Proiect POO ‘ROCK the SHOP’

Sistem pentru administrarea unui magazine

STRESNA GEORGE 323AA

Proiectul simuleaza administrarea automata a unui magazin ce comercializeaza produse din zona muzicala.

Pentru implementare, au fost necesare 3 tipuri de clase:

- Angajatii (iar prin polimorfism subclasele Manager, Operator, Asistent)
- Produse (iar prin polimorfism subclasele Vestimentar, Disc, Disc Vintage)
- Comenzi

```
10   class Angajat
```

```
3   class Asistent: public Angajat
```

```
3   class Manager: public Angajat
```

```
3   class Operator: public Angajat
```

```
10   class Produse
```

```
3   class Vestimentar: public Produse
```

```
3   class Disc: public Produse
```

```
3   class Disc_vintage: public Disc
```

```
class Comanda{
private:
    /*vector<Vestimentar> vV;
    vector<Disc*> dV;*/
    int durataSolutionare = 0, id;
    string dataPrimire = "";
    vector<int> vV, vD;

public:
    Comanda() = default;

    Comanda(vector<int>, vector<int>, string, list<Produse*>&, int);

    void afisare();
    string getDataPrimire();
    double getPret(list<Produse*>&);
    int getDurataSolutionare();
    vector<int> getVV();
    vector<int> getDV();

    //Comanda(const Comanda&);
    //Comanda& operator=(const Comanda&);
    ~Comanda();
};
```

Pentru a incarca datele din fisiere externe in program au fost folosite **liste C11** pentru *Angajati* si *Produse*, si un **queue** pentru *Comenzi*.

Lista de angajati este la baza **list<Angajat*>** iar cea de Produse **list<Produse*>**.

Cand sunt incarcate obiectele, se verifica erorile din documentatie cu blocuri **try{} catch{}** .
Aceast precautie este luata si cand adaugam sau stergem elemente din lista creeata.

Salariul angajatilor nu este stocat intr-o variabila, este mereu calculat la apelarea functiei **getSalariu()**, insa pentru reprezentarea corecta a acestor salarii a fost implementata in plus fata de cerinta in clasa Angajati variabila **comision**, pentru a stoca valoarea comenzilor procesate de Operatori.

Pentru restul angajatilor variabila comision ramane 0 si nu afecteaza salariul final.

Pentru implementrea produselor exista 2 tipuri de functii care returneaza valoarea comenzii: o functie **getPretInit(),** care apare doar in clasa de baza Produse, si care returneaza doar Pretul initial al produselor, si o functie **getPret()** care este overridden in fiecare clasa in parte dupa specificatii.

Cand se adauga un angajat nou, **data de angajare** este setata automat ca **data curenta**. Implementarile cu date au fost de asemenea folosite in procesarea **salariilor angajatilor**(de ziua lor primesc un bonus), si pentru a **ordona comenzile**(cele mai vechi comenzi sunt procesate primele)

Ca elemente de C++ modern au fost folosite exclusiv **string**, **liste**, **vectori** si **queue**, **iteratori**, functii **lambda** in interiorul functiei **sort**.

Pentru lizibilitate au fost create fisiere ***.cpp*** si ***.h*** pentru fiecare implementare de clasa, si cate un fisier ***.cpp*** si ***.h*** pentru functiile specifice pentru angajati, produse si comenzi (procesarea **rapoartelor** se intampla in interiorul fisierului pentru functiile aferente comenzilor).

Tot ce se intampla in program este afisat in **consola**, exista un meniu pentru fiecare actiune posibila pentru fiecare tip de clasa, si exista posibilitatea de relure a altor comenzi in aceeasi sesiune de terminal.

Pentru claritate au fost folosite functiile **clear()** si **sleep()**, pentru a simula un **timer** al Angajatilor care proceseaza in **timp real** comenzi. Durata de solutionare a comenzilor este in secunde pentru a usura testarea programului.

Fiecare angajat, produs si comanda are un ID unic, care este identificat usor prin intermediul **iteratorilor** ce parcurg listele.

Procesarea comenzilor se intampla astfel:

Toti operatorii sunt scosi din lista si introdus intr-un nou vector.

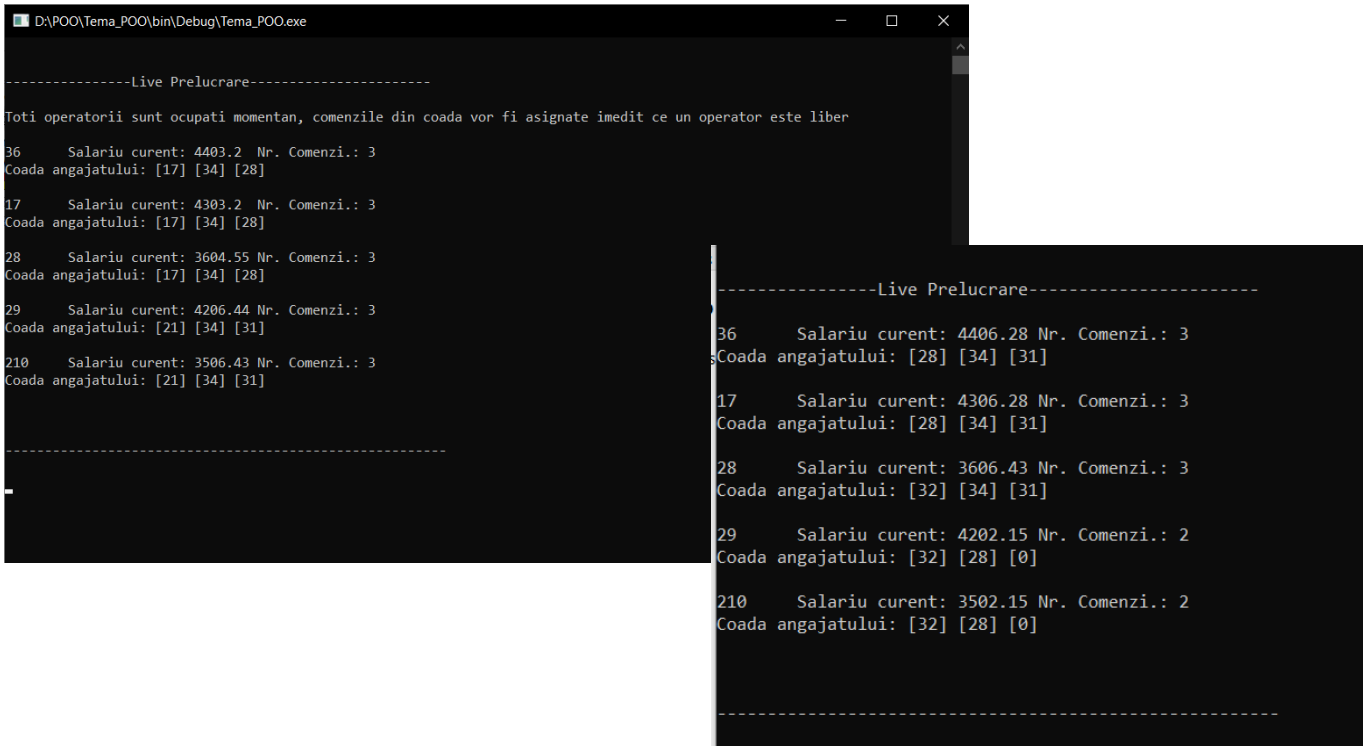
Cat timp inca avem comenzi neasignate in coada, se scaneaza vectorul pentru a vedea care angajat are cele mai putine comenzi. Daca toti angajatii au deja 3 comenzi in coada, atunci se asteapta o secunda, dupa care se reia procesul. Daca exista angajati liberi, angajatului cu cele mai putine comenzi in coada i se asigneaza aceasta comanda, i se atribuie comision, se retine faptul ca proceseaza o noua comanda, si se stocheaza timpul de solutionare al acestei comenzi.

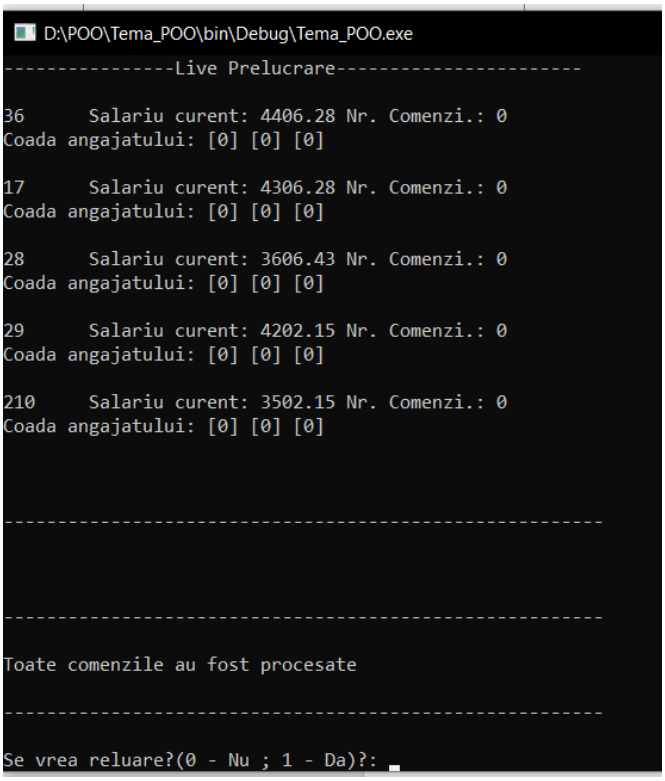
Aceasta comanda va fi ultima din coada de comenzi, asa ca va fi procesata dupa ce toate celelalte comenzi asignate au fost solutionate. Acest loop se intampla cat timp inca exista comenzi in queue.

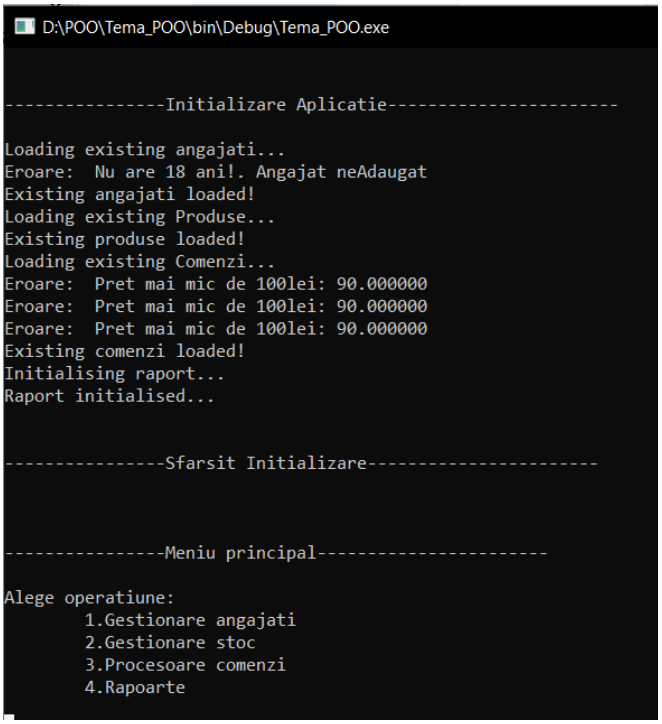
Pe langa algoritmul de asignare a comenzilor se afla si cel care se ocupa de solutionare. La fiecare secunda trecuta, cea mai veche comanda din queue ul angajatului se decrementeaza la timpul de solutionare. Cand aceasta variabila ajunge la 0, adica comanda a fost procesata complet, angajatul poate primi o noua comanda in queue, si isi incepe urmatorea comanda din queue, daca o are.

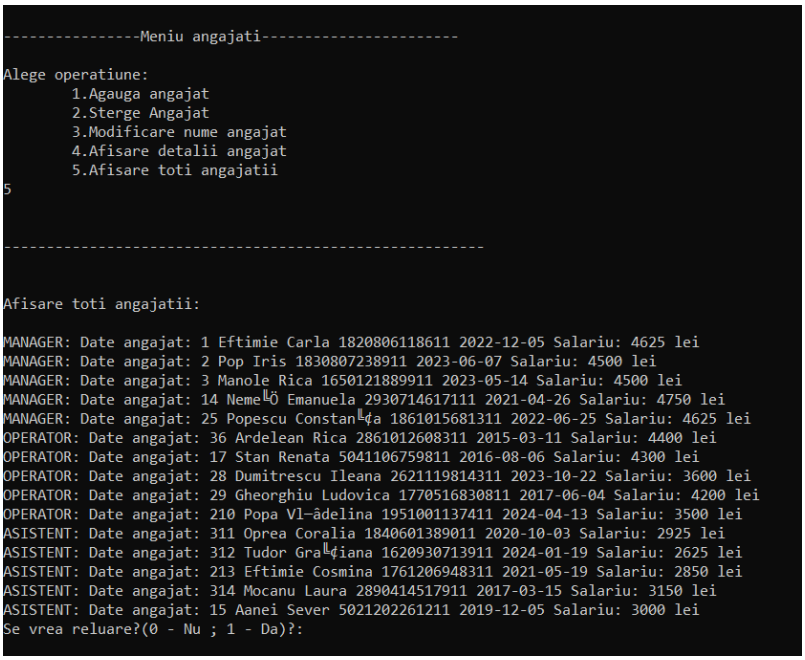
Astfel, incarcand Operatorii treptat cu cate 3 comenzi si asteptand ca (virgula) coada sa se elibereze, timpul lor de incarcare este relativ acelasi pentru un numar mai mare de comenzi.

Implementarile pentru care a trebuit sa ma documentez separat au fost cele legate de **procesarea timpului**, manipulare a **output-ului consolei** si **stringstream**, pe care am folosit-o pentru usurarea citirii din documente. Totodata am lucrat pentru prima data cu fisiere de tip **.csv** si inteleg utilitatea acestora.









Atasat temei sunt toate fisierele folosite, excluzand cele pentru interfata CodeBlocks. Sunt prezente comentarii sugestive si implementari alternative comentate. Fisierele de imput sunt de asemenea incarcate.