**11. tétel**

**Adatbázis- és szoftverfejlesztés - Programozási alapismeretek (2.1.1.)**

**Változók létrehozása, tulajdonságai felhasználása**

Ismertesse a C# programozási nyelvben található különböző egyszerű és összetett változó típusokat. Azok deklarálási, valamint értékadási folyamatát. Illetve esetleges felhasználási lehetőségeiket.

***Szempontok a tartalom értékeléséhez***

• Ismerteti a két fő változótípust, és azok elemeit (egyszerű és összetett)

• Bemutatja a deklarációs és értékadási folyamatot, és a köztük lévő különbséget változótípusonként, az egyszerű változók esetén

• Bemutatja a deklarációs és „feltöltési” folyamatot, a tömbök illetve a listák esetében.

Mikor melyik adatstruktúrát, miért érdemes használni.

• Ismertet néhány lehetőséget mikor érdemes az adott változótípust használni, és miért.

Adattípusok:

Egyszerű

* egész szám típusok
* valós szám típusok
* logikai típus
* karakter típus
* szöveg típus

Összetett

* tömbök
* listák
* rekordok

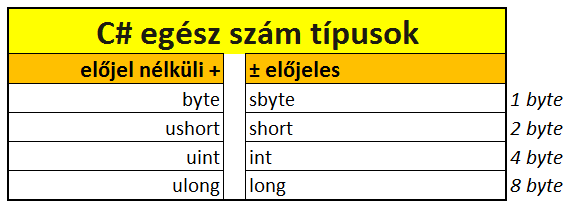
**Egész szám típusok**

Ez egy típuscsalád, konkrétan 8 típus sorolható ide. Ezek a típusok egymástól két dologban különböznek:

* memóriabeli **tárigényük** (ez 1 byte, 2 byte, 4 byte és 8 byte lehet)
* **előjelességük** (vannak előjeles és előjel nélküli változatok)

Ezen két szempont meghatározza az adott konkrét egész szám típus tárolási lehetőségeit. Ez a tárolási lehetőség egy számintervallumot ír le. Például a **byte** típus [0,255] intervallumbeli egész számokat képes tárolni.

Példa deklarálásra és értékadása: **int szam = 5;**



# Valós szám típusok

A valós számtípusok esetén tört (racionális véges) számértékek jelenlétére kell készülnünk. Amennyiben a tárolandó, kezelendő adataink mondjuk hőmérséklet adatok, úgy általában egész számok is elegendők szoktak lenni. Ha súlyadatokkal dolgozunk, mondjuk egy gyógyszertári alkalmazásban, akkor a dekagramm tört részeit is tudnunk kell leírni.

A valós számtípusoknak két konkrét típusa létezik:

A **float** és a **double** típusok helyigényükben, és pontosságukban térnek el. A **double** helyigénye és tárolási pontossága pontosan duplája a **float**-nak (innen is kapta a nevét).

Példa deklarálásra és értékadása: **double valos = 3.14;** vagy **double pi = Math.PI;**

# Logikai típus

A logikai típus használata esetén mindössze két érték tárolását tudjuk végezni: az **igaz**-**hamis**, **igen**-**nem** értéket. A C# programban ezek angol megfelelőjét kell használni, ezek a true és false értékek.

A logikai típus neve bool, amelyet egy matematikusról, **George Boole**-ról kapta, aki a matematikai logika alapjait vetette meg.

A logikai típus tárolási igénye 1 byte. Mivel a számítógép memóriája számértékek tárolására képes, a logikai igaz/hamis értékek tárolása során **trükköt** kell alkalmazni. A logikai hamis (false) értéket 0 számérték, az igaz (true) értéket pedig az 1 számérték jelzi.

**Megj.**: igazából a true értéket a **nem nulla** érték jelöli, vagyis bármi más érték, például az 1, de akár a 2, 3, 126, ezek mind **true** értéket jeleznek.

Példa deklarálásra és értékadása: **bool logikai = true;** vagy **bool logikai = false;**

# Karakter típus

A karakter típus neve **char**, és ezen típus egyetlen karaktert képes kezelni, tárolni.

A karakter fogalma bővebb, mint a **betű**, vagy a **számjegyek**, ebbe beletartoznak az írásjelek, egyéb szimbólumok is, mint pl. az e-mail címekben szereplő @ karakter is.

A karakterek tárolását szintén számokkal kell megoldani. Egy táblázat, név szerint az **ASCII** kódtáblázat írja le melyik karakternek mi a számkódja. Ez tehát egy egyszerű kétoszlopos táblázatnak fogható fel, melyben egyszerűen fel vannak, sorolva melyik karakternek milyen számérték felel meg. A kis **a** betű kódja pl. a 97, míg a nagy **A** betűjé a 65. A É szimbólum kódja a 144.

Példa deklarálásra és értékadása: **char betu = ’a’;** vagy **char szam = ’5’;**

Ascii kódtábla: http://www.asciitable.com/

# Szöveg típus

Amennyiben nem egy, hanem sok karakterről, karaktersorozatról beszélünk, akkor a **char** típus már nem jöhet szóba, helyette a **string** típust kell használnunk.

A **string** típus rugalmas tárigényű. A **szia** szöveg tárolása kevesebb memóriába kerül, mint a **malacka** szövegé. A tárolási igény számításához ismerni kell a tárolandó karaketrek számát. A **szia** szöveg 4 karakteres, karakterenként 2-2 byte igényű, vagyis 8 byte. Ennyi azonban nem elég, mivel a memóriában egymást követő **s**, **z**, **i**, **a** karakterek tárolása 8 byte-on megoldható, de jelen esetben jelezni kell azt is, hogy itt vége is van a szövegnek. Ezt egy újabb, speciális karakterrel kell jelezni. Ez a 0-a kódú karakter a sorozat végére illesztésével oldható meg. Vagyis a **szia** végére még egy plusz karaktert kell tárolni, szintén 2 byte-on.

Példa deklarálásra és értékadása: **int szoveg = „valamiszoveg”;**

**Tömbök típusok (például:vektor)**

A vektor olyan változónak tekinthető, mely nem egyetlen, de sok, több érték tárolására képes egy időben. A **vektor** a **tömb** típuscsoport tagja **(1 dimenziós tömb)**, s mint minden tömb, az alábbiak jellemzik:

* **összetett adattípus**, vagyis egyetlen vektor típusú változó egy időben több értéket is képes tárolni,
* **homogén**, vagyis minden egyes tárolt érték ugyanolyan alaptípusba tartozik, ezt a vektor létrehozásakor meg kell adni
* **random elérésű**, vagyis bármely tárolt érték kiolvasása vagy módosítása ugyanannyi ideig tart
* **fix méretű**, vagyis a vektorok létrehozásakor meg kell adni hány értéket kívánunk tárolni majd benne, és ez a darabszám később nem változhat meg
* **folytonos tárolású**, vagyis a vektorban tárolt értékek a memória egy folytonos területén, egymás mögött elhelyezkedve

Példa deklarálásra: **int[] tomb  =** [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) **int[20];**

Feltöltése számokkal:

**Random rnd=new Random();**

**for(int i=0;i<tomb.Lenght;i++)**

**{**

**tomb[i]=rnd.Next(1,101);**

**}**

A feltöltés esetén bármilyen ciklus megfelel a lényeg az, hogy annak segítségével végigmenjünk a tömbnek az elemein, majd egyesével adjunk neki valamilyen értéket. (Természetesen ez lehet érték generálás, érték beolvasása billentyűzetről, vagy épp külső fájlból)

A kiíratás hasonlóan zajlik, ugyanúgy ciklus kell hozzá, és az elemek egyesével való megjelenítése a lényeg.

**for(int i=0;i<tomb.Lenght;i++)**

**{**

**Console.WriteLine(„{0},”,tomb[i]);**

**}**

**Lista típus**

A lista az összetett adatszerkezetek családjába sorolható. Ennek megfelelően egyetlen **lista** típusú változóban egy időben több értéket is tudunk tárolni - ebben hasonlítanak a vektorokhoz. Abban is hasonlítanak, hogy minden egyes értéknek ugyanazon típusba kell tartoznia (ezt nevezzük a lista alaptípusának). Azonban van egy lényeges eltérés a lista és a vektorok között: **a lista mérete dinamikusan változtatható**.

Ezt úgy kell érteni, hogy míg a vektort létrehozásakor méretezni kell, fix méretre, és ezen méret nem változhat meg a későbbiek során - addig a listák létrehozásukkor üresek. Egy lista induláskor nulla elemet tárol, de később tetszőlegesen sok elemet lehet **hozzáadni** a listához. Több hozzáadási technika is létezik, a legegyszerűbb alkalmazása esetén az új elemek a **lista végére kerülnek**. A lista elemszáma nem végtelen, de gyakorlatilag több milliós elemszámmal is képes dolgozni. Ennél nagyobb elemszám esetén már nem szoktunk listát alkalmazni, helyette vagy hash táblát, vagy külső adattáblákat használunk.

Példa deklarálásra: **List<int> lista  =** [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) **List<int>();**

Feltöltése számokkal:

**Random rnd=new Random();**

**for(int i=0;i<lista.Count;i++)**

**{**

**lista.Add(rnd.Next(1,101));**

**}**

A feltöltés esetén, mint a tömbnél bármilyen ciklus megfelel a lényeg az, hogy annak segítségével végigmenjünk a listának az elemein, majd egyesével adjunk neki valamilyen értéket. (Természetesen ez lehet érték generálás, érték beolvasása billentyűzetről, vagy épp külső fájlból). *A tömbtől eltérően itt a Count tulajdonság adja meg a listának a métertét, valamint a tömbhöz való „hozzáfűzéshez” is külön függvény van ami az Add()*.

A kiíratás hasonlóan zajlik, mint a tömbnél korábban, ugyanúgy ciklus kell hozzá, és az elemek egyesével való megjelenítése a lényeg. *Ügyeljünk ismét arra, hogy a lista méretét a Count tulajdonság adja majd meg!*

**for(int i=0;i<lista.Count;i++)**

**{**

**Console.WriteLine(„{0},”,lista[i]);**

**}**

**Rekord típus**

**Ez külön nincs a tételben említve, de ha visual basicben használtátok már valaha, akkor írhatok róla majd!**