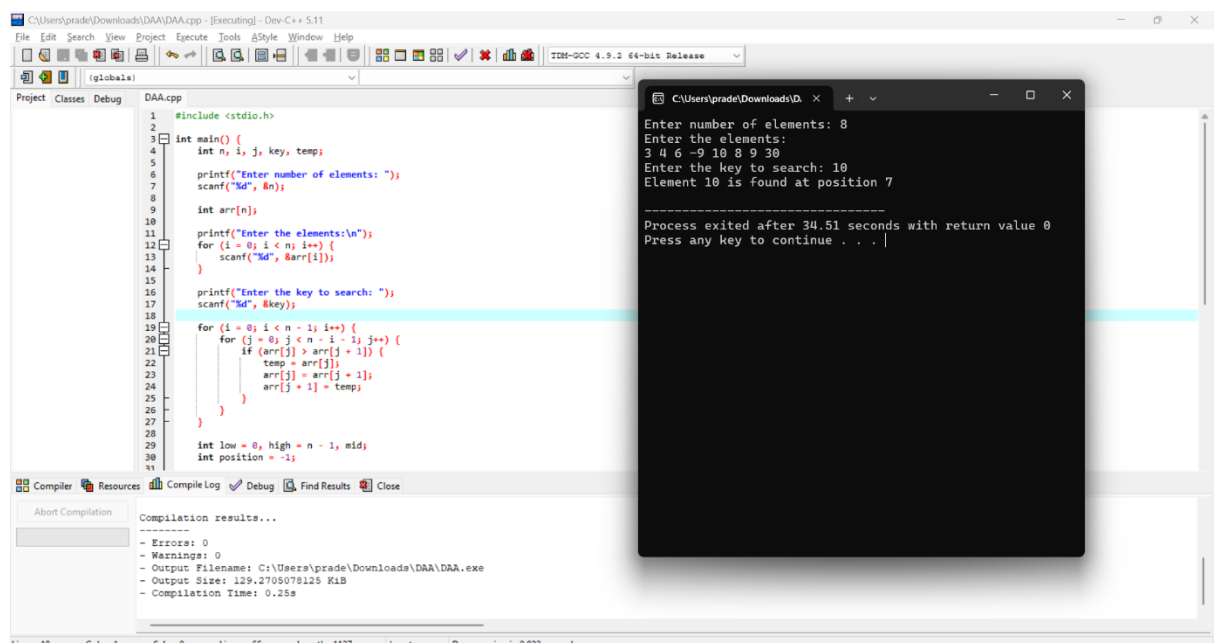
Test Case:

Example X={ 3,4,6,-9,10,8,9,30} KEY=10

Output: Element 10 is found at position 5

Example X={ 3,4,6,-9,10,8,9,30} KEY=100

Output : Element 100 is not found



The screenshot shows a C++ IDE with a file named DAA.cpp. The code implements a binary search algorithm. It prompts the user to enter the number of elements, the elements themselves, and the key to search for. The output shows the position of the key if found, or a message indicating it is not found.

```
#include <stdio.h>

int main() {
    int n, i, j, key, temp;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];

    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

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

    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;
            }
        }
    }

    int low = 0, high = n - 1, mid;
    int position = -1;

    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key) {
            position = mid;
            break;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    if (position != -1) {
        printf("Element %d is found at position %d", key, position);
    } else {
        printf("Element %d is not found", key);
    }
}
```

Execution output:

```
Enter number of elements: 8
Enter the elements:
3 4 6 -9 10 8 9 30
Enter the key to search: 10
Element 10 is found at position 5

Process exited after 34.51 seconds with return value 0
Press any key to continue . . .
```

10. Given an array of integers `nums`, sort the array in ascending order and return it. You must solve the problem without using any built-in functions in $O(n \log(n))$ time complexity and with the smallest space complexity possible.

The screenshot shows a C++ IDE with a project named 'DAA'. The code implements a heap sort algorithm. The `main` function reads an array of integers and sorts it using `heapSort`. The output window shows the sorted array: 2 3 5 9 15. The process exited after 7.047 seconds with return value 0.

```
18 }
19 }
20 }
21 void heapSort(int arr[], int n) {
22     for (int i = n / 2 - 1; i >= 0; i--)
23         heapify(arr, n, i);
24     for (int i = n - 1; i > 0; i--) {
25         int temp = arr[0];
26         arr[0] = arr[i];
27         arr[i] = temp;
28         heapify(arr, i, 0);
29     }
30 }
31
32 int main() {
33     int n;
34     scanf("%d", &n);
35
36     int arr[n];
37     for (int i = 0; i < n; i++)
38         scanf("%d", &arr[i]);
39
40     heapSort(arr, n);
41
42     for (int i = 0; i < n; i++)
43         printf("%d ", arr[i]);
44
45     return 0;
46 }
47
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 129.166015625 KiB
- Compilation Time: 0.28s

11. Given an $m \times n$ grid and a ball at a starting cell, find the number of ways to move the ball out of the grid boundary in exactly N steps.

Example:

Input: $m=2, n=2, N=2, i=0, j=0$ · Output: 6

Input: $m=1, n=3, N=3, i=0, j=1$ · Output: 12

The screenshot shows a C++ IDE with a project named 'DAA'. The code implements a dynamic programming solution for the grid movement problem. The `main` function reads the grid dimensions m, n , the number of steps N , and the starting cell (i, j) . It then calculates the number of ways to move the ball out of the grid boundary in exactly N steps. The output window shows the result: 3 3 2 1 1 4. The process exited after 17.79 seconds with return value 0.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     int m, n, N, si, sj;
7     if (scanf("%d %d %d %d %d", &m, &n, &N, &si, &sj) != 5) return 0;
8     unsigned long long *dp_prev = (unsigned long long*)calloc((size_t)m * n, sizeof(unsigned long long));
9     unsigned long long *dp_curr = (unsigned long long*)calloc((size_t)m * n, sizeof(unsigned long long));
10    if (!dp_prev || !dp_curr) return 0;
11    dp_prev[si * n + sj] = 1ULL;
12    unsigned long long ans = 0;
13    int dr[] = {-1, 1, 0, 0};
14    int dc[] = {0, 0, -1, 1};
15    for (int step = 1; step <= N; step++) {
16        memset(dp_curr, 0, (size_t)m * n * sizeof(unsigned long long));
17        for (int r = 0; r < m; r++) {
18            for (int c = 0; c < n; c++) {
19                unsigned long long ways = dp_prev[r * n + c];
20                if (ways == 0) continue;
21                for (int d = 0; d < 4; d++) {
22                    int nr = r + dr[d];
23                    int nc = c + dc[d];
24                    if (nr < 0 || nr >= m || nc < 0 || nc >= n) ans += ways;
25                    else dp_curr[nr * n + nc] += ways;
26                }
27            }
28        }
29        unsigned long long *tmp = dp_prev;
30        dp_prev = dp_curr;
31        dp_curr = tmp;
32    }
33    printf("%llu", ans);
34    return 0;
35}
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 129.2724609375 KiB
- Compilation Time: 0.31s

12. You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security systems connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.

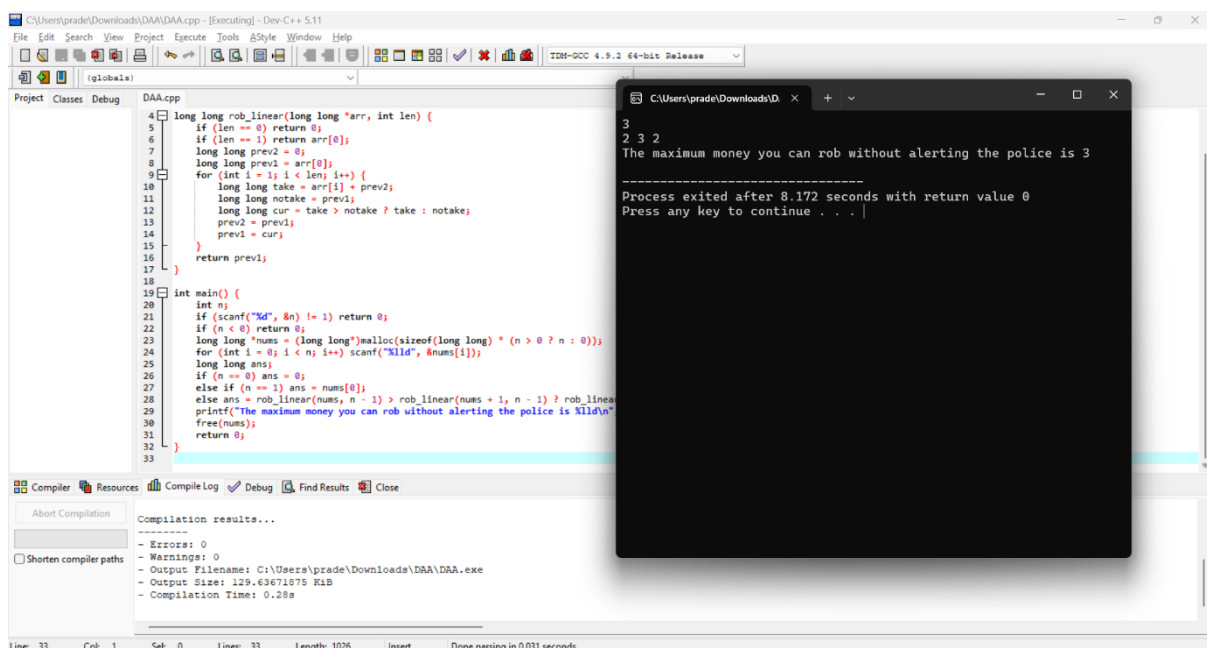
Examples:

(i) Input : nums = [2, 3, 2]

Output : The maximum money you can rob without alerting the police is 3 (robbing house 1).

(ii) Input : nums = [1, 2, 3, 1]

Output : The maximum money you can rob without alerting the police is 4 (robbing house 1 and house 3).



```
4 long long rob_linear(long long *arr, int len) {
5     if (len == 0) return 0;
6     if (len == 1) return arr[0];
7     long long prev2 = 0;
8     long long prev1 = arr[0];
9     for (int i = 1; i < len; i++) {
10        long long take = arr[i] + prev2;
11        long long notake = prev1;
12        long long cur = take > notake ? take : notake;
13        prev2 = prev1;
14        prev1 = cur;
15    }
16    return prev1;
17 }
18
19 int main() {
20     int n;
21     if (scanf("%d", &n) != 1) return 0;
22     if (n < 0) return 0;
23     long long *nums = (long long*)malloc(sizeof(long long) * (n > 0 ? n : 0));
24     for (int i = 0; i < n; i++) scanf("%lld", &nums[i]);
25     long long ans;
26     if (n == 0) ans = 0;
27     else if (n == 1) ans = nums[0];
28     else ans = rob_linear(nums, n - 1) > rob_linear(nums + 1, n - 1) ? rob_linear(nums, n - 1) : rob_linear(nums + 1, n - 1);
29     printf("The maximum money you can rob without alerting the police is %lld\n", ans);
30     free(nums);
31     return 0;
32 }
33 }
```

```
3
2 3 2
The maximum money you can rob without alerting the police is 3

Process exited after 8.172 seconds with return value 0
Press any key to continue . . .
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 129,636,718 bytes
- Compilation Time: 0.28s

Line: 33 Col: 1 Sel: 0 Lines: 33 Length: 1026 Insert Done parsing in 0.031 seconds

13. You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Examples:

(i) Input: $n=4$ Output: 5

(ii) Input: $n=3$ Output: 3

```

1 #include <stdio.h>
2
3 int main() {
4     int n;
5     scanf("%d", &n);
6
7     if (n == 0) {
8         printf("0");
9         return 0;
10    }
11    if (n == 1) {
12        printf("1");
13        return 0;
14    }
15
16    long long a = 1, b = 1, ways;
17    for (int i = 2; i <= n; i++) {
18        ways = a + b;
19        a = b;
20        b = ways;
21    }
22
23    printf("%lld", b);
24    return 0;
25 }

```

Compilation results...

```

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 128.7734375 KiB
- Compilation Time: 0.30s

```

Process exited after 1.944 seconds with return value 0
Press any key to continue . . .

14. A robot is located at the top-left corner of a $m \times n$ grid. The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid. How many possible unique paths are there?

Examples:

(i) Input: $m=7, n=3$ Output: 28

(ii) Input: $m=3, n=2$ Output: 3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int m, n;
6     scanf("%d %d", &m, &n);
7
8     long long *dp = (long long *) malloc(m * sizeof(long long));
9     for (int i = 0; i < m; i++)
10         dp[i] = (long long *) malloc(n * sizeof(long long));
11
12     for (int i = 0; i < m; i++)
13         dp[i][0] = 1;
14
15     for (int j = 0; j < n; j++)
16         dp[0][j] = 1;
17
18     for (int i = 1; i < m; i++)
19         for (int j = 1; j < n; j++)
20             dp[i][j] = dp[i-1][j] + dp[i][j-1];
21
22     printf("%lld", dp[m-1][n-1]);
23
24     for (int i = 0; i < m; i++)
25         free(dp[i]);
26     free(dp);
27
28     return 0;
29 }

```

Compilation results...

```

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DAA\DAA.exe
- Output Size: 128.6015625 KiB
- Compilation Time: 0.30s

```

Process exited after 16.82 seconds with return value 0
Press any key to continue . . .

15. In a string S of lowercase letters, these letters form consecutive groups of the same character. For example, a string like $s = \text{"abbxxxxzzy"}$ has the groups "a", "bb", "xxxx", "z", and "yy". A group is identified by an interval $[\text{start}, \text{end}]$, where start and end denote the start and end indices (inclusive) of the group. In the above example, "xxxx" has the interval $[3, 6]$. A group is considered large if it has 3 or more characters. Return the intervals of every large group sorted in increasing order by start index.

Example 1:

Input: $s = \text{"abbxxxxzzy"}$

Output: $[[3, 6]]$

Explanation: "xxxx" is the only large group with start index 3 and end index 6.

Example 2:

Input: $s = \text{"abc"}$

Output: $[]$

Explanation: We have groups "a", "b", and "c", none of which are large groups.

The screenshot shows a C++ IDE with a file named `DAA.cpp`. The code implements a solution to find large groups in a string. It uses a dynamic programming approach with a 2D array `dp` to store the end index of the largest group starting at each position `i`. The code reads a string `s` and its length `n`, then iterates over each character in the string, updating the `dp` array based on the current character and the previous group's end index. Finally, it prints the end index of the largest group for each starting position.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int n;
6     if (scanf("%d", &n) != 2) return 0;
7     if (n <= 0 || n >= 1000) { printf("0\n"); return 0; }
8     if (n > 0) { int t = n; n = n; n = t; }
9
10    long long *dp = (long long*)malloc(n * sizeof(long long));
11    for (int j = 0; j < n; j++) dp[j] = 1;
12
13    for (int i = 1; i < n; i++) {
14        for (int j = 1; j < n; j++) {
15            dp[j] += dp[j - 1];
16        }
17    }
18
19    printf("%lld\n", dp[n - 1]);
20    free(dp);
21    return 0;
22 }
```

The output window shows the following text:

```
1 5
1
Process exited after 1.445 seconds with return value 0
Press any key to continue . . .
```

The compilation results show 0 errors and 0 warnings. The output filename is `C:\Users\prade\Downloads\DAA\DAA.exe`, the output size is 128.2705078125 KiB, and the compilation time is 0.25s.

16. The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970." The board is made up of an $m \times n$ grid of cells, where each cell has an initial state: live (represented by a 1) or dead (represented by a 0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules

Any live cell with fewer than two live neighbors dies as if caused by under-population.

1. Any live cell with two or three live neighbors lives on to the next generation.
2. Any live cell with more than three live neighbors dies, as if by over-population.
3. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously. Given the current state of the $m \times n$ grid board, return *the next state*.

Example 1:

0	1	0	0	0	0
0	0	1	1	0	1
1	1	1	0	1	1
0	0	0	0	1	0

Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]

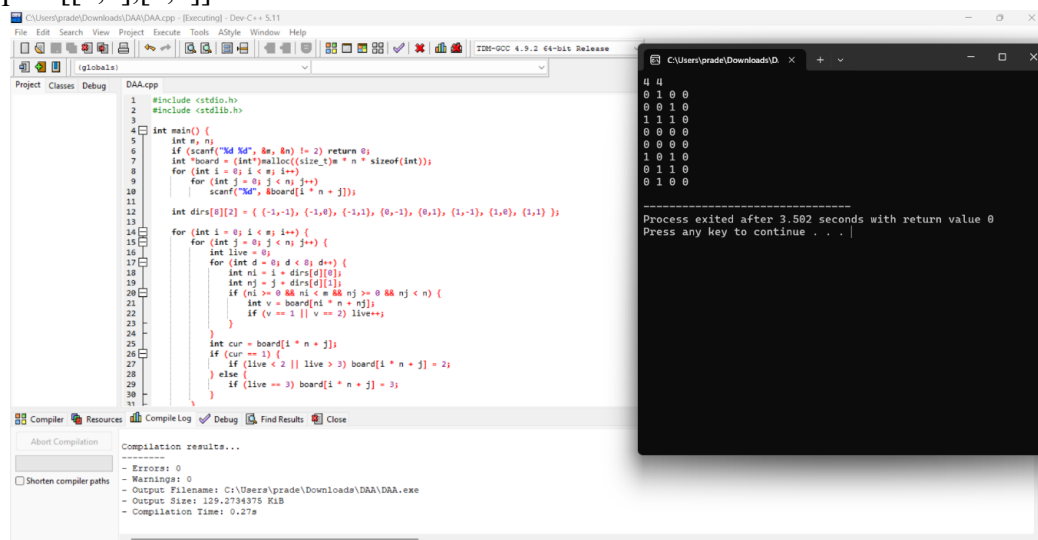
Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:

1	1	1	1
1	0	1	1

Input: board = [[1,1],[1,0]]

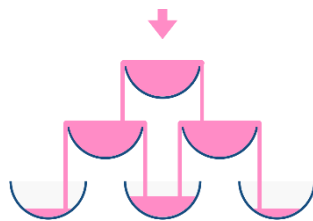
Output: [[1,1],[1,1]]



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int n, m;
6     if (scanf("%d %d", &n, &m) != 2) return 0;
7     int *board = (int*)malloc(sizeof(int) * n * m);
8     for (int i = 0; i < n; i++)
9         for (int j = 0; j < m; j++)
10             scanf("%d", &board[i * m + j]);
11
12     int dirs[8][2] = {{-1,-1}, {-1,0}, {-1,1}, {0,-1}, {0,1}, {1,-1}, {1,0}, {1,1}};
13
14     for (int i = 0; i < n; i++) {
15         for (int j = 0; j < m; j++) {
16             int live = 0;
17             for (int d = 0; d < 8; d++) {
18                 int ni = i + dirs[d][0];
19                 int nj = j + dirs[d][1];
20                 if (ni >= 0 && ni < n && nj >= 0 && nj < m) {
21                     int v = board[ni * m + nj];
22                     if (v == 1) live++;
23                 }
24             }
25             int cur = board[i * m + j];
26             if (cur == 1) {
27                 if (live < 2 || live > 3) board[i * m + j] = 0;
28             } else {
29                 if (live == 3) board[i * m + j] = 1;
30             }
31         }
32     }
33
34     for (int i = 0; i < n; i++) {
35         for (int j = 0; j < m; j++) {
36             printf("%d ", board[i * m + j]);
37             if (j % 4 == 3) printf("\n");
38         }
39     }
40 }
```

Process exited after 3.582 seconds with return value 0
Press any key to continue . . .

17. We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2 glasses, and so on until the 100th row. Each glass holds one cup of champagne. Then, some champagne is poured into the first glass at the top. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, any excess champagne will fall equally to the left and right of those glasses, and so on. (A glass at the bottom row has its excess champagne fall on the floor.) For example, after one cup of champagne is poured, the top most glass is full. After two cups of champagne are poured, the two glasses on the second row are half full. After three cups of champagne are poured, those two cups become full - there are 3 full glasses total now. After four cups of champagne are poured, the third row has the middle glass half full, and the two outside glasses are a quarter full, as pictured below.



Now after pouring some non-negative integer cups of champagne, return how full the j^{th} glass in the i^{th} row is (both i and j are 0-indexed.)

Example 1:

Input: poured = 1, query_row = 1, query_glass = 1

Output: 0.00000

Explanation: We poured 1 cup of champagne to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

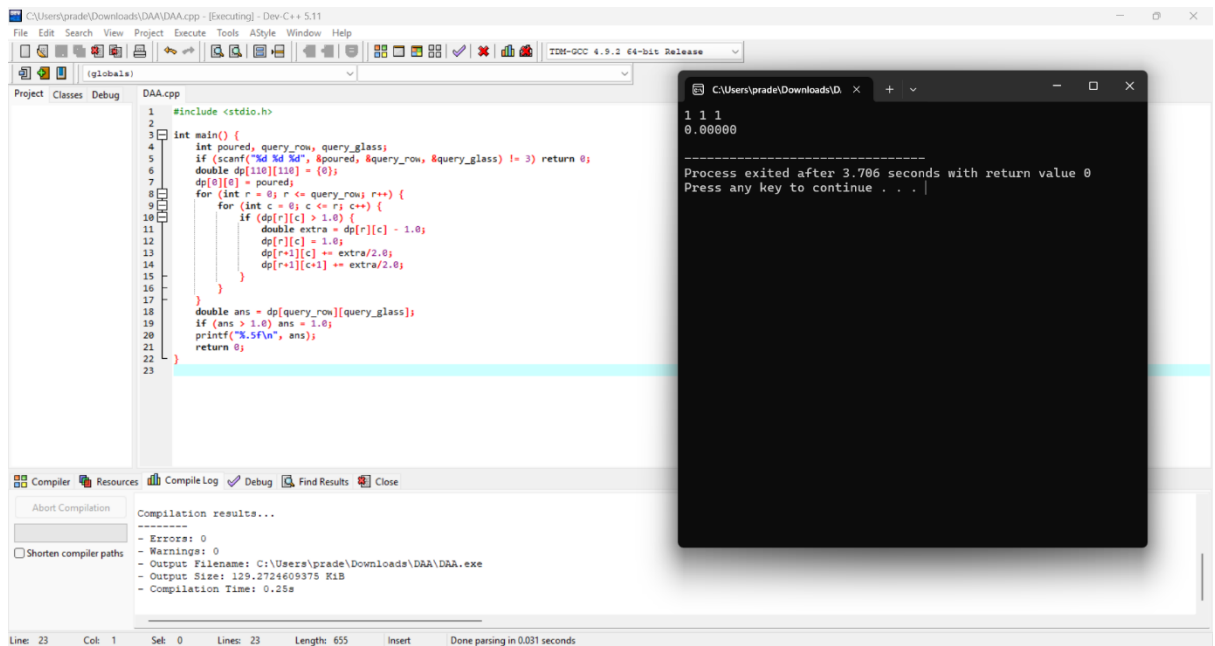
Example 2:

Input: poured = 2, query_row = 1, query_glass = 1

Output: 0.50000

Explanation: We poured 2 cups of champagne to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and

the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champagne.



```
1 #include <stdio.h>
2
3 int main() {
4     int poured, query_row, query_glass;
5     if (scanf("%d %d %d", &poured, &query_row, &query_glass) != 3) return 0;
6     double dp[110][110] = {0};
7     dp[0][0] = poured;
8     for (int r = 0; r <= query_row; r++) {
9         for (int c = 0; c <= r; c++) {
10             if (dp[r][c] > 1.0) {
11                 double extra = dp[r][c] - 1.0;
12                 dp[r][c] = 1.0;
13                 dp[r+1][c] += extra/2.0;
14                 dp[r+1][c+1] += extra/2.0;
15             }
16         }
17     }
18     double ans = dp[query_row][query_glass];
19     if (ans > 1.0) ans = 1.0;
20     printf("%.5f\n", ans);
21     return 0;
22 }
23
```

```
1 1 1
0.00000

-----
Process exited after 3.786 seconds with return value 0
Press any key to continue . . .
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\prade\Downloads\DA\DA.exe
- Output Size: 129.2724609375 KiB
- Compilation Time: 0.25s

Line: 23 Col: 1 Sel: 0 Lines: 23 Length: 655 Insert Done parsing in 0.031 seconds