Tema 2: eStore Platform

Introducere

Obiective

Obiectivul temei îl reprezintă exercițiul cu mai multe concepte de bază ale Programării Orientate pe Obiecte, presupunând că totul se întâmplă în contextul unui proiect pe care trebuie să îl implementați la locul de muncă si o parte din implementare a fost deja facuta de alti colegi.

Tema 2 utilizează următoarele concepte:

- Derivare și agregare
- Clase și metode virtuale
- Liste neomogene
- STL

Informatii

Dată publicare: **13.12.2020**
Deadline: **24.01.2020**
Punctaj: **15p + BONUS (3p)**

Responsabili: BARAC Ilie-Constantin, CRAIOVEANU Sergiu-Ionuț, IONIȚĂ Alexandru, PARASCHIVA Mihai-Florian

Checker

Link Github: click aici [https://github.com/poo-is/tema2_checker]

Arhiva: click aici

Git Clone: La rularea comenzii de

```
git clone <repo_link>
```

sugeram folosirea WSL. Git Bash introduce probleme in formatarea fisierelor intre Windows/Linux (CRLF vs LF). Cei care au Linux/VM cu Linux nu vor intampina probleme. De asemenea, puteti downlada arhiva fara probleme.

Schelet cod

Link Github: https://github.com/poo-is/tema2_schelet [https://github.com/poo-is/tema2_schelet]

Arhiva: tema2_schelet-master.zip

Modificări enunț

- **24-12-20**: Modificare enunt Cerinta 3, Query 3c: Se vor cauta doar produse resigilate (in loc de produse returnate si resigilate). Se mentioneaza ca sortarea va fi crescatoare, in functie de pret. Am adaugat si informatii suplimentare legate de **dynamic cast**.

- **04-01-21**: Am adaugat o sectiune de "Debugging", care explica cum puteti rula executabilul fara ajutorul checker-ului.

- **05-01-21**: Adăugat în descrierea temei cele două scheme ce prezintă structura unui ID pentru obiectele de tip produs si user. Adăugat informații suplimentare în cadrul cerinței LRU Cache (nu fixați dimensiunea la 4, precum cea din exemplu. Aceasta poate fi oricât).

- **07-01-21**: Adăugat clarificare suplimentară Query 3f: este nevoie de sortare doar dacă nu iterați lista cu utilizatori în ordinea **begin()** → **end()**.

- **08-01-21**: Adăugat clarificare suplimentară Query 3e: este nevoie de sortare dupa ID-ul utilizatorilor inainte de a returna lista.

- **11-01-21**: Adăugat clarificare suplimentară Query 3f: orice produs poate avea cupon (nu doar cele deja reduse). Map-ul unui utilizator premium poate conține cupoane pentru produse care nu mai sunt disponibile. Căutați doar produsele ce se găsesc și în magazin (deci în lista de produse de pe server).

- **20-01-21**: **BUG Query 3d**. Există două perechi de produse alimentare cu același preț si cu acelasi nume (Offtopic: nu stiu care erau sansele. Am generat preturile folosind rand() din MATLAB). Acestea au ID-urile **1238-12112** si **11110-11130**. In cazul in care output-ul pare similar cu fisierul de referinta pentru testul 8, verificati ordinea acestor perechi de produse. O modificare a capatului listei prin care adaugati elemente ar trebui sa rezolve problema (**push_back** in loc de **push_front**). Multumiri lui **Dennis-Andrei Bicu** si **Bogdan-Andrei Enache** pentru descoperirea bug-ului.

Modificări checker

- **23-12-20**: Varianta de arhiva continea executabilul si output-ul ce putea genera nota maxima. Era inconsistenta intre varianta git si cea din arhiva, acum ambele sunt in varianta fara executabil/output. Au aparut nedumeriri in legatura cu 'ce mai trebuie facut'. Corectarea se va face cu rerularea codului sursa si regenerarea tuturor output-urilor. Acel executabil era mai degraba ca sa vedeti ca tema functioneaza corect.

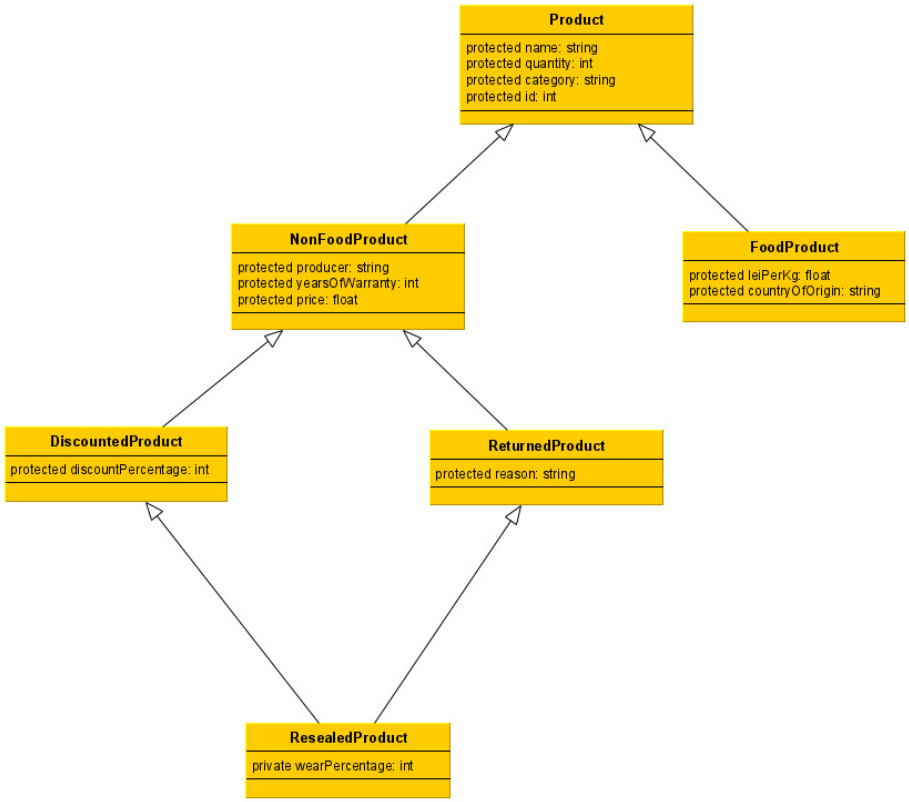
- **04-01-21**: Cerinta 3f (Query f) avea in fisierul de .ref obiecte cu UserID duplicate. Acestea au fost eliminate. Dupa sine, asta a atras regenerarea fisierului de referinta, care difera de versiunea anterioara. Concret, doar fisierul ref10.out din folderul .ref a fost modificat. Kudos Petronel-Valeriu MARIN.

Descrierea temei

Prin această temă, dorim să implementam un sistem pentru o platforma online dedicata vanzarii de produse diverse. Entitatile principale sunt **Produsele**, **Utilizatorii** si **Serverul** care stocheaza detalii despre produse si utilizatori si unde se pot efectua diverse operatii asupra entitatilor, precum căutări de produse, de utilizatori sau modificări de diferite cantități (adăugare sau scoatere de produse din coș).

Mai jos detaliem clasele, cu scopul de a oferi o intelegere mai buna asupra proiectului.

Ierarhia Product



Sageata alba inspre o clasa: 'derivare din'

Aceasta este cea mai complicata ierarhie de clase a intregului proiect - din motive evidente - cand intrati pe un site de retail, tipurile de produse pot fi deseori coplesitoare prin pura lor varietate. Numele Claselor (si implicit al tipurilor) sunt in engleza pentru consecventa (ca sa minimizam utilizarea romglezei in acest proiect).

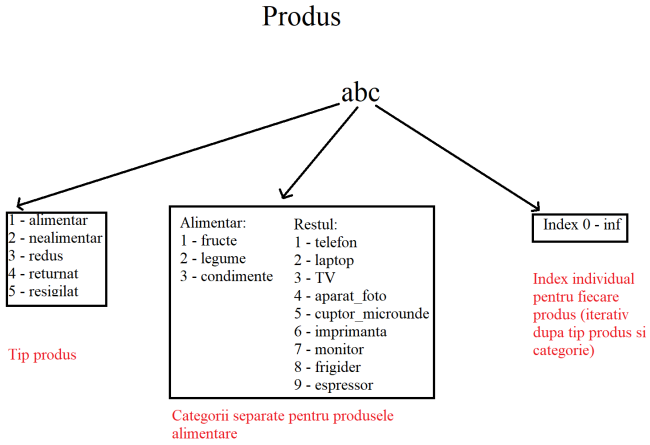
Asadar, clasa **Product** (Produs) este abstracta din dorinta de a ramifica tipurile de produse de pe site. Niciodata nu o sa avem un produs (simplu), intotdeauna o sa apartina de o 'categorie' anume. Ea reprezinta o schita pentru restul tipurilor de produse ce se vor deriva din aceasta.

Mai departe, distingem 2 tipuri de produse: **FoodProduct** (Produs Alimentar) si **NonFoodProduct** (Produs Nealimentar) - ca in viata reala. Fiecare din aceste clase aduce dupa sine campuri/metode noi.

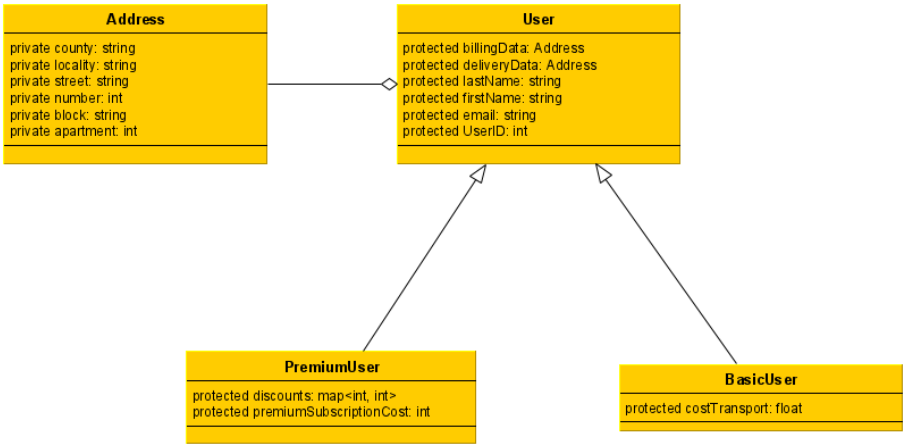
Pentru ca viata reala e deseori mai complicata decat am spera, in sectiunea produselor nealimentare avem de rezolvat o problema de tip 'Death Diamond', conform diagramei de mai sus. **NonFood Product** (Produsele Nealimentare) pot fi de 2 tipuri: **Returned Products** (Produse Returnate, care probabil aveau defecte) si **Discounted Products** (Produse Reduse, care au un pret scazut fata de cel normal).

Din cele 2 ia nastere **Resealed Product** (Produs Resigilat), care indeplineste atat proprietatea de a fi un produs returnat, cat si unul redus.

ID-ul unui produs este unic și respectă următoarea schemă:



Ierarhia User



Romb alb inspre o clasa: 'agregare'

Urmatoarea ierarhie de clase are ca scop implementarea urmatoarei paradigme: "Dai un ban, dar stai in fata".

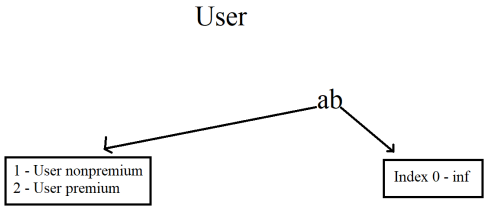
Clasa **User** este una abstracta, care reprezinta schita celor 2 tipuri de Useri: **Basic User** si **Premium User**. Astfel, reusim sa distingem tipurile de Useri prin diferite campuri/metode, oferind posibilitatea particularizarilor (planuri de plata diferite, in functie de User).

Trebuie precizat că **Userii Premium** au un **map** de reduceri de tipul <int ProductID, int discount>, în care sunt stocate reduceri pentru diferite produse. Atenție: Map-ul poate conține și ID-uri ale produselor ce nu se mai află în acest moment în magazin (nu se află în lista cu produse de pe server)

Ca in ierarhia de clase Product, clasa **User** este abstracta, neputand sa avem un User simplu, ci doar **Basic** sau **Premium**.

Are loc, de asemenea, procesul de agregare printr-o clasa **Address** al carui scop este de a introduce campuri complexe de adresa Userilor.

ID-ul unui utilizator este unic și are următoarea schemă:



Clasa ShoppingCart

Nevoia unei clase de **Shopping Cart** (Cos Produse) este evidenta pentru toata lumea, mai ales pentru cei care aleg sa nu ia un cos la intrarea in magazin si tin toate produsele in mana pana la casa.

Cel mai important aspect al acestei clase este reprezentat de campul "shoppingCart", un map ce contine 2 numere intregi per intrare.

```
map <int, int> shoppingCart
```

Cheia (primul numar intreg) reprezinta **Product ID**, adica ID-ul unui produs din 'Baza de Date' a produselor. **Valoarea** (al doilea numar intreg) reprezinta **Quantity**, adica cantitatea unui produs din cos. Practic, per intrare in cosul nostru de produse, vom avea o pereche (ProdusID, Quantity).

```
e.g. (225200, 420) -> Produsul cu ID-ul 225200 are cantitatea de 420 in cosul nostru.
```

Cosul joaca un rol esential mai departe in integrarea lui pe Server, in concordanta cu UserID.

Clasa Server

Clasa Server este piesa ce leaga tot. Aici stocam toate datele legate de Useri si Produse (in liste neomogene). Astfel, ne-ar placea sa avem 'acces' la aceste date de 'oriunde'.

Analogia de mai sus se traduce prin implementarea unui **Design Pattern** numit **Singleton**. Concret, **Singleton** ne permite instantierea unui singur obiect de un anume tip (un singur obiect de tip Server, numit **server**), care poate fi accesat din multiple clase.

Pentru a construi o mai buna intuitie asupra Design Pattern-ului **Singleton**, click aici [https://www.youtube.com/watch?v=hUE_j6q0LTQ]. Pentru un exemplu de cod, click aici [https://refactoring.guru/design-patterns/singleton/cpp/example].

Avem o instanta a **clasei Server**, pe care o numim **server**. Pentru a obtine lista de Produse prin instanta **server**, putem scrie ceva de genul:

```
server->getProductsList()
```

Obiectul nostru reprezinta un pointer catre singura instanta a serverului. Astfel, putem avea pointere catre **instanta server** a **clasei Server** din mai multe fisiere.

Analogie: Imaginati-va ca **instanta server** este Bucuresti. Daca va aflati in Constanta, exista directii catre Bucuresti. La fel daca va aflati in orice alt oras al Romaniei. Puteti sa va imaginati ca desi pointerii (directiile) toate indica o cale diferita catre **instanta** (in cazul nostru Bucuresti), destinatia finala este aceeasi.

Concret, exista un singur Bucuresti, la care se poate ajunge din orice oras al Romaniei.

Scop: Scopul acestui pattern este de a nu permite instanțierea a mai mult de **1** obiect dintr-o clasă.

Un camp important din clasa Server este reprezentat de map-ul:

```
  map<int, ShoppingCart*> __UserID__ProductsCart__
```

Cheia este un UserID (id-ul unui User/Utilizator). **Valoarea** este un pointer catre un obiect de tip **ShoppingCart** (Cos Produse). Practic, acest map reprezinta legatura dintre un User si cosul sau de produse, o functionalitate prevalenta in toate site-ul de retail.

Clasa QuerySolver

Aceasta clasa este pusa la dispozitia voastra pentru a implementa cerintele ce tin de **Queries/Interogari**. Fiecare subcerinta are o metoda aferenta.

Ea nu se afla in propriul folder fiindca nu reprezinta un 'tip' utilizat in proiectul nostru. Este doar o clasa ajutatoare, care speram ca va facilita abordarea cerintelor de **Query** (Cerinta 3).

Clasa LRU Cache

In general, cand vorbim despre resurse computationale si servicii oferite, mereu apare cuvantul “optimizare”. Motivul e simplu: Daca operatiile se efectueaza suficient de repede, avem clienti multumiti. De asemenea, reducand spatiul ocupat pe servere, ajungem sa platim mai putin pentru serviciile Cloud. Astfel, atunci cand rulam milioane de operatii zilnic, optimizarea unei operatii ce are loc des (de exemplu accesul la o resursa populara), poate avea un impact masiv asupra intregului serviciu oferit.

Intuitiv, produsele populare vor fi destul de des accesate, deci ne-ar placea sa oferim acces “privilegiat” acestora. Acesta paradigma se traduce intr-o modificare hardware.

Asadar, pentru implementarea paradigmei de mai sus, vom simula comportamentul unui 'Least Recently Used Cache', care va retine id-ul produselor ce reprezinta ultimele accesari ale utilizatorilor. Pentru o scurta explicatie asupra LRU Cache, click aici [https://www.youtube.com/watch?v=R5ON3iwx78M].

Ierarhia de fisiere a proiectului

La deschiderea folderului principal al Proiectului (temei), veti fi intampinati de 2 entitati:

- Folderul **src**
- Makefile

Makefile

Pentru un recap in termeni de Linux, click aici.

Fisierul Makefile are ca scop compilarea intregului proiect. El este auto-generat si oferit inspre utilizare, deci puteti respira usurati.

Exista 2 reguli de baza:

Prima regula:

```
make
```

Aceasta regula va compila si link-edita intregul proiect. Implicit, asta inseamna generarea unui executabil si a unor fisiere obiect.

Se va genera un nou folder numit **bin**. Executabilul se va gasi la calea

```
bin/realease/eStore
```

sau chiar in proiect.

Se vor genera de asemenea si multiple ierarhii de fisiere obiect. Acestea se gasesc in folderul **build**

Asadar, dupa o compilare cu succes, noul continut al folderului radacina va fi urmatorul:

- bin
- build
- src
- Makefile
- eStore (executabilul)

A doua regula:

```
make clean
```

Comanda de mai sus sterge fisierele **bin**, **build** si **eStore**.

Compilarea poate dura oriunde intre 1-2 minute. Atunci cand ati compilat (cu succes) si efectuati cateva modificari asupra unor fisiere, nu se va recompila tot, ci doar fisierele modificate. Astfel, pentru a reduce timpul pierdut, rulati comanda de mai sus doar cand este necesar.

Intelegerea folderului sursa

Folderul **src** (sursa) contine 2 foldere:

- **Solution**
- utils

Voi veti avea de lucrat exclusiv cu ierarhia de fisiere din cadrul folderului **Solution**. Tot ce se afla in folderul **utils** ar trebui sa ramana in mare parte nemodificat (cu exceptia **utility.h** si **utility.cpp**). Prin modificarea fisierelor mentionate mai jos in **utils**, **json.hpp** sau **main.cpp**, va supuneti riscului de penalizare.

```
src/
├ Solution/
│ ├ LRUCache/
│ │ ├ LRUCache.cpp
│ │ └ LRUCache.h
│ ├ Product/
│ │ ├ DiscountedProduct.cpp
│ │ ├ DiscountedProduct.h
│ │ ├ FoodProduct.cpp
│ │ ├ FoodProduct.h
│ │ ├ NonFoodProduct.cpp
│ │ ├ NonFoodProduct.h
```

```
│   ├ Product.cpp
│   ├ Product.h
│   ├ ResealedProduct.cpp
│   ├ ResealedProduct.h
│   ├ ReturnedProduct.cpp
│   └ ReturnedProduct.h
├ Server/
│   ├ Server.cpp
│   └ Server.h
├ ShoppingCart/
│   ├ ShoppingCart.cpp
│   └ ShoppingCart.h
├ User/
│   ├ Address.cpp
│   ├ Address.h
│   ├ BasicUser.cpp
│   ├ BasicUser.h
│   ├ PremiumUser.cpp
│   ├ PremiumUser.h
│   ├ User.cpp
│   └ User.h
├ QuerySolver.cpp
└ QuerySolver.h
├ utils/
│   ├ objectFactory/ ~~~~~~ NU MODIFICATI ~~~~~~~
│   │   ├ objectFactory.cpp ~~~~~~ NU MODIFICATI ~~~~~~~
│   │   └ objectFactory.h ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ FinalQuestionsHelper.cpp ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ FinalQuestionsHelper.h ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ JSONSerializer.cpp ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ JSONSerializer.h ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ TestHelper.cpp ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ TestHelper.h ~~~~~~ NU MODIFICATI ~~~~~~~
│   ├ utility.cpp
│   └ utility.h
├ json.hpp ~~~~~~ NU MODIFICATI ~~~~~~~
└ main.cpp ~~~~~~ NU MODIFICATI ~~~~~~~
```

Asadar, luand in calcul mentiunile de mai sus, atentia voastra ar trebui sa fie asupra ierarhiei de fisiere din cadrul folderului **Solution**. Aici, fiecare clasa/ierarhie de clase se va gasi intr-un folder cu nume sugestiv.

Mai mult, majoritatea implementarilor voastre se vor efectua in fisierele **.cpp**. Fisierele **.h** contin deja toate metodele de care aveti nevoie pentru a implementa cerintele temei. Cu toate acestea, puteti aduce modificari fisierelor **.h** care nu vor fi penalizate daca sunt bine justificate (metode create de voi, dar pe care le veți detalia în **README**). Atlfel, va supuneti riscului de penalizare.

Punem la dispozitie clasa **utility**, din cadrul folderului **utils**. Aici, puteti implementa propriile metode menite sa va usureze munca. Aceasta clasa va fi de mare ajutor in ceea ce priveste sortarea listelor (Cerinta 3). Implicit, modificarea acestor 2 fisiere (**utility.cpp** si **utility.h**) se poate face dupa bunul plac, fara depunctari.

Sugestii de abordare

Aceasta sectiune are ca scop formularea unor recomandari care noi credem ca va vor ajuta.

Mentiuni Non Tech

1. Cititi toata cerinta temei inainte de a va apuca de tema.
2. Recititi cerintele de oricate ori e nevoie ca sa va asigurati ca ati inteles ce se cere de la voi.
3. Nu asteptati sa vina cunostintele catre voi. Cautati-le activ - la propriu. Nu uitati ca va aflati la un Google Search distanta de orice.
4. Nu va apucati de tema in ultima zi.
5. Nu va apucati de tema in ultimele 2 zile.

- Nu va apucati de tema cu o zi inainte.

Recititi de vreo 3 ori ultimele 3 afirmatii. Folositi-o ca **MANTRA** pentru acest assignment.

Lasand gluma la o parte, noi consideram aceasta tema a fi una complexa. Atentie, **greu != complex** . Exista o multitudine de lucruri noi de care va veti lovi in acest assignment, lucruri care nu pot fi acaparate intr-o singura zi. Nici noi, cei care am scris tema, deja vazand o modalitate de rezolvare integrala a temei, nu am putea termina tot intr-o singura zi. Speram sa ne credeti pe cuvant.

Mentiuni Tech

Ca punct de plecare, asigurati-va ca ati facut vazut ghidul pentru Setup Environment Teme.

Nu vrem sa va impunem un 'Tech Stack' pe care sa il folositi pentru a implementa tema - cu toate acestea, vom mentiona ce a functionat foarte bine pentru noi.

Ca idee principala, folosim:

- WSL2 (pe Windows 10)
- Visual Studio Code (A se distinge de Visual Studio)

De ce WSL2?

Pe scurt: Avem acces la comenzi Linux din Windows 10, mediu in care ne desfasuram activitatile normale. Implicit, exista integrare cu Git, Make sau g++. Mai multe detalii in ghidul de mai sus.

De ce Visual Studio Code?

Asta e o intrebare mai buna. Din urmatoarele motive:

- Editor text, deci implicit mai lightweight ca un IDE (Recomandare Google Search: Text Editor vs IDE).
- **Foarte important:** Extensii. Visual Studio Code are o multitudine de extensii, precum de Intellisense pentru C/C++ sau pentru Formatarea (a se citi: indentarea automata printr-un singur buton) a codului.
- Integrarea cu Terminalul WSL2 (Practic, Visual Studio Code - daca e in zilele lui bune - va detecta prezenta WSL 2 si va instala extensiile necesare). Asta inseamna ca vom putea rula comenzi Linux in terminal si scrie cod intr-un singur loc - fara a face switch intre ferestre.

Desigur, Visual Studio Code ramane o recomandare excelenta si pentru un Environment Linux (adica VM sau Dual Boot).

Pasii de setup ar fi (in mare) acestia:

- Faceti setup-ul cu WSL 2 mentionat in ghid.

- Luati VS Code (adica click aici) [https://code.visualstudio.com/download].
- Sperati sa detecteze ca aveti WSL2 instalat. Daca il gaseste, instalati tot ce recomanda el. E baiat destept, aveti incredere in el.
- Luati extensii pentru C/C++ (intellisense). In principiu, alea mai populare.
- Pentru indentare: (Ctrl + K, apoi Ctrl + F).
- Terminalul se acceseaza folosind **Ctrl + ~**. Alternativ, click in stanga jos. Dupa ce detecteaza WSL 2 si setati ca terminal implicit, va fi necesar si un setup pe Distributia de Linux.
- Setup-ul de pe distributia de Linux e relativ trivial: Username, parola si instalarea Make, g++ (si orice mai simtiti voi nevoia).
- Cea mai importanta mentiune: Daca va blocati, nu asteptati sa vina raspunsurile la voi. Cautati pe Google. Exista foarte multe resurse care va vor ajuta in solutionarea problemei (daca cautati in engleza).

Dupa ce efectuati tot setup-ul mentionat mai sus, o sa aiba mai mult sens si recomandarea noastra din sectiunea **Non Tech**, de a nu va apuca in ultima zi. Lucrurile nu sunt grele, dar necesita timp si sunt predispuse erorilor.

Cerinte

Cerinta 1 (60/150p)

Informatii generale

Pentru a putea crea și stoca pe server cele două liste neomogene, destinate **Produselor** și **Utilizatorilor**, este necesară crearea de clase pentru aceste tipuri de date. Am inclus în scheletul temei fișierele **.h** și **.cpp** primite de la superiori, în care sunt descrise și comentate semnăturile funcțiilor, pentru fiecare tip de date, pe care voi trebuie să le implementați.

De asemenea, după cum veți vedea mai târziu, fiecare utilizator va avea alocat un coș de produse. Vom intra în mai multe detalii în cadrul Cerinței 5. Până atunci, completați și metodele clasei **ShoppingCart**, pe baza semnăturilor și comentariilor lor.

Astfel, pornind de la diagrama UML prezentată la începutul temei și bazându-vă pe cunoștințele proprii, completați metodele ierarhiilor de clase **Product** și **User**, pentru a putea stoca mai târziu, pe server, datele necesare. În plus, completați metodelei clasei **ShoppingCart**.

Fiecare **ierarhie** de clase, plus clasa **ShoppingCart**, are alocată în checker câte un test. Fiecare test verifică toate metodele. Astfel, pentru a trece acel test, asigurați-vă că **toate** metodele sunt corect implementate.

Implementati urmatoarele metode pentru a obtine punctajul aferent:

Cerinta 1.1: Ierarhia Product (20/150p)

```
    // Constructori
    Product();
    Product(const string &, int, const string &, int);
    Product(const Product &p);

    // Operator =
    const Product &operator=(const Product &);

    // Set
    void setCategory(const string &category);
    void setId(int);
    void setQuantity(int);
    void setName(const string &);

    // Get
    string &getCategory();
    int getQuantity();
    int getId();
    string &getName();
```

```
    // Constructori
    FoodProduct();
    FoodProduct(const string &, int, const string &, float, const string &, int);
    FoodProduct(const FoodProduct &);

    // Operator =
    const FoodProduct &operator=(const FoodProduct &);

    // Set
    void setLeiPerKg(float);
    void setCountryOfOrigin(const string &countryOfOrigin);

    // Get
    float getLeiPerKg();
    string &getCountryOfOrigin();
```

```
  // Constructori
  NonFoodProduct();
  NonFoodProduct(const string &, int, const string &, const string &, float, int, int);
  NonFoodProduct(const NonFoodProduct &pn);

  // Operator =
  const NonFoodProduct &operator=(const NonFoodProduct &);

  // Set
  void setYearsOfWarranty(int);
  void setPret(float);
  void setProducator(const string &);

  // Get
  int getYearsOfWarranty();
  float getPrice();
  string &getProducer();

  // Metode Auxiliare
  string getProductType();

  // Operator ==
  bool operator==(const NonFoodProduct &) const;
```

```
    // Constructori
    DiscountedProduct();
```

```
    DiscountedProduct(const string &, int, const string &, const string &, float, int, int, int);
    DiscountedProduct(const DiscountedProduct &);

    // Operator =
    const DiscountedProduct &operator=(const DiscountedProduct &);

    // Set
    void setDiscountPercentage(int);
    // Get
    float getDiscountPercentage();

    // Metode Auxiliare
    float priceAfterDiscount() const;
```

```
// Constructori
    ReturnedProduct();
    ReturnedProduct(const string &, int, const string &, const string &, float, int, const string &, int);
    ReturnedProduct(const ReturnedProduct &);

    // Operator =
    const ReturnedProduct &operator=(const ReturnedProduct &);

    // Get
    string &getReason();

    // Set
    void setReason(string &);

    // Metode Auxiliare
    string getProductType();
```

```
    // Constructor
    ResealedProduct();
    ResealedProduct(const string&, int, const string&, const string&, float, int, int, const string&, int, int);
    ResealedProduct(const ResealedProduct&);

    //Set
    void setWearPercentage(int);

    //Get
    string getProductType();
    float getWearPercentage();

    // Operator =
    const ResealedProduct& operator = (const ResealedProduct&);
```

Cerinta 1.2: Ierarhia User (20/150p)

```
  // Constructori
  Address();
  Address(const Address &);
  Address(const string &, const string &, const string &, int, const string &, int);

  // Operator =
  const Address &operator=(const Address &);

  // Set
  void setCounty(const string &);
  void setLocality(const string &loc);
  void setStrada(const string &);
  void setNumar(int);
  void setBlock(const string &bl);
  void setApartment(int);

  // Get
  string &getCounty();
  string &getLocality();
  string &getStreet();
  int getNumber();
  string &getBlock();
  int getApartment();

  // Operatori
  bool operator==(const Address &);
  bool operator!=(const Address &);
  friend ostream &operator<<(ostream &, const Address &);
```

```
  // Constructori
  User();
  User(const string &, const string &, const string &, int, const string &, int, const string &, const string &, const string &, int, const string &, int, int, const string &, const
  User(const User &);

  // Operator =
  const User &operator=(const User &);

  // Set
  void setLastName(const string &lastName);
  void setFirstName(const string &firstName);
  void setEmail(const string &);
  void setUserID(int);
  void setBillingData(const Address &billingData);
  void setDeliveryData(const Address &deliveryData);

  // Get
  string &getLastName();
  string &getFirstName();
  string &getEmail();
  int getUserID() const;
  Address &getBillingData();
  Address &getDeliveryData();
```

```
  // Constructori
  BasicUser();
  BasicUser(const string &, const string &, const string &, int, const string &, int, const string &, const string &, const string &, int, const string &, int, int, const string &,
  BasicUser(const BasicUser &);

  // Operator =
  BasicUser &operator=(const BasicUser &);
```

```
    // Set
    void setTransportCost(int);

    // Get
    float getTransportCost();
    string getUserType();
```

```
    // Constructori
    PremiumUser();
    PremiumUser(const string &, const string &, const string &, int, const string &, int, const string &, const string &, const string &, int, const string &, int, int, const string &
    PremiumUser(const PremiumUser &);

    // Operator =
    const PremiumUser &operator=(const PremiumUser &);

    // Set
    void setDiscounts(map<int, int> red);
    void setPremiumSubscriptionCost(int cap);

    // Get
    float getTransportCost();
    map<int, int> &getDiscounts();
    int getPremiumSubscriptionCost();
    string getUserType();
```

Cerinta 1.3: Clasa ShoppingCart (20/150p)

```
    // Constructori
    ShoppingCart();
    ShoppingCart(const string&);

    // Get
    string& getPayMethod();
    int getQuantity(int); // Atenție la cazul în care produsul nu este în Coș. Este returnat -1!
    map<int,int>& getShoppingCart();

    //Set
    void setPayMethod(const string &);

    // Auxiliare
    void addProduct(int, int);
    void raiseQuantity(int, int);
    void lowerQuantity(int, int);
    void deleteProduct(int);
```

Cerinta 2: Server (10/150p)

Informații Generale

Cerința 2 se concentrează asupra clasei **Server**, în care sunt stocate datele magazinului nostru online. Pe lângă metodele ce trebuie completate, considerăm că este important să înțelegeți și metodele gata implementate, pe care vi le oferim tocmai în acest scop.

După cum puteți observa și în cod, instanța respectivă este stocată într-o variabilă de tip **static**, iar inițializarea acesteia se face în funcția **static Server* ServerInit()**, care la **prima** apelare creează și întoarce instanța de tip **Server**, iar la apelările ulterioare întoarce instanța deja creată. Astfel, este asigurată unicitatea obiectului de tip Server, în tot spațiul de lucru al programului nostru.

Detalii Tehnice

Metodele ce trebuie completate în cadrul acestei cerințe sunt:

```
    //Get
    map<int, ShoppingCart*> get__UserID__ProductsCart__();
    list<Product*>& getProductsList();
    list<User*>& getUsersList();

    //Set
    void set__UserID__ProductsCart__();
```

Map-ul UserID__ProductsCart reprezintă legătura dintre un utilizator și coșul său de cumpărături. Pentru această cerință, map-ul va face legătura între ID-ul lui utilizator și un obiect de tipul ShoppingCart **gol** (fără elemente).

Metodele **requestAddProduct** și **requestDeleteProduct** nu trebuie completate acum. Ele fac parte din Cerința 5.

Cerinta 3: Queries (60/150p)

Deoarece in acest moment avem stocate pe server toate datele legate de produse si utilizatori, putem filtra cele doua liste pentru a gasi elemente ce ne intereseaza, in functie de valorile lor. Astfel, am gandit Cerinta 3 ca o colectie de **interogari/queries**, similare cu cele pe care le-am realiza intr-o Baza de Date (in contextul nostru, altă parte a echipei lucrează la partea de interfatare cu baza de date). Am incercat sa prezentam fiecare interogare intr-un contex real.

Pentru a putea realiza aceste interogari, asigurati-va ca ați implementat corect Cerințele 1 (mai puțin clasa **ShoppingCart**) și 2, deoarece veti avea nevoie de cele doua liste, cea cu produse si cea cu utilizatori, populate cu instanțe ale celor două ierarhii

3a: Query 1

Din pacate, intr-o dimineata, George a constatat ca vechiul sau espressor s-a stricat. Pentru ca are nevoie rapid de unul nou, el a intrat pe site-ul magazinului nostru. Totusi, George are o problema. Fiind inainte de salariu, el cauta un produs redus. Astfel, cautati in **lista de produse espressoarele reduse** si returnati intr-o lista aceste produse.

3b: Query 2

Pentru analiza ulterioara, avem nevoie de toti utilizatorii cu un cost de transport redus. Returnati lista ce contine **utilizatorii nonpremium** (cei care platesc transportul) care sunt nevoiti sa plateasca un **cost pentru transport ≤ 11.5** lei.

3c: Query 3

Gasiti toate **produsele resigilate**, cu motivul **"lipsa_accesorii"** . Sortati-le **crescator** dupa **pret** si returnati lista neomogena.

O modalitate prin care puteți accesa câmpurile unei **clase derivate**, având o listă neomogenă, este **dynamic_cast**. Parcurgând lista, încercați să faceți dynamic_cast pentru fiecare element. Atât timp cât rezultatul este **diferit de nullptr**, operația a fost realizată cu succes.
Puteți găsi aici [https://en.cppreference.com/w/cpp/language/dynamic_cast] mai multe informații.

3d: Query 4

Vrem sa vedem ce produse alimenare se afla in magazin, grupandu-le pe cele cu acelasi nume. (ex mere din Spania si din Polonia). Mai mult, daca intalnim doua produse din aceeasi tara, vrem sa le sortam dupa pret. Astfel, sortati **produsele alimentare** dupa **nume, tara si pret** si returnati lista obtinuta.

3e: Query 5

Tot pentru analize de marketing, vrem sa vedem **utilizatorii** care au atat **adresa de livrare**, cat si cea de **facturare la casa** si locuiesc in **judetul cu cei mai multi utilizatori**. **Sortati** lista dupa ID-ul utilizatorilor. Refacem urmatoarele precizari:

- Judetele se scriu cu prima litera mare, iar cele formate din 2 cuvinte sunt separate prin _ (ex **Satu_Mare , Iasi**)
- Numarul de locuitori din fiecare judet nu trebuie salvat. Este folosit doar pentru gasirea **utilizatorilor**
- O adresa se afla la casa daca in campul **bloc** se afla **-** , iar in campul **apartament** se afla **0**

```
{
  "type": "nonPremium",
  "UserId": 1278,
  "firstName": "Ecaterina",
  "lastName": "Stefan",
  "email": "ecaterinastefan@gmail.com",
  "billingData": {
    "county": "Mures",
    "locality": "Iernut",
    "street": "Mihai_Viteazu",
    "number": 122,
    "block": "-", //adresa facturare la casa
    "apartment": 0
  },
  "deliveryData": {
    "county": "Mures",
    "locality": "Iernut",
    "street": "Mihai_Viteazu",
    "number": 122,
    "block": "-", //adresa livrare la casa
    "apartment": 0
  },
  "costTransport": 12.80
}
```

3f: Query 6

Deoarece cupoanele de reduceri sunt generate automat, vrem sa vedem **utilizatorii premium** care au cupoane de reducere pentru **telefoane sau imprimante**. (Re)facem urmatoarele precizari:

- map-ul cu reduceri al unui utilizator este de forma **<ID_produs, Procent_reducere>**

```
{
  "type": "premium",
  "UserId": 2163,
  "firstName": "Magdalena",
  "lastName": "Apostol",
  "email": "magdalenaapostol@gmail.com",
  "billingData": {
    "county": "Calarasi",
    "locality": "Fundulea",
    "street": "Parcului",
    "number": 132,
    "block": "X13",
    "apartment": 10
  },
  "deliveryData": {
    "county": "Calarasi",
    "locality": "Fundulea",
    "street": "Parcului",
    "number": 132,
    "block": "X13",
    "apartment": 10
  },
  "premiumSubscriptionCost": 98.75,
  "discounts": [    //exemplu map reduceri
    [
      1435,
      10
    ],
    [
      3669,
      8
    ],
    [
      2497,
      24
    ],
    [
      5402,
      17
    ],
    [
      2888,
      24
    ]
  ]
}
```

- un produs contine atat ID-ul, cat si categoria. Puteti gasi produsele dupa un camp sau altul
- ID-urile produselor sunt unice si respecta structura prezentata in descrierea temei
- Orice tip de produs poate fi redus
- Map-ul utilizatorilor premium poate conține și produse ce nu se mai află în magazin. Asigurați-vă că verificați doar produsele ce se găsesc și pe server.

- Daca parcurgeti lista cu utilizatori in ordinea in care aceasta este citita din fisier (begin() → end()), nu mai trebuie sa o sortati inainte de a o returna, deoarece utilizatorii au fost generati cu ID-uri in ordine crescatoare

```
  {
    "type": "redus",
    "category": "telefon", //categoria produsului
    "id": 311, //3 - redus; 1 - telefon; 1- numar ordine
    "producer": "Samsung",
    "name": "Note20",
    "price": 5120.93,
    "yearsOfWarranty": 2,
    "discountPercentage": 14,
    "quantity": 3
  }
```

Cum sortam o lista?

```
#pragma once
#include <iostream>
#include <string>
using namespace std;

/*
    Definim o clasa person
*/
class Person
{
private:
    string name;
    int age;

public:
    Person(string, int);

    string getName() const;

    int getAge() const;

    void printPerson() const;
};
```

```
#include "Person.h"

Person::Person(string name, int age)
{
  this->name = name;
  this->age = age;
}

string Person::getName() const
{
  return this->name;
}

int Person::getAge() const
{
  return this->age;
}

void Person::printPerson() const
{
  cout << "\nName: " << this->name << "\nAge: " << this->age;
}
```

```
#pragma once
#include "Person.h"

using namespace std;

class Utility
{
public:
    /*
      Metoda de mai jos va fi folosita de functia sort din STL pentru compararea a doua persoane.
    */
    static bool comparePersons(const Person &, const Person &);
};
```

```
#include "Utility.h"

bool Utility::comparePersons(const Person &pers1, const Person &pers2)
{
  /*
    Returneaza true atunci cand doua string-uri sunt ordonate alfabetic
    iar false in caz contrar.     .
  */
  if (pers1.getName() < pers2.getName()){return true;}
  if (pers1.getName() > pers2.getName()){return false;}

  /*
    Atunci cand doua string-uri sunt identice se trece la al doilea criteriu de sortare.
    In acest caz al doilea criteriu va compara varsta.
  */
  if (pers1.getAge() < pers2.getAge()){return true;}
  if (pers1.getAge() > pers2.getAge()){return false;}

  /*
    OBS: putem adauga oricate criterii de sortare dorim.
    putem, spre exemplu, sa sortam dupa nume alfabetic, iar dupa varsta descrescator.
  */

  return false;
}
```

```
#include <iostream>
#include <list>
#include <string>
```

```
#include "Person.h"
#include "Utility.h"

/*
        In exemplul de mai jos voi demonstra modalitatea prin care se poate
        face o sortare prin mai multe criterii folosind functii STL
*/

using namespace std;

void printPersons(list<Person> &persons) {
        /*
                "it" este un iterator (pointer) care pleaca de la inceputul listei si pointeza pe rand catre fiecare element al listei pana cand ajunge la finalul acesteia.
    "auto" spunem compilatorului sa deduca tipul iteratorului it din tipul variabilei cu care initializam (person.begin())
        */
        for (auto it = persons.begin(); it != persons.end(); it++) {
                it->printPerson();
                cout << endl;
        }
}

int main() {
        /*
                Declaratia listei de persoane.
        */

        list<Person> persons;

        /*
                Adaugam persoanle in lista.
        */
        persons.push_back(Person("Ion", 3));
        persons.push_back(Person("Ion", 21));
        persons.push_back(Person("Alex", 31));
        persons.push_back(Person("Alex", 27));
        persons.push_back(Person("Ana", 12));
        persons.push_back(Person("Ion", 1));
        persons.push_back(Person("Ana", 12));

        /*
                Se vor afiasa personale in ordinea in care au fost introduse in lista
        */
        cout << "LISTA NEORDONATA :\n";
        printPersons(persons);

        /*
        Sortam lista dupa urmatoarele doua criterii:
                1) alfabetic, dupa nume
                2) dupa varsta, crescator
                Daca mai multe persoane au acelasi nume, atunci vor fi sortate intre ele dupa varsta, crescator.
        */

  /*
    Dam ca parametru functia de comparare a doua persoane.
  */
        persons.sort(Utility::comparePersons);

        cout << "\n\nLISTA ORDONATA :\n";
        printPersons(persons);
}
```

Pentru a experimenta cu acest cod, click aici [https://repl.it/@thesergiu/SortareLista#main.cpp].

Cerinta 4: Least Recently Used Cache (20/150p)

Informatii generale

Pentru a implementa cerinta, nu este nevoie sa stiti prea multe despre scopul si functionalitatile cache-ului intr-un sistem hardware, dar aici [https://www.youtube.com/watch?time_continue=8&v=4McNhpkDNpQ&feature=emb_logo] se afla o resursa utila in caz ca sunteti curiosi si vreti sa intelegeti mai bine puterea unui cache intr-un sistem de calcul.

Exemplu de functionare

Presupunem ca primim un sir de numere, care pot reprezenta ID-urile unor produse accesate pe site-ul nostru.

```
1 2 3 2 5 3 4 5 8 9
```

Primim un sir de 10 numere, iar dimensiunea LRU Cache-ului nostru este de 4 elemente. Asta inseamna ca in oricare moment de timp, noi vom stoca maxim 4 elemente (ultimele accesate).

```
1 _ _ _      1
2 1 _ _      2
3 2 1 _      3
2 3 1 _      2
5 2 3 1      5
3 5 2 1      3
4 3 5 2      4
5 4 3 2      5
8 5 4 3      8
9 8 5 4      9
```

Detalii tehnice

Pentru a obtine punctajul aferent Cerintei 4, implementati urmatoarele metode:

```
// Constructor care initializeaza capacitatea (nu doar campul)
LRUCache(int capacity)

// Metoda ce primeste ca parametru un vector de ID-uri
// Returneaza stadiul final al LRU Cache-ului
vector<int> processRequests(vector<int> requestsNo)

// Get
int getCapacity();
int getSize();
vector<int> getLRU();
int getLRUCapacity();
```

```
// Set
void setCapacity(int);
void setLRU(vector<int>);
```

Chiar dacă în exemplul prezentat dimensiunea este 4, implementarea se va face folosind capacitatea obiectului de tip LRU (nu fixați dimensiunea la 4 în cadrul funcțiilor)

(BONUS) Cerinta 5 (30/150p)

Cerința 5 poate fi rezolvată doar dacă **Cerințele 1, 2 și 4** au fost implementate corect, adică ați primit punctaj pe ele.

Informații Generale

Ultimul concept pe care vrem să îl implementăm este de a da posibilitatea utilizatorilor să adauge sau să scoată produse din coșurile de cumpărături. Aceste operații sunt deservite de Server sub formă de **cereri** (requests) și primesc un răspuns de tip Bool, în funcție de realizarea cu succes a cererii sau nu.

O astfel de cerere are ca parametri ID-ul utilizatorului care a inițializat-o, pentru a putea accesa coșul de cumpărături corect, ID-ul produslui ce se dorește a fi adăugat în coș și cantitatea dorită din acel produs.

De asemenea, cerința reutilizează coada cu priorități creată la Cerința 4 (LRU Cache), dar voi **nu trebuie** să apelați efectiv acest update, el făcându-se în background (vedeți funcția **TestCerinta5** din clasa **TestHelper**)

Detalii Tehnice

Metodele ce trebuie implementate fac parte din clasa **Server** și sunt următoarele:

```
bool requestAddProduct(int userID, int productID, int quantity);
bool requestDeleteProduct(int userID, int productID, int quantity);
```

Multe metode ajutătoare au fost implementate în ierarhia de Produse și în ShoppingCart. Le puteți utiliza pentru rezolvarea cerinței. Aceste metode sunt:

```
Product:
    bool checkQuantity(int quantity);    // Verifică dacă cantitatea de produs dată ca parametru este disponibilă sau nu.
    void decreaseQuantity(int quantity); // Crește cantitatea disponibilă a produsului
    void increaseQuantity(int quantity); // Scade cantitatea disponibilă a produsului

ShoppingCart:
    int getQuantity(int id);                 // Întoarce cantitatea produsului cu id-ul dat ca parametru. Atenție la cazul în care produsul nu există în coș => return -1
    void addProduct(int id, int quantity);   // Adaugă produsul în Coș, cu cantitatea dată
    void raiseQuantity(int id, int quantity); // Crește cantitatea pentru produsul din Coș
    void lowerQuantity(int id, int quantity); // Scade cantitatea pentru produsul din Coș
    void deleteProduct(int id);              // Sterge produsul din Coș
```

Vă recomandăm să urmăriți modalitatea de testare a cerinței în funcția **TestCerinta5** din clasa **TestHelper**. De asemenea, vrem să vă oferim următoarele hint-uri legate de această cerință:

- Ambele metode întorc o valoare **bool**. **True** în cazul în care operația a fost realizată cu succes și **False** dacă nu a fost posibilă;
- Pentru ambele metode, dacă parametrul **quantity** este **mai mic sau egal cu 0**, operația este **respinsă**;
- Căutați existența utilizatorului și a produsului în Server. Dacă una dintre entități nu există, operația trebuie **respinsă**;
- Dacă produsul **nu există** în coșul de cumpărături, se încearcă **adăugarea** lui (cantiatea fiind dată ca parametru) în cazul operației **requestAddProduct**. În cazul operației **requestDeleteProduct**, operația trebuie **respinsă**, deoarece nu putem șterge din coș un produs pe care nu îl avem;
- Dacă produsul **există** în coșul de cumpărături, se încearcă modificarea cantității cu cea specificată ca parametru. Adăugarea se face **peste** cantitatea deja prezentă în coș, iar ștergerea se face prin **scăderea** cantității (atenție la cazul în care se dorește ștergerea unei cantități mai mari decât cea din coș: se șterge **toată** cantitatea din coș, inclusiv produsul);
- Aveți grijă să updatați tot timpul cantitatea disponibilă în **lista de produse**.

Exemplu Funcționare

Un exemplu de funcționare este prezentat mai jos. Presupunem că avem **un sigur utilizator** și **un singur produs** pe Server:

```
    {
      "type": "nonPremium",
      "UserID": 13,
      "firstName": "Paul",
      "lastName": "Nastase",
      "email": "paulnastase@yahoo.com",
      "billingData": {
        "county": "Dambovita",
        "locality": "Moreni",
        "street": "Closca",
        "number": 44,
        "block": "VX3",
        "apartment": 10
      },
      "deliveryData": {
        "county": "Dambovita",
        "locality": "Moreni",
        "street": "Closca",
        "number": 44,
        "block": "VX3",
        "apartment": 10
      },
      "costTransport": 10.40
    }

    {
      "type": "alimentar",
      "category": "condimente",
      "id": 1375,
      "name": "cimbru",
      "leiPerKg": 28.59,
      "countryOfOrigin": "China",
      "quantity": 6
    }

    Coșul de produse al utilizatorului este inițializat gol. Aplicăm următoarele operații:
    requestAddProduct(10, 1375, 2)      // Return False, deoarece Utilizatorul cu ID 10 nu există pe Server
    requestAddProduct(13, 2500, 2)      // Return False, deoarece Produsul cu ID 2500 nu există pe Server
    requestAddProduct(13, 1375, -2)     // Return False, deoarece quantity <= 0
```

```
requestAddProduct(13, 1375, 10)     // Return False, deoarece cantitatea cerută e mai mare decât cea disponibilă (6)

requestDeleteProduct(13, 1375, 10)  // Return False, deoarece nu avem produsul cu ID 1375 în Coș
requestDeleteProduct(13, 1375, -10) // Return False, deoarece quantity <= 0

requestAddProduct(13, 1375, 2)      // Return True. Se adaugă în Coș cantitatea de 2. În lista de produse, cantitatea disponibilă scade la 4
requestAddProduct(13, 1375, 2)      // Return True. Se mai adaugă cantitatea de 2, peste cea din coș. În lista de produse, cantitatea disponibilă scade la 2
requestDeleteProduct(13, 1375, 1)   // Return True. Se scade cantitatea de 1 din Coș, devenind 3. Crește cantitatea disponibilă de produs la 3
requestDeleteProduct(13, 1375, 10)  // Return True. Vrem să ștergem 10, dar în Coș avem doar 3. Se sterge cantitatea de 3 și se adaugă la cantitatea disponibilă, devenind 6
```

Checker si Verificare

Pentru a intelege ce trebuie implementat, dar si felul in care lucrurile sunt testate, va trebui sa inspectati 2 entitati:

- Checkerul (folderele data, in, out)
- Clasa TestHelper (locatie: src/utils/TestHelper.h, TestHelper.cpp)

Checker

Checker-ul contine 3 foldere si script-ul de rulare. Cele 3 foldere sunt:

- data
- out
- ref

data contine toate informatiile ce ar putea fi procesate. Nu toate testele au nevoie de continutul 'data', dar el este pastrat acolo pentru a satisface structura checker-ului.

out contine output-ul obtinut dupa rularea functiilor implementate de voi. In esenta, folderul acesta va fi rezultatul generat de proiectul vostru.

ref contine 'raspunsul' la care ar fi trebuit sa ajungeti. Checker-ul verifica daca fisierele din 'out' se potrivesc cu cele din 'ref'. Daca da, testul va trece.

Este important sa intelegeti rolul fiecarui folder pentru a face debug. Extrem de util pentru Query-uri (Cerinta 3).

Clasa TestHelper

Clasa TestHelper contine o functie pentru fiecare test ce va rula. Ea este apelata din main. Puteti inspecta fiecare functie in parte (aferenta fiecarui test), pentru a intelege mai bine cum sunt implementarile voastre verificate.

Foarte util pentru teste care nu fac citiri din fisier (Cerinta 1, 4, 5).

Debugging

Pentru a face debug asupra proiectului, trebuie sa puteti genera un executabil care compileaza. Odata ce l-ati generat, va puteti folosi de urmatoarea regula pentru a testa functionarea executabilului:

```
./eStore date.in <output_file> <numar_test_din_main>
```

- eStore e numele executabilului (duuuh)
- "date.in" e fisierul ce contine toate datele de intrare. Poate fi gasit in folder-ul cu checker-ul. Va trebui sa o copiati in locatia de unde executati comanda sau sa oferiti in schimb calea absoluta sau relativa la care se gaseste.
- <output_file> e un fisier de output ce se va genera daca nu exista. Ii puteti da orice nume.
- <numar_test_din_main> este numarul (indicele) testului din main.cpp care va verifica testul pe care vreti sa faceti debugging. E.g. Cerinta 2 == indicele 4.

Pentru simplitate, rulati regula din folder-ul principal al proiectului (acolo unde se genereaza si executabilul)

Checker-ul nu va genera niciun output aferent executabilului. Mai exact, checker-ul redirectioneaza tot output-ul din executabil (cout-uri) in **dev/null/**. Nu va bazati pe checker pentru debugging.

Reguli finale

Trimiterea temei

Tema se va incarca pe Moodle, ca arhiva.

Format nume arhiva: GrupaSerie_Nume_Prenume_TemaNr.zip

Puteţi încărca mai multe soluţii, se va lua în considerare ultima solutie incarcata, termen limita 24.01.2020

Arhiva trimisă conține (direct în rădăcină):

- Fişierele (.cpp si .hpp) cu codul programului (proiectul). In cazul acestei teme: folder-ul **src** cu toate fisierele, intr-o forma compilabila;
- Makefile-ul (cu regulile make build și make clean). Executabilul generat trebuie să se numească eStore (adica la fel ca atunci cand primiti proiectul) - practic, Makefile-ul ar trebui sa ramana nemodificat;
- Fișierul README în care va fi descrisă soluția problemei. Pentru o indrumare in scrierea README-ului, click aici.

Pentru neincluderea README-ului sau scrierea sa incompleta, depunctam pana la 5p/180!

Pentru coding style haotic, depunctam pana la 5p/180!

Recomandarea noastra este sa nu modificati Makefile-ul. Am implementat noi aceasta bucata ca sa va puteti concentra exclusiv pe codul proiectului.

Restrictii

- Nu se va modifica scheletul propus.
- Scaderea punctajului pentru hard-codare. Hard-codarea ar putea fi definita prin implementarea unei cerinte fara abilitatea de a generaliza unor input-uri variate, in ciuda unui test din checker trecut.

poo-is/tema2.txt · Last modified: 2021/03/19 10:48 by alexandru.ionita99