Visual Basic 2005 Express Edition

ELSŐ LÉPÉSEK

Összeállította: Juhász Tibor (2008)

NEM LEKTORÁLT VÁLTOZAT

A dokumentumban előforduló hibákat és egyéb megjegyzéseket kérjük a juhaszt@zmgzeg.sulinet.hu címen jelezni.

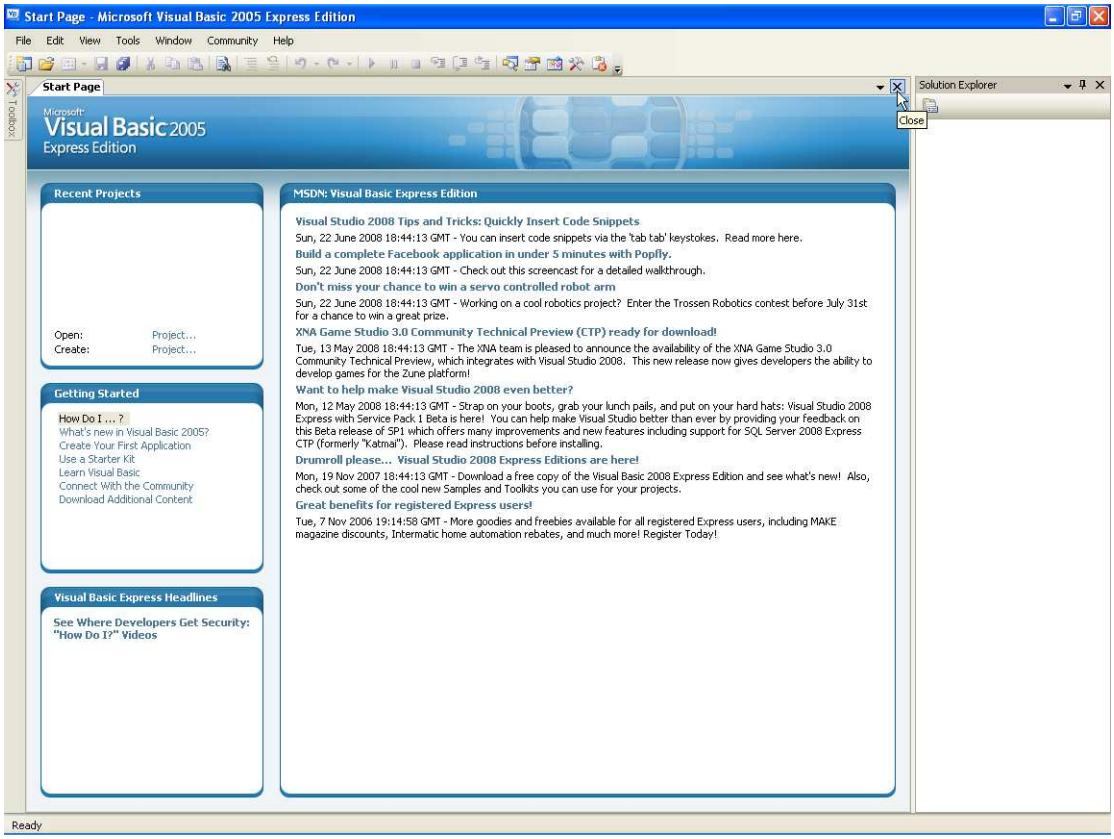
Programozás Visual Basicben

A Visual Basic letöltésére és telepítésére vonatkozóan lásd: *Telepítési útmutató*.

Az alábbiakban az integrált fejlesztői környezetre vonatkozó parancsoknál megadjuk a parancs helyét a menüben, majd zárójelben az eszköztár megfelelő ikonját, illetve a hozzá tartozó billentyűkombinációt. Az integrált fejlesztői környezet (Visual Studio) általános ismertetését és a használatára vonatkozó további tudnivalókat lásd: *A Visual Studio használata*.

A Visual Basic indítása és felhasználói felülete

A Visual Basicet a *Start*/*Minden program* menüjének vagy az *Asztal* parancsikonjának a segítségével indíthatjuk el. A megjelenő ablakban láthatjuk a Windowsban megszokott menüsort, eszköztárat, alul pedig az állapotsort.



A Visual Basic integrált fejlesztői környezete a startlappal

Az ablak jobb szélén helyezkedik el az egyelőre üres megoldástallózó (*Solution Explorer*), melyet a fájlok kezeléséhez használunk. A bal szélen találjuk az összecsukott eszközkészletet (*Toolbox*), amely akkor nyílik/zárul, ha az egeret a *Toolbox* felirat fölé visszük, illetve elvisszük onnan. Ezt a viselkedést a munkaablak *Auto hide* ikonjával módosíthatjuk. Fekvő gombostű ( ) esetén a nyitás/zárás az egérrel végezhető, míg álló gombostűnél ( ) folyamatosan nyitva marad az ablak. Az ikont megtaláljuk a többi munkaablak címsorában is.

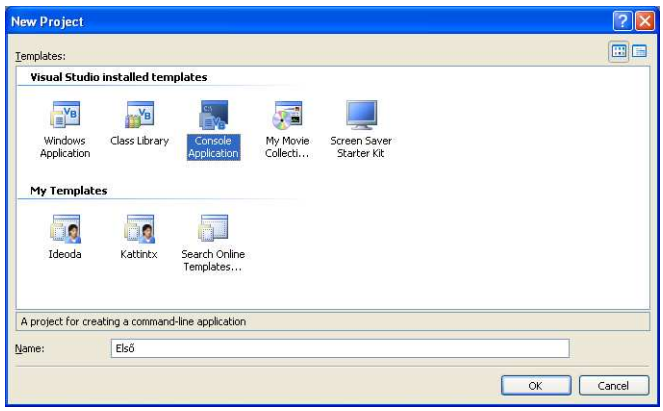
A Visual Basic indítása után a *Close* (×) gombbal zárjuk be a startlapot!

Visual Basic programok létrehozása

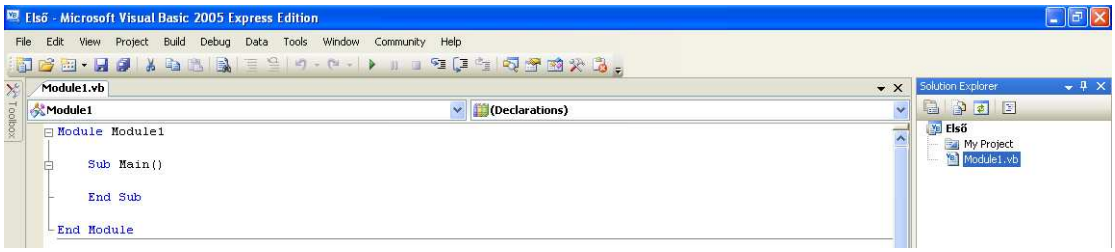
Az új program írását a *File* menü *New project* parancsával kezdjük (, *Ctrl+N*). A Visual Basic több, előre elkészített sablonnal segíti a programozást. A sablon a készülő programot keretbe foglaló utasításokat tartalmazza.

A megjelenő *New Project* ablakban látjuk a rendelkezésre álló sablonokat. A *Windows Application* sablon egy grafikus felületen futó alkalmazás, a *Console Application* pedig egy úgynevezett konzolalkalmazás megírását készíti elő. A konzolalkalmazások nem használják ki a Windows grafikus felhasználói felületét. A felhasználóval a parancssori ablakon[1] keresztül tartják a kapcsolatot. Először konzolalkalmazást fogunk készíteni, de hamarosan megismerkedünk a Windows-alkalmazások létrehozásával is.

Válasszuk ki a *Console Application* sablont, majd a *Name* szövegdobozba írjuk be a nevét (*Első*). Ha rákattintunk az OK gombra, létrejön első programunk.



Konzolalkalmazás létrehozása



Az első program a forráskóddal és a megoldástallózóval

A megoldástallózó alatt megjelenik a *Tulajdonságok* ablak (*Properties*), ezzel egyelőre nem foglalkozunk. A képernyő legnagyobb részét a kódszerkesztő ablak foglalja el, melyben láthatjuk a sablonból átvett néhány utasítást.

A Visual Basic programok szerkezetét részletesen *A Visual Studio használata* című dokumentumban ismertetjük. Most csak annyit jegyzünk meg, hogy a program utasításait egy eljárásba írjuk bele, melyet a *Sub Main() … End Sub* utasítások határolnak. Maga az eljárás pedig egy modul része, amely a *Module … End Module* utasítások között helyezkedik el. Bár a szerkezet bonyolultnak tűnik, eddig egyetlen karakter begépelésére sem volt szükség, mert ezt a keretet a fejlesztői környezet magától létrehozta!

Megfigyelhetjük, hogy a kódszerkesztő ablakban az utasítások kulcsszavai kék színnel és nagy kezdőbetűvel jelennek meg. **A Visual Basic nem különbözteti meg a kis- és nagybetűket a forráskódban. A kódszerkesztő a kulcsszavak kezdőbetűit átírja nagybetűre még akkor is, ha kisbetűvel gépeljük be.** Ezzel megkönnyíti a szintaktikus hibák keresését. A behúzások alkalmazása pedig áttekinthetővé teszi a forráskódot.

[1] A parancssori ablak a *Start*/*Minden program*/*Kellékek* menüből indítható, de most nem kell megnyitnunk.

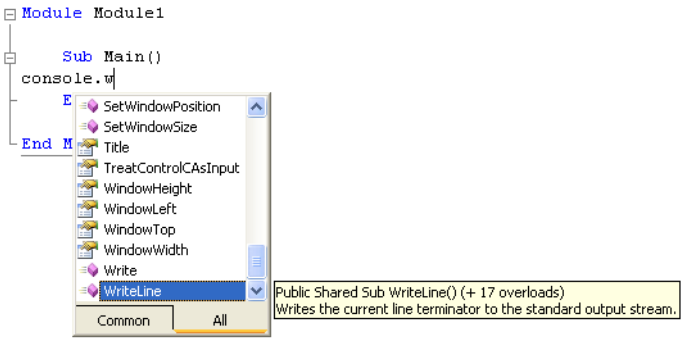
Első Visual Basic programunk

A sablon csak a forráskód kereteit biztosítja a számukra. A program további utasításait nekünk kell begépelnünk.

Első programunk írjon ki egy üzenetet a képernyőre. A program futásakor megjelenő ablakba a *console.writeline* utasítással[2] írhatjuk ki az utána szereplő, zárójelben és idézőjelek között álló szöveget. Kattintsunk a *Sub* és *End Sub* utasítások közötti üres sorra, majd a sor elejétől indulva kezdjük el gépelni a következő utasítást:

```
console.writeline("Ez az első program.")
```

Amint lenyomtuk a *console*-t követő pontot, pillanatnyi szünet után megjelenik egy lista és egy magyarázat a kiválasztott listaelemhez:



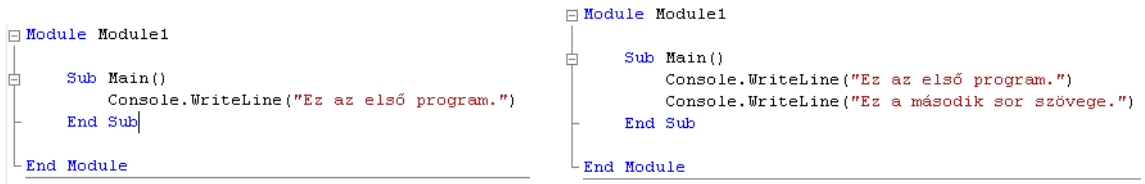
Az intelligens segítség a forráskód begépelésénél

A forráskód begépelésénél megjelenő intelligens segítség (*IntelliSense*) listája a pontot megelőző azonosító lehetséges folytatásait sorolja fel.[3] A listában a gördítősávval vagy a kurzormozgató billentyűkkel mozoghatunk. A keresést meggyorsítja, ha lenyomjuk a keresett szó első karakterét, esetünkben a w-t. Az intelligens segítség egyből a *WriteLine*-ra áll.

A *WriteLine* kiválasztása után a következő karakter, azaz a kezdőzárójel begépelésével folytassuk a forráskód szerkesztését! A kódszerkesztő automatikusan elhelyezi a forráskódban a *WriteLine* utasítást, és eltűnik a lista. A forráskód elkészítését az idézőjeles karaktersorozat, majd a végzárójel beírásával zárjuk. A kódszerkesztő az idézőjelbe tett karaktersorozatot barna színnel jelzi. Közben az intelligens segítség folyamatosan megjeleníti az adott utasításra vonatkozó rövid, angol nyelvű súgót.

A végzárójel begépelése után lépjünk a kurzorral egy másik sorba! A kódszerkesztő elvégzi helyettünk a behúzás megfelelő kialakítását. Lépjünk vissza az előző sorba, majd nyomjuk le az *Entert*! A létrejövő új sor behúzása megfelel az előző sornak. Gépeljünk be a fenti módon még egy utasítást:

```
console.writeline("Ez a második sor szövege.")
```



A készülő program forráskódja

A folytatás előtt mentsük el a programot! Ehhez válasszuk a *File* menü *Save all* parancsát (, *Ctrl+Shift+S*). Hagyjuk meg a felajánlott neveket (*Name*, *Solution name*). A *Location* szövegdobozban a *Browse* gomb segítségével válasszuk ki azt a mappát, ahová programjainkat menteni fog-

2 Valójában a *Console* objektum *WriteLine* metódusát hívjuk meg.

3 Az *IntelliSense* valójában a *Console* objektum attribútumait és metódusait listázza ki. Lásd később.

juk. Figyeljünk arra, hogy a programhoz új mappa jöjjön létre (*Create directory for solution* jelölőnégyzet). A mentéshez kattintsunk a *Save* gombra.

A program futtatása – várakozás az ablak bezárására

Itt az ideje, hogy elindítsuk programunkat. A fejlesztői környezeten belül ezt a *Debug*/*Start Debugging* paranccsal (, *F5* funkcióbillentyű) tehetjük meg. Így a programot az úgynevezett hibakereső üzemmódban futtatjuk.

Hálózati környezetben végzett munka esetén előfordulhat, hogy nincs elegendő jogunk a hibakereső üzemmódban történő futtatáshoz. Ekkor használjuk a *Debug*/*Start without Debugging* (*Ctrl+F5*) parancsot (futtatás hibakeresés nélkül).

A program indítása után egy pillanatra feltűnik előttünk a fekete hátterű parancssori ablak, de azonnal be is zárul. Egyszerűbb esetben egy újabb utasítás, a *Console.ReadLine*[4] alkalmazásával késleltethetjük az ablak bezárását. A *ReadLine* hatására csak az *Enter* lenyomása után zárul be az ablak.

Gépeljük be a *console*-t a *Console.WriteLine* utasításokat követő újabb üres sorba, majd írjuk be a pontot, és keressük meg az intelligens segítség listájában a *ReadLine* utasítást! Mivel most nem folytatjuk a gépelést, **a tabulátor billentyű lenyomásával illesztjük be az utasítást a listából a forráskódba**. Ha átlépünk egy másik sorba, akkor a kódszerkesztő átírja a kezdő kisbetűket nagybetűkre, és kiteszi az utasítás végére az üres zárójelpárt:

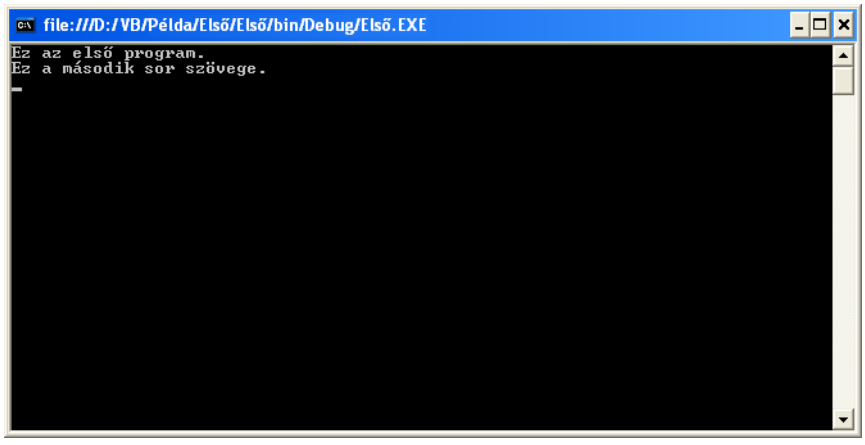
```
Module Module1

    Sub Main()
        Console.WriteLine("Ez az első program.")
        Console.WriteLine("Ez a második sor szövege.")
        Console.ReadLine()
    End Sub

End Module
```

A ReadLine utasítással kiegészített program

Mentsük a megoldást, és futtassuk a programot! Megjelenik előttünk a parancssori ablak a két sor szövegével. Megfigyelhetjük, hogy a *WriteLine* utasítás a karaktersorozat kiírása után sort emel.



Első programunk a parancssori ablakban

Jegyezzük meg az *exe* fájl helyét a háttértáron, ami az ablak címsorában látható! Bár a teljes elérési út függ a mentésnél megadott helytől, a fájl mindig a *Debug* nevű almappában található.

Az *Enter* lenyomásával zárjuk be az ablakot. Így visszatérünk a fejlesztői környezethez.

[4] A *Console* objektum *ReadLine* metódusát hívjuk meg.

Megjegyzések a forráskódban

A forráskódot célszerű magyarázatokkal, megjegyzésekkel ellátni. A megjegyzések elősegítik az egyre növekvő terjedelmű programok áttekintését, későbbi értelmezését, módosítását.

A Visual Basic programok forráskódjába egy aposztrófjel (') után írhatunk megjegyzést. A megjegyzés az aposztrófjeltől a sor végéig tart. Ezt a karaktersorozatot a fordítóprogram nem veszi figyelembe. A megjegyzés bárhol állhat a forráskódban, tehát elhelyezhetjük az eljáráson vagy akár a modulon kívülre. A megjegyzés kezdődhet egy sor elején vagy a sorban lévő utasítások után.

Egészítsük ki megjegyzésekkel első programunk forráskódját:

```
' Első Visual Basic programunk (Első mappa)
Module Module1
    ' A program fő-eljárása
    Sub Main()
        Console.WriteLine("Ez az első program.") ' sor kiírása
        Console.WriteLine("Ez a második sor szövege.")
        Console.ReadLine()
    End Sub
End Module
```

Az első program forráskódja (Első mappa)

Mentsük, majd futtassuk a programot!

A program lefordítása

Mindeddig csak hibakereső üzemmódban, a fejlesztői környezet segítségével futtattuk a programot. A fejlesztői környezettől független program készítéséhez le kell fordítanunk a forráskódot gépi kódra (illetve a .NET köztes kódjára). A fordítást a *Build*/*Build* parancs segítségével végezhetjük el. A fordítás után az *.exe* fájl a mentésnél megadott *Solution name**Name* mappa *bin**Release* almappájába[5] kerül. A fájlt az Intéző segítségével a szokásos módon futtathatjuk (például dupla kattintással).

A *Release* mappában lévő *.exe* programot akkor is elindíthatjuk, ha a hálózati korlátozások miatt nem használhatjuk a hibakereső üzemmódban történő futtatást.

A *Build* menü *Publish* parancsával telepítőkészletet készíthetünk az alkalmazásunkhoz. Ezzel a lehetőséggel itt nem foglalkozunk.

[5] A *Release* mappát a *Debug* mappa mellett találjuk. A projekt mappaszerkezetét *A Visual Studio használata* című dokumentum ismerteti.

Hibajavítás

Hibák a forráskódban

A kódszerkesztő aláhúzással jelöli a forráskódban előforduló, elsősorban szintaktikai hibákat. Az alábbi forráskódban például lehagytuk az első sor elejéről a megjegyzésre utaló aposztrófjelet. Ezért a fordítóprogram nem tudta azonosítani az *Első* szót. A második sorban pedig a *Module* kulcsszó végéről hiányzik az „e” betű. Állapítsuk meg, mi a hiba a negyedik sorban!

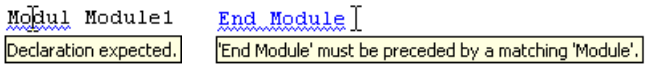
```
Első Visual Basic programunk (Első mappa)
Modul Module1
' A program fő-eljárása
Sub Man()
    Console.WriteLn("Ez az első program.") ' sor kiírása
    Console.WriteLine("Ez a második sor szövege.")
    Console.ReadLine()
End Sub

End Module
```

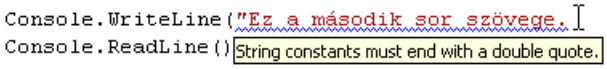
Hibák a forráskódban

Figyeljünk arra, hogy egy-egy hibának további következményei lehetnek. A program végén az *End Module* utasítás azért minősül hibásnak, mert az előző szintaktikai hiba miatt nem előzi meg egy *Module* kulcsszó.

Ha az aláhúzott rész fölé visszük az egérkurzort, akkor megjelenik a hiba angol nyelvű magyarázata. Kezdő programozóként azonban nem mindig tudunk következtetni a szövegből a hiba okára. Az *End Module* magyarázata valóban a *Module* hiányára utal, de a *Modul* hibánál megjelenő szöveg nem a szintaktikus hibát jelzi:

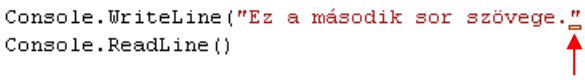


A hibák magyarázata

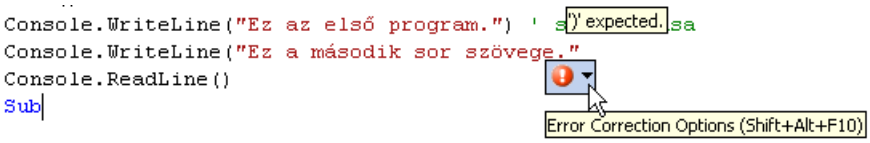


Gyakori hiba a karaktersorozat záró idézőjelének elhagyása

Ha a hiba helye alatt egy kis téglalapot látunk (intelligens címke), akkor az egérkurzor föléje helyezésével megjelenik a javításra vonatkozó angol nyelvű javaslat:

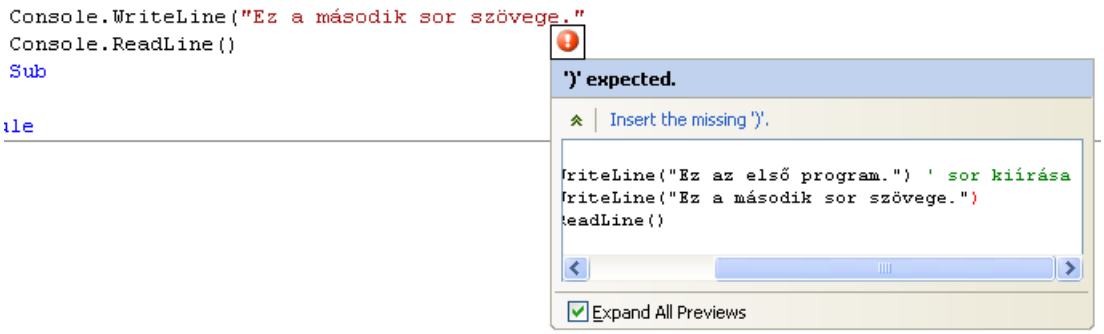


A hibára utaló téglalap



Javaslat a hiba javítására: ')' expected

A legördülő menü megnyitásával részletesebb magyarázatot kapunk (sajnos szintén angolul):

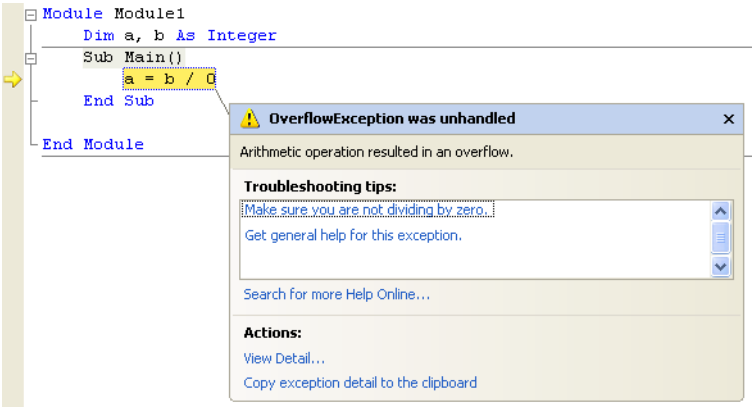


A hiba részletes magyarázata

Ha rákattintunk az *Insert the missing ')'* javaslatra, akkor a kódszerkesztő elvégzi a hiba kijavítását.

Futási hibák

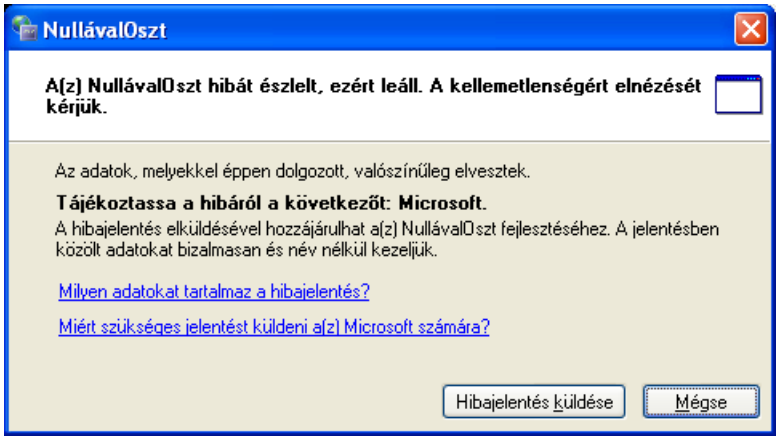
Ha a programot a fejlesztői környezetből futtatjuk, és a futás során következik be valamilyen hiba, akkor a vezérlés visszatér a kódszerkesztő ablakhoz, amelyben megjelenik a hiba helye és magyarázata. **A forráskód javítása előtt a *Debug*/*Stop Debugging* (, *Ctrl*+*Alt*+*Break*) parancscsal lépjünk ki ebből az úgynevezett nyomkövető-hibakereső üzemmódból!**



Futási hiba (nullával való osztás) megjelenítése

Hibás program futtatása esetén a kódszerkesztő alatt megjelenik a hibák listája (*Error List*). Itt is előfordulhat, hogy egyetlen hiba további hibákat generál.

Ha nincs elegendő jogunk a hibakereső üzemmódban történő programindításhoz, akkor a *Build*/*Build* parancs segítségével fordítsuk le a forráskódot, majd futtassuk a *bin\Release* mappába kerülő *.exe* fájlt. Ebben az esetben az operációs rendszer jelzi a futási hibát.



Futási hiba a lefordított program futtatásakor

Windows-alkalmazás létrehozása

A Windows-alkalmazás létrehozása előtt olvassuk el az Objektumok és események *című tájékoztatót, amely összefoglalja az objektumokkal, eseménykezeléssel kapcsolatos legfontosabb tudnivalókat.*

A konzolalkalmazások algoritmus-vezérelt programok. A Windows-alkalmazásokat általában események vezérlik. Egyszerűbb esetben az adatokat szövegdobozokba írjuk be, a feldolgozást, futtatást parancsgombokkal irányítjuk, az eredményeket címkeobjektumokkal jelenítjük meg.

Egy eseményvezérelt program esetén kilépés és újraindítás nélkül módosíthatjuk az adatokat, majd a parancsgombokkal ismét elvégezhetjük a feldolgozást. Ez nagyon kényelmessé teszi a program használatát.

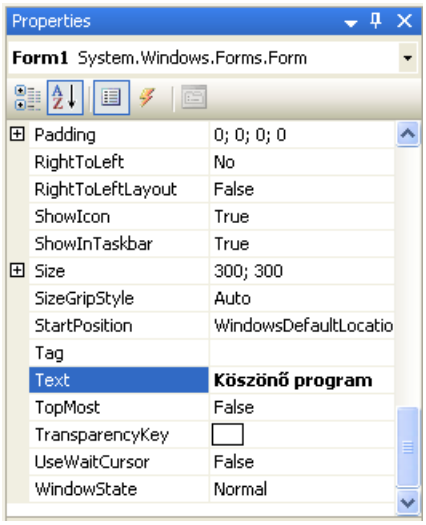
Az ablak létrehozása és tulajdonságainak módosítása

Első programunk egy szövegdobozt és egy parancsgombot fog tartalmazni. Ha a felhasználó beírja a nevét a szövegdobozba, majd rákattint a parancsgombra, akkor a program kiír a képernyőre egy üdvözlő szöveget.

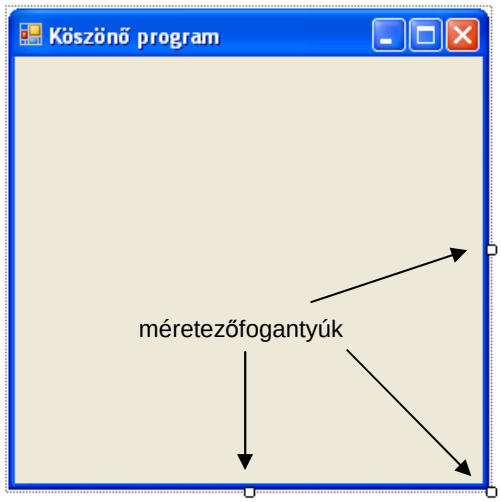
Windows-alkalmazás létrehozásához a *File/New project* ablakban válasszuk a *Windows Application* sablont! Írjuk be a projekt nevét: *Köszön*, majd kattintsunk az OK-gombra.

Létrejön egy projekt a *Form1* nevű űrlappal, amely a képernyőn megjelenő ablakot jelképezi. A forráskód helyett most az úgynevezett tervezőablakot (*Design*) látjuk, ebben foglal helyet a készülő ablak (űrlap[6]).

Kattintsunk rá az űrlapra. A *Properties* (tulajdonságok) munkalapon megkapjuk az űrlapobjektum tulajdonságainak listáját. Keressük meg a *Text* (szöveg) tulajdonságot, ami az ablak címsorának a szövegét határozza meg. Írjuk be a *Köszönő program* címet. Ha lenyomjuk az Entert vagy az ablak más területére kattintunk, máris megjelenik az űrlap címsorában a begépelt szöveg.



Az űrlap tulajdonságai



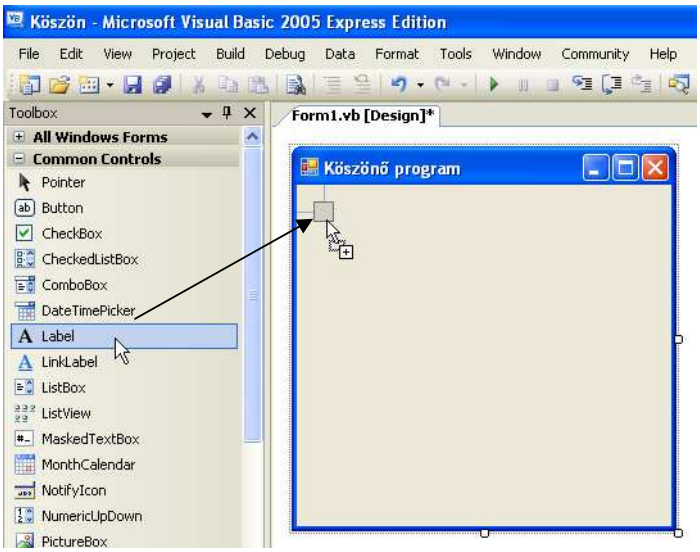
A készülő űrlap a módosított címmel

Szükség esetén további tulajdonságokat is módosíthatunk. A *BackColor* például a háttérszínt, a *Size* a méretet jelenti pixelben. Ez utóbbit a méretezőfogantyúk segítségével a tervezőablakban is megváltoztathatjuk. A (*Name*) tulajdonság az űrlapobjektum azonosítóját adja meg. Több ablak létrehozásánál célszerű ezt is átírni, az ablak funkciójára utaló nevet választani.

[6] Nem tartjuk szerencsésnek az ablakra vonatkozó *űrlap* elnevezést, de elterjedtsége miatt mi is ezt fogjuk használni.

Objektumok az űrlapon

A *Label* (címke) objektum segítségével írhatunk szöveget az ablakba. Az objektumot az eszközkészletben (*Toolbox*) találjuk. Fogjuk meg az egérrel a *Label* ikonját, és helyezzük el az űrlap bal felső részére. (Hasonló eredményt érhetünk el, ha duplán kattintunk az ikonra.) Megjelenik az űrlapon a címke.



Címke elhelyezése az űrlapon

Kattintsunk a címkére, majd a tulajdonságok ablakban módosítsuk a *Text* tulajdonságot: *Írd be a neved, majd kattints az OK-gombra!* Vegyük észre, hogy a tulajdonságok ablak mindig a kijelölt objektum jellemzőit mutatja. A létrehozott objektumok között az ablak felső részén lévő legördülő lista segítségével is válogathatunk.

Helyezzünk el az ablakban egy *TextBox* (szövegdoboz) objektumot, melynek töröljük a *Text* tulajdonságát. A szövegdoboz mellé pedig tegyünk egy *Button* (parancsgomb) objektumot. Írjuk át a parancsgomb feliratát OK-ra (*Text* tulajdonság).

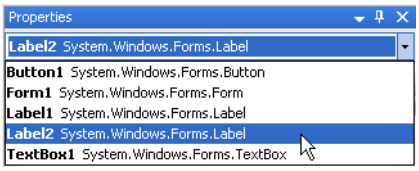
A készülő ablak

Végezetül tegyünk a szövegdoboz alá még egy címkeobjektumot, melynek töröljük a *Text* tulajdonságát, majd mentsük a projektet! A projekt mentése során a fejlesztői környezet létrehozza az objektumok definícióit tartalmazó szövegfájlt, amit *Form1.Designer.vb* néven találunk meg a projekt mappájában. A szövegfájlt a Jegyzettömbbel is megnyithatjuk.

Mielőtt továbblépnénk, a tervezőablakban kattintsunk az egyes objektumokra, és a tulajdonságok ablakban tekintsük meg azonosítóikat (*Name* tulajdonság a lista elején). A szövegdoboz a *TextBox1*, a parancsgomb a *Button1*, a két címkeobjektum pedig a *Label1*, illetve *Label2* nevet

kapta. A későbbiekben ezeket az elnevezéseket célszerű olyan azonosítóval helyettesíteni, amely utal az objektum szerepére. Így sokkal könnyebben tudjuk az eseménykezelő eljárásokat megírni.

Az üres címkét jelölő téglalap más objektum kiválasztása esetén eltűnik az ablakból. A legegyszerűbben úgy találhatjuk meg, hogy kiválasztjuk az objektumot a *Properties* ablak legördülő listájából.

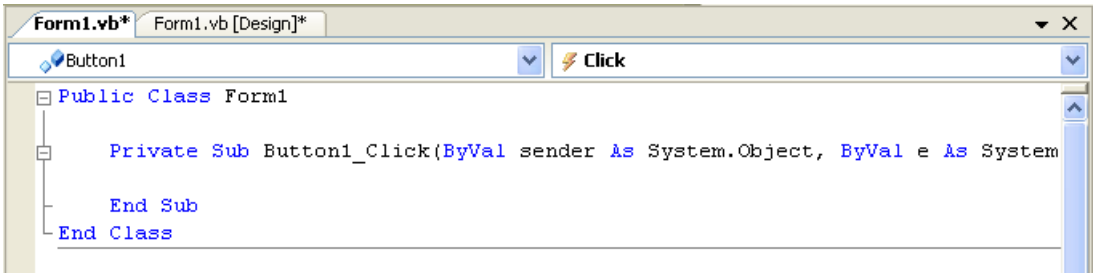


A címkeobjektum kiválasztása a Properties ablak listájában

Eseménykezelés

Az előzőekben a fejlesztői környezet segítségével létrehoztuk az ablakot és elhelyeztük benne a szükséges elemeket. Most az eseménykezelő eljárás megírása következik. Az eseménykezelő eljárások meghatározott objektumok megadott eseményeihez tartoznak. Az eljárások utasításai az események bekövetkezésekor kerülnek végrehajtásra.

Kattintsunk duplán az OK-gombra, melyhez hozzárendeljük az eseménykezelő eljárást. Automatikusan megnyílik a kódszerkesztő ablak. A kódszerkesztő és a tervező ablak között az ablak tetején lévő fülek segítségével válthatunk. A fájlnév mellett lévő csillag azt jelzi, hogy az utolsó mentés óta módosult az ablak tartalma.



A kódszerkesztő ablak a fülekkel és a forráskód sablonjával

A kódszerkesztő a konzolalkalmazásoknál tapasztalt módon most is elkészíti a program keretét. Az eseménykezelő eljárás utasításait a *Sub … End Sub* kulcsszavak közé írjuk. Az eljárás neve célszerűen az objektum (*Button1*) és az esemény (*Click*: kattintás) nevéből áll, amiket aláhúzásjel köt össze. Mint látjuk, a dupla kattintás hatására a kódszerkesztő kiválasztotta az objektumhoz kapcsolódó leggyakoribb, úgynevezett alapértelmezett eseményt (parancsgomb esetén az egérkattintást). Ezt természetesen szükség esetén módosíthatjuk. Az objektumhoz kapcsolódó eseményeket az ablak tetején, a jobb oldalon található legördülő lista tartalmazza.

> Az eseménykezelő eljárás két paraméterrel rendelkezik. Az első paraméter (*sender*) segítségével az eljárást meghívó objektumot, a második paraméterrel (*e*) pedig az esemény tulajdonságait érhetjük el. A paramétereket most nem fogjuk felhasználni. A sor végén álló *Handles* (kezeli) kulcsszó után soroljuk fel azokat az objektumokat és eseményeket, amelyekre az eseménykezelő eljárás vonatkozik. Egy eljárás több objektum több eseményét is kezelheti. A listában az objektum és az esemény közé pontot teszünk. **Az eseménykezelő eljárás keretét a fejlesztői környezet automatikusan elkészíti.**
> A *Sub* előtt álló *Private* kulcsszó azt jelzi, hogy eljárásunk egy másik osztályból/modulból nem hívható meg.

A kódszerkesztő ablakban megfigyelhetjük, hogy az eseménykezelő eljárás a *Form1* osztály egy eljárása. Az osztály tagjai a *Class … End Class* kulcsszavak közé kerülnek. A *Class* után áll az osztály neve. A *Public* kulcsszó arra utal, hogy osztályunkra más modulok is hivatkozhatnak.

Ha egy projekt megnyitásakor nem látjuk a kódszerkesztő ablakot, akkor a megoldástallózóban kattintsunk a jobb egérgombbal az űrlapra (*Form1*) és válasszuk a *View Code* parancsot! Szükség esetén ugyanígy nyithatjuk meg az űrlapot a tervező nézetben (*View Designer*).

Az eseménykezelő eljárás elkészítése

Mint látjuk, a kódszerkesztő automatikusan elkészíti helyettünk programunk keretét. Nekünk csak az eseményt kezelő utasításokat kell begépelnünk. Feladatunk, hogy kiemeljük a *TextBox1* szövegdobozba írt szöveget, majd az üdvözlettel kiegészítve beírjuk a *Label2* címkébe. A szövegdoboz szövegét az objektum *Text* tulajdonsága adja meg. Egy objektum tulajdonságára az objektum és a tulajdonság nevével hivatkozunk, amiket ponttal választunk el egymástól:

```
objektumnév.tulajdonságnév
```

Például:

```
TextBox1.Text
```

A tulajdonságok a program változóihoz hasonlóan kezelhetők. Így a *Label2* objektum *Text* tulajdonságát a következő értékadó utasítással módosítjuk:

```
Label2.Text = TextBox1.Text
```

Ennek hatására a szövegdoboz szövege átkerül a címkeobjektumba.

Egészítsük ki a szöveget az üdvözléssel:

```
Label2.Text = "Szia " & TextBox1.Text & "!"
```

Gépeljük be ezt az utasítást a *Sub … End Sub* utasítások közé. Figyeljük meg, hogy amint beírjuk a *Label2*-t követő pontot, az intelligens súgó megjeleníti az objektum tulajdonságait (és metódusait). A listából a már ismert módon választhatjuk ki a *Text* tulajdonságot. Hasonlóan cselekedhetünk a *TextBox1*-nél is.

Mentsük a projektet, majd futtassuk a programot! Próbáljuk ki a működését. Az üdvözlő szöveg megjelenése után írjunk be egy másik nevet, majd ismét kattintsunk az OK-gombra. Az esetleges hibák kijavítása után készítsük el a lefordított változatot (*Build*/*Build* menüparancs).

A Visual Basic korszerű vizuális fejlesztői környezetének a segítségével gyorsan és egyszerűen tervezhetjük meg az ablakokat és írhatjuk meg eseményvezérelt programjainkat.

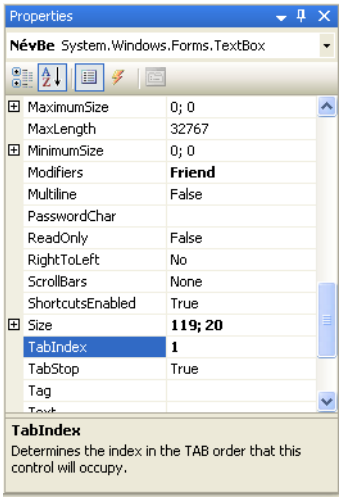
A program futtatása

A fókusz vezérlése és az ablak bezárása

A fókusz vezérlése

Adatokat általában szövegdobozokba írunk be. A beolvasást, feldolgozást parancsgombokkal vezéreljük. Ha egy szövegdobozra kattintunk, akkor egy villogó kurzor jelenik meg benne. Parancsgombok esetén vastagabb keret mutatja a kijelölést. Az elemeknek ezt az állapotát úgy fejezzük ki, hogy megkapták a **fókuszt**.

Az adatbevitelt sokkal kényelmesebbé tehetjük, ha az egér helyett a tabulátorbillentyűvel mozgunk az egyes elemek (szövegdobozok, parancsgombok) között. A sorrendet az objektumok *TabIndex* tulajdonságával adhatjuk meg. A fókusz a tabulátorbillentyű lenyomásakor a következő sorszámú elemre ugrik. A program indításakor a legkisebb *TabIndex*-szel rendelkező elem kapja meg a fókuszt.



A szövegdoboz TabIndex tulajdonsága a Properties munkaablakban

A fókuszt a *Focus* metódus meghívásával adhatjuk át egy elemnek, például:

```
NévBe.Focus()
```

Megkönnyítjük a felhasználó dolgát, ha a *SelectAll* metódussal ki is jelöljük a szövegdoboz tartalmát:

```
NévBe.SelectAll()
```

Ebben az esetben az új név beírásakor automatikusan törlődik a szövegdoboz előző tartalma.

A *Focus* és a *Select* metódus alkalmazását a **Fókusz** projekt mutatja be, amely bekér egy nevet és egy telefonszámot, majd a *Beolvas* gombra kattintáskor egy címkeobjektum segítségével megjeleníti az adatokat.

Ha egy szövegdobozra meghívjuk a *SelectAll* metódust, akkor a fókusz átvételekor kijelölésre kerül a tartalma. Ezért célszerű a telefonszámot fogadó szövegdobozra is meghívni a metódust:

```
TelefonBe.SelectAll()
```

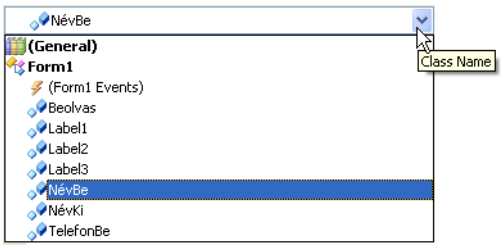
Így, miután átléptünk a telefonszám szövegdobozára, ennek is kijelölésre kerül a tartalma.[7]

[7] A *SelectAll* metódust elegendő lenne egyetlenegyszer, például az ablak megjelenítésekor bekövetkező *Load* esemény eseménykezelő eljárásában meghívni.

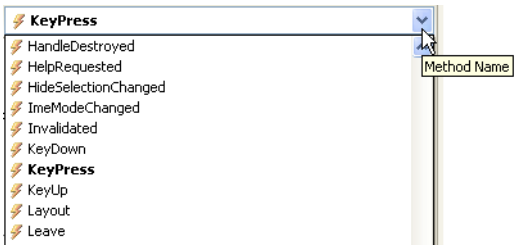
Az Enter billentyű kezelése

Adatbevitel esetén megszoktuk, hogy az Enter billentyű lenyomása is átadja a fókuszt a következő elemnek (vagy végrehajtja a parancsgomb *Click* eseménykezelőjét). Ehhez készítsük el a szövegdobozok *KeyPress* eseménykezelő eljárását. A *KeyPress* esemény akkor következik be, ha lenyomunk egy billentyűt.

A *KeyPress* nem alapértelmezett esemény a szövegdobozoknál, így az eseménykezelő eljárás vázát nem hozhatjuk létre a szövegdobozra való dupla kattintással. Helyette lépjünk át a kódszerkesztő ablakba. A forráskód felett látható bal oldali legördülő listában válasszuk ki a szövegdoboz objektumot, a jobb oldali listában pedig a *KeyPress* eseményt. A forráskódban létrejön az eseménykezelő eljárás váza.



A szövegdoboz objektum kiválasztása



A KeyPress esemény kiválasztása

A *KeyPress* esemény tetszőleges billentyű lenyomásakor bekövetkezik. Meg kell vizsgálnunk, hogy a felhasználó Entert nyomott-e meg. A lenyomott billentyű kódját az eseménykezelő eljárásban használható *e.KeyChar* változó adja meg[8]. Ezt hasonlítjuk össze az Enternek megfelelő *Keys.Return* kódjával, amit a *ChrW* függvénnyel határozunk meg[9]. Ha megegyeznek, akkor átadjuk a fókuszt a telefonszámot beolvasó szövegdoboznak:

```
If e.KeyChar = ChrW(Keys.Return) Then
  TelefonBe.Focus()
End If
```

A telefonszámot beolvasó szövegdoboz *KeyPress* eseménykezelőjében pedig meghívjuk a parancsgomb *Click* eseménykezelőjét, hiszen ugyanazt a műveletet kell végrehajtani. A parancsgomb eseménykezelőjének egyszerűen átadjuk a szövegdoboz eljárásának paramétereit:

```
If e.KeyChar = ChrW(Keys.Return) Then
  Beolvas_Click(sender, e)
End If
```

A kész programot az **Enter** projekt mutatja be.

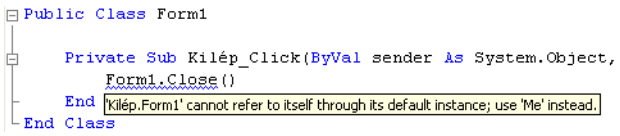
Az ablak bezárása

A programablakot eddig a címsorban lévő *Bezárás* gombbal zártuk be. Ez a módszer nem teszi lehetővé, hogy a futás befejezésekor szükséges tennivalókat, például az adatok mentését elvégezhessük. A felhasználó akár a program utasításainak végrehajtása közben is bezárhatja az ablakot.

A szabályos kilépéshez helyezzünk el egy parancsgombot az ablakba, melynek *Click* eseménykezelőjében hívjuk meg az ablak (*Form*) objektum *Close* metódusát:

```
Form1.Close()
```

[8] Az *e* eseményobjektum *KeyChar* tulajdonságára hivatkozunk.
[9] A *ChrW* a két bájtos Unicode-ot adja meg.



Hibaüzenet a Form1 hivatkozás alkalmazásánál

Amint beírjuk az utasítást, hibaüzenet jelzi, hogy az ablak (űrlap) forráskódján belül nem használhatjuk a *Form1* nevet. Helyette a *Me* azonosítóval kell hivatkoznunk az aktuális ablakra:

```
Me.Close()
```

A *Close* metódus használatát a **Kilép** projekt mutatja be. Az ablak bezárása előtt üzenettel figyelmeztetjük a felhasználót.[10]

[10] A *Me.Close*() helyett használhatjuk az *Application.Exit*() metódushívást is.