Laborator 1

Problema 1

Codul sursa al aplicatie de extragere de radical dintr-un numar real

```
#include <iostream>
#include <time.h>
#define EPS 1e-150

using namespace std;

double sqrt_1(double number) {
    // Descriere: Functia sqrt_1 va aplica sirul lui newtow pentru a calcula radicalul dintr-un numar.
    //     Daca numarul este unul subunitar sirul va incepe de la 1, iar daca numarul este
    //     unul supraunitar sirul va incepe de la numarul respectiv.
    // Date Intrare: number de tip double care este numarul din care extragem radical
    double X1, X2, y;
    if (number > 0 && number < 1) {
        X2 = 1;
    } else {
        X2 = number;
    }
    do {
        X1 = X2;
        X2 = 0.5 * (X1 + number / X1);
        if ((y = X1 - X2) < 0) {
            y = -y;
        }
    } while (y >= EPS);
    return X2;
}

double sqrt_2(double number) {
    // Descriere: Functia sqrt_2 calculeaza radicalul dintr-un numar inmultind radicalul inversului numarului cu numarul in sine.
    //     Radicalul va fi calculat tot cu metoda lui Newtown
    // Date Intrare: number de tip double care este numarul din care extragem radical
    double inverse = 1 / number;
    double X1, X2, y;
    X2 = 1;
    do {
        X1 = X2;
        X2 = 0.5 * (X1 + inverse / X1);
        if ((y = X1 - X2) < 0) {
            y = -y;
        }
    } while (y >= EPS);
    return X2 * number;
}
```

```
int main() {
    clock_t time1, time2; // Pentru a monitoriza cat dureaza executia metodelor
    cout << "Program pentru a calcula radicalul numerelor reale. Programul se va opri
din executie la introducerea numarului 0" << "\n";
    while (true) { // Bucla pentru a mentine aplicatia in viata pana cand user-ul va
introduce cifra 0
        double number; // Declaram numarul pe care il vom citi de la tastatura
        cout << "Numarul este: ";
        cin >> number;
        if (number < 0) { // Daca numarul este mai mic de 0 nu avem rezultat
            cout << "Nu se poate extrage radical din numar negativ in multimea reala"
<< "\n";
            continue;
        } else if (number == 0) { // Daca numarul este 0 rezultatul va fi 0
            cout << "Rezultat: 0" << "\n";
            exit(1);
        } else if (number == 1) { // Daca numarul este 1 rezultatul va fi 1
            cout << "Rezultat: 1" << "\n";
            continue;
        }

        // Pentru numere subunitare si supraunitare vom folosi prima metoda cea din
laborator
        time1 = clock();
        cout << "Rezultat prima metoda: " << sqrt_1(number) << "\n";
        time1 = clock() - time1;
        cout << "Prima metoda a durat: " << time1 << " milisecunde \n";

        if (number > 1) {
            // Pentru numerele subunitare vom folosi si cea de-a doua metoda
            time2 = clock();
            cout << "Rezultat a doua metoda: " << sqrt_2(number) << "\n";
            time2 = clock() - time2;
            cout << "A doua metoda a durat: " << time1 << " milisecunde \n";
        }
    }
}
```

Problema 2

> *Analizati care din obiectivele si principiile IP le putem regasi in aceasta implementare. Modificati eventual aplicatia astfel incat sa le puteti identifica.*

In aceasta implementare se pot regasi mai multe obiective ale ingineriei programarii:

- Adaptabilitatea programelor: Acest program este unu care este usor de trasferat de pe un calculator pe altul, si de asemenea este usor de trecut de pe un sistem de operare pe altul. De asemenea programul fiind deja impartit in functii/metode este usor de adaugat metode noi si/sau de a le modifica deja pe cele existente.

- Eficienta programelor: Programul este unul destul de eficient atat din punct de vedere al memoriei cat si din punct de vedere al timpui de executie avand o complezitate de O(n).
- Fiabilitatea programelor: In momentul implementarii codului am luat in vedere mai multe cazuri care ar fi putut duce la creearea de erori (impartire la 0, extragere radical din numere negativa, samd). In momentul de fata o eroare cunoscuta este aceea ca in momentul in care in mediul principal se introduce un string in loc de un int programul va oprii executia dar nu este un caz tratat specific. Aceasta ar putea fi o imbunatatire a aplicatiei.
- Perceptibilitatea: Programul este unul scurt, usor de inteles. De asemenea programul este comentat pentru o mai buna intelegere a user-ului.

In aceasta implementare se pot regasi mai multe principii ale ingineriei programarii:

- Principiul modularizarii: Functiile care determina radicalul suns scoase din executia principala a programului pentru o mai buna modularizare (modularizare functionala)
- Principiul uniformitatii: Au fost folosite doar denumiri sugestive pentru o claritate mai buna a programului.
- Principiul completitudinii.
- Principiul abstractizarii: s-au identificat problemele generale si au fost omise detalile neesentiale

Problema 3

> *Considerati o alta aplicatie (licenta, etc.) in care sa identificati obiectivele si principiile IP, daca le-ati considerat. Analizati posibilitatea de a le introduce.*

Ca o alta aplicatie pentru a identifica obiectivele si principiile IP-ului am ales lucrarea mea de licenta (link repo). Ca si obiective ale Ip-ului au fost luate in calcul mare parte dintre obiective:

- Adaptabilitatea programelor: Aplicatia a fost realizata sa aiba un design la fel de bun pe orice dimensiune de ecran te-ai afla (de la telefon, tableta pana la monitoare). Este usor de mutat de pe un dispozitiv pe altul (avand un environment creeat care permite rularea aplicatiei oriunde exista un container de docker). Aplicatie este una care nu foloseste multe resurse pe partea de client, iar computatiile de pe partea de server fiind facute intr-un mod cat mai eficient (modele pentru ai gata antrenate, computatii cat mai eficiente samd)
- Eficienta programelor: Pe partea de client aplicatie este una care nu consuma foarte multe resurse
- Fiabilitatea programelor
- Perceptibilitatea: Pentru aceasta aplicatie am creeat mai multe diagrame UML pentru a fi mai usor de inteles. Am creat diagrame de clasa pentru partea de server, diagrame ale bazei de data si de asemenea o diagrama pentru componentele de frontend

Ca si principi se pot regasi mai multe:

- Principiul modularizarii: A fost folosita modularitate obiectuala atat pe partea de frontend cat si pe cea de backend.
- Principiul abstractizarii: s-au identificat problemele generale si s-au omis detalile neesentiale

- Principiul localizarii: s-a folosit mai multe pe partea de front-end avand componentele in relatie mai aproape unele de altele
- Principiul incapsularii informatiei: Nu a fost folosit neputant face asta pe un limbaj de programare compila cum este (Python si JavaScript)
- Principiul uniformitatii: Notatiile au fost unele cat se poate de clara, numele de functii avand o consistenta in toata aplicatie etc.
- Principiul completitudinii
- Principiul confirmabilitatii