1. Uvod u liste

Lista predstavlja kolekciju elemenata istog tipa podataka. Slično kao niz, omogućava da više vrednosti čuvamo pod jednim zajedničkim nazivom.

Glavna razlika je u tome što se broj elemenata niza određuje prilikom njegovog kreiranja i kasnije se ne može promeniti, dok se broj elemenata liste tokom izvršavanja programa može menjati.

Kod liste nije potrebno unapred odrediti koliko će elemenata sadržati:

```
List<int> brojevi = new List<int>();
```

Tip naveden između znakova < > određuje tip elemenata koje lista može da sadrži. Lista je u ovom trenutku prazna, ali joj se kasnije mogu dodavati elementi.

Mogu se kreirati liste različitih tipova:

```
List<int> brojevi = new List<int>();
List<string> gradovi = new List<string>();
List<double> temperature = new List<double>();
List<char> znakovi = new List<char>();
```

Ako već znamo početne vrednosti, možemo ih navesti prilikom kreiranja liste:

```
List<int> brojevi = new List<int>
{
    21, 4, 94, 10, 2
};
List<string> gradovi = new List<string>
{
    "Sombor", "Subotica", "Beograd"
};
```

2. Kreiranje liste, indeksiranje i svojstvo Count

Kao i kod niza, elementi liste imaju svoj položaj, odnosno indeks. Indeksiranje počinje od nule, pa prvi element ima indeks 0, drugi indeks 1 itd.

Pristup elementima liste

Elementu liste pristupa se navođenjem njegovog indeksa između uglastih zagrada: `brojevi[indeks]`

Na primer:

```
List<int> brojevi = new List<int>
{
    21, 4, 94, 10, 2
};

Console.WriteLine(brojevi[0]); // 21
Console.WriteLine(brojevi[2]); // 94
Console.WriteLine(brojevi[4]); // 2
```

Promena vrednosti elementa

Vrednost postojećeg elementa može se promeniti pomoću njegovog indeksa.

Na primer:

```
List<int> brojevi = new List<int>
{
    12, 7, 5, 18, 9
};

brojevi[1] = 20; // brojevi: 12 20 5 18 9
brojevi[2] = brojevi[3] - brojevi[0]; // brojevi: 12 20 6 18 9
brojevi[4] += brojevi[1]; // brojevi: 12 20 6 18 29
```

Svojstvo Count

Kod nizova se broj elemenata dobija pomoću svojstva Length, a kod lista se koristi svojstvo Count.

Na primer:

```
List<int> brojevi = new List<int>
{
    12, 7, 5, 18, 9
};

Console.WriteLine(brojevi.Count); // rezultat je 5
```

3. Dodavanje, umetanje i uklanjanje elemenata

Dodavanje elementa – Add()

Metoda Add() dodaje novi element na kraj liste. Svaki poziv metode Add() povećava broj elemenata liste za jedan. Piše se `lista.Add(vrednost);`

```
List<string> gradovi = new List<string>();

gradovi.Add("Sombor"); // gradovi: Sombor
gradovi.Add("Subotica"); // gradovi: Sombor Subotica
gradovi.Add("Beograd"); // gradovi: Sombor Subotica Beograd
```

Umetanje elementa – Insert()

Metoda Insert() dodaje element na određeni indeks. Piše se `lista.Insert(indeks, vrednost);`

Na primer (ako su članovi liste Sombor, Subotica i Beograd):

```
gradovi.Insert(1, "Zrenjanin"); // gradovi: Sombor Zrenjanin Subotica Beograd
```

Naziv Zrenjanin umeće se na indeks 1.

Uklanjanje elementa po vrednosti – Remove()

Metoda Remove() pokušava da ukloni element određene vrednosti. Piše se `lista.Remove(vrednost);`

Na primer (ako su članovi liste Sombor, Subotica i Beograd):

```
gradovi.Remove("Subotica"); // gradovi: Sombor Beograd
```

Ako vrednost postoji više puta i iskoristimo Remove() ukloniće se samo prva pojava te vrednosti.

Uklanjanje elementa po indeksu – RemoveAt()

Metoda RemoveAt() uklanja element koji se nalazi na određenom indeksu. Piše se `lista.RemoveAt(indeks);`

Na primer (ako su članovi liste Sombor, Subotica i Beograd):

```
gradovi.RemoveAt(0); // gradovi: Subotica Beograd
```

Elementi koji se nalaze iza uklonjenog elementa pomeraju se za jedno mesto ulevo.

Uklanjanje svih elemenata – Clear()

Metoda Clear() uklanja sve elemente liste. Piše se `lista.Clear();`

Na primer:

```
gradovi.Clear(); // gradovi:
```

Sama lista i dalje postoji, samo više ne sadrži nijedan element.

4. Prolazak kroz listu (for i foreach) i unos elemenata

Kao i kod jednodimenzionalnih nizova, kroz elemente liste možemo prolaziti pomoću petlji for i foreach. Osnovna razlika je da for omogućava pristup indeksu, dok foreach prolazi direktno kroz vrednosti elemenata.

Prolazak pomoću for petlje

Petlja for koristi se kada nam je potreban indeks elementa.

Posmatrajmo listu:

```
List<int> brojevi = new List<int>
{
    4, 8, 3, 6, 2
};
```

Kroz sve elemente možemo proći ovako:

```
for (int i = 0; i < brojevi.Count; i++)
{
    Console.WriteLine(brojevi[i]);
}
```

Promenljiva i predstavlja indeks elementa koji se trenutno obrađuje.

Za razliku od niza, gde se u uslovu petlje koristi Length, kod liste koristimo lista.Count.

Prolazak pomoću foreach petlje

Petlja foreach prolazi direktno kroz vrednosti elemenata.

```
List<int> brojevi = new List<int>
{
    4, 8, 3, 6, 2
};

foreach (int broj in brojevi)
{
    Console.WriteLine(broj);
}
```

Kod foreach petlje nije potrebno koristiti indeks ni svojstvo Count.

Razlika između for i foreach

for	foreach
Omogućava pristup indeksu	Pristupa direktno vrednosti
Koristi Count	Nije potrebno koristiti Count
Može menjati element preko indeksa	Najčešće se koristi za čitanje vrednosti
Pogodan kada je položaj važan	Jednostavniji za prolazak kroz celu listu

Unos elemenata u listu

Kod liste možemo krenuti od prazne liste i svaki uneti element dodavati pomoću metode Add().

Unos poznatog broja elemenata

Ako želimo da korisnik unese pet brojeva:

```
List<int> brojevi = new List<int>();

for (int i = 0; i < 5; i++)
{
    Console.Write($"Unesite broj {i + 1}: ");
    int broj = int.Parse(Console.ReadLine());

    brojevi.Add(broj);
}
```

Važno je primetiti da ne pišemo brojevi[i] = broj; jer prazna lista još nema element sa indeksom i.

Umesto toga koristimobrojevi.Add(broj); kojim svaki novi element dodajemo na kraj liste.

Broj elemenata određuje korisnik

Korisnik može prvo uneti koliko elemenata želi da doda:

```
List<int> brojevi = new List<int>();

Console.Write("Koliko brojeva želite da unesete? ");
int brojElemenata = int.Parse(Console.ReadLine());

for (int i = 0; i < brojElemenata; i++)
{
    Console.Write($"Unesite broj {i + 1}: ");
    int broj = int.Parse(Console.ReadLine());

    brojevi.Add(broj);
}
```

Unos nepoznatog broja elemenata

Jedna od prednosti liste je što unapred ne moramo znati koliko će elemenata biti uneto. Na primer, korisnik može unositi brojeve sve dok ne unese 0:

```
List<int> brojevi = new List<int>();
Console.Write("Unesite broj (0 za kraj): ");
int broj = int.Parse(Console.ReadLine());

while (broj != 0)
{
    brojevi.Add(broj);

    Console.Write("Unesite broj (0 za kraj): ");
    broj = int.Parse(Console.ReadLine());
}
```

Vrednost 0 se ne dodaje u listu, jer predstavlja znak da je unos završen.

5. Pretraga, sortiranje i korisne metode

Liste u C# imaju ugrađene metode koje olakšavaju pronalaženje elemenata i promenu njihovog redosleda. Umesto da za svaki jednostavan slučaj sami pišemo petlju, možemo koristiti metode kao što su Contains(), IndexOf(), LastIndexOf(), Sort() i Reverse().

Provera da li lista sadrži element – Contains()

Metoda Contains() proverava da li se određena vrednost nalazi u listi.

Rezultat je logička vrednost, true ako element postoji odnosno false ako element ne postoji u listi.

Na primer:

```
List<int> brojevi = new List<int>
{
    4, 8, 3, 6, 2
};

Console.WriteLine(brojevi.Contains(8));  // True
Console.WriteLine(brojevi.Contains(10)); // False
```

Metoda se često koristi zajedno sa proverom nekog uslova:

```
Console.Write("Unesite broj za pretragu: ");
int broj = int.Parse(Console.ReadLine());

if (brojevi.Contains(broj))
{
    Console.WriteLine("Broj se nalazi u listi.");
}
else
{
    Console.WriteLine("Broj se ne nalazi u listi.");
}
```

Pronalaženje indeksa elementa – IndexOf()

Metoda IndexOf() pronalazi indeks prve pojave određene vrednosti.

Na primer:

```
List<int> brojevi = new List<int>
{
    4, 8, 3, 6, 2, 3, 1
};

Console.WriteLine(brojevi.IndexOf(3)); // 2
Console.WriteLine(brojevi.IndexOf(10)); // -1
```

Ako se element nalazi više puta u listi metoda vraća samo indeks prve pojave elementa. Ako se element ne nalazi u listi metoda vraća vrednost -1.

Poslednja pojava elementa – LastIndexOf()

Ako želimo da pronađemo indeks poslednje pojave vrednosti, možemo koristiti LastIndexOf().

Na primer:

```
List<int> brojevi = new List<int>
{
    5, 8, 3, 8, 2
};

Console.WriteLine(brojevi.LastIndexOf(8)); // 3
```

Dakle metoda IndexOf() vraća prvu pojavu tražene vrednosti, a LastIndexOf() poslednju pojavu tražene vrednosti.

Sortiranje liste – Sort()

Metoda Sort() uređuje elemente liste. Kod brojeva ih raspoređuje od najmanjeg ka najvećem, a kod teksta po abecedi.

Na primer:

```
List<int> brojevi = new List<int>
{
    8, 3, 12, 2, 6
};

brojevi.Sort(); // 2  3  6  8  12
```

Obrtanje redosleda – Reverse()

Metoda Reverse() obrće trenutni redosled elemenata liste.

Na primer:

```
List<int> brojevi = new List<int>
{
    4, 8, 3, 6
};

brojevi.Reverse(); // 6  3  8  4
```

Reverse() ne sortira elemente, već samo obrće njihov trenutni redosled.