PROGRAMSKI JEZICI 1

Zadatak 4 – Izuzeci

Postavka

Kreirati projekat pod nazivom *Zadatak4* unutar rješenja (i.e. *Visual Studio solution*) pod nazivom *PJ1Zadaci2020*, te kreirati projekat pod nazivom *Zadatak4.Demo* unutar istog rješenja. Projekat *Zadatak4.Demo* treba da referencira biblioteku iz projekta *Zadatak4* i demonstrira njen rad.

Zadatak

Unutar projekta *Zadatak4*, kao statičku biblioteku, implementirati klase za rad sa dinamičkim kolekcijama, i to prema sljedećim uputstvima.

- → Implementirati klasu **`Complex`** koja predstavlja kompleksan broj sa svim relevantnim operacijama.
- → Implementirati klasu **`BufferedCollection`** koja predstavlja neuređenu kolekciju elemenata tipa **`weak_ptr<Complex>`**.
  - ⇝ Elementi kolekcije predstavljaju slabe pametne pokazivače na objekte tipa Complex.
  - ⇝ Pri radu sa pametnim pokazivačima provjeravati validnost originalnog objekta prije pristupa elementu.
- → Klasa **`BufferedCollection`** treba da obezbijedi odgovarajuće funkcije za umetanje, izbacivanje i pristup elementima kolekcije.
- → Koristiti dinamičku alokaciju memorije, uz poštovanje odgovarajućih pravila OOP programiranja u programskom jeziku C++.
- → Klasa treba da koristi uvezanu listu pokazivača na dinamički alocirane nizove veličine proslijeđene kroz konstruktor.
  - ⇝ Nizovi na koje pokazuju elementi uvezane liste predstavljaju bafere u kojima se čuvaju podaci. Ukoliko je u baferisanoj kolekciji umetnuto n elemenata, a veličina bafera je k, pri čemu je $k < n$ tada će trenutni broj elemenata liste da bude $\left\lceil \frac{n}{k} \right\rceil$, gdje će svaki da pokazuje na niz dužine k. Pristup elementu sa indeksom iz opsega $[0, k-1]$ izvešće se nad prvim baferom u listi, dok će pristup elementu sa indeksom iz opsega $[k, 2k-1]$ biti izveden nad drugim baferom u listi. Na ovaj način, baferisana kolekcija je segmentisana u podnizove dužine k.
  - ⇝ Pri dodavanju novog elementa u kolekciju, ukoliko je neophodno proširivanje bafera, izvršiti sve potrebne alokacije memorije uz obradu i dalju propagaciju izuzetaka pri pristupu memoriji.
  - ⇝ Pri izbacivanju elemenata, nije neophodno smanjivati bafer, ali je neophodno pomjeriti preostale elemente.
- → Pri svakom instanciranju memorije, uhvatiti odgovarajući izuzetak i spriječiti curenje memorije.
- → Spriječiti curenje memorije u konstruktoru, u slučaju da alokacija baca izuzetak. Omogućiti propagaciju (*rethrowing*) izuzetaka koje bacaju standardni operatori, uz sprječavanje curenja memorije.
- → Implementirati operator indeksiranja.
  - ⇝ Spriječiti izlazak iz opsega pri indeksiranju korištenjem mehanizma izuzetaka.
  - ⇝ Pri izlasku iz opsega, podići izuzetak klase izvedene iz **`std::out_of_range`**, tako da se u objektu izuzetka čuva indeks sa kojim je pokušan pristup.

- ⇝ U slučaju da se pristupa elementu za koji je originalni objekat nevažeći, izbaciti dati element iz kolekcije i podići izuzetak izveden iz odgovarajuće klase izuzetka iz standardne biblioteke. Koristiti metode `lock()` i `expired()`.

→ Ispravno označiti sve metode, s obzirom na korištenje izuzetaka.

Sve implementirane klase treba da poštuju pravilo trojke i pravilo petorke. Sve implementirane klase treba da budu objedinjene u jedinstven prostor imena, te treba da imaju ispravno razdvojena zaglavlja od implementacije. Pored toga, implementirane klase treba da obezbjeđuju mogućnost korištenja u obliku konstantnih objekata. Preklopiti operatore u svim situacijama kada je to semantički ispravno. Izbjeći svako dupliranje koda.

U projektu *Zadatak4.Demo* demonstrirati rad svih implementiranih klasa i metoda individualno. Ilustrovati sve slučajeve kada dolazi do bacanja izuzetka i obraditi ove izuzetke. Pri izvršavanju programa ne smije doći do poziva funkcije **`terminate()`**.