# Pomoćne funkcije: Objekti i fajlovi

Za rad sa fajlovima možemo definisati pomoćne funkcije koje će pojednostaviti postupak snimanja i čitanja iz fajla. Npr.

|  |
| --- |
| **Fajlovi.h – Pomoćne funkcije (verzija 1)** |
| #pragma once  #include <fstream>  void f\_snimi(ofstream& f, int &tip)  {  int v = sizeof(int);  f.write((char\*)&tip, v);  }  void f\_snimi(ofstream& f, float &tip)  {  int v = sizeof(float);  f.write((char\*)&tip, v);  }  void f\_snimi(ofstream& f, Datum &tip)  {  int v = sizeof(Datum);  f.write((char\*)&tip, v);  }  //itd. |

Pomoću generičkog mehanizma može se broj funkcija za čitanje fajla smanjiti na jedan (isto tako za vrijedni za funkciju snimi):

|  |  |
| --- | --- |
| **Fajlovi.h – Pomoćne funkcije (verzija 2)** | |
| template<typename Tip>  void f\_snimi(ofstream& f, Tip &tip)  {  int v = sizeof(Tip);  f.write((char\*)&tip, v);  } | template<typename Tip>  void f\_ucitaj(ifstream& f, Tip &tip)  {  int v = sizeof(Tip);  f.read((char\*)&tip, v);  } |

|  |  |
| --- | --- |
| **Primjer korištenja pomoćnih funkcija (verzija 2)** | |
| int a = 6;  float b = 2.4;  int c[3] = { 4,5,6 };  Datum d = Datum::Napravi(1, 11, 2015);  ofstream f("aa.txt");  f\_snimi(f, a);  f\_snimi(f, b);  f\_snimi(f, c);  f\_snimi(f, d);  f.close();  cout << "int a = " << a << endl;  cout << "float b = " << b << endl;  cout << "int c[3] = " << c[0] << c[1] << c[2];  d.Ispis(); | int a ;  float b ;  int c[3] ;  Datum d ;  ifstream f("aa.txt");  f\_ucitaj(f, a);  f\_ucitaj(f, b);  f\_ucitaj(f, c);  f\_ucitaj(f, d);  f.close();  cout << "int a = " << a << endl;  cout << "float b = " << b << endl;  cout << "int c[3] = " << c[0] << c[1] << c[2];  d.Ispis(); |

# Pomoćne funkcije: Pokazivači i fajlovi

Ukoliko se pozove generička funkcija **f\_snimi (verzija 2)** za parametar pokazivač, generički parametar **Tip** će imati vrijednost **„Tip = Datum\*“** a ne **„ Tip = Datum“.** Zatim će se unutar funkcije pozvati **sizeof(Tip),** tj. **sizeof(Datum\*).** Veličina pokazivača je uvijek ista jer svaki pokazivač zauzima istu memorijsku veličinu.

Znači, **Sizeof** za pokazivač **ne** računa veličinu za Tip na koji pokazuje (tj. Datum) već veličinu pokazivača. Stoga se na ovakav način ne može odrediti veličinu objekta tipa **Datum**.

|  |
| --- |
| **Primjer poziva za verziju 2.** |
| Datum\* p = new Datum;  ofstream f("aa.txt");  f\_snimi<Datum\*>(f, p); //sa eksplicitnim navođenjem generičkog parametra  f\_snimi(f, p); //  ... |

Da bi ispravno odredili veličinu za generički tip podatka (npr. Datum) potrebno je prilagoditi funkciju tako da je moguće napisati sljedeći izraz:

|  |
| --- |
| **Primjer poziva za verziju 3.** |
| Datum\* p = new Datum;  ofstream f("aa.txt");  f\_snimi<Datum>(f, p); //sa eksplicitnim navođenjem generičkog parametra  ... |

U ovom slučaju, gdje je „Tip =Datum“, operator **sizeof(Tip)** odnosno **sizeof(Datum)** će vrati ispravnu vrijednost.

Da bi riješili prethodno, potrebno je dodati novu generičku funkcije sa ulaznim parametrom koji je pokazivač na Tip.

|  |
| --- |
| template<typename Tip>  void f\_snimi(ofstream& f, Tip &tip)  {  ...  }  template<typename Tip>  void f\_snimi(ofstream& f, Tip\* &tip)  {  ...  } |

Također, može se uvesti parametar funkcije za veličinu niza (čija je defaultna vrijednost 1). Implementacija obje funkcije je data u nastavku.

|  |  |
| --- | --- |
| **Fajlovi.h – Pomoćne funkcije (verzija 3)** | |
| template<typename Tip>  void f\_snimi(ofstream& f, Tip &tip)  {  //ako je tip vrijednost  int v = sizeof(Tip);  f.write((char\*)&tip, v);  }  template<typename Tip>  void f\_snimi(ofstream& f, Tip\* &tip, int broj=1)  {  //ako je tip adresa  int v = sizeof(Tip) \* broj;  f.write((char\*)tip, v);  } | template<typename Tip>  void f\_ucitaj(ifstream& f, Tip &tip)  {  //ako je tip vrijednost  int v = sizeof(Tip);  f.read((char\*)&tip, v);  }  template<typename Tip>  void f\_ucitaj(ifstream& f, Tip\* &tip, int broj=1)  {  //ako je tip adresa  int v = sizeof(Tip) \* broj;  f.read((char\*)tip, v);  } |

Ukoliko se prilikom poziva funkcije izostavi parametar „broj“, njegova defaultna vrijednost će biti „1“.

|  |  |
| --- | --- |
| **Souce.cpp – Primjer korištenja pomoćnih funkcija (verzija 3)** | |
| Datum\* p = new Datum;  ofstream f("aa.txt");  f\_snimi(f, p);  f\_snimi<Datum>(f, p); //ovo je sada dozvoljeno jer je definisana funkcija Tip\* &tip,  ... | |
| int\* p = new int[3];  p[0] = 4;  p[1] = 5;  p[2] = 6;  f\_snimi(f, p, 3);  f\_snimi<int>(f, p, 3); //ili eksplicitno  cout << "new int[3] = " << p[0] << p[1] << p[2]; | int\* p = new int[3];  f\_ucitaj(f, p, 3);  f\_ucitaj<int>(f, p, 3); //ili eksplicitno  cout << "new int[3] = " << p[0] << p[1] << p[2]; |

Alokacija dinamičke memorije može se premjestiti i u pomoćne funkcije. Implementacija je prikazana u verziji 4.

|  |  |
| --- | --- |
| **Fajlovi.h – Pomoćne funkcije (verzija 4)** | |
| template<typename Tip>  void f\_snimi(ofstream& f, Tip &tip)  {  //ako je tip vrijednost  int v = sizeof(Tip);  f.write((char\*)&tip, v);  }  template<typename Tip>  void f\_snimi(ofstream& f, Tip\* &tip, int broj=1)  {  //ako je tip adresa  int v = sizeof(Tip) \* broj;  f.write((char\*)tip, v);  } | template<typename Tip>  void f\_ucitaj(ifstream& f, Tip &tip)  {  //ako je tip vrijednost  int v = sizeof(Tip);  f.read((char\*)&tip, v);  }  template<typename Tip>  void f\_ucitaj(ifstream& f, Tip\* &tip, int broj=1)  {  //ako je tip adresa  tip = new Tip[broj];  int v = sizeof(Tip) \* broj;  f.read((char\*)tip, v);  } |

|  |  |
| --- | --- |
| **Souce.cpp – Primjer korištenja pomoćnih funkcija (verzija 4)** | |
| int\* p = new int[3];  p[0] = 4;  p[1] = 5;  p[2] = 6;  f\_snimi(f, p, 3);  f\_snimi<int>(f, p, 3); //ili eksplicitno  cout << "new int[3] = " << p[0] << p[1] << p[2]; | int\* p = new int[3];  f\_ucitaj(f, p, 3);  f\_ucitaj<int>(f, p, 3); //ili eksplicitno  cout << "new int[3] = " << p[0] << p[1] << p[2]; |

Pomoćne funkcije: Dinamički niz karaktera i fajlovi

Ukoliko je potrebno zapisati dinamički niz karaktera u fajl ostaje problem kako odrediti njegovu dužinu prilikom čitanja. Najjedostavna implementacija je da se ispred niza karaktera zapiše integer vrijednost koja predstavlja dužinu stringa. Implementacija je prikazana u nastavku.

|  |  |
| --- | --- |
| **Fajlovi.h – Pomoćne funkcije za dinamički niz karaktera** | |
| void f\_snimi\_str(ofstream& f, char\* &str)  {  //za dinamički niz karaktera  int v = strlen(str) + 1;  f\_snimi(f, v);  f\_snimi(f, str, v);  } | void f\_ucitaj\_str(ifstream& f, char\* &str)  {  //za dinamički niz karaktera  int v;  f\_ucitaj(f, v);  str = new char[v];  f\_ucitaj(f, str, v);  } |
| char\* p = new char[100];  strcpy\_s(p, 100, "Ovo je string");  ofstream f("aa.txt");  f\_snimi\_str(f, p); | char\* p;  ifstream f("aa.txt");  f\_ucitaj\_str(f, p); |

# Zadatak 1

Definisati strukturu **Kolekcija** u istoimeni header fajl.

|  |
| --- |
| typedef int Tip;  struct Kolekcija  {  int max\_size = 30;  Tip\* podaci=new Tip[max\_size];  int brojac = 0;  void Prosiri()  {  max\_size \*= 2;  Tip\* podaciNew = new Tip[max\_size];  for (int i = 0; i < brojac; i++)  {  podaciNew[i] = podaci[i];  }  delete[]podaci;  podaci = podaciNew;  }  Tip& Get(int i)  {  return podaci[i];  }  void Dodaj(Tip newV)  {  if (brojac == max\_size)  Prosiri();  podaci[brojac++] = newV;  }  void ForEach(function<void(Tip&)> p)  {  for (int i = 0; i<brojac; i++)  {  Tip& t = podaci[i];  p(t);  }  }  Kolekcija Filter(function<bool(Tip&)> f)  {  Kolekcija newK;  for (int i = 0; i < brojac; i++)  {  Tip &x = podaci[i];  if (f(x))  {  kDodaj(newK, x);  }  }  return newK;  }  }; |

Definisati strukturu **Datum** u istoimeni header fajl.

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

U glavnom programu definisati sljedeće

1. Instancirajte Kolekcija k1. Dodajte 5 elemenata u k1.

Instancirajte kolekcija k2 u dinamičkoj memoriji. Dodajte 5 elemenata u k2

1. Prepravite **int Tip** u **Datum Tip**. Umjesto brojeva dodajte datuma u k1 i k2.
2. Prepravite Datum Tip u **Datum\* Tip**.

|  |
| --- |
| **Rješenje c** |
| typedef Datum\* Tip;  struct Kolekcija  {  ...  } |
| void main()  {  Kolekcija k1;  k1.Dodaj(&Napravi(5, 3, 2017));  k1.Dodaj(&Napravi(5, 3, 2017));  k1.Dodaj(&Napravi(5, 3, 2017));  k1.Dodaj(&Napravi(5, 3, 2017));  k1.Dodaj(&Napravi(5, 3, 2017));  k1.ForEach([](Datum\* p) { p->Ispis(); });  cout << endl;  Kolekcija\* k2 = new Kolekcija;  k2->Dodaj(&Napravi(5, 3, 2017));  k2->Dodaj(&Napravi(5, 3, 2017));  k2->Dodaj(&Napravi(5, 3, 2017));  k2->Dodaj(&Napravi(5, 3, 2017));  k2->Dodaj(&Napravi(5, 3, 2017));  k2->ForEach([](Datum\* p) { p->Ispis(); });  cout << endl;  delete k2;  } |

# Zadatak 2 – Generičke strukture

U istom programu definišite

1. Kolekciju k1 za integer
2. Kolekciju k2 za float
3. Kolekciju k3 za Datum\*
4. Kolekciju k4 za Datum

Npr.

|  |
| --- |
| Kolekcija<int> k1  Kolekcija<float k2  Kolekcija<Datum> k3 |

**Rješenje za glavni program**

|  |
| --- |
| void main()  {  Kolekcija<int>k1; //  k1.Dodaj(25);  k1.Dodaj(30);  Kolekcija<float>k2;  k2.Dodaj(33.5);  k2.Dodaj(22.5);  Kolekcija<Datum\*>k3;  k3.Dodaj(Datum::Napravi(25, 6, 2017));  k3.Dodaj(Datum::Napravi(25, 6, 2017));  k3.Dodaj(Datum::Napravi(25, 6, 2017));  k3.Dodaj(Datum::Napravi(25, 6, 2017));  k3.Dodaj(Datum::Napravi(4, 5, 2013));  Kolekcija<Datum>k4;  k4.Dodaj(\*Datum::Napravi(22, 4, 2017));  k4.Dodaj(\*Datum::Napravi(11, 4, 2017));  } |

**Rješenje za strukturu Kolekcija**

umjesto

|  |
| --- |
| typedef int Tip;  struct Kolekcija  {... |

prepraviti u

|  |
| --- |
| template<typename Tip>  struct Kolekcija  {... |

# Pomoćne funkcije: Kolekcija i fajlovi

Slijedi primjer implementacije snimanja kolekcije pod pretpostavkom da Tip podataka koji je korišten kao generički parametar poseduje funkcije Snimi(ofstream& f) i Ucitaj(ifstream& f);

|  |  |
| --- | --- |
| **Fajlovi.h – Pomoćne funkcije za strukturu Kolekcija** | |
| template<typename T>  void f\_snimi\_k(ofstream& f, Kolekcija<T> &k)  {  //za kolekciju koja čuva vrijednosti  f\_snimi(f, k.brojac);  k.ForEach([&f](T& t) {  t.Snimi(f);  });  }  template<typename T>  void f\_snimi\_k(ofstream& f, Kolekcija<T\*> &k)  {  //za kolekciju koja čuva adrese  f\_snimi(f, k.brojac);  k.ForEach([&f](T\*& t) {  t->Snimi(f);  });  } | template<typename T>  void f\_ucitaj\_k(ifstream& f, Kolekcija<T> &k)  {  //za kolekciju koja čuva vrijednosti  f\_ucitaj(f, k.brojac);  k.ForEach([&f](T& t) {  t.Ucitaj(f);  });  }  template<typename T>  void f\_ucitaj\_k(ifstream& f, Kolekcija<T\*> &k)  {  //za kolekciju koja čuva adrese  f\_ucitaj(f, k.brojac);  k.ForEach([&f](T\*& t) {  t = new T;  t->Ucitaj(f);  });  } |

# Pomoćne funkcije: Korisnički odabiri elemenata kolekcije

Često korisnik mora u konzilni programima odabrati jedan od elemenata. Ovaj postupak se može ubrazi pomoću *helper* funkcije. Implementira je generička pomoćna funkcija koja kao izvor podataka koristi generičku strukturu Kolekcija. Funkcija radi uz pretpostavku da Tip posjeduje funkciju **Ispis**.

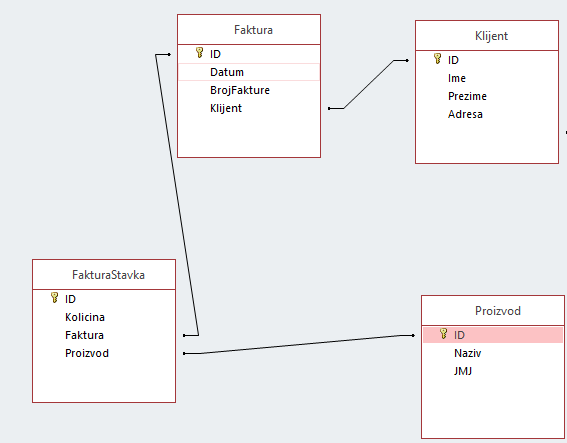
|  |
| --- |
| template <typename Tip>  Tip\* odaberi(Kolekcija<Tip\*> &kolekcija)  {  int unos;  do  {  int rb = 1;  kolekcija.ForEach([&](Tip\* g) {  cout << rb++ << ". ";  g->Ispis();  });  cout << "0. Cancel" << endl;  cout << "Unesite RB: " << endl;  cin >> unos;  if (unos == 0)  return nullptr;  if (unos >= 1 && unos <= kolekcija.brojac)  return kolekcija.podaci[unos - 1];  cout << " !!Pogrešan unos" << endl;  } while (true);  } |

Primjer korištenja generičke funkcije

|  |
| --- |
| Kolekcija<Proizvod\*> Proizvodi;  Kolekcija<Faktura\*> Fakture;  Kolekcija<Klijent\*> Klijenti;  ...  Proizvod \* p = odaberi(Proizvodi);  Faktura \* f = odaberi(Klijenti);  Klijent\* k = odaberi(Fakture); |

# Zadatak 3

Definišite strukture kao u bazi podataka u prilogu **StoreApp.accdb**.



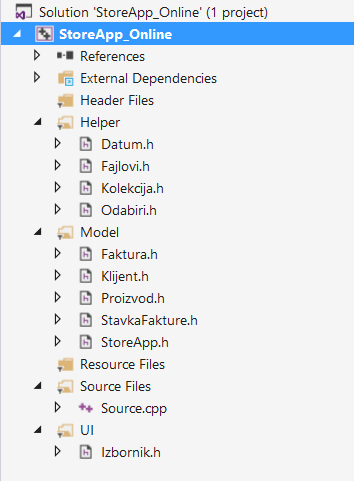
Dodajte strukturu **StoreApp**

|  |
| --- |
| struct StoreApp  {  int \_ID;  char \_Naziv[20];  Kolekcija<Proizvod\*> Proizvodi;  Kolekcija<Faktura\*> Fakture;  Kolekcija<Klijent\*> Klijenti;  void Snimi(ofstream& f)  {  cout << "read: STORE APP " << endl;  f\_snimi(f, \_ID);  f\_snimi(f, \_Naziv);  f\_snimi\_k(f, Proizvodi);  f\_snimi\_k(f, Klijenti);  f\_snimi\_k(f, Fakture);  }  void Ucitaj(ifstream& f)  {  cout << "save: STORE APP " << endl;  f\_ucitaj(f, \_ID);  f\_ucitaj(f, \_Naziv);  f\_ucitaj\_k(f, Proizvodi);  f\_ucitaj\_k(f, Klijenti);  f\_ucitaj\_k(f, Fakture);  }  static StoreApp \*Napravi(int Id, char\* nazivStore)  {  StoreApp \*y = new StoreApp;  y->\_ID = Id;  strcpy\_s(y->\_Naziv, nazivStore);  return y;  }  }; |

**Izbornik**

|  |
| --- |
| 1. OPEN STORE FROM FILE  2. NEW STORE  3. SAVE CHANGES TO FILE []  4. DODAJ: GRAD  5. DODAJ: KLIJENT  6. DODAJ: PROIZVOD  7. DODAJ: PRODAJA (FAKTURA)  8. TEST APP  0. EXIT |

Rješenje se nalazi u Visual Studio projektu u prilogu.



# Zadatak 4

Šta je rezultata sljedećeg programa

|  |
| --- |
| shared\_ptr<Datum> d1 = make\_shared<Datum>(Datum::Napravi(1, 1, 2017));  shared\_ptr<Datum> d2 = d1;  d1->\_dan++;  cout << d2->\_dan << endl; |

Odgovor:

|  |
| --- |
| d1 i d2 su pametni pokazivači koji pokazuju na isti objekat tipa Datum.  Objektu tipa datum može se pristupiti preko d1 i d2. |

# Zadatak 5

Prepraviti prethodni program tako da se na svim mjestima umjesto polazivača koristi **shared\_ptr**

Rješenje se nalazi u Visual Studio projektu u prilogu.

# Zadatak 6.

Dodajte atribut PDV\_procenat u aplikaciju (struktura StoreApp). Dodajte maloprodajne cijene u strukturu Proizvod. Treba dodati opciju za modifikaciju PDV-a ali tako da izmjena neće uticati na postojeće podatke.

Autor:

mr. Adil Joldić

[adil@edu.fit.ba](mailto:adil@edu.fit.ba)