1 Program fogalma, C# program felépítése

A program előre megadott utasítások logikus sorozata, amely közli a számítógéppel, hogy mit tegyen a betáplált adatokkal az adott feladat elvégzése érdekében.

Program = algoritmus + adatszerkezet

Az algoritmus fogalmát már megismertük, de miért van szükségünk a programozás során adatszerkezetre? Az algoritmus tulajdonsága az általános érvényűség, ezt csak változók segítségével tudjuk megvalósítani a programozás során. A programban használt változók halmazát nevezzük **adatszerkezetnek**.

A program adatszerkezetét táblázatos formában szoktuk megadni, mely táblázat a következő oszlopokat tartalmazza:

- Funkció (Mire használjuk a változót?)
- Azonosító (Milyen névvel hivatkozunk a változóra?)
- Típus (Milyen jellegű adatokat vehet fel a változó? pl.: egész, valós, szöveges, logikai, stb.)
- Jelleg (Input/Output/Work)

A modern programozási nyelvekben (objektum orientált nyelvek) a speciális utasításokat **osztályok** segítségével érhetjük el. pl.: A képernyő törlése a `Console` osztály `Clear()` utasításával érhető el: `Console.Clear();` végrehajtásával. A példából jól látható, hogy az osztály azonosítója és az utasítás azonosítója közé pontot kell tenni, az utasítást a pontosvessző zárja.

Az objektum orientált nyelvekben az utasításokat (alprogramokat (függvények és eljárások)) összefoglaló néven **metódusoknak** hívjuk.

Az osztályokat **névterek** segítségével rendszerezzük és tároljuk. A `Console` osztály a `System` névtér része.

A C# program felépítése:

```
using System; //System névtérből használjuk a Console osztályt

namespace HelloJedlik //A saját programunk osztálya az itt megadott névtérbe kerül
{
    class Program //A saját osztályunk azonosítója (neve) „Program” lesz
    {
        //Speciális Main() metódus (eljárás)
        static void Main() //A végrehajtás itt kezdődik
        { //Utasítás csoportot (blokkot) indító kapcsos zárójel
            Console.Clear(); //Paraméter nélküli metódus (utasítás)
            Console.WriteLine("Hello Jedlik!"); //Metódus a paraméterével
            Console.ReadKey(); //Billentyű leütését váró metódus
        } // Utasítás csoportot (blokkot) záró kapcsos zárójel
    }
}
```

A fenti rövid program `using`, `namespace`, `class`, `static`, `void` szavait foglalt szavaknak hívjuk. A foglalt szavak alkotják a programozási nyelv (itt C#) „szókincsét”. A foglalt szavak listáját a következő linken találod meg: http://msdn.microsoft.com/en-us/library/x53a06bb.aspx

2 Változók és adattípusok

2.1 A változó fogalma és jellemzői

Az algoritmusokban a **változók** a megadott típusú **értékek** (adatok) tárolására használt memóriatartományok elnevezései. A változó betűkből, számokból és néhány írásjelből álló elnevezése az **azonosító**. Ezen azonosítók segítségével lehet lekérdezni és módosítni a változók értékét.

A **változó típusa** – azt jelenti, hogy milyen fajta adatot tárolhat a változó (szöveges, numerikus, logikai, stb...). (pl.: a `byte` típusú változó numerikus adatok, egészek tárolására alkalmas)

A **változó értéktartománya** – a változó által felvehető lehetséges értékek halmaza.
(pl.: A `byte` típusú változó 0-255-ig vehet fel értékeket)

2.2 Deklaráció és definíció

A változókat a felhasználásuk előtt deklarálni kell. A **deklaráció** azt jelenti, hogy meghatározzuk a változó típusát és azonosítóját. Ha a változónak a deklaráláskor kezdőértéket is adunk, akkor a változót **definiáljuk**.

A deklaráció általános alakja (szintaxis):

szintaxis: `változóTípusa változóAzonosítója;` pl.: `int x;`

A definíció általános alakja (szintaxis):

szintaxis: `változóTípusa változóAzonosítója = kezdőérték;`
pl.: `byte hónap = 12;`

2.3 A változók típusai

2.3.1 Egészek

A leggyakrabban a 32 bites helyfoglalású `int` és `uint` típust használjuk:

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

A 8, 16 és 64 bites helyfoglalású változókra is szükségünk lehet a feladatok megoldása során:

Név	Leírás	Értéktartomány
sbyte	8 bites előjeles egész	-128 : 127
byte	8 bites előjel nélküli egész	0 : 255
short	16 bites előjeles egész	-32 768 : 32 767
ushort	16 bites előjel nélküli egész	0 : 65535
long	64 bites előjeles egész	-9 223 372 036 854 775 808 : 9 223 372 036 854 775 807
ulong	64 bites előjel nélküli egész	0 : 18 446 744 073 709 551 615

2.3.2 Valósak

Név	Leírás	Értékes jegy	Értéktartomány
float	32 bites lebegőpontos	7	$\pm 1{,}5*10^{-45} : \pm 3{,}4*10^{38}$
double	64 bites lebegőpontos	15	$\pm 5{,}0*10^{-324} : \pm 1{,}7*10^{308}$
decimal	128 bites nagypontosságú	28	$\pm 1{,}0*10^{-28} : \pm 7{,}9*10^{28}$

2.3.3 Logikai

Név	Leírás	Értéktartomány
bool	Logikai adattípus	true vagy false (igaz vagy hamis)

2.3.4 Karakteres

Név	Leírás	Értéktartomány
char	Egyetlen Unicode karakter	16 bites (UTF-16) kódtartomány
string	Unicode karaktersorozat	Legfeljebb 2^{32} db Unicode karakter

3 Kifejezések

A **kifejezések** (expression) adatokat szolgáltató operandusokból és a rajtuk valamilyen műveletet végző operátorokból állnak, tehát **operátoroknak** a műveleti jeleket hívjuk, melyek az **operandusokon** végeznek műveleteket. Pl.: a+b, vagy 7-a*b

A kifejezések egymásba ágyazhatóak: egy kifejezés operandusa lehet egy másik kifejezés is. pl.: (a+b) * (a/b)

Több operátor esetén az operátorok fontossági sorrendje (precedenciája) határozza meg a kiértékelés sorrendjét. A sorrend zárójelezéssel explicit módon is meghatározható. Azonos precedenciájú operátorok kiértékelése balról jobbra történik.

3.1 Értékadó operátorok

Az értékadó operátorok közül leggyakrabban az **egyszerű értékadás** operátorát használjuk, melynek jele (operátora) az egyenlőségjel. Pl.:

```
int a = 12;
byte b = 2*c;
double x = Math.Sqrt(y);
```

Az értékadás jobboldalán álló literálnak (12), kifejezésnek (2*c) vagy függvényhívásnak (Math.Sqrt(y)) a típusa azonosnak vagy kompatibilisnek kell lennie a baloldalon álló változó típusával.

Literálnak azokat az állandókat (konstansokat) hívjuk, melyekhez nem rendelünk azonosítót.

A kompatibilitás azt jelenti, hogy az átalakítás (konverzió) automatikusan (**implicit**) végbemegy.

3.2 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

3.3 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

3.4 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**

3.5 Egyéb operátorok

A programozási nyelvek egyéb operátorokat is használnak. A c# programozási nyelv operátorainak teljes listáját a következő linken találod: (nézd meg a mintaprogramokat is!)
http://msdn.microsoft.com/en-us/library/6a71f45d%28v=vs.140%29.aspx

4 Osztály, objektum fogalma

Az **osztály** (`class`) az objektum orientált programozás legsokoldalúbb adattípusa. Vannak olyan osztályok, melyekből példányokat kell létrehoznunk ahhoz, hogy használni tudjuk őket. Ezeket az osztályokat **dinamikus osztályoknak** hívjuk, és a példányosításukkal **objektumot** hozunk létre. Az objektumoknak, a változókhoz hasonlóan azonosítót kell adni:

szintaxis: `osztályTípusa objektumAzonosítója = new osztályKonstruktora();`
pl.: `Random r = new Random();`

A **konstruktor** egy speciális metódus (alprogram), ami felkészíti az objektumot a használatra, azonosítója kötelezően megegyezik az osztály azonosítójával.

Azokat az osztályokat, melyeket nem kell példányosítani a használathoz, **statikus osztályoknak** hívjuk. Ilyen például a **`Console`** vagy a **`Math`** osztály.

Az osztályok (objektumok) különböző típusú **tagokat** tartalmaznak. A két legfontosabb tagtípus a metódus és a jellemző.

metódus példa: `Console.Clear(); //Törli a képernyőt`

jellemző példa: `Console.Title = ”Konzol ablak”; //Ablak címsora`

Az általunk készített program egyetlen tagja (metódusa) (`Main()`) előtt a `static` szó látható, így a programunk statikus osztály típusú, azaz példányosítás nélkül indítható, használható.

5 Console osztály használata

A `Console` osztály használatával tudunk egyszerű programokat készíteni, valójában a konzolablak („képernyő”) kezelését valósítja meg. A `Console` osztály a `System` névtérben található.

Fontosabb alprogramjai (metódusai):

- WriteLine() //Képernyőre írás + soremelés
- Write() //Képernyőre írás soremelés nélkül

- ReadLine() //Szöveges típusú adat beolvasása
- ReadKey() //Karakter (ConsolKeyInfo) beolvasása
- Clear() //Képernyő törlése
- SetWindowSize() //Képernyő méretének a beállítása
- SetCursorPosition() //Kurzor pozíciójának állítása

Fontosabb jellemzők:

- BackgroundColor //Háttér színe
- ForegroundColor //Előtér színe
- Title //Konzolablak címsora
- WindowWidth //Konzolablak szélessége
- WindowHeight //Konzolablak magassága

Teljes leírás, mintapéldákkal: http://msdn.microsoft.com/en-us/library/system.console.aspx

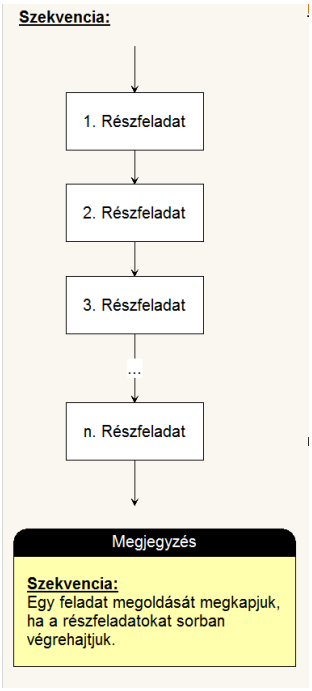
6 Vezérlési szerkezetek

A program **vezérlési szerkezete** meghatározza, hogy a programban (algoritmusban) leírt utasításokat a számítógép milyen sorrendben hajtsa végre.

A vezérlési szerkezetek fajtái:

- szekvencia,
- szelekció,
- iteráció.

Strukturáltnak nevezzük azt a programot, amely csak a fenti három vezérlési szerkezetet tartalmazza.

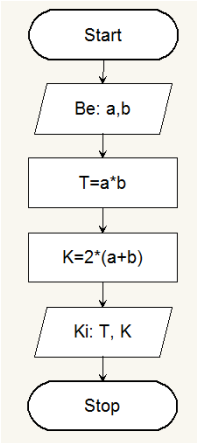


6.1 Szekvencia

A **szekvencia** olyan vezérlési szerkezet, amelyben az utasítások leírásának sorrendje meghatározza a végrehajtásuk sorrendjét is. Tipikusan az egymás alá írt utasítások leírási sorrendje a végrehajtási sorrend.

A téglalap kerületét és területét meghatározó program egyszerű utasítások megfelelő sorrendben történő végrehajtásával algoritmizálható. A következő pontokban az algoritmus megadását és a program forráskódját adjuk meg.

6.1.1 Téglalap kerülete és területe folyamatábrával



6.1.2 Téglalap kerülete és területe mondatszerű leírással

```
Program:
   Be: a, b
   T:=a*b
   K:=2 * (a + b)
   Ki: K, T
Program vége.
```

6.1.3 Téglalap kerülete és területe C# forráskóddal

```
using System; //System névtér importja a Console osztályhoz

namespace TéglalapTK //Névterünk azonosítója: TéglalapTK
{
    class Program //Osztályunk azonosítója: Program
    {
        static void Main() //Itt kezdődik a végrehajtás
        {
            Console.WriteLine("Téglalap kerülete és területe");
            Console.Write("a=");
            //A Console.ReadLine() "beolvasó" metódus (függvény)
            //szöveges (string) típusú adattal tér vissza.
            //Az int a = Console.ReadLine(); értékadás hibás,
            //mert az int <- string átalakítás (konverzió)
            //automatikusan (implicit) nem megy végbe,
            //ezért az int osztály Parse() metódusa végzi el a konverziót:
            int a = int.Parse(Console.ReadLine());
            Console.Write("b=");
            int b = int.Parse(Console.ReadLine());
            int T = a * b;
            int K = 2 * (a + b);
            Console.WriteLine("T={0} K={1}", T, K);
            Console.ReadKey(); //Billentyű leütésére vár
        }
    }
}
```