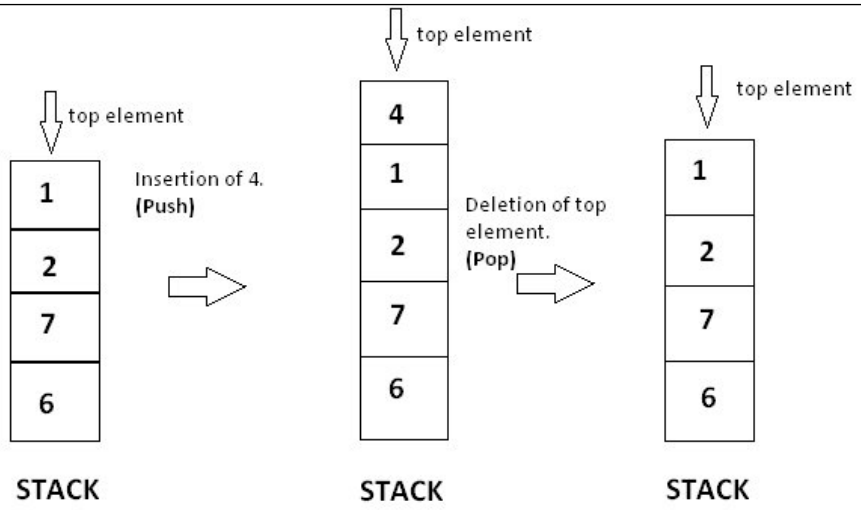
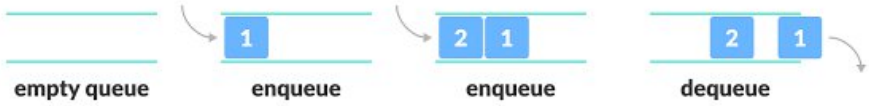
11. Kolekcijas (1.daļa)

Kolekcijas ir līdzīgas masīviem, tas nodrošina elastīgāku veidu, kā strādāt ar objektu grupu. Masīvos jūs būtu pamanījuši, ka jums iepriekš ir jādefinē elementu skaits masīvā. Tas bija jādara, kad tika deklarēts masīvs. Bet kolekcijā jums **nav iepriekš jānosaka kolekcijas lielums**. Jebkurā brīdī varat pievienot elementus vai pat noņemt elementus no kolekcijas. Šajā nodaļā galvenā uzmanība tiks pievērsta tam, kā mēs varam strādāt ar dažādām kolekcijām, kas pieejamas C#.
Kolekciju izmantošanai projektam jāpievieno atsauce using System.Collections;

Kolekcijas nosaukums	Paskaidrojums
ArrayList	Tā attēlo sakārtotu objektu kolekciju (var būt ar dažādiem datu tipiem), ko var indeksēt atsevišķi. Būtībā tā ir alternatīva masīvam. Tomēr atšķirībā no masīva varat pievienot un noņemt vienumus no saraksta noteiktā vietā, izmantojot indeksu, un kolekcijas automātiski maina izmērus. Tas arī ļauj dinamiski piešķirt atmiņu, pievienot, meklēt un kārtot vienumus sarakstā.
Hashtable	Tā izmanto atslēgu, lai piekļūtu kolekcijas elementiem. Jaukšanas tabula tiek izmantota, ja jums ir nepieciešams piekļūt elementiem, izmantojot atslēgu, un jūs varat identificēt noderīgu atslēgas vērtību. Katram jaukšanas tabulas vienumam ir atslēgas/vērtības pāris. Atslēga tiek izmantota, lai piekļūtu kolekcijas vienumiem.
SortedList	Tā izmanto atslēgu, kā arī indeksu, lai piekļūtu saraksta vienumiem. Sakārtots saraksts ir masīva un hashtable tabulas kombinācija. Tajā ir to vienumu saraksts, kuriem var piekļūt, izmantojot atslēgu vai indeksu. Ja vienumiem piekļūstat, izmantojot indeksu, tas ir ArrayList, un, ja vienumiem piekļūstat, izmantojot atslēgu , tas ir Hashtable. Vienumu kolekcija vienmēr tiek sakārtota pēc atslēgas vērtības.
Stack	Tā reprezentē objektu kolekciju pēc **LIFO principa** (last in, first out). Kad sarakstam pievienojat vienumu, to sauc par **push**, un, kad to noņemat, to sauc par vienuma **pop**.

	
Queue	Tā reprezentē objektu kolekciju pēc **FIFO principa** (first in, first out). Kad sarakstā pievienojat vienumu, to sauc par **enqueue**, un, noņemot vienumu, to sauc par **dequeue**. 
BitArray	Tā attēlo bināru masīvu, izmantojot vērtības 1 un 0. To izmanto, ja nepieciešams saglabāt bitus, bet iepriekš nezināt bitu skaitu. Varat piekļūt vienumiem no BitArray kolekcijas, izmantojot veselu skaitļu indeksu, kas sākas no nulles.

5.1. ArrayList

ArrayList kolekcija ir līdzīga masīvam. Lielākā atšķirība ir masīvu sarakstu kolekcijas dinamiskais raksturs. Masīviem ir jādefinē elementu skaits, ko masīvs var saturēt masīva deklarēšanas laikā. Bet kolekcijas gadījumā tas nav jādara iepriekš. Elementus var pievienot vai noņemt no kolekcijas Array List jebkurā brīdī.

ArrayList izveidošana ir sniegta zemāk. Masīvu saraksts tiek izveidots ar **ArrayList** datu tipa palīdzību. Atslēgvārds "jauns" tiek izmantots, lai izveidotu ArrayList objektu. Pēc tam objekts tiek piešķirts mainīgajam **a1**. Tāpēc tagad mainīgais **a1** tiks izmantots, lai piekļūtu dažādiem masīvu saraksta elementiem.

```
ArrayList a1 = new ArrayList();
```

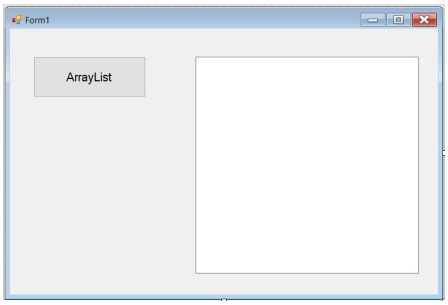
Add metode tiek izmantota, lai pievienotu elementu ArrayList. Pievienošanas metodi var izmantot, lai masīvu sarakstam pievienotu jebkāda veida datu tipa elementus. Tātad ArrayList varat pievienot veselu skaitli vai virkni vai pat Būla vērtību:

```
a1.Add(element);
```

Zemāk varat redzēt piemērus, kā ArrayList kolekcijai var pievienot veselu skaitļu, simbolu virkni un pat Būla vērtības:

a1.Add(1); — kolekcijai tiks pievienota vesela skaitļa vērtība;
a1.Add("Piemers"); —kolekcijai tiks pievienota simbolu virknes vērtība;
a1.Add(true); —kolekcijai tiks pievienota Būla vērtība.

Apskatīsim visu piemēru C# vizuālajā aplikācijā:




```
private void button1_Click(object sender, EventArgs e)
{
   ArrayList al = new ArrayList();

   //Dažu skaitļu pievienošana
   al.Add(45);
   al.Add(78);
   al.Add(33);
   al.Add(56);
   al.Add(12);
   al.Add(23);
   al.Add(9);

   textBox1.Text = "Kolekcijas saturs:"+Environment.NewLine;
   foreach (int i in al)
   {
      textBox1.Text=textBox1.Text+ i.ToString() + Environment.NewLine;
   }
   textBox1.Text = textBox1.Text+"Sašķirotais saturs:" + Environment.NewLine;
   //kolekcijas elementu šķirošana
   al.Sort();
   foreach (int i in al)
   {
      textBox1.Text = textBox1.Text + i.ToString() + Environment.NewLine;
   }
}
```

Metode **Add** ļauj pievienot jauno objektu kolekcijai, savukārt metode **AddRange** ļauj pievienot kolekcijai elementu virkni no citas kolekcijas

```
   //izveidojam kolekciju
   ArrayList kolekcija = new ArrayList();
   //un pievienojam trīs elementus
   kolekcija.Add(10);
   kolekcija.Add("otrais");
   kolekcija.Add(100);
   //vēl kolekcija
   ArrayList lielaKolekcija = new ArrayList();
   //pievienojam elementu
   lielaKolekcija.Add(1);
   //un visus elementus no pirmās kolekcijas
```

```
lielaKolekcija.AddRange(kolekcija);
for (int i = 0; i < kolekcija.Count; i++)
{//foreach ciklu izmantot nevaram, jo kolekcijā ir dažāda tipa mainīgie
   string t = kolekcija[i].ToString();
   textBox1.Text = textBox1.Text + t + " ";
}
textBox1.Text = textBox1.Text+Environment.NewLine;
for (int i = 0; i < lielaKolekcija.Count; i++)
{
   string t = lielaKolekcija[i].ToString();
   textBox1.Text = textBox1.Text + t + " ";

}
```

Metode **Remove** nodzēš objektu no kolekcijas pēc norādītas vērtības, savukārt metode **RemoveAt** nodzēš elementu ar norādīto indeksu:

```
ArrayList kolekcija = new ArrayList();
kolekcija.Add(10);
kolekcija.Add("otrais");
kolekcija.Add(100);
//dzēšam elementu ar nosaukumu otrais
kolekcija.Remove("otrais");
//kolekcijā palika 2 elementi, tagad dzēšam 1.elementu
kolekcija.RemoveAt(0);
for (int i = 0; i < kolekcija.Count; i++)
{
   string t = kolekcija[i].ToString();
   textBox1.Text = textBox1.Text + t + " ";
}
```

Metode **RemoveRange** nodzēš objektus norādītajā indeksu intervālā, bet metode **Clear** nodzēš visus elementus no kolekcijas.
Metode **Insert** ieraksta kolekcijā elementu ar norādīto indeksu:

```
ArrayList kolekcija = new ArrayList();
kolekcija.Insert(0, "Vēl viens elements");
```

Metode **Sort** sašķiro kolekcijas elementus augošā kārtībā (simboliskās informācijas gadījumā pēc alfabēta), bet metode **Reverse** maina elementu kārtību uz pretējo.

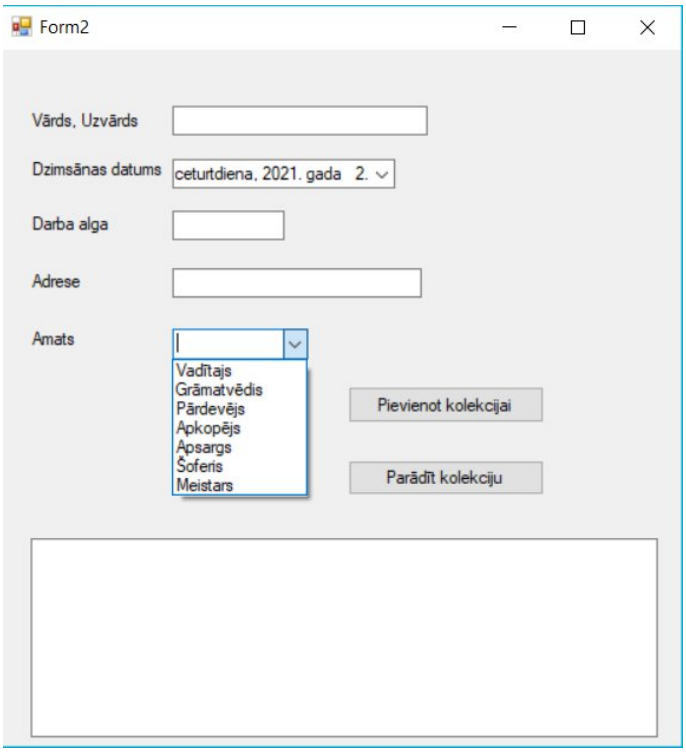
Darbam ar kolekcijām var izmantot kursorus, kurus var izveidot no klases **IEnumerator**. Šāda tipa kursors pārvietojas pa kolekcijas elementiem un satur tikai divas metodes – **MoveNext** (pāriet pie nākošā) un **Reset** (atgriezties kolekcijas sākumā):

```
ArrayList kolekcija = new ArrayList();
kolekcija.Insert(0, "Tāda");
kolekcija.Insert(1, "kolekcija");
kolekcija.Insert(2, "satur");
kolekcija.Insert(3, "svarīgu");
kolekcija.Insert(4, "informāciju");
//izveidojam kursoru mūsu kolekcijai
IEnumerator kursors = kolekcija.GetEnumerator();
//cikls izpildās kamēr visi elementi nebūs izskatīti
while (kursors.MoveNext())
{
   //piešķiram mainīgajam t kursora tekošo vērtību
```

```
            string t = kursors.Current.ToString();
            textBox1.Text = textBox1.Text + t + " ";
        }
```

UZDEVUMI:

1. Izveidot ArrayList kolekciju pirkuma saraksta veidošanai. Jāparedz iespēja pievienot pirkumu kolekcijai, nodzēst un parādīt aktuālo pirkumu sarakstu.
2. Izveidot programmu, kura realizē ievadīto datu saglabāšanu ArrayList kolekcijā un izvadīt visus kolekcijas elementus, lietotāja saskarni izveidojiet pēc zemāk dota piemēra:


Form2
Vārds, Uzvārds
Dzimšanas datums
ceturtdiena, 2021. gada 2.
Darba alga
Adrese
Amats
Vadītajs
Grāmatvēdis
Pārdevējs
Apkopējs
Apsargs
Šoferis
Meistars
Pievienot kolekcijai
Parādīt kolekciju