Koncz Balázs

Hogyan válj jól fizetett C# programozóvá?

BBS-INFO Kiadó - 2019.

A könyv nagyobb mennyiségben megrendelhető a kiadónál:
BBS-INFO Kiadó, Tel.: 407-17-07 info@bbs.hu

Papírkönyv: ISBN 978-615-5477-69-0
E-book: ISBN 978-615-5477-70-6

Kiadja a BBS-INFO Kft.
1630 Budapest, Pf. 21.
Felelős kiadó: a BBS-INFO Kft. ügyvezetője
Nyomdai munkák: Biró Family Nyomda
Felelős vezető: Biró Krisztián

Tartalomjegyzék

1. Bevezetés

1.1. Mibe kerül megtanulni programozni?

A tanulásra legjobb, ha befektetésként tekintünk. Ez a hasonlat többé-kevésbé meg is állja a helyét, ugyanis két tényezőt mindenképpen figyelembe kell venni: a befektetéshez szükséges pénz mennyiségét, valamint a gyarapodási idő hosszát (azt az időt, amíg a befektetett értékünk csendben növekszik).

1.1.1. Pénzügyi befektetés vagy tanulmányi befektetés?

Amikor elmész a bankba és lekötöd a pénzed, vagy valamilyen részvényt vásárolsz; akkor érsz el nagyobb hasznot (adott idő alatt), ha nagyobb tőkével indulsz. Pl.: Jelenleg egy banki lekötött betét nagyságrendileg 1%-ot kamatozik. Tehát, ha van 100.000 forintod januárban, akkor a futamidő végére 101.000 forintod lesz. A nyereség 1000 Ft. Azonban, ha 1 millió forintot kötsz le, a haszon 10.000 Ft (ami igen szerény).

Jól látható: ahhoz, hogy jelentős nyereséget könyvelhess el, legalább egy nagyságrenddel növelned kell a befektetett pénz mértékét.

Ezzel szemben, ha tanulmányaidba fektetsz, a kritikus pont a tanulással eltöltött idő lesz. Ez pedig az a bruttó idő, amit tanulással töltesz. Onnantól kezdve, hogy elindul a géped, és megírod az első programod, odáig, hogy elhelyezkedsz a munkaerőpiacon.

A következő dolgokat kell számításba venned:

- **Anyagi befektetés:** Mindenképpen szükséged lesz egy számítógépre. Egy megfelelő laptopot már 130.000–150.000 forintért is kaphatsz. Akár előre telepített operációs rendszerrel is. (A könyv használatához Windows 10-et ajánlok). Ha nagyon profi akarsz lenni, szerezhetsz mellé plusz 1 monitort, egeret és

billentyűzetet is. Ez egy kicsit megdobhatja a költségeket, de sokkal kényelmesebb így dolgozni.

- **Tanulmányi idő:** Ez változhat attól függően, hogy mennyi időt tudsz arra szánni, hogy fejlődj. (Fontos, hogy rendszeresen ülj le tanulni, mert különben el fogod felejteni a korábbi anyagot.) Amennyiben akár napi 8 órát is tudsz foglalkozni a programozással, akkor nagyon szerencsés vagy (és valószínűleg eltökélt is). Így akár 3–6 hónap alatt is megtalálhatod a helyedet a munkaerőpiacon.

Ha eleinte lassan haladsz, ne add fel! Az a képesség, hogy megtanulj dolgokat, folyamatosan fejlődik. De csak akkor, ha használod. Ezért fontos, hogy legyél kitartó, és áldozz időt a céljaid elérésére.

Lehetséges, hogy karriert készülsz váltani és napközben sokat dolgozol, vagy utazol. Ebben az esetben nagyon figyelmesnek kell lenned az órarendeddel. Részmunkaidős állás mellett napi 4, teljes munkaidő esetén napi maximum 2 óra áll rendelkezésünkre, valamint a hétvégék.

Ezzel a módszerrel 1-2 éven belül juthatsz el egy munkaképes programozó szintjére. Ne félj, az erőfeszítéseid meg fognak térülni.

Most valószínűleg a legtöbben azt gondolják: minden hétköznapra bezsúfolnak 2 óra tanulást, és a hétvégéket is ezzel töltik. Így majd pikk-pakk profik lesznek, és jöhet is az álommeló. A rossz hír az, hogy a dolgok nem működnek ilyen könnyen. Legyél őszinte magaddal, és próbálj előrelátó lenni. A munkán kívül sok minden más is fontos az életben, például a család, a párkapcsolat, a barátok, vagy a kutyus, aki alig várja, hogy elvidd a parkba sétálni. Mindemellett, ha egész nap terhelő szellemi munkát végeztél, nem fogsz tudni minden nap plusz 2 órában eredményesen tanulni.

Tanulság: Alakíts ki egy követhető órarendet magadnak, amivel rendesen tudsz fejlődni, és megengedő a magánéleteddel kapcsolatban.

Alternatív megoldás lehet a részmunkaidős állás. Napi 4 óra munka után kell, hogy maradjon idő és energia a tanulásra is. Azonban ne feledkezzünk meg a kieső jövedelemről sem. (Mivel kevesebbet fogunk dolgozni, a fizetésünk is kevesebb lesz.)

1.1.2. Mire számíthatok a munkaerőpiacon?

A befektetett időd hamar meg fog térülni, amennyiben sikerül elhelyezkedned valamelyik cégnél. A laptopba és egyebekbe fektetett pénzt megnövekedett jövedelmünk 1-2 hónap alatt visszahozza. Sőt, lakhelyünktől függően bőven az átlag felletti keresethez is juthatunk.

Ezenkívül számos más előnye is van annak, ha elsajátítasz egy-egy programnyelvet:

- Megtanulsz racionálisan gondolkodni és problémákat megoldani.
- Sokkal könnyebb lesz más programozási nyelveket megtanulni.
- Kialakítja benned a „fejlődnőm kell" hozzáállást.

1.2. Hogyan gondolkodjak úgy, mint egy programozó?

Mielőtt belevágnék a dolgokba, le kell szögeznem, hogy a programozás A-tól Z-ig problémamegoldásból áll. Nagyon távolról nézve ez azt jelenti, hogy van valamilyen feladat, esetleg üzleti lehetőség, aminek a megoldását egy bizonyos szoftvertermék jelenti. Pontosabban, a te szoftvered, amit meg fogsz írni. Dióhéjban a következő fog történni: végiggondolod a problémát, megkeresed a megoldást, majd lépésről lepésre leírod a hozzá vezető utat. Az ilyen módszereket nevezzük 'mechanikus módszernek'. Egyszerű, nemde?

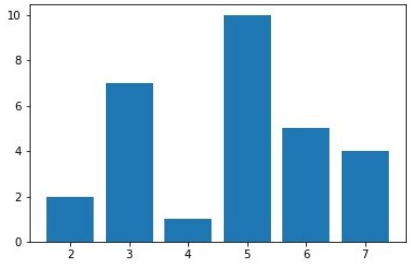
A problémát a programozás megtanulása közben az szokta jelenteni, hogy az emberi gondolkodás és a gépi működés alapvetően eltérő. Mindkettőnek vannak előnyei és hátrányai is. Pl.: A legtöbb embernek nem jelent problémát egy kosárlabda megkülönböztetése a tejszállító autótól, viszont egy nehezebb egyenlet megoldása már igen. A számítógépekkel is ugyanez a helyzet, csak fordítva.[1]

A te dolgod ezért az lesz, hogy elsajátíts egy olyan gondolkodási módszert, amit a számítógépek használnak. Ezt hívják algoritmikus gondolkodásnak. Tudnod kell, hogy ez egy szigorú világ kemény szabályokkal, amiket be kell tartani ahhoz, hogy sikeres legyél.

[1] Léteznek számítógépes eljárások, amik az emberi információfeldolgozást utánozzák. Ezekről ebben a könyben nem esik szó.

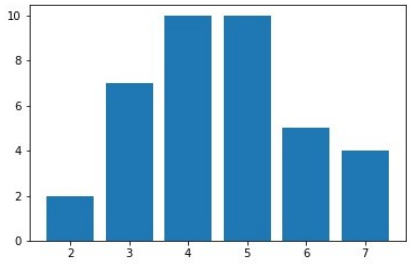
Szerencsére van egy jó hír is: ha megbarátkozol ezekkel a szabályokkal, a programozás jó szövetségesed lesz, bármibe is fogsz később.

Nézzünk egy példát! Az alábbi ábrán hogyan választanánk ki a legmagasabb oszlopot?



1. ábra

A feladat elég egyszerű: rámutatsz a 4. oszlopra középen, mivel kimagasodik a többi közül. De mi a helyzet ezzel?



2. ábra

A 3. oszlop ugyanolyan nagy, mint a 4. oszlop. Hogyan döntesz, ha csak az egyiket választhatod? Valószínűleg az olvasók közel fele a 3.-ra, a másik fele pedig a 4.-re voksolna.

Egy számítógépes algoritmussal azonban más a helyzet. Ugyanazon bemenő adatokra minden esetben ugyanazt a választ kell adnia! Összehasonlításképpen nézzük meg a probléma megoldását mechanikus módszerrel!

1. Lépés: Nevezd ki az 1. oszlopot a legnagyobbnak és az aktuális oszlopnak is.
2. Lépés: **Ha** az aktuálistól jobbra van egy oszlop, akkor hasonlítsd össze a kettőt:
   Ha a jobbra lévő oszlop nagyobb, mint az aktuális oszlop,

akkor: a jobbra lévő oszlopot nevezd ki a legnagyobb oszlopnak.
Különben: (ha jobbra már nincsen 1 oszlop sem) eredményként meg kell mutatni a legnagyobb oszlop sorszámát. **Program vége**

3. Lépés: A jobbra lévő oszlopot kinevezed az aktuális oszlopnak.
4. Lépés: Ugorj vissza a 2. Lépésre.

Gondolatban menjünk végig a lépéseken, és vegyük át, mi fog történni, ha alkalmazzuk az 1. ábrára!

1. [1. Lépés]
   Aktuális oszlop: 1.
   Legnagyobb oszlop: 1.
2. [2. Lépés]
   Jobbra van egy oszlop, és az nagyobb, mint az aktuális.
3. [3. Lépés]
   Aktuális oszlop: 2.
   Legnagyobb oszlop: 2.
4. [2. Lépés]
   Jobbra van egy oszlop, és nagyobb, mint az eddig ismert legnagyobb.
   Legnagyobb oszlop: 3.
5. [3. Lépés]
   Aktuális oszlop: 3.
   Legnagyobb oszlop: 3.
6. [2. Lépés]
   Jobbra van egy oszlop, de az nem nagyobb, mint az eddig ismert legnagyobb (pont egyformák).
7. [3. Lépés]
   Aktuális oszlop: 4.
   Legnagyobb oszlop: 3
8. [2. Lépés]
   Jobbra van egy oszlop, de az kisebb, mint az eddig ismert legnagyobb oszlop.
9. [3. Lépés]
   Aktuális oszlop: 5.
   Legnagyobb oszlop: 3.

10. [2. Lépés]
    Jobbra van egy oszlop, de az kisebb, mint az eddig ismert legnagyobb.
11. [3. Lépés]
    Aktuális oszlop: 6.
    Legnagyobb oszlop: 3.
12. [2. Lépés]
    Jobbra már nincsen oszlop, ezért eredményként megmutatjuk a legnagyobb oszlop sorszámát: 3.

Végigolvasva a 12 lépéses felsorolás kissé sportosnak tűnhet az algoritmusunk 4 pontjához képest. De gondolj arra, hogy csak a 4 elvi lépés parancsait kell megírnod, a munka többi részét pedig a számítógép fogja végezni.

Ha alaposan végigolvasod mind a 12-t, és úgy érzed, érted a lényeget, akkor figyeld meg, milyen minták ismétlődnek benne!

- **Értékadás:**
  „Aktuális oszlop: #”
  „Legnagyobb oszlop: #”
  Az aktuális és a legnagyobb oszlop mindketten egy-egy *változó*k. A változók adatok tárolására használhatók a RAM-ban.
- **Elágazás:**
  „Ha... akkor…”
  Ezt a szerkezetet feltételes utasításnak vagy elágazásnak hívják. Ez azt jelenti, hogy a „ha” után megfogalmazott feltétel kiértékelésétől függően fog folytatódni a program futása. A kiértékelés eredménye csakis logikai IGAZ vagy HAMIS lehet. Amennyiben a feltétel igaz, a hozzá kötött utasítások lefutnak. Amennyiben hamis, a program tovább fut a főágon vagy lefutnak a „Különben” ágban megfogalmazott utasítások.
- **Vezérlésátadás:**
  „Ugorj a # lépésre”
  Minden program fentről lefelé hajtja végre az utasításokat, ha csak valami meg nem akadályozza ebben. Mondjuk egy hiba vagy egy vezérlésátadó utasítás. Ha ilyen parancsot adunk, akkor a következő végrehajtandó sor nem az eggyel alatta lévő lesz, hanem a paraméterben megfogalmazott sorszámú.

- **Ciklus:**
  „2. Lépés -> 3. Lépés -> 2. Lépés -> 3. Lépés -> 2. Lépés -> 3. Lépés -> …”
  A szemfülesek kiszúrhatták, hogy bizonyos lépések ismétlődnek egymás után. Ezt hívják ciklusnak. Minden ciklus rendelkezik ciklusmaggal (az utasítások, amik ciklikusan ismétlődnek), valamint egy futási feltétellel, amely azt fogalmazza meg, hogy meddig kell ismételni az adott utasításokat.
  A mi példánkban a ciklusmag a 2. és a 3. Lépés. A futási feltétel pedig a „ha jobbra van oszlop” elágazási utasítás.

1.3. Hogyan tanuljak meg programozni?

Elképzelhető, hogy az előző fejezet nagyon sok új információt tartalmazott számodra. Lehet, hogy többször is át kell nézned, mire úgy érzed, hogy megérted az ott leírtakat. Az is lehet, hogy úgy érzed, nehéz lesz később visszaemlékezni erre mind. Ezért fontos, hogy újra közöljem: igenis meg tudsz tanulni programozni! Ez olyan, mint a biciklizés vagy a görkorcsolya: ha megszokod, és belerázódsz, már magától fog menni, és nem kell gondolkozni a részletkérdéseken.

Így ezt a részt szeretném a tanulási folyamatod mikéntjének szánni. Mutatni fogok néhány tippet és trükköt, amivel sokkal könnyebben és hatékonyabban fogsz haladni. (Plusz néhol egy-két motiváló dolgot is.) Ha viszont az előző rész már a kisujjadban van, akkor nyugodtan ugord át ezt a fejezetet, vagy tekintsd egyfajta üdítőnek a munka előtt.

A programozáshoz nem tehetség kell, hanem kitartás.

Ha filozofálni akarnék (sokat nem fogok), azt is mondhatnám, hogy a klasszikus értelemben vett tehetség nem létezik. Illetve, nem úgy létezik, ahogyan azt sokan gondolják. A sikert, amit elérsz – az esetek túlnyomó többségében – nem a veled született képességek hozzák, hanem az, amit kihozol magadból. Mozart sem azért volt csodagyerek, mert kottákkal a fejében született. Ilyen nincsen, ettől a gondolattól jobb, ha megválsz. Azért tudott kiemelkedő zeneszerzővé válni, mert sok időt szentelt a darabjainak, és közben arra törekedett, hogy minél jobbak legyenek a darabjai. Ennyi az egész. Nincs varázslat. Ha követed ezt a mintát és időt szentelsz a tanulásnak, közben pedig erősen törekszel, hogy minél jobb legyél, meglepően gyorsan fogsz fejlődni!

Hogyan tanuljak?

A programozás elsősorban szakértelem. Amire szükséged van: az elmélet megértése és gyakorlás. Sok gyakorlás. Minél tapasztaltabb leszel, annál jobb szakember fog válni belőled. Ehhez az is szükséges, hogy merj új dolgokat kipróbálni. Ezért én a következőket javaslom a könyv használatához:

- A gyakorlati részekben lépésről lépésre kövesd az utasításokat, és próbáld megérteni, az adott feladat miért szükséges ahhoz, hogy később nagyszerű szoftvereket tervezz.
- Gyakorolj! A fejezetek végén oldd meg a gyakorlófeladatokat olyan módszerrel, ahogy a mintafeladatokat mutattam.
- Próbáld meg ugyanazt a feladatot különböző módszerekkel megoldani. Keress rövidebb és hatékonyabb megoldásokat.

Amennyiben az elméletet megérted, megcsinálod az összes gyakorlófeladatot, és igyekszel minél jobb kódot írni, valószínűleg hamarosan nagyon jó programozó fog válni belőled!

1.4. Milyen lehetőségeim lesznek C# programozóként?

Az egyszerű válasz: elég sok. A bonyolultabb: elég sok, de .... Jól fontold meg, mivel szeretnél dolgozni. Azt a területet válaszd, amelyik a legjobban érdekel, mert:

- Később a váltás nehéz lesz.
- Csak így lesz a munkád a lehető legjobb minőségű. Ez pedig a fizetéseden is meg fog látszani.

Ennek a könyvnek az a célja, hogy adjon egy erős alaptudást és megmutassa, hogyan helyezkedj el a munkaerőpiacon. Lássuk mit nyerhetsz:

- Elsajátíthatod a programozók gondolkodásmódját.
- Megtanulhatod, hogyan kell összetett szoftvertermékeket tervezni.
- Sokkal jobb esélyeid lesznek egy állásinterjún.[2]
- Jelentkezhetsz Junior C# programozó állásokra.

[2] A könyv elolvasása nem garancia arra, hogy az olvasó lesz a legjobb jelölt bármelyik interjún.

2. Alapok dióhéjban

2.1. Számítógép-lélektan

A közhiedelemmel ellentétben a számítógép egy nagyon egyszerű dolog. Mindent úgy csinál, ahogy megtervezték, és nincsenek gonosz kis titkai. Ha eleget tanulunk, minden részét megérthetjük, és kényelmesen fogunk tudni vele dolgozni.

A legtöbb mai számítógép a Neumann-elvű számítási modellre épül. Ez azt jelenti, hogy:

- Adatokon hajt végre műveleteket.
- Az adatokat változók képviselik.
- A változók értékei akárhányszor változtathatók.
- Az adatok és az utasítások azonos memóriában helyezkednek el.
- Egy feladatot egymás után végrehajtandó utasításokkal old meg.

Ezek egy kevés magyarázatra szorulnak: Ha visszalapozol egy fejezettel, a maximum kiválasztás problémáján láthatod is a példákat.

Az értékadó műveleteknél az *aktuális oszlopot* és az eddig ismert *legnagyobb oszlopot* el kellett tárolni. Ezek a program szempontjából változók, és a gép műveleteket hajt végre rajtuk, amikor az aktuális oszlop száma eggyel növekszik, vagy új értéket írsz a legnagyobb oszlop változóba. Ezt pedig akárhányszor megteheted. Az adatok és az utasítások egyformán az operatív tárba (RAM) töltődnek be a program futása során. Ha elég kicsi lennél, és végigsétálnál egy memória sínrendszerén, mint egy városi utcán, akkor minden egyes házba becsöngetve csak azt tapasztalnád, hogy mindenütt egyforma hosszú egyesek és nullák sorozata lakik, és csak nagyon keveset tudnak arról, hogy miért is léteznek, és miért pont ott, ahol. Amikor egy ház (adott memóriacím) lakója értesítést kap arról, hogy keresik, akkor kiballag az „utcára”, és megkezdi (igen gyors) útját a processzor felé, ahol majd a megírt program függvényében eldől, hogy mi lesz a további sorsa.

Az, hogy egy feladatot egymás után végrehajtódó utasításokkal oldunk meg, bizonyos körökben magától értetődő, de nem mindenhol. Amikor az első számítógépeket tervezték, azok elsősorban nagyon hatékony számológépek voltak – és igen nagyok is. A korabeli tudósok még a legvadabb álmaikban sem remélték, hogy sok évvel később már akárkinek a zsebébe is költözhet egy ilyen gép, vagy hogy alkalmas lesz cicás videók megosztására. Ha még emlékszünk a matematika órákra, az egyenletek megoldása is egymás után következő lépésekből áll – amiknek többé-kevésbé szigorú a sorrendje. Pl.: közös nevezőre hozás, nullára rendezés, kiemelés stb. Ha valami hétköznapibbal szeretnéd összehasonlítani, keress egy szimpatikus szakácskönyvet. Minden recept elején fel vannak sorolva a hozzávalók (változók), majd az egymás utáni lépések, ahogyan el kell készíteni az ebédre valót.

Azonban jaj annak, aki nem követi pontosan az ott leírtakat. Elég, ha 10 perccel tovább sütöd a húst, és a lélekmelengető fogás helyett vajas kenyér lesz csak az ebéd. A programozásnál is így van. Akár egy rossz helyen meghívott függvény is jelentheti az elkövetkezendő órák (vagy napok) tennivalóját is. Szerencsére azonban, a sült hússal ellentétben, emiatt nem fog ugrani az ebéd. Maximum önkéntes alapon. Bárkivel előfordulhat, hogy miközben mazsolázgat egy kódot, hirtelen elmegy a nap.

2.2. Hogyan illeszkedik a képbe a C#?

A C# egy magas szintű programozási nyelv, tehát közelebb áll az emberi gondolkodásmódhoz, mint a hardveres logikához. (Bár nagyon közel sajnos azért így sincs hozzá.) Ahhoz, hogy megértsd a C# helyét a programnyelvek között, meg kell még értened egy-két dolgot a számítógépekkel kapcsolatban:

A hardveres logika azt a logikát jelenti, ahogyan a processzor áramkörei csatlakoznak, illetve ahogy ide csatlakozik az operatív tár, a háttértár és más hardverek. Minden processzortípusnak saját utasításkészlete van. Ezt a programozási nyelvet Assemblynek hívják. Tudnod kell róla, hogy a gép az operatív táron kívül lényegében a regisztereket és ezt-azt látja, valamint csak az alapműveleteket ismeri. Mivel eléggé limitált ez a történet, azt szokták mondani, hogy Assembly szinten nincs különbség a normális és a kártékony szoftverek között, miután a processzor annyit lát a dolgokból, hogy 1-2 byte ide, 1 jelzőbit

oda és minden rendben. Ez ihlette egyrészt a szakembereket arra, hogy magasabb rendű programozási nyelveket hozzanak létre, másrészt a hackereket arra, hogy Assemblyt tanuljanak, és jót mulassanak közepesen törvénytelen dolgokon.

A magas szintű programnyelvek egyrészt az említett biztonsági rés, valamint a produktivitás miatt jelentek meg. A problémát az jelenti, hogy processzorközeli nyelven megoldani egy egyszerűbb feladatot, akár ami mondjuk kedvenc versünkből kikeresi az összes betűt és írásjelet, valamint megszámolja, hogy melyikből mennyi van – igen nehéz. Egy gyakorlottabb szakembernek ezt megírni Assemblyben lehet, több mint 1 óráig tartana, C#-ban azonban könnyed ujjgyakorlat az egész.

A másik előnye: nem hagy olyan dolgot tenni, amivel potenciálisan romba dönthetjük a számítógépet vagy egy éppen futó másik programot. Assemblyben senki sem akadályoz meg minket abban, hogy szabadon beleírhassunk bármelyik memóriacímbe, még akkor sem, ha azt egy másik program éppen használja. Már óvodában is tanítják, hogy mások játékait elvenni nem szép dolog. Ezért jó, ha tudod: ugyanez a helyzet más programok által használt memóriacímekkel is!

Mivel más embereket megóvni attól, hogy butaságot csináljanak, meglehetősen nagy falat (bólogató baleseti sebészek és pszichológusok tömegét látom magam előtt), ezért a Microsoft megalkotta az úgynevezett .NET keretrendszert, aminek több feladata is van. A legfontosabbak, amiket ismerned kell:

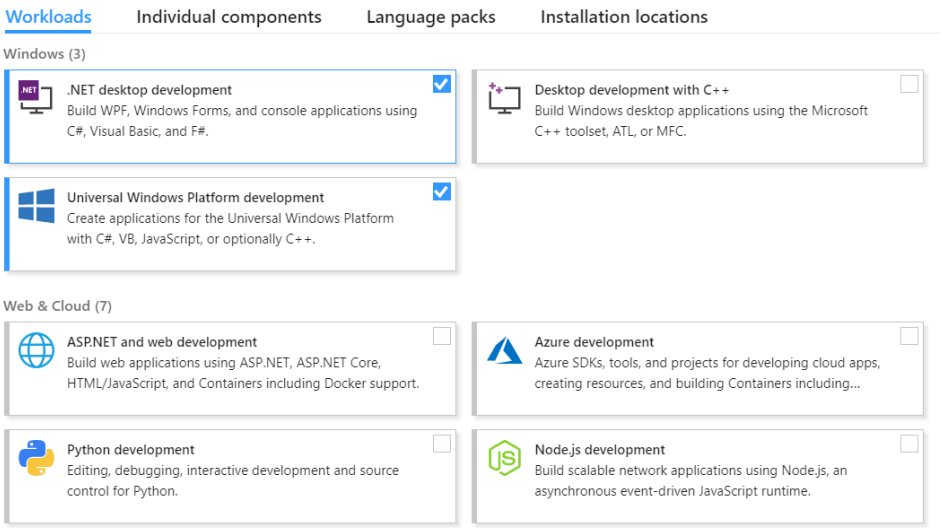
- Biztonságosabbá teszi a programozást, és nem hagy olyan memóriacímekhez nyúlni, amikkel nincs dolgod. Ezzel a programozó romba döntési képességeit jelentősen korlátozza.
- A C# nyelven megírt kódok menedzselt kódok. Ez azt jelenti, hogy fordítás után egy köztes nyelvre fordulnak le, majd a fordító ezt a köztes kódot fordítja le a processzor által értelmezhető utasításokra.
- A menedzselő környezet futás közben képes leállítani egy folyamatot, ha az gyanúsan viselkedik, valamint optimalizálhatja a kódot.
- Mivel ez jelentős teljesítményveszteséget jelent, a fordítónak egy trükkre van szüksége: A fordítás nem a futás előtt, hanem közben történik.

3. Egy fejezet azoknak, akik még nem programoztak

3.1. A fejlesztőeszköz telepítése

A C# fejlesztéshez több IDE (Integrated Development Environment) közül is választhatsz. Ott van például a Code::Blocks, MonoDevelop vagy a Visual Studio Code. Ezeket nyugodtan próbáld ki, de én minden esetben a Visual Studio legfrissebb verziójának a Community kiadását ajánlom. Ez egy otthoni projektekre ingyenes fejlesztőeszköz, rengeteg hasznos funkcióval és bővítménnyel. Használhatod asztali és mobilalkalmazások fejlesztésére ugyanúgy, mint háromdimenziós játékokhoz. Egyszóval a kategóriában messze a legjobb.

A letöltő linket a visualstudio.com vagy a visualstudio.microsoft.com oldalon keresztül érheted el.



3. ábra

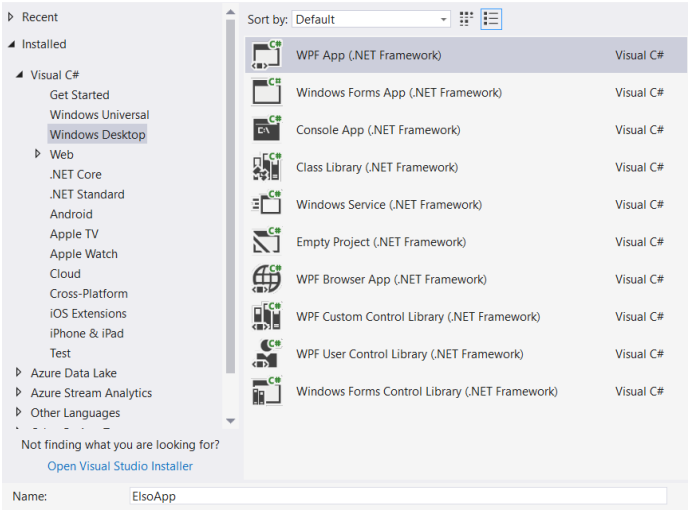
A telepítő indítása után ki kell jelölnöd a telepítendő csomagokat. Ezek határozzák meg, hogy milyen típusú projekteket tudsz majd létrehozni a jövőben. Mindenképpen telepítsd a .NET desktop development, Universal Windows Platform development, Mobile development with .NET nevűeket.

Szükség lesz jó néhány GB szabad helyre, tehát, ha tele van mindenfélével a géped, jó, ha végzel egy kis lomtalanítást. Ha megvagy, kattints az Install gombra, és várd meg, amíg a Visual Studio feltelepül.

3.2. A Visual Studio kezelőfelülete

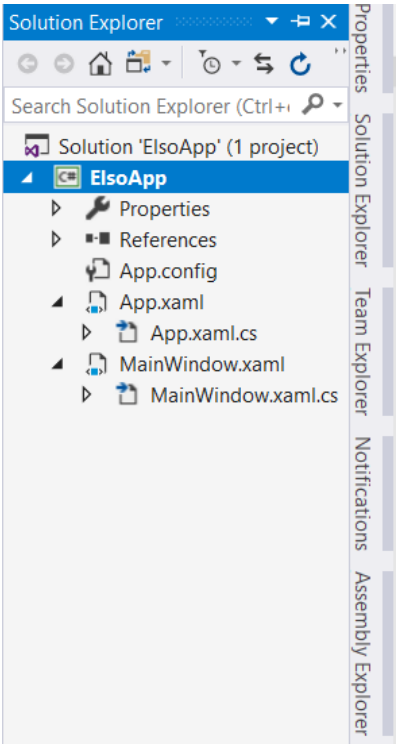
Mielőtt belecsapnánk a lecsóba, megmutatok egy-két dolgot a Visual Studioval kapcsolatban. Mivel várhatóan sokáig fogtok együtt élni, ezért jó, ha kiismered, mi hol található rajta. Ne félj, nem fogok belemenni ilyen hasznavehetetlen kínos részletekbe, hogy itt a File menü, itt tudsz menteni, stb.

Helyette inkább folytassuk idegenvezető üzemmódban. Tehát mindenki beszállás, indítsunk egy Visual Studiot egy új WPF projekttel! Ezt úgy teheted meg, hogy elnavigálsz a File menüben a New, azon belül pedig a Project … fülre. Itt pedig válaszd a WPF App (.NET Framework) lehetőséget, ahogy a 4. ábra mutatja. Bármilyen nevet adhatsz neki, most meg fog felelni, de válassz egy olyan elérési útvonalat, ahol meg fogod találni később is.



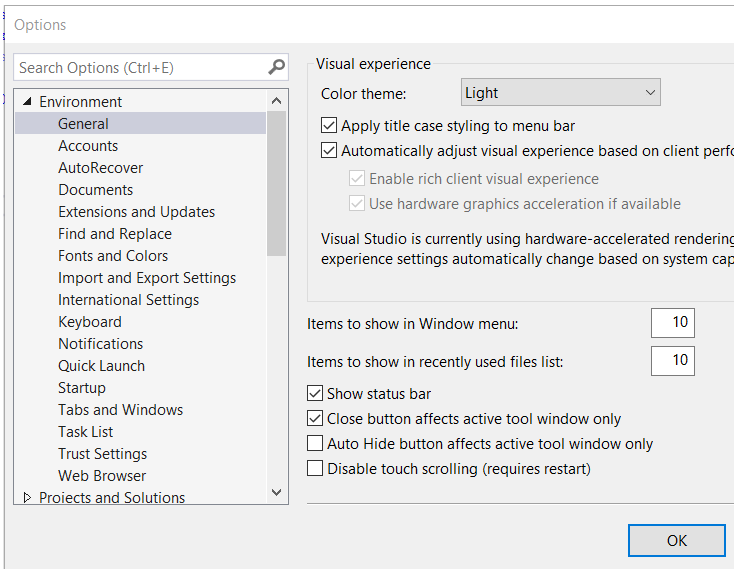
4. ábra

Ha létrejött a sablon, szemlélődj egy kicsit. A File menüvel egy sorban vannak dolgok, amiket használni fogsz, de most egyesével mindet nem magyarázom el. A könyv további részében mindent meg fogsz tudni, hogyan működik, pont akkor, amikor erre szükség lesz. Ami viszont nem fog szembejönni és szükséges, arra most vesztegessünk egy kis időt. Gyalogolj el a Tools/Options menüponton az Environment/General szekcióra (5. ábra). Itt a Color theme melletti legördülő menüben megváltoztathatod az IDE színének témáját. Próbáld ki a lehetőségeket, és válaszd azt, amelyik a legjobban tetszik!



5. ábra

Jobbra segítő ablakokkal találkozhatsz. Ezek közül a legfontosabb a Solution Explorer és a Team Explorer. Ha bármelyik hiányozna a File mellett kettővel lévő View menüben előhívhatod őket.



6. ábra

Tehát a Solution Explorerben az adott projekted fájlrendszerét láthatod. Referenciákkal, mappákkal, fájlokkal, szó szerint mindennel. A Team Explorer majd akkor lesz érdekes, ha egy verziókezelt projekten dolgozol. Ezzel majd egy későbbi részben foglalkozunk (egészen pontosan az „Egy projekt, amivel felturbózhatod az önéletrajzodat" címűben.)

Most nézd meg a MainWindow.xaml fájlt! Amit itt látsz, az egy ablakos alkalmazás üres sablonja. Igazán kár lenne, ha nem volna benne semmi, ezért adj hozzá egy feliratot valamilyen szöveggel. Ez látható az 1. kódon.

```
<Window x:Class="ElsoApp.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:local="clr-namespace:ElsoApp"
      mc:Ignorable="d"
      Title="MainWindow" Height="80" Width="80">
   <Grid>
      <TextBlock Text="42" />
   </Grid>
</Window>
```

1. kód

Ha szeretnél egy saját ikont beállítani a Solution Explorerben, kattints a jobb egérgombbal a projektedre (nem a solutionre!) és keresd meg a properties lehetőséget. Itt az Icon: szekciónál kattints a Browse... gombra, és keresd meg kedvenc ikonod. Bármilyen .ico kiterjesztésű fájl megteszi.

Ahhoz pedig, hogy elindítsd a programot, nyomj F5-öt, vagy bökj a Start gombra felül.

A .exe fájlt (ami a kódból készül) a következőképpen tudod megszerezni: kattints szintén a projektre jobb klikkel, ahogyan az előbb, de most válaszd az Open Folder in File Explorer lehetőséget. Itt navigálj a bin, majd pedig a Debug vagy Release mappába, ahol megtalálhatod a [projekted neve].exe állományt. Még az ikonját is megismerheted!

3.3. Programozási alapok

3.3.1. Változók és konstans mezők

Ahogyan azt már többször is emlegettem, minden programnyelvben változókat kell használnod az információ tárolására. Ahhoz, hogy egy változót *deklarálhass*, szükség van a nevére és a típusára. C#-ban a leggyakrabban használt beépített típusok a következők:

Típus kulcs szava	Tárolható érték	Érték tartomány
bool	Logikai igaz/hamis	n/a
byte	8 bites előjel nélküli egész szám	0 és 255 között
int	32 bites, előjeles egész szám	-2.147.483.648 és 2.147.483.647 között
char	egyetlen karakter	16 bites Unicode karakterkészlet.
string	szöveg	n/a
float	32 bites, egyszerű tizedestört	-3,4 * 10^38 és 3,4 * 10^38 között
double	64 bites duplapontos tizedestört	+/- 5 * 10^-324 és +/-1,7 * 10^308 között
long	64 bites előjeles egész szám	9,223,372,036,854,775,808 és 9,223,372,036,854,775,807 között

1. táblázat

Szerencsére ezt a táblázatot nem kell fejből megtanulni, viszont jó, ha nagyjából tudod, hogy hol milyen típust kell használni. Pl.: egy vonat utasait számlálva fölösleges a long vagy a double értéket használni.

Tegyük fel, hogy egy változóban szeretném tárolni egy gomb nevét, egy másikban pedig azt, hogy hányszor nyomtuk meg. Ez C#-ban így fog kinézni:

```
string buttonName = "Ne nyomd meg";
int numberOfPress = 200;
```

Amit itt látsz, az egy típussal és kezdőértékkel megadott változó-deklarálás. Ilyen esetben meg kell adni a változó nevét, típusát, az egyenlőségjel után pedig az értékét. Megteheted azt is, hogy kihagyod az értékadást. Ilyenkor a változó értéke az úgynevezett „default value”,

vagyis típusa szerinti alapértelmezett értéket fogja megkapni. Ezeket az értékeket az alábbi táblázat tartalmazza.

bool	byte	int	char	float	double	long
false	0	0	'\0'	0.0f	0.0d	0.0l

Ezekre a változókra érvényes az a Neumann-elv, miszerint akárhányszor változtathatják az értéküket. Néha viszont célszerű megszegni ezt a szabályt. De mikor és miért? Tömören: vannak esetek, amikor nem célszerű bántanunk bizonyos változókat. Sőt, biztos, ami biztos, valahogyan meg kell tiltanod, hogy matatni lehessen velük. (Például, ha egy adott helyen a gravitációs gyorsulással vagy a fénysebességgel kell számolni.)

Erre valók a konstans mezők, ilyeneket a const kulcsszóval lehet deklarálni. Így:

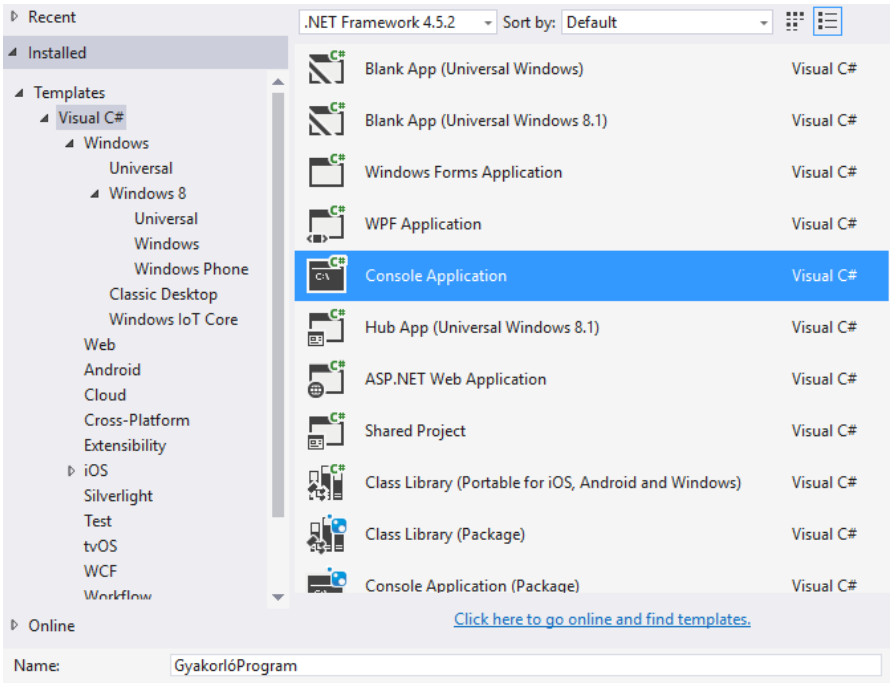
```
const int lightspeedInmps = 299792458;
```

Így az univerzum és a fizika törvényei biztonságban lesznek, mert a fény sebessége konstans értékként van tárolva.

3.3.2. Egyszerű műveletek

Eljött az ideje, hogy a tettek mezejére lépj, és fejest ugorj a programozásba! Nyisd meg a gépeden kedvenc Visual Studio verziódat és hozzunk létre egy új projektet. Mondjuk egy konzolos alkalmazást! Ehhez kattints a fájlmenüben a New, majd a Project fülre. A felugró ablakban pedig válaszd ki a konzolos alkalmazást, és adj neki egy meggyőző nevet, ahogyan a 7. ábra is mutatja.

Az újonnan létrejött kódsablon a konzolos alkalmazás elindításához szükséges elemeket tartalmazza. A namespace kulcsszóval jelölt és kapcsos zárójelekkel határolt terület egy névtér. Ez szó szerint arra utal, hogy a definiált változók és osztályok nevei ezen a területen belül érvényesek. Egy névtéren belül nem definiálhatsz azonos nevű osztályokat vagy változókat. Legfelül pedig a using parancsok láthatók, így szerezhetünk meg a .NET keretrendszerből vagy máshonnan osztályokat. Jelenleg a System névtér Console osztályára lesz szükség.



7. ábra

A `static void Main(string[] args)` a belépési pont. Minden programnak kell definiálni egy `Main` függvényt (metódust), amit az operációs rendszer meghívhat indításkor. A függvényekről és eljárásokról még egy későbbi fejezetben olvashatsz bővebben.

A kapcsos zárójelek az egyes tagok (absztrakciós szintek) határait jelölik. Ez a szintaxis jellemző a C típusú nyelvekre.

Gyakorlásképpen pedig írd meg az első programodat. A „Hello world!”-öt. Minden programozó karrierjében ez az első lépés, szóval a tiéd is így fog kezdődni. Kettő paranccsal kell most megismerkedned: a kiíratással és a beolvasással. Ha egy szöveg tartalmát ki kell íratni a képernyőre, akkor a `Console.WriteLine()`-t , ha pedig be kell olvasni, akkor pedig a `Console.ReadLine()`-t kell használni. A `WriteLine` paraméterként vár egy `string` változót, amit kiírhat, ezért a „Hello world!” szöveg kiíratásához a

```
Console.WriteLine("Hello World!");
```

sort használd.

```
using System

namespace GyakorlóProgram
{
        class Program
        {
                static void Main(string[] args)
                {
                        Console.WriteLine("Hello world!");
                }
        }
}
```

2. kód

Ha megnyomod most a Start gombot (vagy az F5-öt) és lefuttatod, amit alkottál, azt fogod tapasztalni: Megjelenik egy konzolos ablak, és kiírja, hogy „Hello world!”, aztán pedig el is tűnik. Milyen kár! Sokkal jobb lenne, ha csak akkor tűnne el az ablak, ha már eleget nézegetted, és kilépésre utasítod. Mondjuk egy Enterrel. Ez a probléma azért van, mert amint a konzolos alkalmazás elérte az utolsó sorát, már nem vár semmilyen inputra. Így visszaadja a vezérlést az operációs rendszernek, aki pedig jobb híján bezárja a programot. A te feladatod, hogy ezt megakadályozd, és leblokkold a folyamatot egy beolvasási paranccsal. Adj hozzá egy Console.ReadLine() parancsot a kódhoz, és figyed meg, mi történik futtatás közben.

```
using System

namespace GyakorlóProgram
{
        class Program
        {
                static void Main(string[] args)
                {
                        Console.WriteLine("Hello world!");
                        Console.ReadLine();
                }
        }
}
```

3. kód

Megnyílik az ablak, és addig ott is marad, amíg Entert nem ütsz. Szuper! Szárnypróbálgatásként írj egy programot, ami bekéri a neved,