### Zadatak 1

Implementirati generički kontejner DoubleLinkedList. Neophodno je implementirati sve deklarisane metode da bi svi testovi uspješno prošli. Za način implementacije svakog od metoda provjeriti odgovarajući test gdje je navedeno ponašanje svakog od metoda. Napisati testni program u fajlu main.cpp gdje je neophodno testirati svaki od metoda, te uspostaviti analogiju sa std::list kontejnerom.

### Zadatak 2

Implementirati klasu Number koja će čuvati jedan cijeli broj. Broj je potrebno čuvati na način da su cifre elementi dvostruko linkane liste iz prvog zadatka.

Objekat tipa Number treba da se ponaša kao i primitivni tipovi, treba imati sve aritmetičke i logičke operatore.

Primjer upotrebe:

```
Number num1{123};
Number num2{457};
Number num3{-5};
Number num4 = num1 + num2 + num3;
num4 == Number{575}; //true
```

### Zadatak 3

Implementirati program za planiranje putovanja kroz Bosnu i Hercegovinu. Potrebno je implementirati klasu TripPlanner u kojoj cete koristiti dvostruko linkanu listu iz prvog zadatka da cuvate podatke o znacajnim mjestima na putu. Program treba da ima moguc-nost dodavanja lokacija (sa geografskim koordinatama), dodavanja mjesta zaustavljanja na putu, uklanjanje mjesta, ispis cijelog puta, iscrtavanje puta na karti i ispis ukupne duzine puta. Kada se program pokrene treba da ima izgled slican ovom:

```
Choose option number:
1. Add Location
2. Add Stop
3. Insert Stop
4. Remove Stop
5. Print Trip
6. Show Trip Distance
7. Render Trip
8. Print Locations
0. Exit
```

Objasnjenje svake od opcija: 1. Add Location: omogucava dodavanje nove lokacije u spisak mogucih mjesta za posjetiti (format unosa je proizvoljan), u prilogu je dat fajl locations.txt sa nekoliko mjesta unutar drzave kojeg mozete koristiti za ucitavanje pri pokretanju programa 2. Add Stop: omogucava dodavanje stanice na putu, mjesto se bira prema imenu na osnovu ucitanih/poznatih lokacija 3. Insert Stop: za razliku od prethodne naredbe, omogucava dodavanje stanice negdje osim na kraj trenutne putanje 4. Remove Stop: omogucava uklanjanje stanica sa putanje 5. Print Trip: ispisuje sve stanice na putanji 6. Show Trip Distance: ispisuje ukupnu duzinu puta od prvog mjesta do zadnjeg, za racunanje udaljenosti koristiti metod objasnjen ovdje 7. Render Trip: omogucava ispis putanje u html fajl sa formatom objasnjenim ispod, otvaranjem ovog fajla u browseru se vidi putanja od prve do zadnje stanice na karti BiH 8. Print Locations

omogucava ispis svih dostupnih lokacija

**Iscrtavanje putanje na karti**

Da bismo ispravno iscrtali putanju na karti potrebno je koristiti template html fajl. Izlazni html fajl treba da bude oblika

```
<sadrzaj-template-fajla>
<vas-sadrzaj>
</svg>
```

u dijelu vas-sadrzaj potrebno je dodati linije oblika

```
<line x1="" y1="" x2="" y2="" stroke="orange" stroke-width="4"></line>
```

gdje cete za vrijednosti x1, y1, x2, y2 unijeti kranje tacke linije koja povezuje dvije uzastopne lokacije u putanji. Za racunanje ovih vrijednosti koristiti sljedece jednacine

```
x = (longitude_ - WESTEDGE)/(EASTEDGE - WESTEDGE) * IMAGEWIDTH;
y = (latitude_ - NORTHEDGE)/(SOUTHEDGE - NORTHEDGE) * IMAGEHEIGHT;
```

gdje vrijede konstante

```
IMAGEWIDTH=1063.
IMAGEHEIGHT=1014.
NORTHEDGE=45.4
SOUTHEDGE=42.4
WESTEDGE=15.5
EASTEDGE=19.9
```

Po zelji moguce je napisati i potreban kod za snimanje i ucitavanje putovanja.

## Zadatak 4

Implementirati Red strukturu podataka korištenjem niza. Neophodno je implementirati sve deklarisane metode kako bi svi testovi uspješno prošli. Napisati testni program main.cpp koji testira sve funkcionalnosti te odraditi analogiju sa std::queue strukturom podataka.

## Zadatak 5

Implementirati generički kontejner Stack korištenjem niza. Neophodno je implementirati sve deklarisane metode da bi svi testovi uspješno prošli. U fajlu main.cpp testirati sve metode te odraditi analogiju sa std::stack kontejnerom.

## Zadatak 6

Implementirati postfix kalkulator sa cijelim brojevima i sa osnovnim operacijama (+,-,*,/). Postfiksna notacija ili postfiksni sistem oznaka je matematička notacija u kojoj svaki operator slijedi nakon svih svojih operanada, te je poznatija kao obrnuta poljska notacija (ili samo RPN od eng. Reverse Polish notation). Objašnjenje i upute za implementaciju ovog kalkulatora možete pronaći na poljska notacija.

Ovu implementaciju realizovati u klasi PostfixCalc. Implementirati metode klase, kao i način korištenja prema svojoj intuiciji. Unos operanada i operatora se vrši preko standardnog ulaza. Argumenti na ulazu su razmaknuti praznim mjestom. Operandi (brojevi) mogu biti pozitivni i negativni brojevi, dok su osnovne operacije jedan karakter. Potrebno je implementirati čitanje ulaza na način da svaka linija ulaza predstavlja jedan matematički izraz za koju se ispisuje rezultat izraza ili informacija o grešci, nakon čega

korisnik unosi narednu liniju ulaza, koja je novi izraz.

Zadatak možete riješiti korištenjem kontejnera iz standardne biblioteke.

### Zadatak 7

Haris nastavlja raditi na svom novom programskom jeziku, boltić. Program u boltiću može sadržavati samo zagrade svih vrsta:

- uglaste zagrade < >,
- oble zagrade ( ),
- uglaste zagrade [ ],
- vitičaste zagrade { }.

Program koji sadrži samo zagrade, da bi se uspješno kompajlirao, mora imati sve zagrade pravilno umetnute. Da bi izraz bio pravilan, svaka otvorena zagrada mora imati svoju uparenu zatvorenu zagradu, te se mora poštovati hijerarhija zagrada.

Na žalost, kompajler koji Haris pravi još uvijek ne može reći koji izraz u program ima pogrešno umetnute zagrade već jednostavno ne prihvata nepravilan program. Pomozite Harisu da napravi program koji će odrediti koji izraz nije pravilan.

Zagrade poštuju hijerarhiju: uglaste zagrade <> su najviše unutrašnje zagrade, pa onda oble (), pa onda uglaste [] pa onda vitičaste {}. Drugim riječima, vitičasta zagrada ne može biti otvorena unutar nekog drugog tipa zagrada. Uglaste zagrade [] ne mogu biti otvorene unutar oblih () ili uglastih <> zagrada, ali može biti otvorena unutar vitičastih zagrada. Obla zagrada ne može biti otvorena unutar para uglastih zagrada <>.

Potrebno je učitavati linije sa standardnog ulaza, gdje svaka linije predstavlja jedan izraz. Karakteri na ulazu su neki od '[ ] ( ) { } < >'. Na izlazu treba ispisati samo „dobar" ili „pogrešan". Program se završava tako što pročitamo sve karaktere sa ulaza, odnosno kad korisnik završi unos slanjem EOF karaktera (ctrl+d na Linuxu ili ctrl+z na Windowsu).

Zadatak možete riješiti korištenjem kontejnera iz standardne biblioteke.