Fundamentele programării

Temă pentru acasă. Iteraţii

1. Completaţi definiţiile următoarelor funcţii care prelucrează tablouri unidimensionale (vectori) de numere întregi, corespunzător cerinţelor precizate. Tablourile au n elemente.

i) `int minProeminente(int a[], int n){...}`
returnează valoarea minimă a elementelor proeminente pozitive dacă există astfel de elemente, altfel returnează -1. Un element este *proeminent* dacă este strict mai mare decât toţi vecinii săi;

ii) `int minRedundante(int a[], int n){...}`
returnează valoarea minimă a elementelor redundante pozitive dacă există astfel de elemente, altfel returnează -1. Un element este *redundant* dacă în tablou mai există unul cu aceeaşi valoare;

iii) `int deltaMaxEgale(int a[], int n){...}`
returnează valoarea maximă a distanţei dintre două elemente egale dacă există elemente egale, altfel returnează 0. Prin distanţa dintre două elemente înţelegem aici diferenţa indicilor în valoare absolută;

iv) `int distinctiaMaxima(int a[], int n){...}`
returnează valoarea maximă a distincţiei elementelor. *Distincţia* unui element este egală cu minimul diferenţelor în modul dintre valoarea elementului şi valorile vecinilor;

v) `int lgMaxSecventaInlantuita(int a[], int n){...}`
returnează lungimea celei mai lungi subsecvenţe de elemente consecutive înlănţuite. Două elemente sunt *înlănţuite* dacă nu diferă prin mai mult de o unitate;

vi) `int lgMaxSecventaUnisemn(int a[], int n){...}`
returnează lungimea celei mai lungi subsecvenţe de elemente consecutive nenule şi toate cu acelaşi semn;

2. Definiţi, corespunzător fiecărui caz de mai jos, funcţia

```
void afizeazaPeRanduri(int a[], int n){ ... }
```

care afişează în ordinea citirii tabloul `a` pe mai multe rânduri astfel încât acestea să respecte cerinţele cazului. Se vor forma cât mai puţine rânduri.

i) pe fiecare rând elementele au aceeaşi paritate;

ii) pe fiecare rând elementele nenule au acelaşi semn;

iii) pe fiecare rând avem un şir strict crescător;

iv) pe fiecare rând avem un şir strict monoton;

v) pe fiecare rând avem un şir monoton;

3. Definiţi, corespunzător fiecărui caz de mai jos, funcţia

```
void afizeazaPeSarite(int a[], int n){ ... }
```

care afişează tabloul `a` dintr-o singură parcurgere, sărind eventual peste unele elemente, astfel încât şirul afişat să respecte condiţia cerută. Vor fi sărite cât mai puţine elemente.

i) şirul afişat este strict crescător;

ii) paritatea numerelor afişate este alternantă;

iii) ordinea strictă a numerelor afişate este alternantă (mai precis: nu există trei numere consecutive α, β, γ astfel încât $\alpha \leq \beta \leq \gamma$ sau $\alpha \geq \beta \geq \gamma$);

Exemplu de rezolvare:

```
#include<iostream>
using namespace std;
bool suntAlternanti(int a, int b, int c){
    return a > b && b < c || a < b && b > c;
}
void afiseazaPeSarite3(int a[], int n){
    if (n <= 0){  cout << endl;
        return;
    }
    int alfa = a[0];
    cout << alfa << " ";
    if (n == 1) {  cout << endl;
        return;
    }
    int beta = a[1];
    cout << beta << " ";
    for (int i = 2; i < n; i++){
        int gama = a[i];
        if (suntAlternanti(alfa, beta, gama)){
            cout << gama << " ";
            alfa = beta;
            beta = gama;
        }
    }
    cout << endl;
}
int main(){
    int tab[10] = { 10, 12, 8, 4, 5, 6, 7, 16, 19, 10 };
    afiseazaPeSarite3(tab, 10);
    return 0;
}
/* REZULTAT:
10 12 8 16 10
Press any key to continue . . .*/
```