

OOP II.

A C# nyelv alapelemei I.

Hello, C# World
Szintaktikai alapszabályok és konvenciók
Alaptípusok (1. rész)
Változók, kifejezések
Operátorok és precedenciájuk (1. rész)
Utasítások: üres utasítás, if, switch, while, do...while, break

Készítette:

Miklós Árpád

Dr. Kotsis Domokos

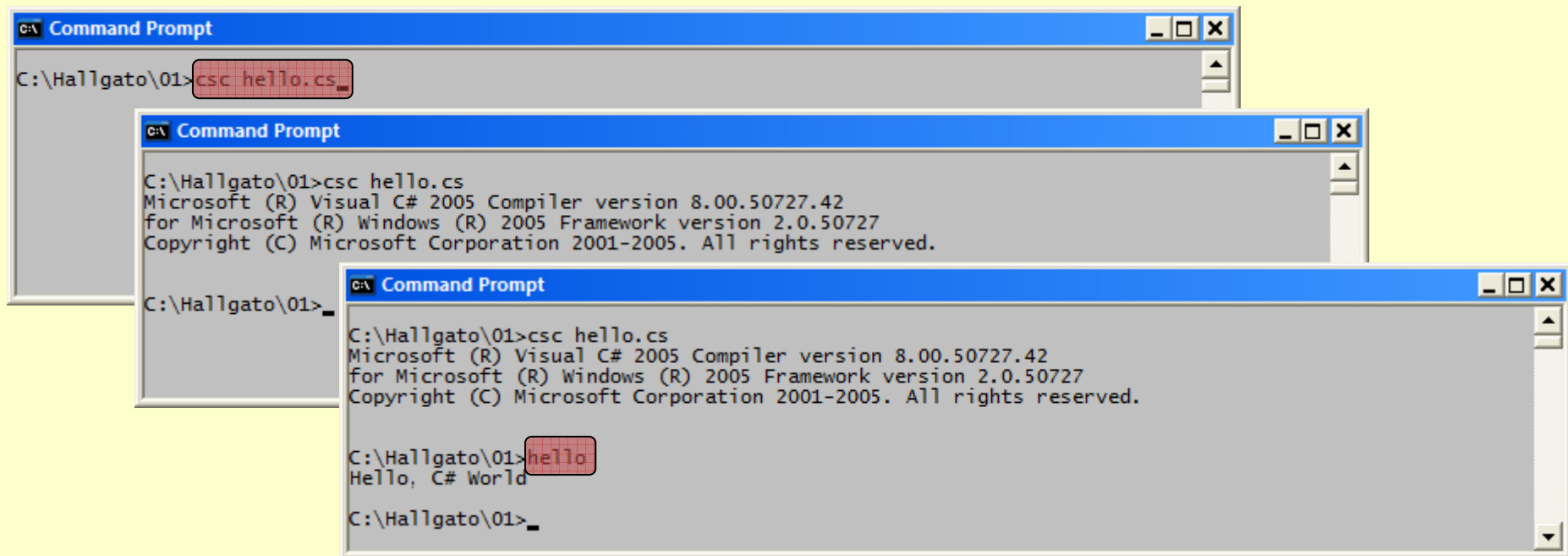
Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

Hello, C# World

```
// Első programunk C# nyelven
class ElsőProgram
{
    static void Main()
    {
        System.Console.WriteLine("Hello, C# World");
        System.Console.ReadLine();
    }
}
```



The image shows three overlapping Command Prompt windows. The top window shows the command `csc hello.cs` being entered. The middle window shows the output of the compilation: `C:\Hallgato\01>csc hello.cs`, `Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42`, `for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727`, and `Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.`. The bottom window shows the command `hello` being entered, followed by the output `Hello, C# World`.

Néhány szintaktikai alapszabály

```
// Első programunk C# nyelven
```

Egysoros megjegyzés: // karakterek után
Többsoros megjegyzés: /* és */ karakterpárok között

```
class ElsőProgram
```

Minden azonosító (név) Unicode formátumú, azaz ékezetes karakterek is használhatók e célra

```
{
```

```
static void Main()
```

Minden futtatható programnak rendelkeznie kell egy „Main” nevű függvénnyel (amely a program egy tetszőleges osztályának statikus, visszatérési érték nélküli, illetve egy egész számmal, mint eredménykóddal visszatérő metódusa)

```
{
```

```
System.Console.WriteLine("Hello, C# World");
```

Az utasítások végén pontosvessző áll

```
System.Console.ReadLine();
```

A C# nyelvben a kis- és nagybetűk jelentése különbözik (tehát pl. „writeline” ≠ „WriteLine”)

Kapcsos zárójelekkel több utasítás is összefogható egyetlen összetett utasítássá („blokk”) (a blokkok egyúttal a hatóköröket is kijelölik)

```
}
```

```
}
```

Általános szintaktikai konvenciók

- Kis- és nagybetűs elnevezések használata
 - Azonos hatókörben* elérhető függvényneveknél, illetve paraméterneveknél kerüljük a kizárólag kis- és nagybetű alapján történő megkülönböztetést

~~void SzépNevűFüggvény()~~

~~void Szépnevűfüggvény()~~

~~void szépnevűfüggvény()~~

~~void HasonlóParaméterek(string a, string A)~~

- Rövidítések használata
 - Elnevezések meghatározásánál önkényesen ne rövidítsünk le szavakat (pl. „ElsőAblak” helyett „ElsAbl”)
 - Nagy valószínűséggel nem közismert betűszavakat ne használjunk
 - Hosszú, többtagú nevek helyett használjunk közismert betűszót, ha létezik (pl. „OnlineTransactionProcessing” helyett „OLTP”)

Általános szintaktikai konvenciók

- Foglalt azonosítók a C# nyelvben (C# Specification 3.0, 2006. május)

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	var	virtual
void	volatile	while		

- Egyéb, korlátozottan használható azonosítók

get	set
-----	-----

Adattípusok

- Beépített alaptípusok
 - Ezek a C# nyelv külön definíció nélkül, alapértelmezésben is rendelkezésre álló adattípusai
- Saját típusok
 - A programozók által definiált, az alaptípusok valamelyikére épülő összetett típusok tartoznak ide
 - A futtatókörnyezethez tartozó típuskönyvtárak számos saját típust definiálnak, amelyek szintén azonnal felhasználhatók
 - Később részletesebben tárgyaljuk
 - A teljes típusrendszer összefoglalását a következő gyakorlaton tárgyaljuk
 - A saját típusok létrehozását több részben, a következő gyakorlattól kezdve tárgyaljuk

A C# beépített alaptípusai (1)

- Egész számok (1)

Név	Leírás	Értéktartomány
<code>int</code>	32 bites előjeles egész	-2 147 483 648 : 2 147 483 647
<code>uint</code>	32 bites előjel nélküli egész	0 : 4 294 967 295

- Logikai típusok

Név	Leírás	Értéktartomány
<code>bool</code>	Logikai adattípus	true vagy false (igaz vagy hamis)

Egész számok gépi ábrázolása

- Bináris (kettes számrendszerbeli) számábrázolás
 - Tárolásuk 0 és 1 értékű számjegyek (bitek) sorozataként történik
 - Terjedelmi okokból gyakran 16-os számrendszerben hivatkozunk rájuk
 - Ez a „hexadecimális kód” (például: A3D7 értéke tízes számrendszerben 41943)
- Helyfoglalás: 8/16/32/64 bit (azaz 1/2/4/8 bájt)
 - Az elfoglalt bájtok száma mindig 2 valamelyik hatványa
- Pozitív és negatív számok kezelése
 - Előjel nélküli ábrázolás
 - A legkisebb érték 0, a legnagyobb érték 2^x-1 , ahol x az elfoglalt bitek száma
 - Előjeles ábrázolás
 - Kettes komplement kód
 - Célja a műveletvégzés egyszerűsítése (ennél a megoldásnál ui. nem kell tekintetbe venni az előjelet sem összeadásnál, sem kivonásnál, az ábrázolásból következően automatikusan a helyes eredmény adódik)
 - A legkisebb érték $-(2^{x-1})$, a legnagyobb érték $2^{x-1}-1$, ahol x az elfoglalt bitek száma
- Abszolút (teljes) pontosságú számábrázolás
- Viszonylag kis ábrázolható számtartomány

Előjeles egész számábrázolás I.

Egyszerű szabály a 2-es komplement képzésére:
A bináris szám komplementerét képezzük
(1 helyett 0, 0 helyett 1), majd 1-et hozzáadunk.
Ami kicsordul, elvész.

Előny csak összeadás és komplement képzés kell
az additív műveletekhez.

Pé. 4 biten $1=0001$ komplemente $-1=1111$.

Előjeles egész számbábrázolás II.

(4 biten)

	0	0000		
1	0001		-1	1111
2	0010		-2	1110
3	0011		-3	1101
4	0100		-4	1100
5	0101		-5	1011
6	0110		-6	1010
7	0111		-7	1001
			-8	1000

Figyelem! Negatív számból eggyel nagyobb helyi értékű ábrázolható, mint pozitívból!

Előjeles egész összeadás I.

(azonos, pozitív előjel)

1	0001
2	0010
---	-----
3	0011

Előjeles egész összeadás II.

(azonos, negatív előjel)

-1	1111
-2	1110
---	-----
-3	11101

Túlcsondul,
elvész

Előjeles egész összeadás III.

(különböző előjel I.)

1	0001
-3	1101
---	-----
-2	1110

Előjeles egész összeadás IV.

(különböző előjel II.)

3	0011
-1	1111
---	-----
2	1 0010

Túlcsondul,
elvész

Előjeles egész kivonás

(a kivonandó komplementjét vesszük)

0010 komplemente 1110

1	0001
- 2	1110
-----	-----
-1	1111

Vigyázat I.

(pozitív túlcsordulás)

7	0111
1	0001
---	-----
-8?	1000

**A +8 már
nem ábrázolható,
hibás eredmény!**

Vigyázat II.

(negatív túlcsordulás)

-8	1000
-1	1111
---	-----
7?	10111

A -9 már
nem ábrázolható,
hibás eredmény!

A C# beépített alaptípusai (1)

- Egész számok (1)

Név	Leírás	Értéktartomány
<code>int</code>	32 bites előjeles egész	-2 147 483 648 : 2 147 483 647
<code>uint</code>	32 bites előjel nélküli egész	0 : 4 294 967 295

- Logikai típusok

Név	Leírás	Értéktartomány
<code>bool</code>	Logikai adattípus	true vagy false (igaz vagy hamis)

Logikai típusok gépi ábrázolása

- A logikai típusok kétértékűek
 - Értékeiket „igaz” („true”) és „hamis” („false”) kifejezéssel jelöljük
- Helyfoglalás: általában 1/8/16/32/64 bit
 - Általában a csupa 0 értékű bit jelenti a „hamis”, a csupa 1 értékű bit az „igaz” értéket
 - 16 bites ábrázolás esetén:
 - „hamis” („false”) érték = 0000 0000 0000 0000 (számként kiolvasva 0)
 - „igaz” („true”) érték = 1111 1111 1111 1111 (számként kiolvasva -1)
 - Teljesítményekből szokás 1 bitnél többet felhasználni a mindössze két érték ábrázolására

A C# beépített alaptípusai (1)

- Karakterek és karaktersorozatok

Név	Leírás	Értéktartomány
<code>char</code>	Egyetlen Unicode karakter	16 bites (UTF-16) kódtartomány
<code>string</code>	Unicode karaktersorozat	Legfeljebb 2³² db Unicode karakter

Speciális karakterek:

Jelölés	Karakter
<code>\0</code>	Null karakter
<code>\a</code>	Sípszó
<code>\b</code>	Visszatörlés
<code>\f</code>	Lapdobás
<code>\n</code>	Soremelés
<code>\r</code>	Kocsi vissza
<code>\t</code>	Vízszintes tabulátor

Jelölés	Karakter
<code>\v</code>	Függőleges tabulátor
<code>\x....</code>	Hexadecimális kód
<code>\u....</code>	Unicode karakter
<code>\U....</code>	Unicode karakter
<code>\'</code>	Aposztróf
<code>\"</code>	Idézőjel
<code>\\</code>	Backslash

Karakterek gépi ábrázolása

- Helyfoglalás: 8/16/32 bit (azaz 1/2/4 bájt)
- Kódolt ábrázolás
 - Minden karakternek egy megállapodás szerinti szám (kód) felel meg
 - Az ábrázolható karakterek maximális száma a helyfoglaláshoz kötődik
 - 8 biten 256, 16 biten 65 536, 32 biten 4 294 967 296 különböző karakter tárolható
 - Kódolási módszerek
 - ASCII / ANSI
 - 7 / 8 bites ábrázolás (az ANSI szabvány a „felső” 128 karakterre különböző kódlapokat kínál)
 - EBCDIC
 - 8 bites ábrázolás (az IBM fejlesztette ki lyukkártyás adattároláshoz)
 - Unicode

- UTF-32: 32 bites ábrázolás
(minden karakternek saját, egyedi, univerzális kódja van)

A	Ω	語	Ⅲ
00000041	000003A9	00008A9E	00010384

- **UTF-16: 16 bites ábrázolás**
(a 65 536 karaktert tartományokra osztja fel; egyes különleges karaktereket két kód azonosít)

A	Ω	語	Ⅲ
0041	03A9	8A9E	DC00 DB84

- UTF-8: 8 bites ábrázolás
(az ASCII kóddal le nem írható karakterek saját, változó hosszúságú kódolást kapnak)

A	Ω	語	Ⅲ
41	CE A9	E8 AA 9E	F0 90 8E 84

A C# beépített alaptípusai (1)

- Karakterek és karaktersorozatok

Név	Leírás	Értéktartomány
<code>char</code>	Egyetlen Unicode karakter	16 bites (UTF-16) kódtartomány
<code>string</code>	Unicode karaktersorozat	Legfeljebb 2³² db Unicode karakter

Speciális karakterek:

Jelölés	Karakter
<code>\0</code>	Null karakter
<code>\a</code>	Sípszó
<code>\b</code>	Visszatörlés
<code>\f</code>	Lapdobás
<code>\n</code>	Soremelés
<code>\r</code>	Kocsi vissza
<code>\t</code>	Vízszintes tabulátor

Jelölés	Karakter
<code>\v</code>	Függőleges tabulátor
<code>\x....</code>	Hexadecimális kód
<code>\u....</code>	Unicode karakter
<code>\U....</code>	Unicode karakter
<code>\'</code>	Aposztróf
<code>\"</code>	Idézőjel
<code>\\</code>	Backslash

Változók deklarációja és használata

```
int i;  
int j = -10;  
int x = 10, y = 20;  
uint y = 1234;
```

Mindkét változó egész típusú lesz és felveszi a megadott értéket

```
const int száz = 100;  
int összeg = 23 * (45 + 67);
```

A „száz” változó értéke később már nem módosítható (konstans)

Előre kiszámítható kifejezéseket is megadhatunk alapértékként

```
char c;  
char d = 'x';  
char UnicodePélda = '\\u0170'; // Ez az "Ű" karakter
```

A „d” változó karakter típusú lesz és felveszi a megadott értéket

```
static void Main()  
{  
    int i = 1;  
    System.Console.WriteLine(i);  
}
```

A változóknak az első felhasználás előtt kötelező értéket adni

Változók deklarációja és használata

```
string s;  
string jegy = "jeles";
```

A változó karaktersorozat típusú lesz és felveszi a megadott értéket

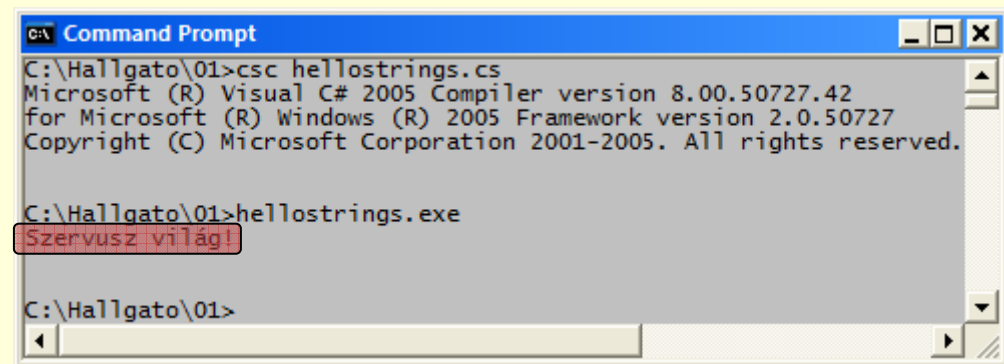
```
string ElérésiÚt = "C:\\Program Files\\";  
string SzóSzerintiElérésiÚt = @"C:\Program Files\";
```

```
string SzóSzerintiKaraktersorozatSortöréssel = @"Hová merült el  
szép szemed világa";
```

class MásodikProgram

```
{  
    static void Main()  
    {  
        string str1 = "Szervusz ";  
        string str2 = "világ!";  
        string str3 = str1 + str2;  
        System.Console.WriteLine(str3);  
        System.Console.ReadLine();  
    }  
}
```

Itt két karaktersorozatot kapcsolunk össze



```
C:\> Command Prompt  
C:\Hallgato\01>csc hellostrings.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\01>hellostrings.exe  
Szervusz világ!  
  
C:\Hallgato\01>
```

Feladat

Készítsünk programot, amely kiírja a konzolra a „Szervusz, hallgató!” szöveget!

```
class Szervusz
{
    static void Main()
    {
        System.Console.WriteLine("Szervusz, hallgató!");
        System.Console.ReadLine();
    }
}
```

Feladat

Készítsünk programot, amely a konzolról beolvas egy nevet, majd név szerint üdvözli az illetőt!

```
class Üdvözlet
{
    static void Main()
    {
        string név;
        System.Console.WriteLine("Hogy hívnak?");
        név = System.Console.ReadLine();
        System.Console.WriteLine("Szervusz, " + név + "!");
    }
}
```

Kifejezések

- A kifejezések („expression”) adatokat szolgáltató operandusokból és rajtuk valamilyen műveletet végző operátorokból állnak
 - Operandus: pl. bármely változó vagy konkrét megadott érték
 - Operátor: pl. + - / *
- A kifejezések egymásba is ágyazhatók
 - Egy kifejezés operandusa maga is lehet kifejezés
- Több operátor esetén ezek fontossági sorrendje (precedenciája) határozza meg a kiértékelés sorrendjét
 - Példa: az „ $x + y * z$ ” kifejezés kiértékelés szempontjából „ $x + (y * z)$ ”
 - A sorrend zárójelezéssel explicit módon is meghatározható
- Az operátorok jelentése általában módosítható
 - A művelet neve operátor-átdefiniálás („operator overloading”)
 - Később részletesebben tárgyaljuk

Operátorok és precedenciájuk (1)

- Aritmetikai operátorok

Operátor	Kifejezés	Precedencia	Jelentés
+	$+x$	2	Előjelképzés
	$x + y$	4	Összeadás vagy kombináció
-	$-x$	2	Előjelképzés
	$x - y$	4	Kivonás
*	$x * y$	3	Szorzás
/	x / y	3	Osztás
%	$x \% y$	3	Maradékképzés
++	$x++$	1	Növelés eggyel x kiértékelése után
	$++x$	2	Növelés eggyel x kiértékelése előtt
--	$x--$	1	Csökkentés eggyel x kiértékelése után
	$--x$	2	Csökkentés eggyel x kiértékelése előtt

Operátorok és precedenciájuk (1)

- Relációs (összehasonlító) operátorok

Operátor	Kifejezés	Precedencia	Jelentés
==	x == y	7	Egyenlő
!=	x != y	7	Nem egyenlő
<	x < y	6	Kisebb
>	x > y	6	Nagyobb
<=	x <= y	6	Kisebb vagy egyenlő
>=	x >= y	6	Nagyobb vagy egyenlő

Operátorok és precedenciájuk (1)

- Bináris logikai (bitenkénti műveletvégző) operátorok

Operátor	Kifejezés	Precedencia	Jelentés
\sim	$\sim x$	2	Bitenkénti NEM művelet
$\&$	$x \& y$	8	Bitenkénti ÉS művelet
\wedge	$x \wedge y$	9	Bitenkénti KVAGY (kizáró VAGY) művelet
$ $	$x y$	10	Bitenkénti VAGY művelet
\ll	$x \ll y$	5	Eltolás balra (x eltolása y helyiértékkel)
\gg	$x \gg y$	5	Eltolás jobbra (x eltolása y helyiértékkel)

Operátorok és precedenciájuk (1)

- Logikai (feltételvizsgáló) operátorok

Operátor	Kifejezés	Precedencia	Jelentés
!	!x	2	A kifejezés értéke x ellentettje
&&	x && y	11	A kifejezés akkor igaz, ha x és y is igaz
	x y	12	A kifejezés akkor igaz, ha x vagy y igaz

Operátorok és precedenciájuk (1)

- Értékadó operátorok

Operátor	Kifejezés	Precedencia	Értékadás típusa
=	$x = y$	14	Egyszerű (x értéke legyen egyenlő y -nal)
+=	$x += y$	14	Összeadással ($x = x + y$)
-=	$x -= y$	14	Kivonással ($x = x - y$)
*=	$x *= y$	14	Szorzással ($x = x * y$)
/=	$x /= y$	14	Osztással ($x = x / y$)
%=	$x \% = y$	14	Maradékképzéssel ($x = x \% y$)
&=	$x \& = y$	14	Bitenkénti ÉS művelettel ($x = x \& y$)
^=	$x \wedge = y$	14	Bitenkénti KVAGY művelettel ($x = x \wedge y$)
=	$x = y$	14	Bitenkénti VAGY művelettel ($x = x y$)
<<=	$x \ll = y$	14	Bitenkénti eltolással balra ($x = x \ll y$)
>>=	$x \gg = y$	14	Bitenkénti eltolással jobbra ($x = x \gg y$)

Utasítások

- Egy program alapvetően (alacsony absztrakciós szinten szemlélve) utasítások sorozatából áll
- Egyszerű utasítások („statement”)
 - Az egyszerű utasítások lehetnek deklarációk, kifejezések vagy előre definiált (beépített) utasítástípusok
 - Az egyszerű utasítások előtt szerepelhet címke is („label”)
 - Címke megadási módja: címkeazonosító:
 - Az egyszerű utasításokat „ ; ” karakter zárja le
- Összetett utasítások („compound statement”)
 - Több utasítás sorozata összefogható egy összetett utasítássá
 - Ehhez az összefogandó egyszerű utasítások sorozatát „ { } ” karakterek közé írjuk
 - Összetett utasítások is összefoghatók nagyobb összetett utasításokká
 - Az összetett utasítások végén nem szerepel „ ; ” karakter
 - Az összetett utasítás másik neve: „blokk” vagy „kódblokk”

Túlcsordulás I.

Készítsünk programot, mely 1 byte hosszúságú, 255 értékű előjel nélküli egész szám változóhoz 1-t hozzáad. Mi lesz az eredmény?

(Az egy byte hosszúságú előjel nélküli egész típus neve a C#-ban *byte*.

Figyeljünk arra, hogy a C# az egész kifejezéseket integerként kezeli, így az értékadás módja:

változó = (byte)(kifejezés);

az u.n. casting.)

a=255, b=a+1=0!!!

Túlcsordulás II.

```
class Bytetúl
{
    static void Main()
    {
        byte a, b;
        a = 255;
        b = (byte)(a+1);
        System.Console.WriteLine("a =" + a + ", b=a+1=" + b);
        System.Console.ReadLine();
    }
}
```

Túlcsordulás III.

Készítsünk programot, mely 1 byte hosszúságú, -127 értékű előjeles egész szám változóból 1-et, majd 2-t kivon.
Mi lesz az eredmény?

(Az egy byte hosszúságú előjeles egész típus neve a C#-ban *sbyte*.)

Figyeljünk arra, hogy a C# az egész kifejezéseket integerként kezeli, így az értékadás módja:

változó = (sbyte)(kifejezés);

az u.n. casting.)

$a=-127, b=a-1=-128, c=a-2=127!!!$

Túlcsordulás IV.

```
class Sbytetúl
{
  static void Main()
  {
    sbyte a, b, c;
    a = -127;
    b = (sbyte)(a-1);
    c = (sbyte)(a-2);
    System.Console.WriteLine("a =" + a + ", b=a-1=" + b + ", c=a-2=" + c);
    System.Console.ReadLine();
  }
}
```

Túlcsoordulás V.

De

$a = -127$, $b = a - 1 = -128$, $c = a - 2 = -129!!!$

ha

```
class Sbytetul1
{
    static void Main()
    {
        sbyte a;
        a = -127;
        System.Console.WriteLine("a =" + a + ", a-1=" + (a-1) + ", a-2=" + (a-2));
        System.Console.ReadLine();
    }
}
```

Az üres utasítás

;

- Szintaktikai szerepe van
 - Egyszerű utasítások lezárására szolgál
 - Olyan helyeken használjuk, ahol nincs teendő, de a C# nyelv megköveteli, hogy ott utasítás szerepeljen

Az if utasítás

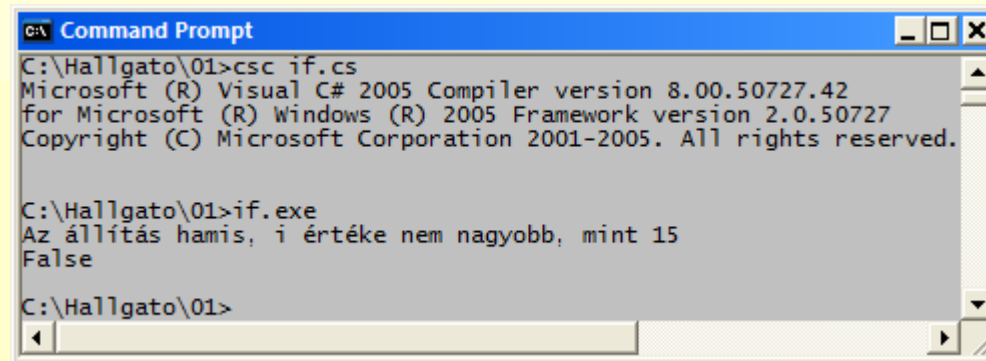
if (*feltétel*)
 utasítás
[else
 utasítás]

- Az if utasítások egymásba is ágyazhatók
 - Minden feltételhez kapcsolódhat else ág, de jelenléte nem kötelező
 - Minden else ág az utolsó (őt közvetlenül megelőző) if utasításra vonatkozik
- Egyenlőségvizsgálat az „==” (és nem az „=”) operátorral

Az if utasítás (példa)

```
int i = 12;  
if (i == 10)  
    System.Console.WriteLine("Ez bizony pontosan 10");
```

```
bool állítás;  
if (i > 15)  
{  
    állítás = true;  
    System.Console.WriteLine("Az állítás igaz, i értéke nagyobb, mint 15");  
}  
else  
{  
    állítás = false;  
    System.Console.WriteLine("Az állítás hamis, i értéke nem nagyobb, mint 15");  
}  
System.Console.WriteLine(állítás);
```



```
C:\ Command Prompt  
C:\Hallgato\01>csc if.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\01>if.exe  
Az állítás hamis, i értéke nem nagyobb, mint 15  
False  
  
C:\Hallgato\01>
```

Feladat

Készítsük el az előző feladatnak azt a változatát, melyben az `i` változó értéke input adat!

A beolvasott `s` string-et egész számmá kell konvertálni. Ez pl. az `i=int.Parse(s)` kifejezéssel lehetséges.

```
int i;  
i=int.Parse(System.Console.ReadLine());
```

```
if (i == 10)
```

```
...
```

A while utasítás

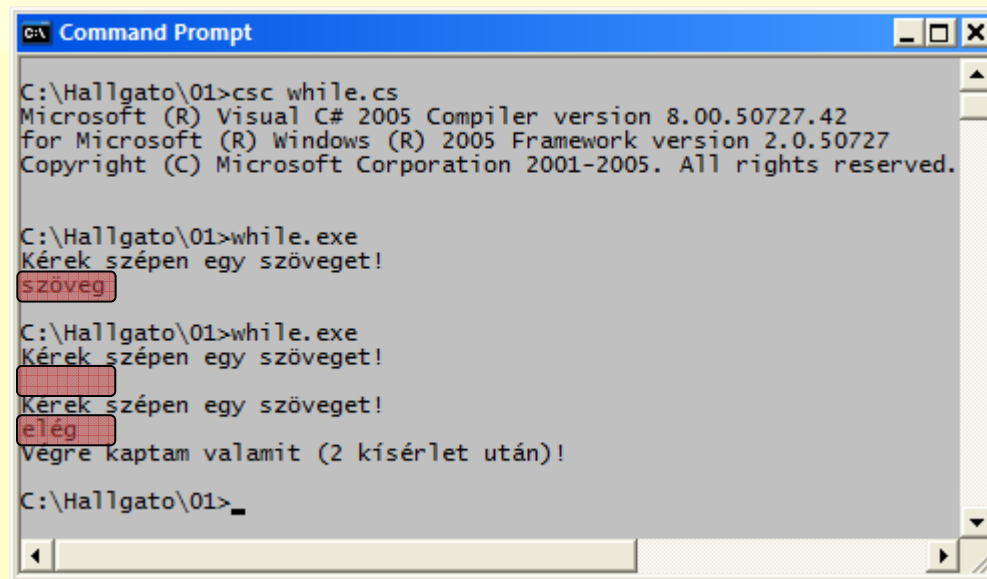
*while (feltétel)
utasítás*

- Szokványos elnevezése: előtesztelő ciklus („loop”)
- Ha a feltétel mindig teljesül, végtelen ciklusról beszélünk („infinite loop”)
 - A végtelen ciklus gyakori programozói hiba forrása
- Akkor használjuk, ha valamely utasítást kizárólag bizonyos feltétel fennállása esetén kell végrehajtani

A while utasítás (példa)

```
string s = " ";  
int számláló = 0;
```

```
while (s == " ")  
{  
    System.Console.WriteLine("Kérek szépen egy szöveget!");  
    s = System.Console.ReadLine();  
    számláló++;  
    if ((s != " ") && (számláló > 1))  
        System.Console.WriteLine("Végre kaptam valamit (" + számláló + " kísérlet után)!");  
}
```



```
C:\Hallgato\01>csc while.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\01>while.exe  
Kérek szépen egy szöveget!  
szöveg  
  
C:\Hallgato\01>while.exe  
Kérek szépen egy szöveget!  
  
Kérek szépen egy szöveget!  
elég  
Végre kaptam valamit (2 kísérlet után)!  
  
C:\Hallgato\01>
```

A do...while utasítás

do

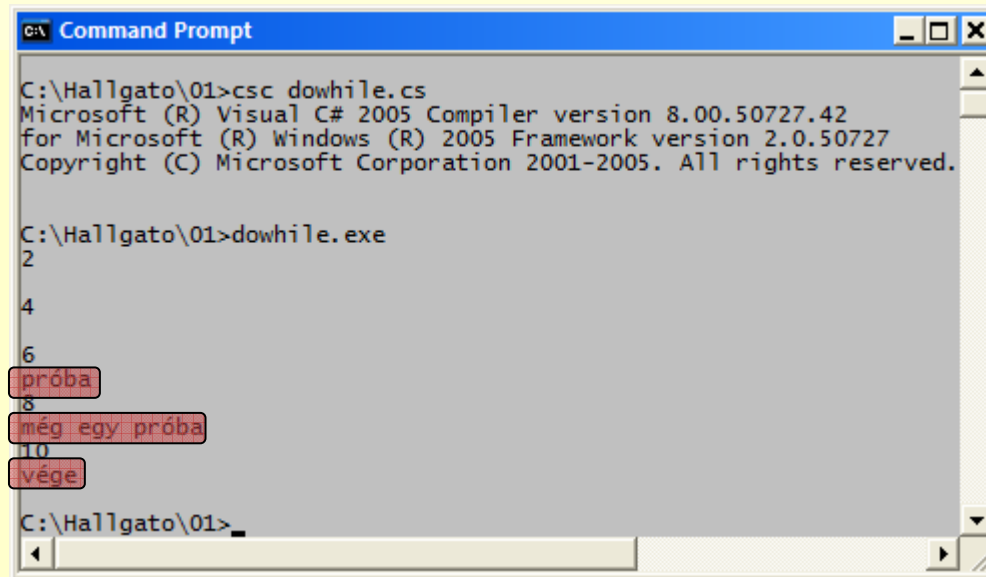
utasítás

while (*feltétel*)

- Szokványos elnevezése: hátultesztelő ciklus
- Ha a feltétel mindig teljesül, végtelen ciklusról beszélünk
- Akkor használjuk, ha valamely utasítást legalább egyszer biztosan végre kell hajtani, majd ezek után kizárólag bizonyos feltétel fennállása esetén kell ismételten végrehajtani őket

A do...while utasítás (példa)

```
string válasz;  
int i = 0;  
  
do  
{  
    i += 2;  
    System.Console.WriteLine(i);  
    válasz = System.Console.ReadLine();  
}  
while (válasz != "vége");
```



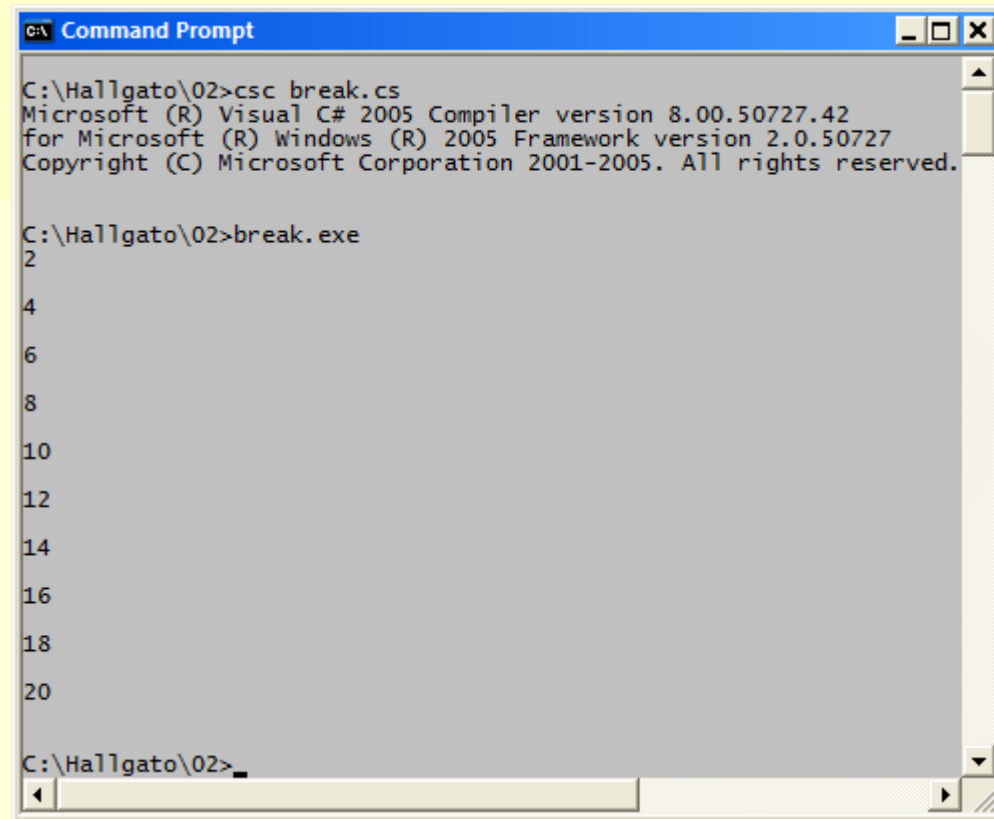
```
C:\Hallgato\01>csc dowhile.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\01>dowhile.exe  
2  
4  
6  
próba  
8  
még egy próba  
10  
vége  
  
C:\Hallgato\01>
```

A break utasítás

`break ;`

- A végrehajtás megszakítása, folytatás a következő utasítással
 - Segítségével kiléphetünk az aktuális switch, while, do...while, for, illetve foreach utasítás belsejéből

```
string válasz;  
int i = 0;  
  
do  
{  
    i += 2;  
    if (i > 20)  
        break;  
    System.Console.WriteLine(i);  
    válasz = System.Console.ReadLine();  
}  
while (válasz != "vége");
```



```
C:\Hallgato\02>csc break.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\02>break.exe  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20  
C:\Hallgato\02>
```

A switch utasítás

switch (*kifejezés*)

{

case címkekonstans1:

utasítássorozat

 break;

 ...

case címkekonstansN:

utasítássorozat

 break;

[default:

utasítássorozat

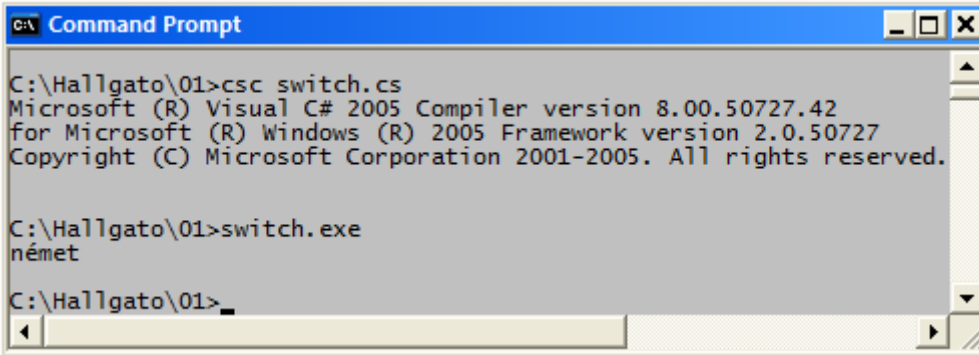
 break;]

}

- Minden címkekonstans értéke egyszer szerepelhet
- A címkekonstansok sorrendje tetszőleges
 - Ez a default ágra is vonatkozik

A switch utasítás (példa)

```
string nyelv;  
string országkód = "de";  
  
switch (országkód)  
{  
    case "hu":  
        nyelv = "magyar";  
        break;  
    case "en":  
        nyelv = "angol";  
        break;  
    case "ch":  
    case "de":  
        nyelv = "német";  
        break;  
    default:  
        nyelv = "ismeretlen nyelv";  
        break;  
}  
System.Console.WriteLine(nyelv);
```



```
C:\ Command Prompt  
C:\Hallgato\01>csc switch.cs  
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42  
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727  
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.  
  
C:\Hallgato\01>switch.exe  
német  
  
C:\Hallgato\01>
```

Feladat

Készítsünk programot, mely beolvas a billentyűzetről két számot és egy műveleti jelet, majd kiírja a két számmal elvégzett művelet eredményét. A műveleti jelek megkülönböztetéséhez használjunk többágú (switch, case) elágaztatást.

Gyakorló feladat

A korábban elkészített algoritmus struktogramja alapján készítsünk másodfokú egyenletet megoldó programot!

Segítség.

a **System.Math.Sqrt(x)**; függvény az **x** nem negatív szám négyzetgyökét adja **double** típusban. Ebből **float**-ot u-n. **cast**-olással készíthetünk:

```
float x,z;
```

```
double y;
```

```
.
```

```
.
```

```
y = System.Math.Sqrt(x);
```

```
z = (float)y;
```