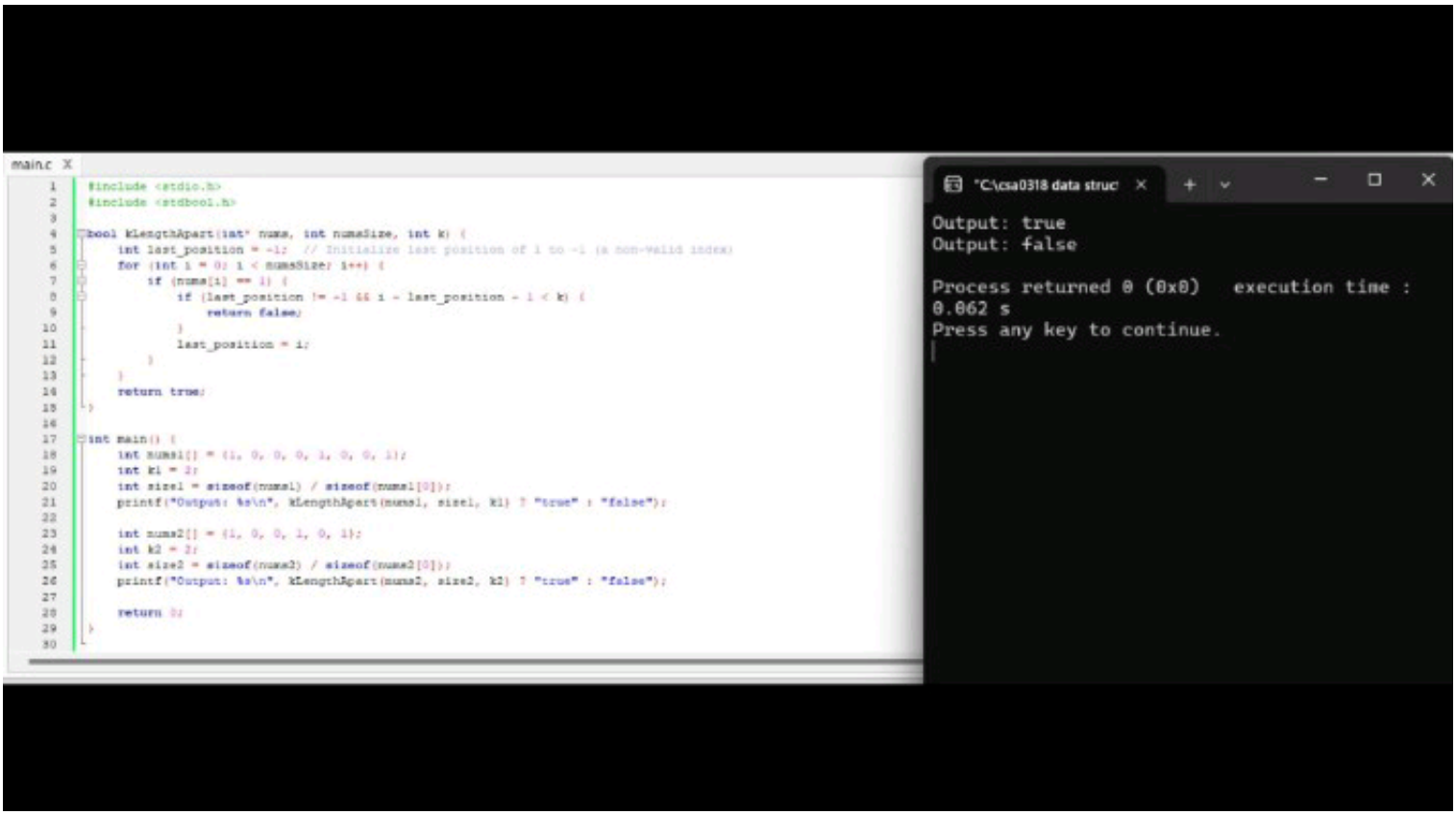
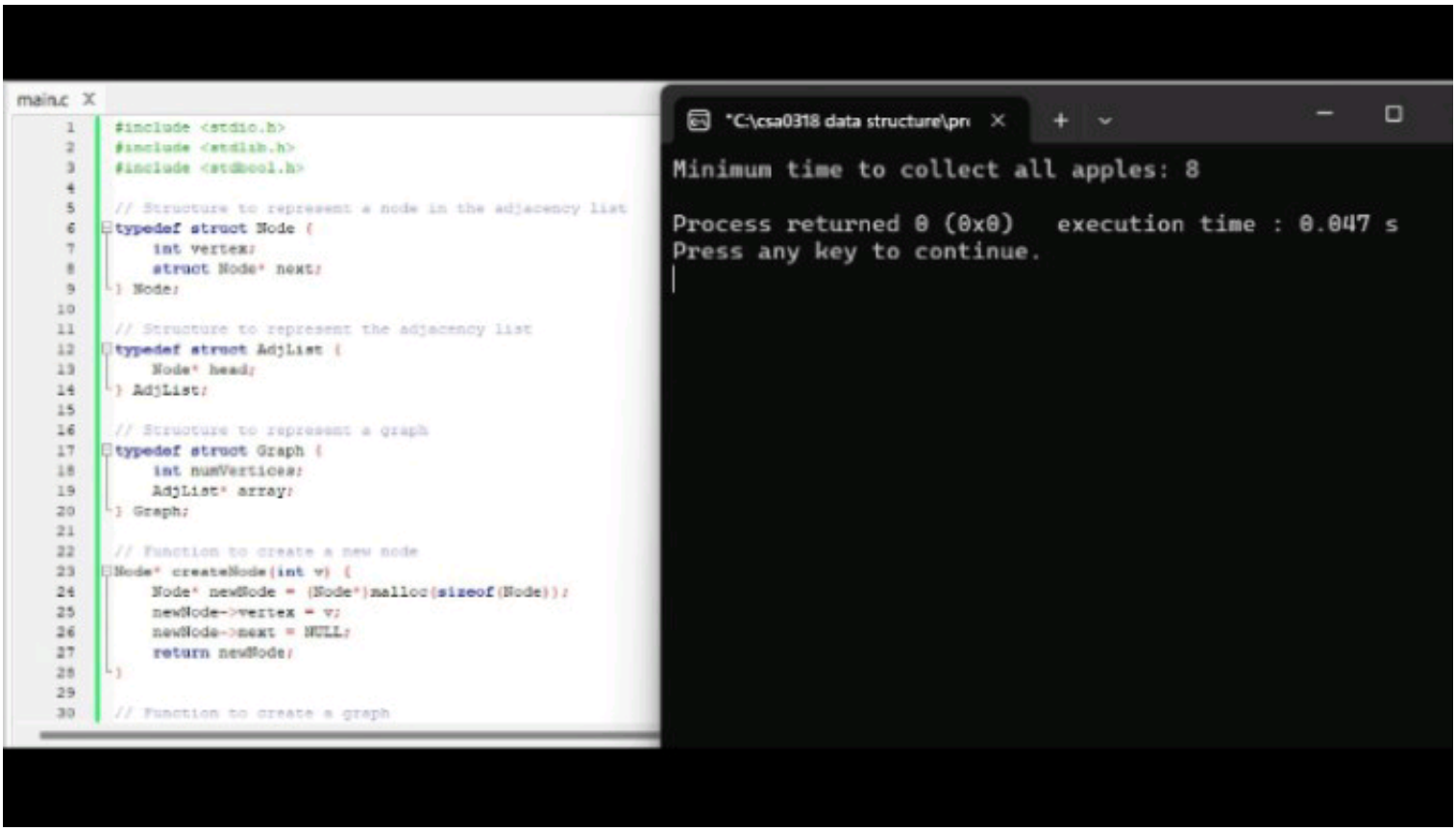
main.c

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

// Function to compare elements for the priority queue
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

// Function to find the kth smallest sum
int kthSmallest(int** mat, int matSize, int* matColSize, int k) {
    // Create a min-heap (priority queue)
    int heapSize = 1;
    for (int i = 0; i < matSize; i++) {
        heapSize *= matColSize[i];
    }
    int* heap = (int*)malloc(heapSize * sizeof(int));
    int* temp = (int*)malloc(heapSize * sizeof(int));

    int size = matColSize[0];
    for (int i = 0; i < size; i++) {
        heap[i] = mat[0][i];
    }

    for (int i = 1; i < matSize; i++) {
        int tempSize = size * matColSize[i];
        for (int j = 0; j < size; j++) {
            for (int l = 0; l < matColSize[i]; l++) {
                temp[j * matColSize[i] + l] = heap[j] + mat[i][l];
            }
```

"C:\csa0318 data structure\pro

```
The 7th smallest sum is: 8

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```


main.c
#include <stdio.h>
#include <stdbool.h>
bool kLengthApart(int* nums, int numsSize, int k) {
int last_position = -1; // Initialize last position of 1 to -1 (a non-valid index)
for (int i = 0; i < numsSize; i++) {
if (nums[i] == 1) {
if (last_position != -1 && i - last_position - 1 < k) {
return false;
}
last_position = i;
}
}
return true;
}
int main() {
int nums1[] = {1, 0, 0, 0, 1, 0, 0, 1};
int k1 = 2;
int size1 = sizeof(nums1) / sizeof(nums1[0]);
printf("Output: %s\n", kLengthApart(nums1, size1, k1) ? "true" : "false");
int nums2[] = {1, 0, 0, 1, 0, 1};
int k2 = 2;
int size2 = sizeof(nums2) / sizeof(nums2[0]);
printf("Output: %s\n", kLengthApart(nums2, size2, k2) ? "true" : "false");
return 0;
}
"C:\csa0318 data struc
Output: true
Output: false
Process returned 0 (0x0) execution time :
0.062 s
Press any key to continue.

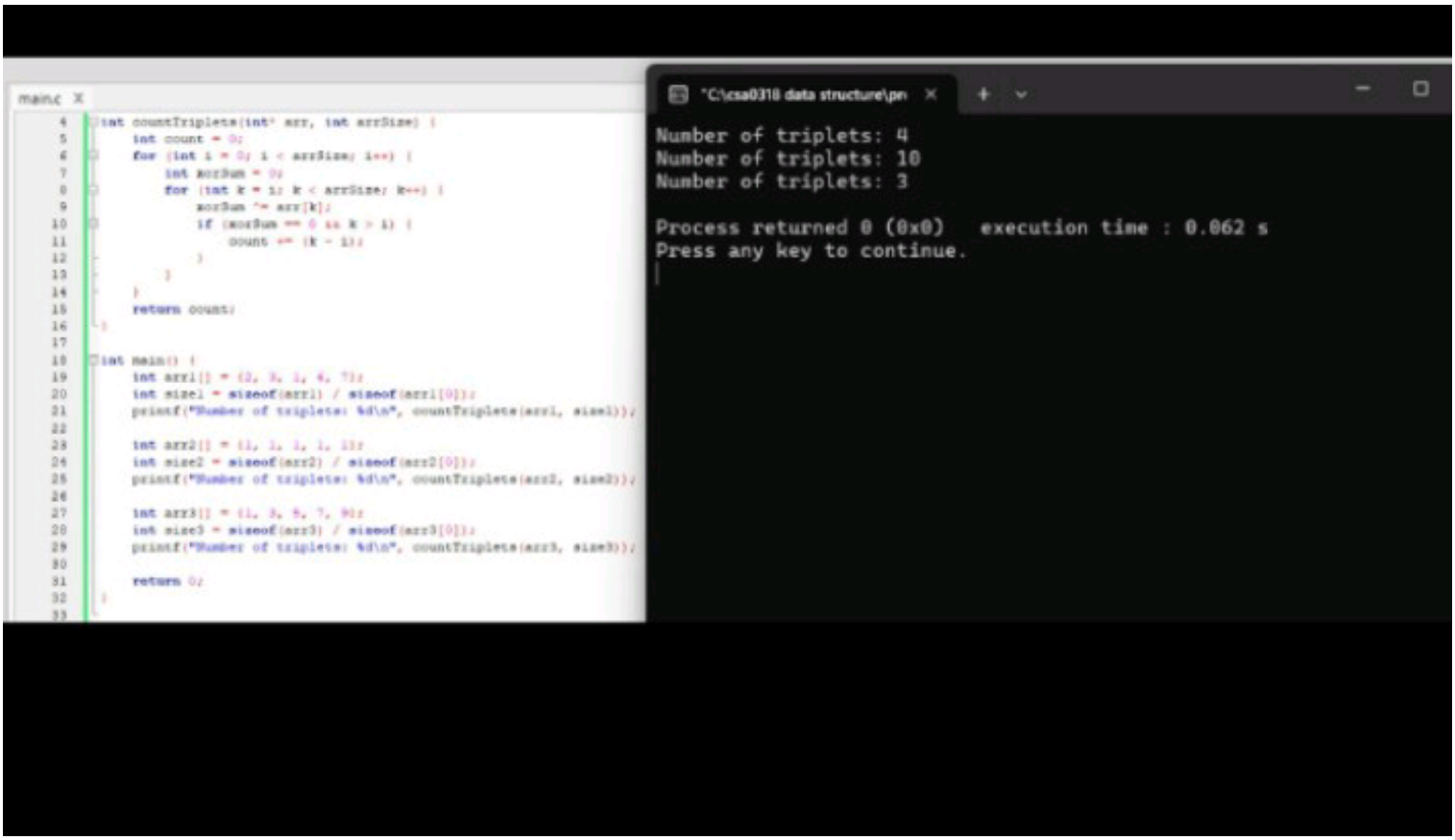

main.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
// Structure to represent a node in the adjacency list
typedef struct Node {
int vertex;
struct Node* next;
} Node;
// Structure to represent the adjacency list
typedef struct AdjList {
Node* head;
} AdjList;
// Structure to represent a graph
typedef struct Graph {
int numVertices;
AdjList* array;
} Graph;
// Function to create a new node
Node* createNode(int v) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->vertex = v;
newNode->next = NULL;
return newNode;
}
// Function to create a graph
"C:\csa0318 data structure\pr
Minimum time to collect all apples: 8
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.

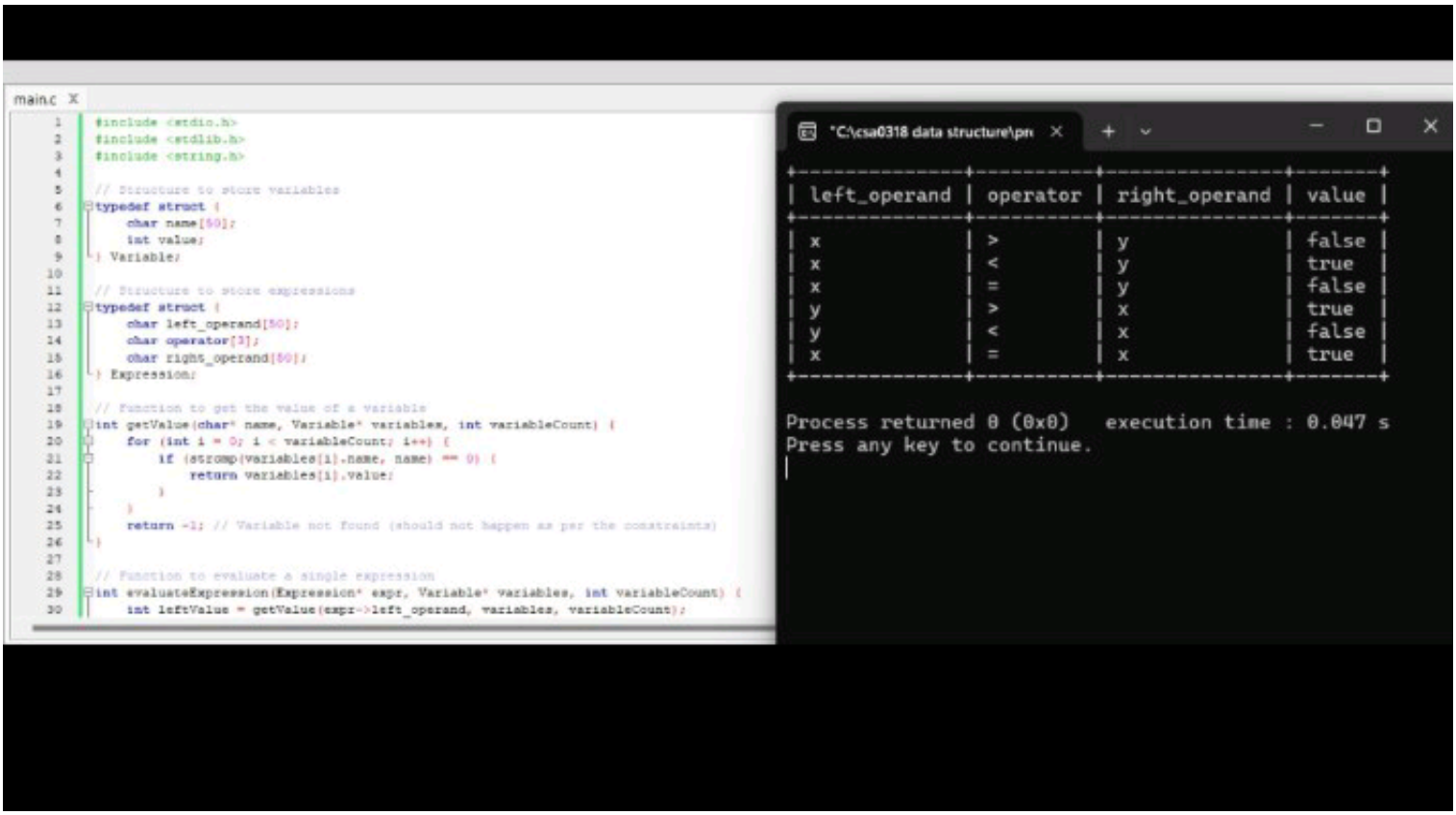
main.c

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

#define MOD 1000000007

// Function to check if there is at least one apple in the given submatrix
int hasApple(int** applePrefixSum, int row1, int col1, int row2, int col2) {
    int apples = applePrefixSum[row1][col1];
    if (row2 > 0) apples -= applePrefixSum[row2 - 1][col1];
    if (col2 > 0) apples -= applePrefixSum[row1][col2 - 1];
    if (row2 > 0 && col2 > 0) apples += applePrefixSum[row2 - 1][col2 - 1];
    return apples > 0;
}

// Function to count the number of ways to cut the pizza
int ways(char** pizza, int pizzaSize, int* pizzaColSize, int k) {
    int rows = pizzaSize;
    int cols = pizzaColSize[0];

    // Create prefix sum array for apples
    int** applePrefixSum = (int**)malloc(rows * sizeof(int*));
    for (int i = 0; i < rows; i++) {
        applePrefixSum[i] = (int*)malloc(cols * sizeof(int));
        for (int j = 0; j < cols; j++) {
            applePrefixSum[i][j] = (pizza[i][j] == 'A' ? 1 : 0);
            if (i > 0) applePrefixSum[i][j] += applePrefixSum[i - 1][j];
            if (j > 0) applePrefixSum[i][j] += applePrefixSum[i][j - 1];
            if (i > 0 && j > 0) applePrefixSum[i][j] -= applePrefixSum[i - 1][j - 1];
        }
```

*C:\csa0318 data struc

```
Number of ways to cut the pizza: 0

Process returned 0 (0x0)   execution time :
0.078 s
Press any key to continue.
```


main.c
int countTriplets(int* arr, int arrSize) {
int count = 0;
for (int i = 0; i < arrSize; i++) {
int xorSum = 0;
for (int k = i; k < arrSize; k++) {
xorSum ^= arr[k];
if (xorSum == 0 && k > i) {
count += (k - i);
}
}
}
return count;
}
int main() {
int arr1[] = {2, 3, 1, 6, 7};
int size1 = sizeof(arr1) / sizeof(arr1[0]);
printf("Number of triplets: %d\n", countTriplets(arr1, size1));
int arr2[] = {1, 1, 1, 1, 1};
int size2 = sizeof(arr2) / sizeof(arr2[0]);
printf("Number of triplets: %d\n", countTriplets(arr2, size2));
int arr3[] = {1, 3, 5, 7, 9};
int size3 = sizeof(arr3) / sizeof(arr3[0]);
printf("Number of triplets: %d\n", countTriplets(arr3, size3));
return 0;
}
"C:\csa0318 data structure\pr
Number of triplets: 4
Number of triplets: 10
Number of triplets: 3
Process returned 0 (0x0) execution time : 0.062 s
Press any key to continue.


main.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Structure to store variables
typedef struct {
char name[50];
int value;
} Variable;
// Structure to store expressions
typedef struct {
char left_operand[50];
char operator[3];
char right_operand[50];
} Expression;
// Function to get the value of a variable
int getValue(char* name, Variable* variables, int variableCount) {
for (int i = 0; i < variableCount; i++) {
if (strcmp(variables[i].name, name) == 0) {
return variables[i].value;
}
}
return -1; // Variable not found (should not happen as per the constraints)
}
// Function to evaluate a single expression
int evaluateExpression(Expression* expr, Variable* variables, int variableCount) {
int leftValue = getValue(expr->left_operand, variables, variableCount);
"C:\csa0318 data structure\pr
left_operand	operator	right_operand	value
x	>	y	false
x	<	y	true
x	=	y	false
y	>	x	true
y	<	x	false
x	=	x	true
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.

main.c

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

// Function to find the longest subarray with absolute difference less than or equal to the
int longestSubarray(int* nums, int numsSize, int limit) {
    // Deques to store the indices of the maximum and minimum elements in the current windo
    int* maxDeque = (int*)malloc(numsSize * sizeof(int));
    int* minDeque = (int*)malloc(numsSize * sizeof(int));
    int maxFront = 0, maxRear = -1;
    int minFront = 0, minRear = -1;

    int left = 0, right = 0, maxLength = 0;

    while (right < numsSize) {
        // Maintain the maxDeque
        while (maxRear >= maxFront && nums[maxDeque[maxRear]] <= nums[right]) {
            maxRear--;
        }
        maxDeque[++maxRear] = right;

        // Maintain the minDeque
        while (minRear >= minFront && nums[minDeque[minRear]] >= nums[right]) {
            minRear--;
        }
        minDeque[++minRear] = right;

        // Check if the current window is valid
        while (nums[maxDeque[maxFront]] - nums[minDeque[minFront]] > limit) {
            left++;
            if (maxDeque[maxFront] < left) {
```

```
The longest subarray length is: 2
The longest subarray length is: 4
The longest subarray length is: 3

Process returned 0 (0x0)   execution time : 0.047
s
Press any key to continue.
```