# Zadatak 1

Koristeći rješenje zadnjeg zadatka od prošle sedmice prepravite strukturu **Kolekcija** tako da ima sljedeće funkcije

|  |
| --- |
| typedef float Tip;  const int max\_size = 30;  struct Kolekcija  {  shared\_ptr<Tip> podaci[max\_size];  int brojac = 0;  void Dodaj(Tip newV);  void Dodaj(Kolekcija k);  Kolekcija Filter(function<bool(Tip&)> f);  void ForEach(function<void(Tip&)> f);  }; |

**Code za zadatak (rješenje V08 – Zadatak 4)**

|  |
| --- |
| #include <iostream>  #include <functional>  #include <algorithm>  #include<memory>  using namespace std;  typedef float Tip;  const int max\_size = 30;  struct Kolekcija  {  shared\_ptr<Tip> podaci[max\_size];  int brojac = 0;  };  void kDodaj(Kolekcija& k, Tip newV)  {  //dodaj newV u podaci na poziciju brojac  //uvecati brojac  k.podaci[k.brojac++] = make\_shared<Tip>(newV);  }  void kGenerisiPodatke(Kolekcija& k)  {  for (int i = 0; i < max\_size / 2; i++)  {  int newV = rand() - rand();  //newV dodati u kolekciju k  kDodaj(k, (Tip) newV);  }  }  void kPrinter(Kolekcija& k, char\* opis)  {  cout << opis << endl;  for (int i = 0; i < k.brojac; i++)  {  shared\_ptr<Tip>& p = k.podaci[i];  cout << i << ": " << \*p << endl;  }  }  Kolekcija kFilter(Kolekcija& oldK, function<bool(Tip&)> f)  {  Kolekcija newK;  //iz kolekcije k u kolekciju newK kopirati elemente koji zadovoljavaju uslov f  for (size\_t i = 0; i < oldK.brojac; i++)  {  shared\_ptr<Tip> &x = oldK.podaci[i];  if(f(\*x)==true)  kDodaj(newK, \*x);  }  return newK;  }  void kForEach(Kolekcija& k, function<void(Tip&)> f)  {  //za svaki element iz kolekcije k izvrsi zadatak f  for (int i = 0; i < k.brojac; i++)  {  shared\_ptr<Tip> &x = k.podaci[i];  f(\*x);  }  }  void main()  {  //a. Instanciraj Kolekciju k1 i generiši podatke  Kolekcija k1;  kGenerisiPodatke(k1);  //b. Iz kolekcije k1 filtiraj parne brojeve u kolekciju k2  Kolekcija k2 = kFilter(k1, [](Tip &b) { return b % 2 == 0; });  //c. Iz kolekcije k2 filtiraj pozitivne brojeve u kolekciju k3  Kolekcija k3 = kFilter(k2, [](Tip &b) { return b>0; });  //d. Ispisati na ekran sve tri kolekcije  kPrinter(k1, "kolekcija 1 ");  kPrinter(k2, "kolekcija 2 ");  kPrinter(k3, "kolekcija 3 ");  //f. koristeci kForEach uvecati sve elemente kolekcije k3  kForEach(k3, [](Tip &b) {b++; });  //g. koristeci kForEach kopirati sve elemente kolekcije k2 u novu kolekciju  Kolekcija k4;  kForEach(k2, [&k4](Tip &v) {kDodaj(k4, v); });  //h. koristeci kForEach izračunati sumu elemenata k4  int s = 0;  kForEach(k4, [&s](Tip &v) { s += v; });  //i. koristeci kForEach zamjeniti negativne elemente sa 0  kForEach(k4, [&s](Tip &v) { if(v<0) v=0; });  //j. koristeci kForEach sve elemente k4 zamjeniti sa vrijednostima njihovog korijena  kForEach(k4, [&s](Tip &v) { v = sqrt(v); });  } |

|  |
| --- |
| Rješenje zadataka br1 |
|  |

# Zadatak 2

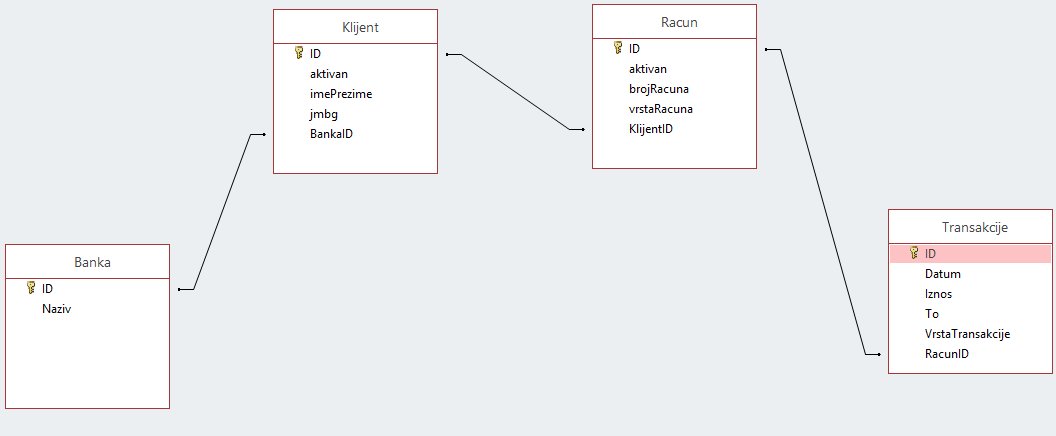
*Implementirajte konzolnu aplikaciju sa sljedećim funkcionalnostima*

|  |
| --- |
| 1. Dodaj korisnika 2. Dodaj racun 3. Dodaj uplatu 4. Dodaj isplatu 5. Stanje na racunu 6. Pretraga po min. iznosu transakcije 7. Pretraga po datumu transakcije 8. Deaktiviranje korisnika 9. Kraj |

*Implementirajte sljedeće strukture (u odvojene header fajlove)*

|  |
| --- |
| 1. struct Datum    1. int \_dan;    2. int \_mjesec;    3. int \_godina;    4. static Datum Napravi(int d, int m, int g) ;    5. void Ispis();    6. bool jednako(Datum& drugi) 2. enum VrstaTransakcije    1. UPLATA    2. ISPLATA 3. struct Transakcija    1. Datum \_datumTransakcije;    2. char \_TO[20];    3. float \_iznos;    4. VrstaTransakcije \_vrstaTransakcije;    5. bool \_aktivna;    6. static Transakcija NapraviUplatu(float iznos, Datum datum) ;    7. static Transakcija NapraviIsplatu(float iznos, char\* to, Datum datum) ;    8. void Ispis(); 4. struct Kolekcija   *(typedef Transakcija Tip; preuzeti rješenje zadataka br1)*   1. enum VrstaRacuna    1. TEKUCI    2. DEVIZNI    3. STEDNI 2. struct Racun    1. VrstaRacuna \_vrstaRacuna;    2. char \_brojRacuna[20];    3. Kolekcija transakcije;    4. bool \_aktivan;    5. static Racun\* Napravi(VrstaRacuna vrsta\_racuna, char\* broj\_racuna) ;    6. void Ispis();    7. void Uplata(float iznos, Datum datum) ;    8. void Isplata(float iznos, char\* to, Datum datum) ;    9. float Saldo() //stanje na računu: suma uplata minus suma isplata    10. Kolekcija GetTransakcijeByIznos(float iznos) ;    11. Kolekcija GetTransakcijeByDatum(Datum& datum) ; 3. struct Klijent    1. int maxR = 10;    2. char \_JMBG[14];    3. char \* \_imePrezime;    4. Racun \*\* \_racuni = new Racun\*[maxR];    5. int \_trenutnoRacuna;    6. bool \_aktivan;    7. static Klijent\* Napravi(char \* jmbg, char \* ime\_prezime) ;    8. void Ispis();    9. void Dealociraj();    10. void DodajRacun(VrstaRacuna vrstaRacuna, char\* brojRacuna) ;    11. void DodajUplatu(int indexRacuna, float iznos, Datum datum) ;    12. void DodajIsplatu(int indexRacuna, float iznos, char\* to, Datum datum) ;    13. Kolekcija GetTransakcijeByIznos(float iznos) ;// vraca Kolekciju transakcija koje su imale iznos veci od trazenog    14. Kolekcija GetTransakcijeByDatum(Datum datum) ;// vraca transakcija koje su obavljene za trazeni datum    15. void Deaktiviraj(); 4. struct Banka    1. int maxK = 100;    2. int trenutnoKlijenata = 0;    3. Klijent \*\* klijenti = new Klijent\*[maxK];    4. void DodajKlijenta(char\* jmbg, char\* ime\_prezime) ;    5. void Dealociraj(); |

Radi lakšeg razumjevanja uloge struktura, pogledajte Access bazu podataka.

**

*U nastavku su date strukture bez implementiranih funkcija*

|  |
| --- |
| Datum.h |
| #pragma once  #include <iostream>  using namespace std;  struct Datum  {  int \_dan, \_mjesec, \_godina;    static Datum Napravi(int d, int m, int g)  {  }  bool jednako(Datum& drugi)  {  }  void Ispis()  {  }  }; |

|  |
| --- |
| Transakcija.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Datum.h"  enum VrstaTransakcije { UPLATA, ISPLATA };  char \*vrstaTransakcije[] = { "UPLATA", "ISPLATA" };  struct Transakcija  {  Datum \_datumTransakcije;  char \_TO[20];  float \_iznos;  VrstaTransakcije \_vrstaTransakcije;  /\* ako je isplata vrijednost \_TO postaviti na brojRacuna sa koga se transakcija vrsi\*/  bool \_aktivna;  /\*u slucaju da se transakcija izbrise ona postaje neaktivna, ali je bitno da o njoj ostane zapis\*/  void Ispis()  {  }  static Transakcija NapraviUplatu(float iznos, const Datum& datum)  {  }  static Transakcija NapraviIsplatu(float iznos, char\* to, Datum& datum)  {  }  }; |

|  |
| --- |
| Kolekcija.h |
| #pragma once  #include <iostream>  #include <functional>  #include <memory>  using namespace std;  #include "Transakcija.h"  typedef Transakcija Tip;  const int max\_size = 30;  struct Kolekcija  {  shared\_ptr<Tip> podaci[max\_size];  int brojac=0;  void Dodaj(Tip newV)  {  podaci[brojac++] = make\_shared<Tip>(newV);  }  void Dodaj(Kolekcija k)  {  k.ForEach([&](Tip& x) {Dodaj(x); });  }  Kolekcija Filter(function<bool(Tip&)> f)  {  Kolekcija newK;  //iz kolekcije k u kolekciju newK kopirati elemente koji zadovoljavaju uslov f  for (size\_t i = 0; i < brojac; i++)  {  shared\_ptr<Tip> &x = podaci[i];  if (f(\*x) == true)  newK.Dodaj(\*x);  }  return newK;  }  void ForEach(function<void(Tip&)> f)  {  //za svaki element iz kolekcije k izvrsi zadatak f  for (int i = 0; i < brojac; i++)  {  shared\_ptr<Tip> &x = podaci[i];  f(\*x);  }  }  }; |

|  |
| --- |
| Račun.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Kolekcija.h"  enum VrstaRacuna { TEKUCI, DEVIZNI, STEDNI };  char \*vrstaRacuna[] = { "TEKUCI", "DEVIZNI", "STEDNI" };  struct Racun  {  VrstaRacuna \_vrstaRacuna;  char \_brojRacuna[20];  bool \_aktivan;  Kolekcija transakcije;    void Ispis()  {  }  void Dealociraj()  {  }  void Deaktiviraj()  {  }  float Saldo()  {  }  void Uplata(float iznos, Datum datum)  {  }  void Isplata(float iznos, char \*to, Datum datum)  {  }  Kolekcija GetTransakcijeByIznos(float iznos)  {  }  Kolekcija GetTransakcijeByDatum(Datum& datum)  {  }  static Racun\* Napravi(VrstaRacuna vrsta\_racuna, char\* broj\_racuna)  {  }  }; |

|  |
| --- |
| Klijent.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Racun.h"  struct Klijent  {  char \_JMBG[14];  char \* \_imePrezime;  int maxR = 10;  Racun \*\* \_racuni = new Racun\*[maxR];  int \_trenutnoRacuna = 0; //brojac  bool \_aktivan;  static Klijent\* Napravi(char \* jmbg, char \* ime\_prezime)  {  }  void Ispis()  {  }  void Dealociraj()  {  }  void DodajRacun(VrstaRacuna vrstaRacuna, char\* brojRacuna)  {  }  void DodajUplatu(int indexRacuna, float iznos, Datum datum)  {  }  void DodajIsplatu(int indexRacuna, float iznos, char\* to, Datum datum)  {  }  Kolekcija GetTransakcijeByIznos(float iznos)  {  }  Kolekcija GetTransakcijeByDatum(Datum datum)  {  }  void Deaktiviraj()  {  }  }; |

|  |
| --- |
| Banka.h |
| #pragma once  #include "Klijent.h"  struct Banka  {  int maxK = 100;  int trenutnoKlijenata= 0;  Klijent \*\* klijenti = new Klijent\*[maxK];  void DodajKlijenta(char\* jmbg, char\* ime\_prezime)  {  }  void Dealociraj()  {  }  }; |

|  |
| --- |
| Source.cpp |
| #include <iostream>  #include "Banka.h"  using namespace std;  char \* crt = "\n---------------------------\n";  int PrikaziMenu()  {  int izbor;  cout << crt << "\t::MENU::" << crt;  cout << "1. Dodaj korisnika" << endl;  cout << "2. Dodaj racun" << endl;  cout << "3. Dodaj Uplatu" << endl;  cout << "4. Dodaj Isplatu" << endl;  cout << "5. Stanje na računu" << endl;  cout << "6. Pretraga po min. iznosu transakcije" << endl;  cout << "7. Pretraga po datumu transakcije" << endl;  cout << "8. Deaktiviranje korisnika" << crt; // racuna ili korisnika  cout << "9. Kraj" << crt;  do  {  cout << "Vas izbor -> ";  cin >> izbor;  } while (izbor < 1 || izbor>9);  system("cls");  return izbor;  }  Klijent\* IzborKlijenta(Banka &B)  {  }  Racun\* IzborRacuna(Klijent &K)  {  }  void main()  {  Banka B1;  int izborMenu;  do  {  izborMenu = PrikaziMenu();  if (izborMenu == 1)  {  //Dodaj korisnika  cin.ignore();  cout << "Unesite jmbg"<<endl;  char jmbg[14];  cin.getline(jmbg, 14);  cout << "Unesite ime i prezime" << endl;  char ime\_prezime[30];  cin.getline(ime\_prezime, 30);  B1.DodajKlijenta(jmbg, ime\_prezime);  }  if (izborMenu == 2)  {  //Dodaj racun  Klijent\* k = IzborKlijenta(B1);  cin.ignore();  cout << "Unesite broj racuna" << endl;  char racun[13];  cin.getline(racun, 13);  k->DodajRacun(DEVIZNI, racun);  }  if (izborMenu == 3)  {  //Uplata  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Unesite iznos uplate " << endl;  float iznos;  cin >> iznos;  cout << "Unesite datum uplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  r->Uplata(iznos, date);  }  if (izborMenu == 4)  {  //Isplata  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Unesite iznos isplate " << endl;  float iznos;  cin >> iznos;  cout << "Unesite datum isplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  r->Isplata(iznos, "??", date);  }  if (izborMenu == 5)  {  //Stanje  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Stanje na racunu je " << r->Saldo();  }  if (izborMenu == 6)  {  //Pretraga po min. iznosu  Klijent\* k = IzborKlijenta(B1);  cout << "Pretraga: Unesite min. iznos transakcije" << endl;  float f;  cin >> f;  Kolekcija transakcije = k->GetTransakcijeByIznos(f);  transakcije.ForEach([](Transakcija& t) { t.Ispis(); });  }  if (izborMenu == 7)  {  //Pretraga po datumu  Klijent\* k = IzborKlijenta(B1);  cout << "Pretraga: Unesite datum transakcije" << endl;  cout << "Unesite datum isplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  Kolekcija transakcije = k->GetTransakcijeByDatum(date);  transakcije.ForEach([](Transakcija& t) { t.Ispis(); });  }  if(izborMenu == 8)  {  cout << "Deaktiviranje klijenta: Izaberite klijenta" << endl;  Klijent\* k = IzborKlijenta(B1);  k->Deaktiviraj();  }  } while (izborMenu != 9);  B1.Dealociraj();  } |