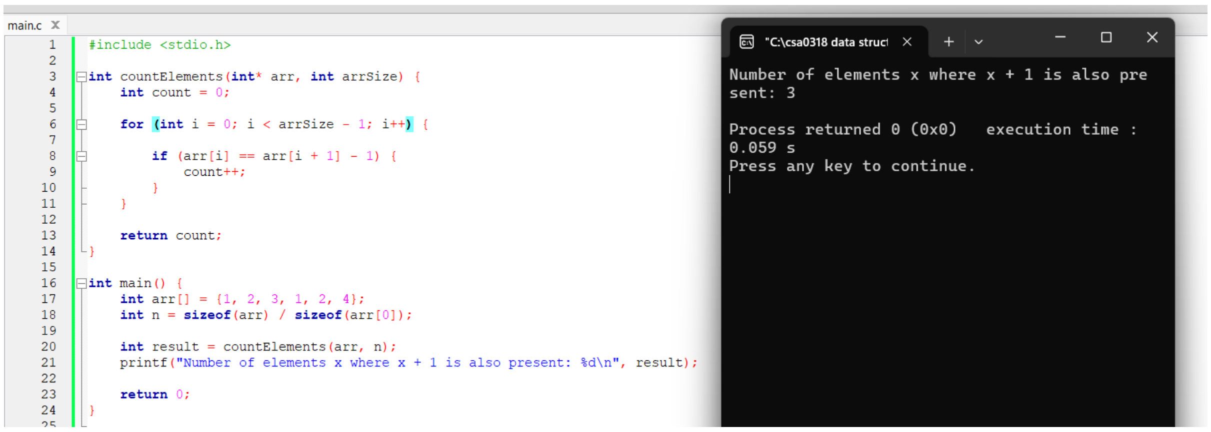
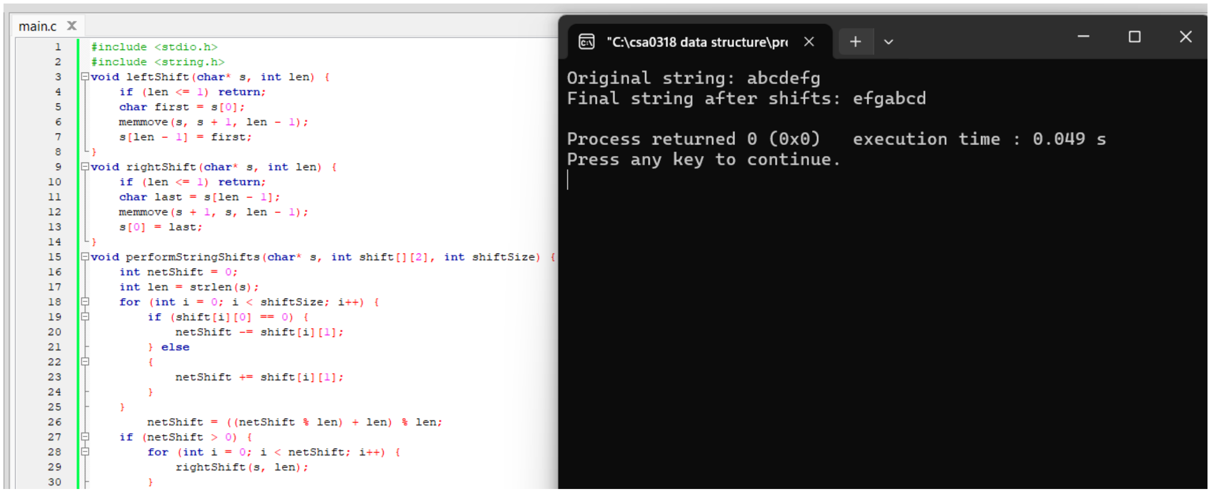
1. You are given a string s, and an array of pairs of indices in the string pairs where pairs[i] = [a, b] indicates 2 indices(0-indexed) of the string.You can swap the characters at any pair of indices in the given pairs any number of times. Return the lexicographically smallest string that s can be changed to after using the swaps.



```c
#include <stdio.h>

int countElements(int* arr, int arrSize) {
    int count = 0;

    for (int i = 0; i < arrSize - 1; i++) {

        if (arr[i] == arr[i + 1] - 1) {
            count++;
        }
    }

    return count;
}

int main() {
    int arr[] = {1, 2, 3, 1, 2, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = countElements(arr, n);
    printf("Number of elements x where x + 1 is also present: %d\n", result);

    return 0;
}
```

```
Number of elements x where x + 1 is also pre
sent: 3

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

2. Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.



```c
#include <stdio.h>
#include <string.h>
void leftShift(char* s, int len) {
    if (len <= 1) return;
    char first = s[0];
    memmove(s, s + 1, len - 1);
    s[len - 1] = first;
}
void rightShift(char* s, int len) {
    if (len <= 1) return;
    char last = s[len - 1];
    memmove(s + 1, s, len - 1);
    s[0] = last;
}
void performStringShifts(char* s, int shift[][2], int shiftSize) {
    int netShift = 0;
    int len = strlen(s);
    for (int i = 0; i < shiftSize; i++) {
        if (shift[i][0] == 0) {
            netShift -= shift[i][1];
        } else
        {
            netShift += shift[i][1];
        }
    }
        netShift = ((netShift % len) + len) % len;
    if (netShift > 0) {
        for (int i = 0; i < netShift; i++) {
            rightShift(s, len);
        }
```

```
Original string: abcdefg
Final string after shifts: efgabcd

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

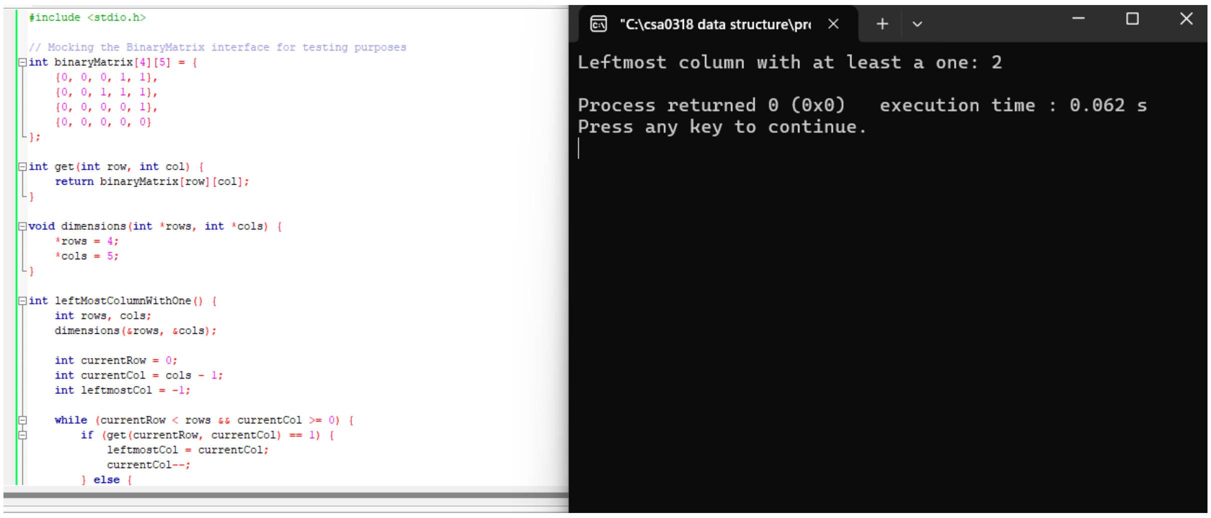
3. You are given a string s. s[i] is either a lowercase English letter or '?'. For a string t having length m containing only lowercase English letters, we define the function cost(i) for an index i as the number of characters equal to t[i] that appeared before it, i.e. in the range [0, i - 1]. The value of t is the sum of cost(i) for all indices i. For example, for the string t = "aab":

cost(0) = 0

cost(1) = 1

cost(2) = 0

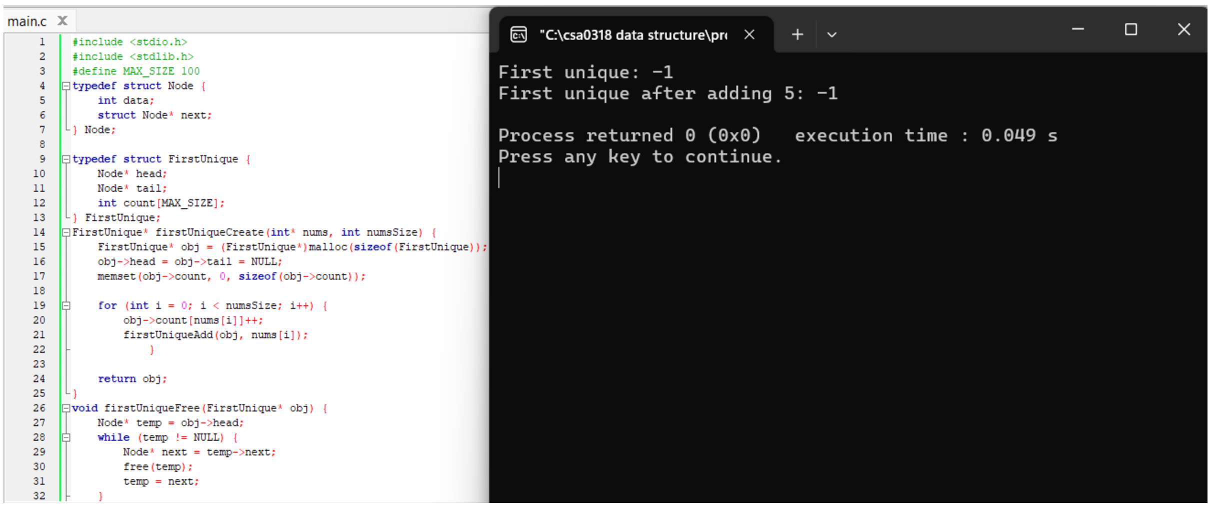
Hence, the value of "aab" is 0 + 1 + 0 = 1. Your task is to replace all occurrences of '?' in s with any lowercase English letter so at the value of s is minimized.



```c
#include <stdio.h>

// Mocking the BinaryMatrix interface for testing purposes
int binaryMatrix[4][5] = {
    {0, 0, 0, 1, 1},
    {0, 0, 1, 1, 1},
    {0, 0, 0, 0, 1},
    {0, 0, 0, 0, 0}
};

int get(int row, int col) {
    return binaryMatrix[row][col];
}

void dimensions(int *rows, int *cols) {
    *rows = 4;
    *cols = 5;
}

int leftMostColumnWithOne() {
    int rows, cols;
    dimensions(&rows, &cols);

    int currentRow = 0;
    int currentCol = cols - 1;
    int leftmostCol = -1;

    while (currentRow < rows && currentCol >= 0) {
        if (get(currentRow, currentCol) == 1) {
            leftmostCol = currentCol;
            currentCol--;
        } else {
```

```
"C:\csa0318 data structure\pr

Leftmost column with at least a one: 2

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

4. You are given a string s. Consider performing the following operation until s becomes empty: For every alphabet character from 'a' to 'z', remove the first occurrence of that character in s (if it exists). For example, let initially s = "aabcbbca". We do the following operations: Remove the underlined characters s = "aabcbbca". The resulting string is s = "abbca". Remove the underlined characters s = "abbca". The resulting string is s = "ba". Remove the underlined characters s = "ba". The resulting string is s = "". Return the value of the string s right before applying the last operation. In the example above, answer is "ba".



main.c

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct FirstUnique {
    Node* head;
    Node* tail;
    int count[MAX_SIZE];
} FirstUnique;
FirstUnique* firstUniqueCreate(int* nums, int numsSize) {
    FirstUnique* obj = (FirstUnique*)malloc(sizeof(FirstUnique));
    obj->head = obj->tail = NULL;
    memset(obj->count, 0, sizeof(obj->count));

    for (int i = 0; i < numsSize; i++) {
        obj->count[nums[i]]++;
        firstUniqueAdd(obj, nums[i]);
            }

    return obj;
}
void firstUniqueFree(FirstUnique* obj) {
    Node* temp = obj->head;
    while (temp != NULL) {
        Node* next = temp->next;
        free(temp);
        temp = next;
    }
```

```
"C:\csa0318 data structure\pr

First unique: -1
First unique after adding 5: -1

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

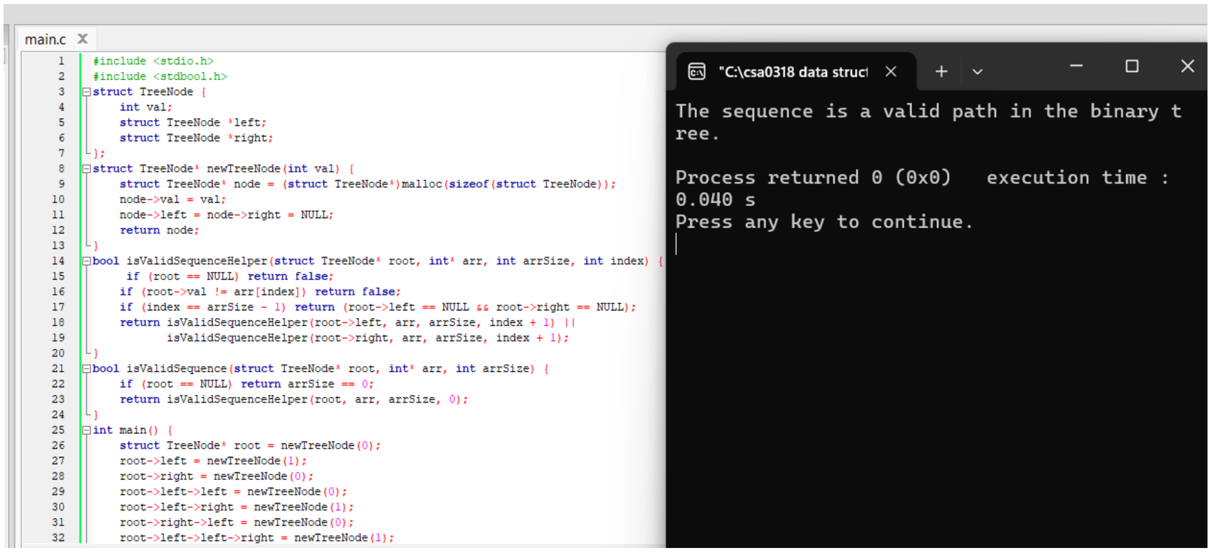
5. Given an integer array nums, find the subarray with the largest sum, and return its sum.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6.



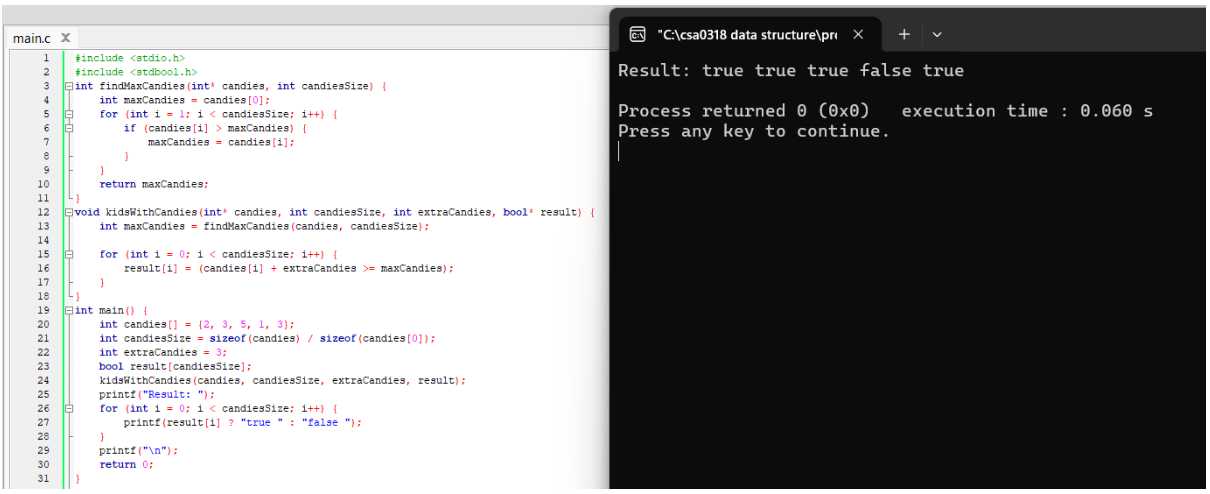
main.c

```c
#include <stdio.h>
#include <stdbool.h>
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* newTreeNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}
bool isValidSequenceHelper(struct TreeNode* root, int* arr, int arrSize, int index) {
     if (root == NULL) return false;
    if (root->val != arr[index]) return false;
    if (index == arrSize - 1) return (root->left == NULL && root->right == NULL);
    return isValidSequenceHelper(root->left, arr, arrSize, index + 1) ||
           isValidSequenceHelper(root->right, arr, arrSize, index + 1);
}
bool isValidSequence(struct TreeNode* root, int* arr, int arrSize) {
    if (root == NULL) return arrSize == 0;
    return isValidSequenceHelper(root, arr, arrSize, 0);
}
int main() {
    struct TreeNode* root = newTreeNode(0);
    root->left = newTreeNode(1);
    root->right = newTreeNode(0);
    root->left->left = newTreeNode(0);
    root->left->right = newTreeNode(1);
    root->right->left = newTreeNode(0);
    root->left->left->right = newTreeNode(1);
```

"C:\csa0318 data struct

```
The sequence is a valid path in the binary t
ree.

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

6. You are given an integer array nums with no duplicates. A maximum binary tree can be built recursively from nums using the following algorithm: Create a root node whose value is the maximum value in nums. Recursively build the left subtree on the subarray prefix to the left of the maximum value. Recursively build the right subtree on the subarray suffix to the right of the maximum value. Return the maximum binary tree built from nums.



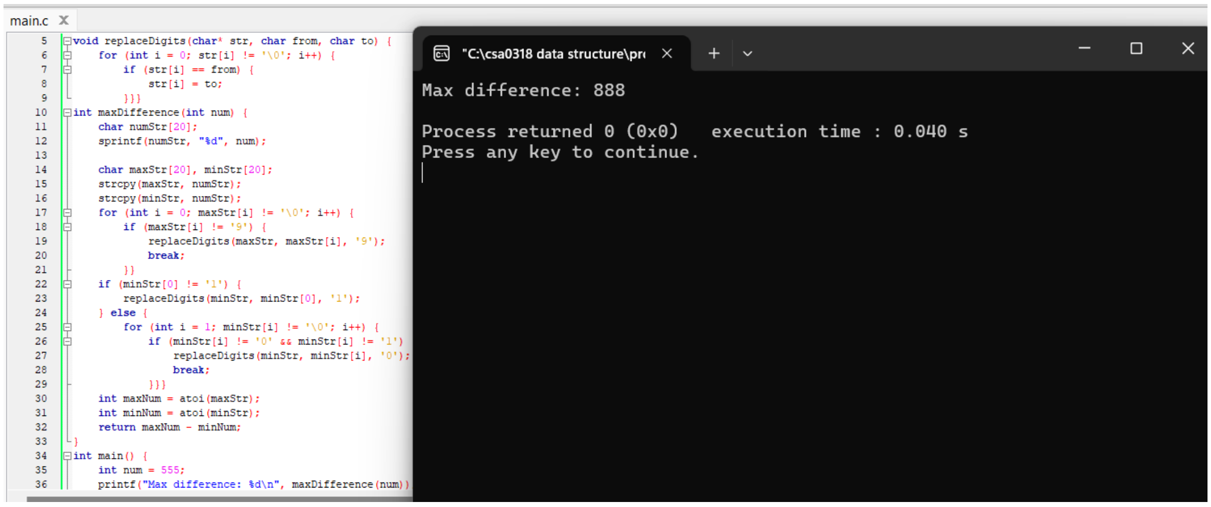
main.c

```c
#include <stdio.h>
#include <stdbool.h>
int findMaxCandies(int* candies, int candiesSize) {
    int maxCandies = candies[0];
    for (int i = 1; i < candiesSize; i++) {
        if (candies[i] > maxCandies) {
            maxCandies = candies[i];
        }
    }
    return maxCandies;
}
void kidsWithCandies(int* candies, int candiesSize, int extraCandies, bool* result) {
    int maxCandies = findMaxCandies(candies, candiesSize);

    for (int i = 0; i < candiesSize; i++) {
        result[i] = (candies[i] + extraCandies >= maxCandies);
    }
}
int main() {
    int candies[] = {2, 3, 5, 1, 3};
    int candiesSize = sizeof(candies) / sizeof(candies[0]);
    int extraCandies = 3;
    bool result[candiesSize];
    kidsWithCandies(candies, candiesSize, extraCandies, result);
    printf("Result: ");
    for (int i = 0; i < candiesSize; i++) {
        printf(result[i] ? "true " : "false ");
    }
    printf("\n");
    return 0;
}
```

"C:\csa0318 data structure\pr

```
Result: true true true false true

Process returned 0 (0x0)   execution time : 0.060 s
Press any key to continue.
```

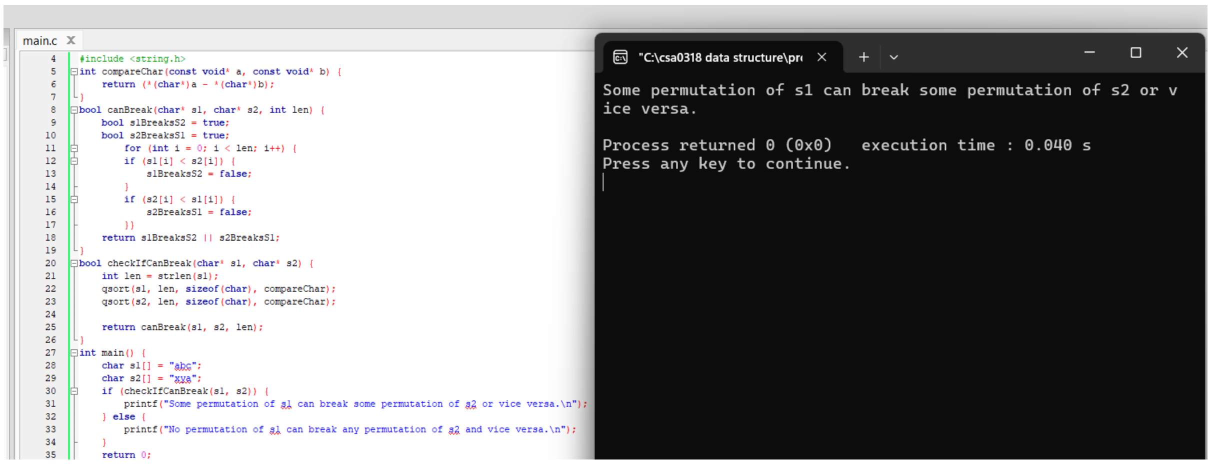
7.Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums.A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].A subarray may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.



```c
void replaceDigits(char* str, char from, char to) {
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == from) {
            str[i] = to;
        }}}
int maxDifference(int num) {
    char numStr[20];
    sprintf(numStr, "%d", num);

    char maxStr[20], minStr[20];
    strcpy(maxStr, numStr);
    strcpy(minStr, numStr);
    for (int i = 0; maxStr[i] != '\0'; i++) {
        if (maxStr[i] != '9') {
            replaceDigits(maxStr, maxStr[i], '9');
            break;
        }}
    if (minStr[0] != '1') {
        replaceDigits(minStr, minStr[0], '1');
    } else {
        for (int i = 1; minStr[i] != '\0'; i++) {
            if (minStr[i] != '0' && minStr[i] != '1')
                replaceDigits(minStr, minStr[i], '0');
                break;
            }}}
    int maxNum = atoi(maxStr);
    int minNum = atoi(minStr);
    return maxNum - minNum;
}
int main() {
    int num = 555;
    printf("Max difference: %d\n", maxDifference(num))
```

```
Max difference: 888

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

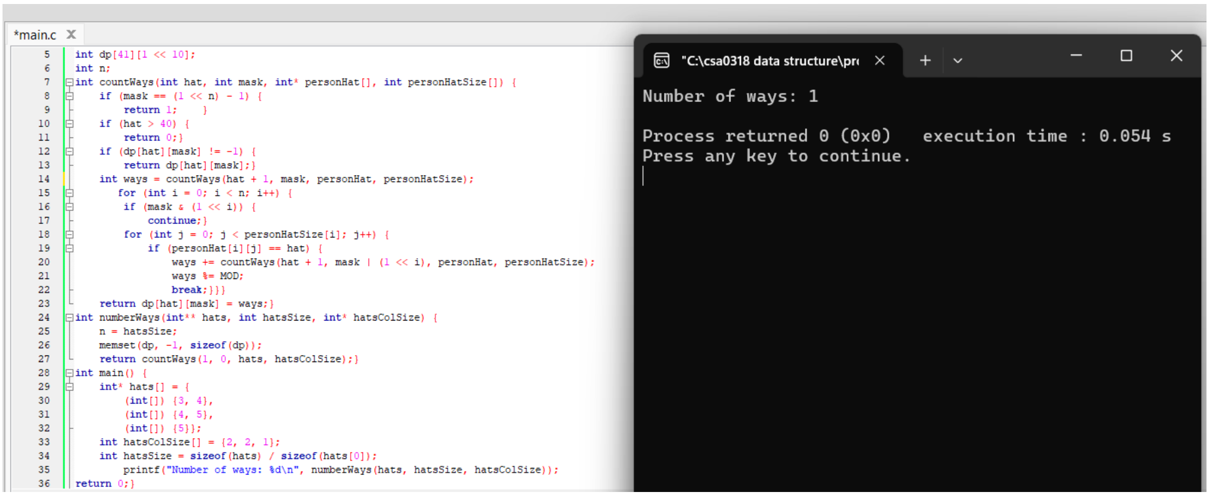
8. You are given an array nums consisting of integers. You are also given a 2D array queries, where queries[i] = [posi, xi].For query i, we first set nums[posi] equal to xi, then we calculate the answer to query i which is the maximum sum of a subsequence of nums where no two adjacent elements are selected. Return the sum of the answers to all queries. Since the final answer may be very large, return it modulo 109 + 7. A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.



```c
#include <string.h>
int compareChar(const void* a, const void* b) {
    return (*(char*)a - *(char*)b);
}
bool canBreak(char* s1, char* s2, int len) {
    bool s1BreaksS2 = true;
    bool s2BreaksS1 = true;
        for (int i = 0; i < len; i++) {
        if (s1[i] < s2[i]) {
            s1BreaksS2 = false;
        }
        if (s2[i] < s1[i]) {
            s2BreaksS1 = false;
        }}
    return s1BreaksS2 || s2BreaksS1;
}
bool checkIfCanBreak(char* s1, char* s2) {
    int len = strlen(s1);
    qsort(s1, len, sizeof(char), compareChar);
    qsort(s2, len, sizeof(char), compareChar);

    return canBreak(s1, s2, len);
}
int main() {
    char s1[] = "abc";
    char s2[] = "xya";
    if (checkIfCanBreak(s1, s2)) {
        printf("Some permutation of s1 can break some permutation of s2 or vice versa.\n");
    } else {
        printf("No permutation of s1 can break any permutation of s2 and vice versa.\n");
    }
    return 0;
```

```
Some permutation of s1 can break some permutation of s2 or v
ice versa.

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

9. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2). You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).



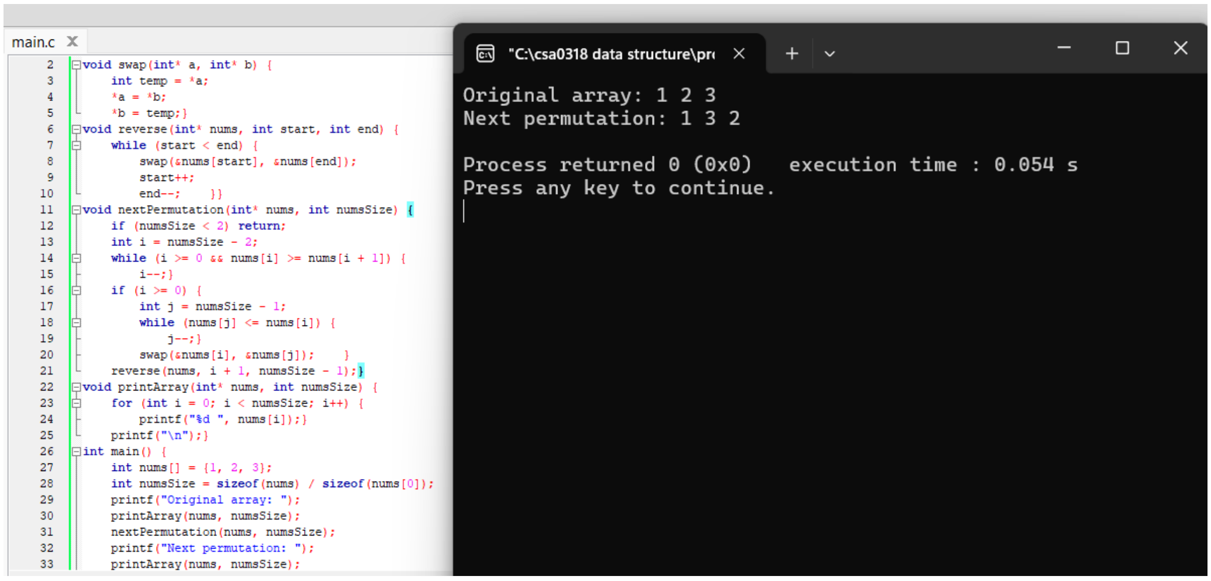
*main.c

```c
int dp[41][1 << 10];
int n;
int countWays(int hat, int mask, int* personHat[], int personHatSize[]) {
    if (mask == (1 << n) - 1) {
        return 1;    }
    if (hat > 40) {
        return 0;}
    if (dp[hat][mask] != -1) {
        return dp[hat][mask];}
    int ways = countWays(hat + 1, mask, personHat, personHatSize);
       for (int i = 0; i < n; i++) {
        if (mask & (1 << i)) {
            continue;}
        for (int j = 0; j < personHatSize[i]; j++) {
            if (personHat[i][j] == hat) {
                ways += countWays(hat + 1, mask | (1 << i), personHat, personHatSize);
                ways %= MOD;
                break;}}}
    return dp[hat][mask] = ways;}
int numberWays(int** hats, int hatsSize, int* hatsColSize) {
    n = hatsSize;
    memset(dp, -1, sizeof(dp));
    return countWays(1, 0, hats, hatsColSize);}
int main() {
    int* hats[] = {
        (int[]) {3, 4},
        (int[]) {4, 5},
        (int[]) {5}};
    int hatsColSize[] = {2, 2, 1};
    int hatsSize = sizeof(hats) / sizeof(hats[0]);
        printf("Number of ways: %d\n", numberWays(hats, hatsSize, hatsColSize));
return 0;}
```

"C:\csa0318 data structure\pr

```
Number of ways: 1

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

10. Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).



main.c

```c
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;}
void reverse(int* nums, int start, int end) {
    while (start < end) {
        swap(&nums[start], &nums[end]);
        start++;
        end--;    }}
void nextPermutation(int* nums, int numsSize) {
    if (numsSize < 2) return;
    int i = numsSize - 2;
    while (i >= 0 && nums[i] >= nums[i + 1]) {
        i--;}
    if (i >= 0) {
        int j = numsSize - 1;
        while (nums[j] <= nums[i]) {
            j--;}
        swap(&nums[i], &nums[j]);    }
    reverse(nums, i + 1, numsSize - 1);}
void printArray(int* nums, int numsSize) {
    for (int i = 0; i < numsSize; i++) {
        printf("%d ", nums[i]);}
    printf("\n");}
int main() {
    int nums[] = {1, 2, 3};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    printf("Original array: ");
    printArray(nums, numsSize);
    nextPermutation(nums, numsSize);
    printf("Next permutation: ");
    printArray(nums, numsSize);
```

"C:\csa0318 data structure\pr

```
Original array: 1 2 3
Next permutation: 1 3 2

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```