UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE

Objektno-orijentirano programiranje

Zadaća 4

Tuzla, januar 2023.

Sadržaj

Zadatak 1

Implementirati korisnički definirani tip podatka *LongNumber* koji treba da podržava sve neophodne operacije (sabiranje, oduzimanje, množenje, dijeljenje, ispis na standardni izlaz, unos sa standardnog ulaza…).
U prilogu zadaće se nalazi deklaracija svih metoda, operatora i konstruktora koje novi tip podatka treba da podržava.
Definiciju svih metoda, operatora i konstruktora odraditi u fajlu *LongNumber.cpp*.
Neophodno je adekvatno testirati sve funkcionalnosti u fajlu *main.cpp*.

Zadatak 2

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 3

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 4

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 5

Napisati klasu `Matrica` koja ima sljedeće karakteristike:

- Klasa implementira matricu generičkog tipa T, dimenzija m x n, pomoću dinamički alociranog jednodimenzionalnog C++ niza elemenata T.
- Objekte klase `Matrica` moguće je konstruisati na više načina:
  - pomoću dimenzija matrice m i n, dok se elementi matrice inicijaliziraju pomoću default konstruktora klase T.
  - pomoću `std::initializer_list`, pri čemu prva dva elementa predstavljaju broj redova i kolona u matrici koja se kreira, a ostatak se koristi za inicijalizaciju elemenata u matrici. Ukoliko je broj elemenata manji od ukupne veličine matrice, nenavedeni članovi se inicijaliziraju pomoću default konstruktora klase T. Ako je broj elemenata veći od ukupne veličine matrice, generirati iznimku tipa `std::domain_error` sa odgovarajućom porukom.
  - na osnovu već postojećeg objekta tipa Matrica.
- Operator () omogućava čitanje i dodjeljivanje vrijednosti elementu matrice u i-tom redu i j-toj koloni. Obezbijediti verzije za konstantan i nekonstantan pristup elementima.
- Operatori + i - omogućavaju sabiranje i oduzimanje dvije matrice. Ukoliko su dimenzije matrica nekompatibilne, pri sabiranju i oduzimanju generirati iznimku tipa `std::domain_error` sa odgovarajućom porukom. Operatori + i - trebaju biti konstantni metodi za objekat nad kojim se pozivaju. Implementirati operatore += i -= koji će vršiti izmjene nad objektom nad kojim su pozvani.
- Operatori * i / kao argument uzimaju jedan element tipa T i vrše operacije množenja i dijeljenja na svaki pojedinačni element matrice. Na primjer, množenje matrice cijelih brojeva

brojem 2 će rezultovati izmjenama na matrici na način da će svi elementi biti pomnoženi brojem 2.

- Objekte tipa Matrica moguće je ispisati pomoću operatora <<.

Implementirati sve dodatne metode koje smatrate potrebnim kako bi klasa pravilno dealocirala dinamički alocirane resurse.

Implementaciju klase `Matrica` testirati sa `main` funkcijom u prilogu zadaće. Ispravno napisana implementacija treba prouzrokovati ispis.

Zadatak 6

Implementirati kontejner pod imenom Lista. Kontejner je implementiran pomoću jednostruko povezanih lokacija (čvorova) gdje svaki čvor čuva vezu na naredni čvor.
Kontejner čuva vezu na prvi čvor head_ koji predstavlja početak liste i tail_ koji predstavlja posljednji čvor u listi.
Neophodno je implentirati naredne funkcionalnosti:

1. Default konstruktor koji kreira praznu listu (head_ i tail_ su jednaki nullptr)
2. Copy i move konstrukor, te copy i move operator =
3. Destruktor koji dealocira sve dinamički alocirane čvorove (voditi računa da se ne izgubi veza do narednog čvora)
4. Metod push_back() koji dodaje novi čvor na kraj liste. Nakon dodavanja novog čvor, tail_ čvor mora pokazivati na novo-alocirani čvor.
5. Metod push_front() koji dodaje novi čvor na početak liste. Nakon dodavanja novog čvor, head_ čvor mora pokazivati na novo-alocirani čvor.
6. Metod pop_back() koji uklanja posljednji čvor iz liste. Da bi se uklonio posljednji čvor u listi neophodno je sekvencijalno proći kroz sve čvorove u listi [head_, tail_). Dakle, da bismo uklonili posljednji čvor moramo se nalaziti jedan čvor ispred tail_ čvora kako bismo adekvatno prekinuli vezu njegovog prethodnog čvora. Ako je lista prazna, metod baca iznimku tipa std::logic_error.
7. Metod pop_front() koji uklanja prvi čvor iz liste. Za uklanjanje prvog čvora u listi pomjeriti head_ za jedan čvor unaprijed. Ako je lista prazna, metod baca iznimku tipa std::logic_error.
8. Metod empty() koji provjerava da li je lista prazna.
9. Metod size() koji vraća ukupan broj čvorova u listi.
10. Metod extend() koji na postojeću listu nadodaje elemente druge liste.
11. Logički operator == koji provjerava da li su dvije liste jednake..
12. Logički operator != koji provjerava da li su dvije liste različite.
13. Metod front() koji vraća referencu na prvi element u listi, bez uklanjanja čvora. Ako je lista prazna, metod baca iznimku tipa std::logic_error.
14. Metod back() koji vraća referencu na posljednji element u listi, bez uklanjanja čvora. Ako je lista prazna, metod baca iznimku tipa std::logic_error.
15. Privatni metod copyList() koji kreira novu kopiju liste od proslijeđene liste. Metod se može iskoristiti prilikom definicije copy konstruktora i copy operatora =.

16. Privatni metod dealoc() koji dealocira sadržaj liste. Metod se može koristiti prilikom definiranja destrukora ili prilikom implementacije operatora = gdje je neophodno prije alociranja novog sadržaja liste dealocirati staro stanje liste.

U prilogu zadaće je data deklaracija svih metoda, operatora i konstruktora u fajlu Lista.hpp. Također, u fajlu test_lista.cpp je testirana funkcionalnost liste. Svi testovi trebaju uspješno proći nakon završene implementacije. Definiciju metoda iz klase Lista navesti u fajlu Lista.cpp.
Za kompajliranje testova nakon završene implementacije odraditi komandu:

```
g++ -std=c++14 -o lista test_lista.cpp Lista.cpp
```