

Laboratorijska Vježba 3. Kolekcije Podataka u C#

Cilj vježbe:

- Upoznavanje sa različitim vrstama kolekcija podataka;
- Korištenje lambda-funkcija u metodama za rad sa kolekcijama podataka.

1. Korištenje kolekcija podataka u C#

Osnovna biblioteka *System* sadrži osnovne tipove podataka, kao i neke kompleksnije tipove (npr. *String* i *Int32*), ali ova biblioteka najčešće nije dovoljna za pravljenje kompleksnijih aplikacija. U prethodnoj vježbi korišteni su klasični nizovi čije se korištenje izbjegava, jer postoje naprednije klase koje omogućavaju jednostavniji rad sa nizovima. Te klase nazivaju se **kolekcije podataka** i u nastavku će biti objašnjen način na koji se koriste u C#.

1.1. Korištenje *Tuple* klase sa parovima vrijednosti

Jedna od veoma često korištenih vrsta kolekcija podataka je ***Tuple*** klasa. Ova klasa predstavlja par dvije proizvoljne vrijednosti i na taj način enkapsuliše dvije vrijednosti u jednu, što može biti veoma korisno kada se želi izbjeći korištenje više parametara ili različitih nizova u slučajevima kada postoje parovi vrijednosti. Korištenje ove klase može se demonstrirati na jednostavnom primjeru, u kojem je potrebno da se za neki predefinisani niz ocjena napravi histogram, odnosno izračuna broj ponavljanja svake ocjene u nizu, a zatim broj ponavljanja pretvori u postotak od ukupnog broja ocjena u nizu.

Prvo što je neophodno uraditi je inicijalizirati nizove podataka. U ovom programu nije potreban korisnički unos, već se ocjene koje se trebaju izbrojati prethodno inicijaliziraju na način prikazan u listingu ispod. I niz parova za sve ocjene od 6 do 10 se inicijalizira na isti način, s tim što je svaki par potrebno inicijalizirati koristeći ključnu riječ **new**, budući da se radi o složenom, a ne o primitivnom tipu podataka.

```
// inicijalizacija nizova podataka
int[] ocjene = new int[15]
{
    6,6,6,7,7,7,8,8,8,8,9,9,10,10
};

Tuple<int, double>[] ponavljanjaOcjena = new Tuple<int, double>[5]
{
    new Tuple<int, double>(6, 0),
    new Tuple<int, double>(7, 0),
    new Tuple<int, double>(8, 0),
    new Tuple<int, double>(9, 0),
    new Tuple<int, double>(10, 0)
};
```

Nakon inicijalizacije podataka, potrebno je izvršiti brojanje ponavljanja svake od ocjena u nizu ocjena. To je najlakše izvršiti koristeći *foreach* petlju i indeksiranje oduzimajući broj 6 (jer je ocjena 6 na prvom mjestu u nizu ocjena). Veoma je važno napomenuti da parovi

podataka nisu promjenjivi, odnosno nakon inicijalizacije nije moguće promijeniti vrijednost elementa u paru. Iz tog razloga nije moguće napisati liniju koda **ponavljanjaOcjena[ocjena-6].Item2 += 1**, jer bi se ovom naredbom pokušala izvršiti izmjena nad drugim elementom para (brojem ponavljanja) za ocjenu 6. Umjesto toga, potrebno je napraviti novi par u kojem će se iskoristiti postojeće vrijednosti koje se pri inicijalizaciji mogu proizvoljno specificirati. U isječku koda ispod izvršena je upravo takva akcija.

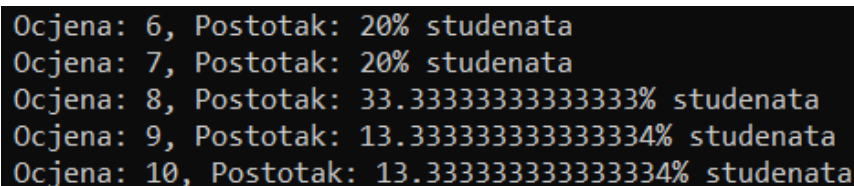
```
// računanje broja ponavljanja svih ocjena
foreach(var ocjena in ocjene)
    ponavljanjaOcjena[ocjena - 6] = new Tuple<int, double>(ocjena,
ponavljanjaOcjena[ocjena - 6].Item2 + 1);
```

Još je potrebno pretvoriti broj ponavljanja u prosjek, a zatim ispisati rezultate na ekranu. Za formiranje prosjeka potrebno je drugi element para izmijeniti po formuli brojPonavljanja/ukupni broj ocjena u nizu * 100 (što se vrši formiranjem novog para na isti način kao u prethodnom isječku koda), a ispis rezultata vrši se koristeći *foreach* petlju kojom se pristupa svim pojedinačnim elementima niza ponavljanja ocjena, odnosno svim ocjenama i pojedinačnim postotcima.

```
// pretvaranje broja ponavljanja u postotke
for (int i = 0; i < ponavljanjaOcjena.Length; i++)
    ponavljanjaOcjena[i] = new Tuple<int,
double>(ponavljanjaOcjena[i].Item1, ponavljanjaOcjena[i].Item2 / ocjene.Length *
100);

// prikaz rezultata
foreach (var ocjena in ponavljanjaOcjena)
    Console.WriteLine(" Ocjena: " + ocjena.Item1 + ", Postotak: " +
ocjena.Item2 + "% studenata");
```

Izlaz iz ovako definisanog programa prikazan je na Slici 1. Vidljivo je da je postotak ispravno određen, kao i da se ocjene ispisuju redom kako su i inicijalizirane. Eventualno sortiranje ocjena prije prikaza i prikaz fiksnog broja decimala ostavlja se studentima za samostalno istraživanje.



```
Ocjena: 6, Postotak: 20% studenata
Ocjena: 7, Postotak: 20% studenata
Ocjena: 8, Postotak: 33.33333333333333% studenata
Ocjena: 9, Postotak: 13.333333333333334% studenata
Ocjena: 10, Postotak: 13.333333333333334% studenata
```

Slika 1. Izlaz iz programa za računanje broja ponavljanja ocjena

1.2. Korištenje *List* klase za lakši rad s nizovima

Jedna od najčešće korištenih klasa za rad sa svim vrstama podataka je **List** klasa. Ova klasa predstavlja naprednu verziju niza i posjeduje veliki broj metoda za manipulaciju elementima kolekcije. Najčešće korištene metode date su u tabeli ispod i one olakšavaju rad s kolekcijama, jer se korištenje petlji i indeksiranja uz vršenje željenih operacija može zamijeniti jednom

linijom koda, što olakšava čitljivost koda i smanjuje mogućnost pravljenja grešaka. Sve informacije o *List* klasi i njenim metodama dostupne su na sljedećem linku: <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0>.

Clear	Briše sve elemente liste
Add	Dodaje novi element na kraj liste
Insert	Ubacuje novi element na specificiranu poziciju
Remove	Briše element iz liste
IndexOf	Vraća indeks elementa u listi
Contains	Provjerava da li specificirani element postoji u listi

Korištenje listi biti će demonstrirano na jednostavnom primjeru. Neka je potrebno abecedno sortirati listu studenata. Svaki student ima svoje ime i broj indeksa. Ukoliko je broj indeksa manji od 16000, studenta treba obrisati iz liste. Ukoliko dva studenta imaju isto ime, potrebno ih je sortirati prema broju indeksa. Ukoliko u listi postoji više studenata sa istim brojem indeksa, potrebno je prikazati poruku o grešci i prekinuti rad programa. U suprotnom je potrebno ispisati listu sortiranih studenata.

Prvi korak je definisanje liste studenata. Kako je lista kompleksni tip podataka, potrebno je inicijalizirati koristeći ključnu riječ **new**, a sami studenti su parovi vrijednosti sa imenom i indeksom, zbog čega je najlakše iskoristiti *Tuple* klasu sa *string* i *int* vrijednostima. Kako pojedinačni elementi parova neće biti mijenjani, već samo njihove pozicije u listi, ova klasa ne bi trebala ni na kakav način otežati rad sa podacima zbog nemogućnosti promjene pojedinačnih elemenata svakog para.

```
// inicijalizacija liste studenata
List<Tuple<string, int>> studenti = new List<Tuple<string, int>>()
{
    new Tuple<string, int>("Emina", 16000),
    new Tuple<string, int>("Amina", 15000),
    new Tuple<string, int>("Edina", 18000),
    new Tuple<string, int>("Emin", 17000),
    new Tuple<string, int>("Edin", 19001),
    new Tuple<string, int>("Edin", 19000),
    new Tuple<string, int>("Amina", 15001)
};
```

Sada je potrebno izvršiti željene operacije nad listom. Prvo će se izvršiti brisanje svih studenata koji imaju broj indeksa manji od 16000. Najlakše je prvo proći kroz listu koristeći *foreach* petlju a zatim obrisati sve takve studente iz liste. Kako je brisanje operacija koja mijenja sadržaj liste, operacija *Remove* (ili, u slučaju korištenja *for* petlje, *RemoveAt*) vrši se nakon prolaska kroz listu i spašavanja svih studenata koji se žele obrisati iz liste. Dakle, prvo je potrebno proći kroz listu studenata i dodati sve studente koji se trebaju obrisati iz liste koristeći funkciju **Add**, a zatim proći kroz listu svih studenata koji se žele obrisati i obrisati ih iz liste koristeći funkciju **Remove**, na način prikazan u isječku koda ispod.

```
List<Tuple<string, int>> studentiZaBrisanje = new List<Tuple<string,
int>>();

// pronalazak studenata koji se trebaju obrisati
foreach (var student in studenti)
    if (student.Item2 < 16000)
        studentiZaBrisanje.Add(student);

// brisanje studenata
foreach (var student in studentiZaBrisanje)
    studenti.Remove(student);
```

Sada je potrebno sortirati studente. Prvi kriterij za sortiranje je ime, a ukoliko dva studenta imaju isto ime, onda je potrebno sortirati ih prema broju indeksa. Ukoliko se pronađu dva ista indeksa, potrebno je ispisati poruku o grešci i završiti s radom programa. Za granicu *for* petlji koje se koriste za iteriranje kroz kolekciju i algoritam sortiranja koristi se funkcija **Count**. Za provjeru abecednog poretka elemenata koristi se funkcija **string.Compare** koja vraća negativan broj ukoliko je prvi string abecedno ispred drugog, a nulu ukoliko su stringovi isti. Na kraju se vrši zamjena najmanjeg pronađenog elementa s trenutnim.

```
// sortiranje preostalih studenata
for (int i = 0; i < studenti.Count; i++)
{
    int min = i;
    for (int j = i + 1; j < studenti.Count; j++)
    {
        // indeksi isti - potrebno ispisati poruku o grešci
        if (studenti[j].Item2 == studenti[min].Item2)
        {
            Console.WriteLine("U listi postoje dva studenta sa istim
indeksom - greška!");
            return;
        }

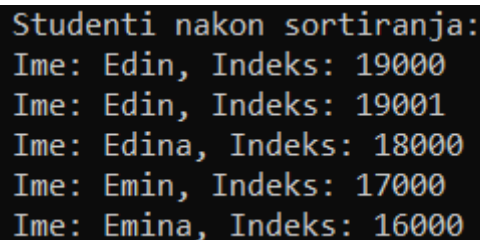
        // pronađeni element je abecedno prije trenutno najmanjeg
        elementa
        if (string.Compare(studenti[j].Item1, studenti[min].Item1) < 0)
            min = j;

        // elementi su isti - potrebno provjeriti broj indeksa
        else if (string.Compare(studenti[j].Item1, studenti[min].Item1)
== 0)
        {
            // indeks pronađenog elementa manji od trenutno najmanjeg
            if (studenti[j].Item2 < studenti[min].Item2)
                min = j;
        }
    }

    // zamjena elemenata
    Tuple<string, int> temp = studenti[i];
    studenti[i] = studenti[min];
    studenti[min] = temp;
}
```

Preostaje još da se izvrši prikaz rezultata, što se vrši koristeći *foreach* petlju za iteriranje kroz listu i pristupanjem pojedinačnim elementima svakog para vrijednosti za sve studente, na način prikazan u isječku koda ispod. Na Slici 2 prikazan je izlaz za ovako definisan program – dva studenta sa brojem indeksa manjim od 16000 su obrisana, a ostali studenti su sortirani abecedno (i prema indeksima, u slučaju istog imena).

```
// prikaz rezultata
Console.WriteLine("Studenti nakon sortiranja:");
foreach(var student in studenti)
{
    Console.WriteLine("Ime: " + student.Item1 + ", Indeks: " +
student.Item2.ToString());
}
```



```
Studenti nakon sortiranja:
Ime: Edin, Indeks: 19000
Ime: Edin, Indeks: 19001
Ime: Edina, Indeks: 18000
Ime: Emin, Indeks: 17000
Ime: Emina, Indeks: 16000
```

Slika 2. Prikaz rezultata sortiranja studenata

1.3. Korištenje lambda-funkcija u metodama za rad sa kolekcijama podataka

Prethodno kreirani program može se pojednostaviti koristeći ugrađene metode koje *List* klasa posjeduje. Pretraga elemenata ne mora se vršiti manuelnim prolaskom kroz *for* petlju, jer postoje metode klase *List* koje kao parametar primaju **lambda-funkciju** koja definiše sekvencu akcija koja će se izvršiti za sve elemente liste. Ovisno od metode koja se koristi, sekvenca akcija može se iskoristiti da promijeni elemente, provjeri da li ijedan element ispunjava neki uslov i zatim te elemente vrati kao rezultat i sl. U nastavku će biti demonstrirano korištenje lambda-funkcija za pojednostavljenje prethodno kreiranog programa.

Prvo što se treba pojednostaviti je način brisanja studenata koji imaju indeks manji od 16000. Studente sa takvim indeksom moguće je pronaći koristeći metodu *FindAll*, pa zatim izvršiti brisanje svih elemenata iz te liste. Jednostavnije je cijeli proces brisanja zamijeniti metodom *RemoveAll*. Kako metoda ***RemoveAll*** kao parametar prima lambda-funkciju, ovom funkcijom definisati će se da se žele obrisati svi studenti koji imaju vrijednost indeksa manju od 16000. Na taj način cijeli proces brisanja ovih studenata mijenja se jednom linijom koda koja je lakša za razumijevanje i manje podložna greškama od koda u prethodnoj implementaciji, na način prikazan u isječku koda ispod.

```
// brisanje svih studenata koji ispunjavaju kriterij
studenti.RemoveAll(student => student.Item2 < 16000);
```

Prije sortiranja studenata potrebno je provjeriti da li u listi postoje dva studenta sa istim indeksom. U tu svrhu potrebno je izvršiti sljedeću sekvencu akcija:

- Nad listom studenata primijeniti metodu **Any** koja provjerava da li postoji ijedan student koji ispunjava zadani uslov. Zadani uslov je da u listi postoji više od jednog studenta koji imaju isti indeks. Ukoliko metoda u uslovu vrati vrijednost veću od 1, metoda **Any** kao rezultat vraća *true*, odnosno to znači da više od jednog studenta u listi imaju definisan isti indeks (što nije dozvoljeno).
- Zadani uslov definisati putem metode **Where** koja kao rezultat vraća sve studente koji ispunjavaju drugi željeni uslov. Drugi željeni uslov vraća sve studente koji imaju isti indeks kao trenutni student koji se posmatra u funkciji **Any**. Metoda **Where** kao rezultat vraća generički niz koji se putem metode **ToList** pretvara u listu, a zatim se provjerava broj elemenata te liste putem metode **Count**.

Prethodno opisana sekvenca akcija odvija se u jednoj liniji koda, koja se može interpretirati na sljedeći način: „*ako postoji ijedan student za kojeg postoji student2 sa istim indeksom kao taj student*“. Ukoliko je uslov ispunjen, prikazuje se poruka o grešci i terminira rad programa, dok se u suprotnom nastavlja sa sortiranjem studenata, o čemu će biti riječi u nastavku.

```
// pokušaj pronalaska dva studenta s istim indeksom
if (studenti.Any(student => studenti.Where(student2 => student2.Item2
== student.Item2).ToList().Count > 1))
{
    Console.WriteLine("U listi postoje dva studenta sa istim indeksom -
greška!");
    return;
}
```

Sam proces sortiranja prethodno je bio definisan putem velikog broja provjeravanja uslova (jer postoje dva uslova sortiranja – sortiranje abecedno i sortiranje po indeksu za studente s istim imenom). Nakon toga je bilo neophodno izvršiti zamjenu mjesta pronađenih elemenata kako bi se sortiranje efektivno izvršilo. Sve ove korake sada je moguće zamijeniti sa jednom metodom **OrderBy** kojom se vrši sortiranje elemenata prema željenom atributu (u ovom slučaju to je prvi element para, odnosno *string* vrijednost). Kako postoji i drugi uslov sortiranja za elemente gdje nije moguće izvršiti sortiranje po prvom uslovu, potrebno je iskoristiti i metodu **ThenBy** i definisati drugi željeni atribut (u ovom slučaju to je drugi element para, odnosno *int* vrijednost). Kako ove metode vraćaju generičku listu kao rezultat, još je potrebno upotrijebiti metodu **ToList** kako bi se pretvorila u željenu listu, a zatim je tu listu potrebno sačuvati u varijablu *studenti*.

```
// sortiranje preostalih studenata
studenti = studenti.OrderBy(student => student.Item1).ThenBy(student =>
student.Item2).ToList();
```

Prikaz rezultata ostaje isti, tako da je ovime proces zamjene jednostavne programske logike naprednijim načinom korištenja kolekcija podataka završen. Na ovaj način korištenje petlji zamijenjeno je sa funkcijama koje su jednostavne za razumijevanje. Ove petlje u općem slučaju su optimizirane i brže, kao i manje podložne greškama od korisnički definisanih petlji i funkcija, pa se njihovo korištenje preporučuje kad god je to moguće.

2. Zadaci za samostalni rad

Programski kod aplikacije koja je kreirana u vježbi dostupan je u repozitoriju na sljedećem linku: <https://github.com/ehlymana/OOADVjezbe>, na *branchu* **lv3**.

1. Implementirajte i analizirajte zadatke sa prošle vježbi korištenjem naprednijih tipova kolekcija podataka.
2. Implementirajte metodu *ToSortedArray* (*List<string> lista*) koja će kao parametar primiti listu stringova, a kao rezultat vratiti novu listu stringova sa svim elementima iz prve liste. Elementi u listi trebaju biti sortirani prema dužini (najkraći string prvi, najduži string posljednji).
3. Napravite program koji omogućava korisniku da unese deset brojeva. Ukoliko su svi uneseni brojevi parni, potrebno je zanemariti sve neparne brojeve u daljem radu programa. U suprotnom, potrebno je zanemariti sve parne brojeve. Nakon toga potrebno je izračunati sumu svih preostalih brojeva. Ukoliko je suma negativan broj, potrebno je ispisati poruku o grešci. U suprotnom, potrebno je ispisati sve elemente koji nisu zanemareni na početku rada programa i dobivenu sumu.
4. Napravite program koji za datu listu stringova vraća broj stringova koji su abecedno ispred riječi „etf“, broj stringova koji su isti kao ta riječ, kao i broj stringova koji su međusobno jednaki u toj listi.