UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE


# Objektno-orijentirano programiranje

Zadaća 3

Tuzla, decembar/prosinac 2020.

# Sadržaj


**Sadržaj** **2**

**Zadatak 1** **3**

**Zadatak 2** **3**

**Zadatak 3** **4**

**Zadatak 4** **4**

**Zadatak 5** **4**

**Zadatak 6** **5**

**Zadatak 7** **5**

**Zadatak 8** **5**

**Zadatak 9** **7**

**Zadatak 10** **7**

**Zadatak 11** **8**

# Zadatak 1

Napisati funkciju `parni_neparni` koja uzima dva iteratora na početak i kraj kontejnera `std::list<int>` kao argumente te nazad vraća novi `std::list<int>` kontejner takav da su svi parni članovi na lijevoj strani, a svi neparni na desnoj strani kolekcije. Funkcija ne treba modifikovati elemente proslijeđene kolekcije na koje iteratori pokazuju.

Kako treba izgledati potpis funkcije? Zašto?

A. `std::list<int> parni_neparni(std::list<int>::iterator, std::list<int>::iterator);`
B. `std::list<int> parni_neparni(std::list<int>::const_iterator, std::list<int>::const_iterator);`

# Zadatak 2

Napisati poboljšanu verziju prethodne funkcije takvu da osim iteratora na `std::list` uzima i `predicate` funkciju kao argument (`std::function<bool(int)>` iz `functional` zaglavlja). Funkcija treba da poziva `predicate` nad svakim članom ulaznog argumenta, te ukoliko je rezultat `predicate` poziva tačan, član treba da se nalazi na lijevoj strani, odnosno ukoliko je netačan na desnoj strani rezultujuće kolekcije.

Primjer poziva `partition` funkcije:

```
#include <iostream>
#include <functional>
#include <vector>

/* partition implementation */
int main() {
      std::list<int> nums{15,20,25,-10,-75,100,-255,430,200};
      auto result = partition(nums.cbegin(), nums.cend(),
                                    [](int num) {
                                       return !(num % 2);
                                    });
     for (auto num : result) {
           std::cout << num << ‘ ‘;
      }
      std::cout << std::endl;
      return 0;
}
```

Mogući ispis:

```
200 20 430 -10 100 -255 -75 25 15
```

Testirati implementaciju sa sljedećim predicate funkcijama:

- `[](int num) { return num > 0; }`
  Mogući ispis: `15 20 25 200 430 100 -255 -75 -10`
- `[](int num) { return !(num % 10); }`
  Mogući ispis: `200 20 430 -10 100 -255 -75 25 15`
- `[](int num) { return std::abs(num) > 99; }`
  Mogući ispis: `200 430 -255 100 -75 -10 25 20 15`

# Zadatak 3

Modifikovati funkciju `predicate` iz prethodnog zadatka tako da tretira iznimke koje se mogu javiti iz `predicate` funkcije. Ukoliko `predicate` funkcija baci iznimku potrebno je vratiti nazad identičnu kolekciju onoj koja je proslijeđena putem iteratora. Za listu brojeva:

`15,20,25,-10,-75,100,-255,430,200`

u kombinaciji sa sljedećom predicate lambdom za poziv `partition`:

```
[](int num) { if (num > 400) throw 5; return num > 0; }
```

program treba da ispiše kolekciju identičnu originalnoj.

# Zadatak 4

Napisati funkciju koja prima std::vector<int> kao prvi argument, te vektor funkcija koje će biti primijenjene nad svakim elementom u vektoru. Ukoliko element ne zadovolji kriterije svih predicate funkcija, u tom slučaju taj element zamijeniti sa defaultnim trećim argumentom kojeg funkcija prima koji ima vrijednost -1. Ukoliko bilo koja predicate funkcija vrati false, taj element zamijeniti sa defaultnim argumentom.

Funkcija ima sljedeci potpis:

```
typedef std::vector<std::function<bool(int)>> Funkcije;
void default_if_not_all(std::vector<int>&, Funkcije, int def=-1);
```

# Zadatak 5

Napisati funkciju `filter` koja uzima listu integer-a kao prvi argument i predicate funkciju kao drugi argument. Filter treba nazad da vrati novu instancu liste integer-a koja sadrži sve elemente proslijeđene liste (prvi argument) koji zadovoljavaju datu predicate funkciju. Kako treba da izgleda tip drugog argumenta funkcije `filter`?

Primjer:

Input list: `[1, 9, 8, 4, 11, 0, 2, 6, 15, 3, 10];`
Predicate: `[](int n) { return n % 2 == 0; }`

Output: `[8, 4, 0, 2, 6, 10]`

Nakon implementacije funkcije `filter` za listu integer-a, odraditi implementaciju i za listu proizvoljnog tipa podataka te implementaciju za proizvoljan tip kontejnera.

# Zadatak 6

Napisati funkciju `map` koja uzima listu integer-a kao prvi argument te funkciju transformacije kao drugi argument. Map treba nazad da vrati novi instancu liste integer-a koja sadrži elemente proslijeđene liste transformirane kroz predicate funkciju. Kako treba da izgleda tip drugog argumenta funkcije `map`?

Primjer:

Input list: `[7, 2, -4, 5, 0, 6, 3]`
Funkcija: `[](int n) { return n * 2 + 1; }`

Output: `[15, 5, -7, 11, 1, 13, 7]`

Nakon implementacije funkcije `map` za listu integer-a, odraditi implementaciju i za listu proizvoljnog tipa podataka te implementaciju za proizvoljan tip kontejnera.

# Zadatak 7

Napisati funkciju `reduce` koja uzima listu integer-a kao prvi argument, funkciju koju će koristiti da elemente kontejnera svede na jedan element te početnu vrijednost date akumulacije. Reduce treba da elemente proslijeđenog kontejnera svede na 1 element koristeći funkciju koja je proslijeđena kao argument. Kako treba da izgleda tip drugog argumenta funkcije `reduce`?

Primjer:

Input list: `[11, 4, 5, 12, 6, 8, 9]`
Funkcija: `[](int a, int b) { return a + b + 1; }`
Start: `-12`

Output: `50`

Nakon implementacije funkcije `reduce` za listu integer-a, odraditi implementaciju i za listu proizvoljnog tipa podataka te implementaciju za proizvoljan tip kontejnera.

# Zadatak 8

Potrebno je implementirati program koji ima mogućnost unosa i evidencije studenata, predmeta i ocjena. Program treba da podržava operacije dodavanja novih predmeta, dodavanje studenata,

editovanje polja jednog studenta te ispis svih studenata zajedno sa svim ocjenama iz pojedinih predmeta i prosjekom. Sve ove operacije implementirati korisničkim menijem. Uzeti u obzir da naziv predmeta i odsjeka na kojem se predmet nalazi može imati više riječi.

Strukture Predmet i Student modelirati kao što je to ispod prikazano:

```
struct Predmet {
	std::string naziv;
	std::string odsjek;
};

struct OcjenaIzPredmeta {
	int ocjena;
	std::list<Predmet>::const_iterator predmet;
};

struct Student {
	std::string brojIndeksa;
	std::string ime;
	std::string prezime;
	std::string grad;
	std::vector<OcjenaIzPredmeta> ocjene;
};
```

Iz modela strukture `OcjenaIzPredmeta` jasno je da je za predmete potrebno koristiti kontejner tipa `std::list`. Zašto ne vektor? Šta se sa iteratorima vektora dešava kada se vektor kontejner proširuje?

Unos novih predmeta treba da bude vrlo jednostavan, od korisnika se traži unos naziva predmeta i odsjeka. Novi predmet je potrebno dodati na kraj kontejnera tipa `std::list<Predmet>` osim ako predmet sa unešenim nazivom već ne postoji. U tom slučaju ispisati grešku korisniku. te se vratiti na početni meni.

Unos studenta treba da ide u redoslijedu:

1. Indeks (samo jedna riječ). Ukoliko student sa unesenim brojem indeksa već postoji, ispisati grešku na ekran te se vratiti nazad na početni meni.
2. Ime (može sadržavati više riječi)
3. Prezime (može sadržavati više riječi)
4. Grad (može sadržavati više riječi)
5. Unos ocjena
    a. Unos predmeta za koju je ocjena vezana. Ukoliko predmet ne postoji ispisati grešku na ekran, vratiti se na početni meni i ne dodavati ništa u kolekciju studenata. U suprotnom kreirati instancu strukture Ocjena iz predmeta čiji iterator pokazuje na konkretan predmet iz kolekcije.
    b. Unos ocjene (moguće vrijednosti od 5 do 10). Ukoliko korisnik unese vrijednost manju od 5 ili veću od 10 zatražiti ponovni unos ocjene.

Opcija editovanja studenta treba od korisnika da traži unos broja indeksa. Ukoliko student sa tim brojem indeksa ne postoji, ispisati grešku i vratiti se na početni meni, u suprotnom potrebno je ponuditi novi meni gdje korisnik može odabrati kakvu operaciju editovanja želi odraditi. Moguće operacije su:

1. Mijenjanje broja indeksa studenta
2. Mijenjanje imena studenta
3. Mijenjanje prezimena studenta
4. Mijenjanje grada studenta
5. Brisanje svih ocjena
6. Dodavanje novih ocjena

Struktura Student i sve potencijalne funkcije vezane za nju je potrebno izdvojiti u poseban hpp file. Svu implementaciju ovih funkcija vezanih za studente je potrebno prebaciti u poseban cpp file.

# Zadatak 9

Napisati generičku funkciju `jednako` koja poredi sekvence `[od1, do1)` i `[od2, do2)`, pri čemu se pretpostavlja da sekvence imaju isti broj elemenata. Funkcija vraća vrijednost `true` ukoliko su svi elementi obje sekvence isti, u suprotnom vraća netačnu vrijednost. Funkcija uzima tri parametra - prva dva označavaju početak i kraj prve sekvence, a treći parametar označava početak druge sekvence.
Napomena: iteratori koji operiraju na sekvencama mogu biti različitog tipa.

Koristeći funkciju `jednako`, napisati program koji učitava riječi koje korisnik unosi sa tastature i ispisuje svaku riječ koju identificira kao palindrom. Palindrom je riječ koja ima isto značenje bez obzira da li se čita sa lijeva na desno ili obrnuto. Primjer pozivanja programa se nalazi na linku.

# Zadatak 10

Implementirati generičku funkciju `sort(p,k,f)` koja kao argumente uzima početak i kraj sekvence koju treba sortirati `[p,k)` te binarnu predikat funkciju `f(a,b)` koja definiše kriterij za sortiranje. Binarna predikat funkcija vraća tačnu vrijednost ukoliko element `a` treba u sekvenci biti prije elementa `b`, u suprotnom vraća netačno. Ukoliko se funkcija `sort` pozove bez trećeg argumenta (pozove se `sort(p,k)`), elemente sortirati od manjeg ka većem. Funkciju `sort` definisati u namespace-u `my`, u zaglavlju *my.hpp*.

Pseudo-kod algoritma za sortiranje kojeg implementira funkcija `sort`, skupa sa primjerima koji uključuju animaciju može se naći na adresi: https://visualgo.net/en/sorting.

Funkciju testirati pomoću file-a *test.cpp* datog u prilogu zadaće. Obratiti pažnju na činjenicu da file-ovi *test.cpp*, *my.hpp* i *doctest.h* (nalazi se u prilogu zadaće) moraju biti u istom direktoriju. Na linku se nalazi poziv izvršnog file-a sa ispravno napisanom funkcijom `sort`.

# Zadatak 11

Implementirati klase Radnik i BazaRadnika koja implementira bazu radnika i omogućava:

- Unos pojedinačnog radnika u bazu. Baza ne dozvoljava unos dva radnika istog imena. Ukoliko se pokuša unijeti isti radnik dva puta, metod generira iznimku tipa `domain_error`.
- Brisanje radnika određenog imena iz baze. Metod za brisanje kao rezultat vraća uspješnost operacije.
- Unos i ispis kompletne baze pomoću objekata tipa `istream` i `ostream`.
- Računanje prosječne plate i starosti radnika.

Klasu testirati sljedećim glavnim programom:

```
#include “Radnik.hpp”
#include “Baza.hpp”
#include <iostream>

int main() {
  BazaRadnika b1;
  b1.ucitaj(std::cin);
  if(b1.izbrisi_radnika(“Alen”)) {
    std::cout << “Alen izbrisan” << std::endl;
  } else {
    std::cout << “Brisanje nije moguce!” << std::endl
  }
  std::cout << “U bazi su: “ << std::endl;
  b1.ispisi(std::cout);
  std::cout<<“Prosjek plata je: “<<b1.prosjek_plata()<<std::endl;
  std::cout<<“Prosjek godina je: “<<b1.prosjek_godina()<<std::endl;
  return 0;
}
```

Na linku je prikazan poziv izvršnog file-a sa navedenom `main` funkcijom.