**LAB 1 - (3)Qns - Array Qn - Pointer**

**Qn - Structures**

💡

1)

Array Question

Given an array arr[] of integers. Find a peak element i.e., an element that is not smaller

than its neighbours. Note: For corner elements, we need to consider only one neighbor.

The Given question have only one peak element. Corner Case: {10, 20, 30, 40, 50}.

Output is 4 Corner Case: {100, 80, 60, 50, 20}. Output is 0

**Input Format**

First Line of input is number of elements in array Next n line contains the elements of

the arrays.

**Constraints**

For corner elements, we need to consider only one neighbor.

**Output Format**

The program prints the neighbour element.

**Sample Input 0**

5

10

20 30 40

50

**Sample Output 0**

4

**Sample Input 1**

5

12

29 18 3

2

**Sample Output 1**

1

**Sample Input 2**

7

22

24 32 36 35 31

20

**Sample Output 2**

3

Answer :

#include <stdio.h>

int FindingPeakElement(int arr[], int n) {

if (n == 1) {

return 0;

}

if (arr[0] >= arr[1]) {

return 0;

}

if (arr[n - 1] >= arr[n - 2]) {

return n - 1;

}

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

if (arr[i] >= arr[i - 1] && arr[i] >= arr[i + 1]) {

return i;

}

}

return -1;

}

int main() {

int n;

scanf("%d", &n);

int arr[n];

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

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

}

int peak\_index = FindingPeakElement(arr, n);

printf("%d\n", peak\_index);

return 0;

}

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

void reverseWords(char \*s) {

int len = strlen(s);

💡

2)

Pointer Question

Given an input string s, reverse the order of the words. Return a string of the words in

reverse order concatenated by a single space. Note that s may contain leading or

multiple spaces between two words. The returned string should only have a single

space separating the words. Do not include any extra spaces.

>

**Input Format**

Input is a string. The input string can include letters, digits, symbols, and spaces.

>

**Constraints**

Note that s may contain leading or multiple spaces between two words. The returned

string should only have a single space separating the words. Do not include any extra

spaces.

>

**Output Format**

The program outputs the input string with the order of words reversed. (Print)

**Sample Input 0**

Testing multiple spaces

**Sample Output 0**

spaces multiple Testing

**Sample Input 1**

one two three

**Sample Output 1**

three two one

**Sample Input 2**

Programming is fun!

**Sample Output 2**

fun! is Programming

**Sample Input 3**

DSA is fun

**Sample Output 3**

fun is DSA

Answer :

// Remove trailing spaces

while (len > 0 && s[len - 1] == ' ') {

len--;

}

s[len] = '\0';

int wordEnd = len;

bool inWord = false;

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

if (s[i] == ' ') {

if (inWord) {

int wordStart = i + 1;

if (i < len - 1) {

printf(" ");

}

for (int j = wordStart; j < wordEnd; j++) {

printf("%c", s[j]);

}

inWord = false;

}

} else {

if (!inWord) {

wordEnd = i + 1;

inWord = true;

}

}

}

if (inWord) {

int wordStart = 0;

if (len > wordEnd) {

printf(" ");

}

for (int j = wordStart; j < wordEnd; j++) {

printf("%c", s[j]);

}

}

}

int main() {

char input\_str[1000];

//printf("Enter the input string: ");

fgets(input\_str, sizeof(input\_str), stdin);

reverseWords(input\_str);

printf("\n");

return 0;

}

Alternatively:

#include <stdio.h>

#include <string.h>

void reverseWords(char \*s) {

int length = strlen(s);

int i, j = length - 1;

while (j >= 0) {

while (j >= 0 && s[j] == ' ') {

j--; // Skip trailing spaces

}

i = j;

while (i >= 0 && s[i] != ' ') {

i--; // Move to the start of the word

}

// Print the word

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

printf("%c", s[k]);

}

if (i >= 0) {

printf(" "); // Print space between words

}

j = i - 1;

}

}

int main() {

char input[1000];

// Read input from the user

fgets(input, sizeof(input), stdin);

// Remove newline character from input

input[strlen(input) - 1] = '\0';

// Reverse the order of words and print the result

reverseWords(input);

return 0;

}

#include <stdio.h>

#include <math.h>

// Structure to represent a term of the polynomial

💡

3)

Structures

Evaluate the polynomial for a given input number, and determine whether the input

number is a root of the polynomial..

Complete the Structures: Task is to complete the structure definitions by adding the

necessary members. struct Term should have members to store the coefficient and

exponent of a term. struct Polynomial should have members to store an array of terms

and the number of terms in the polynomial.

Implement inputPolynomial Function: This function takes as input the polynomial

structure, arrays of coefficients and exponents, and the number of terms. Your task is

to fill in this function to populate the struct Polynomial with the provided coefficients

and exponents.

Implement evaluatePolynomial Function: The evaluatePolynomial function is

responsible for evaluating the polynomial for a given input value x..

Check if Input Number is Root: After evaluating the polynomial, you'll need to

determine whether the input number is a root of the polynomial or not.

**Input Format**

Enter the number of terms in the polynomial: [integer] For each term, provide the

coefficient and exponent separated by a space: Coefficient: [float] Exponent: [integer]

Enter the input number for evaluation: [float]

**Constraints**

Follow the instructions given in the problem stmt

**Output Format**

The program will display the result of evaluating the polynomial at the given input

number. Additionally, it will indicate whether the input number is a root of the

polynomial or not.

**Sample Input 0**

2

1

2

-1

1

1

**Sample Output 0**

Answer is: 0.00

1

is a root.

Answer:

struct Term {

float coefficient;

int exponent;

}

;

// Structure to represent a polynomial

struct Polynomial {

struct Term terms[100]; // Assuming a maximum of 100 terms

int num\_terms;

}

;

// Function to populate the polynomial with provided coefficients and exponents

void inputPolynomial(struct Polynomial \*poly, float coefficients[], int exponents[], int num\_terms) {

poly->num\_terms = num\_terms;

for (int i = 0; i < num\_terms; i++) {

poly->terms[i].coefficient = coefficients[i];

poly->terms[i].exponent = exponents[i];

}

}

// Function to evaluate the polynomial for a given input value x

float evaluatePolynomial(struct Polynomial poly, float x) {

float result = 0.0;

for (int i = 0; i < poly.num\_terms; i++) {

result += poly.terms[i].coefficient \* pow(x, poly.terms[i].exponent);

}

return result;

}

int main() {

struct Polynomial poly;

float coefficients[100];

int exponents[100];

int num\_terms;

float x;

// Read input for the polynomial

//printf("Enter the number of terms in the polynomial: \n");

scanf("%d", &num\_terms);

//printf("For each term, provide the coefficient and exponent separated by a space:\n");

for (int i = 0; i < num\_terms; i++) {

scanf("%f %d", &coefficients[i], &exponents[i]);

}

// Populate the polynomial

inputPolynomial(&poly, coefficients, exponents, num\_terms);

//printf("Enter the input number for evaluation: /n");

scanf("%f", &x);

float result = evaluatePolynomial(poly, x);

printf("Answer is: %.2f\n", result);

if (fabs(result) < 0.001) {

int y=(int)x;

printf("%d is a root.\n", y);

} else {

int y=(int)x;

printf("%d is not a root.\n", y);

}

return 0;

}

LAB 1 - (3)Qns - Array Qn - Pointer Qn - Structures 8