OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- domaći zadatak broj 2 -

Funkcionalna specifikacija

Na programskom jeziku C++ implementirati biblioteku klasa za čuvanje informacijama o osobama, a zatim kreirati biblioteku klasa za dinamičku reprezentaciju porodičnog stabla (*FamilyTree*). U nastavku teksta opisane su funkcionalnosti svih klasa u sistemu.

Specifikacija funkcionalnosti klase Date.

Datum (*Date*) se sastoji iz dana (*day*), meseca (*month*) i godine (*year*). Godina je nenegativan ceo broj, mesec pripada intervalu [1, 12], a dan pripada nekom od intervala [1, 31], [1, 30], [1, 29], [1, 28] u zavisnosti od meseca i godine (da li je prestupna ili nije).

Stvaranje datuma.

Implementirati konstruktor `Date(int day, int month, int year)`. U slučaju prosleđivanja neispravnih vrednosti podići izuzetak *InvalidDateException* i oporaviti se od greške. Oporavak od greške podrazumeva ispis informacije o grešci a zatim postavljanje datuma na podrazumevanu vrednost (1.1.1900).

Upoređivanje datuma.

Preklopiti operatore <, >, <=, >=, == i != za upoređivanje dva datuma i operator – koji vraća razliku u godinama između dva datuma zaokruženu na manju vrednost.

Ispis datuma.

Implementirati metodu `print` koja ispisuje datum u formatu *dd.mm.yyyy.* Jednocifrene vrednosti za dane i mesece treba dopuniti vodećom nulom.

```
void print(ostream& os, Date& date);
```

Specifikacija funkcionalnosti klasa Person, Male i Female.

Osobu opisuju njeno ime (*firstName*), prezime (*lastName*), datum rođenja (*dateOfBirth*) i jedinstveni lični broj (*personalId*). Osobe se na osnovu pola dele na muške (*Male*) i ženske (*Female*) osobe.

Stvaranje osobe.

Implementirati konstruktor `Person(string first, string last, Date date, long id);` kao i odgovarajuće konstruktore za muške i ženske osobe. Ukoliko je prosleđeni lični broj jedinstven u okviru arhive ličnih brojeva (*PersonalIdArchive*), dodati ga u arhivu i dodeliti osobi. Ukoliko prosleđeni lični broj nije jedinstven u okviru arhive ličnih brojeva, dodeliti sledeći slobodan jedinstven broj iz arhive ličnih brojeva, a zatim ga dodati u arhivu.

Odeđivanje pola.

Implementirati metodu `gender()` koja treba da vrati jednoslovnu oznaku pola. Metoda treba da podrži polimorfizam i vrati oznaku 'M' ukoliko je osoba muško, odnosno oznaku 'Z' ukoliko je osoba žensko. `char Person::gender();`

Dohvatanje informacija o osobi.

`string Person::getLastName(); string Person::getFirstName();` i
`long Person::getPersonalId(); Date Person::getDateOfBirth();`

Upoređivanje osoba.

Preklopiti operatore <, >, <=, >=, == i != za upoređivanje starosti dve osobe.

Ispis osobe.

Implementirati metodu `print` koja ispisuje osobu tako što se prvo ispisuje jedinstveni lični broj na širini od devet cifara. Potom se redom ispisuju prezime, ime, jednoslovna oznaka pola i datum rođenja osobe. Svaki element ispisa je odvojen jednim znakom za tabulaciju.

```
void print(ostream& os, Person& person);
```

Specifikacija funkcionalnosti uslužne klase PersonalIdArchive.

Arhiva ličnih brojeva pamti lične brojeve svih osoba u sistemu (*personalIdArchive*), pri čemu su svi lični brojevi u arhivi jedinstveni. U svakom trenutku se čuva informacija o sledećem slobodnom ličnom broju (*nextPersonalId*) koji je na početku postavljen na 1. Klasa je uslužna i ne mogu se kreirati objekti klase.

Dodavanje broja.

Implementirati metodu `void PersonalIdArchive::addRecord(long id);`. Metoda treba da doda prosleđeni lični broj u arhivu. Ukoliko broj već postoji u arhivi, podići izuzetak *MultiplePersonalIdException*.

Dohvatanje arhive/sledećeg slobodnog ličnog broja.

Implementirati metodu `list<long> PersonalIdArchive::getRecords();` koja vraća sadržaj arhive.

Implementirati metodu `long PersonalIdArchive::nextId();` koja vraća sledeći slobodan lični broj.

Da li arhiva sadrži lični broj?

Implementirati metodu `bool PersonalIdArchive::containsId(long id);` koja proverava da li prosleđeni lični broj postoji u arhivi. Vraća true ukoliko broj već postoji u arhivi.

Specifikacija funkcionalnosti uslužne klase MarriagesArchive.

Arhiva brakova pamti sve sklopljene brakove u sistemu. Klasa je uslužna i ne mogu se kreirati objekti klase.

Sklapanje braka.

Implementirati metodu `void MarriagesArchive::marry(Person* p1, Person *p2);` koja sklapa brak između proseđene dve osobe. Prilikom sklapanja braka, brak se dodaje u arhivu brakova. Greška je ukoliko osobe imaju isto prezime (*SameLastNameMarriageException*), ukoliko je jedna od osoba već u braku (*AlreadyMarriedException*) i ukoliko su osobe istog pola (*SameGenderMarriageException*).

Raskid braka.

Implementirati metodu `void MarriagesArchive::divorce(Person* p);` koja raskida brak između prosleđene osobe i njenog partnera. Prilikom raskida, brak se briše iz arhive brakova. Greška je ukoliko osoba nije u braku (*NotMarriedException*).

Dohvatanje bračnog partnera.

Implementirati metode `Person* MarriagesArchive::getSpouse(Person* p);` i `Person* MarriagesArchive::getSpouse(long id);` koje dohvataju bračnog partnera prosleđene osobe, odnosno bračnog partnera osobe sa prosleđenim ličnim brojem. Metoda vraća `nullptr` ukoliko ne pronađe osobu u arhivi.

Dohvatanje brakova.

Implementirati metode `vector<Person*> MarriagesArchive::getMarriedMen();` i `vector<Person*> MarriagesArchive::getMarriedWomen();` koje dohvataju sve muškarce, odnosno žene, koji su trenutno u braku.

Specifikacija funkcionalnosti klase TreeNode.

Čvor porodičnog stabla sadrži informacije o osobi koje se čuvaju po adresi (*personInfo: Person**), pokazivač na čvor prvog starijeg brata/sestre (*olderSibling*), pokazivač na čvor prvog mlađeg brata/sestre (*youngerSibling*), i pokazivač na čvor najstarijeg deteta (*oldestChild*).

Stvaranje čvora.

Implementirati konstruktor `TreeNode(Person* p);` koji inicijalizuje informacije o osobi na osnovu prosleđenog argumenta. Prilikom stvaranja, svi ostali pokazivači su `nullptr`.

Čvor se može napraviti kao kopija drugog čvora.

Implementirati konstruktor kopije koji kopira samo informacije o osobi (ostali pokazivači kopije su `nullptr`).

Dodavanje brata/sestre, dodavanje deteta.

Implementirati metodu `void TreeNode::addChild(Person* child);` koja u čvor dodaje novo dete. Greška je ukoliko je razlika u godinama osobe kojoj dodajemo dete i deteta manja od osamnaest godina (*UnderageParentException*). Voditi računa da odgovarajući pokazivač čvora stabla posle dodavanja deteta pokazuje na najstarije dete.

Implementirati metodu `void TreeNode::addSibling(Person* sibling);` koja uvezuje novog brata/sestru u čvor stabla u zavisnosti od njihove starosti. Greška je ukoliko je razlika u godinama između braće i sestara manja od godinu dana (*InvalidSiblingAgeException*). Voditi računa da nakon dodavanja brata/sestre sva braća i sestre i dalje budu uređeni po datumu rođenja.

Dohvatanje broja braće/sestara i dece.

`int TreeNode::numberOfSiblings();` i `int TreeNode::numberOfChildren();`

Dohvatanje i postavljanje informacija o čvoru.

```
Person* TreeNode::getPersonInfo();
TreeNode* TreeNode::getOlderSiblingNode();
TreeNode* TreeNode::getYoungerSiblingNode();
TreeNode* TreeNode::getOldestChildNode();
void TreeNode::setOldestChildNode(TreeNode* tn);
```

Dohvatanje specifičnih čvorova.

Implementirati metode koje dohvataju čvor najstarijeg brata/sestre i najmlađeg brata/sestre tekućeg čvora. Ukoliko čvor nema starijeg/mlađeg brata/sestru vratiti tekući čvor.

`TreeNode* TreeNode::getOldestSiblingNode();` i

`TreeNode* TreeNode::getYoungestSiblingNode();`

Implementirati metodu koja dohvata čvor deteta na osnovu prosleđenih informacija o detetu. Ukoliko prosleđena osoba nije dete tekućeg čvora vratiti `nullptr`.

`TreeNode* TreeNode::getChildNode(Person* child);` i

Implementirati metodu koja dohvata čvor brata/sestre na osnovu prosleđenih informacija o bratu/sestri. Ukoliko prosleđena osoba nije brat/sestra tekućeg čvora vratiti `nullptr`.

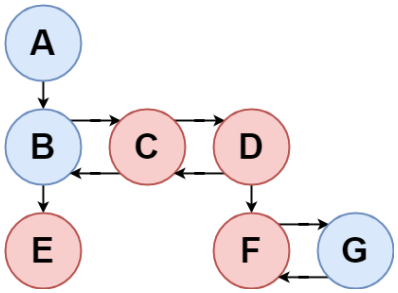
`TreeNode* TreeNode::getSiblingNode(Person* sibling);`

Obezbediti propisno brisanje čvora.

Implementirati destruktor koji prilikom brisanja na propisan način razvezuje sve pokazivače (Nijedan dinamički alociran objekat ne treba da se uništi).

Specifikacija klase FamilyTree

Porodično stablo (*FamilyTree*) je ulančana nelinearna struktura podataka koja se sastoji iz povezanih čvorova (*TreeNode*). Klasa stabla ima privatno polje `root` koje predstavlja pokazivač na koreni čvor stabla, i prezime porodice (*familyName*). Deca u porodičnom stablu mogu da se vode i po očevoj i po majčinoj liniji.



Slika 1 Primer porodičnog stabla. Koreni (root) čvor je čvor A. Pokazivači na čvorove u istom nivou predstavljaju pokazivače na braću/sestre. Pokazivači na čvorove nižih nivoa predstavljaju pokazivače na najstarije dete.

Stablo se stvara na osnovu zadatog ulaznog niza .

Implementirati konstruktor klase

`FamilyTree(string familyName, vector<int> format, vector<Person*> family);`.

Vektor *family* sadrži osobe koje se ubacuju u porodično stablo. Osobe se ubacuju po formatu vektora *format* na način opisan u nastavku. Vektor *format* sadrži po jednu celobrojnu vrednost za svakog člana porodice iz vektora *family*. Ta celobrojna vrednost predstavlja broj dece koje odgovarajući član vektora *family* treba da ima. Deca se biraju kao sledećih nekoliko neobrađenih članova vektora *family*. Za primer vektora *format* : 3, 1, 0, 2, 0, 0, 0 i vektora *family* : A, B, C, D, E, F, G, treba da se dobije stablo sa slike. Greška je ukoliko neki član vektora *family* ima drugačije prezime u odnosu na prezime porodice (*InvalidFamilyNameException*). Greška je ukoliko vektori *format* i *family* nisu iste veličine (*InvalidVectorSize*). *Ne* sme se koristiti rekurzija – implementirati iterativno rešenje.

Stablo se može stvoriti kao kopija već postojećeg stabla.

Implementirati obe varijante konstruktora kopije (duboko kopiranje i kopiranje sa premeštanjem) za potrebe kloniranja stabala. Ne sme se koristiti rekurzija – implementirati iterativno rešenje.

Dodavanje brata/sestre ili deteta odgovarajućoj osobi u stablu.

Implementirati metode za dodavanje brata/sestre ili deteta odgovarajućoj osobi u stablu.

```
bool FamilyTree::addSibling(Person* person, Person* sibling);
bool FamilyTree::addChild(Person* parent, Person* child);
```

Prvi argument metode je osoba kojoj se dodaje brat/sestra ili dete, a drugi argument je osoba koja se dodaje. Metoda vraća false ukoliko u stablu ne pronađe osobu poslatu kao prvi argument.

Pronalaženje osobe i njenog roditelja u stablu.

Implementirati metodu koja traži određenu osobu u stablu i vraća pokazivač na njen čvor u slučaju uspešne pretrage, a nullptr u slučaju neuspešne pretrage.

```
TreeNode* TreeNode::lookUp(Person* p);
```

Implementirati metodu koja traži roditelja određene osobe u stablu i vraća pokazivač na njegov čvor u slučaju uspešne pretrage, a nullptr u slučaju neuspešne pretrage.

```
TreeNode* TreeNode::lookUpParent(Person* p);
```

Izbacivanje osobe iz stabla.

Implementirati metodu `bool FamilyTree::remove(Person* p);` koja pronalazi i iz stabla briše osobu prosleđenu kao argument zajedno sa njenim podstablom dece. Ukoliko se iz primera sa slike izbaci osoba B, stablo će da izgleda na sledeći način (izbačene su osobe B i njegovo dete E):

```
A
  C
  D
    F
    G
```

Ne sme se koristiti rekurzija – implementirati iterativno rešenje.

Ispisivanje stabla.

Implementirati ispis čvorova stabla preorder obilaskom. U svakom redu treba ispisati informacije o jednoj osobi. Generacije u stablu odvajati znakovima za tabulaciju. Za dato stablo sa slike ispis treba da izgleda na sledeći način (gde su A, B, C,... ispisi za osobe po odgovarajućem formatu):

```
A
  B
    E
  C
  D
    F
    G
```

Ne sme se koristiti rekurzija – implementirati iterativno rešenje.

```
void print(ostream& os, FamilyTree& ft);
```

Brak između dve porodice.

Implementirati funkciju koja proverava koliko brakova je sklopljeno između dve porodice.

```
int numberOfMarriages(FamilyTree& ft1, FamilyTree& ft2);
```

Omogućiti propisno uništavanje stabla.

Implementirati destruktor koji treba da oslobodi sve čvorove stabla. Ne sme se koristiti rekurzija – implementirati iterativno rešenje.

Test funkcija

U projektu glavnog programa treba da postoji funkcija `void test();` koja testira rad sa datumima, osobama, uslužnim klasama i porodičnim stablima. Studenti treba da implementiraju datu funkciju i uslovno je prevode ako je definisan makro STUDENT_TEST. Predvideti i postojanje makroa PROF_TEST. Ako su oba makroa definisana, PROF_TEST ima prioritet i tada se prevodi tajni test primer koji će biti dat na odbrani. Funkcija main treba samo da pozove test funkciju.

Tehnički zahtevi i smernice za izradu rešenja

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka. Svaka klasa koja koristi dinamičku memoriju mora biti bezbedna za korišćenje. Programski kod klasa rasporediti u odgovarajuće **.h** i **.cpp** fajlove. Nije dozvoljeno korišćenje globalnih promenljivih za razmenu podataka. Ukoliko u zadatku nešto nije dovoljno jasno definisano, treba usvojiti razumnu pretpostavku i na temeljima te pretpostavke nastaviti izgrađivanje svog rešenja.

VAŽNE NAPOMENE

Za uspešno odbranjen domaći zadatak potrebno je na odbrani pokazati kod podeljen na odgovarajuće projekte, .h i .cpp fajlove.

- Klase kojima su implementirani osnovni koncepti (*Date, Person, PersonalIdArchive, MarriagesArchive*) treba da budu smeštene u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`person.lib`**).
- Klase koje implementiraju stablo (*TreeNode, FamilyTree*) treba da budu smeštene u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`tree.lib`**).
- Glavni program napisati u posebnom projektu koji se prevodi kao Win32 Console Application (**`dz2test.exe`**) i povezati sa bibliotekama za stabla i osnovne koncepte. Glavni program treba implementirati tako da pozove globalnu funkciju `void test();`.
- Studenti treba da implementiraju svoju verziju ove funkcije tako da demonstriraju sve funkcionalnosti sistema.
- Od kolekcija iz STL-a smeju da se koriste `<vector>`, `<list>`, `<stack>`, `<queue>` i `<string>`
- NIJE DOZVOLJENO SMESTITI CEO KOD U JEDAN PROJEKAT ILI CPP fajl!

25.11.2018. godine sa predmeta