Karakterek, karakterláncok

Összefoglaló

Karakter típus (`char`) segítségével egyetlen unikód UTF-16 karaktert (betűt, számjegyet vagy speciális jeleket) ábrázolhatunk. Egy karakter típus alapértéke a `'\0'`, érték nélküli `''` (üres) karakter nem értelmezett.

A karakterláncok (`string`) ilyen karakterekből (`char`) álló sorozatok, amelyekkel szöveges adatot tárolhatunk. Míg karaktereknél nem lehetséges az üres karakter, addig egy `string` esetén üres karakterláncot is definiálhatunk, ami nem keverendő össze a `null` értékkel. Minden típus karakterlánccá alakítható a **ToString** metódussal, és minden `string` karakterek tömbjévé alakítható a **ToCharArray** metódussal.

```
char aChar = 'a';
char newlineChar = '\n';
string nullString = null;
string emptyString = "";
string question = "Vincent! We happy?";
double pi = 3.14159265359;
string text = pi.ToString();    // "3.14159265359"
```

Műveletek karakterekkel[1]

Az **IsLetter** és **IsDigit** metódusok meghatározzák, hogy a paraméterként megadott karakter betű vagy szám-e. Az **IsLower** és **IsUpper** metódusokkal lekérdezhető, hogy az adott karakter kisbetű, illetve nagybetű-e. A **ToLower** és **ToUpper** metódusok rendre az adott karakter kisbetűs, illetve nagybetűs változatát adják meg.

```
bool isLetter = char.IsLetter('a');    // true
bool isDigit = char.IsDigit('1');      // true
bool isUpper = char.IsUpper('a');      // false
char toLower = char.ToLower('A');      // 'a'
```

Egy `char` típusú érték `int` típusúvá konvertálható, ekkor az adatott karakter *karakterkódját* kapjuk. A karakterkód fordított irányú konverziója egy karaktert eredményez. A karakter mögöttes szám értékéből adódóan egy karaktert lehet *növelni* vagy *csökkenteni* is, vagyis érelmezettek rajtuk a ++ és -- operátorok, ilyenkor valójában a karakterkódjukat módosítjuk.

```
int codeOfA = (int)'A';     // 65
char letterZ = (char)90;    // 'Z'
```

Egy `string` típusú változó egyes karaktereire az indexelő operátorral hivatkozhatunk, hasonlóan a tömböknél megismert módon, ilyenkor a hivatkozott elem egy `char` típusú érték lesz. A karakterlánc hossza a `Length` tulajdonsággal kérdezhető le.

[1] Bővebben a hivatalos dokumentációban: https://learn.microsoft.com/en-us/dotnet/api/system.char?view=net-6.0

```
string quote = "We should have shotguns for this kind of deal.";
char firstChar = quote[0];                  // 'W'
char lastChar = quote[quote.Length-1];      // '.'
```

Műveletek karakterláncokkal[2]

A `string` egy úgynevezett *immutable* (nem módosítható) típus, vagyis tartalmát létrehozás után nem tudjuk elemenként megváltoztatni. Elemenkénti módosítást tudunk elérni, ha a `string`-et előbb karakter tömbbé alakítjuk (**ToCharArray**), a tömb elemein módosítunk, majd visszaalakítjuk a tömböt `string`-gé.

```
string quote = "They speak English in What?";
char[] quoteAsArray = quote.ToCharArray();
quoteAsArray[3] = 'Y';
string modQuote = new string(quoteAsArray);    // "TheY speak English in What?"
```

A **Join** metódussal egy karakterláncot állíthatunk elő valamilyen gyűjtemény (pl. tömb) elemeinek összefűzésével. Az elemek közötti elválasztójel paraméterként adható meg.

```
string[] words = { "Jack", "Rabbit", "Slim's" };
string concatenated = String.Join('-', words);    // "Jack-Rabbit-Slim's"
```

Karakterláncokkal végzett műveletek során hasznosak lehetnek az alábbi metódusok. A metódusok egy része *módosít* a karakterlánc tartalmán, de ilyenkor a módosított változatot visszatérési értékként szolgáltatja, vagyis nem közvetlenül az eredeti karakterláncon végzi el (hiszen az nem módosítható).

Két karakterlánc összehasonlítható *lexikografikusan*, vagyis betűrend szerint a **Compare** függvénnyel: az eredmény 0, ha a két karakterlánc megegyezik, -1 ha az első betűrendben megelőzi a másodikat, és 1 ha az első betűrendben a második után következik.

```
int result = String.Compare("Marsellus", "Jules");    // 1
```

Hasznos lehet egy `string` tartalmának ellenőrzésekor, hogy az üres vagy `null` értékű-e, erre szolgálnak az **IsNullOrEmpty** és **IsNullOrWhiteSpace** metódusok.

```
string nothing = "";
string something = "    ";
bool isEmpty = String.IsNullOrEmpty(nothing);                // true
bool isEmptyToo = String.IsNullOrWhiteSpace(something);      // true
```

A **Contains** segítségével lekérdezhetjük egy karakter vagy karakterlánc előfordulását (van vagy nincs). Az **IndexOf** és **LastIndexOf** metódusok rendre egy karakter vagy karakterlánc első és utolsó előfordulásának indexét ajdák meg (ha nincs előfordulás, akkor a visszaadott érték -1 lesz).

[2] Bővebben a hivatalos dokumentációban: https://learn.microsoft.com/en-us/dotnet/api/system.string?view=net-6.0

```
string quote = "Oh man, I shot Marvin in the face.";
bool contains = quote.Contains('M');      // true
int firstIdx = quote.IndexOf('I');        // 8
int lastIdx = quote.LastIndexOf("in");    // 22
int noIdx = quote.IndexOf("Pumpkin");     // -1
```

A **ToLower** és **ToUpper** kisbetűssé, illetve nagybetűssé alakítják a szöveget. Az **Insert** és **Remove** metódusokkal index alapján szöveget szúrhatunk a karakterláncba, illetve eltávolíthatjuk egy részét. A **Replace** metódus az első paraméterként megadott karaktereket a második paraméterként megadott karakterekre cseréli. A **Trim** a karakterlánc elején és végén található üres karaktereket (pl. whitespace, tabulátor) távolítja el. A **Substring** az eredeti karakterlánc rész-karakterláncát állítja elő, a megadott kezdőindextől adott számú karakterrel. A **Split** egy tömböt eredményez, amelyet az eredeti karakterlánc feldarabolásával kapunk a megadott a karaktert vagy karakterláncot használva elválasztóként (amely nem jelenik meg az eredményében).

```
string quote = "Big Kahuna Burger";
string lower = quote.ToLower();                    // big kahuna burger
string inserted = quote.Insert(11, "Cheese ");     //Big Kahuna Cheese Burger
string removed = quote.Remove(quote.IndexOf("Kahuna"), 7);    // Big Burger
string replaced = quote.Replace('B', 'K');         // Kig Kahuna Kurger
string substring = quote.Substring(4, 6);          // Kahuna
string[] splitted = quote.Split(' ');              // { "Big", "Kahuna", "Burger" }
```

Formázott karakterláncok

Formázott karakterláncokat állíthatunk elő többek között karakterlánc *interpolációval*. Ilyenkor a karakterláncot nyitó idézőjel elé a $ jelet téve, a karakterláncban kapcsos zárójelek közé helyezett változónevek (illetve kifejezések vagy metódushívások) azok értékeivel helyettesülnek. Opcionálisan formázást is beállíthatunk (ezt nem részletezünk, csak egy példát mutatunk rá).

```
double normalTime = 13.0;
int mrWolfeTime = 10;
string expression = $"That's {normalTime:F3} minutes away. I'll be there in
   {mrWolfeTime}."
// "That's 13.000 minutes away. I'll be there in 10."
```

Feladatok

1 Írjunk programot, amely meghatározza egy szövegben a betűk, a számjegyek és a magánhangzók számát kategóriánként.

2 Írjunk programot, amely meghatározza, hogy egy szöveg *palindrom* szöveg-e, vagyis hogy előre olvasva ugyanazt adja-e, mint visszafelé.

Példa

A *„Géza kék az ég."*, az *„Indul a görög aludni."* vagy a *„Rád rohan a hordár."* szövegek palindrom szövegek.

3 Írjunk programot, amely sztenderd formátumra hozza egy jármű rendszámát. A jelenlegi sztenderd formátum kétszer két nagybetű, közöttük egy szóköz, majd egy kötőjel és három szám.

Példa

A következő rendszámok mindegyike sztenderd formátumban `AA BC-123`:

- `aabc 123`
- `a a BC123`
- `a a B c 1 2 3`
- `AABc-123`

4 Írjunk programot, amely képes adott számú, különböző, az előző feladatban megadott formátumú véletlen rendszámot generálni.

5 Írjunk programot, amely helyesség szempontjából ellenőrzi a felhasználó által megadott email címet. Az email cím helyes, ha az alábbiak mindegyike teljesül.

a) pontosan egy `@` karaktert tartalmaz
b) tartalmaz legalább egy betű karaktert a `@` előtt
c) tartalmaz legalább egy `.` karaktert a `@` után
d) a `@` és az utolsó `.` karakter között kell legyen legalább egy betű vagy szám karakter
e) ha tartalmaz `.` karaktert a `@` előtt is, akkor a `.` előtt és után is betű vagy szám karakter kell álljon
f) az utolsó `.` karakter után legalább két betűt kell tartalmazzon

6 Készítsünk programot Neptun-kódhoz hasonló szöveg generálására. Egy Neptun-kód pontosan hat karakterből áll, véletlenszerű betűkből és számokból, de az első karakter mindig betű. Számoljuk meg, hogy hányadik véletlenszerűen generált Neptun-kód egyezik meg a saját azonosítónkkal.

7 Írjunk programot, amely egy adott szöveget *SpongeCase* formátumúra alakít. Az átalakítás során a szöveg karakterei véletlenszerűen legyenek kis- és nagybetűk.

Példa

Bemenet: `Well, a Big Mac's a Big Mac, but they call it le Big-Mac.`
Kimenet: `WeLl, A BiG MaC'S A BiG MaC, bUt tHeY CaLl iT Le bIg-mAc.`

8 Írjunk programot, amely egyetlen formázott s karakterlánc tartalmát egy táblázatba (kétdimenziós tömbbe) rendezi. Az s karakterláncban az egyes sorokat sortörés (`'\n'`) karakterek, az egyes oszlopokat pontosvesszők (`';'`) választják el.

Példa

A `"Vincent;Vega;Vince\nMarsellus;Wallace;Big Man\nWinston;Wolf;The Wolf"` karakterlánc feldolgozása után az alábbi kétdimenziós tömböt kapjuk.

Vincent	Vega	Vince
Marsellus	Wallace	Big Man
Winston	Wolf	The Wolf

9 Készítsünk programot, amely egy szöveg formájában adott zárójel-sorozatról eldönti, hogy az szabályos-e. Egy ilyen sorozatot szabályosnak nevezünk, ha benne a zárójelek párosíthatóak úgy, hogy minden párban legyen egy összetartozó kezdő- és egy végzárójel.

Példa

A `[()][()()]()` például szabályos zárójelezés, de például a `[(])` nem az, mivel a kerek nyitó zárójel végzárójel párja szögletes zárójelen kívül található.

10 Készítsünk egyszerű szövegszerkesztő programot az alábbiak szerint. A szöveget tároljuk egy kétdimenziós karaktertömbben, amelynek mérete előre megadott (pl. 50 karakter széles és 20 karakter magas). A program frissítse és jelenítse meg a tömb tartalmát minden billentyűlenyomás után. Legyen lehetőség a kurzorbillentyűk segítségével navigálni a szövegben, továbbá szöveg beírására, amely ilyenkor felülírja a korábban ott lévő szöveget.

11 A `Base64` kódolás egy 64 karakteres ábécén (kódtáblán) alapuló kódolási forma, melynek használatával tetszőleges adatot szöveges formátumúvá alakíthatunk. Írjunk programot, amel egy egyszerűsített, tetszőleges (Unicode) karakterekből álló szöveget képes 64 előre megadott karakter segítségével kódolni. A szótár álljon az alábbi karakterekből:

- az angol ábécé nagybetűi (`A-Z`)
- az angol ábécé kisbetűi (`a-z`)
- számjegyek (`0-9`)
- további két speciális karakter: `+` és `/`

A kódolás során a kódolandó adathalmazt bontsuk fel 3 bájtos egységekre, az így kapott 24 bitet pedig daraboljuk 6 bites szegmensekre. Egy-egy 6 bites szegmens értéke a kódtábla indexeként használható, vagyis minden három kódolatlan karakter négy kódolttá alakul.