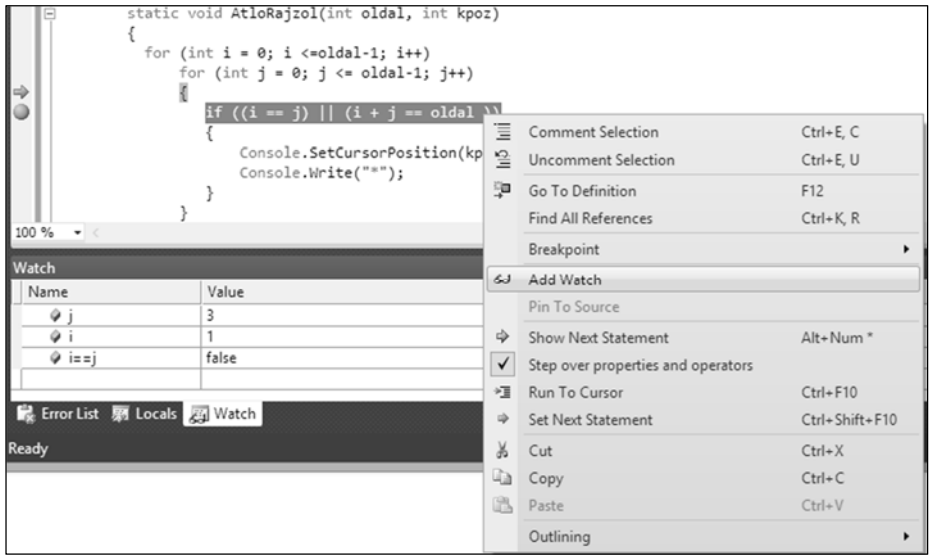


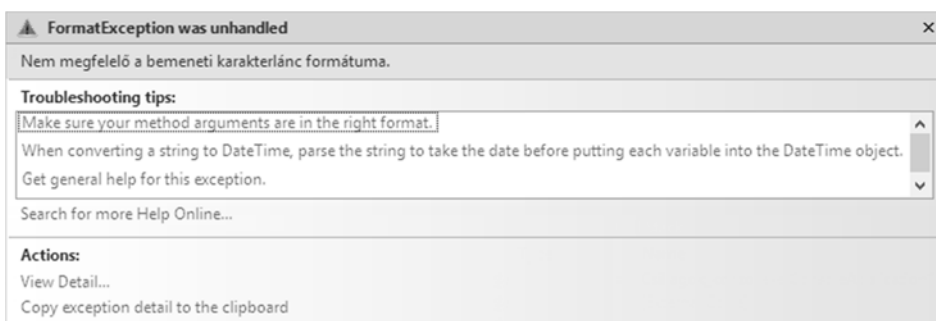
A hiba okát így már biztos sikerül kideríteni. Az elágazás feltételében az oldal helyett `oldal-1` értéknek kell kerülnie, hiszen a ciklusok 0-tól az `oldal-1` értékig mennek.

```
if ((i == j) || (i + j == oldal-1))
```

Kivételkezelés

Bár azt gondoljuk, hogy programunk most már jól fog működni, de még mindig akadnak olyan problémák, amelyekkel akár a program futása is megakad.

Nézzünk néhány ilyen esetet! A felhasználó véletlenül vagy épp direkt karaktereket ír számok helyett. Esetleg túl nagy számot ír be, ami a típushatáron kívül van, ezzel aritmetikai túlcsordulást idéz elő. Előfordulhat az is, hogy negatív számot, esetleg nem egész számot ír be.



Az ilyen helyzetekre is fel lehet készíteni a programot. A legtöbb esetben a megfelelő típusválasztással, illetve az adatbevitel korlátozásával az ilyen problémák kezelhetőek. De előfordulhatnak olyan, a felhasználótól független problémák és hibák, amiket egy hardver meghibásodás vagy esetleg egy szoftver hibás működése okozhat.

Az ilyen esetekben nem hibáról, hanem kivételről beszélünk.

A kivétel a program futását szándékosan vagy nem szándékosan megszakító esemény. A magas szintű nyelvek esetén **a kivételek felismerésére és kezelésére** külön programozási eszközt, az ún. **kivételkezelőt használják**.

A kivételek kezelésének két lehetősége van. Az egyik, amikor a keretrendszer generálja a kivételt, és ezt kapjuk el, a másik pedig, mikor mi magunk generáljuk a kivételt.

A C# nyelv esetén a kivétel is egy objektum, ami a kivétel keletkezésekor jön létre.

A kivételek esetén a teendőnk, hogy figyeljük azokat a programrészleteket, ahol a kivétel előfordulhat. Ha kivétel generálódik, akkor „el kell kapni” a kivételt, és végül kezelnünk kell a problémát.

Az erre létrehozott nyelvi szerkezet a *Try...Catch* strukturált kivételkezelő.

```
try
{
    a lehetséges hibaforrást okozó sorok.
}
catch       első lehetséges ok
{
    a kivétel kezelése ebben az esetben
}
catch       második lehetséges ok
{
    a kivétel kezelése ebben az esetben
}
finally
{
    akár volt kivétel, akár nem ez a blokk végrehajtódik.
}
```

A kivételkezelő struktúrában a try részt szokták még védett blokknak is nevezni. A catch blokk feladata, hogy a keletkezett kivételeket elkapja és kezelje. A catch kulcsszó után meg kell adnunk egy olyan Exception típusú objektumot, aminek tulajdonságaival és metódusaival elég jól azonosítható a hiba. (De nem feltétlenül kell felhasználni!)

Ha a kivétel illeszkedik a catch részben megadott kivételre, akkor a blokkban lévő utasítások végrehajtódnak.

Egy védett blokkhoz akár több catch is tartozhat.

Példa: az előzőekben elkészített program esetén vizsgáljuk meg az adatbevitelt. Azt szeretnénk megoldani, hogy a felhasználó az adatbevitelnél ne adhasson meg szöveget, túl nagy számot stb.

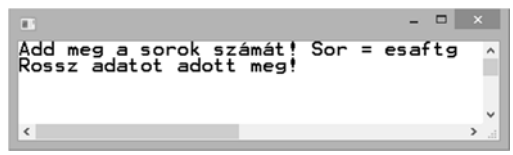
Kezelhetjük egyszerűen és egy kicsit kifinomultabban is a kivételeket.

(Az egyszerűbb átláthatóság kedvéért a következő példákban az adatbekérés rész a kiírásokat és pozicionálásokat nem tartalmazza.)

Az egyik megoldás, hogy amennyiben elrontja az adatbevitelt, akkor kiírjuk a hibaüzenetet, és kilépünk a programból.

A konzol alkalmazásokból az `Environment.Exit()`metódussal tudunk kilépni.

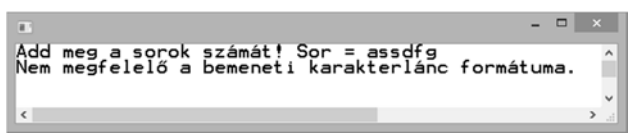
```
try
{
     sor = int.Parse(Console.ReadLine());
     oszlop = int.Parse(Console.ReadLine());
}
catch (Exception kivetel)
{
     Console.WriteLine("Rossz adatot adott meg!");
     Console.ReadLine();
     Environment.Exit(0);
}
```



Ezzel az a probléma, hogy a felhasználó igazából nem tudja meg pontosan, hogy mi is volt a hiba. (Persze itt nyilvánvaló, hogy ha számot várunk, akkor a szöveg nem megfelelő.)

Próbáljunk javítani a kivétel okának kiírásán azzal, hogy felhasználjuk a `kivetel` objektumot a kiírás során, de még mindig kilépünk a programból.

```
try
{
     sor = int.Parse(Console.ReadLine());
     oszlop = int.Parse(Console.ReadLine());
}
catch (Exception kivetel)
{
     Console.WriteLine(kivetel.Message);
     Console.ReadLine();
     Environment.Exit(0);
}
```



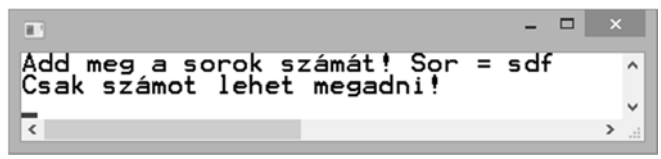
A létrejövő kivétel objektum *Message* tulajdonsága tartalmazza azt az üzenetet, amely a hiba okáról ad információt. Ez már sokkal informatívabb megjelenítés, de még mindig lehet ezt fokozni.

Próbáljuk meg szétválasztani az egyes eseteket, és egyedi hibaüzeneteket küldeni a felhasználóknak!

```
try
{
    sor = int.Parse(Console.ReadLine());
    oszlop = int.Parse(Console.ReadLine());
}
catch (OverflowException)
{
```

```
        Console.WriteLine("Túl nagy számot adtál meg!");
        Console.ReadLine();
        Environment.Exit(0);
    }
    catch (FormatException)
    {
        Console.WriteLine("Csak számot lehet megadni!");
        Console.ReadLine();
        Environment.Exit(0);
    }
    catch (Exception kivetel)
    {
         Console.WriteLine(kivetel.Message);
         Console.ReadLine();
         Environment.Exit(0);

    }
```



Mint látható, itt a lehetséges hibák közül kettőt külön is kezelünk. Az egyik, amikor túl nagy számot ad meg a felhasználó, a másik eset, amikor pedig betűket ír be számok helyett. Az *Exception* ősosztályból rengeteg osztály származik. Ezek speciális, gyakran előforduló kivételekhez kapcsolódnak.

Ezek közül az egyik az *OverflowException* osztály, amely a típus által meghatározott intervallumon való túlcsordulást jelzi, a másik a *FormatException* amely a típuskonvertálások során keletkező hibákat jelzi.

Az `Exception kivetel` rész nem véletlenül került a harmadik *catch* blokkba. Egyrészt azért, mert ezen a két hibán kívül is előfordulhat még bármi. Másrészt a kivételek felépítése hierarchikus. A hierarchia tetején az *Exception* osztály van. Ezért, ha ilyen részletesen kívánjuk a kivételeket kezelni, akkor mindig a speciális kivételek elkapásával kezdjük, és a végére hagyjuk az általános kivételosztályt.

Láthatjuk, hogy elég a *catch* blokknál csak a kivétel osztályát (típusát) megadnunk. Fontos, hogy ekkor viszont az objektum egyéb lehetőségeit nem tudjuk kihasználni.

Kevésbé elegáns az a megoldásunk, hogy ha a felhasználó az adatbevitel során hibázik, akkor kilépünk a programból. Azt is érdemes lenne megoldani, hogy a kivétel kezelése után újra bekérjük az adatokat.

Ezt a legegyszerűbben egy dinamikus ciklussal tehetjük meg, azaz addig kérjük be az adatokat, amíg a felhasználó helyesen nem adja meg azokat.

Ehhez hozzunk létre egy logikai változót, ami majd a ciklusfeltételbe bekerül! A ciklusváltozó értéke mindaddig igaz marad, amíg az adatok beolvasása nem dob kivételt.