Debreceni Egyetem

Informatika Kar

Algoritmusok pszeudo kódban történő leírásának implementálása C#-ban

Dr. habil Boda István
Tanszékvezető

Lengyel Dávid
Programtervező Informatikus M.Sc

Debrecen, 2010

Tartalomjegyzék:

Bevezetés

A leíró nyelv egy magyar nyelvre épülő, programozást támogató segédeszköze az algoritmusok leírásának. Szakdolgozatom témája, hogy algoritmus leíró nyelvi forráskódból futtatható állományt hozzak létre. A dolgozat során bemutatásra kerül, hogy hogyan oldható meg a probléma C# programozási nyelv segítségével, .NET környezetben.

A dolgozat első részében bemutatásra kerül a program logikai felépítése és ezt követően részletesen leírom a fordítás lépéseit. A kódelemzés során fény derül arra, hogy reguláris kifejezésekkel és a forrásban való szabályalapú kereséssel hogyan valósítható meg a fordítóprogram.

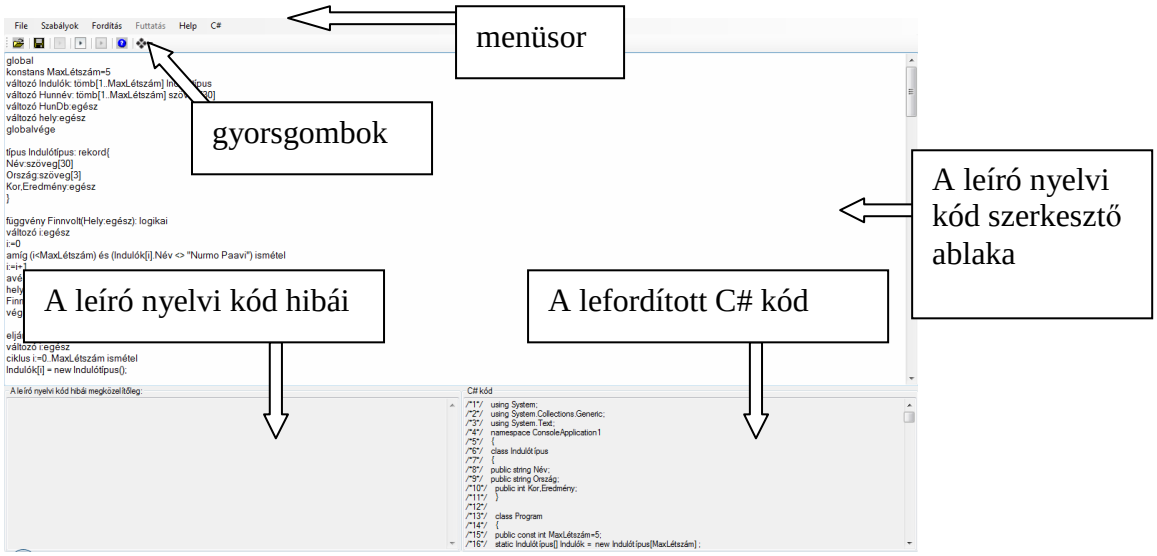
A dolgozat további részeiben a lefordított kód hibaelemzésének és a futtatható állomány előállításának pontjai kerülnek bemutatásra. A program használatát a forráselemzést követő felhasználói útmutató segítheti.

Témaválasztásom fő indoka a programozási nyelvek közötti különbségek áthidalása iránti érdeklődésem volt. A dolgozat célja megvalósítani a leíró nyelv - C# programnyelv konverziót, amely azon diákok és hallgatók segítségére szolgálhat, akik ismerik a leíró nyelv szabályrendszerét és el akarják sajátítani a C# programozási nyelv használatát.

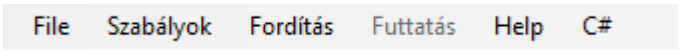
A program készítésekor a két nyelv közötti, szintaxisbeli hasonlóságokat használtam fel, kisebb módosításokat eszközölve, melyek biztosítják a forrás- és lefordított kód kompatibilitását. A dolgozat, a korábbi szakdolgozatom továbbfejlesztése, melyben a régebbi programverziót optimalizáltam és a hibakeresést módosítottam, egyes részek újragondolása és kivitelezése mellett.

A program logikai felépítése

A program a következő elemeket tartalmazza:



- Menüsor: kategóriákba rendezve, ahol megtalálhatók a program főbb funkciói:



A menüsor elemei, kategóriánként:

- Fájl menü
  - Forráskód beolvasása
  - Forráskód mentése
  - Kilépés
- Fordítás menü
  - Leíró nyelvi kód fordítása C#-ra
  - C# kód fordítása
- Futtatás menü
- Help menü
  - Help
  - Fontos Információk
- C# kód menü
  - Megjelenítés
  - Elrejtés
  - Kódhibák megjelenítése

- Gyorsbillentyűk: a leggyakrabban használt menük gyűjteménye:

 - Fájlmegnyitás – elmentett leíró nyelvi kódok megnyitására szolgál

 - Fájlmentés – a leíró nyelvi forráskód mentése végezhető el ezzel a funkcióval

 - Leíró nyelvi kód fordítása C#-ra – leíró nyelvi kód transzformálása C# kódra

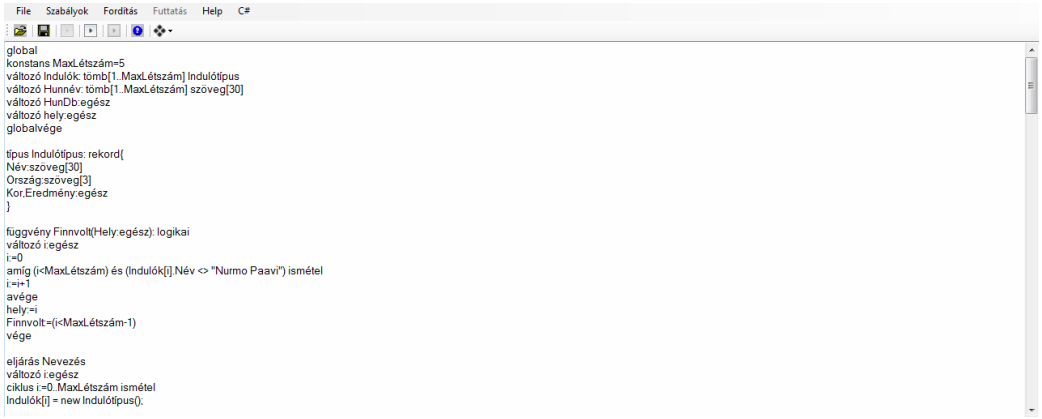
 - C# kód fordítása – a leíró nyelvi kódból generált C# kód fordítása

 - Futtatás – a lefordított C# kód futtatása, parancssoros ablakban

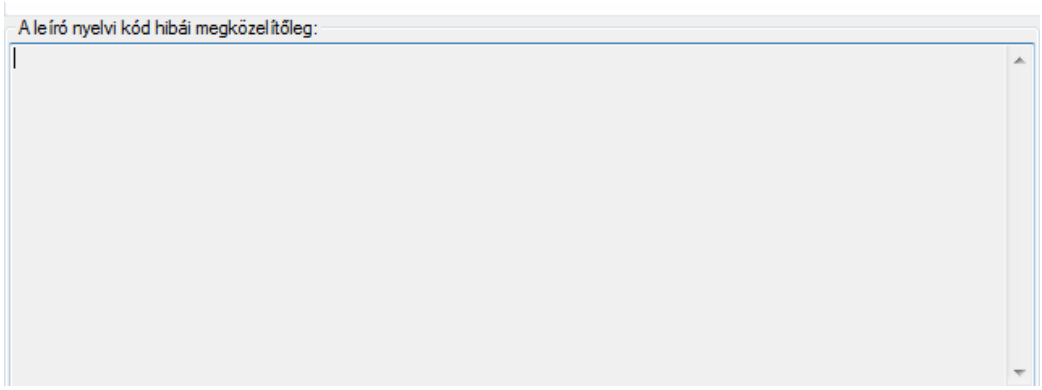
 - Help – a leíró nyelv szintaktikáját és hivatkozásait tartalmazó dokumentumot tekinthetjük meg, ezen menü választásával

 - C# kód megjelenítése, lenyíló sáv használatával – C# kód nevű szövegdoboz megjelenítésére és elrejtésére szolgál

- A leíró nyelvi kód szerkesztőablaka: leíró nyelvi kód begépelésére ad lehetőséget, illetve megnyitás esetén a mentett program kódja ebbe a szerkesztőablakba fog megnyílni:

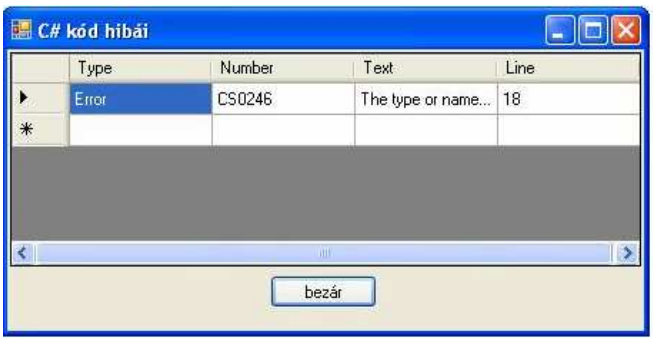


- A leíró nyelvi kód hibái szövegdoboz: ha a lefordított kód esetlegesen tartalmaz valamilyen hibát, ekkor a program a hibás kódrészletet ebben a mezőben fogja megjeleníteni:



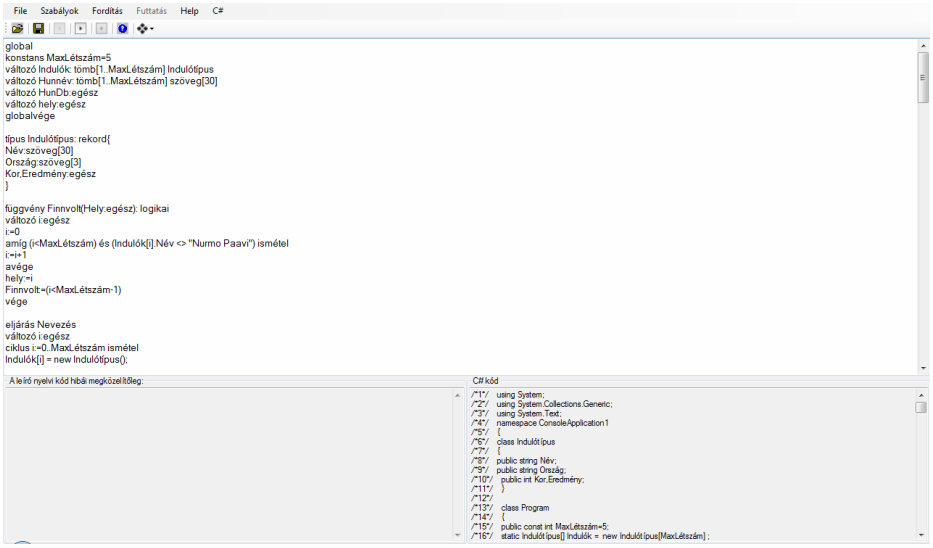
- C# kód szövegdoboz: A leíró nyelvi kód C# programnyelvre lefordított változatát fogja tartalmazni fordítás után:

A program a főablak mellett egy másik ablakot is tartalmaz, amelynek a szerepe a C# kód – kódhibák megjelenítése menüre kattintva, hogy táblázatos formában jelenítse meg a C#kód esetleges hibáit. Ez az opció akkor lehet szükséges, ha a hiba nem a szintaxisbeli, hanem szemantikus eredetű.



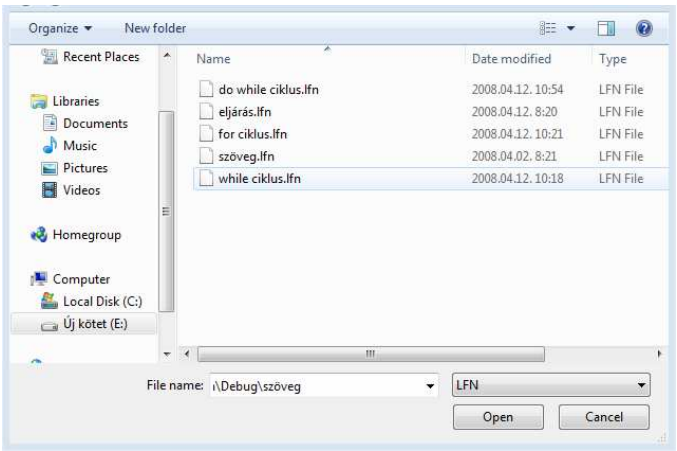
A programkód elemzése:

A program megnyitása után a következő ablak nyílik meg, a fennt említett menükkel, gyorsgombokkal, szerkesztőfelülettel és szövegdobozokkal:



Ekkor lehetősége van a felhasználónak kézzel begépelni a forráskódot, illetve megnyitni egy már mentett változatot. A felhasználók munkáját emelett a Help menüben található két almenü (Help és Fontos Információk) is segíthetik.

A mentett fájlok megnyitását a Fájl menü - Forráskód beolvasása vagy Forráskód megnyitása gyorsgombbal teheti meg a felhasználó.
A Forráskód beolvasása menü választása esetén a következő ablak fog megnyílni:



A felhasználó ekkor kiválaszthat egy LFN kiterjesztésű fájlt, melyet az Open gombra kattintva a leíró nyelvi kód szerkesztőablakába fog megnyitni a program. A program tartalmaz előre megírt példa forráskódokat is, melyek segítséget nyújthatnak a felhasználók számára a leíró nyelv elsajátítása szempontjából.

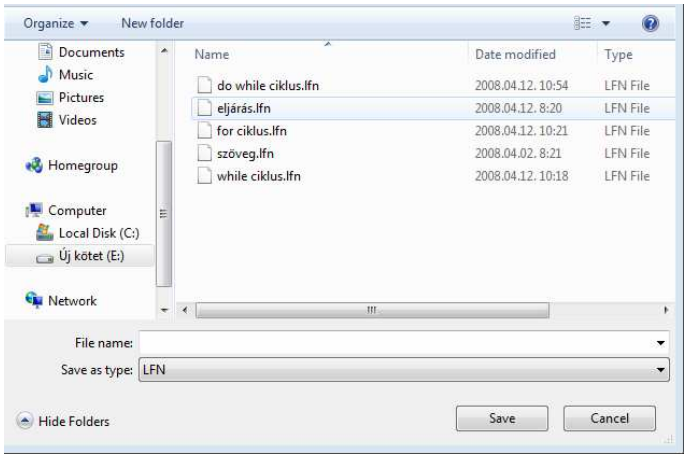
A fájl megnyitás megvalósítása:

```
openFileDialog1.Filter = "LFN|*.lfn";
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
           Stream st = File.Open(openFileDialog1.FileName, FileMode.Open);
          StreamReader sr1 = new StreamReader(st, Encoding.GetEncoding("iso-8859-2"));
           textBox1.Text = sr1.ReadToEnd();
           sr1.Close();
           st.Close();
        }
```

A fájlmegnyitás kezeléséhez a fordítóprogram egy fájlmegnyitó párbeszédablak (OpenFileDialog) komponenst használ fel. A fájlmegnyitó komponens ShowDialog függvényének meghívásával a program megjeleníti a fájlmegnyitó párbeszédablakot, lehetőséget nyújtva a felhasználóknak LFN kiterjesztésű fájlok megnyitására. (Az LFN kiterjesztés a program saját, egyedileg használt formátuma.) A ShowDialog függvény visszatérési értékét megvizsgálva program megállapítja, hogy a felhasználó melyik gombra kattintott (Open vagy Cancel). Ha ennek az eredménye: Open, ekkor megkezdődik a kiválasztott fájl feldolgozása és tartalmának kiírása a leíró nyelvi kód szerkesztőablakába.
A feldolgozáshoz a program egy Stream és egy StreamReader C# komponenseket használ fel. A Stream komponensbe fogja a program eltárolni a megnyitandó fájl nevét, illetve ezt fogja a StreamReader komponens felhasználni a fájl megnyitásához és végigolvasásához. A folyamat végeztével a StreamReader komponensbe beolvasott tartalmat (a fájl tartalmát) a program kiírja a szerkesztőablakba és bezárja a komponenseket.

A felhasználónak lehetősége van a begépelt vagy megnyitott kód szerkesztése után a kód mentésére is a File - Forráskód mentése menü vagy gyorsgomb használatával.

Fájl mentés:
A Forráskód mentése menü választása esetén a következő ablak nyílik meg:



A felhasználónak ekkor lehetősége van a forráskódot LFN kiterjesztésű fájlba menteni.

A fájlmentés megvalósítása:

A fájlmentést a program egy fájlmentés párbeszédablak (SaveFileDialog) komponens használatával valósítja meg, melynek működése hasonló a fennt említett fájlmegnyitó komponenséhez. A fájlmentő komponens megnyitása és LFN kiterjesztés-szűrő beállítása után itt is egy Stream komponenst alkalmaz a program a fájl nevének tárolásához, illetve helyet kap egy StreamWriter komponens is a mentés végrehajtásához. A párbeszédablak eredményét megvizsgálva (Save vagy Cancel) a program itt is megállapítja, hogy felhasználó mely gombra kattintott, és ha a Save gombot választotta megkezdődik a mentés.
A StreamWriter megkapja a fájl nevét tartalmazó Stream-et, létrehozza a felhasználó által definiált fájlt és a szerkesztőablak tartalmát a megadott nevű fájlba írja, majd bezárja a komponenseket.

```
SaveFileDialog saveFileDialog1 = new SaveFileDialog();
        saveFileDialog1.Filter = "LFN|*.lfn";
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
           Stream myStream = saveFileDialog1.OpenFile();
           StreamWriter wText = new StreamWriter(myStream, Encoding.GetEncoding("iso-
8859-2"));
           wText.Write(textBox1.Text);
           wText.Close();
           myStream.Close();
         }
```

A fájl menü utolsó menüpontjában szerepel a Kilépés opció, mellyel bezárhatjuk a programot. Ezt a menüt az Application.Exit(); paranccsal valósítja meg a program.

Ha a felhasználó befejezte a leíró nyelvi forráskód szerkesztését, ekkor lehetősége van a C# kód előállítására a Fordítás – Leíró nyelvi kód fordítása C#-ra menüvel vagy gyorsgombbal.

A leíró nyelvi kód feldolgozása és fordítása

A leíró nyelvi kód feldolgozása:

A program a leíró nyelvi kód előfeldolgozásához négy szöveg típusú változót és az ezekhez a változókhoz tartozó függvényeket tartalmaz. A négy változó és függvény alkalmazása azért szükséges, hogy a leíró nyelvi kódrészletek fordítás után a C# programnyelvnek megfelelő helyükre kerüljenek. (például: a fő osztálytól különböző osztályok a C# programban ne a *Main* függvénybe kerüljenek) A négy változó a fő program utasításait, a fő programtól különböző utasításokat, függvények utasításait és a statikus hatáskörű utasításokat és változókat fogják tartalmazni. Ezeknek megfelelően a hozzájuk tartozó négy függvény fogja az utasításokat ezekbe a változókba elkülöníteni és lefordítani. Az utasítások elválasztását egymástól a program a leíró nyelv nyelvtanának kiaknázásával valósítja meg a következő módon:

- Ha a leíró nyelvi kód statikus hatáskörű utasításokat tartalmaz, ekkor ezek az utasítások *global* kulcsszót tartalmazó sorral kezdődnek és *globalvége* kulcsszót tartalmazó sorral végződnek.
- Ha a kód függvények vagy eljárások utasításait tartalmazza, ekkor az ilyen utasításokat tartalmazó sorok *eljárás* vagy *függvény* kulcsszót tartalmazó sorral kezdődnek és *vége* kulcsszóból álló sorral végződnek.
- Ha a kód fő programtól különböző utasításokat tartalmaz, akkor az ide kapcsolódó utasítások *típus* kulcsszót tartalmazó sorral kezdődnek és *}* karaktert tartalmazó sorral végződnek
- Ha a fennt felsoroltak közül egyik eset sem érvényesül, akkor a főprogram utasításairól van szó.

A leíró nyelvi kódban ezeknek a részeknek az elválasztását a program úgy valósítja meg, hogy a Leíró nyelvi kód fordítása C#-ra gombra/ menüre való kattintás után beolvassa a leíró nyelvi kód szerkesztőablakában található forráskódot egy szövegbeolvasó komponens segítségével, soronként és sorok tartalmát elemzi a fennt említett kulcsszavak vizsgálatával. Például a következő kódrészlet azt vizsgálja meg, hogy a beolvasott sor üres értékű-e vagy *függvény* kulcsszóval kezdődik-e. (függvényeket tartalmazó sorok vizsgálata)

A feltétel teljesülése esetén egy változóba olvassa be a sort és egy újsor karaktert a jobb olvashatóság kedvéért. A kódban látható ciklus addig fog futni, amíg a program *vége* kulcsszóval kezdődő sort nem talál.

```
…
if (line != null && line.StartsWith("függvény"))
                {
                   while (true)
                   {
                      szerkezet += line;
                      szerkezet += System.Environment.NewLine;
                      if (line.StartsWith("vége"))
                      {
                         break;
                      }
                      line = reader.ReadLine();
                   }
                }
if (szerkezet != "")
                {
                   lista1.Add(szerkezet);
                   lista2.Add(fordítspec(szerkezet));
                   leforditottkulon += System.Environment.NewLine;
                }
…
```

A vizsgálatok elvégzése után a program a kulcsszavaknak megfelelő változóba tölti be az utasításokat, illetve a megfelelő hozzá kapcsolódó függvényt fogja végrehajtani. Azonban mielőtt a függvények elkezdenék a forráskód C# programnyelvi megfelelőjére való fordítását, a szétdarabolt forráskódot a program listákba fogja szedni.

A listákba rendezésnek a program elemzése során később, a hibakeresésnél lesz jelentősége. A program két listával fog dolgozni. Az egyik a leíró nyelvi kódrészleteket fogja tartalmazni, melyeket az előzőekben keresett meg a program és szeparált el egymástól.
A másik lista pedig ugyanezeket az utasításokat fogja tartalmazni, csak már C# programnyelvre fordított formában.

A második lista feltöltésére a kódrészlethez tartozó függvény meghívása után kerül sor, amikor a függvény működése befejeződött és a függvény visszatérési értéket szolgáltatott. A fent szereplő kódrészletben a listához való hozzáadást a *lista.Add* függvénnyel valósítja meg a program. A második lista (lista2) feltöltésekor a program végrehajtja a megfelelő függvényt, mely függvény magát a C# programnyelvre való fordítást végzi el (a fennti kódban a *fordítspec* függvény), illetve az ide tartozó változóba fogja helyezni a lefordított kódot (*leforditottkulon*).

Az előzőekben említett négy függvény, működésük során további függvényeket fog segítségül hívni a fordítás megvalósításához. Ezek az alfüggvények fogják elvégezni a leíró nyelvi vezérlési szerkezetek fordítását. Vezérlési szerkezeteken a programon belül a következőket értjük:

- Elágazások
- Ciklusok
- Tömbök
- Változók
- Bekérő és kiíratő műveletek
- Karakterláncok
- Konstansok

A vezérlési utasítások fordítását reguláris kifejezések és mintaillesztés segítségével fogja végrehajtni a program. A következőkben ismertetésre kerülnek a reguláris kifejezések, melyeket a program készítése során alkalmaztam.

Reguláris Kifejezések

Sokszor kell megoldanunk olyan problémákat, ahol szövegben való keresést kell megvalósítanunk. Ekkor rengeteg időnket elveszi, hogy létrehozzuk a megfelelő keresési algoritmusokat, főleg akkor, ha nem egy szót, hanem egy mintát keresünk, például egy telefonszámot. Ezekre a problémákra kínálnak megoldást a reguláris kifejezések.
A program a reguláris kifejezések megvalósításához két névteret használ, és segítségükkel valósítja meg a szövegben való keresést és a cserék végrehajtását. (*VBScript_RegExp_55* és a *System.Text.RegularExpressions*). A kiindulási osztály, amely segítségével a keresés és csere funkciók végrehajtódnak a *RegExpClass* osztály.

A *RegExpClass* osztály tulajdonságai,paraméterei:

- Global - Igaz érték megadása esetén a keresés a teljes szövegben történik és a találati listába az összes találat bekerül.
- IgnoreCase – Igaz érték esetén a keresés nem tesz különbséget kis és nagy betűk között
- Pattern – Ebben az opcióban adhatjuk meg a keresési mintát
- Execute – Keresés végrehajtása a megadott szövegen, visszatérési értéke egy MatchCollection típusú lista lesz, mely egy találati lista
- Replace – Ez a metódus végzi el a cseréket illetve a keresést, visszatérési értéke az a szöveg amely a már kicserélt találatokat tartalmazza
- Test – Végrehajtásával megtudhatjuk, hogy a keresett minta szerepel-e az adott szövegben

A program a reguláris kifejezések szövegben való kiértékeléséhez a RegExpClass osztály Execute metódusát használja a következőképpen:

```
RegExpClass r = new RegExpClass();
        r.IgnoreCase = true;
        r.Global = true;
        r.Pattern = ("ha .+ akkor\r\n(.+\r\n)+(különben\r\n(.+\r\n)+)?hvége");
        VBScript_RegExp_55.MatchCollection m =
(VBScript_RegExp_55.MatchCollection)r.Execute(s);
        if (m.Count>0)
        {…
```

A fent szereplő kódrészletben az elágazások mintaillesztése szerepel. Először a program létrehoz egy új *RegExpClass* osztályt, mellyel a mintaillesztést fogja elvégezni, illetve beállítja, hogy a keresés a teljes szövegen menjen végbe és ne tegyen különbséget kis és nagy betűk között. Ezután következik a minta megadása, ami a fennti kódrészletben: "*ha .+ akkor\r\n(.+\r\n)+(különben\r\n(.+\r\n)+)?hvége*". A keresési minta megadása után következik a mintaillesztés bemeneti szövegre, amely a fennti kód esetében *s* lesz. A mintaillesztés eredményének kiértékeléséhez a program találati listákat használ fel (MatchCollection), melyeket a mintaillesztés után megvizsgál. Ha a találati lista találatainak száma (*m.Count*) nagyobb mint nulla, azaz a minta illeszkedik a vizsgált bemenetre, ekkor megkezdődik a találatok számának vizsgálata.

```
…if (m.Count>0)
            {
if (m.Count == 1)
                {
                   for (int i = 0; i < m.Count; i++)
                   {
                      VBScript_RegExp_55.Match item = (VBScript_RegExp_55.Match)m[i];
                      talalat = item.Value.ToString();
                   }
                   try
                   {
                      talalat = ifelagazas(talalat, r.Pattern);
                      lepes = Regex.Replace(s, r.Pattern, talalat);
                      s = lepes;
                   }
                   catch (Exception e)
                   {
                      MessageBox.Show(e.Message);
                   }
                }
                else
                      s = többtalálat(r, m, lepes, talalat, s);…
```

Ha a találatok száma egy, ekkor a program a találatot egy változóba helyezi (talalat) és végrehajtja a változón a mintához kapcsolódó C# programnyelvre fordító függvényt (ifelagazas). Ezután egy másik változó felhasználásával (lepes) kicseréli a bemenetben (*s*) a mintát a lefordított kódra.

A kódrészletben szereplő *többtalálat* függvény alkalmazására akkor kerül sor, ha a találati listában a találatok száma nagyobb mint egy, azaz több illeszkedésről van szó.

A vezérlési utasítások fordítását tehát a fent említett mintaillesztések és reguláris kifejezések alkalmazásával valósítja meg a program, az alábbi szintaktikát felhasználva:

Reguláris kifejezések szintaktikája C#-ban:

- A ’.’ karakter bármilyen betűt helyettesít, kivéve az újsor karaktert
- A \w karakter bármilyen betű típusú karaktert helyettesít
- A \d karakter bármilyen szám típusú karaktert helyettesít
- A [ad].. egy olyan szót helyettesít, amelynek első karakterje lehet ’a’ vagy ’d’
- A ’^’ karakter használatával tagadhatunk például: [^ad].. egy olyan szót helyettesít, amelynek első karakterje sem ’a’ sem ’d’ nem lehet
- Megadhatunk zárójelek között karakter intervallumot is: [a-d].. egy olyan szót helyettesít, amelynek első karakterje lehet ’a’ vagy ’b’ vagy ’c’ vagy ’d’

A minták megadásánál lényeges lehet az egyezések száma szerinti keresés is:

- A ’*’ karakter nulla vagy annál több egyezést fog mutatni egy szón belül

Például:

```
Szöveg: Anna Jones and a friend owned an anaconda
Minta: a\w*
Találatok: Anna Jones and a friend owned an anaconda
```

- A ’+’ karakter egy vagy annál több egyezést fog mutatni egy szón belül

Például:

```
Szöveg: Anna Jones and a friend owned an anaconda
Minta: an?
Találatok: Anna Jones and a friend owned an anaconda
```

- A ’?’ karakter nulla vagy egy egyezést fog mutatni egy szón belül

Például:

```
Szöveg: Anna Jones and a friend owned an anaconda
Minta: an?
Találatok: Anna Jones and a friend owned an anaconda
```

A leíró nyelvi kifejezések szintaxisa

Konstansdeklaráció

konstans konstansnév=érték

Változódeklaráció

változó változónév1[,változónév2]…:típus

Egydimenziós tömb deklarációja

változó tömbnév:tömb[alsóhatár..felsőhatár] elemtípus

Többdimenziós tömb deklarációja

változó tömbnév:tömb[ah1..fh1[ah2..fh2]…] elemtípus

Karakterlánc deklarációja

változó karakterláncnév: szöveg[maxhossz]

Rekord deklarációja

típus rekordnév: rekord{
mező11[,mező12]…:típus1
[mező21[,mező22]…:típus2
…]
}

Eljárás deklaráció

eljárás eljárásnév[(formális paraméterek)]
    deklarációk
    …
    utasítások
vége

Függvény deklaráció

függvény függvénynév[(formális paraméterek)]: elemtípus
    deklarációk
    …
    utasítások
    függvénynév:=kifejezés
vége

Vezérlő utasítások

Értékadó utasítás

változó:=kifejezés

Beolvasó és kiíró utasítások

be: változó

ki: kifejezés

Szelekciós (feltételes) utasítások

Feltételes elágazás

ha feltétel **akkor**
    utasítás
[**különben**
    utasítás…]
hvége

Iterációs (ciklus-) utasítások

Általános elöltesztelő ciklus

amíg feltétel **ismétel**
        utasítás…
avége

Előírt lépésszámú elöl tesztelő ciklus – Elöl tesztelő, léptető ciklus

ciklus cv:kezdő érték...végérték **lépésköz:** lk **ismétel**
        utasítás…
cvége

Hátul tesztelő ciklus

ismétel
  utasítás…
ivége feltétel **esetén**

Statikus hatáskörű változók deklarációja

global

változó deklarációk

globalvége

Hivatkozási formák

Egydimenziós tömbök

tömbnév[indexkifejezés]

Többdimenziós tömbök

tömbnév[indexkif1[,indexkif2]…]

Karakterláncok

karakterláncnév

Rekord egy mezője

rekordnév.mezőnév

Eljáráshívás

eljárásnév[aktuális paraméterek]

Függvényhívás

függvénynév[aktuális paraméterek]

A szintaktika ismeretében a programnak a következő táblázatban leírt konverziókat kell megvalósítania:

Utasítás	Leíró nyelvi megfelelő	C# kódbeli megfelelő
Konstans deklaráció	**konstans** konstansnév=érték	const int konstansnév=érték;
Változó deklaráció	**változó** változónév:típus	típus változónév;
Egydimenziós tömb deklaráció	**Változó** tömbnév:tömb[alsóhatár..felsőhatár] elemtípus	elemtípus[] tömbnév = new elemtípus[felsőhatár];
Többdimenziós tömb deklaráció	**Változó** tömbnév:tömb[ah1..fh1[ah2..fh2]…] elemtípus	elemtípus[,,] tömbnév = new elemtípus[fh1,fh2,…];
Karakterlánc deklaráció	**változó** karakterláncnév: szöveg[maxhossz]	string karakterláncnév=””;

Eljárás deklaráció	**eljárás** eljárásnév[(formális paraméterek)]
	deklarációk, utasítások
	Vége
		static void
		eljárásnév(paraméterek)
		{
		deklarációk, utasítások
		}

Függvény deklaráció	**függvény** függvénynév[(formális paraméterek)]: elemtípus
	deklarációk,utasítások
	függvénynév:=kifejezés
	Vége
		static elemtípus
		függvénynév(paraméterek)
		{
		deklarációk,utasítások
		return kifejezés;
		}

Értékadó utasítás	változó:=kifejezés	változó = kifejezés
Beolvasó utasítás	**be:** változó	a=Console.ReadLine(); a=int.Parse.Console.Readline();
Kiíró utasítás	**ki:** kifejezés	Console.WriteLine(„kifejezés”);

Feltételes elágazás	**ha** feltétel **akkor**
	utasítás
	[**különben**
	utasítás…]
	Hvége
		if(feltétel)
		{
		utasítás…
		[else
		{
		utasítás…
		}]
		}

Általános elöl tesztelő ciklus	**amíg** feltétel **ismétel**
	utasítás…
	Avége
		While(feltétel)
		{
		utasítás
		}

Hátul tesztelő ciklus	**Ismétel**
	utasítás…
	ivége feltétel **esetén**
		do
		{
		utasítás
		}
		while(feltétel)

Előírt lépésszámú elöl tesztelő ciklus	**ciklus** cv:kezdő érték...végérték
	lépésköz: lk **ismétel**
	utasítás…
	cvége
		for(cv=kezdő
		érték;i<végérték;i=i+lépésköz)
		{
		utasítás…
		}

Globális változók deklarációja	Global változó deklarációk Globalvége	A változók static hatáskört kapnak és a programkód elején lesznek deklarálva.

Rekordok deklarációja	típus rekordnév: rekord{
	mezőnév:típus
	}
		class rekordnév
		{
		Public típus mezőnév;}

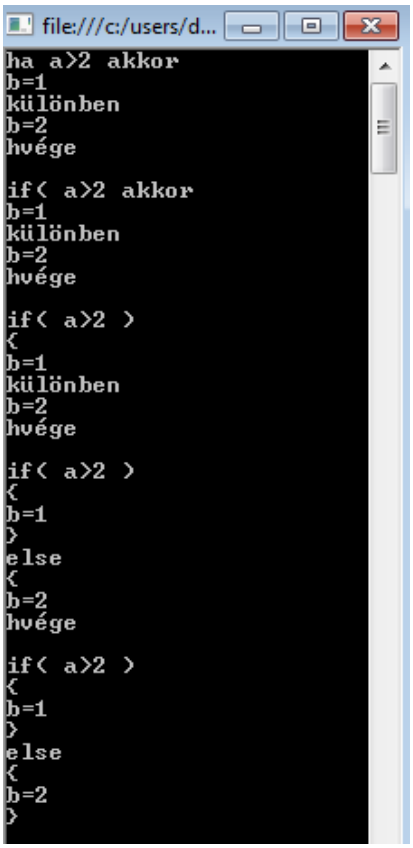
A cserék végrehajtása reguláris kifejezésekkel

Feltételes elágazások cseréje:

A feltételes elágazások esetén a program a következő mintát illeszti a bemeneti szövegre: *„ha .+ akkor\r\n(.+\r\n)+(különben\r\n(.+\r\n)+)?hvége”*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül fog megvalósulni:

1) A *”ha”* szó cseréje *”if(”*-re
2) Az *”akkor”* szó cseréje *”)új sor karakter és {”*-re
3) A *”különben”* szó cseréje *”}újsor else újsor {”*-ra
4) A *”hvége”* szó cseréje *’}’*-ra

```
public string ifelagazas(string talalat, string Pattern)
    {
        string lepes = "";
        lepes = Regex.Replace(talalat, "ha", "if(");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "akkor", ")\r\n{");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "különben", "}\r\nelse\r\n{");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "hvége", "}");
        talalat = lepes;
        return talalat;
      }
```

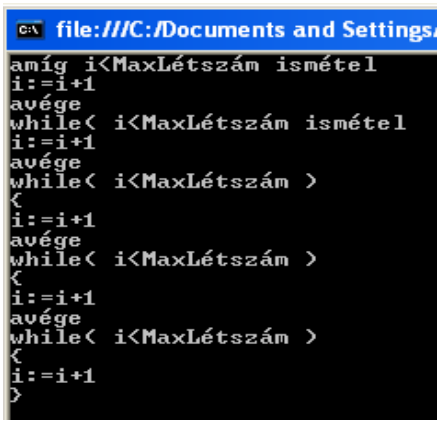


A forráskódban szereplő függvény egyik paramétere a feltételes elágazásra vonatkozó találati lista eleme, a másik pedig a fent említett minta. A program két változó segítségével hajtja végre a cseréket (lepes és talalat) és ezek közül az egyik (talalat) lesz a függvény visszatérési értéke, mely tartalmazza a lefordított kódot, melyet a fent említett listák közül a C# kódot tartalmazóhoz fogja hozzáadni (lista2).

Elöltesztelő ciklusok cseréje:

Az elöltesztelő ciklusok esetén a program a következő mintát illeszti a bemeneti szövegre: *"amíg .* ismétel\r\n(.*\r\n)*avége"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül történik:

1) Az *„amíg”* szó cseréje *„while(”* -ra
2) Az *„ismétel”* szó cseréje *„) újsor {”*-ra
3) Az *„avége”* szó cseréje *„}”*-ra



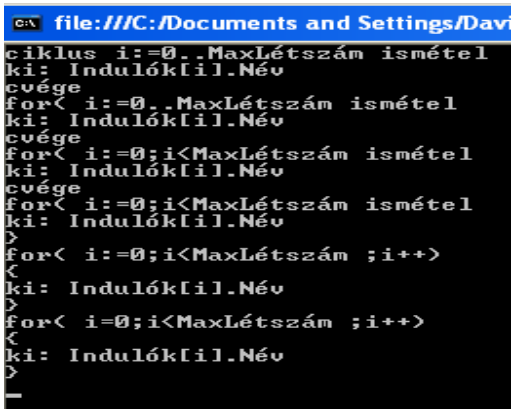
A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

```
public string whileciklus(string talalat, string Pattern)
    {
        string lepes = "";
        lepes = Regex.Replace(talalat, "amíg", "while(");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "ismétel", ")\r\n{");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "avége", "}");
        talalat = lepes;
        return talalat;
    }
```

Előírt lépésszámú ciklusok cseréje:

Az előírt lépésszámú ciklusok esetén a program a következő mintát illeszti a bemeneti szövegre: *"ciklus .:=.* ismétel\r\n(.+\r\n)+cvége(\r\n)?"*. A mintaillesztést követő fordítási lépések ebben az esetben:

1) Először a program deklarál egy karakter típusú változót, mely felveszi a ciklusváltozó értékét.
2) A *„ciklus”* szó cseréje *„for(”*-ra
3) A *”..”* cseréje *„; ciklusváltozó <”*-re
4) A *„cvége”* szó cseréje *„}”*-re
5) Ha a talalat tartalmazza a lépésköz szót akkor lecseréli a *„lépésköz:”* szót *”; ciklusváltozó +=”* -re és az *„ismétel”* szót *„) újsor {”* -re
6) Ha nem tartalmaz lépésközt akkor az *„ismétel”* szót *”; ciklusváltozó ++) újsor {”*-ra cseréli

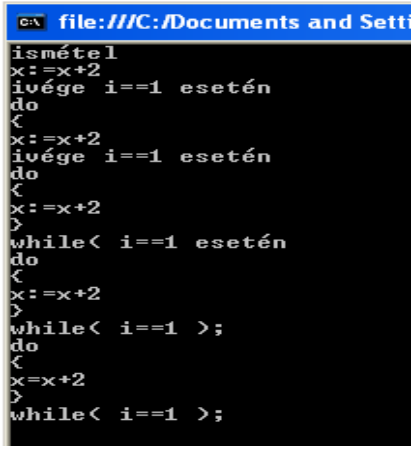


A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

Hátultesztelő ciklusok cseréje:

A hátultesztelő ciklusok esetén a program a következő mintát illeszti a bemeneti szövegre: *"ismétel\r\n(.+\r\n)+ivége .* esetén"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül fog megvalósulni:

1) *„ismétel”* szó cseréje *”do újsor {”*-ra
2) *„ivége”* szó cseréje *”} újsor while(”*-ra
3) *„esetén”* szó cseréje *”);”*-ra



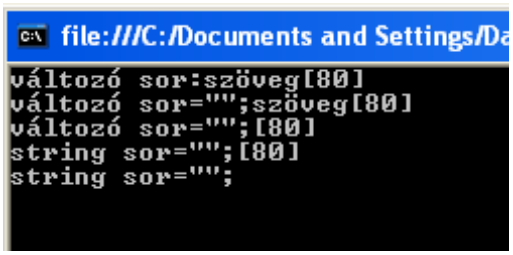
A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

```
public string dowhileciklus(string talalat, string Pattern)
    {
        string lepes = "";
        lepes = Regex.Replace(talalat, "ismétel", "do\r\n{");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "ivége", "}\r\nwhile(");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "esetén", ");");
        talalat = lepes;
        return talalat;
    }
```

Karakterláncok cseréje:

Az karakterláncok esetén a program a következő mintát illeszti a bemeneti szövegre: *"változó .*: string[[].*[]]"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül válósul meg:

1) *':'* karakter cseréje *"="";"*-re
2) *"string"* szó cseréje üres karakterre
3) *"változó"* szó cseréje *"string"*-re
4) *"[szám]"* cseréje üres karakterre



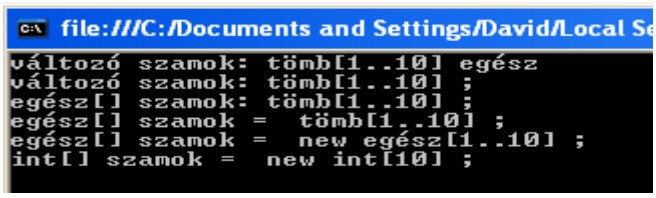
A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

```
public string karakterláncok(string talalat, string Pattern)
    {
        string lepes = "";
        talalat.Remove(talalat.Length - 1, 1);
        lepes = Regex.Replace(talalat, ":", "=\"\";");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "string", "");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "változó", "string");
        talalat = lepes;
        lepes = Regex.Replace(talalat, "[[].*[]]", "");
        talalat = lepes;
        return talalat;
    }
```

Egydimenziós tömbök cseréje:

Az egydimenziós tömbök esetén a program a következő mintát illeszti a bemeneti szövegre: *"változó .*: tömb" + "\\[.*\\]" + ".*"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül történik:

1) A tömb típusának eltárolása egy változóban
2) A szó végéről a típus levágása
3) A *„változó”* szó cseréje *„típus + [ ]”* –re
4) A *’:’* karakter cseréje egyenlőségjel (=) karakterre
5) A tömb szót a *”new”* szóra és a típusára cseréli
6) A típus nevét kicseréli a C# megfelelőjére

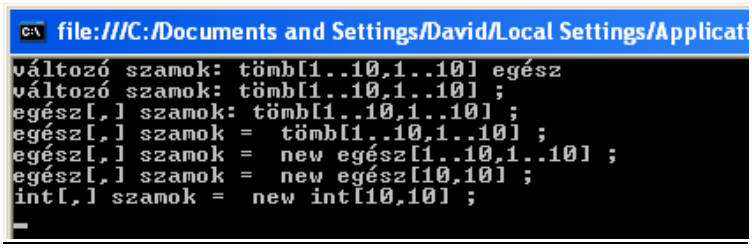


A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

Többdimenziós tömbök cseréje:

Az többdimenziós tömbök esetén a program a következő mintát illeszti a bemeneti szövegre: *"változó .*: tömb" + "\\[[0-9]+..[0-9]+(,[0-9]+..[0-9]+)+\\]" + ".*"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül fog megvalósulni:

1) Eltárolja a tömb típusát egy változóban
2) A string végéről levágja a típusát
3) A *„változó”* szót helyettesíti a típussal
4) A kettőspontot egyenlőségjellel helyettesíti és a szögletes zárójelekből eltávolítja a maximális index előtti karaktereket
5) A tömb szót *”new”* szóra és a típusára cseréli
6) A típus nevét kicseréli a C# megfelelőjére



A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

Változó deklarációk cseréje:

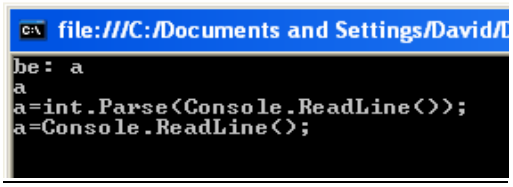
A változó deklarációk esetén a program a következő mintát illeszti a bemeneti szövegre: *"változó .+:.+"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül fog történni:

1) Megvizsgálja hogy a szó tartalmazza-e tömb szót
2) Ha igen továbblép, ha nem elkezdődhet a csere
3) Egy string típusú változóban eltárolja a változó típusát
4) A „*változó*” szót levágja a szó elejéről
5) A változó típusát elhagyja a szó végéről és az elejére szúrja be
6) A *':'* karaktert pontosvesszővel helyettesíti és a típus-t C# típusra írja át

Beolvasó utasítás cseréje:

A beolvasó utasítások esetén a program a következő mintát illeszti a bemeneti szövegre: *"be: .*"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül fog megvalósulni:

1) A string elejéről levágja a *"be: "* szövegrészt
2) A bekérendő változó neve után illeszti a *"=int.Parse(Console.ReadLine());"* szót
3) A bekérendő változó neve után illeszti a *"=Console.ReadLine();"* szót



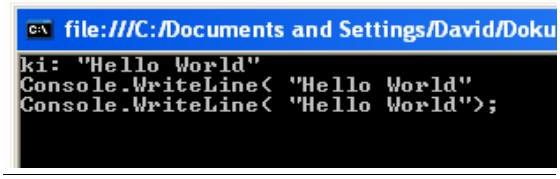
A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

Megjegyzés: Ez a megvalósítás azért szükséges, mert a leíró nyelv szintaxisa nem biztosítja a szöveg és az egész szám bekérés közötti különbséget, ami viszont C#-ban parse (típus) hibát okoz. A létrejövő két sor közül a rosszat fordítás során a fordító ki fogja szűrni, és törölni fogja.

Kiíró utasítás cseréje:

A kiíró utasítások esetén a program a következő mintát illeszti a bemeneti szövegre: *"ki:.*"*. A mintaillesztés után a leíró nyelvi kód fordítása a következő lépéseken keresztül egy függvényen keresztül fog megvalósulni:

1) A *"ki:"* kifejezést cseréje *"Console.WriteLine("* kifejezésre
2) Végül a string után beilleszti a *„);"* karaktereket



A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

Függvények és eljárások cseréje:

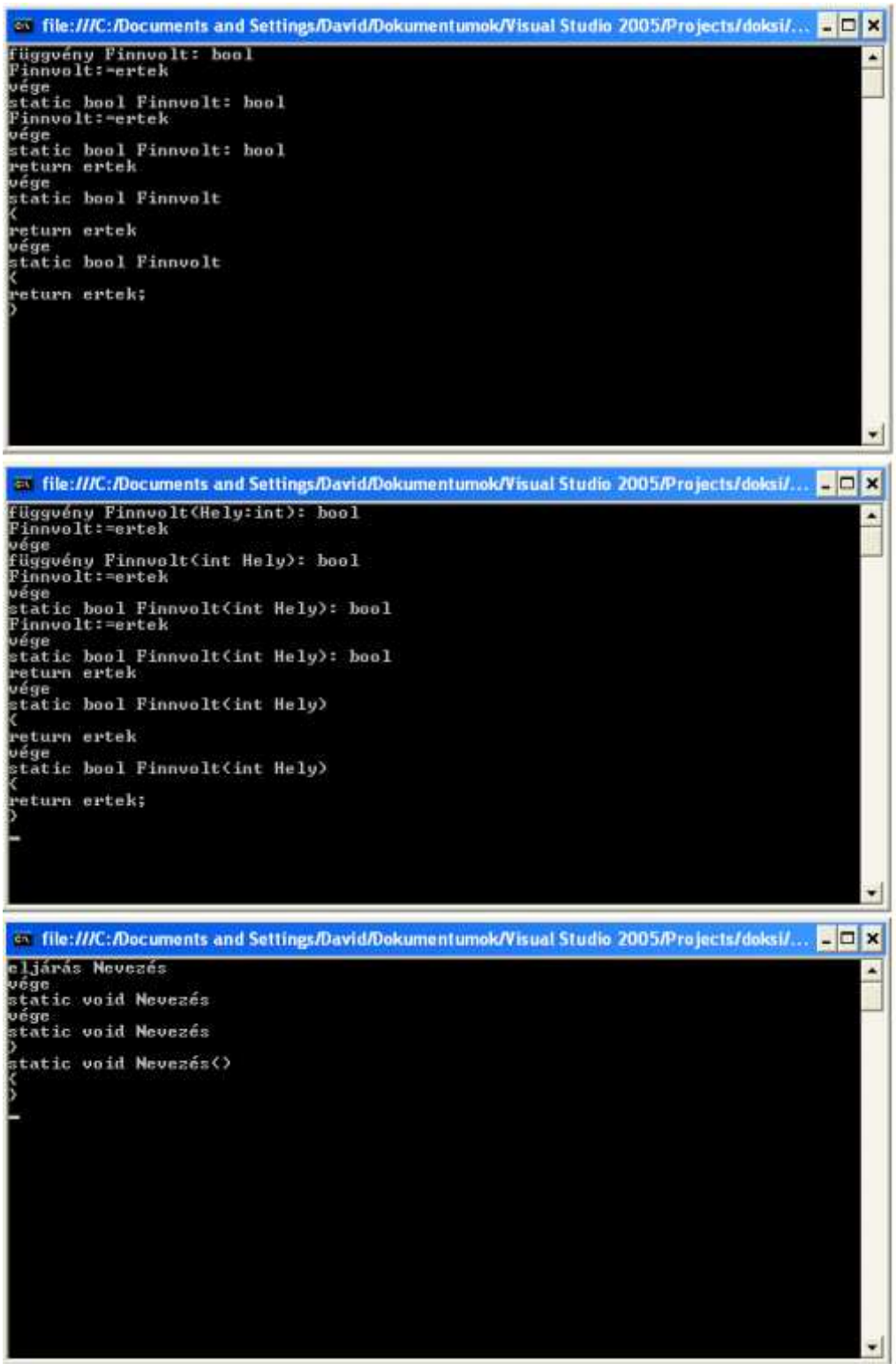
Paraméter nélküli függvények esetén a minta: *"függvény .+():.+\r\n(.+\r\n)+vége"*
Eljárások esetén a minta: *"eljárás .+()\r\n(.+\r\n)+vége"*
Paraméteres függvények esetén a minta: *"függvény .+(.+:.+):.+\r\n(.+\r\n)+vége"*

A program a függvények cseréjét a következőképp valósítja meg:

1) A típus eltárolása és „*static*" szóval való kibővítése után, ezen kifejezést a függvény neve elé illeszti
2) Paraméteres esetben a paramétereknél a típus és változó helyének cseréje, az egyenlőségjel elhagyásával
3) A visszatérési érték (függvényneve:= érték) cseréje „*return értékre*"
4) A függvény típusának levágása és zárójelek beillesztése

Eljárások cseréjének lépései:

1) Az *„eljárás"* szó cseréje *„static void"* szavakra
2) Zárójelek beillesztése



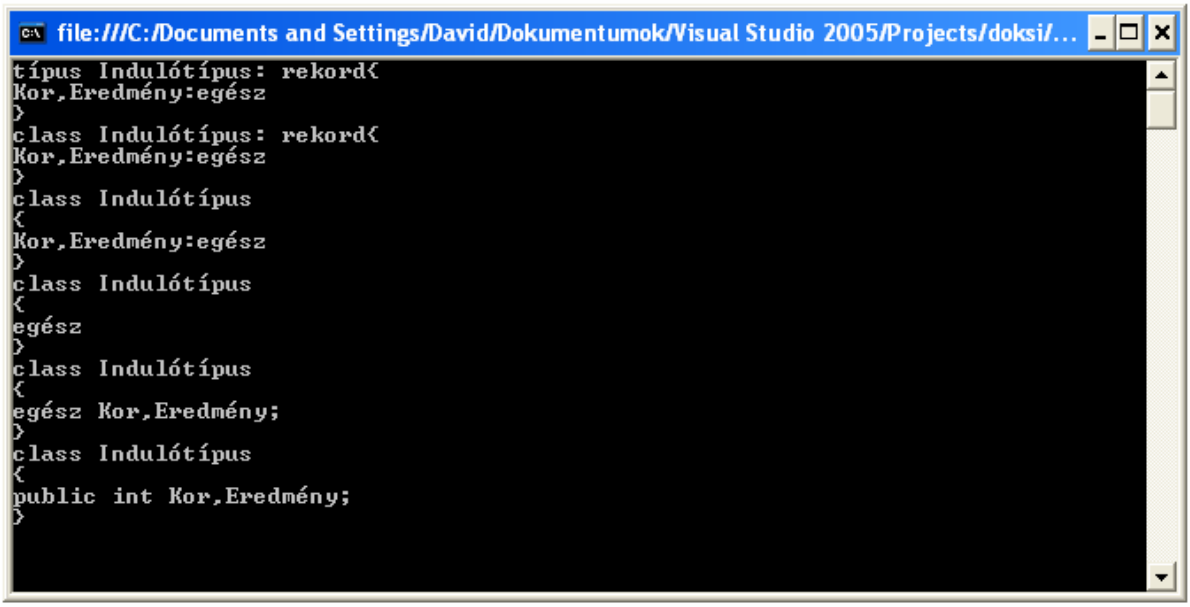
A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

A főprogramtól különböző osztályok fordítása a következő lépésekben történik:

1) A *string* típusú változók típusát cseréljük *public string*-re
2) A *double* típusú változók típusát cseréljük *public double*-re
3) Az *int* típusú változók típusát cseréljük *public int*-re
4) A *bool* típusú változók típusát cseréljük *public bool*-ra

A program az osztályokat a rekord típusú változók cseréjével hozza létre, illetve ez a függvény tartalmazza a benne található változók cseréjét is.

Rekordok cseréje:



A program szerinti megvalósítás lépésenkénti kiiratással szemléltetve

A statikus hatáskörű változók fordítását a program a változó típusának megváltoztatásával fogja megvalósítani, olyan módon, hogy a változó típusa elé beszúr egy ún. *static* szót, mely a változó hatáskörét ki fogja terjeszteni a teljes C# programkódra.

A fent felsorolt függvényeket használja a leíró nyelvi kód szeparáláshoz szükséges, előzőekben említett négy függvény. A fent ismertetett fordítási lépések mellett a program a cserék végrehajtása előtt a következő konverziókat is végrehajtja:

- *egész → int*
- *valós → double*
- *logikai → bool*
- *karakter → char*
- *szöveg → string*
- *konstans → const int*

Ide tartozik még a ”nem egyenlő”, az ”és” szavak és az értékadás cseréje is:

- és → &&
- <> → !=
- := → =

A fordítás során a következő sorrendet követi a program a vezérlési és egyéb szerkezetek fordítása esetén:

1) Elágazás vizsgálat
2) Elöltesztelő ciklus vizsgálat
3) Előírt lépésszámú ciklus vizsgálat
4) Hátultesztelő ciklus vizsgálat
5) Többszörös elágazás vizsgálat
6) Karakterlánc vizsgálat
7) Egy dimenziós tömb vizsgálat
8) Több dimenziós tömb vizsgálat
9) Változó vizsgálat
10) Bekérés vizsgálat
11) Kiiratás vizsgálat
12) Konstans vizsgálat

Abban az esetben, ha a mintára illeszkedő találatok találati listájában több egyezés van, a program a következő utasításokat fogja végrehajtani:

```
public string többtalálat(RegExpClass r, VBScript_RegExp_55.MatchCollection m, string
lepes, string talalat, string s)
      {
                  string sok = "";
                   int szam2 = m.Count;
                   string csere = "";
                   for (int i = 0; i < m.Count; i++)
                   {
                      VBScript_RegExp_55.Match item = (VBScript_RegExp_55.Match)m[i];
                      talalat = item.Value.ToString();
                      sok += sokatfordít(talalat);
                      sok += System.Environment.NewLine;
                   }
                   sok = Regex.Replace(sok, "\r", "");
                   csere = Regex.Replace(csere, "\r", "");
                   lepes = Regex.Replace(s, r.Pattern, sok);
                   s = lepes;
                   string[] tömbcsere = new string[s.Length];
                   StringReader reader2 = new StringReader(s);
                   string line1;
                   int b = 0;
                   while ((line1 = reader2.ReadLine()) != null)
                   {
                      tömbcsere[b] += line1;
                      b++;
                   }
                   s = "";
                   string[] helyes = RemoveDuplicates(tömbcsere);
                   for (int i = 0; i < helyes.Length; i++)
                   {
                      if (helyes[i] == "" || helyes[i] == null)
                      {
                         continue;
                      }
                      s += helyes[i];
                      s += System.Environment.NewLine;
                   }
                   return s;
      }
```

A program ennek a problémának a megoldását egy függvény segítségével valósítja meg (*többtalálat*), melynek paraméterei egy reguláris kifejezéseket kezelő osztálypéldány (*RegExpClass r*), a mintára illesztett találati lista (*VBScript_RegExp_55.MatchCollection m*), két szöveges változó melyekkel a cseréket fogja végrehajtani (*lepes, talalat*) és a bemenet (*s*).

Az egymásba ágyazott leíró nyelvi utasítások fordítását azért valósítottam meg ilyen formában, mert a reguláris kifejezések és találati listák erre a problémára (pl. feltételes utasítások/elágazások egymásba ágyazása) nem nyújtanak megoldást. Az ilyen típusú problémák kiküszöbölésére a program minden vezérlési utasítás fordítását külön-külön, egyesével fogja megvalósítani. Egy példán keresztül szemléltetem a megoldást:

Ha a leíró nyelvi kód egymásba ágyazott vezérlési utasításokat tartalmaz (pl. két elágazást), ekkor először lefordítja a külsőt, azonban ekkor a benne szereplő utasítások is lefordításra kerülnek, a tartalmazott vezérlési utasítással együtt. Második lépésben pedig a belső vezérlési utasítást fogja lefordítani. A két lépés végrehajtása után azonban a lefordított kód egyes részeit duplikáltan fogjuk visszakapni, azért mert a tartalmazott vezérlési utasítást többször is lefordítottuk. Ezt a problémát úgy küszöböli ki a program, hogy tömbökbe szedi a lefordított kódot és egy ún. *RemoveDuplicates* függvényt fog alkalmazni, mellyel a duplikált részeket fogja eltávolítani. A tömbök közül egy egyik a műveletek után a duplikációktól megtisztított kódot fogja tartalmazni és ennek a tömbnek a tartalmát fogja a program a C# kód nevű mezőbe feltölteni.

Megjegyzés: A program a fordítási folyamat során, ha függvények és eljárások fordításával találkozik, ekkor kigyűjti ezeknek a vezérlési szerkezeteknek a neveit (függvénynevek, eljárásnevek) egy listába (fuggvenynevek). Ennek a lépésnek a fordítási folyamat után következő hibakeresés során lesz fontos szerepe.

A leíró nyelvi kód fordítása akkor fejeződik be, amikor a forrás utolsó sorát is feldolgozta a program. Ekkor megkezdődik a C# kód nevű szövegdoboz feltöltése a lefodított részekkel.

Először az osztályok kerülnek be a C# kódba, a Program osztály definiálása után.
A kód jobb értelmezhetősége és olvashatósága szempontjából minden beillesztett kódrészlet után a program elhelyez egy újsor karaktert.

Az osztályok feltöltése után a statikus hatáskörű változók majd a függvények és eljárások kerülnek be a C# szabályoknak megfelelő helyükre. Végül a program definiálja a Main függvényt (főprogramot), ahová a főprogram utasításai fognak bekerülni.

A programnak ezután a C# kódon esetlegesen további változtatásokat kell elvégeznie.
Első lépésben a program soronként beolvassa a lefordított kódot. Ha a beolvasott sor tartalmaz függvény vagy eljárásnevet (fuggvenynevek listából ellenőrzi) és nem szerepel utána zárójel (függvény- vagy eljáráshívás), akkor a Regex.Replace (reguláris kifejezések cseréje) metódussal kicseréli a sorban a függvénynevet, függvénynév() – re.
Ha a sor nem üres és nem tartalmaz kulcsszakat (kulcsszavakon a namespace, class, static, if, else, while, for, do, int , string, bool, double, Console.Readline, Console.WriteLine, char, const, public, global kifejezések értendők) ekkor megvizsgálja és ha egy olyan szóból áll, ami betűkből és számokból tevődik össze (lehetséges függvény vagy eljáráshívás) ekkor a sor végére illeszti a (); szót, illetve ha a sor több szóból áll ekkor egy *';'* karaktert tesz a végére.

Végül olyan sorokat keres, amelyek tartalmazzák a *while*, *if* és = kifejezéseket és nem tartalmaznak ==, >=, <= vagy *!*= relációt. Ha ilyen sort találunk akkor kicseréljük az egyenlőségjelet (=) dupla egyenlőség jelre (==).

A fordítás utolsó lépésének elvégzése után a program aktivizálja a C# kód fordítása menüpontot és gyorsgombot, illetve inaktívvá teszi a leíró nyelv fordítása C#-ra opciókat.

A C# kód fordítása

A C# kód fordítása menü vagy gomb kiválasztásával megkezdődik a lefordított kód hibaellenőrzése és fordítása. A program a hibakereséshez adattáblákat alkalmaz, melyek a C# kód hibáinak típusát, hibaszámát, hibaszövegét és a hibás sorokat fogják tartalmazni. A program létrehoz egy példányt a *Microsoft.CSharp* névtér *CSharpCodeProvider* osztályából, hogy hozzáférjen a fordító-, illetve kódgeneráló-objektumhoz. Az osztály *CreateCompiler* metódusával létrehoz egy *ICompiler* interfészt, melynek metódusait és tulajdonságait felhasználjuk a fordításhoz.

```
            CSharpCodeProvider codeProvider = new CSharpCodeProvider();
            ICodeCompiler compiler = codeProvider.CreateCompiler();
            CompilerParameters parameters = new CompilerParameters();
            parameters.GenerateExecutable = true;
            parameters.OutputAssembly = "sample.exe";
            parameters.MainClass = "ConsoleApplication1.Program";
            parameters.IncludeDebugInformation = true;
            foreach (Assembly asm in AppDomain.CurrentDomain.GetAssemblies())
            {
               parameters.ReferencedAssemblies.Add(asm.Location);
            }
            CompilerResults results = compiler.CompileAssemblyFromSource(parameters,
textBox4.Text);
            if (results.Errors.Count == 0)
            {
               bToolStripMenuItem.Enabled = true;
               toolStripButton6.Enabled = true;
               futtathato = true;
            }
```

A *CompilerParameters* nevű osztályt példányosítva létrehoz egy objektumot, mely tulajdonságaiban tartalmazza a fordítási paramétereket. A *GenerateExecutable* tulajdonsággal pedig megadja, hogy futtatható állományt generáljon a fordító és a futtatható állomány nevét beállítja *sample.exe*-re. A program ezután beállítja, hogy a futtatható állomány tartalmazzon *DEBUG* információkat és a *CompilerParameters* osztály *ReferencedAssemblies* property-jébe összegyűjti a fordításkor hivatkozott assemblyket.

Végül a C# szövegmező szövegét, mint forráskódot, valamint a feltöltött paraméterlistát tartalmazó objektumot átadjuk a fordítót reprezentáló objektum *CompileAssemblyFromSource* metódusának, mely elvégzi a fordítási műveletet.
Leegyszerűsítve az előző lépéseket a program létrehoz egy C# kód fordító objektumot, melyet a lefordított leíró nyelvi kódra fog alkalmazni. A C# kód fordító objektuma megvizsgálja a kódot és hiba esetén feltölti a hiba paramétereit az adattáblákba.

Ha a fordító nem talál hibát a program elérhetővé teszi a Futtatás menüt és gyorsgombot.

Ha a fordító viszont hibát talál a forráskódban, ekkor a hiba típusát, számát, szövegét és sorát feltölti az adattáblákba. Az adattáblákban szereplő hibákat a fordítás végeztével a felhasználó megjelenítheti a C# kód - kódhibák megjelenítése menüvel egy új ablakban, táblázatos formában. A fordítás során a program minden hiba esetén megvizsgálja, hogy a hiba száma egyezik-e a *CS0029* hibakóddal. Ez a hibakód azt fogja jelenteni, hogy a lefordított kódban ún. parse (típuskényszerítés) hiba található, melyet egy erre a problémára kialakított függvénnyel fog javítani a program (parsehibak függvény).

Hibakeresés fordítás után

A program a C# kód fordítása után hibakeresést végez el a lefordított kódon, megvizsgálva, hogy a kiíró és beolvasó utasítások esetén milyen típuskényszerítést kell alkalmaznia.
Megjegyzés: Az ilyen típusú utasításoknál a leíró nyelvi kód fordítófüggvénye két típuskényszerítést alkalmaz (egész és szöveg típusra), azaz ugyanaz a kódsor kétszer fog szerepelni a C# programnyelvre lefordított kódban, egyszer egész típusra, másszor pedig szöveg típusra kényszerítve.

A hibakeresés során a program a hibákat tartalmazó adattábla segítségével soronként végigolvassa a lefordított kódot, ellenőrizve, hogy a kódsor szerepel-e az adattáblában.
Ha a kód szerepel a hibás sorokat tartalmazó adattáblában és a hiba típusa egy ún. *parse* hiba, amely típuskényszerítés hibát jelöl, a program törölni fogja az adott sort a lefordított kódból, illetve a lefordított kódot és leíró nyelvi kódot tartalmazó listákból.
Az ilyen típusú hibák javítása után a program újra meghívja a fordító metódusokat, elölről kezdve a fordítási folyamatot.

Abban az esetben, ha a program hibás sort talál az ellenőrzés során és a hiba nem típuskényszerítésből származik, ekkor megvizsgálja az adott sort a következőképpen:

A leíró nyelvi kód fordítása során a program a leíró nyelvi kód sorait és vezérlési szerkezeteit egy listába gyűjti ki és a fordítás után az ezeknek megfelelő C# kódot is listába fogja szedni. A két lista elemszáma és indexelése meg fog egyezni azzal a különbséggel, hogy az egyik a leíró nyelvi kódot a másik pedig annak a lefordítottját fogja tartalmazni. A hibakeresés során, ha a hiba nem javítható (nem típuskényszerítés), ekkor a program a fent említett két listát úgy fogja felhasználni, hogy a hibát tartalmazó sor számát visszakeresi a leíró nyelvi kódot tartalmazó listából és kiírja a Leíró nyelvi kód hibái szövegdobozba.

A C# kód hibáit emellett a felhasználók megtekinthetik a C# menü - Debug információk opciójának választásával. Az itt jelenlévő hibák a leíró nyelvi kód szemantikai hibáit tartalmazzák, pl. ha a leíró nyelvi kódban a felhasználó egy deklarálatlan változót használ.

A Debug információk menü ezeket a hibákat táblázatos formában fogja megjeleníteni egy új programablakban. A táblázat pedig a hiba típusát, számát, szövegét és sorát fogja tartalmazni. Ez az opció a felhasználó számára akkor lehet hasznos, ha a Leíró nyelvi kód hibáit tartalmazó hibajegyzék nem írná le pontosan a hiba okát.

A hibakeresést megvalósítását két függvény végzi el a fordítás után. (hibakeres és hibakeres2)

```
public void hibakeres()
      {
         StringReader reader = new StringReader(textBox4.Text.ToString());
         string line1;
         int szam = 1;
         hibassorok = new string[dataSet1.hibak.Rows.Count];
         while ((line1 = reader.ReadLine()) != null)
         {
            for (int i = 0; i < dataSet1.hibak.Rows.Count; i++)
            {
               if (dataSet1.hibak.Rows[i][dataSet1.hibak.LineColumn].ToString() ==
szam.ToString())
               {
                  for (int j = 0; j < hibassorok.Length; j++)
                  {
                     if (hibassorok[j] == "" || hibassorok[j] == null)
                     {
                        line1 = Regex.Replace(line1, "/*.*/    ", "");
                        hibassorok[j] = line1;
                        break;
                     }
                  }
               }
            }
               szam++;
         }
      }
```

```
public void hibakeres2()
      {
         string hibalista = "";
         for (int i = 0; i < hibassorok.Length; i++)
         {
            for (int j = 0; j < lista2.Count; j++)
            {
               if (lista2[j].ToString().Contains(hibassorok[i]) &&
!hibalista.Contains(lista1[j].ToString()))
               {
                  hibalista += lista1[j].ToString();
                  hibalista += System.Environment.NewLine;
               }
            }
         }
         for (int i = 0; i < hibassorok.Length; i++)
         {
            if (!textBox2.Text.Contains(hibassorok[i]))
            {
               textBox2.Text += hibassorok[i] + System.Environment.NewLine; }}}
```

A kód lefordítása után a program inaktívvá teszi a C# kód fordítása menüpontot és gyorsgombot. Ha a kód hibákat tartalmaz, egyik fordítás opció és a futtatás menük sem lesznek elérhetőek.

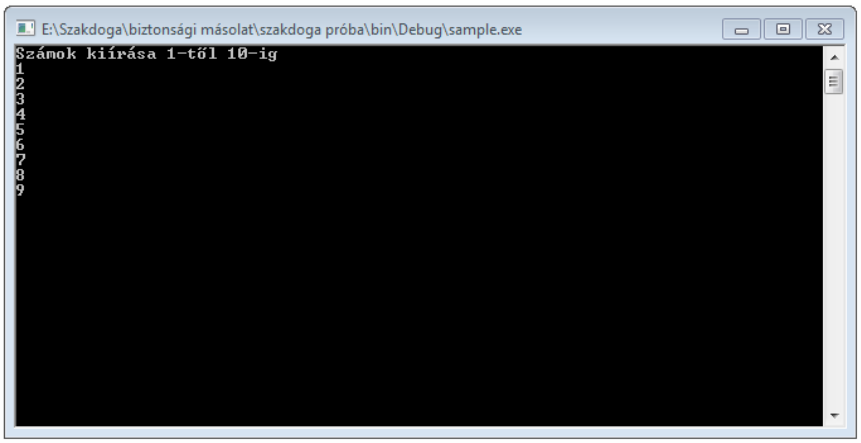
Ekkor az egyetlen megoldás a leíró nyelvi kód és a C# kód hibáit felhasználva módosítani a leíró nyelvi forráskódon, amely után aktívvá válik a Leíró nyelvi kód fordítása C#-ra opció.

Hibátlan kód esetén a program elérhetővé teszi a futtatás opciókat.

Futtatás és egyéb funkciók

A futtatást a program a fordítás során létrehozott *sample.exe* állomány elindításával valósítja meg. Futás közbeni hiba esetén a hiba okáról felugró ablakban kapunk tajékoztatást.
A lefordított leíró nyelvi kód parancssoros felületen, konzolablakban fog elindulni/lefutni.

```
private void toolStripButton6_Click(object sender, EventArgs e)
      {
         {
            ProcessStartInfo pInfo = new ProcessStartInfo("sample.exe");
            pInfo.ErrorDialog = true;
            Process.Start(pInfo);
         }
      }
```



Példaprogram futtatási képe

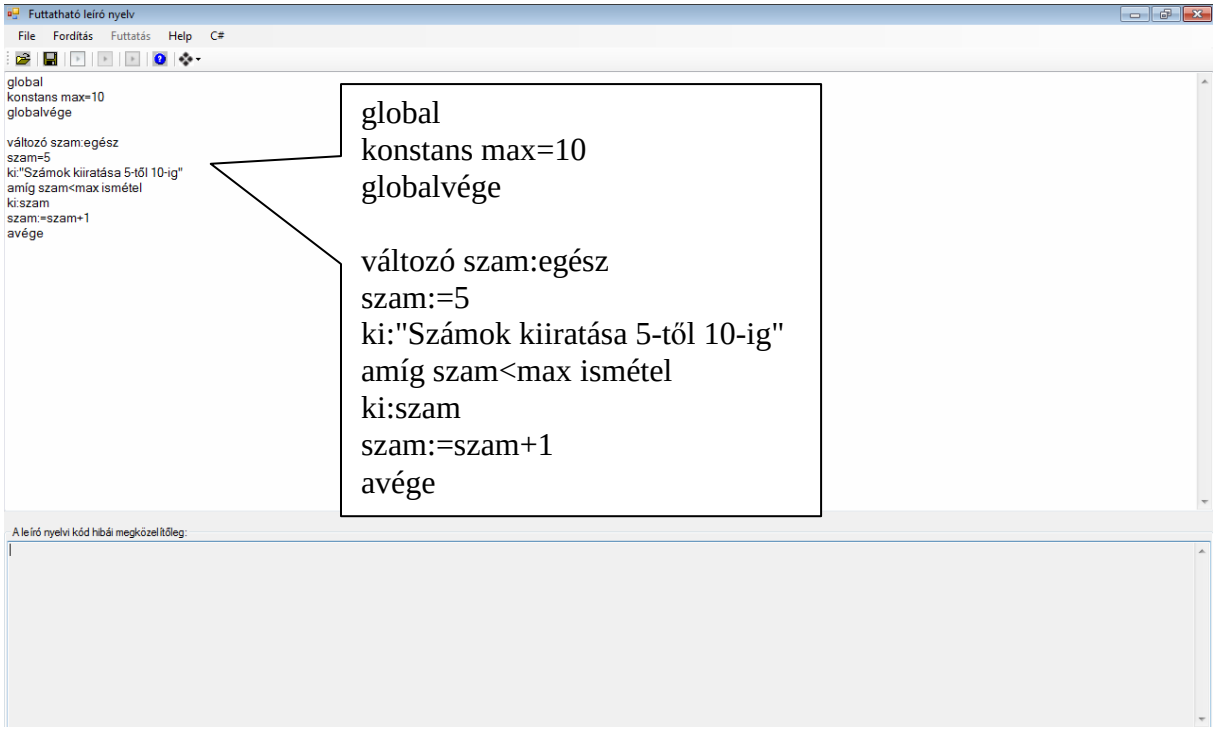
A Help menü Help opcióját kiválasztva a program megnyitja a Help.rtf fájlt (mely a program könyvtárában található), Internet Explorer segítségével. Ekkor lehetősége van a felhasználónak menteni vagy megnyitni a leíró nyelv szintaxisát tartalmazó dokumentumot.

Ugyanzen menü Fontos Információk opciójának választásával a program felugró ablakban tájékoztatja a felhasználót hogy a leíró nyelvi kód írása során, ha egy osztályt tömb formában példányosít és a tömb elemeinek értéket akar adni ekkor a következő sort kell alkalmaznia:

```
tömbnév[i] = new osztálynév();
```

Példaprogram

A program használatát egy példaprogramon keresztül szemléltetem, mely egy ciklus segítségével kiírja a számokat 5től 10ig. Indításkor a File menü Forráskód beolvasása menüpont kiválasztása után a *while ciklus.lfn* fájl megnyitásával a mentett kód betöltődik a leíró nyelvi kód szerkesztőablakába.



A forráskód megnyitása után a Fordítás menü – Leíró nyelvi kód fordítása C#-ra opciójára kattintva elkezdődik a fordítási művelet a következőképpen:

A program beolvassa a szerkesztőablak tartalmát és elkezdi a leíró nyelvi részek szétválogatását:

Ebben az esetben a kód statikus hatáskörű változóval kezdődik, ezért a *global* és *globalvége* szavak közötti kódot fogja a program először megtalálni és fordítani. A két szó közt található utasítást felveszi a lista1 listába, mely a leíró nyelvi kód utasításait fogja tartalmazni. Majd a lista2 listához a program ennek a kódrésznek (*konstans max = 10*) a lefordítottját fogja hozzáadni, úgy hogy a listához való hozzáadás közben meghívja az ide tartozó statikus hatáskörű változókat fordító függvényt. Ez a függvény lefordítja az említett kódsort és hozzáadja a lefordított statikus hatáskörű utasításokat összegyűjtő változóhoz.

A soronkénti beolvasás szerinti következő utasítás a *változó szám:egész* lesz. A sorról a program megállapítja, hogy nem tartalmaz vezérlési szerkezetet és ezért magát a sort fogja lefordítani. Ekkor a fent említett listákhoz való hozzáadás művelet fog ismét végrehajtódni, azzal a különbséggel, hogy ebben az esetben a változó deklarációhoz tartozó fordítási lépések kerülnek végrehajtásra és a lefordított kód a főprogram utasításait tartalmazó változóba fog kerülni, a típus cseréje után (*egész -> int*). Meghívódik a változókat fordító függvény a vizsgált sort tartalmazva paraméterben és elkezdődik a cserék végrehajtása.

Először a *„változó”* szót cseréli ki a program üres szóra, így a bemenet a következő sorra redukálódik: *„szam:int”*. Majd a kettőspont karakter által elhatárolt változónevet és típust egy-egy változóba gyűjti ki és ezek felhasználásával a visszatérési értéket a „típus változónév;” minta alapján fogja szolgáltatni a függvény. (*int szam;*)

A kód következő sora a *„szám := 5”* lesz. Mivel ez a sor sem tartalmaz vezérlési utasításokat, ezért a sort fogja a program lefordítani és listákba venni. Az egyedüli változtatás, melyet a program végre fog hajtani ezen a soron, hogy kicseréli a *„:=”* karaktereket az egyenlőségjel karakterre.

Tovább haladva a kódon ismét egy egysoros utasítás fog következni:
„ki: "Számok kiiratása 5-től 10-ig"”. Az előző sorhoz hasonlóan a program itt is magát sort fogja lefordítani. A fordítás során a következő mintára illesztve a sort: "ki:.*". A program egyezést fog találni és meghívja a kiíró utasítások fordítását végző függvényt, mely a *„ki:”* szót fogja lecserélni a *„Console.WriteLine(”* szóra és a szó utolsó karaktere után beszúrja a *„);”* karaktereket.
A program itt is hozzáadja a lefordított és leíró nyelvi utasítást az említett listákhoz.

A következő kódsor az *„amíg szam<max ismétel”* lesz. A program ekkor felismeri, hogy egy leíró nyelvi vezérlési szerkezetről (ciklusról) van szó és a forráskód tartalmát egy változóba fogja beolvasni egészen az *„avége”* szóig és a változó tartalmát fogja mintákra illeszteni.
A mintaillesztés során pedig a program a *"amíg .* ismétel\r\n(.*\r\n)*avége"* mintával fog egyezést találni, mely után meghívja az elöltesztelő ciklusok fordításáért felelős függvényt.
A függvény a következő három lépést fogja végrehajtani:

1. Az *„amíg”* szót kicseréli *„while(”*-ra

```
while( szam<max ismétel
ki:szam
szam:=szam+1
avége
```

2. Az *„ismétel”* szót kicseréli *„)/r/n{”* karakterekre, ahol a *„/r/n”* az újsor karaktert jelöli

```
while( szam<max )
{
ki:szam
szam:=szam+1
avége
```

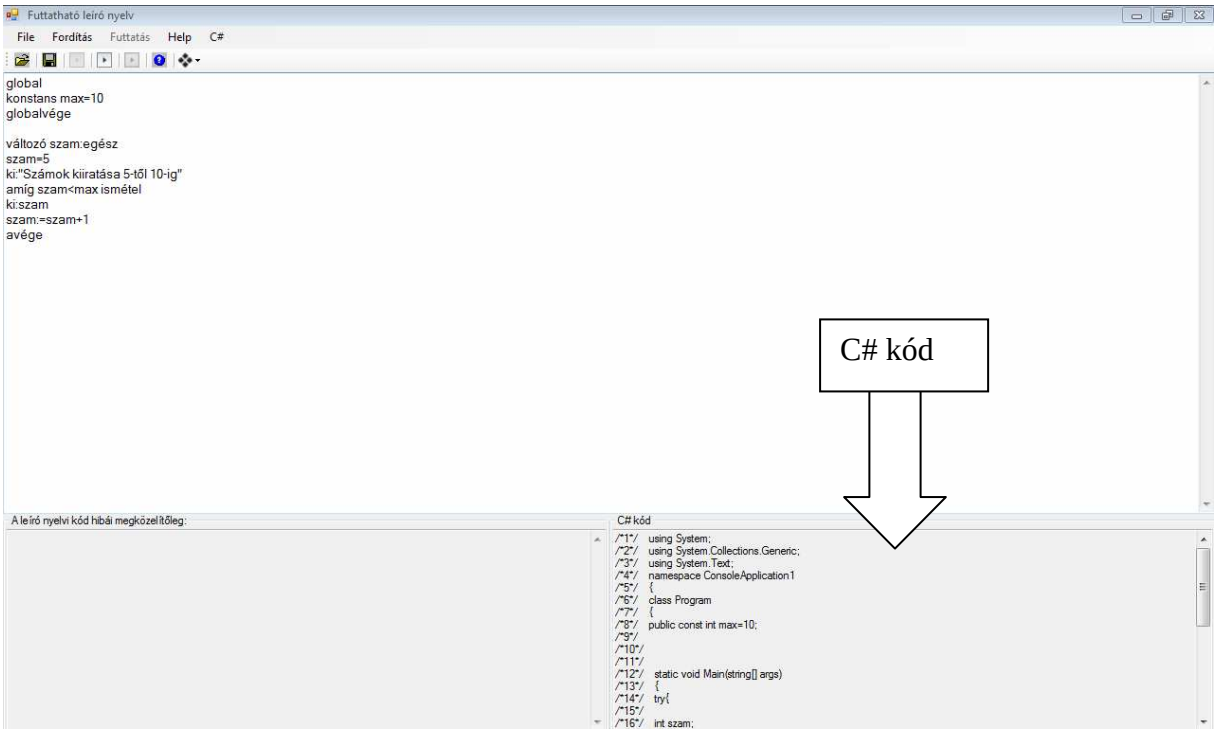
3. Végül az *„avége”* szót kicseréli a *„}”* karakterre

```
while( szam<max )
{
ki:szam
szam:=szam+1
}
```

A fenti lépések után következik a ciklusmag fordítása soronként. Itt az előzőekben ismertetett kiíró és értékadó utasítások cseréje fog lezajlani.

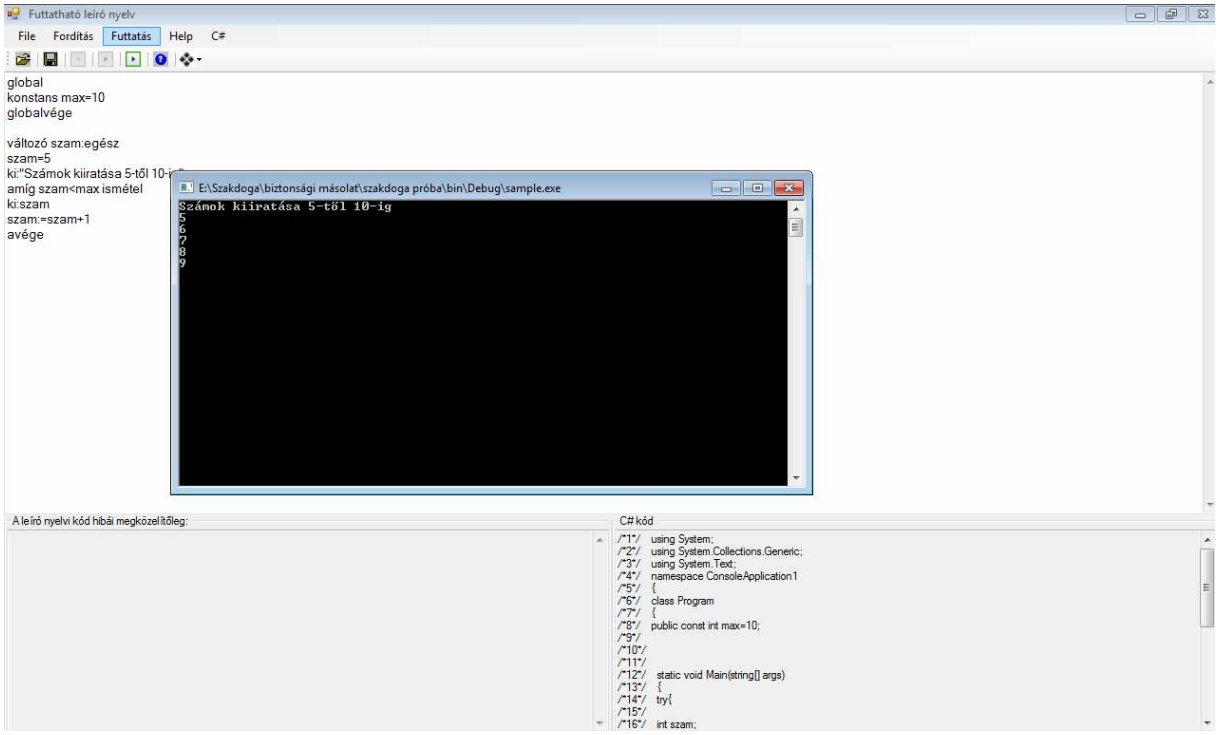
Ezután a program feltölti a C# kód szövegdobozt a lefordított kódsorokkal. (A C# alaposztály és Main függvény definiálása után.) Az így létrejövő kódsorokat a program kommentek beillesztésével sorszámozni fogja. A sorszámozást befejezve pedig elérhetővé teszi a C# kód fordítása gombot és menüt, miután inaktív állapotba helyezte a Leíró nyelvi kód fordítása menüket.

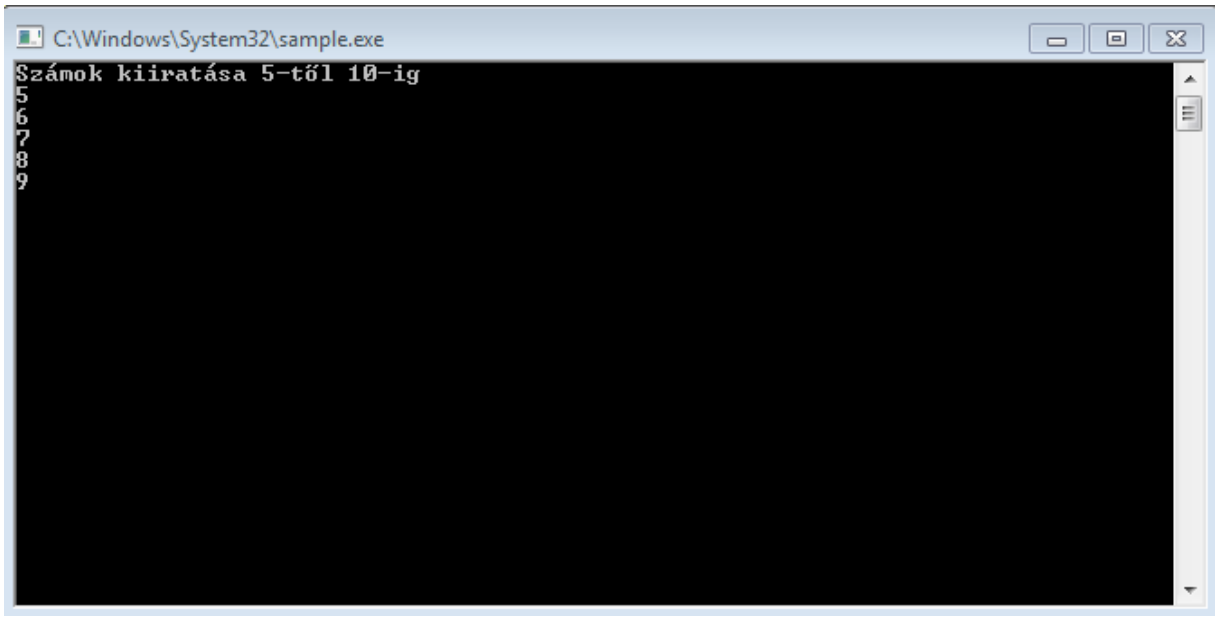
A C# menü – megjelenítés opcióját kiválasztva ekkor láthatóvá tehetjük a lefordított kódot, mely a C# kód szövegdobozban fog megjelenni:



A Fordítás menü – C# kód fordítása menüpontjának kiválasztásával a program megkezdi a lefordított kód hibaellenőrzését, meghívja a C# kód fordító objektumát és generál egy *sample.exe* nevű futtatható állományt. A jelenleg megnyitott kód esetében a fordító nem talál hibát, de ettől függetlenül a program végrehajtja a típuskényszerítésre vonatkozó felesleges sorok kiszűrését végző függvényt (Jelen kód esetében ez a függvény nem fog semmilyen módosítást végezni a kódon), majd újra meghívja a C# kód fordítását végző programsorokat. A program az újrafordítás előtt a típuskényszerítés ellenőrzését vizsgáló logikai változó értékét igazra állítja. Az újrafordítás során a fent említett lépések fognak végrehajtódni, az említett típuskényszerítés hibákat kiszűrő függvény alkalmazása nélkül.
Mivel a kód nem tartalmaz hibákat a program engedélyezni fogja a Futtatás gomb és menü használatát, miután inaktív állapotba helyezte a C# kód fordítása menüket.

Az előző műveletek után a Futtatás gombra vagy menüre kattintva elindíthatjuk a lefordított programot. Ekkor az előzőekben generált *sample.exe* fájl konzolablakban való futtatása fog megvalósulni.





A konzolos ablakban futó program futását az „*Enter”* billentyű leütésével tudjuk terminálni.

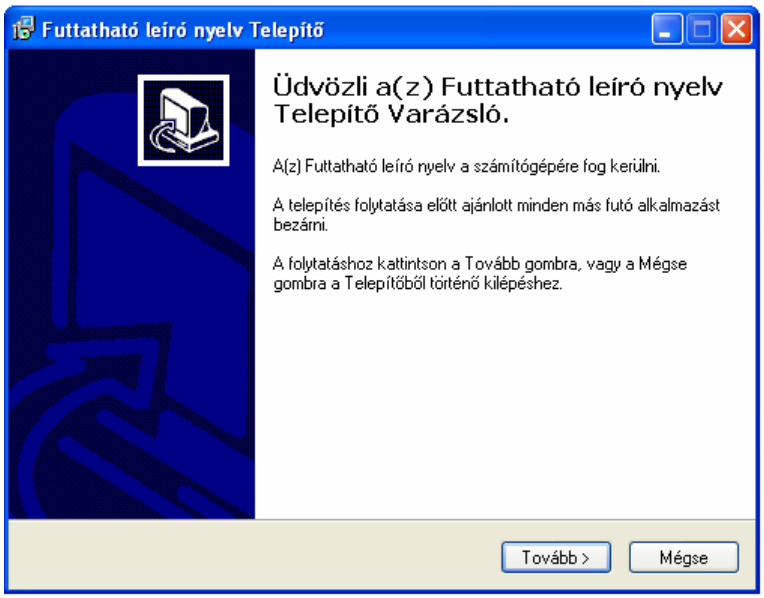
A program által generált C# kód:

```
/*1*/   using System;
/*2*/   using System.Collections.Generic;
/*3*/   using System.Text;
/*4*/   namespace ConsoleApplication1
/*5*/   {
/*6*/   class Program
/*7*/   {
/*8*/   public const int max=10;
/*9*/
/*10*/
/*11*/
/*12*/    static void Main(string[] args)
/*13*/    {
/*14*/    try{
/*15*/
/*16*/    int szam;
/*17*/    szam=5;
/*18*/    Console.WriteLine("Számok kiiratása 5-től 10-ig");
/*19*/    while( szam<max )
/*20*/    {
/*21*/    Console.WriteLine(szam);
/*22*/    szam=szam+1;
/*23*/    }
/*24*/
/*25*/
/*26*/
/*27*/
/*28*/    }catch(Exception e){Console.WriteLine("Hiba a futtatás során!");}
/*29*/    Console.ReadLine();
/*30*/    }
/*31*/    }
/*32*/    }
```

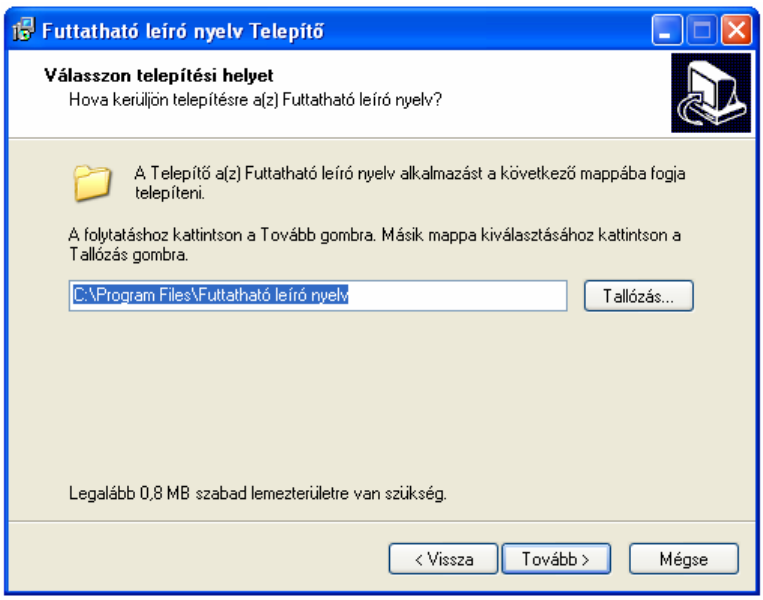
Felhasználói dokumentáció

Installálás, telepítés

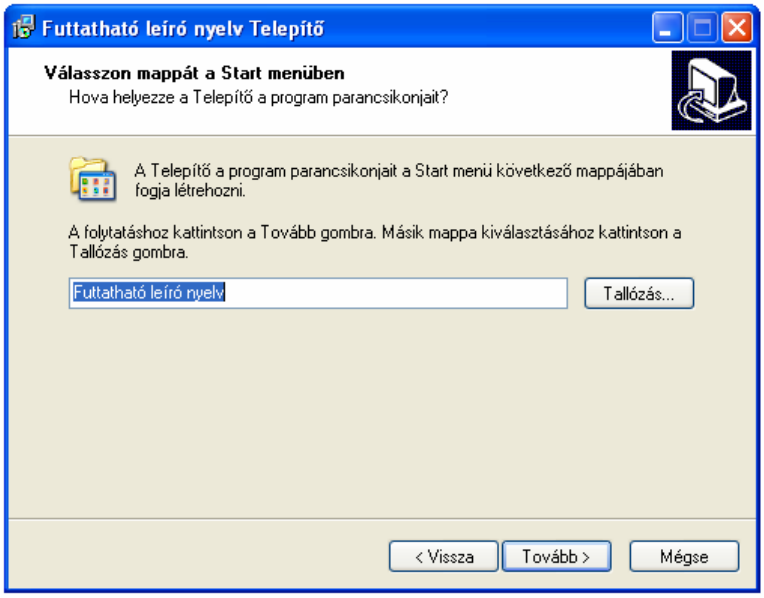
A szoftver telepítője általános módon üdvözli a felhasználót és érdeklődik a telepítésről.



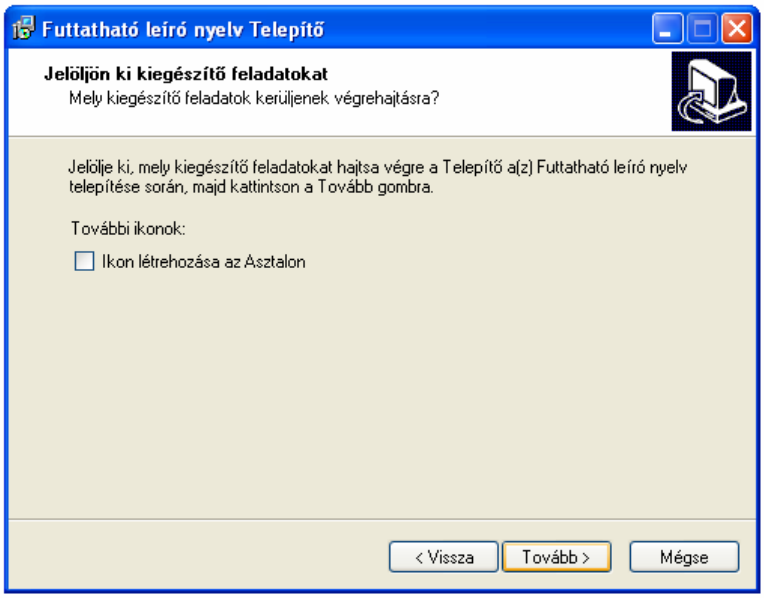
A Tovább gombra kattintva folytatódik a telepítés. A következő ablakban beállíthatjuk a program telepítésének helyét, ami alapértelmezetten: C:\Program Files\Futtatható leíró nyelv.



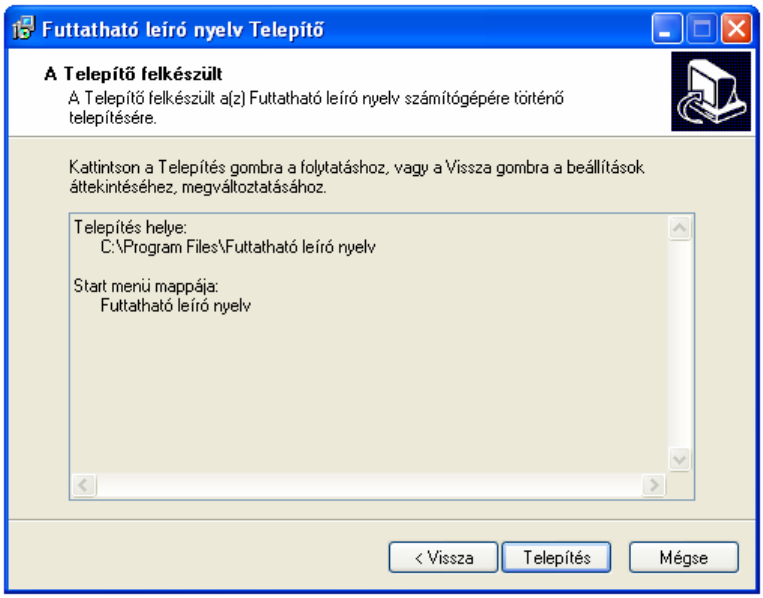
A következő lépésben a Start menübeli csoportot adhatjuk meg, ahova a telepítés után a programindító parancsikon kerülni fog.



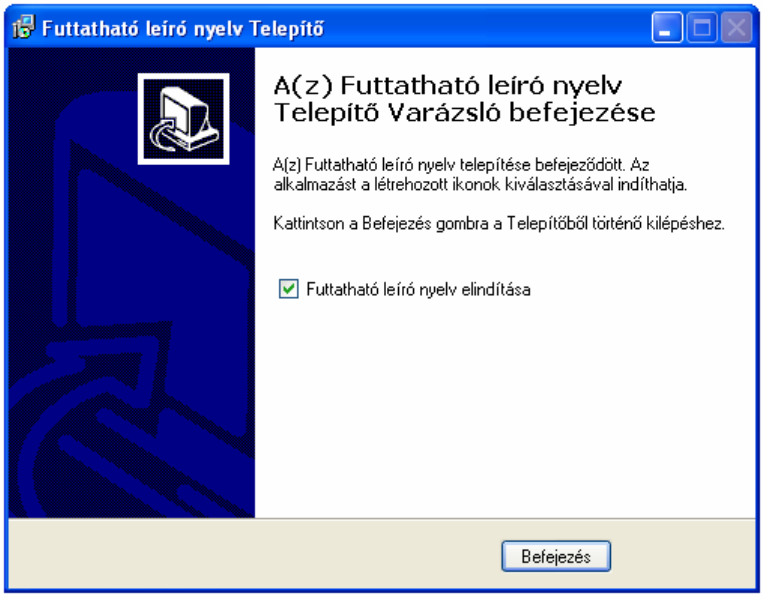
Ezután lehetőségünk van parancsikont elhelyezni az Asztalon.



A fájlok másolása előtti utolsó lépésben egy összegzést kapunk a beállításainkról. A Telepítés gombra kattintva elkezdődik a program állományainak másolása.

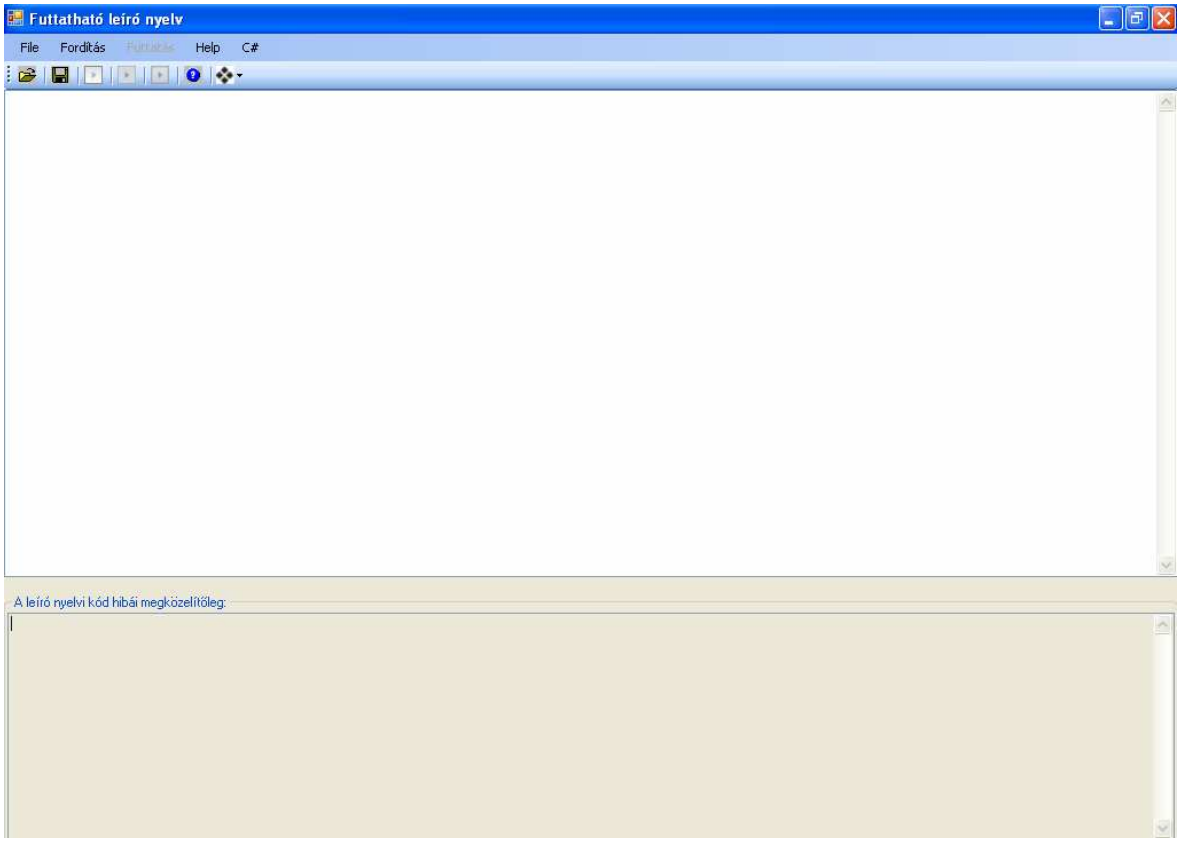


A telepítés befejezése után futtathatjuk is a programot.



A program használata

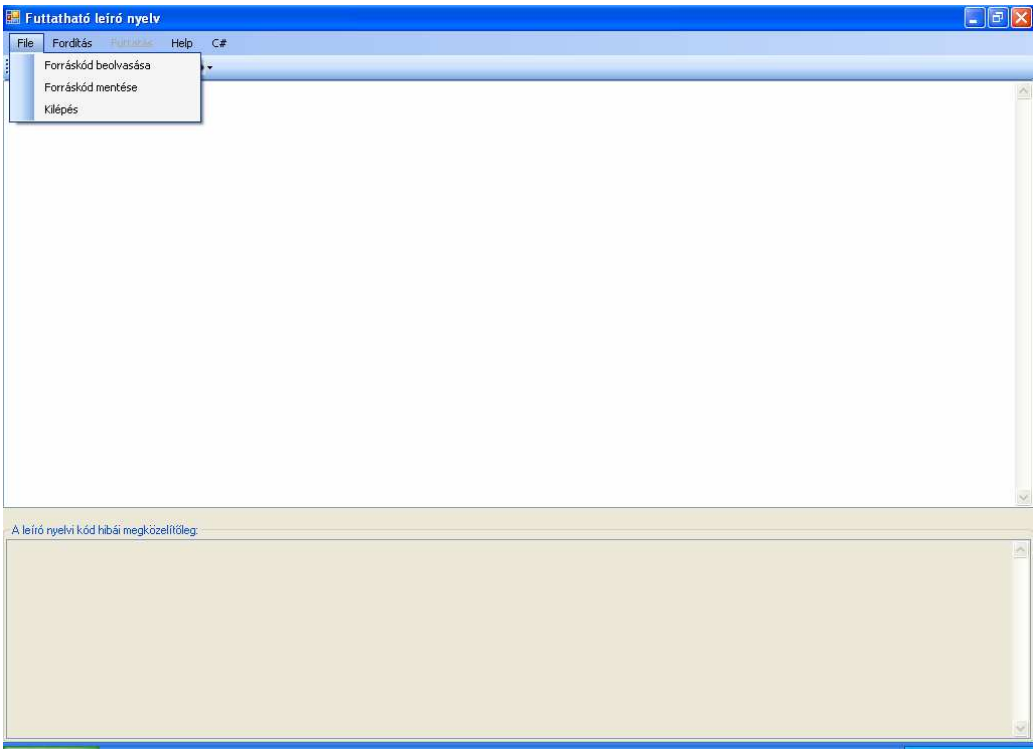
A program, indítás után a következő ablak jelenik meg:



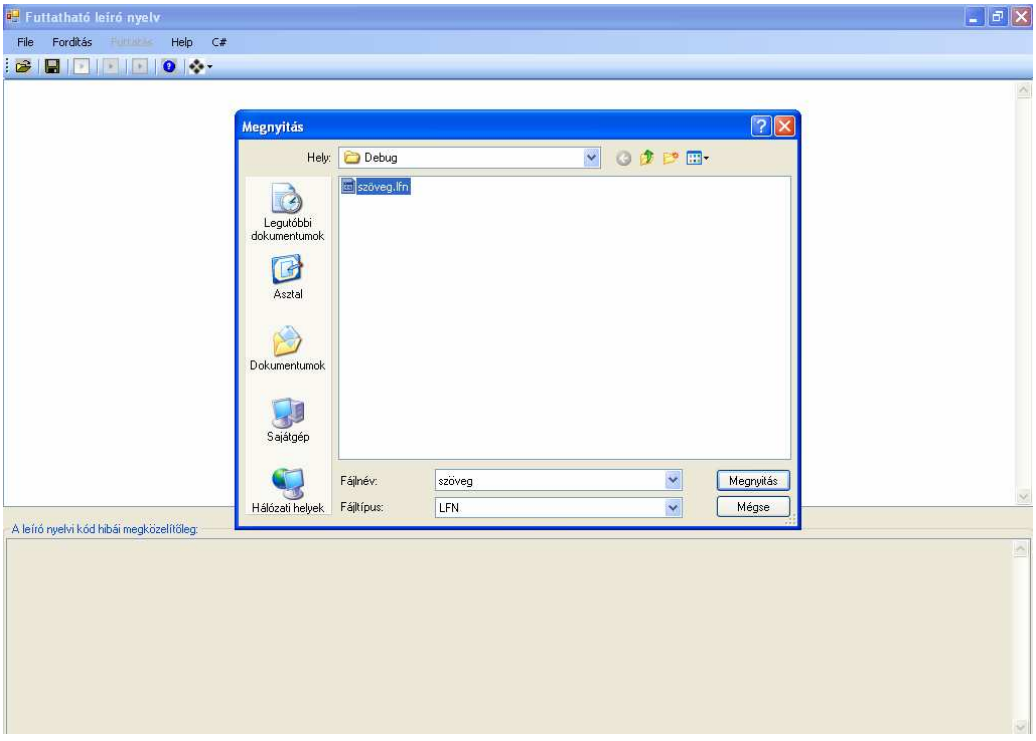
1. 0 ábra

Középen található a leíró nyelvi kód szerkesztőablaka, alatta jobb oldalon a lefordított C# kód, illetve baloldalon a leíró nyelvi forráskód hibáinak mezője. A felső sorban a Fájl, Fordítás, Futtatás, Help és C# kód menüpontok helyezkednek el.

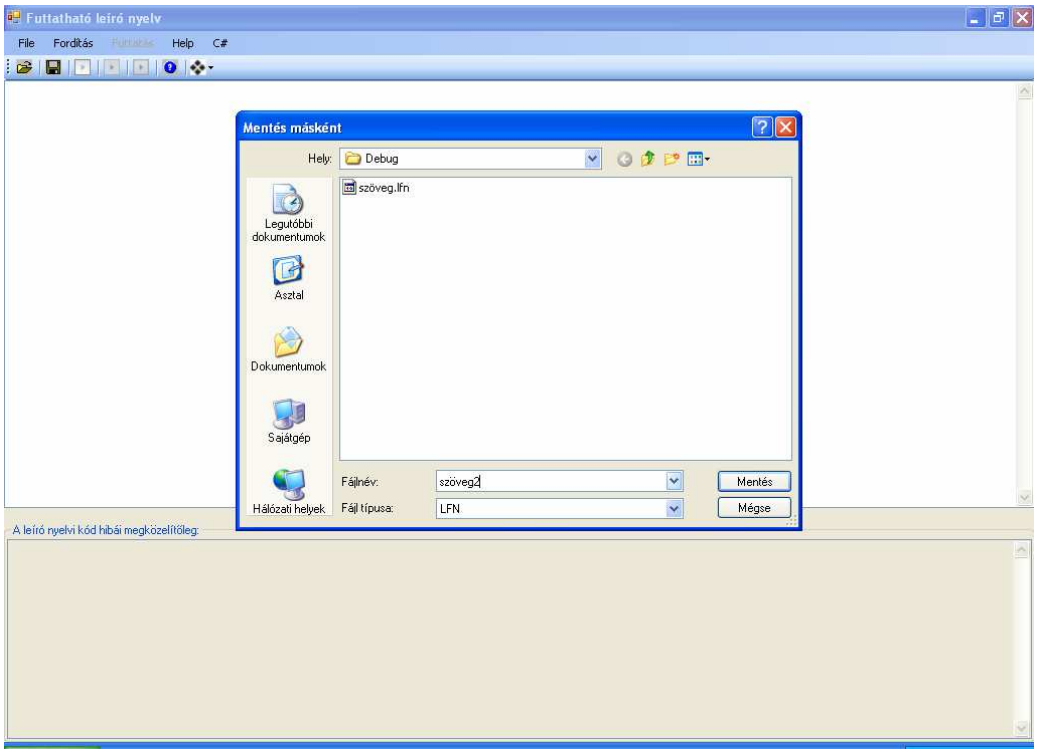
A Fájl menüben lehetőségünk van már mentett leíró nyelvi forráskódok beolvasására, a szerkesztőbe beírt forrás mentésére, illetve itt található a kilépés menüpont is. (1.1, 1.2, 1.3 ábra)



1. 1 ábra

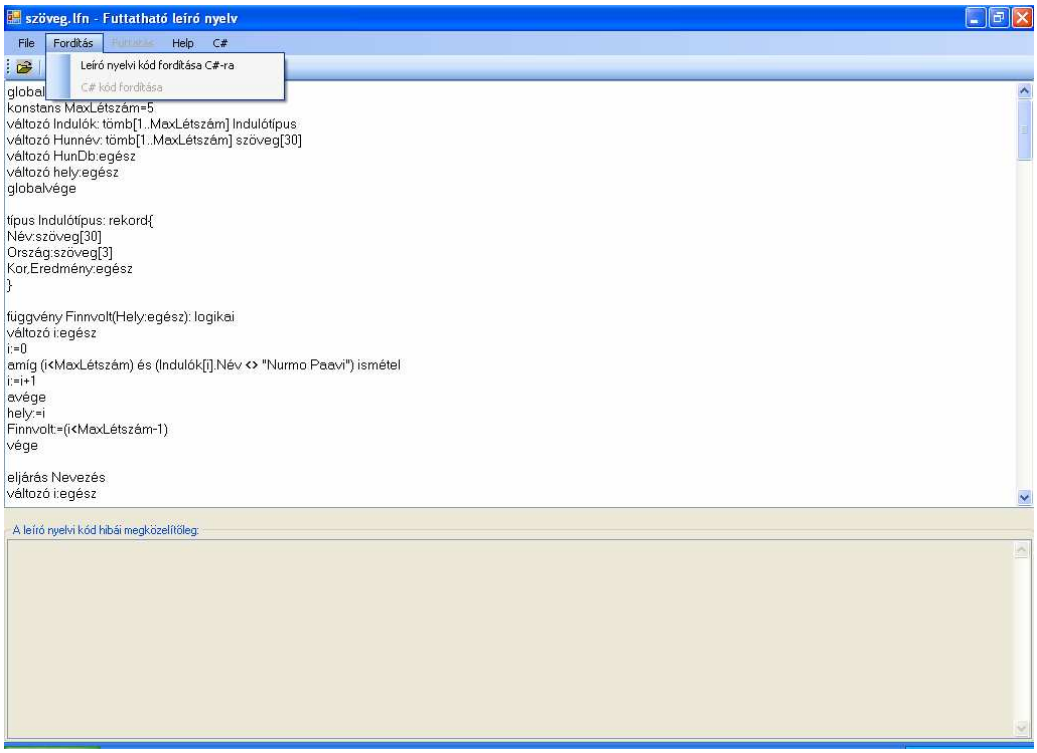


1. 2 ábra



1. 3 ábra

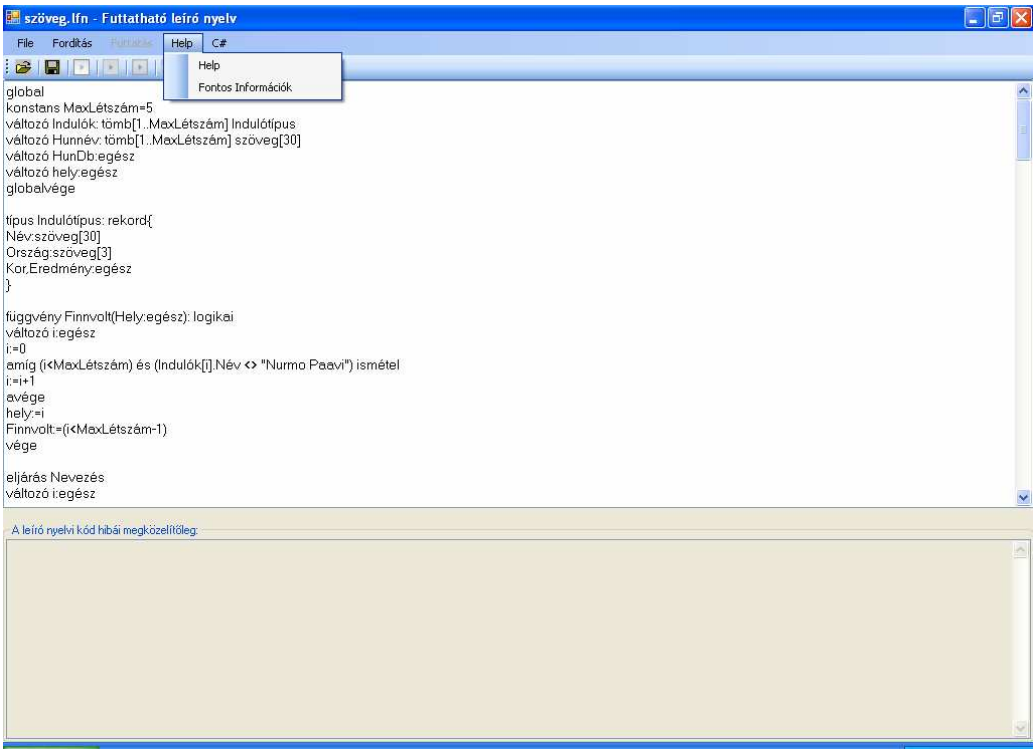
A Fordítás menüben a Leíró nyelvi kód fordítása C#-ra és a C# kód fordítása menüpontokat találhatjuk. (1.4 ábra) A Leíró nyelvi kód fordítása menüpont használatával a szerkesztőben lévő forrás lefordul C# kódra. Ekkor ez a menüpont inaktívvá válik és elérhető lesz a C# kód fordítása menüpont. A C# kód fordítása során kerül sor a programkód hibáinak vizsgálatára. Ha a C# kód valamilyen hibát tartalmaz, a program visszavezeti azt a leíró nyelvi kódra és kiírja a leíró nyelvi kód hibajegyzékbe. A C# kód fordításának végeztével inaktív állapotba kerül a C# kód fordítása menü. Ha a forrás hibátlannak bizonyul, aktivizálódik a Futtatás menü. Ellenkező esetben egyik fordítás gomb sem és a futtatás gomb sem lesz elérhető. Ekkor a forráskód módosításával újra aktivizálhatjuk a Leíró nyelvi kód fordítása menüt.
A Futtatás menüre kattintva elindul a lefordított programkód.



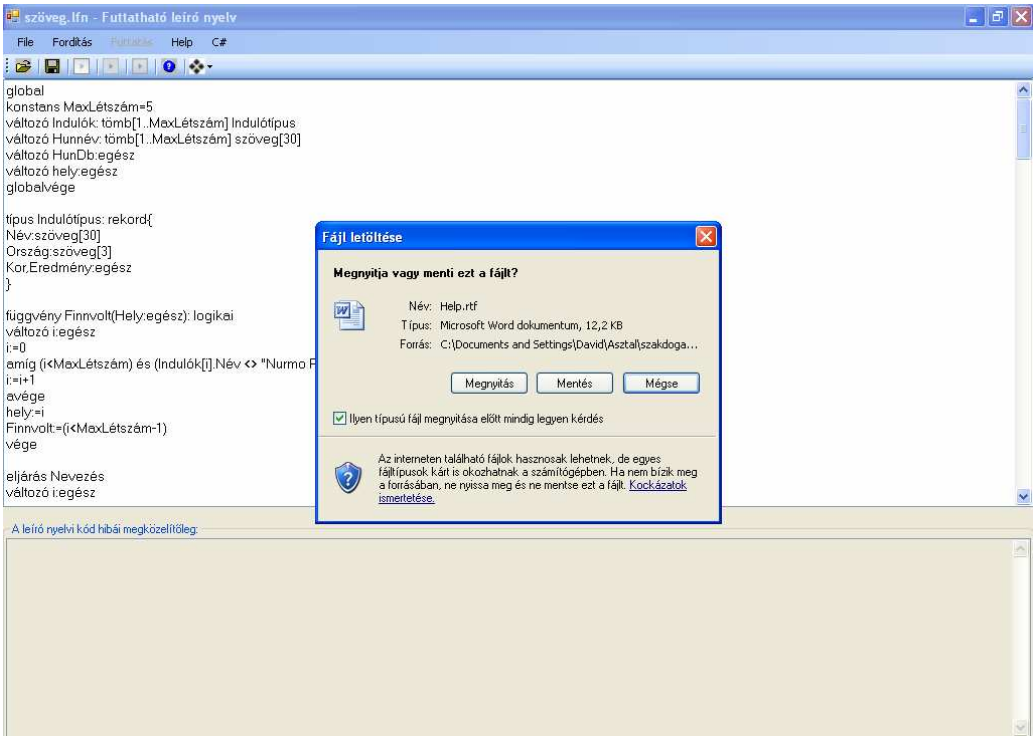
1. 4 ábra

A Help menü segítséget nyújt a program használatával kapcsolatban. (1.5 ábra) Itt a Help és a Fontos Információk menüpontokat találhatjuk. A Help almenü választásával a program Internet Explorer segítségével megnyitja a leíró nyelv szintaxisát tartalmazó Help.rtf fájlt. (1.6 ábra)

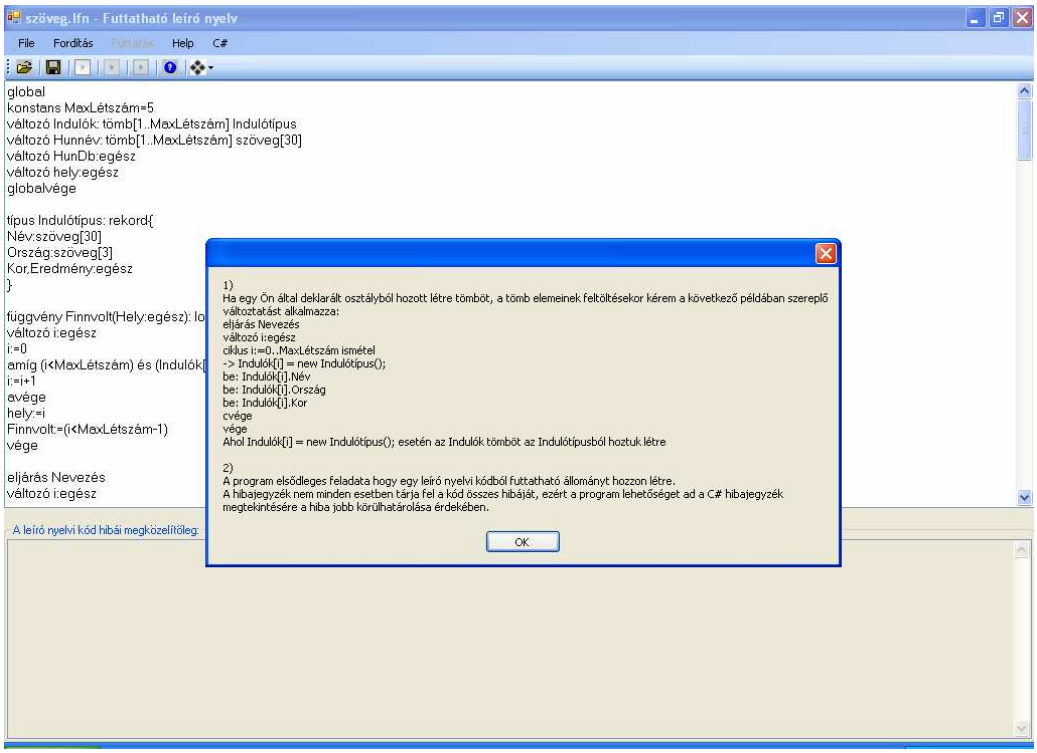
A Fontos Információk almenüben pedig tájékoztatást kaphatunk a program működéséről és egy használatbeli eset alkalmazásával kapcsolatban. (1.7 ábra)



1. 5 ábra



1. 6 ábra

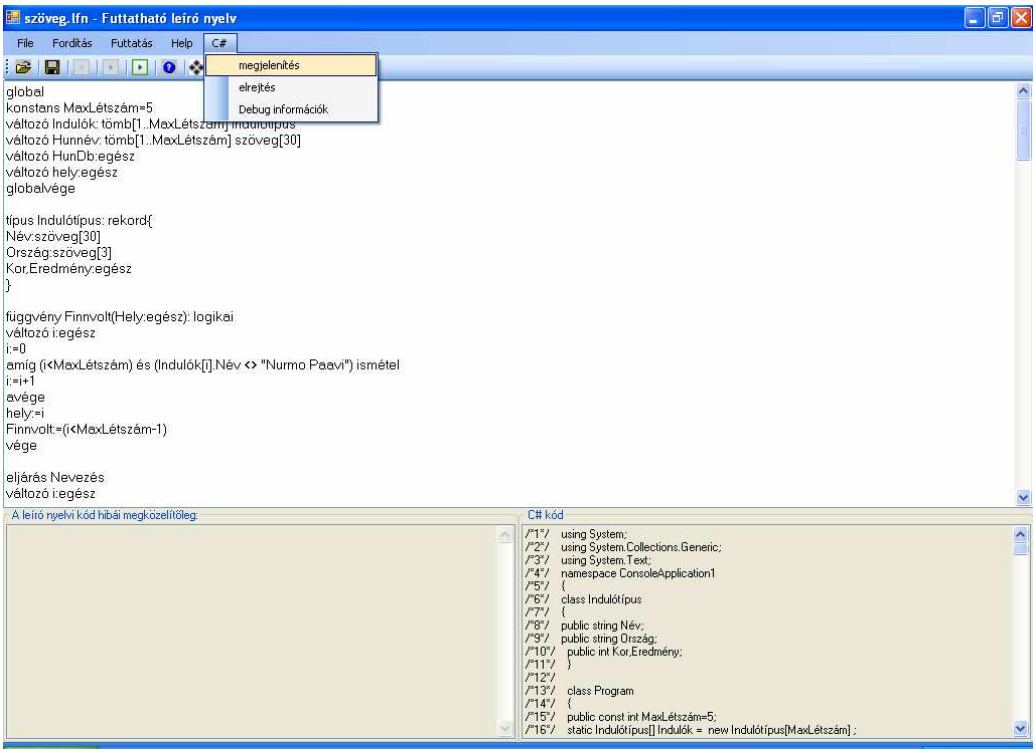


1. 7 ábra

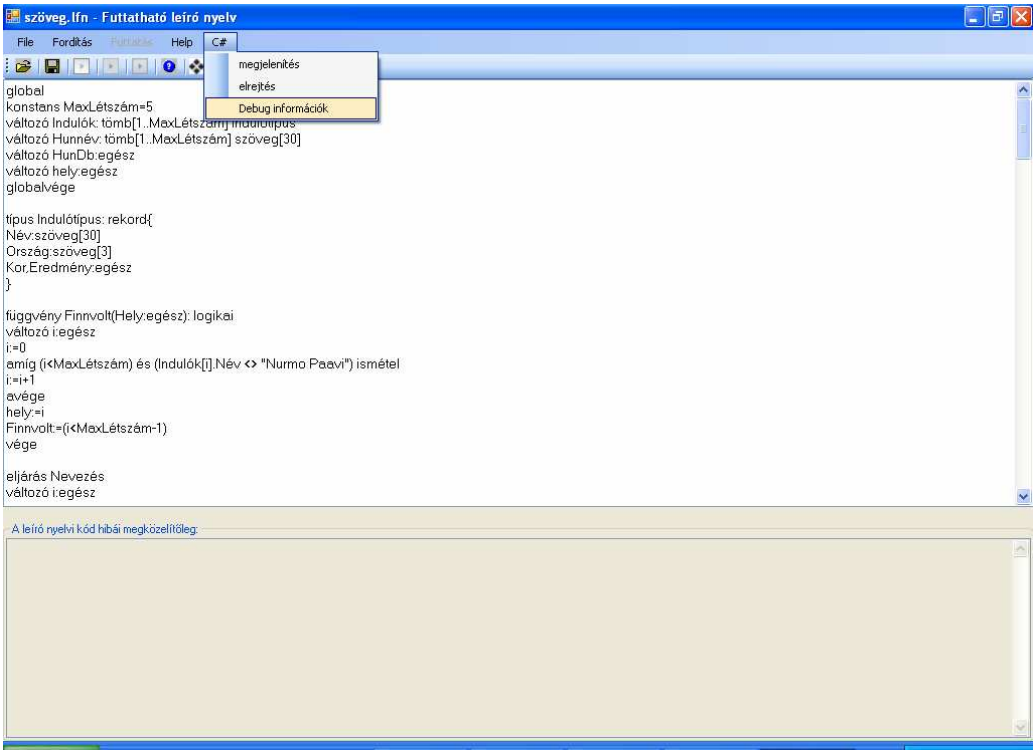
A C# kód menüben lehetőségünk van a C# kód megjelenítésére és elrejtésére, illetve a jobb hibaelemzés érdekében itt található a C# kód hibáit megjelenítő menü, a Debug. (1.8, 1.9, 1.10 ábra)

A legfontosabb menüpontok gyorsgombok formájában is elérhetők a következők alapján:

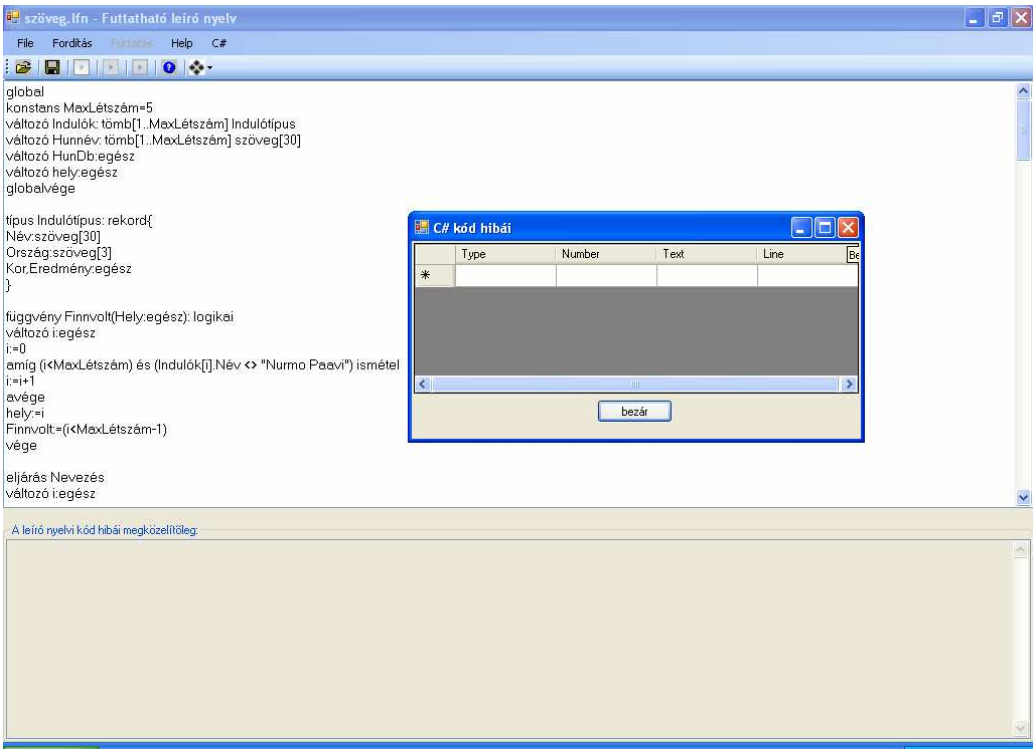
- - fájlmegnyitás – LFN típusú forráskódok megnyitása
- - fájlmentés – forráskód mentése LFN típusú fájlba
- - Leíró nyelvi kód fordítása C#-ra – leíró nyelvi forrás konvertálása C# kódra
- - C# kód fordítása – C# kód hibaellenőrzése
- - Futtatás – hibátlan kód esetén a forrás futtatása
- - Help – A Help.rtf fájl megtekintése, amely a leíró nyelv szintaxisát tartalmazza
- - C# kód megjelenítése, lenyíló sáv használatával



1. 8 ábra



1. 9 ábra


szöveg.lfn - Futtatható leíró nyelv
File
Fordítás
Futtatás
Help
C#
global
konstans MaxLétszám=5
változó Indulók: tömb[1..MaxLétszám] Indulótípus
változó Hunnév: tömb[1..MaxLétszám] szöveg[30]
változó HunDb:egész
változó hely:egész
globalvége
típus Indulótípus: rekord{
Név:szöveg[30]
Ország:szöveg[3]
Kor,Eredmény:egész
}
függvény Finnvolt(Hely:egész): logikai
változó i:egész
i:=0
amíg (i<MaxLétszám) és (Indulók[i].Név <> "Nurmo Paavi") ismétel
i:=i+1
avége
hely:=i
Finnvolt:=(i<MaxLétszám-1)
vége
eljárás Nevezés
változó i:egész
C# kód hibái
Type
Number
Text
Line
bezár
A leíró nyelvi kód hibái megközelítőleg:

1. 10 ábra

Összefoglalás

A program egy specifikus szintaxisú leíró nyelven írt forráskód fordítását és futtatását valósítja meg C# programnyelv segítségével, .NET környezetben. A leíró nyelvi kód fordítása során a program reguláris kifejezésekből álló mintákat alkalmaz a vezérlési utasítások fordításához. A vezérlési utasításokat (ciklusok, elágazások, függvények, eljárások, stb.) a program a fordítást megelőzően elválasztja egymástól, mely a szintaxis speciális felépítése miatt lehetséges és szükséges ezen utasítások a C# programnyelvnek megfelelő helyére való illesztéséhez. Az elválasztott vezérlési utasításokra ezután a program mintákat illeszt.
A mintaillesztés eredménye egy találati lista lesz, mely a mintára illeszkedő kódrészt fogja tartalmazni. A találati lista elemeit a program külön-külön fogja lefordítani, kiküszöbölve ezzel a program tesztelése során tapasztalt hibalehetőségeket. Az elválasztás és mintaillesztés után a kód fordítását a speciálisan ezekre az utasításokra kialakított függvények fogják elvégezni, reguláris kifejezések alapján történő cserék végrehajtásával. A fordítás ilyen szintű megvalósításához a leíró nyelv és C# programnyelv szintaxisbeli hasonlóságait használtam fel. A fordítási műveletek elvégzése után a program az elválasztott részeket a C# kódnak megfelelő helyére fogja pozícionálni és ezáltal a fordítás utolsó lépéseként létrejön a lefordított kód.

A lefordított kódon ezután hibakeresések fognak végbemenni a C# kód fordítása során.
A hibakeresés első részét a C# beépített fordítója fogja elvégezni, ezután egy általam készített hibakereső módszer fog végbemenni. Ezen módszer alkalmazása azért szükséges, mert a leíró nyelvi kód és a C# kód egyes részei nem fordíthatók kompatibilisen, szintaxisbeli különbségek miatt. (típuskényszerítés) A hibakeresés során az esetleges hibákat a program adattáblákba fogja kigyűjteni és egy speciálisan erre kialakított szövegdobozban megjeleníteni (C# kód hibái). A fordítási folyamat során a forrás és lefordított, elszeparált kódokat a program listákban fogja eltárolni, mely listák indexelése megegyezik és így a C# kódbeli hiba visszavezethető lesz a leíró nyelvi kódra.

Ha a forráskód a hibakeresés elvégzése után nem tartalmaz hibákat, lehetőségünk van futtatni a kódot.

A program a fordítás és futtatás funkciók mellett tartalmaz fájl megnyitási és mentési funkciókat is. A leíró nyelvi kód szerkesztése során a felhasználók segítségére szolgálhat a programból futtatható szintaxisleírás és pár fontos információ, melyet figyelembe kell venniük egyes utasítások készítése során.

A leíró nyelvi kód szerkesztése, fordítása és futtatása során a program biztosítja a lefordított kód megtekintését illetve a C# kód esetleges hibáinak táblázatos megjelenítését, ezzel is segítve a hibák javítását.

A dolgozatom témáját képező program nagyon hasznosnak bizonyulhat az algoritmus leíró nyelvet ismerő és C# programozási nyelvet elsajátítani akaró diákok körében.

A program továbbfejlesztésének lehetőségét a fordítás Java programozási nyelvre való kiterjesztésében látom.

Felhasznált irodalom:

1. Járdán Tamás – Pomaházi Sándor – Adatszerkezetek és Algoritmusok EKF Líceum Kiadó , Eger 2004

2. http://www.radsoftware.com.au/articles/regexlearnsyntax.aspx

3. http://www.softwareonline.hu – program futtatása programból cikk

4. Visual Studio 2005 Help – http://msdn2.microsoft.com/hu-hu/default(en-us).aspx

5. Lengyel Dávid – Futtatható leíró nyelv szakdolgozat