**TPEC – Solutions:**

**M4:**

# 1D\_arrays\_in\_c:

# #include <stdio.h>

# #include <string.h>

# #include <math.h>

# #include <stdlib.h>

# int main() {

# int n;

# 

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

# n++;

# int\* arr=(int\*)malloc(n \* sizeof(int));

# arr[0]=0;

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

# scanf("%d",arr+i);

# arr[0]+=arr[i];

# if(i==n-1)

# printf("%d",arr[0]);

# }

# free(arr);

# 

# return 0;

# }

1. **BST\_Insertion:**

struct node\* creatNode(int data){

struct node\* root = (struct node\*)malloc(sizeof(struct node));

root->data = data;

root->left = NULL;

root->right = NULL;

return root;

}

struct node\* insert( struct node\* root, int data ) {

if(root == NULL){

struct node\* tree = creatNode(data);

return tree;

}else{

struct node\* cur;

if(data <= root->data){

cur = insert(root->left,data);

root->left = cur;

}else{

cur = insert(root->right,data);

root->right = cur;

}

}

return root;

}

1. **Array reversal**

#include <stdio.h>

#include <stdlib.h>

int main()

{int num, \*arr, i;

scanf("%d", &num);

arr = (int\*) malloc(num \* sizeof(int));

for(i = 0; i < num; i++) {

scanf("%d", arr + i);

}

/\* Write the logic to reverse the array. \*/

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

arr[i]= arr[i] + arr[num-(i+1)];

arr[num-(i+1)] = arr[i] - arr[num-(i+1)];

arr[i] = arr[i] - arr[num-(i+1)];

}

for(i=0; i<num; i++)printf("%d ",arr[i]);return 0;

}

1. **Remove duplicates from sorted array:**

int removeDuplicates(int\* nums, int numsSize) {

// Edge case: if the array is empty, return 0

if (numsSize == 0) {

return 0;

}

// 'k' will track the number of unique elements

int k = 1; // The first element is always unique

// Iterate through the array starting from the second element

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

// If the current element is not equal to the previous one, it's unique

if (nums[i] != nums[i - 1]) {

// Assign the unique element to the next available position

nums[k] = nums[i];

// Increment 'k' to move the index forward for the next unique element

k++;

}

}

// Return the number of unique elements

return k;

}

1. **Sorting array of strings:**

int lexicographic\_sort(const char\* a, const char\* b) {

return strcmp(a, b);

}

int lexicographic\_sort\_reverse(const char\* a, const char\* b) {

return strcmp(b, a);

}

int sort\_by\_number\_of\_distinct\_characters(const char\* a, const char\* b) {

int count\_a = 0, count\_b = 0;

int char\_count[30] = {0};

for (const char\* p = a; \*p; p++) {

if (!char\_count[\*p - 'a']) {

char\_count[\*p - 'a'] = 1;

count\_a++;

}

}

memset(char\_count, 0, sizeof(char\_count));

for (const char\* p = b; \*p != '\0'; p++) {

if (!char\_count[\*p - 'a']) {

char\_count[\*p - 'a'] = 1;

count\_b++;

}

}

if (count\_a == count\_b) {

return strcmp(a, b);

} else {

return count\_a - count\_b;

}

}

int sort\_by\_length(const char\* a, const char\* b) {

int len\_a = strlen(a);

int len\_b = strlen(b);

if (len\_a == len\_b) {

return strcmp(a, b);

} else {

return len\_a - len\_b;

}

}

void string\_sort(char\*\* arr,const int len,int (\*cmp\_func)(const char\* a, const char\* b)){

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

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

if (cmp\_func(arr[i], arr[j]) > 0) {

char\* temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

**M5:**

1. **2D Array DS:**

#include <stdio.h>

#include <limits.h>

// Function to calculate the hourglass sum

int hourglassSum(int arr[6][6]) {

int hourGlassMax = INT\_MIN;

int hourGlassCurrent = 0;

// Iterate over each possible hourglass

for (int x = 0; x < 4; x++) { // 4 is the number of valid row positions for the top of the hourglass

for (int y = 0; y < 4; y++) { // 4 is the number of valid column positions for the left of the hourglass

// Calculate the sum of the current hourglass

hourGlassCurrent = arr[x][y] + arr[x][y + 1] + arr[x][y + 2] +

arr[x + 1][y + 1] +

arr[x + 2][y] + arr[x + 2][y + 1] + arr[x + 2][y + 2];

// Update the maximum hourglass sum

if (hourGlassCurrent > hourGlassMax) {

hourGlassMax = hourGlassCurrent;

}

}

}

return hourGlassMax;

}

int main() {

// Declare a 6x6 array

int arr[6][6];

// Read input for the 6x6 array

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

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

}

}

// Call the hourglassSum function and store the result

int result = hourglassSum(arr);

// Output the result

printf("%d\n", result);

return 0;

}

1. **Dynamic array**

#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

int\* dynamicArray(int n, int queries\_rows, int queries\_columns, int\*\* queries, int\* result\_count) {

int lastAnswer = 0;

\*result\_count = 0; // To count how many results we will return

int\* result = malloc(queries\_rows \* sizeof(int)); // Allocate space for storing results

// Create an array of n sequences (lists)

int\*\* arr = malloc(n \* sizeof(int\*));

int\* sizes = malloc(n \* sizeof(int)); // This will store the size of each sequence (list)

// Initialize arrays

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

arr[i] = malloc(100 \* sizeof(int)); // Allocate an initial size for each sequence

sizes[i] = 0; // Initially, the size of each sequence is 0

}

// Process the queries

for (int i = 0; i < queries\_rows; i++) {

int type = queries[i][0];

int x = queries[i][1];

int y = queries[i][2];

int idx = (x ^ lastAnswer) % n;

if (type == 1) {

// Type 1: Append y to arr[idx]

arr[idx][sizes[idx]++] = y; // Increment the size after appending

} else {

// Type 2: Set lastAnswer

lastAnswer = arr[idx][y % sizes[idx]];

result[\*result\_count] = lastAnswer; // Store the result

(\*result\_count)++; // Increment the count of results

}

}

// Free the dynamically allocated memory for sizes and arr

free(sizes);

return result;

}

int main() {

FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

int n = parse\_int(\*(first\_multiple\_input + 0));

int q = parse\_int(\*(first\_multiple\_input + 1));

int\*\* queries = malloc(q \* sizeof(int\*));

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

\*(queries + i) = malloc(3 \* (sizeof(int)));

char\*\* queries\_item\_temp = split\_string(rtrim(readline()));

for (int j = 0; j < 3; j++) {

int queries\_item = parse\_int(\*(queries\_item\_temp + j));

\*(\*(queries + i) + j) = queries\_item;

}

}

int result\_count;

int\* result = dynamicArray(n, q, 3, queries, &result\_count);

for (int i = 0; i < result\_count; i++) {

fprintf(fptr, "%d", \*(result + i));

if (i != result\_count - 1) {

fprintf(fptr, "\n");

}

}

fprintf(fptr, "\n");

fclose(fptr);

return 0;

}

char\* readline() {

size\_t alloc\_length = 1024;

size\_t data\_length = 0;

char\* data = malloc(alloc\_length);

while (true) {

char\* cursor = data + data\_length;

char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

if (!line) {

break;

}

data\_length += strlen(cursor);

if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

break;

}

alloc\_length <<= 1;

data = realloc(data, alloc\_length);

if (!data) {

data = '\0';

break;

}

}

if (data[data\_length - 1] == '\n') {

data[data\_length - 1] = '\0';

data = realloc(data, data\_length);

if (!data) {

data = '\0';

}

} else {

data = realloc(data, data\_length + 1);

if (!data) {

data = '\0';

} else {

data[data\_length] = '\0';

}

}

return data;

}

char\* ltrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

while (\*str != '\0' && isspace(\*str)) {

str++;

}

return str;

}

char\* rtrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

char\* end = str + strlen(str) - 1;

while (end >= str && isspace(\*end)) {

end--;

}

\*(end + 1) = '\0';

return str;

}

char\*\* split\_string(char\* str) {

char\*\* splits = NULL;

char\* token = strtok(str, " ");

int spaces = 0;

while (token) {

splits = realloc(splits, sizeof(char\*) \* ++spaces);

if (!splits) {

return splits;

}

splits[spaces - 1] = token;

token = strtok(NULL, " ");

}

return splits;

}

int parse\_int(char\* str) {

char\* endptr;

int value = strtol(str, &endptr, 10);

if (endptr == str || \*endptr != '\0') {

exit(EXIT\_FAILURE);

}

return value;

}

1. **Find index of first occurrence leetcode:**

// Function to find the first occurrence of needle in haystack

int strStr(char \*haystack, char \*needle) {

// If needle is an empty string, return 0

if (\*needle == '\0') {

return 0;

}

// Traverse haystack to find the first occurrence of needle

for (int i = 0; haystack[i] != '\0'; i++) {

// If the remaining part of haystack is shorter than needle, break early

if (strlen(haystack + i) < strlen(needle)) {

break;

}

// Compare substring starting from haystack[i] with needle

int j = 0;

while (haystack[i + j] == needle[j] && needle[j] != '\0') {

j++;

}

// If we found a match, return the starting index

if (needle[j] == '\0') {

return i;

}

}

// If no match found, return -1

return -1;

}

1. **Permutations of strings**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int next\_permutation(int n, char \*\*s)

{

// Step 1: Find the largest index k such that s[k] < s[k + 1]

int k = -1;

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

if (strcmp(s[i], s[i + 1]) < 0) {

k = i;

}

}

if (k == -1) {

return 0; // No next permutation

}

// Step 2: Find the largest index l greater than k such that s[k] < s[l]

int l = -1;

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

if (strcmp(s[k], s[i]) < 0) {

l = i;

}

}

// Step 3: Swap the value of s[k] with that of s[l]

char \*temp = s[k];

s[k] = s[l];

s[l] = temp;

// Step 4: Reverse the sequence from s[k + 1] to s[n - 1]

for (int i = k + 1, j = n - 1; i < j; i++, j--) {

temp = s[i];

s[i] = s[j];

s[j] = temp;

}

return 1; // Next permutation exists

}

1. **Printing tokens**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int main() {

char \*s;

s = malloc(1024 \* sizeof(char));

scanf("%[^\n]", s);

s = realloc(s, strlen(s) + 1);

while(\*s != '\0')

{

if(\*s == ' ')

{

printf("\n");

}

else

{

printf("%c",\*s);

}

s++;

} //Write your logic to print the tokens of the sentence here.

return 0;

}