UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE

Objektno-orijentirano programiranje

Zadaća 5

Tuzla, januar/sijecanj 2022.

Sadržaj

Zadatak 1

Potrebno je implementirati klasu MojNizInt koja će predstavljati kontejner integer-a. Kontejner treba da podržava:

1. Default konstruktor
2. Metod `size()` koji nazad vraća broj elemenata prisutnih unutar kontejnera
3. Metod `at` koji uzima indeks elementa kao argument te nazad vraća taj element po referenci. Ukoliko je kao argument proslijeđen indeks van granica kontejnera potrebno je generirati iznimku tipa `std::invalid_argument`. Obratiti pažnju da se ovom metodu može pristupiti i iz const konteksta.
4. Konstruktor koji uzima `std::initializer_list<int>` kao argument te kreira kontejner sa datim elementima. Veličina kontejnera mora odgovarati broju proslijeđenih elemenata kroz inicijalizacijsku listu.
5. Copy, move konstruktore, copy i move operatore dodjeljivanja te destruktor.
6. Operator uglasta zagrada [] koji uzima indeks elementa kao argument te nazad vraća taj element po referenci. Obratiti pažnju da se ovom operatoru može pristupiti i iz const konteksta.
7. Operator klase `MojNizInt` za množenje sa skalarom (`operator*`). Rezultat treba da bude nova instanca klase `MojNizInt` koja ima isti broj elemenata kao i objekat sa kojim je množenje izvršeno pri čemu svaki element treba da bude multipliciran za dati skalar. Operatoru je potrebno moći pristupiti i iz const konteksta.
8. Operator sabiranja dvije instance klase `MojNizInt` (`operator+`). Operator treba da baci iznimku tipa `std::invalid_argument` ukoliko oba niza koja učestvuju u operaciji sabiranja nisu iste dužine. Rezultat treba da bude nova instanca klase `MojNizInt` koja ima isti broj elemenata kao i dva objekta koja su učestvovala u operaciji sabiranja, stim da svaki član treba da bude zbir članova na odgovarajućim pozicijama. Operatoru je potrebno moći pristupiti iz const konteksta.
9. Sufix `operator++` koji će uvećati svaki član klase `MojNizInt` za jedan. Povratna vrijednost treba da bude instanca objekta `MojNizInt` čije stanje odgovara starom stanju objekta nad kojim je `operator++` pozvan.
10. Prefix `operator++` koji će uvećati svaki član klase `MojNizInt` za jedan. Povratna vrijednost treba da bude referenca na isti objekat nad kojim je operator pozvan.
11. Metod `push_back` koji uzima integer kao argument. Metod treba dinamički da alocira dodatno `size() + 1` memorije, kopira stari niz u ovaj novo-alocirani dio memorije te na kraj smjesti proslijeđeni argument. Ne zaboraviti dealocirati stari niz nakon kopiranja. Konačno, potrebno je i size niza uvećati za `1`.

Za sve iznad spomenute stvari implementirani su unit testovi koristeći `doctest` alat i nalaze se u prilogu zadaće pod imenom `test1.cpp`. Obratiti pažnju na alociranje i dealociranje resursa jer, iako moguće, nije baš trivijalno ispitati unit testovima tako da se memory-leakage ne testira.

Zadatak 2

Potrebno je poboljšati `push_back` implementaciju klase `MojNizInt` na način da se uvede dodatan član unutar klase koji će predstavljati maksimalni kapacitet klase `MojNizInt`. Implementirati preemptive alociranje memorije koristeći ovaj novi član. Član size (na predavanju korištena varijable n) treba da govori koliko elemenata se trenutno nalazi u kontejneru, dok kapacitet govori koliko maksimalno elemenata može stati u kontejner, odnosno, koliko je članova prealocirano za buduće dodavanje elemenata. Potrebno je slijediti naredna pravila:

1. Default konstruisan `MojNizInt` treba da ima kapacitet od jednog elementa, samim time i alociran niz od jednog elementa.
2. MojNizInt konstruisan koristeći `std::initializer_list<int>` treba da ima kapacitet jednak veličini proslijeđene liste za inicijalizaciju. U ovom slučaju size i kapacitet trebaju imati istu vrijednost.
3. Kada se dodaje novi element koristeći `push_back` potrebno je pogledati da li imamo slobodnih slotova u prealociranom nizu. U slučaju da imamo, element smjestimo na prvi slobodni slot, te uvećamo size za `1`. U slučaju da je size došao do kapaciteta, potrebno je alocirati dodatan komad memorije koji je duplo veći od prethodno alociranog niza (samim time i kapacitet uvećati duplo), kopirati cijeli stari niz u prvi dio novog komada memorije, te nadodati proslijeđeni argument `push_back` metoda. Osloboditi staru memoriju nakon kopiranja!
4. Kod implementacije `pop_back` metoda potrebno je samo size smanjiti za jedan, kapacitet se ne treba mijenjati niti memorija realocirati.

Dodatno implementirati i metode `front` i `back` koji vraćaju referencu na početak (prvi element) odnosno kraj (zadnji element) kontejnera. U slučaju da je kontejner prazan funkcija ne treba imati definirano ponašanje. Unit testovi se nalaze u file-u `test2.cpp`. Obratiti pažnju da zadnji test prethodnog zadatka ne treba da prolazi nakon pravilne implementacije ovog zadatka. Pogledati ponovno implementaciju copy i move konstruktora i operator= nakon dodavanja capacity parametra u klasu.

Zadatak 3

Implementirati klasu MojNiz koja predstavlja generičku implementaciju MojNizInt za bilo koji tip. Klasa MojNiz može čuvati bilo kakve podatke koji podržavaju sve operacije koje klasa MojNiz zahtjeva. Primjer instanciranja ovakve klase:

```
MojNiz<double> mojDouble{1.1, 1.2, 1.3};
MojNiz<short> mojShort{2, 5, 7};
```

Osim svih operacija koje je podržavala klasa MojNizInt, klasa MojNiz treba dodatno da podržava:

1. Copy konstruktor koji uzima instancu bilo koje klase MojNiz te kopira elemente u svoje stanje ukoliko je moguće izvršiti konverziju argumenata.
2. Copy operator= koji uzima instancu bilo koje klase MojNiz te kopira elemente u svoje stanje ukoliko je moguće izvršiti konverziju argumenata. Obratiti pažnju da je potrebno dealocirati stari niz.
3. Operator sabiranja **bilo koje** dvije instance klase `MojNiz` (`operator+`) ukoliko se elementi dvije instance klase `MojNiz` mogu sabirati.. Operator treba da baci iznimku tipa `std::invalid_argument` ukoliko oba niza koja učestvuju u operaciji sabiranja nisu iste dužine. Interni tip rezultujuće klase `MojNiz` treba da bude tip koji se dobije iz operacije:
   ```
   first[0] + second[0];
   ```
   Gdje je: `MojNiz<T> first; MojNiz<U> second;`
   Tako na primjer za zbir `MojNiz<double>` i `MojNiz<int>` uvijek rezultat treba da bude `MojNiz<double>`.
   Operatoru je potrebno moći pristupiti iz const konteksta.

Unit testovi se nalaze u prilogu zadaće u file-u `test3.cpp`.

Zadatak 4

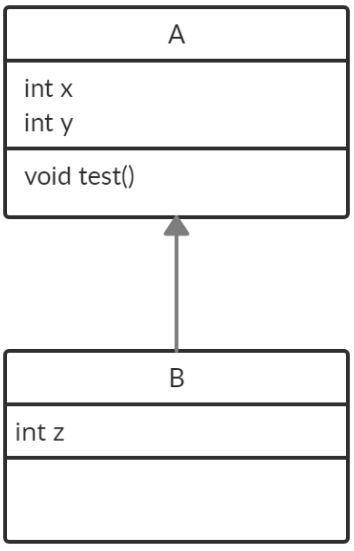
Sljedeći zadatak je osmišljen kako bi testirao razumijevanje koncepta nasljeđivanja, te modeliranja problema iz pravog života uz pomoć objektno orijentiranog programiranja. Nije potrebno pisati kod, samo nacrtati skicu odnosno koncept kako bi program bio osmišljen te uporediti sa rješenjem na zadnjoj strani.

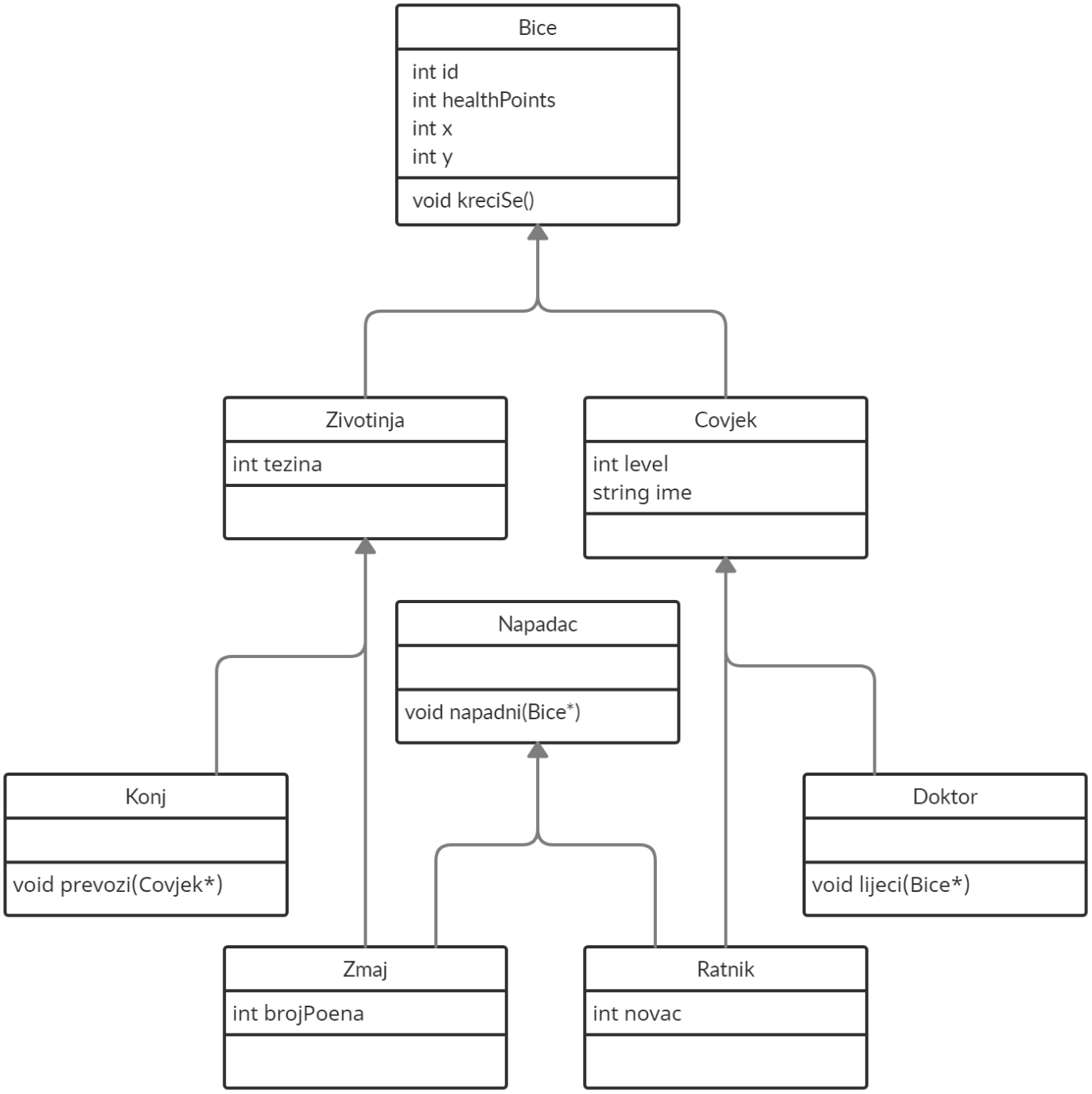
Potrebno je modelirati živa bića u video igri i njihovo ponašanje. U igri postoje ljudi i životinje. Čovjek može igrati ulogu ratnika, ili doktora. Od životinja postoje zmaj i konj. Sva živa bića se mogu kretati, dok samo zmaj i ratnik mogu napadati (mogu napasti bilo koje biće). Svako živo biće ima svoj ID i poene zdravlja (health points), ime, i x y koordinate koje predstavljaju poziciju bića na karti. Unikatna sposobnost doktora je mogućnost liječenja bilo kojeg živog bića, dok je unikatna sposobnost konja da prevozi jedno ljudsko biće. Ljudska bića su opisana i sa imenom i nivoom iskustva (level), dok samo ratnik ima novac kao atribut koji ga opisuje. Od životinja samo zmaj ima broj poena iskustva koje osvaja ratnik kada ga pobijedi. Sve životinje imaju atribut težina koji brojčano opisuje koliko je teško istu životinju pobijediti.

Primjer dijagrama za sljedeći kod:

```
class A{
  int x;
  int y;
  void test();
};

class B : public A{
  int z;
};
```




Bice
int id
int healthPoints
int x
int y
void kreciSe()
Zivotinja
int tezina
Covjek
int level
string ime
Napadac
void napadni(Bice*)
Konj
void prevozi(Covjek*)
Doktor
void lijeci(Bice*)
Zmaj
int brojPoena
Ratnik
int novac