***compiler:*** fordító (compiler) program, ami a nyelvet lefordította gépi byte kódokra.

***Szintaktikai hiba:*** Az az eset, amikor programozási nyelv "nyelvtanában" hibáztunk valamit (pl. sor végén ; kell).

***Szemantikai hiba:*** amikor a programunk hiba nélkül lefut, de működése közben valami nem stimmel vele (pl.: prímszámokat kiíró program esetén kiírja az 1-et is).

***Program.cs:*** ezzel a fájlal kezdődik az egész, amit később lefordítunk.

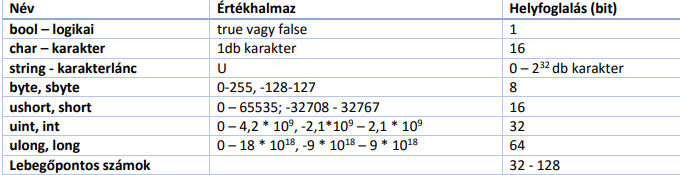
***modul:*** Az első sorba/sorokba kerül az egyes modulok közti hivatkozás, ezzel gyakorlatilag "lerövidíthetjük" a programkód írását. Használata: using modul; Pl.: using System; Ezzel beemeltük a System osztályt, ami később kell nekünk.

***névtér:*** A névtér (namespace) az a logikai egység, melyben az azonosítónak egyedinek kell lennie a többi névtértől.

***kiíratás:*** Amikor a program kimeneteként megadunk, vagyis egy kiírató utasítással kiíratunk egy változó értékét, tömb elemét, elemeit, objektum adattagját, adattagjait, kifejezést, vagy literált.

***változó:*** A változó a memóriában lefoglalt bitek azonosítója és tartalma. deklarálás - egy változónévvel ellátott típust hozunk létre; definiálás - a változónévvel ellátott típusnak értéket is adunk. A típusok egy-egy algoritmusban a megadott típus értékének tárolására használt memória tartományának elnevezése. Mérete korlátokhoz kötött.

***változó típusok:***

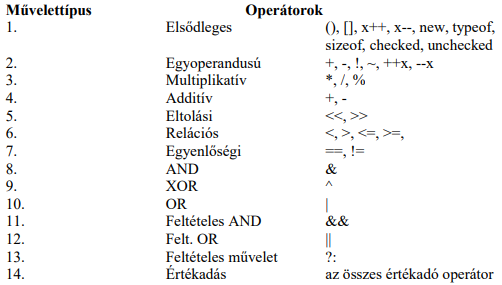


***Prefix és Postfix forma:*** A programozásban nagyon sokszor kell használni az eggyel való növelést (pl.: a = a + 1), az egyszerűbb írásmódra létrehozták a ++a (prefix) és az a++ (postfix) változatot. A prefix változatnál először növeli az a változó értékét majd a többi részt, viszont a postfix alaknál csak utólag adódik hozzá az a változóhoz az 1.

***Túlcsordulás:*** Akkor beszélünk túlcsordulásról, ha az adott érték már nem fér el a változónak fenntartott tárterületen (értékhalmaz, lásd: előző oldal). Például byte (0-255; 8bit) típusnál még a 255 belefér a tárterületbe, de ha eggyel (vagy többel) növeljük, akkor túlcsordulás következik be: a 255 kettes számrendszerben: 11111111 (8bit), ha 1-el növeljük, akkor már nem fér bele a 8bitbe így (1)00000000 lesz tehát 0 tizesszámrendszerben. C#-ban az adattípus.MaxValue/MinValue megtudhatjuk az adott típus legnagyobb/legkisebb MÉG eltárolható értékét, példa: Console.Writeline(byte.MaxValue); Ami a konzolra annyit ír ki, hogy 255.

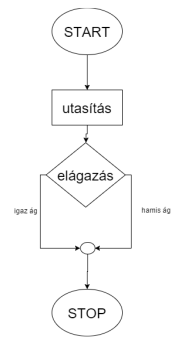
***precedenciai rangsor:*** A precedenciai rangsor egy olyan rangsor, amelyet a nyelv követ és olyan sorrendben értelmezi a műveleteket, amilyen sorrendben le van írva ebben a táblázatban. Például a matematikában is a () jeles részek után a szorzást, majd az egyéb műveleteket vesszük figyelembe.

***művelettípusok, operátorok:***

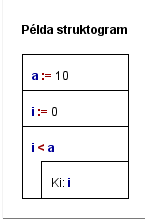
******

***algoritmust leíró eszközök:*** Az algoritmust leíró eszközök Pascal nyelvből származnak, ezért alapvető pascal tudás kell hozzá. Az értékadás a megszokott = jel helyett, itt := jelet használják. Nem kell típusokat adni a változókhoz elég, ha pl.: a := 2 használjuk. Pontosvessző nem kell!

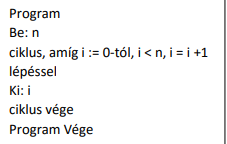
***folyamatábra:*** olyan algoritmust leíró eszköz, mely a programot gráfként írja le. Ez egy olyan irányított gráf, amely csomópontokból áll, egyetlen induló és befejező része van.



***struktogram:*** jóval könnyebben rajzolható, viszont nehezebben áttekinthető, mint a folyamatábra. Itt egy téglalap írja le az egész programot, amelyet további téglalapokra, háromszögekre osztanak.

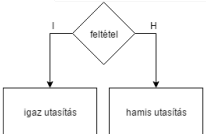
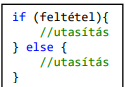


***Mondatszerű leírás:*** ez esetben a programot mondatokkal írjuk le a Program szóval és a Program vége szavakkal ér véget.

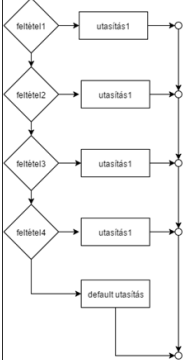


***adatbekérés a konzolról:*** Mit kell tennünk akkor, hogy ha a felhasználótól akarjuk elkérni az adatait, megkérjük, hogy adjon meg valami információt? Erre szolgálnak a Console.ReadLine() és a Console.ReadKey() függvények, a ReadLine() enter lenyomásra vár és az enter lenyomása után beolvassa a beírt szöveget, ezért stringet ad vissza, viszont ha valamilyen matematikai műveletet szeretnénk rajta elvégezni, akkor szintaktikailag hibás lesz. Ennek elkerülése érdekében át kell alakítani szám típussá a string-et, hogy működő képes legyen.  
  
***konvertálás:*** Amikor egy értéket vagy egy kifejezés eredményét a programkódban egy adott kulcsszóval, utasítással vagy kifejezéssel más típusúvá konvertálunk.

***Szelekciók – elágazások: if-else:*** Ha egy feltétel (jellemzően logikai kifejezés) teljesül, akkor különböző utasítást vagy utasítássorozatot hajt végre a program, mint amikor a feltétel nem teljesül. Az elágazásokat egymásba is lehet ágyazni, vagyis több feltételt is meg lehet adni.  
Az is elágazásnak számít, ha csak akkor hajt végre a program utasításokat, ha a feltétel teljesül, ha pedig nem teljesül, akkor nem hajt végre utasításokat a program.  
Azokat az utasításokat, amik akkor futnak le, ha a feltétel igaz, összefoglaló néven if ágnak nevezzük, esetleg az elágazás if blokkjának, azokat az utasításokat, amik akkor futnak le, ha a feltétel hamis, azokat pedig else ágnak nevezzük.



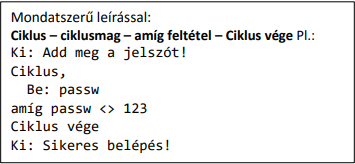
***Szelekciók – elágazások: switch:*** A switch egy olyan elágazás, amelyben szinte végtelen elágazás lehetséges.



***Iterációk – ciklusok: elöltesztelő ciklus (while):*** Az elöltesztelő ciklus "először vizsgál, és utólag végez", tehát mielőtt belépünk a ciklusmagba, a ciklus megvizsgálja a feltételt és utána végzi el az utasításokat, majd újra vizsgál és végez, újra, újra és újra, ameddig a feltétel hamis nem lesz.

***Break:*** a ciklusból való kilépést, és a ciklus futásának felfüggesztését eredményezi.  
  
***Continue:*** a ciklusfutását úgy változtatja meg, hogy ha elér a continuig a ciklusmag az utána következő ciklusutasításokat nem fogja figyelembe venni, hanem visszatér a ciklus elejére és folytatja azt, átugorja az adott ciklusmagot és a következőnél folytatja.

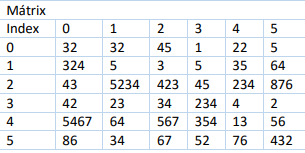
***Iterációk – ciklusok: hátultesztelő ciklus:*** "először végez, utána vizsgál", hátultesztelő ciklus a nevéből adódóan először a ciklusmagban lévő utasítást hajtja végre, majd utána vizsgálja meg a feltételt, ha a feltétel igaz, akkor újra végrehajtja az utasítást…



***Iterációk – ciklusok: számlálóvezérelt ciklus (for):*** A számlálóvezérelt ciklus (for-ciklus) teljesen úgy működik, mint egy while azzal a különbséggel, hogy itt létrehoztunk egy számlálót (i) és ennek változtatásával tudjuk a feltételt folyton vizsgálni. az első része int i = 0; létrehozzuk a segédváltozót a második része a feltétel, és végül az i++ az i értékét növeli 1-el. For-nál mindig az i++ formát használjuk! A for-ciklusnak van egy "testvére" a foreach ciklus, amit a tömbök körbejárására használunk.  
  
***A Math osztály függvényei:*** A Math osztályban található függvények a matematikában megszokott képletek helyettesítésére szolgálnak.  
  
***Randomolás:*** Sokszor előfordul, hogy egy értéket randomoljunk, magyarul véletlenszerűen generáljunk egy számot. Hogy randomolni tudjunk a System.Random osztályt kell használnunk és létre kell hoznunk egy Random objektumot névvel ellátva: Random név = new Random(); A new operátor egy objektum létrehozására szolgál.

***Összetett adatszerkezetek: Tömbök:*** A tömbök is adatok tárolására alkalmas, viszont a sima adatszerkezetekkel ellentétben ez már egy összetett adatszerkezet, ami képes egy bizonyos változónév alatt több elemet is tárolni egy index segítségével.

Ha a tömb egy dimenziós (egy indexe van) ***vektornak***, ha 2, akkor már ***mátrixnak*** nevezzük.



***Bővebben a mátrixról:*** a mátrix egy olyan kétdimenziós tömb melynek van szélessége és magassága is, míg a vektor csak 1 magas volt és bármekkora széles addig ez már legegyszerűbben egy táblázatként fogható fel, használata majdnem ugyan az, mint a vektornál csak itt már két indexet kell megadnunk, hogy elérjünk egy cellát.

***mutatóvektorok***: olyan többdimenziós tömbök, melyeknek a mutató vektoron (fő vektor) belül lévő vektorok mérete nem azonos.  
  
***konstans:*** létrehozhatunk c# nyelvben olyan változókat melyeknek az értéke állandó, tehát biztos, hogy nem fog változni, így a program csak a változó értékét foglalja le a memóriában. Létrehozhatunk konstans tömböt is (pl.: a magyar abc-t tartalmazó betűkészlet tömböt), általános létrehozási alakja, amit általában a deklaráláskor kell megadnunk: const változótípus változónév = érték;

***Alprogramok:*** A programok felépítése alapvetően szekvenciális , a főprogram a Main a programunk belépési pontja, innen kezdődik a futás, az itt meghívott függvények hajtódnak végre. Az alprogramok a programkód lerövidítésére és átláthatóságára szolgál, minden egyes saját eljárásnak vagy függvénynek meg van a maga szerepe és a szerepére utaló neve, így ha valamilyen javítandó probléma lenne a programunkban, akkor könnyen nyomon követhető a program felépítése és jobban korrigálható.

***Eljárások:*** nincs visszatérési értéke, ezért void (üres) típusú.  
  
***Függvények:*** van visszatérési értéke, a visszaadott értéktől függ, a return utasítással adjuk meg a visszaadott értéket, ami minden esetben kötelező a megadása.

***Az alprogramok paraméterezése:*** A metódus deklarálásánál megadott paramétereket formális, a metódus meghívásakor megadott paramétereket pedig aktuális paraméternek hívjuk. Az eljárásokat és függvényeket 3-féle módon paraméterezhetjük: érték, cím és egy u. kimeneti paraméterként, a paramétereket vesszővel választjuk el egymástól, ha értéket is adunk az alprogramunk fejlécében a paramétereknek, akkor a meghíváskor nem lesz kötelező megadni az adott paramétert.

**1. Érték sz. paraméterátadás:** Ha nem adunk meg semmilyen kulcsszót, akkor ilyen módon adódnak át a változók (kivéve tömbök, objektumok... ). Ilyenkor az alprogram meghívásakor megadott aktuális paraméterek értéke átmásolódik az alprogram egy-egy megfelelő formális paraméterbe, ezután a formális paramétert ugyanúgy használhatjuk, mintha egy változó lenne.

**2. Cím (Referencia) sz. paraméterátadás:** A ref kulcsszóval érhető el a cím sz. p., a ref kulcsszót az aktuális és a formális paraméter előtt is kell használni! Ebben az esetben az átadott aktuális paraméterre, változóra egy hivatkozás jön létre a memória tartományára, csak a memóriaterület címe/azonosítója adódik át, így az alprogramon belül végzett változtatások érvénybe lépnek az átadott változóban is!  
**3. Kimeneti változó:** Az out kulcsszóval érhető el ez a paraméter átadási mód, szintén a meghíváskor és az alprogram deklarálásakor is meg kell adni! Hasonlít a cím sz. p. -ra viszont az átadott paraméternek kezdetben nincs érétke a függvényünk hívatott ennek értéket adni, de kötelező az értékadás!

***Rekurzió:*** Az olyan műveletet, melynek végrehajtásakor a saját műveletsorait hajtja végre, tehát önmagát ismétli, rekurziónak nevezzük. Programozás esetén a függvények saját magukat hívják meg. Amikor egy függvény nem önmagát, hanem egy másik függvényt hív meg, majd ez a függvény pedig meghívja azt a függvényt, amiből meghívták (…), akkor azt kölcsönös rekurziónak nevezzük (pl. A() meghívja B()-t, majd B() meghívja A()-t…). Figyelem: A rekurzió nagyon erőforrás igényes, hiszen akár több százszor is létrejönnek a függvény változói és ezzel memóriát foglalnak el.  
  
***A felsorolás típus – Enum:*** A felsorolás egy olyan adatszerkezet, amelyben egy meghatározott konstans (nem változhat) értékek adategységét jelenti. Enum típust csak eljárásokon kívül, de még osztályon belül szabad csak deklarálni, különben szintaktikai hibát kapunk. Általános alakja:

***Hibakezelés:*** Abban az esetben eddig, ha egy nem kezelt kivétel (Exception) keletkezett be, akkor kifagyott a programunk és a hibát vagy a visual studio, vagy az operációs rendszer kezelte le, és egy hibaüzenetet dobott. C#-ban a kivétel egy objektum, amit akkor indít útjára egy függvény, ha valamilyen hiba történt, ekkor a programunk hibakezelés nélkül elszáll. A kivételek elkapását és lekezelését a try-catch blokk végzi, a try blokkot védett blokknak nevezik, itt keletkezhet a kivétel, melyet majd a catch blokk fog majd lekezelni, a catch-nak meg kell adni egy kivétel objektumot, ami utal a hiba típusára, egy try-blokk után több catch-blokk is következhet, hiszen érdemes a különböző kivételekre, különböző módon reagálni, nem csak azt kiírni, hogy Hiba!  
**Finally blokk:** a finally blokkot minden esetben az utolsó blokk-ként kell használni, az itt megadott utasítások minden esetben lefutnak, ha keletkezett kivétel, ha nem!

***Generikusok:***

***Listák - List -Generikusok:*** A lista ezzel szemben egy dinamikus tömb, méretét automatikusan változtatja. A lista egy generikus típus, tehát bármilyen típusból (akár tömbből, osztályból, struktúrából, listából…) létrehozhatunk saját listát.

***Referencia másolás:*** Mint már említettem a lista egy referencia típus, tehát cím szerint adódik át, ennek eredményeként, ha egy már meglévő listából szeretnénk létrehozni egy újat, akkor nem a lista értékei, hanem a címe másolódik át az újonnan létrehozott listánkba, ezt nevezzünk értékadáskor bekövetkező referencia másolásnak, így a listákon külön-külön végzett változtatások hatással lesznek a másik listára is.

***Összetett Listák:*** Tömbök esetében megtudtuk tenni azt, hogy egy vektoron belül létrehozzunk még tömböket, hát listák esetében se sincs ez máskép, így gyakorlatilag kiküszöbölhető az 1dimenziós határ. Nézzük meg hogy hogyan: A lista deklarációjánál a <> részek közé egy újabb List veszünk fel, és ezután pedig a {} jelek közé új listákat hozunk létre vesszővel elválasztva, új belsőlistát a főlista.Add(new List)- vel tudunk felvenni, a belső listákat az indexük segítségével tudunk elérni, és úgy felvenni bele elemeket. Létrehozhatunk struktúrából, tömbből, osztályból, halmazból is listákat, hasonlóan, mint itt, de azt nem tárgyalom ennél a résznél.

***Láncolt listák - LinkedList -Generikusok:*** A sima listák esetében az adatokat egy háttértömbben tároljuk el, így ez az adatszerkezet lassabb lesz akkor, ha sok mozgatást végzünk benne, pl. az Insert metódussal új elemeket szúrunk be a lista elejébe, ekkor az azt követő értékeket 1-el arrébb kell tolni, és ez nagyon időigényes. A lassúság kiküszöbölésére hozták létre a láncolt listákat (LinkedList), amit akkor érdemes használni, ha adatstruktúránkban sok mozgatás van, fontos megjegyezni, hogy a láncolt lista nem biztos, hogy gyorsabb, mint a sima lista, amíg a listánkban csak a végére szúrunk be elemeket addig teljesen elég. A láncolt listák nagyobb helyet foglalnak, mint a sima elődjeik, a láncolt lista több, egymáshoz hivatkozás által összekapcsolt úgy. csomópontokból épül fel, ebben a csomópontban szerepel egy adat, valamint egy (egyszeresen láncolt lista) vagy kettő (kétszeresen láncolt lista) hivatkozás, ami az előző és/vagy a következő csomópontra hivatkozik.



***A Csomópontok - LinkedListNode -Generikusok:***