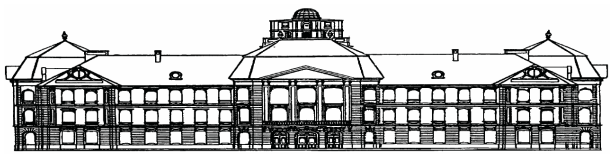
A jegyzet a TÁMOP-4.1.2-08/1/A-2009-0038 támogatásával készült.



Eszterházy Károly Főiskola
Matematikai és Informatikai Intézet

Bevezetés az informatikába

Alcím: Szemelvények

Dr. Kovács Emőd, Biró Csaba, Dr. Perge Imre

Eger, 2013

Tartalomjegyzék

Ábrák jegyzéke

Bevezetés

Ez a jegyzet három szerző műve. Sajnos Dr. Perge Imre már nem érhette meg a jegyzet megjelenését, mivel 2011. május 27-én meghalt.

Dr. Perge Imre főiskolai tanár 1932. május 31-én született Sirokban. 1953-ban matematika-fizika szakon végez Egerben, majd 1958-ban az ELTE alkalmazott matematikus szakán szerez diplomát. Az Eszterházy Károly Főiskolán 43 éven keresztül oktatja a hallgatókat, közben a Matematika Tanszék számítástechnikai csoportját vezeti. 1972-ben elkészíti a tanárképző főiskolák számítástechnikai oktatási tantervét. 1984-től főigazgató-helyettesi megbízást kap, melyet hat éven át tölt be. 1990-ben vezetésével létrehozzák a Számítástechnikai Tanszéket.

Perge Tanár Úr alapítója és meghatározó egyénisége a számítástechnikai ismeretek oktatásának Egerben. 10 szakkönyve és több mint 20 tanulmánya jelent meg. Munkáját 1989-ben „Tarján-díjjal" ismeri el a Neumann János Számítógép-tudományi Társaság.

Perge Imre kérése volt, hogy jegyzete átdolgozva ismét kiadásra kerüljön. Biró Csaba kollégámmal együtt reméljük, hogy ezt a feladatot sikerült méltóképpen teljesítenünk.

Mivel ez a jegyzet ingyenesen elérhető mindenki számára, reméljük, hogy nemcsak a hallgatók, hanem más érdeklődők is szívesen olvassák. Minden esetleges hibát, észrevételt szívesen veszünk a kedves olvasótól az alábbi email címen: emod@ektf.hu. Az olvasók segítségével és az elektronikus jegyzet adta formai lehetőség révén a javítást rövid időn belül tudjuk elvégezni.

A szerzők nevében Dr.Kovács Emőd

1. SZÁMÍTÁSTECHNIKA - INFORMATIKA

1.1. A számítástechnika, informatika helye a tudományok rendszerében

A tudomány kialakulásának a kezdetén csak egyetlen egységes tudományról, a *filozófiáról* beszélhetünk, amely más tudományok „csíráit" is magába foglalta. A filozófia a valóság egészét és nem valamely speciális szakterületet hivatott vizsgálni. Az idők folyamán leváltak róla a *szaktudományok,* amelyeket két csoportba oszthatunk:

$$\boxed{\text{TERMÉSZETTUDOMÁNYOK} = \{\ldots,\ \text{fizika, kémia, biológia,}\ \ldots\}}$$

illetve

$$\boxed{\text{TÁRSADALOMTUDOMÁNYOK} = \{\ldots,\ \text{történelem, irodalom,}\ \ldots\}}$$

A tudományok differenciálódása még napjainkban is folyik. A szaktudományok szintjén újabb határtudományok keletkeznek pl. biokémia, fizikaikémia stb., amelyek előbb utóbb önálló tudományt képviselnek a tudományok rendszerében.

A fenti osztályozási rendszerből azonban hiányzik még néhány fontos tudomány. Ilyen pl. a matematika, amely az egyik legrégebbi tudomány. (Az ókor filozófusai általában matematikusok is.) A matematika a tudománycsoportokhoz való viszonyában, eredetét tekintve, közvetlen kapcsolatban áll a természettudományokkal, de ő maga nem az. A matematika ugyanis nem egy szaktudományra jellemző *primer absztrakciókkal* foglalkozik, hanem ezen absztrakt fogalmak absztrakcióival, vagyis *szekunder és ennél* magasabb absztrakciókkal is. Ennek köszönhető, hogy ma a matematika alkamazási területei valamennyi szaktudományban jelentősek.

A matematika, tehát ilyen értelemben, a szaktudományok és a filozófia között egy újabb szinten helyezkedik el. Nevezzük ezt a szintet az *általános tudományok* szintjének. További kérdés, hogy ezen a szinten van-e más kialakult új tudomány. Sok szempontból hasonló a helyzet a *kibernetikával* is,

mint a matematikával. A vezérlés, szabályozás, automatizálás is számtalan mozgásformára, élő és élettelen szervezetre alkalmazható.

ÁLTALÁNOS TUDOMÁNYOK = {..., matematika, kibernetika, ...}

A matematika és a kibernetika határtudományaként jött létre napjainkban a *számítástechnika,* mint általános tudomány, amely ugyancsak a valóság minden területén és napjainkban szinte minden szaktudományban alkalmazható. A számítástechnika a matematikai modellek, eljárások és módszerek segítségével és kibernetikai technikai eszközök felhasználásával (amelyek közt kiemelkedő szerepet kap a számítógép), az *információ* átalakításával, feldolgozásával foglalkozik. Az információ a való világban, az anyaghoz és az energiához hasonló szerepet kap, amelyet a későbbiek során pontosítani fogunk.

A számítástechnika elnevezés nem a legszerencsésebb (hasonló volt a fizikában a „lóerő" elnevezés, amely sem nem ló, sem nem erő), mivel sokkal általánosabb célokat valósít meg mint a számítás, vagy egyszerűen a technika. Az elnevezés több nyelven is hasonló a magyarhoz. A francia, illetve a német szakirodalomban a számítástudomány szónak az *informatique,* illetve az *Informatik* szó felel meg. Napjainkban ennek magyaros *informatika* alakja hazánkban általánossá vált. Tudnunk kell azonban, hogy az informatika nevet nemzetközi szinten már lekötötték régebben a könyvtárüggyel és dokumentációval foglalkozó tájékoztatásra, amely maga sem nélkülözheti a számítástechnikát. Ilyen értelemben az informatika elnevezés sem szerencsés. A szóbanforgó tudomány kialakulása és sikere azonban nem az elnevezésen múlik. Határtudomány lévén több olyan tudományterület is tárgyalásra kerülhet, amely más tudományokhoz kapcsolja.

ÁLTALÁNOS TUDOMÁNYOK=
= {..., matematika, számítástechnika, kibernetika, ...}

A tudományok itt ismertetett osztályozását nem tekintjük lezártnak, újabb szintek keletkezhetnek és az egyes szinteken is újabb tudományok jöhetnek létre.

1.2. Magyarországi informatikai programok az ACM 2005-ös oktatási programjainak tükrében

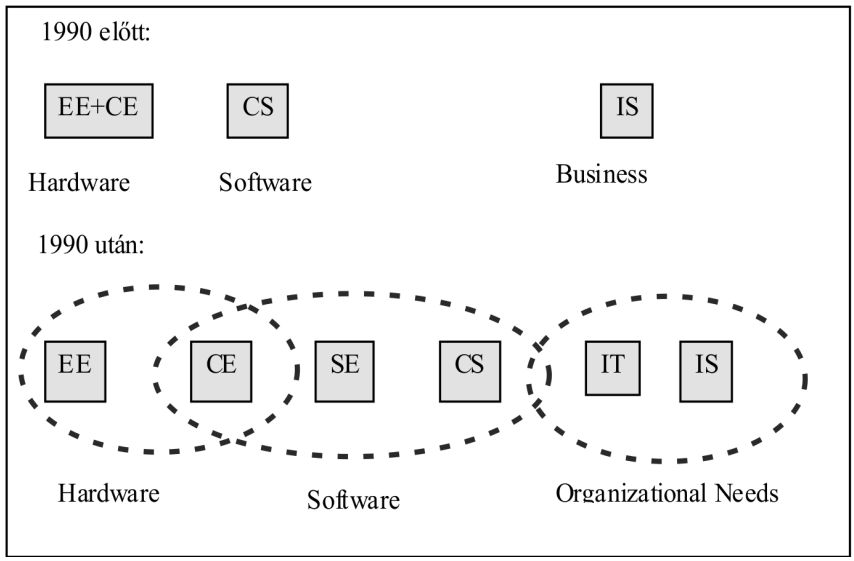
1.2.1. Computing Curricula 2005

A számítástudományon, bővebb értelmezésként az informatikán belül már a hatvanas években felmerült az igény, hogy ajánlásokat fogalmazzanak meg a képzési programok tartalmára vonatkozóan. A BSc és MSc képzési programokon belül a minta a képzés jellegéből adódóan angolszász kell hogy legyen. A táblázatban ismertetett, a terület legjelentősebb szakmai szervezetei 1960-tól dolgoznak ki folyamatosan szakmai ajánlásokat. 1991-től már közös ajánlásokat jelentettnek meg.

Legjelentősebb szakmai szervezetek:	Webcím
Association for Information Systems, AIS	start.aisnet.org
Computer Society of the Institute for Electrical and Electronic Engineers, IEEE-CS	www.computer.org
Association for Machinery, ACM	www.acm.org
Association for Information Technology Professionals, AITP	www.aitp.org

2005 áprilisában ennek a közös munkának eredményeképpen jelent meg a CC 2005, először draft, un. vázlat formában, majd átdolgozás után 2005 szeptemberétől már a végleges változat is publikálásra került. A CC 2005 az informatikán (computing) belül öt szakterületet (diszciplínát) definiál, ennek megfelelően öt BSc programot határoz meg: Computer Engineering (CE), Computer Science (CS), Software Engineering (SE), Information Systems (IS), Information Technology (IT). A következőkben röviden ismertetjük ezeket a programokat.

2008-ban a Computer Science és a Information Technology (IT), 2010-ben az Information Systems (IS) curriculumát frissitették. Ezeket az ajánlásokat bárki szabadon letöltheti az alábbi webcímről:
http://www.acm.org/education/curricula-recommendations.



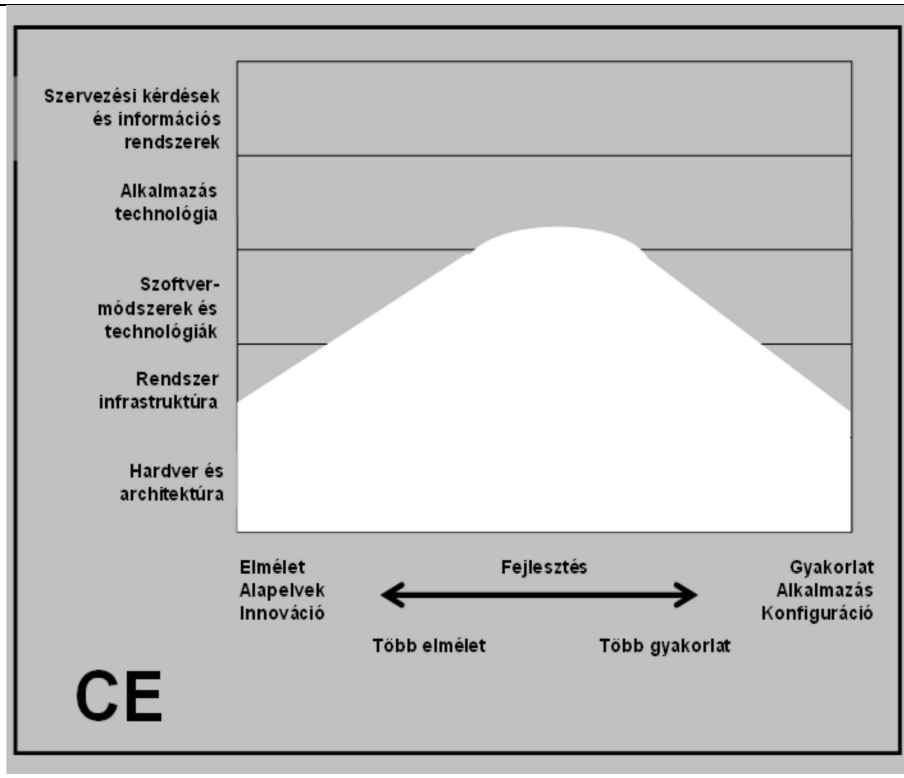
1.1. ábra. A képzési programok változása

A képzések informatikai terét koordinátarendszerben ábrázolva mutatja meg az egyes diszciplínák által lefedett tartományokat. A vízszintes tengelyen balról jobbra haladva az elmélettől a gyakorlat felé haladunk, míg függőlegesen alulról felfelé az informatikai rendszerek jellegzetes rétegein haladhatunk végig a hardver és architektúra szintjétől a rendszer infrastruktúrán, a szoftvermódszerek és technológiákon, az alkalmazástechnológián át a szervezési kérdések és információs rendszerekig. Az öt diszciplína elhelyezkedését az informatikai térben a táblázat mutatja, s megtaláljuk a rövid értelmezést is.

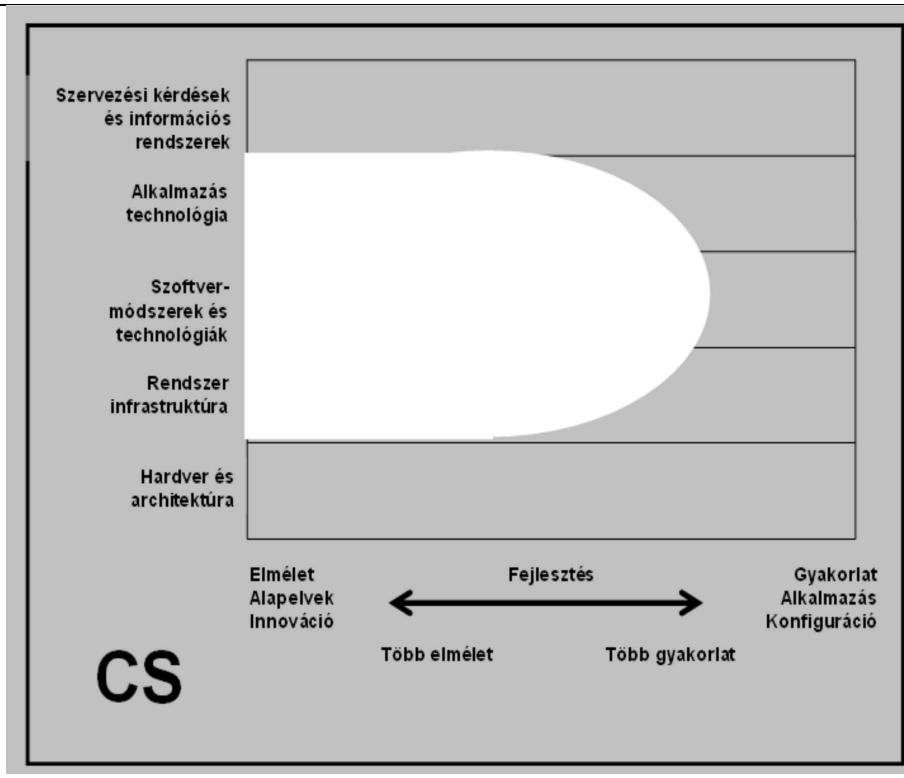
A részletesebb elemzéshez az informatikai teret a CC 2005 36 témakörre osztja. Emellett 21 nem informatikai témakört is bevezet, 59 készséget, képességet (kompetenciát) is definiál az ajánlás.

1.3. Magyarországi helyzet

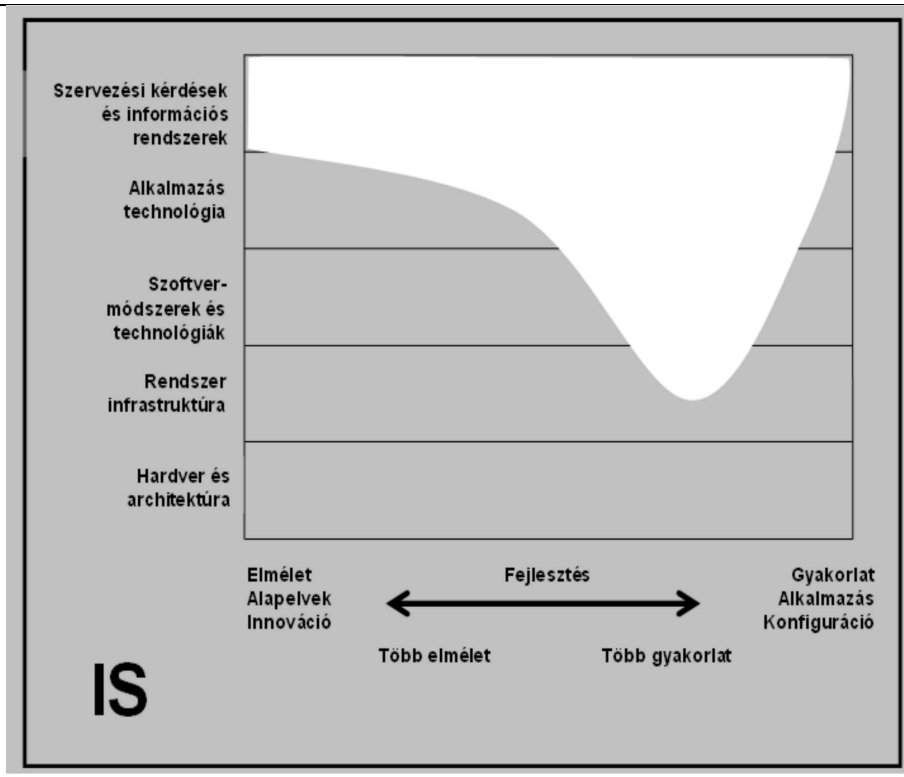
Magyarországon is hasonló folyamat játszódott le, mint az angolszász oktatást alkalmazó országokban. Lényeges változás a megfordíthatatlan Bologna folyamat elindulása, és egységes bevezetése. 2006 szeptemberétől már csak



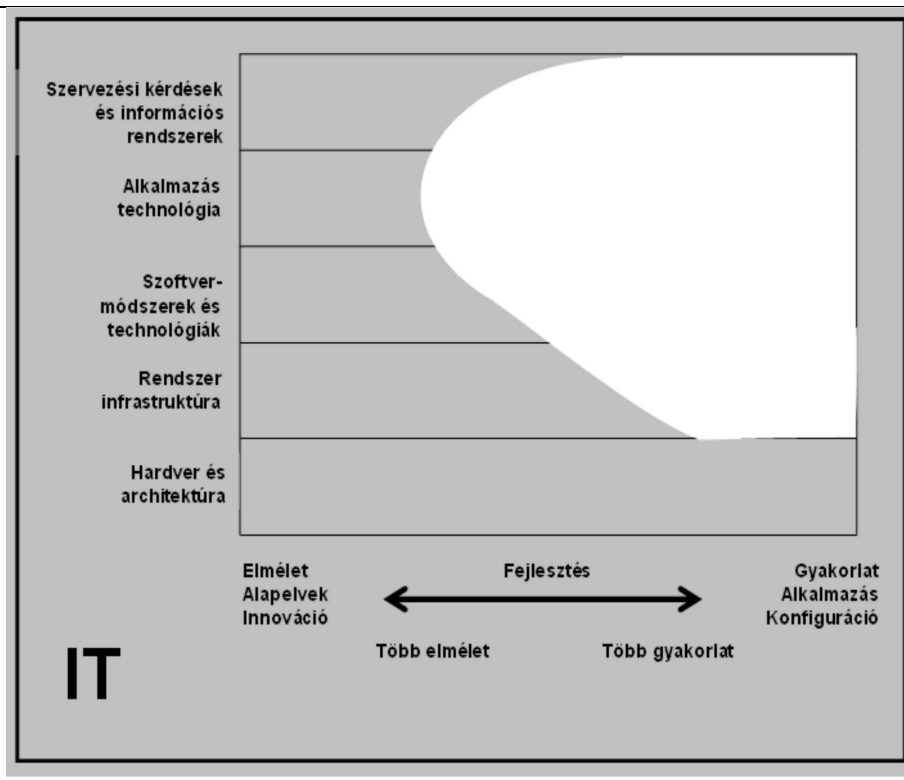
A **Computer Engineering** számítógépek és számítógépeken alapuló rendszerek tervezésével és fejlesztésével foglalkozik. A képzésben mind a hardver, mind a szoftver, mind pedig a kettő egymásra tett hatása jelentős. A képzési programokban erős súllyal jelennek meg a villamosmérnöki és matematikai ismeretek, illetve ezek informatikai alkalmazása. A Computer Engineering hallgatók tanulnak a digitális hardver rendszerek tervezéséről, beleértve a számítógépet, a kommunikációs rendszereket és eszközöket, és minden olyan eszközt, amely tartalmaz számítógépet. Tanulnak szoftverfejlesztést is elsősorban a digitális eszközökre, s nem a felhasználókra koncentrálva (beágyazott szoftverek). A tananyagban a hangsúly a szoftver helyett a hardverre van inkább helyezve nagyon erős mérnöki aspektussal.



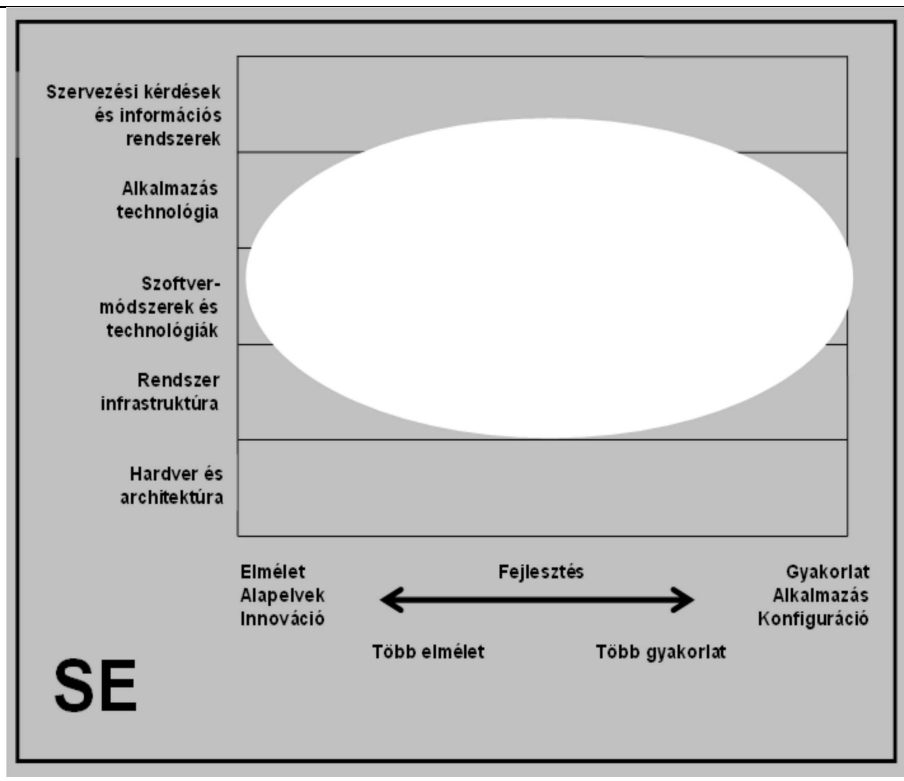
A **Computer Science** magában foglalja a szoftvertervezést és implementációt, az informatikai problémák eredményes és hatékony algoritmikus megoldási módszereit, valamint a számítógépek új felhasználási útjainak keresését. Ez a leginkább általános tudást adó képzési program, szemben a többi speciális képességeket kívánó diszciplínával.
Három fő terület:
Hatékony módszerek keresése számítási problémák megoldására.
A számítógéphasználat új területeinek a tervezése, keresése.
Szoftverek tervezése és megvalósítása



Az **Information Systems** fő célja az információtechnológia és az üzleti folyamatok megfelelő szintû integrálása. A hangsúly meghatározóan az információn van, ezért az IS tanulmányokban az informatikai tárgyak mellett jelentős a gazdasági tárgyak súlya.
Gazdasági problémák felismerése, az informatikai támogatás és/vagy fejlesztés kezdeményezése, szükség szerint végrehajtása az üzleti területtel és egyéb informatikai szakemberekkel együttmûködve, felhasználva a modellezési és fejlesztési eszközöket. Informatikai rendszerek és a szervezet menedzselése, kisebb fejlesztési és üzemeltetési projektek tervezése és irányítása, együttmûködés informatikai feladatok outsourcing megoldásaiban.



Az **Information Technology** az Information System-mel szemben sokkal inkább a technológiára koncentrál, ezért e terület technológus feladata megbízható informatikai háttér rendelkezésre állásának biztosítása, karbantartási, upgrade, installálási, adminisztrátori feladatok ellátása.



A **Software Engineering** a megbízható, hatékony, nagy és költséges szoftverek, szoftverrendszerek tervezésével, fejlesztésével és mûködtetésével foglalkozik. Közel áll a Computer Science-hez, de kevésbé általános tudást nyújt. Mérnöki szemlélettel szigorúan és gyakorlati nézőpontból tekintenek a szoftver megbízhatóságára és karbantarthatóságára. Úgy koncentrálnak a fejlesztési technikákra, hogy elkerüljék a költséges és veszélyes szituációkat, amely egy szoftver életciklusa során bekövetkezhet.

az új rendszerû BSc szakok indulhattak Magyarországon. A régi osztatlan rendszerben 500 alapszak helyett kb.100 BSc., BA alapszak jött létre. Mivel az informatika képzési terület úttörő szerepet játszott, ezért már korábban, 2004-ben elindult az első programtervező informatikus Bsc képzés Debrecenben a Debreceni Egyetemen, s 2005-ben a főiskolák között elsőként Egerben is, az Eszterházy Károly Főiskolán.

Az első képzési ciklus alapszakjait, a korábbi és új szakok megfeleltetését a többciklusú lineáris képzési szerkezet bevezetésének egyes szabályairól és az első képzési ciklus indításának feltételeiről szóló 252/2004.(VIII.30.) Korm. rendelet határozza meg. 2013-ban a felsőoktatási intézmények az alábbi informatikai szakokon[1] indítanak képzést:

Felsőoktatási szakképzés (szakirányok), 2 éves képzés
gazdaságinformatikus
mérnökinformatikus (rendszergazda, hálózati informatikus)
programtervező informatikus (fejlesztő, multimédia)

alapszak (BSc)	mesterszak (MSc)
gazdaságinformatikus (GI)	gazdaságinformatikus
mérnökinformatikus (MI)	info-bionika
programtervező informatikus (PTI)	mérnökinformatikus
	orvosi biotechnológia
	programtervező informatikus

[1] Az info-bionika mester szak a tervek szerint átkerül a mérnöki képzések közé.

A következő táblázat az informatikai alapszakok jellemzőit mutatja.[2]

	GI.	MI.	PTI
Szemeszter	7	7	6
Össz. kredit	210	210	180
Természettudományi alapismeretek	20-40	40-50	
Gazdasági és humán ismeretek	30-40		20-25
Szakmai törzsanyag			
Rendszertechnika	10-20	30-55	10-20
Programozás	10-20	20-30	30-50
Informatikai rendszerek	40-60	20-30	10-20

A Magyarországon más országokhoz hasonlóan nem lehet teljesen megfeleltetni a BSc szakokat a CC 2005 ajánlásában szereplőkkel. Az alapképzési szakokra az alábbi megállapítások tehetőek. A mérnök informatikus szak a Computer Engineering (CE) és a Computer Science (CS), a gazdaságinformatikus szak az Information Systems (IS), Information Technology (IT), és a programtervező informatikus a Computer Science (CS) és a Software Engineer (SE) képzések tartalmát ötvözi, kicsit hasonlóan a 1990-es évek előtti amerikai állapotokhoz. Más képzési területek is hordoznak informatikai tartalmat: informatikus könyvtáros, társadalomtudományi képzési terület, informatikus és szakigazgatási agrármérnök, agrár képzési terület.

2006-os jelentkezési és felvételi adatok azt tükrözik, hogy az informatika az ötödik legnagyobb képzési területté nőtte ki magát Magyarországon. Számos felsőoktatási intézményben jöttek létre informatikai karok, és intézetek. A szakma elérte, hogy 2006 őszétől önálló Informatikai Bizottsággal képviselteti magát a Magyar Rektori Konferencia mellett működő bizottságok között.

[2] A táblázat a Selényi Endre: Informatika szakok a kétciklusú képzésben: eredmények és tervek[5] cikkéből származik.

2013. szeptemberben induló képzések

Jelentkezők megoszlása - jelentkezési helyük szerint
» **Jelentkezők száma képzési területenként**

« Vissza az előző oldalra

Képzési terület ▲	Jelentkezők				
	Össz.	Első	[AO]N	Mester-képzés	Állami
agrár képzési terület	7893	4703	6140	674	7315
bölcsészettudomány képzési terület	13314	8984	8489	3562	11665
gazdaságtudományok képzési terület	21114	15341	14002	3612	14572
informatika képzési terület	8319	5725	6607	697	7552
jogi képzési terület	5139	3517	3056	330	3339
közigazgatási, rendészeti és katonai képzési terület	7808	5492	4817	1065	6375
műszaki képzési terület	19536	14532	14626	2583	18260
művészet képzési terület	4699	4092	3757	832	4368
művészetközvetítés képzési terület	1076	543	949	0	791
orvos- és egészségtudomány képzési terület	9165	7140	7593	643	8959
pedagógusképzés képzési terület	13986	10299	7454	4667	12831
sporttudomány képzési terület	4094	2628	3086	293	3734
társadalomtudomány képzési terület	8913	4553	5461	2392	7155
természettudomány képzési terület	8240	4354	6153	1759	7917

Jelmagyarázat:
Össz.: Összesen
Első.: Első helyes jelentkezők száma
AN.: Alapképzés nappali munkarendben, mindkét finanszírozási formában
Állami.: Állami ösztöndíjjal támogatott képzések minden képzési formában és munkarendben

1.2. ábra. *2013-os jelentkezési adatok, forrás a http://www.felvi.hu*

Megállapíthatjuk, hogy az informatikai képzési terület nagyon hamar és igen dinamikusan állt át az új a bolognai folyamatnak megfelelő képzési szerkezetre. Az átálláskor figyelembe kellett venni a hagyományokat és lehetőségeket. Magyarországon nincs még meg a realitása, hogy a CC2005 ajánlásának megfelelően öt vagy annál több informatikai alapszak jöjjön létre. A közel jövő feladata, az MSc szakok mellett az informatika tanár szakok 2013-as indulása osztatlan formában. A tanár szak általános (4+1 félév) és középiskolai (5+1 félév) formában kerül bevezetésre. Az informatika tanár végzettséget adó diploma csak szakpárban szerezhető meg. Az első 6 szemeszter megegyezik a kétfajta képzésben az átjárhatóság kedvéért. Természetesen az informatika alap- és mesterszakok elvégzése után is van lehetőség tanárszakra jelentkezni, ami így a képzési idő rövidülésével jár.

Meg kell említenünk, hogy a BSc, MSc lineáris rendszer mellett létezik a felsőoktatási szakképzés is. Jelenleg a felsőoktatási szakképzésben megszerzett kreditek 75%-a beszámít a ráépülő alapszakba. Ez gyakorlatban azt jelentheti, hogy a féléves összefüggő gyakorlat kivételével, ami 30 kredit, a maradék 90 kredit teljes egészében beszámításra kerül. Ez a szabály komoly szakmai vitákat eredményezett. Véleményünk szerint a jelenlegi helyzet nem azt tükrözi, amit ennek a képzésnek a lehetőségei rejtenek.

1.4. Számológép, számítógép

Az emberiség már régóta igyekezett olyan segédeszközöket megteremteni, amelyek a számolást, általában a szellemi munkák mechanizálható részének végzését megkönnyítik.

A legrégibb ilyen segédeszköz a golyós számológép, vagy abacus, amely már az ókorban megjelent.

Az első automatikus működésű mechanikus összeadógépet 1642-ben egy francia matematikus Pascal építette meg, amelyet Leibniz 1671-ben továbbfejlesztett a négy alapművelet elvégzésére. Ezek a mechanikus gépek fogaskerekekkel működő gépek.

Az első vezérlésre használt eszköz ötlete Falcon (1728) nevéhez kapcsolódik, aki egy automatikusan működő szövőgéphez, az ismétlődő szövésminta kialakítását lyukkártya vezérlésével tervezte. Ötletét Jacquard valósította meg 1798-ban.

A lyukkártyás vezérlést egyéb területekre is alkalmazva 1884-ben Hermann Hollerith valósította meg és 1890-re kifejlesztett egy olyan lyukkártyarendszert, amelyet pl. a népszámlálásban fel is használtak. Az általa készített gépeken nemcsak a számok, hanem a gép működését szabályozó utasítások is a lyukkártyákon kerültek rögzítésre.

A számológépek mai modernebb megvalósítása az elektromos impulzusokkal manipuláló, elektromos asztali számológép 1944-től áll rendelkezésünkre. (Összeadógép, pénztárgép, zsebszámológép stb.) Ezeken a gépeken minden egyes művelet végzésénél meg kell adni adatot és a műveletre vonatkozó információt.

Az adat az úgynevezett regiszterbe kerül.

Definíció. A regiszter az adat átmeneti tárolóhelye, amelyben meghatározott nagyságú vagy számjegyet tartalmazó számot tudunk tárolni.

Pl. a nyolc helyiértékes regiszter sémája, amelyben minden pozícióra egy-egy számjel kerülhet.

A regiszterbe kerülő számoknál figyelembe kell venni, hogy annak véges pozíciója miatt bizonyos számoknak csak a közelítő értékeit adhatjuk meg meghatározott számjegyre pontosan.

A legtöbb összeadógép két regiszterrel rendelkezik, az egyikben az éppen beolvasott adatot, a másikban pedig a másik operandust, illetve a művelet eredményét tárolja. Az asztali számítógépek többsége négy regiszteres, amely közbülső adatok tárolására is lehetőséget ad.

A fentiek során ismertetett eszközök a számítógép ősei. Hiba lenne azonban csupán úgy értékelni, hogy a számítógép egy igen gyors asztali számológép. A gyors végrehajtás mellett a „lyukkártyaelv" megszületése is nagyon fontos szerepet játszott a számítógép létrejöttében, mivel a lyukkártyán előkészített adatokat már a feldolgozás előtt be lehet vinni a gépbe, és csökken a műveletek végrehajtásához szüksége idő. Ezen kívül lényeges szempont az is, hogy a kártyára rögzített adatok akárhányszor feldolgozhatók. Ehhez szükséges, hogy a számítógépnek sok regisztere, tárolóhelye legyen. De ha sok tárolóhelye van a számítógépnek, akkor azokba az adatokhoz hasonlóan a gépet működtető utasítások is tárolásra kerülhetnek, amelyek alapján a kijelölt feladat automatikusan, folyamatosan elvégzésre kerülhet (belső programvezérlés). Büszkén említhetjük, hogy e gondolatsor a magyar származású Neumann János nevéhez fűződik, aki a számítógépet az idegrendszer modelljének tekintette.

E modell megvalósítását, a számítógépet úgy is tekinthetjük, mint azokat az eszközöket, amelyeket az ember érzékszervei meghosszabbítására hozott létre (látcső, rádió, televízió stb.). Csakhogy itt az emberi agy bizonyos „képességeinek meghosszabbításáról" van szó, nevezetesen az emberi agy tárolókapacitásának és a „gondolkodás sebességének"a kiterjesztéséről. (Az emberi agy kb. 12-14 milliárd agysejtből áll, és maximum 100 m/sec sebességgel gondolkodik.) E két „képesség"szabja meg azt is, hogy milyen feladatok megoldására célszerű felhasználni a számítógépet.

A számítógép tárolóiban minden adat és utasítás számok formájában kerül tárolásra, de hogy ezek jelentése mit takar, hogy az adat szám vagy szöveg, illetve a művelet aritmetikai vagy logikai stb., az már a konkrét feladattól függ, ami univerzális információ-feldolgozó géppé tünteti ki. Alátámasztja ezt, hogy a számítógép űrhajót, atomerőművet, robotot, stb. vezérelhet, újságok, könyvek szövegeinek szerkesztését és nyomtatását végez-

heti, tervrajzok, képek megalkotásában és tárolásában segíthet, nyelvtani elemzéseket végezhet, nagy adatbázisokat, információs rendszereket működtet stb. Szerényen megemlíthetjük, hogy természetesen számolni, egyenletet megoldani is tud, és még akkor nem is említettük azokat a lehetőségeket, amelyek a hálózatba kapcsolással megvalósíthatók.

Ne feledjük azonban, hogy a számítógép nem képes az önálló gondolkodásra, csak az ember által megírt (utasítássorozat) program szolgai végrehajtására. Ha ezek a programok jók, akkor valóban csodákra képes a számítógép. A programokat szakemberek írják. A programok a gépben cserélhetők, így egy gépen végtelen számú probléma megoldható.

Definíció. A számítógépet alkotó fizikai eszközök összefoglaló neve hardver (hardware), a gépet működtető programrendszeré pedig a szoftver (software).

Mivel a hardver és szoftver elválasztható, ezért a számítógép minőségileg különbözik minden nem programozható géptől:

– a hardver állandó[3],

– a szoftver cserélhető,

és ezáltal a számítógép univerzális.

Azokat a számítógépeket, amelyek csak diszkrét értékeket tudnak feldolgozni, digitális számítógépeknek nevezzük (digit = számjegy). Azokat a számítógépeket pedig, amelyek a bemenő jeleknek, azokkal arányos analóg kimenőjeleket feleltetnek meg, analóg számítógépnek nevezzük. Mi a továbbiakban csak a digitális számítógéppel foglalkozunk.

Az első számítógép 1949-ben készült, de igazában 1951-től számíthatjuk a megjelenését, amikor is kereskedelmi forgalomba került. A számítógépek történetében azóta már több generációról is beszélhetünk, de kiemelkedő jelentőséggel bír az első chip vezérlésű számítógép 1969. évi megjelenése.

A lyukkártyás rendszer ma már teljesen elavult és adatokat közvetlenül billentyűzeten keresztül vagy egyéb automatizált módon viszik a gépbe

[3] Ma már lehetséges a számítógépek hardver elemeit is cserélni, akár működés közben is.

és lyukkártya helyett pl. mágneses vagy optikai adathordozókon (pl. DVD lemezen) tároljuk.

Definíció. Minden olyan gépet, amely a beolvasási és kiírási, adattárolási, aritmetikai és vezérlési műveletek elvégzése mellett programok tárolására és utasításonkénti végrehajtására is képes emberi beavatkozás nélkül, számítógépnek nevezzük.

1.5. Helyiértékes számrendszer

A számítástechnikában alapvető a számrendszerek ismerete. A következő fejezetben igyekszünk részletesen és alaposan kitárgyalni ezt a témát.

A mechanikus eszközökön és gépeken lehetséges volt sokfajta számrendszer kezelése. Az emberi természetből adódóan a legtöbb megvalósítás a tízes számrendszeren alapult. A számítógépek elterjedésével általánossá vált a kettes (és a kettő hatványain alapuló) számrendszer használata. A 10-es számrendszerben egy szám értékét a számot formálisan felépítő számjegy értéke és helye határozza meg, az ilyen rendszereket helyiértékes számrendszernek nevezzük. Vannak más elven felépülő számrendszerek is, például a római számrendszer nem helyiértékes. Itt például az I jel értéke a mögötte lévő jelektől függ. Tekintsük például az Eszterházy Károly Főiskola megnyításának a dátumát:

$$1774 = MDCCLXXIV$$

A decimális számrendszerben a

$$\{0,1,2,3,4,5,6,7,8,9\}$$

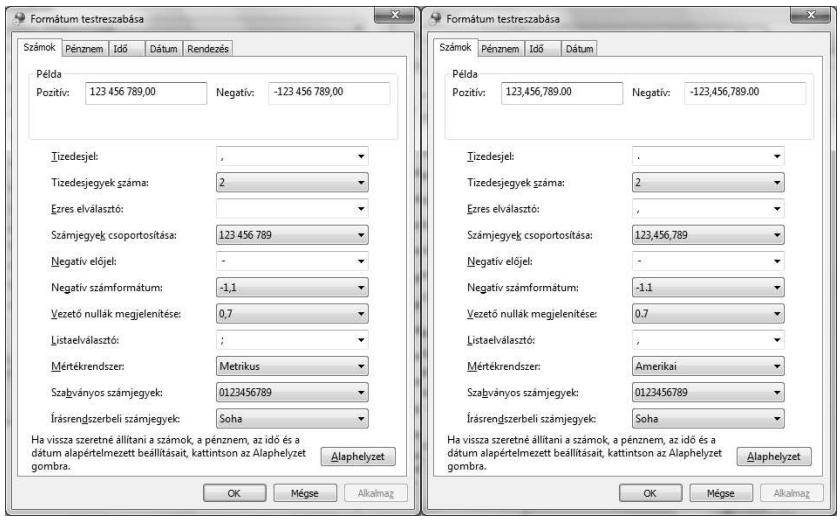
véges számjegyhalmaz elemeiből képzett végtelen sorozatok és a valós számok között kölcsönösen egyértelmű leképezést valósítunk meg figyelembe véve a helyiértékes jelölést. Például az 543,21 helyiértékes szám 5 db 100-ast, 4 db 10-est, 3 db 1-est, 2 db tizedet és 1 db századot jelöl, azaz

$$543{,}21 = 5 * 10^2 + 4 * 10^1 + 3 * 10^0 + 2 * 10^{-1} + 1 * 10^{-2}.$$

Ebben a jegyzetben végig a magyar területi beállításoknak megfelelően tizedes vesszővel választjuk el az egészeket a törtektől, nem pedig az angolszász terminológiának megfelelően tizedesponttal. Figyeljük meg a 1.3 ábrán, hogy az amerikai mértékrendszerben tizedes vessző az ezres elválasztó.

A fenti példát általános formában felírva az

$$(a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m})_{10}$$



1.3. ábra. A magyar és az amerikai formátum Windows 7-ben

alakban jelölt szám értéke

$$N = \sum_{i=-m}^{n} a_i 10^i, \qquad \text{ahol} \quad a_i \in \mathbb{Z} \quad \text{és} \quad 0 \leqslant a_i < 10.$$

Definíció. Ha $p > 1$ egy egész szám, akkor az

$$(a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m})_p$$

p alapú számrendszerbeli szám értéke a 10-es számrendszerben

$$\begin{aligned} N \quad = \quad & \sum_{i=-m}^{n} a_i p^i = a_n p^n + a_{n-1} p^{n-1} + \dots + a_1 p + a_0 + \\ & + a_{-1} p^{-1} + \dots + a_{-m} p^{-m}, \end{aligned}$$

ahol $a_i \in \mathbb{Z}$ *és* $0 \leqslant a_i < p$, vagyis $a_i \in \{0,1,2,\dots,p-1\}$.

Ha $p > 10$, akkor az

$$a_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,...\}$$

jelölést alkalmazzuk a számjegyek megadására.

A továbbiakban a tízes számrendszerbeli számoknál nem jelöljük a számrendszer alapszámát. Más alapú számrendszereknél többféle jelölés a szokásos:

$$(123{,}12)_4 \quad , \quad 123{,}12_{(4)} \quad .$$

Ebben a jegyzetben az első változatot fogjuk követni.

Példák.

$$\begin{aligned}
(123{,}12)_4 &= 1*4^2+2*4+3+1*4^{-1}+2*4^{-2}= \\
&= 16+8+3+0{,}25+0{,}125=27{,}375 \\
(123{,}12)_8 &= 1*8^2+2*8+3+1*8^{-1}+2*8^{-2}=83{,}15625 \\
(123{,}12)_{16} &= 1*16^2+2*16+3+1*16^{-1}+2*16^{-2}=291\frac{18}{256}=291\frac{9}{128}.
\end{aligned}$$

A fenti formula egy algoritmust szolgáltat nekünk arra, hogyan tudunk tetszőleges p alapú számrendszerből áttérni 10-es alapú számrendszerbe. Ha külön választjuk a szám egész és törtrészét, akkor az úgynevezett Horner-féle elrendezéssel is megoldhatjuk a feladatot. Pl.

$$(a_3a_2a_1a_0)_p$$

egész szám a következő alakban is felírható

$$((a_3p+a_2)p+a_1)p+a_0.$$

Tehát a szorzás és összeadás ismétlődő alkalmazásával a részeredmények tárolása nélkül is kiszámítható az eredmény. Például

$$(1203)_4=((1*4+2)*4+0)*4+3=99.$$

A törtrészekre vonatkozólag:

$$(a_{-1}a_{-2}a_{-3})_p=((a_{-3}:p+a_{-2}):p+a_{-1}):p,$$

tehát például

$$(0{,}12)_4 = (2 : 4 + 1) : 4 = 1{,}5 : 4 = 0{,}375.$$

A fenti példákat általános formában felírva megadhatjuk a definíciót is.

Definíció. Ha $p > 1$ egy egész szám, akkor egy tetszőleges

$$E = (a_n a_{n-1} \dots a_1 a_0)_p = a_n p^n + a_{n-1} p^{n-1} + \dots + a_1 p + a_0$$

egész szám felírható

$$E = (\dots((a_n p + a_{n-1})p + a_{n-2})p \dots + a_1)p + a_0$$

alakban. Hasonlóan a

$$T = (a_{-1} a_{-2} \dots a_{-m})_p = a_{-1} p^{-1} + a_{-2} p^{-2} + \dots + a_{-m} p^{-m}$$

törtszám is felírható

$$T = (\dots(a_{-m} : p + a_{m-2}) : p \dots + a_{-1}) : p$$

alakban. A fenti alakokat a polinomok Horner-féle elrendezésének mintájára a szám *Horner-féle* alakjának is nevezhetjük.

A kétfajta előállításhoz tartozik egy-egy algoritmus, amit az algoritmusokkal foglalkozó fejezetben tárgyalunk.

1.5.1. Véges pozíción ábrázolt számok

A hétköznapi életben műszereken, vagy gépek kijelzőin számtalan számláló típussal találkozhatunk. Pl. a gépjárművek kilométer számlálója általában 6 pozíción mutatja a megtett kilométereket:

9	9	9	9	9	9

Ha szerencsések vagyunk az óra elérheti a 999999 km-t. Ha még egy kilométert megyünk, akkor a kijelző 000000-ra vált mivel a keletkezett magasabb helyiérték a hetedik egyes a regiszterből kicsordul. Ugyanez történt Al Bundy Dodge autójával, amely a "Get outta Dodge" (0817) részben több mint egymillió kilométert futott, és ezért a kilométer számláló még egyszer csupa nullát mutatott. Az ócska Dodge-ért Al kapott volna egy új Vipert, de mint annyiszor, ezt is elhibázza.

Az informatikában gyakori probléma, hogy adott számú pozíción határozzuk meg egy számrendszerben a legnagyobb és a legkisebb ábrázolható számot. Legyen tehát h a pozíciók száma, p a számrendszer alapszáma. Az

$$(a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m})_p$$

szám esetén

$$h = n + 1 + m.$$

A legnagyobb számot kapjuk, ha minden pozíción $(p-1)$ áll, tehát

$$N_{\max} = \sum_{i=-m}^{n} (p-1)p^i = (p-1)(p^n + p^{n-1} + \dots + p^{-m}).$$

A geometriai (más néven mértani) sor összegképletét felhasználhatjuk[4]:

$$N_{\max} = (p-1)p^{-m}\frac{p^{m+n+1}-1}{p-1} = p^{n+1} - p^{-m}.$$

Ha a számnak nincs törtrésze ($m = 0$), akkor $h = n + 1$ és

$$N_{\max} = p^h - 1.$$

Például $h = 6$ esetén

10-es számrendszerben: $10^6 - 1 = 999999$,

2-es számrendszerben: $2^6 - 1 = 64 - 1 = 63 = (111111)_2$.

[4] Ha egy geometriai (mértani) sor első eleme $a_1 \neq 0$ és hányadosa (kvóciense) p, akkor sor első h elemének az összege: $s_h = a_1 + a_1 p + a_1 p^2 + \dots + a_1 p^{h-1} = a_1 \frac{p^h - 1}{p-1}$, ha $p \neq 1$. A mi esetünkben $a_1 = p^{-m}$ és $h = m + n + 1$.

Ha a számnak elmarad az egész része, azaz nulla az egész rész, akkor $n+1=0$ (azaz $h=m$) és

$$N_{\max}=1-p^{-h},$$

tehát $h=6$ esetén

10-es számrendszerben: $1-10^{-6}=0{,}999999$

2-es számrendszerben: $1-2^{-6}=(0.111111)_2$.

Nem negatív számok esetében minden számrendszerben a legkisebb szám a nulla. Viszont sokszor meg kell határoznunk azt a legkisebb törtszámot is, amely még ábrázolható h pozíción. Ebben az esetben az utolsó pozíción van 1-es jegy a többi helyen 0,

$$N_{\min}=0p^n+\cdots+0p^0+0p^{-1}+\cdots+1p^{-m}=p^{-m}.$$

Például 4 egész és 2 törtszámjegy esetén leírható legnagyobb és legkisebb szám

2-es számrendszerben: $(1111{,}11)_2$, illetve $(0000{,}01)_2$

10-es számrendszerben: 9999,99, illetve 0000,01

16-os számrendszerben: $(FFFF,FF)$, illetve $(0000{,}01)_{16}$.

(A törtet elválasztó vessző jel nem számít bele a pozíciók számába.)

A számrendszer alapszámától függően ugyanolyan hosszúságú regiszterben több vagy kevesebb különböző számot tudunk ábrázolni. Ha az alapszám nagyobb és így a jelölő számjegyek száma is, akkor több számot tudunk ábrázolni egy konstans h hosszúságú (pozícióval rendelkező) regiszterben.

Felmerül a kérdés úgy is, hogy milyen alapú számrendszerben ábrázolhatjuk a legkevesebb jelölővel a legfeljebb N különböző számot magába foglaló számhalmazt.

Tétel. *Az a p érték, amely alapú számrendszerben a legkevesebb jelölővel ábrázolhatjuk a legfeljebb N különböző számból álló halmazt, a természetes alapú logaritmus alapja.*

Bizonyítás. Tegyük fel tehát, hogy N =konstans, és hogy a p alapú

számrendszerben ehhez h pozíciót kell felhasználni. A szükséges jelölők száma összesen

$$hp \ ,$$

mivel mindegyik pozícióra p különböző jel kerülhet. Másrészt az N konstans értéke is meghatározható, mivel p elem h tagú ismétléses variációjával egyenlő, azaz $N = p^h$, amelyből

$$h = \frac{\ln N}{\ln p}$$

(ln az $e \approx 2{,}71$ ... (természetes) alapú logaritmus).

Ezt behelyettesítve az előző egyenletbe, a feladat visszavezethető az

$$f : (1, +\infty) \longrightarrow \mathbb{R}, \qquad f(p) = \frac{\ln N}{\ln p} p = \ln N \frac{p}{\ln p}$$

függvény minimumának meghatározására.

Az f első differenciálhányadosa:

$$\mathrm{diff}(f, p) = f'(p) : (1, +\infty) \longrightarrow \mathbb{R}, \qquad f'(p) = \ln N \frac{\ln p - 1}{\ln^2 p}.$$

A szélsőérték meghatározásához az f' zérushelyét megkeresve az

$$\ln p - 1 = 0$$

egyenlethez jutunk, ahonnan

$$\ln p = 1, \qquad \text{azaz} \qquad p = e \approx 2{,}71...$$

Mivel $f'(p) < 0$, ha $1 < p < e$ és $f'(x) > 0$, ha $p > e$, így f szigorúan monoton csökkenő a $(1, e)$ és szigorúan monoton növekvő az $(e, +\infty)$ intervallumon, tehát f-nek az $p = e$ helyen abszolút minimuma van.

Másrészt $2 < e < 3$, továbbá

$$\frac{2}{\ln 2} \approx 2{,}885 \qquad \text{és} \qquad \frac{3}{\ln 3} \approx 2{,}731,$$

így $\ln N > 0$ miatt $f(3) < f(2)$, de a fizikai megvalósítás miatt, amire már az előzőek során utaltunk érthető, hogy a számítástechnikában a $p = 2$ alapszámú {0,1} jelölőhalmazú bináris számrendszert használják. Kísérletek történtek $p = 3$ alapú számrendszerben működő számítógépre is, de ezek nem váltak be.

Tekintsük pl. az $N = 2^{10} = 1024$ különböző számot. Ezek a számok a 2-es számrendszerben 10 pozíción ábrázolhatók $(0 - 1023)$,

$$1023 = (1111111111)_2 \ ,$$

így ezek ábrázolásához $2 * 10 = 20$ jelölő szükséges. 10-es számrendszerben 1000 szám ábrázolásához 0-tól 999-ig 3 pozíció szükséges, de már ezek megvalósításához is $3 * 10 = 30$ jelölő szükséges. Az 5-ös számrendszerben

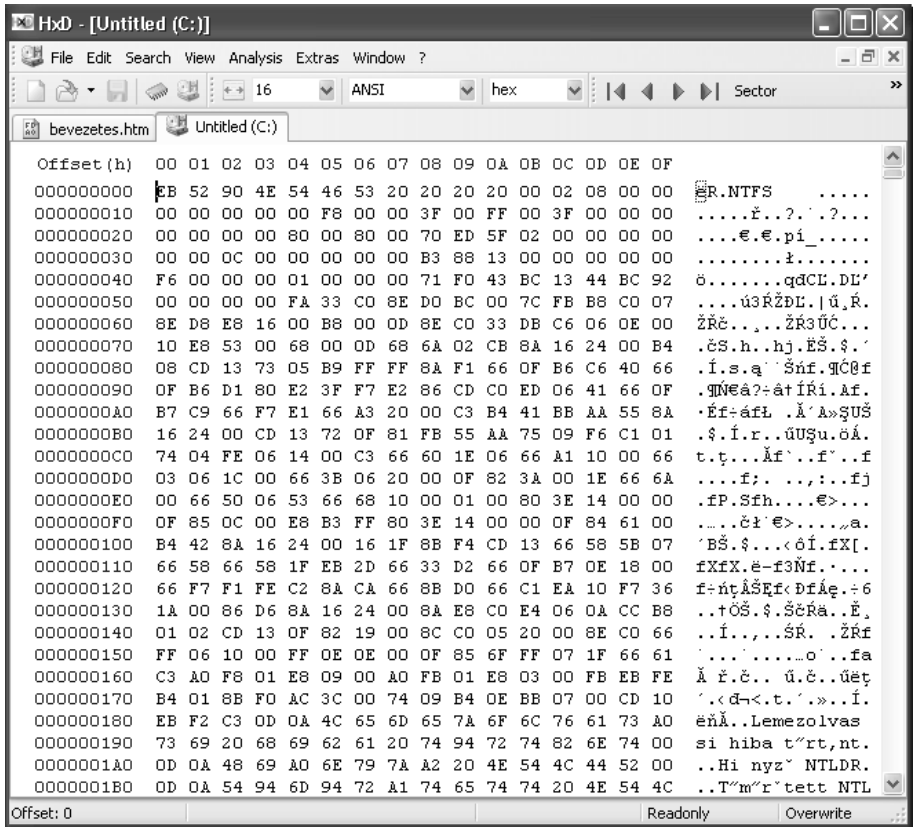
$$1023 = (13043)_5$$

már csak 5 pozíción ábrázolható, de már 4 pozíción $5 * 4 = 20$ jelölő szükséges.

1.5.2. Bináris, oktális és hexadecimális számrendszer

Könnyen belátható, hogy a bináris számrendszerben egy szám ábrázolásához viszonylag sok pozícióra van szükség. A számítógépi adatábrázolásnál – pl. a memória vagy lemez fizikai tartalmának a megjelenítésekor – gyakran a tömörebb, kevesebb pozíciót használó 8-as vagy 16-os számrendszert használják. Ezeket oktális és hexadecimális számrendszereknek nevezzük. A 1.4 ábrán láthatjuk egy hexaeditor futási képét.

Hosszadalmas átváltási algoritmus alkalmazása helyett a konvertálás egyszerű csoportképzési módszerrel hajtjuk végre. Kihasználjuk, hogy mind a 8, mind a 16 a 2-nek egész hatványa, azaz $2^3 = 8$ és $2^4 = 16$. Ennek megfelelően, ha egy bináris (2-es számrendszerbeli) számot át akarunk váltani, akkor a "kettedes vesszőtől" (törtrészt az egészrésztől elválasztó jeltől) balra és jobbra hármas (triáda) ill. négyes (tetrád) csoportokat képezhetünk, ami megfelel egy 8-as ill. 16-os számrendszerbeli számjegynek. Az algoritmus



1.4. ábra. HxD hexaeditor egy html file bináris tartalmát mutatja

visszafelé is működik.

Oktális számjegy	Bináris triáda
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Hexadec. számjegy	Bináris tetrád
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Hexadec. számjegy	Bináris tetrád
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Például

$$(123{,}54)_8 = \underbrace{\boxed{001}\boxed{010}\boxed{011}}_{1\quad 2\quad 3} \;,\; \underbrace{\boxed{101}\boxed{100}}_{5\quad 4}$$

Ellenőrizzük le az eredményt azzal, hogy mindkét számot átváltjuk 10-es számrendszerbe:

$$\begin{aligned}(123{,}54)_8 &= 1*64+2*8+3*1+\frac{5}{8}+\frac{4}{64}=83\frac{44}{64}=83{,}6875\\ (1010011{,}1011)_2 &= 1*64+1*16+1*2+1+\frac{1}{2}+\frac{1}{8}+\frac{1}{16}=83\frac{11}{16}=83{,}6875\ .\end{aligned}$$

Alakítsuk vissza a 2-es számrendszerbeli számot a tetrádok segítségével 16-os számrendszerbe:

$$(1010011{,}1011)_2 = \underbrace{\boxed{0101}\boxed{0011}}_{5\quad 3} \;,\; \underbrace{\boxed{1011}}_{\mathrm{B}}$$

ami 10-es számrendszerben

$$(53,\mathrm{B})_{16} = 5*16+3*1+\frac{11}{16}=83{,}6875\ .$$

A visszaalakítás során plusz nullákra is lehet szükségünk a triádák vagy a tetrádok kialakításához. Ennek elfelejtése gyakori hibát okoz. Például az $(11{,}101)_2 \neq (3{,}5)_{16}$, hanem a helyes megoldás:

$$(11{,}101)_2 = \underbrace{\boxed{0011}}_{3} \;,\; \underbrace{\boxed{1010}}_{\mathrm{A}}$$

Definíció. A bináris számrendszerbeli kód egyes pozícióit, amelyre a 0 és 1 bináris számjegyek valamelyike írható az angol **bi**nary digi**t** elnevezésből rövidítve **bit**nek nevezzük.

1.5.3. Tízes számrendszerbeli számok konvertálása

A számrendszer definíciója egyúttal eljárást is ad tetszőleges ($p > 1$) alapú számrendszerből 10-es számrendszerbe való konvertálásra. A következő-

ekben fordítva, 10-es számrendszerbeli számok tetszőleges számrendszerbe való konvertálásához keresünk eljárást. Ezzel megoldjuk a tetszőleges számrendszerből tetszőleges számrendszerbe konvertálás problémáját is.

Tekintsünk egy tetszőleges N tízes számrendszerbeli számot. Az N szám egész része minden helyiértékes és egész alapú számrendszerben egész, a törtrész pedig törtszám. Ezért az egész rész és a törtrész konverzióját külön végezzük el. Legyen tehát

$$N = N_0 + T_0$$

ahol N_0 a szám egész részét, T_0 pedig a törtrészét jelöli.

Egész rész konvertálása. Általában ez az algoritmus jól ismert, hiszen már a középiskolában tanítják a diákoknak. A következőekben bebizonyítjuk az algoritmus helyességét. Az N_0 egész számot felírhatjuk p alapú számrendszerben (figyeljük meg, hogy a szummás kifejezésben most i 0-tól n-ig megy):

$$\begin{aligned} N_0 &= \sum_{i=0}^{n} a_i p^i = a_n p^n + a_{n-1} p^{n-1} + \cdots + a_1 p + a_0 = \\ &= p(a_n p^{n-1} + a_{n-1} p^{n-2} + \cdots + a_1) + a_0 = pN_1 + a_0 \ . \end{aligned}$$

Feladatunk a $a_0, a_1, \ldots, a_n$ ismeretlen számjegyek meghatározása. A felírt formula alapján a legutolsó jegyet a a_0-t meghatározhatjuk, mert ha az N_0-t elosztjuk p-vel, akkor kapunk egy N_1 egész számot, és a maradék egyenlő lesz a_0-val. Nyilván igaz, hogy $a_0 < p$.

Folytatva az eljárást, azaz p-t ismét kiemelve:

$$N_1 = p(a_n p^{n-2} + a_{n-1} p^{n-3} + \cdots + a_2) + a_1,$$

vagyis N_1osztva p-vel megkapjuk N_2-t, és marad a_1. Tehát a_1 lesz a következő számjegy a p alapú számrendszerben a következő helyiértéken. Az algoritmust a nulla hányadosig kell folytatnunk, az így adódó maradék, az

a_n lesz a legmagasabb helyiértékű jegy:

$$N_n = p * 0 + a_n \ .$$

Példák.

1. Váltsuk át a 123-at 8-as számrendszerbe!

$$\begin{aligned} 123 : 8 = 15 \quad &\text{és marad} \quad 3, \\ 15 : 8 = 1 \quad &\text{és marad} \quad 7, \\ 1 : 8 = 0 \quad &\text{és marad} \quad 1. \end{aligned}$$

A fenti műveletsorhoz jól ismert az alábbi formula, ahol a hányados balra, a maradék jobbra kerül, és a számjegyeket alúlról felfelé olvassuk ki.

123	÷ 8
15	3
1	7
0	1

$123 = (173)_8$

2. Váltsuk át 16-os számrendszerbe a 123-at!

123	÷ 16
7	B
0	7

$123 = (7B)_{16}$

Fontos megjegyeznünk, hogy az algoritmus *véges*.

Törtrész konvertálása. Vizsgáljuk meg a szám T_0 törtrészét:

$$T_0 = \sum_{i=-1}^{-m} a_i p^i = a_{-1}p^{-1} + a_{-2}p^{-2} + \cdots + a_{-m}p^{-m} \ ,$$

szorozzuk meg T_0-t p-vel

$$T_0 p = a_{-1} + a_{-2}p^{-1} + \cdots + a_{-m}p^{-m+1} = a_{-1} + T_1,$$

ahonnan a_{-1} a szorzat egész része, az első törtjegy, a törtrész pedig tovább szorozva p-vel

$$T_1 p = a_{-2} + a_{-3}p^{-1} + \cdots + a_{-m}p^{-m+2} = a_{-2} + T_2 \ .$$

A szorzat egész része a_{-2} adja a második törtjegyet, és az eljárást megismételve tovább folytathatjuk. Ellentétben az előző algoritmussal az eljárás nem biztos, hogy *véges*, azaz nem mindig ér véget oly módon, hogy a törtrész nullává válik.

Tétel. *A véges formában felírt tízes számrendszerbeli szám nem mindig írható fel véges formában egy másik számrendszerben.*

Példák.

1. Konvertáljuk a 0,6875-öt 8-as számrendszerbe!

$$\begin{aligned} 0{,}6875 * 8 &= 5{,}5000 = 5 + 0{,}5 \\ 0{,}5 * 8 &= 4{,}0000 = 4 \end{aligned}$$

Leírhatjuk egyszerűbb formában is az eljárást:

	0 , 6875 $*$ 8
5	5000 $*$ 8
4	0000

$0{,}6875 = (0{,}54)_8$

A kiolvasás felülről lefelé és a baloldalon az egészeknél történik.

2. Írjuk át a 0,6875-öt a fenti eljárással 2-es számrendszerbe!

	0 , 6875 $*$ 2
1	3750 $*$ 2
0	7500 $*$ 2
1	5000 $*$ 2
1	0000

$0{,}6875 = (0{,}1011)_2$

3. Váltsuk át 16-os számrendszerbe a 0,2175 számot!

0 ,	$2175 * 16$
3	$4800 * 16$
7	$6800 * 16$
A	$8800 * 16$
E	$0800 * 16$
1	$2800 * 16$
4	4800
$\vdots$	$\vdots$

$$0{,}2175 = (0{,}37AE147AE14\ldots)_{16}$$

1.5.4. Feladatok a számrendszerek konvertálására

1. Alakítsuk át decimális számrendszerbe az alábbi számokat:

   a) $(101101)_2$ b) $(10101000{,}11)_2$
   c) $(0{,}001101)_2$ d) $(1111111{,}11111)_2$
   e) $(64{,}75)_8$ f) $(177{,}77)_8$
   g) $(12A3D)_{16}$ h) $(45{,}55)_{16}$
   i) $(1FA{,}1A)_{16}$ j) $(FFFE)_{16}$
   k) $(11111111)_2$ l) $(7FFF)_{16}$

2. Alakítsuk át a következő számokat binárissá:

   a) 275 b) 25,373
   c) 0,1 d) 0,3333
   e) 0,00304 f) 999,9

3. Konvertáljuk 16-os számrendszerbe az alábbi számokat:

   a) 32769 b) -11200
   c) 3215675 d) 77,875
   e) 1,39 f) 1234,0985

4. Mekkora a hexadecimális számrendszerben hat egész jegyen felírható legnagyobb szám? Az így kapott számot írjuk át decimális számrendszerbe!

5. A 99999 szám ábrázolásához hány helyiérték kell a 16-os, a 8-as, a 4-es, és a kettes számrendszerben?

6. Mekkora a 4, a 8, a 15, a 16, a 24 és a 32 biten ábrázolható legnagyobb szám a 2-es számrendszerben? Fogalmazza meg válaszát nagyságrendben is, pl.: 16 bit esetén több tízezres a nagyságrend.

7. Készítsen táblázatkezelőben munkalapot, amely a 2, a 8 és a 16 hatványait tartalmazza.

8. Végezzük el az alábbi műveleteket a bináris számokkal, majd ellenőrizzük decimálisra való konvertálással.

a)
$$\begin{array}{r} 11110{,}01 \\ +\ 1011{,}10 \\ \hline \end{array}$$

b)
$$\begin{array}{r} 111100101{,}01 \\ +\ 111111101{,}11 \\ \hline \end{array}$$

c)
$$\begin{array}{r} 11110{,}01 \\ -\ 10001{,}10 \\ \hline \end{array}$$

d)
$$\begin{array}{r} 100111{,}1000 \\ -\ 10111{,}1111 \\ \hline \end{array}$$

9. Végezzük el az alábbi műveleteket a hexadecimális számokkal.

a)
$$\begin{array}{r} \mathrm{ABBA} \\ +\ \mathrm{EDDA} \\ \hline \end{array}$$

b)
$$\begin{array}{r} \mathrm{ABC{,}DE} \\ +\ 123{,}\mathrm{AA} \\ \hline \end{array}$$

c)
$$\begin{array}{r} \mathrm{BB{,}BB} \\ +\ \mathrm{CCC{,}CC} \\ \hline \end{array}$$

d)
$$\begin{array}{r} \mathrm{AAA{,}AA} \\ -\ \mathrm{A{,}AB} \\ \hline \end{array}$$

e)
$$\begin{array}{r} 2\mathrm{DB{,}28} \\ +\ 17\mathrm{D{,}60} \\ \hline \end{array}$$

f)
$$\begin{array}{r} 2\mathrm{DB{,}28} \\ -\ 17\mathrm{D{,}60} \\ \hline \end{array}$$

g)
$$\begin{array}{r} 1000{,}10 \\ -\ 111{,}11 \\ \hline \end{array}$$

10. A 10-es számrendszerben a $0{,}1 * 10 = 1$. Igaz ez egy véges helyiértékes kettes számrendszerben is? Válaszát indokolja!

2. AZ INFORMÁCIÓ

2.1. Az információ fogalma

Az ember először az *anyaggal* barátkozott meg és csak a 18–19. században ismerte meg az energiát, majd a 20. században felfedezte az *információt.* A szervezettség mai szintjére kellett emelkedni ahhoz, hogy felismerje: az információ éppen olyan főszerepet játszik a világban, mint az anyag és az energia. A levegő, víz, táplálék és hajlék mellett az *információ is az ember alapvető szükségletei közé tartozik.* Életünk, sőt létünk fenntartása is attól függ, hogy a megfelelő információk időben eljussanak hozzánk, illetve érzékeljük azokat. Lássuk az előttünk lévő szakadékot, akadályt, halljuk a közeledő autó zúgását, érzékeljük a hőmérsékletet, megértsük a számunkra jelentőséggel bíró szóbeli vagy írásbeli közléseket stb. Az agy is csak úgy őrizheti meg normális egészségi állapotát, ha állandóan új információkkal táplálkozik, amivel ismeretünk, tudásunk gyarapszik.

Az ismeretnek ahhoz, hogy közölhető legyen, azaz *információvá váljék, anyagi hordozóra,* és ahhoz, hogy eljusson a címzetthez, energiára van *szükség.* Az információt az különbözteti meg az anyagtól és az energiától, hogy nem érvényesek rá a megmaradási tételek. Az információ megsemmisíthető és létrehozható. A fontosabb információk megóvására külön védelmi előírások vannak.

Az előzőek során már próbáltuk körülírni az információ fogalmát. Köznapi értelemben az információk értesülések, ismeretek, hírek ismeretlen vagy kevéssé ismert dolgokról vagy eseményekről. Egzakt megfogalmazása éppoly nehéz mint az anyag definíciója. Az értesülés, ismeret, hír ugyanis csak más szóval helyettesíti az információt. Az értesülés, ismeret vagy hír azonban önmagában még nem információ. Mert egyrészt, ha valaki azt már ismeri, annak semmi információt nem jelent, másrészt viszont, ha valaki nem érti meg, és nem tudja felfogni, annak sem információ.

Az **információnak** tehát fontos velejárója, hogy valami újat közölhet, vagy másképpen fogalmazva *bizonytalanságot szüntethet* meg, amely *döntésre, válaszra, magatartásunk megváltoztatására késztethet.*

Barátunk, akivel megállunk beszélgetni, az újság amit elolvasunk, a rádió hangszórója, a televízió képernyője, a hangversenyen hallgatott zeneszám, a közlekedési jelzőtábla, a virág amit megszagolunk, az étel amit megkóstolunk stb. mind információt közöl velünk.

Az információ közlésnek fontos részelemei, hogy birtokosa közlésre alkalmas formába öltöztesse, *kódolja,* amit el kell juttatni a befogadóhoz, aki – ha valóban befogadta, magáévá tette, *dekódolta* – cselekvéssel, magatartásának megváltoztatásával, vagy új információkkal válaszolhat.

Definíció. Az **információk** tartalmukban és formájukban különböznek, de lényegük ugyanaz, új ismereteket hordozó jelek tartalmi jelentésének befogadása, amely valamilyen tevékenységre (válasz, döntés stb.) késztethet.

Az információ és az ember kapcsolatában, az emberiség eddigi történetében hat jelentős állomás volt.

- A *beszéd,* ami a gondolatok, információk közlésének alapvető formája.
- Az *írás,* amellyel az információtárolás függetlenné vált az emlékezettől.
- A *könyvnyomtatás,* amely az információ terjesztésénél játszott fő szerepet.
- A *távközlés,* az információs összeköttetéseknek nyitott lehetőséget.
- Az *elektronikus információfeldolgozás,* amely megteremtette az ember – gép párbeszéd lehetőségét.
- Az *internet megjelenése, amely lehetővé tette az információ szabad áramlását, és robbanásszerű növekedését.*

A társadalom fejlődését az utóbbi évszázadokban az információtermelés exponenciális növekedése és az információk áramlásának a gyorsulása jellemezte. Az embernek mind bonyolultabb szituációkban, mind nagyobb mennyiségű információk alapján, mind gyorsabban kell döntéseket hoznia. A legszélsőségesebb példa erre az űrrakéták irányítása, ahol a pálya paramétereinek a figyelembevételével, a szó szoros értelmében „azonnal" kell dönteni.

Az információrobbanás következtében *az információ is a munka tárgya lett* mint az anyag és az energia.

Az anyaggal és energiával kapcsolatban négy fő műveletet végzünk: *tárolás, szállítás, alakítás, feldolgozás,* amelyekhez megfelelő technikai eszközök, gépek is rendelkezésre állnak. Mivel az információ az anyaggal és az energiával szoros kapcsolatba hozható, ezért célszerűnek tűnik, hogy az információval is mint a munka tárgyával a fenti négy fő műveletet vizsgáljuk, amelyek meghatározzák a műveletekhez tartozó technikai eszközöket is:

- gyűjtés: mérőeszközök, érzékelők stb.,
- tárolás: film-, hangfelvevő, DVD, Bluray Disc, merevlemez, szerver farmok, felhők, Cloud computing stb.,
- szállítás: híradástechnikai, hálózati eszközök. Vezetékes és vezetéknélküli adatátvitel,
- alakítás: számítástechnikai eszközök, digitalizálók.

A számítástechnikai eszközöknél noha az átalakítás a fő művelet – fejlettségüktől függően – megtaláljuk a többi műveletet (gyűjtés, tárolás, szállítás) és azok speciális eszközeit is. Például mérőeszközről gyűjt információt valamely folyamatban, vagy hírközlési eszközök felhasználásával nagyobb távolságról, és az eredmények tárolására sokféle eszközzel rendelkezik.

2.2. Az információ útja (átvitele)

Minden információ-közlésnél legalább három alkotórészt ismerünk fel:

1. Adó vagy forrás
2. Vevő vagy nyelő
3. Továbbító közeg, vagy csatorna, amely a közleményt az adótól a vevőhöz eljuttatja, illetve összeköti azokat.

A csatornán bizonyos – a csatorna fizikai tulajdonságaitól függő – meghatározott típusú jelek továbbíthatók. Ezért a továbbítandó közleményt a csatornán átvihető jelek segítségével kell kifejezni, vagyis az információt a továbbításhoz megfelelően át kell alakítani *(kódolás),* majd a csatorna után ismét át kell alakítani a vevő számára (dekódolás). Az információ-közlési rendszer általános sémája tehát a következő:

$$\begin{array}{c} Adó -> \quad kódoló -> csatorna -> dekódoló -> vevő \\ zaj \end{array}$$

Az *adó* az az objektum, ami az információt, vagyis a továbbítandó közleményt szolgáltatja, pontosabban

$$A_1, A_2, ..., A_n$$

jeleket ad le. Például az ábécé betűi, morze jelek: pont, vonás, szünetjel stb.

A *kódoló* ezt a közleményt a csatornán való továbbításhoz megfelelően átalakítja, vagyis a csatornán átvihető jelek segítségével fejezi ki. A forrás által adott jeleket nevezhetjük egy *közlési nyelv* ábécéjének is, amelyekből *szó* vagy *közlemény* állítható elő.

Kódolás: olyan eljárás, amely egy nyelv véges ábécéjéből képezett szavakat kölcsönösen egyértelműen hozzárendel egy másik nyelv meghatározott szavaihoz.

A *csatorna* a kódközleményt továbbítja a vevő felé. A csatornában lehetnek nem kívánt források is, mint pl. rádiónál és televíziónál a zörej, telefonnál az áthallás stb. Az ilyen forrásokat *zajforrásoknak* vagy egyszerűen zajnak nevezzük.

A kódolt közleménynek olyannak kell lenni, amely kevésbé érzékeny a zajra. A számítástechnikában követelmény a zajmentes továbbítás.

A *dekódoló* a csatorna kimeneti oldalán vett közleményt megfejti, vagyis az információt a vevő számára eredeti formájába visszaalakítja.

Dekódolás: a kódolás megfordítása.

2.3. Az információ mérése

Az információ átviteli berendezések előállításának, megvalósításának csak akkor van értelme, ha tudjuk mérni az információt és átvitelét valamilyen formában. Ezért szükséges, hogy az információt matematikailag is kezelhetővé tegyük. Az információ tárolás, átalakítás és továbbítás matematikai problémáit, a valószínűségszámítás egy új ága az információelmélet vizsgálja. Az információelmélet elvi alapjait C. Shannon rakta le az 1948–49-es években.

Az információ méréséhez definiálni kell a mértékegységet. A mérték fogalmának kialakításánál figyelembe kell venni, hogy az független az információ

– tartalmától és

– alakjától.

Itt is úgy kell eljárni, ahogy azt a táviratfeladásnál a postai alkalmazottak teszik. A költség meghatározásánál, csak a szavakat veszik figyelembe, a tartalmat nem nézik.

A mérték általános definíciója előtt, vizsgáljunk egy egyszerűbb információforrást, amely

$$\{0,1,2,3,4,5,6,7\}$$

jeleket bocsát ki egyenlő valószínűséggel. Határozzuk meg az egy jelre jutó információ mennyiségét. A kérdés úgy is megfogalmazható, hogy mennyi információt jelent a 8 jelből egy konkrét jel (számjegy) kiválasztása. Átfogalmazva a problémát, megkérünk valakit, hogy válasszon ki a 8 számjegyből egy számot és feleljen a kérdésünkre igennel vagy nemmel, amivel kérdéseink után információhoz jutunk, bizonytalanságot szüntetünk meg. Hány kérdéssel tudjuk a kiválasztott számot meghatározni?

Eljárás:

1. kérdés: Nagyobb mint 3?

   Ezzel a felére csökkentjük a bizonytalanságot, mert vagy az első felében van, vagy a másodikban.

2. kérdés: Ha az első felében van, akkor nagyobb mint 1?

   Ha a második felében van, akkor nagyobb mint 5? Ezzel ismét felére csökkentjük a bizonytalanságot.

3. kérdés: Attól függően, hogy melyik két számjegy marad meg, közvetlenül rákérdezünk a számjegyre:

   $$1 \text{ vagy } 3 \text{ vagy } 5 \text{ vagy } 7,$$

   amelyre ha a válasz igen, akkor megtaláltuk a keresett számot, ha nem, a válasz akkor is megtalálható.

A kérdésekre adott igen vagy nem válaszokat 1 illetve 0-val felírva egyébként a keresett számjegy 2-es számrendszerbeli alakját kapjuk, ami ugyancsak szolgáltatja a keresett számot.

Tehát a számjegy kiválasztásához 3 kérdés, vagy 3 db 2-es számrendszerbeli jegy szükséges, így azt is mondhatjuk, hogy az egy jelre jutó információ 3 egységnyi.

Definíció. Ha a forrás (adó)

$$A_1, A_2, \ldots, A_n$$

jeleket bocsátja ki és

$$n = 2^m \quad (n > 0),$$

továbbá a jelek kibocsátásának valószínűsége egyenlő, vagyis

$$p_i = \frac{1}{n} \quad (i = l,2,\ldots,n),$$

akkor az előző eljárást alkalmazva az n elemű jelhalmaz egy konkrét elemének kiválasztásához m kérdésre van szükség, tehát az *egy jelre jutó információ* m.

Ezen gondolatok alapján javasolták az információ mértékéül az n 2-es

alapú logaritmusát, ugyanis

$$\log_2 n = \log_2 2^m = m \ .$$

(A továbbiakban a kettes alapú logaritmust röviden log jelöli).

A mértékegysége így

$$\log n = 1 \quad \text{miatt} \quad n = 2, \quad p = \frac{1}{2}$$

és a neve *1 bit.*

Definíció. Az információ mértékegysége **1 bit** (binary unit)[5]: két egyenlően valószínű jel egyikének kiválasztásához tartozó információmennyiség.

$$\log 2 = -\log \frac{1}{2} = -\log p = 1 \text{ bit} \ .$$

Ha a tanár vagy diáktárs a felelőnek bólint, megerősíti annak válaszát a feleletnél, akkor 1 bit információt közöl.

PÉLDÁK.

1. Hány bit információt képvisel egy 32 lapos magyar kártya egy lapja?

$$\log 32 = \log 2^5 = 5 \text{ bit.}$$

Hogyan fogalmazhatók meg a kérdések?

2. Hány bit információt jelent a sakktáblán egy figura helye, amely minden mezőre léphet?

$$\log 64 = \log 2^6 = 6 \text{ bit.}$$

[5] Ne keverjük össze a kettes számrendszerbeli számok egy helyiértékével (binary digit-jével), amelyet szintén bit-nek nevezünk. Megjegyezzük, hogy egy kettes számrendszerbeli számjegy (digit) által hordozott átlagos információ mennyisége maximum 1 bit lehet.

3. Hány bit információt képvisel egy decimális számjegy?

$$\log 10 \approx 3{,}32 \text{ bit.}$$

(Tehát 3 kérdéssel nem mindig határozható meg.)

4. Az élő nyelvben nem minden jel hordoz információt, például a

jelentékte...

szótöredék után, már mindenki tudja, hogy a „len" betűkombináció következik. Az élő nyelvek bizonyos *redundanciával* rendelkeznek, amelyek jó szolgálatot tesznek a hétköznapi kommunikációnál, ahol a „csatornában" levő zajok miatt a szófoszlányokból is értelmesen tudunk dekódolni.

A mi értelmezésünk szerint ilyen nem lehetséges és minden jelkombinációnak értelmes szónak kell lenni. Ez azt jelenti, hogy pl. a 24 betűs ábécében már 3 jeles szavakból összeállítható lenne egy közelítőleg 14 ezres szókincsű nyelv kialakítása (24^3 = 13824). Ez a redundancia-mentes nyelv azonban nehezen lenne beszélhető.

2.4. Bináris előtagok (prefixumok) használata

Ebben a fejezetben röviden áttekintjük a bináris prefixumok használatát és változását. A táblázatok a Wikipédián található táblázatok alapján készültek el.

Először egy áttekintő táblázatot készítettünk a metrikus prefixumok használatáról. A táblázatban megadjuk a magyar elnevezéseket is. Figyeljük meg a billió és a milliárd eltérő használatát az angol és a magyar nyelvben.

Metrikus prefixumok						
Prefixum	Jel	1000^m	10^n	Decimális érték	angol név	magyar név
yotta	**Y**	1000^8	10^{24}	1 000 000 000 000 000 000 000 000	septillion	kvadrillió
zetta	**Z**	1000^7	10^{21}	1 000 000 000 000 000 000 000	sextillion	trilliárd
exa	**E**	1000^6	10^{18}	1 000 000 000 000 000 000	quintillion	trillió
peta	**P**	1000^5	10^{15}	1 000 000 000 000 000	quadrillion	billiárd
tera	**T**	1000^4	10^{12}	1 000 000 000 000	trillion	billió
giga	**G**	1000^3	10^9	1 000 000 000	billion	milliárd
mega	**M**	1000^2	10^6	1 000 000	million	millió
kilo	**k**	1000^1	10^3	1 000	thousand	ezer
hecto	**h**	$1000^{2/3}$	10^2	100	hundred	száz
deca	**da**	$1000^{1/3}$	10^1	10	ten	tíz
		1000^0	10^0	**1**	**one**	**egy**
deci	**d**	$1000^{-1/3}$	10^{-1}	0.1	tenth	tized
centi	**c**	$1000^{-2/3}$	10^{-2}	0.01	hundredth	század
milli	**m**	1000^{-1}	10^{-3}	0.001	thousandth	ezred
micro	μ	1000^{-2}	10^{-6}	0.000 001	millionth	milliomod
nano	**n**	1000^{-3}	10^{-9}	0.000 000 001	billionth	milliárdod
pico	**p**	1000^{-4}	10^{-12}	0.000 000 000 001	trillionth	billiomod
femto	**f**	1000^{-5}	10^{-15}	0.000 000 000 000 001	quadrillionth	billiárdod
atto	**a**	1000^{-6}	10^{-18}	0.000 000 000 000 000 001	quintillionth	trilliomod
zepto	**z**	1000^{-7}	10^{-21}	0.000 000 000 000 000 000 001	sextillionth	trilliárdod
yocto	**y**	1000^{-8}	10^{-24}	0.000 000 000 000 000 000 000 001	septillionth	kvadrilliomod

A memória gyártók 1024-nek értelmezik a kilo- előtagot, míg a merevlemezgyártók 1000-nek. A következő táblázat az eltérő értelmezésből adódó hibákat mutatja:

Prefixum	Bin÷Dec	Dec÷Bin	Százalékos különbség
kilo	1.024	0.9766	+2.4% vagy -2.3%
mega	1.049	0.9537	+4.9% vagy -4.6%
giga	1.074	0.9313	+7.4% vagy -6.9%
tera	1.100	0.9095	+10.0% vagy -9.1%
peta	1.126	0.8882	+12.6% vagy -11.2%
exa	1.153	0.8674	+15.3% vagy -13.3%
zetta	1.181	0.8470	+18.1% vagy -15.3%
yotta	1.209	0.8272	+20.9% vagy -17.3%

Felmerült az igény egy új szabvány elkészítéssére. Az SI által befogadott IEC 60027-2 szabványt a Magyar Szabványügyi Testület 2007-ben honosította, és MSZ EN 60027-2 néven kihirdette. IEC=International Electrotechnical Commission, Nemzetközi Elektrotechnikai Bizottság, http://www.iec.ch/.

Az ajánlás szerint az SI rendszerben rögzített prefixumokat ezután kizárólag a decimális alapú értelmezésükben (kilo=1000) lehessen használni, még a számítógépes technikában is. Viszont mivel a számítástechnikának bizonyítottan szüksége van egységes bináris prefixumokra, azokra új elnevezések bevezetését javasolják.

SI (decimális)				IEC (bináris)			
jel	**név**	**érték**		**jel**	**név**	**érték**	
k	kilo	10^3	1000^1	Ki	kibi	2^{10}	1024^1
M	mega	10^6	1000^2	Mi	mebi	2^{20}	1024^2
G	giga	10^9	1000^3	Gi	gibi	2^{30}	1024^3
T	tera	10^{12}	1000^4	Ti	tebi	2^{40}	1024^4
P	peta	10^{15}	1000^5	Pi	pebi	2^{50}	1024^5
E	exa	10^{18}	1000^6	Ei	exbi	2^{60}	1024^6
Z	zetta	10^{21}	1000^7	Zi	zebi	2^{70}	1024^7
Y	yotta	10^{24}	1000^8	Yi	yobi	2^{80}	1024^8

A táblázat alapján megállapítható például, hogy 1 kibibit (Kibit)= = 1024 bit, azaz 1,024 kilobit (kbit). Hasonlóképp: 1 gibibyte (Gibyte)= = 1 073 741 824 byte $\approx$ 1073,7 megabyte (Mbyte), vagy 1024 mebibyte (Mibyte, MiB).

A bit rövidítésére a *b* használható, bár a tévedés kizárása érdekében ezt kevésszer alkalmazzák. A byte rövidítése pedig B, azaz a terabyte Tbyte vagy TB alakban rövidíthető.

Nagy az ellenállás az új szabvány elfogadásában, pl. a JEDEC (félvezetőipari mérnöki tanács, Solid State Technology Association, korábbi neve alapján Joint Electron Devices Engineering Council) egy 2002-ben frissített kiadványának szójegyzékében ez áll: a kilo (K) (félvezetőmemória kapacitásegységének prefixumaként) 1024 (2^{10}) értékű szorzó. Figyeljük meg a nagy k (K) használatát a kilo jelölésére. Hasonlóan a a mega (M) és a giga (G) is 1024^2 és 1024^3 értékű szorzók.

Hasonló ellentmondást találhatunk az adatátviteli sebesség mérésekor.

Itt az alapegység a bit/s, vagy bps azaz a bit per másodperc (secundum).

Bit ráták		
Decimal prefixumok (SI)		
Név	jel	hatvány
kilobit per second	kbit/s	10^3
megabit per second	Mbit/s	10^6
gigabit per second	Gbit/s	10^9
terabit per second	Tbit/s	10^{12}
Binary prefixumok (IEC 60027-2)		
kibibit per second	Kibit/s	2^{10}
mebibit per second	Mibit/s	2^{20}
gibibit per second	Gibit/s	2^{30}
tebibit per second	Tibit/s	2^{40}

Ellentétben a memória kapacitás mérésével itt sosem használtak 1024-es megközelítést, tehát mindig az SI rendszer szerint adták meg a mértékeket, azaz a IEC szabvány használatára nincs szükség a gyakorlatban.

$$\begin{aligned} 1\ 000\ \text{bit/s} &= 1\ \text{kbit/s (egy kilobit vagy ezer bit másodpercenként)} \\ 1\ 000\ 000\ \text{bit/s} &= 1\ \text{Mbit/s (egy megabit vagy egy millió bit másodpercenként)} \\ 1\ 000\ 000\ 000\ \text{bit/s} &= 1\ \text{Gbit/s (egygigabit vagy egy milliárd bit másodpercenként)} \end{aligned}$$

Tipikus példákat találunk az vezeték mentes (WiFi, wireless) adatátviteli szabványokban. :

802.11g	54 Mbit/s
802.11n	600 Mbit/s
802.11ac	$\approx$ 1000 Mbit/s
802.11ad	$\approx$ 7 Gbit/s

A digitális multimédiában a bitráta gyakran azt reprezentálja, hogy körülbelül mi az a minimális érték egy átlagos hallgató vagy néző számára, ami a hozzáférhető legjobb tömörítés használata esetén nem jelent érezhető különbséget a referencia mintához képest.

Az adatvesztéses tömörítés használó MP3 szabvány esetében a bitráta 32-320 kbps tart. Ami az éppen még érthető beszédtől a legmagasabb minőségig terjed. Adatvesztés mentes tömörítést használ a FLAC (Free Lossless Audio Codec) szabvány audio CD tömörítésére 400 kbps–tól 1 411 kbps-ig terjedő bitrátával. Maximum 40 Mbps bitrátát alkalmaznak a Blu-ray lemezeknél videók tárolására.

2.5. Az entrópia és tulajdonságai

A mérték definiálásánál az $n = 2^m$ és az egyforma valószínűség erős kikötések. A forrásból kibocsátott jelek előfordulási valószínűsége ugyanis különböző. Például a magyar nyelvben (angolban is) a leggyakoribb betű az „e". (Ezt a billentyűt használjuk a legtöbbet.) Ez a valószínűség 0,1, ami azt jelenti, hogy egy elég hosszú szövegben a betűk 10%-a „e" betű.

Definíció. Az $A_1, A_2, \ldots, A_n$ jeleket rendre $p_1, p_2, \ldots, p_n$ valószínűségekkel kibocsátó adó (rendszer), ahol

$$p_1 + p_2 + \cdots + p_n = 1 \quad \text{és} \quad 0 \leqslant p_i \leqslant 1 \quad (i = 1,2,\ldots,n),$$

átlagos információját a valószínűségekkel súlyozott középértékkel jellemezhetjük, vagyis

$$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i \ ,$$

amit a rendszer **bizonytalanságának**, határozatlanságának vagy **entrópiájának** is nevezünk.

Megjegyezzük, hogy a rendszer entrópiája objektív mérőszám, függetlenül attól, hogy az információt értjük, vagy nem. Az információ ugyanis a rendszerben van és nem a megfigyelő tudatában. A bizonytalanság szóhasználat arra utal, hogy egy jel kibocsátásakor annyi információt nyerünk, amennyi bizonytalanság éppen megszűnik.

A fenti definíció nincs ellentmondásban az előzőek során már ismertetett fogalommal, ahol a jelek kibocsátásának valószínűsége egyenlő.

$$p_i = \frac{1}{n}, \quad (i = 1,2,\ldots,n)$$

és

$$p_1 + p_2 + \cdots + p_n = n\frac{1}{n} = 1$$

miatt

$$-\sum_{i=1}^{n} p_i \log p_i = -\sum_{i=1}^{n} \frac{1}{n} \log \frac{1}{n} = -\sum_{i=1}^{n} \frac{1}{n}(-\log n) = n\frac{1}{n} \log n \ ,$$

vagyis

$$H(p, p, \ldots, p) = H(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) = \log n \ .$$

Az, hogy a definíció hűen tükrözi-e a valóságot, a rendszer bizonytalanságait, azt az alkalmazás, a gyakorlat dönti el. Ennek bemutatására vizsgáljunk meg néhány példát:

1. Hasonlítsuk össze három adó entrópiáját, amelyek mindegyike egyformán két-két jelet bocsát ki, de más-más valószínűséggel.

adó	jelek	valószínűségek
1.	$A_1,\ A_2$	1/2 1/2
2.	$B_1,\ B_2$	5/8 3/8
3.	$C_1,\ C_2$	15/16 1/16

$$\begin{aligned} H_1(1/2{,}1/2) &= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1 \text{ bit} \ , \\ H_2(5/8{,}3/8) &= -\frac{5}{8}\log\frac{5}{8} - \frac{3}{8}\log\frac{3}{8} \approx 0{,}954 \text{ bit} \ , \\ H_3(15/16{,}1/16) &= -\frac{15}{16}\log\frac{15}{16} - \frac{1}{16}\log\frac{1}{16} = \\ &= \frac{15}{16}(4 - \log 15) + \frac{1}{16}(4 - \log 1) \approx 0{,}337 \text{ bit} \ . \end{aligned}$$

A harmadik adónál majdnem biztos, hogy a C_1 jel kerül kibocsátásra, a másodiknál már sokkal nehezebb, az elsőnél pedig a legnehezebb megjósolni, hogy melyik jel kerül kibocsátásra. Ez összhangban van azokkal az eredményekkel, amelyeket kaptunk

$$H_3 < H_2 < H_1 \ .$$

Az első adóhoz tartozó bizonytalanság jóval nagyobb mint a harma-

dikhoz tartozó és nagyobb a másodikhoz tartozónál is.

2. Adott helyen annak a valószínűsége,

$$\text{hogy június 10-én esik az eső: } p_1 = 0{,}4\ ,$$
$$\text{hogy nem esik: } p_2 = 0{,}6\ ;$$
$$\text{hogy november 20-án esik az eső: } q_1 = 0{,}65\ ,$$
$$\text{hogy hó esik: } q_2 = 0{,}15\ ,$$
$$\text{hogy nem esik: } q_3 = 0{,}2\ .$$

   a) Ha csak az érdekel bennünket, hogy esik vagy nem esik, akkor mivel

$$H(p_1, p_2) = -0{,}4 \log 0{,}4 - 0{,}6 \log 0{,}6 \approx 0{,}97 \text{ bit}$$

   és

$$H(q_1 + q_2, q_3) = -0{,}8 \log 0{,}8 - 0{,}2 \log 0{,}2 \approx 0{,}72 \text{ bit},$$

   ezért a június 10-i időjárás határozatlanabb.

   b) Ha a csapadék minősége is (eső, hó) érdekel, akkor viszont a november 20-i időjárás határozatlanabb, mert

$$H(q_1, q_2, q_3) = -0{,}65 \log 0{,}65 - 0{,}15 \log 0{,}15 - 0{,}2 \log 0{,}2 \approx 1{,}28 \text{ bit}$$

   és

$$H(p_1, p_2) \approx 0.97 \text{ bit}.$$

3. Kilenc db egyforma pénzünk van, ezek közül 1 könnyebb, hamis. Serpenyős mérlegen mérősúlyok nélkül hány méréssel állapíthatjuk meg, hogy melyik a hamis?

   Mivel hamis a 9 érme közül bármelyik ugyanolyan valószínűséggel lehet, ezért

$$H_1 = \log 9 = 2 \log 3\ .$$

Végezzünk n mérlegelést. Ezeknek egyenként 3 eredménye (kimenete) lehet:

— bal serpenyő süllyed,

— jobb serpenyő süllyed,

— egyensúlyban lesz a két serpenyő,

így

$$n \log 3 \geqslant 2 \log 3 \ ,$$

ahonnan

$$n \geqslant 2$$

Ha a mérlegelést úgy végezzük, hogy a kimenetek valószínűsége közel egyenlő, akkor $n = 2$ mérés elegendő is:

1. mérésnél:	3 db	3.db	kimarad: 3 db
valószínűségek:	$^1/_3$	$^1/_3$	$^1/_3$
2. mérésnél:	1 db	1 db	kimarad: 1 db
valószínűségek:	$^1/_3$	$^1/_3$	$^1/_3$

Az entrópiafüggvény tulajdonságai.

Tétel. *A $H(p_1, p_2, \ldots, p_n)$ függvény folytonos valamennyi p_i változójában a* (0,1] *intervallumon.*

Bizonyítás. A (0,1] intervallumon a logaritmus függvények és azok összege is folytonos. □

Tétel. *Az entrópiafüggvény minden változójában szimmetrikus.*

Bizonyítás. A definíció szerint az entrópiafüggvény invariáns a sorrendre nézve, mivel az összeg felcserélhető, vagyis

$$H(p_1, p_2, p_3, \ldots, p_n) = H(p_2, p_1, p_3, \ldots, p_n) \ .$$

□

Tétel. *Az entrópiafüggvény maximumát akkor veszi fel, amikor a valószínűségek egyenlők, vagyis*

$$H_{\max} = H\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right) = \log n > 0.$$

Bizonyítás. A bizonyítást csak $n = 2$ esetén végezzük el. Ekkor $p_1 = p$ és $p_2 = 1 - p$, továbbá

$$H(p,1-p) = -p \log p - (1-p) \log(1-p).$$

Ha jobb oldal függvénynek tekintjük, akkor az ott veheti fel a maximumát, ahol a p szerinti első deriváltja egyenlő nullával[6]

$$diff(-p \log p - (1-p) \log(1-p), p) = \frac{\ln(1-p) - \ln p}{\ln 2}$$

innen $\ln(1-p) - \ln p = 0$ egyenletből

$$\ln \frac{(1-p)}{p} = 0 \ ,$$

így $\frac{(1-p)}{p} = 1$ $(\ln 1 = 0)$, ahonnan $1 - p = p$, azaz $2p = 1$ és $p = \frac{1}{2}$. Mivel a második deriált

$$diff(\ln(1-p) - \ln p, p) = \frac{1}{p-1} - \frac{1}{p} = \frac{1}{p(p-1)}$$

$p = \frac{1}{2}$ helyen negatív, ezért a $H(p,1-p)$ függvénynek a $p = \frac{1}{2}$ pontban abszolút maximuma van. Így

$$H\left(\frac{1}{2}, \frac{1}{2}\right) = \log 2 > 0$$

□

[6] A felírásban a MAPLE komputeralgebrai rendszer szintaktikáját használtunk.

Általános bizonyítás nélkül igaz,

$$\begin{aligned} H\max &= H\left(\frac{1}{n},\frac{1}{n},\ldots,\frac{1}{n}\right)=\log n>0,\\ H\min &= H(1,0,\ldots,0)=H(0,1,\ldots,0)=\ldots=0. \end{aligned}$$

Tétel. *A jelek számának növelésével, továbbá bontással a bizonytalanság nem csökken, vagyis*

$$H(p_1,p_2,\ldots,p_{n-1},q_1,q_2,\ldots,q_k,)\geqslant H(p_1,p_2,\ldots,p_n)\ ,$$

ahol $p_n=q_1+q_2+\cdots+q_k$.

Bizonyítás. Ha az alábbi állítást bebizonyítjuk, akkor a tételt is igazolást nyer:

$$\begin{aligned} &H(p_1,p_2,\ldots,p_{n-1},q_1,q_2,\ldots,q_k,)=\\ &\qquad=H(p_1,p_2,\ldots,p_n)+p_nH\left(\tfrac{q_1}{p_n},\ldots,\tfrac{q_k}{p_n}\right)\ . \end{aligned} \tag{2.1}$$

Az egyenlet bal oldalát átalakítva

$$\begin{aligned} H(p_1,p_2,\ldots,p_{n-1},q_1,q_2,\ldots,q_k,) &= -\sum_{i=1}^{n-1}p_i\log p_i-\sum_{i=1}^{k}q_i\log q_i=\\ &= -\sum_{i=1}^{n}p_i\log p_i+p_n\log p_n-\sum_{i=1}^{k}q_i\log q_i=\\ &= H(p_1,p_2,\ldots,p_n)+p_n\log p_n-\sum_{i=1}^{k}q_i\log q_i. \end{aligned}$$

Ezek után elég a kapott összeg bal felével foglalkozni.
Mivel $q_1+q_2+\cdots+q_k=p_n$, ezért

$$\frac{q_1}{p_n}+\frac{q_2}{p_n}+\cdots+\frac{q_k}{p_n}=1\quad\text{azaz}\quad\sum_{i=1}^{k}\frac{q_i}{p_n}=1$$

és

$$p_n \log p_n - \sum_{i=1}^{k} q_i \log q_i = p_n \sum_{i=1}^{k} \frac{q_i}{p_n} \log p_n - p_n \sum_{i=1}^{k} \frac{q_i}{p_n} \log q_i =$$
$$= -p_n \sum_{i=1}^{k} \frac{q_i}{p_n} \log \frac{q_i}{p_n} = p_n H\left(\frac{q_1}{p_n}, \ldots, \frac{q_k}{p_n}\right) .$$

ezzel bebizonyítottuk a (2.1) állítást és így a tételt is.

Tekintsünk egy példát:

$$p_1 = \frac{1}{2}, \quad p_2 = \frac{1}{4}, \quad p_3 = \frac{1}{4}$$

esetén a p_3-at bontsuk fel $q_1 = \frac{1}{8}$, és $q_2 = \frac{1}{8}$ -ra, azaz

$$p_3 = q_1 + q_2.$$

Ekkor

$$H(p_1, p_2, q_1, q_2) = H(p_1, p_2, p_3) + p_3 H\left(\frac{q_1}{p_3}, \frac{q_2}{p_3}\right)$$

alapján

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot \left(\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1\right)$$

és

$$1\frac{3}{4} = 1\frac{1}{2} + \frac{1}{4} .$$

□

3. A KÓDOLÁS

3.1. A kódolás célja, feladata

A kódolás az információátvitel és alkalmazások szempontjából az információelmélet egyik legfontosabb területe. A kódolást szükségessé teszi egyrészt az a tény, hogy az adó jeleit, a továbbító csatorna általában nem tudja értelmezni, mert technikailag csak más jelek továbbítására alkalmas. Másrészt a kódolással az átvitel hatásfokát is javítani szeretnénk, és végül feltesszük, hogy a csatorna az adó jeleit nem torzítja el, vagyis a csatorna zajmentes.

Tegyük fel, hogy az adó

$$A_1, A_2, \ldots, A_n$$

jeleket bocsát ki

$$p_1, p_2, \ldots, p_n$$

valószínűségekkel, és a csatorna

$$x_1, x_2, \ldots, x_m$$

jeleket képes továbbítani, ahol $n \gg m$ (általában n jóval nagyobb mint m). Mi többnyire az $x_1 = 0$, $x_2 = 1$ és $m = 2$, tehát a két jelet fogadó bináris csatornával foglakozunk.

Definíció. A **kódolás** az A_i jeleknek az x_i jelek sorozatára történő kölcsönösen egyértelmű leképezése úgy, hogy az egyértelműen dekódolható legyen.

A kölcsönösen egyértelmű megfeleltetés azt jelenti, hogy az A_i-*hez* rendelt kódszó, különbözik az A_k-*hoz* rendeltől ($i \neq k$). Az egyértelmű dekódolhatóság pedig azt jelenti, hogy különböző közleményekhez különböző kódsorozatok tartoznak.

Egy adott jelrendszerhez több kódolási előírást is megvalósíthatunk ugyanazokkal a csatornajelekkel. Ezek hatásfoka különböző lehet. Célszerű tehát ezeket közelebbről is megvizsgálni. Vegyünk egy egyszerű kódolási problémát:

Legyen $n = 4$, a jelek $A_1,\ A_2,\ A_3,\ A_4$, a megfelelő valószínűségek $^1/_2$, $^1/_4$, $^1/_8$, $^1/_8$ és $m = 2,\ x_1 = 0,\ x_2 = 1$, valamint a kódolt közlemény

$$\ldots 10010001101110 \ldots$$

Tekintsük a következő táblázatban megadott kódolási szabályokat (K1, K2, K3, K4):

		K1	K2	K3	K4
p	Jel	Kódok			
$^1/_2$	A_1	00	1	0	0
$^1/_4$	A_2	01	10	10	01
$^1/_8$	A_3	10	100	110	10
$^1/_8$	A_4	11	1000	111	11

Dekódoljuk a közleményt.

K1 esetén $\ldots \mid 10 \mid 01 \mid 00 \mid 01 \mid 10 \mid 11 \mid 10 \mid \ldots$
$A_3\ A_2\ A_1\ A_2\ A_3\ A_4\ A_3$

K2 esetén $\ldots \mid 100 \mid 1000 \mid 1 \mid 10 \mid 1 \mid 1 \mid 10 \mid \ldots$
$A_3\ A_4\ A_1\ A_2\ A_1\ A_1\ A_2$

K3 esetén $\ldots \mid 10 \mid 0 \mid 10 \mid 0 \mid 0 \mid 110 \mid 111 \mid 0 \mid \ldots$
$A_2\ A_1\ A_2\ A_1\ A_1\ A_3\ A_4\ A_1$

K4 esetén 1.eset $\ldots \mid 10 \mid 0 \mid 10 \mid 0 \mid 0 \mid 11 \mid 01 \mid 11 \mid 0 \mid \ldots$
$A_3\ A_1\ A_3\ A_1\ A_1\ A_4\ A_2\ A_4\ A_4$

K4 esetén 2.eset $\ldots \mid 10 \mid 01 \mid 0 \mid 0 \mid 01 \mid 10 \mid 11 \mid 10 \mid \ldots$
$A_3\ A_2\ A_1\ A_1\ A_2\ A_3\ A_4\ A_3$

Láthatjuk, hogy a K4 kódrendszer nem egyértelműen dekódolja a közle-

ményt, de a többi igen. Vegyük észre, hogy az egyértelmű megfejtéshez nem volt szükségünk az elemek közötti határolójelekre sem a K1, K2, K3 kódrendszernél.

Definíció. Azokat a kódokat, amelyekkel a közlemények az elemek közötti határolójelek alkalmazása nélkül egyértelműen dekódolhatók szeparálható, vagy egyértelműen megfejthető kódoknak nevezzük.

A közönséges nyelv szavai nem szeparálhatók, mert pl.

– mást jelent a kalapács, mint a kalap meg az ács szóközzel elválasztva

– vagy pl. búvár; bú, vár,

– törülköző; tör, ül, köz, ő.

Az egyértelmű megfejthetőségnek elégséges feltétele az, hogy egyetlen kódot se lehessen megkapni valamely másikból további betűk hozzávételével. (Egyik kód sem eleje egy másik kódnak.)

Definíció. Az olyan kódokat, amelyeknél egyik kód sem eleje egy másik kódnak, prefix tulajdonságú, vagy irreducibilis kódoknak nevezzük.

A K2 esetnél megadott kódok nem irreducibilisek, mert pl. 1 a 10-nek eleje, 10 a 100-nak eleje és így tovább, de szeparábilis vagyis egyértelműen megfejthető. Tehát az irreducibilis kódok az egyértelműen megfejthető kódok egy szűkebb osztályát alkotják.

3.2. A kódolás hatásfoka

A kódoknak a csatornán való továbbítása bizonyos „költséggel" jár. (Gondoljunk arra, hogy pl. a távirati díj nemcsak a szavak számától, hanem a hosszától is függ.) A legegyszerűbb költségfüggvényt úgy kapjuk, hogy minden egyes A_i-hoz hozzárendeljük a kódját alkotó jelek h_i számát, vagyis a kód h_i hosszát ($i = 1,2,\ldots,n$), mert akkor a közleményenkénti átlagos költség arányos a közleményeket alkotó jelek számának átlagával.

Definíció. Az $A_1, A_2, \ldots, A_n$ jeleket $p_1, p_2, \ldots, p_n$valószínűséggel kibocsátott jelek $h_1, h_2, \ldots, h_n$ hosszúságú kódsorozatának **átlagos hossza**, a hosszaknak a megfelelő valószínűségekkel súlyozott középértéke (várható érték), vagyis

$$h_{átl} = \sum_{i=1}^{n} p_i h_i \ .$$

Hatásosabb kódrendszernek azt a kódrendszert nevezhetjük, melyhez kisebb *átlagos hossz* (szóhossz) tartozik. A feladat pedig olyan kódolási eljárás keresése és megvalósítása, amelynél $h_{átl}$ *értéke minimális.*

Az előzőek során megadott kódrendszereknél:

K1 esetén: mivel minden kódban $h_i = 2$, ezért itt $h_{átl} = 2$,

$$h_{átl} = 2 \cdot \ (^1/_2 + {}^1/_4 + {}^1/_8 + {}^1/_8) = 2.$$

K2 esetén:

$$h_{átl} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 4 = \frac{15}{8} = 1\frac{7}{8}.$$

K3 esetén:

$$\begin{aligned} h_{átl} &= \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = \frac{14}{8} = 1\frac{6}{8} \\ &= 1\frac{3}{4} = \frac{7}{4}. \end{aligned}$$

Az átlagos hossz tehát a K3 kódrendszernél a legkisebb a fenti kód rend-

szerek közül. (A K4 kódrendszer nem szeparálható kódrendszer, így az átlagos hossz vizsgálatának itt nincs is értelme.)

Bebizonyítható, hogy az átlagos hossz minimuma

$$\min(h_{átl}) = \frac{H(A)}{\log m} \ ,$$

ahol $H(A)$ az adó entrópiája, m pedig a kódábécé jeleinek a száma, tehát

$$h_{átl} \geqslant \frac{H(A)}{\log m} \ .$$

Egyenlőség akkor érhető el, ha

$$p_i = \frac{m^{-h_i}}{\sum\limits_{i=1}^{n} m^{-h_i}} \qquad \text{és} \qquad \sum_{i=1}^{n} m^{-h_i} = 1 \ .$$

Bináris kódolásnál, mivel $m = 2$, ezért

$$\min(h_{átl}) = H(A) \ ,$$

és ez akkor érhető el, ha

$$p_i = 2^{-h_i} \qquad \text{és} \qquad \sum_{i=1}^{n} 2^{-h_i} = 1 \ .$$

Mivel példánkban

$$H(A) = -\sum_{i=1}^{4} p_i \log p_i = \frac{1}{2} + \frac{1}{4} \cdot 2 + 2 \cdot \frac{1}{8} \cdot 3 = \frac{7}{4} \ ,$$

tehát a K3 kódrendszer minimális átlagos hosszat ad, tehát ezt tarthatjuk a leghatékonyabbnak.

Definíció. Egy **kódokási eljárás hatásfoka** [7] a kódolt jelek jelenkénti átlagos információjának $(H(A)/h_{átl})$ és a kódábécé lehetséges maximális

[7] A definícióban a hatásfok jelölésére a szokásos görög η (éta) betűt használjuk.

jelenkénti információjának a hányadosa, azaz

$$\eta = \frac{H(A)}{h_{átl} \log m}.$$

Bináris kódolásnál: $\eta = \frac{H(A)}{h_{átl}}$. Mivel $\eta \leqslant 1$, ezért

$$\frac{H(A)}{\log m} \leqslant h_{átl} \ , \qquad \text{vagyis} \qquad \min(h_{átl}) = \frac{H(A)}{\log m} \leqslant h_{átl} \ .$$

Definíció. Egy kódolási eljárás **redundanciája** (terjengössége) az

$$r = 1 - \eta$$

értékkel jellemezhető.

A hatásfok és a redundancia értékeit is általában %-ban fejezzük ki. A definíció értelmében a legnagyobb hatásfok egyenértékű a legkisebb redundanciával.

Mintafeladatunkban

	Hatásfok	Redundancia
K1	$\frac{7}{4} : 2 = \frac{7}{8} = 87{,}5\%$	12,5%
K2	$\frac{7}{4} : \frac{15}{8} = \frac{14}{15} \approx 93{,}3\%$	$\approx 6{,}7\%$
K3	$\frac{7}{4} : \frac{7}{4} = 1 = 100\%$	0%

Tehát valóban a K3 kódrendszer a leghatásosabb. Vegyük észre, hogy a legnagyobb valószínűséggel előforduló jelnek van a legrövidebb kódja.

$^1/_2$ valószínűségű jel kódja 0, – egy jegy,

$^1/_4$ valószínűségű jel kódja 10, – két jegy,

$^1/_8$ valószínűségű jelek kódja három jegy.

3.3. Kódolási eljárások

Nyilván fontos számunkra, hogy hogyan tudunk konstruálni nagy hatásfokú kódokat, egyáltalán szeparálható kódokat. Mi itt döntően csak a bináris kódolással foglalkozunk, mivel azok a számítógépek és automaták fontos alkalmazási területei.

3.3.1. Szeparálható bináris kódolás

Ha csak a szeparálhatóság az egyetlen kikötés, akkor a következő egyszerű kódolási algoritmust alkalmazhatjuk.

1. lépés: Osszuk fel a jelek

$$J = \{A_1, A_2, \ldots, A_n\}$$

halmazát, két tetszőleges, de nem üres J_0, J_1 részhalmazra:

$$J_0 = \{A_1, A_2, \ldots, A_k\} \ , \qquad J_1 = \{A_{k+1}, A_{k+2}, \ldots, A_n\} \ .$$

Rendeljünk 0-át minden J_0-beli és 1-et minden J_1-beli jelhez.

2. lépés: Az első lépést megismételjük a keletkezett részhalmazokra is, mindaddig, amíg csupa egyetlen jelből álló részhalmazokat nem kapunk. A 0 és 1 jeleket a már meglévő kódok mögé írjuk. Tehát J_0-át két részre osztva: a J_{00} és a J_{01} , illetve J_1-et is két részre osztva: a J_{10} és a J_{11} részhalmazokhoz jutunk. A J_{00} minden jeléhez 00-val, J_{01}-beliekhez pedig 01-gyel kezdődő kódok tartoznak és így tovább.

Ha pl. a J_{01011} részhalmaz már egyetlen jelet tartalmaz, akkor ehhez a jelhez a 01011 kód tartozik. Az eljárás bemutatására vegyük elő ismét a mintafeladatunkat, ahol

$$J = \{A_1, A_2, A_3, A_4\} \ .$$

	1. lépés	2. lépés	kód
A_1	0	0	00
A_2	0	1	01
A_3	1	0	10
A_4	1	1	11

A halmazokat itt egyenlő részekre osztottuk.

Az így kapott kódrendszer megfelel a K1 kódrendszernek.

Látható, hogy a bináris kódolási eljárás prefix tulajdonságú kódokat generál és így biztos, hogy szeparálható is.

Az eljárás nem bináris pl. $m = 3$ esetén, úgy alkalmazható, hogy három részre osztjuk a halmazokat, amelyekhez a 0,1,2 jeleket rendeljük és így tovább.

A jelek felosztása sokféleképpen elvégezhető. Megtehetjük pl. hogy egyszerre csak egy jelet választunk le, az alábbi módon:

$$J_0 = \{A_1\} \longrightarrow 0$$

$$J_1 = \{A_2, A_3, \ldots, A_n\} \qquad \text{1-gyel kezdődő kódok}$$

$$J_{10} = \{A_2\} \longrightarrow 10$$

$$J_{11} = \{A_3, A_4, \ldots, A_n\} \qquad \text{11-gyel kezdődő kódok.}$$

Példánkban

	1. lépés	2. lépés	3. lépés	kód
A_1	0			0
A_2	1	0		10
A_3	1	1	0	110
A_4	1	1	1	111

ami megfelel a K3 kódrendszernek.

A kód hatásfoka különösen fontos, ezért a felosztást a valószínűségek figyelembevételével célszerű elvégezni. Erre alapozódik a következő kódolási eljárás.

3.3.2. A Shannon–Fano-féle kódolás

1. lépés: A jeleket valószínűségeik csökkenő sorrendjében írjuk fel.

2. lépés: A jelek halmazát két, lehetőleg egyenlő valószínűségű részhalmazra osztjuk: J_0, J_1. Az egyik részhalmazba tartozó minden jelhez a 0-át, minden megmaradó jelhez pedig az 1-et rendeljük.

3. lépés: A 2. lépést megismételjük valamennyi részhalmazra, mindaddig amíg minden részhalmaz már csak egy jelet tartalmaz.

Ennél az eljárásnál az egyenlő valószínűségi felosztás miatt a 0,1 egyenlő valószínűséggel fordulhat elő, tehát a kódolt jelek jelenként majdnem 1 bit információt továbbítanak.

Példánk alapján az eljárás

p	Jel	1. lépés	2. lépés	3. lépés	kód
$^1/_2$	A_1	0			0
$^1/_4$	A_2	1	0		10
$^1/_8$	A_3	1	1	0	110
$^1/_8$	A_4	1	1	1	111

a K3 kódrendszert szolgáltatja, melyről mint már láttuk 100%-os hatásfokú. Ezt a hatásfokot azonban csak akkor érhetjük el, ha az egyenlő valószínűségre való felosztást sorozatosan megvalósíthatjuk. Törekedni kell legalább a „közelítőleg" egyenlő valószínűségű részekre való felosztásra. Ennek illusztrálására tekintsük a következő példát, ahol az adó hét jelet bocsájt ki különböző

valószínűségekkel:

jelek	A_1	A_2	A_3	A_4	A_5	A_6	A_7
valószínűségek	0,48	0,13	0,13	0,08	0,07	0,06	0,05

A felosztást az alábbi módon végezhetjük:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7
1. lépés	0	1	1	1	1	1	1
2. lépés		10	10	11	11	11	11
3. lépés		100	101	110	110	111	111
4. lépés				1100	1101	1110	1111

Az átlagos hossz:

$$h_{átl} = 0{,}48 \cdot 1 + 0{,}13 \cdot 3 \cdot 2 + 0{,}26 \cdot 4 = 2{,}30 \ .$$

Az entrópia:

$$H(A) = -(0{,}48 \log 0{,}48 + 0{,}26 \log 0{,}13 + 0{,}08 \log 0{,}08 +$$
$$+0{,}07 \log 0{,}07 + 0{,}06 \log 0{,}06 + 0{,}05 \log 0{,}05) = 2{,}2932 \ .$$

A hatásfok tehát:

$$\eta = \frac{H(A)}{h_{átl}} = \frac{2{,}2932}{2{,}30} = 0{,}9969 = 99{,}69\%.$$

3.3.3. Huffman kód

Figyelembe véve, hogy az eredeti input kép pontjai azonos hosszúságú elemekből épülnek fel a használt színek számától függően (pl. 1 képpont 1 byte-on tárolódik). Igen hatékony tömörítést érhetünk el, ha a leggyakrabban előforduló elemeket (kép esetén ez általában háttérszín) rövidebb kóddal helyettesítjük. Ezen az elven alapszik a Huffman kód is. Ennek a kódnak a lényege, hogy meghatározzuk az input kép elemeinek előfordulási valószínűségeit vagy az előfordulási gyakoriságait. Tehát az input kép különböző elemeit tekintsük input ABC-nek, pl. ha a kép maximum 256 színt használ (1 byte-os tárolás), akkor értelemszerűen az input ABC elemszáma is maximum 256. Az input ABC elemeit növekvő sorrendbe állítjuk előfordulási valószínűségik szerint és a sorrendnek megfelelően egyre rövidebb kódot rendelünk az elemekhez. Egy elem előfordulási valószínűségét megkapjuk, ha megszámoljuk az összes előfordulását (ez az előfordulási gyakoriság), és elosztjuk az összes elem összegzett elemszámával. Természetesen adatvesztés nélküli, egyértelműen dekódolható kódot kell előállítanunk. Ezek után tekintsük az algoritmust, amelynek során egy bináris fát építünk fel:

1. Legyen az OP halmaz az előfordulási valószínűségek halmaza!
2. Az előfordulási valószínűségekből létrehozzuk a fa levélelemeit.
3. Legyen P_i és P_j az OP halmaz két legkisebb eleme!
   a) Hozzunk létre egy új csomópontot az N_{ij}-t, amely a P_i és P_j apja lesz a fában!
   b) A kisebbik valószínűségű csomópont él címkéje legyen 0, a nagyobbik él címkéje pedig 1!
   c) Legyen $P(N_{ij}) = P_i + P_j$! A P_i-t és a P_j-t töröljük az OP halmazból és $P(N_{ij})$-t felvesszük az OP halmazba.
4. Ha az OP halmaz 1 elemű, akkor vége, egyébként folytassuk az algoritmust a 3. ponttól!

Az algoritmus vége után egy bináris fát kapunk, melynek levelei az input ABC elemei. A gyökérből az egyes levél elemhez vezető úton lévő címkéket egymásután írva kapjuk az inputelem kódját. Az algoritmusból adódik, hogy a gyökérhez nem rendelünk címkét. Az *OP* halmaz előállítása például történhet úgy, hogy megszámláljuk egy elem (pl. egy szín) előfordulását, majd a kapott értéket elosztjuk az input szöveg hosszával. 256 színű kép esetén minden képpont egy byte-on tárolódik, ezért ilyenkor a kép byte-okban mért hosszával osztunk. A kódolás során figyelembe vesszük, hogy a kiszámított valószínűségek összege 1.

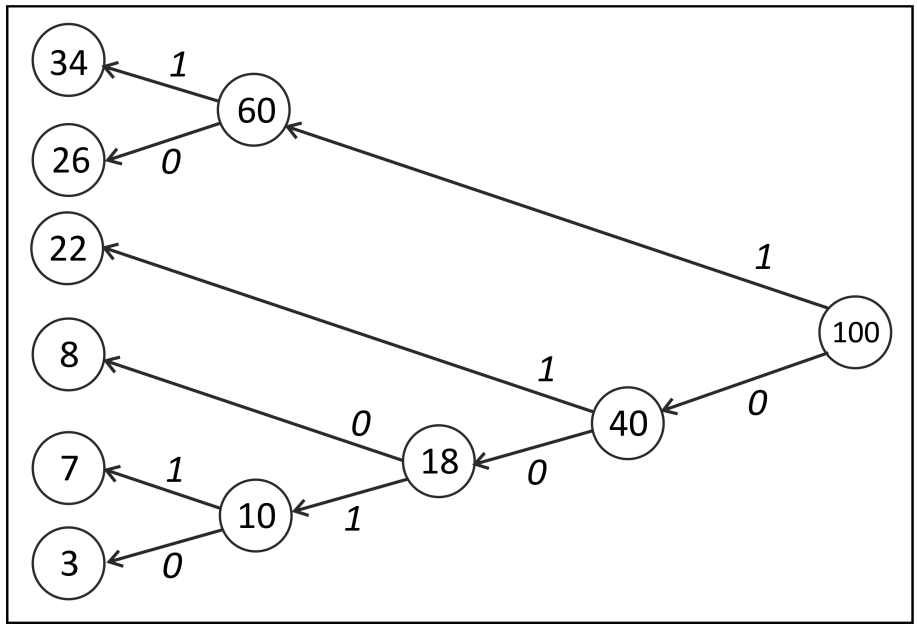
Legyen példánkban az input ABC elemeinek hossza 1 byte!

Előforduló színek:	*Előfordulási valószínűségek (%):*	*Kód*
10001010	34	11
10100101	26	10
00010100	22	01
01101011	8	000
00111011	7	0011
00101001	3	0010

A kódolás után a kódolt állományban le kell tárolni a kódtáblát is, amely tartalmazza, hogy az input ABC elemeihez milyen kód tartozik. Igen hatékony tömörítést tudunk elérni, pl. szövegek esetén a tömörítési arány az 50% -ot is meghaladhatja.

Az átlagos hossz:

$$h_{átl} = 0{,}34 \cdot 2 + 0{,}26 \cdot 2 + 0{,}22 \cdot 2 + 0{,}08 \cdot 3 + 0{,}07 \cdot 4 + 0{,}03 \cdot 4 = 2{,}28 \ .$$



3.1. ábra. A Huffman kód bináris fája

Az entrópia[8]:

$$H(A) = -(0{,}34 \log 0{,}34 + 0{,}26 \log 0{,}26 + 0{,}22 \log 0{,}22 +$$
$$+0{,}08 \log 0{,}08 + 0{,}07 \log 0{,}07 + 0{,}03 \log 0{,}03) \approx 2{,}2268.$$

A hatásfok tehát:

$$\eta = \frac{H(A)}{h_{átl}} = \frac{2{,}2268}{2{,}28} = 0{,}976\dot{6} = 97{,}67\%.$$

A különböző kódolásokról összehasonlító elemzést találunk Roger Seeck által karbantartott Binary Essence angol nyelvű weblapon:
http://www.binaryessence.com

[8] Maple formában:
-(0.34*log[2](0.34)+0.26*log[2](0.26)+0.22*log[2](0.22)+0.08*log[2](0.08)+
+0.07*log[2](0.07)+0.03*log[2](0.03));

3.3.4. Adatvesztéses tömörítés

Akkor használunk adatvesztéses tömörítő algoritmusokat, ha az eredeti adathalmaz tartalmaz fölösleges adatokat is, amelyek szükségtelenek a végső alkalmazás szempontjából. Ilyen technikákat természetesen nem használnak kritikus alkalmazásokban, mint például orvosi képfeldolgozás, de annál inkább a kereskedelmi televíziózásban. Szemünk tehetetlenségéből adódik, hogy a TV képernyőjén bizonyos változásokat nem veszünk észre. Pl. ha 50 pont színe megváltozik a képernyőn, akkor észre sem vesszük, ezért sok esetben szükségtelen a túlzott színmélység vagy az éles kontúrok használata. Kiválóan alkalmazzák ezeket a megoldásokat a JPEG és a MPEG formátumú állományoknál, ahol megválaszthatjuk a kép kódolása során a tömörítés hatásfoka és a kép minősége közötti fordított arányosság mértékét.

3.4. Az egyértelmű kódolás szükséges és elégséges feltétele

Tétel. *Annak, hogy m darab kódoló jel esetén az n jelből álló*

$$A_1, A_2, \ldots, A_n \text{ jelekhez}$$
$$h_1, h_2, \ldots, h_n \quad \text{hosszúságú}$$

szó hozzárendelésével létezzen prefix kódrendszer, ami egyértelműen dekódolható, szükséges és elégséges feltétele, hogy

$$\sum_{i=1}^{n} m^{-h_i} \leqslant 1 \ .$$

Ha pl. $n = 2$, akkor a bináris kódrendszerben az egyértelműen megfejthetőség szükséges és elégséges feltétele

$$\sum_{i=1}^{n} 2^{-h_i} \leqslant 1 \ .$$

Alkalmazásként vizsgáljuk meg, hogy mikor létezik olyan n kódszóból álló prefix kód, amelynek minden szava egyenlő h hosszúságú. A tétel szerint

$$\sum_{i=1}^{n} m^{-h} = n \ m^{-h} \leqslant 1 \ ,$$

vagyis $n \leqslant m^h$, azaz $\log n \leqslant h \log m$.

Az $m = 2$ esetben, $n = 256$ jelre

$$\log 256 = \log 2^8 \leqslant h \log 2 \text{ vagyis } 8 \leqslant h,$$

és így 8 hosszúságú szavakkal kódolható a 256 jel, amely prefix is. A prefix tulajdonság itt egyszerűen abból adódik, hogy minden kódszó különböző. Így lehet például a 256 színt kódolni 1 byte-on.

Megemlítjük, hogy a $\sum_{i=1}^{n} m^{-h_i} \leqslant 1$ feltétel nemcsak a prefix tulajdonságú, hanem valamennyi *egyértelműen megfejthető* kódrendszer szükséges

feltétele.

Vizsgáljuk meg, melyek az egyértelműen megfejthetők a mintafeladatunkban:

$$\text{K1-nél:} \quad \sum_{i=1}^{4} 2^{-h_i} = \tfrac{1}{2^2} \cdot 4 = 1 \ ,$$

$$\text{K2-nél:} \quad \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} = \tfrac{15}{16} < 1 \ ,$$

$$\text{K3-nál:} \quad \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{8} = 1 \ ,$$

$$\text{K4-nél:} \quad \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{4} + \tfrac{1}{4} = 1\tfrac{1}{4} > 1 \ .$$

Tehát a következőt állapíthatjuk meg

K1	prefix	szeparálható,
K2	nem prefix	szeparálható ,
K3	prefix	szeparálható,
K4	nem prefix	nem szeparálható.

Példa. Határozzuk meg azt a legkisebb m értéket, amelyre tervezhető prefix tulajdonságú kód az alábbi szóhossz-gyakoriságokkal:

$$s_1 = 0; \qquad s_2 = 3; \qquad s_3 = 0; \qquad s_4 = 5 \ .$$

Az ismertetett tétel alapján

$$\frac{0}{m} + \frac{3}{m^2} + \frac{0}{m^3} + \frac{5}{m^4} \leqslant 1 \ ,$$

azaz

$$3m^2 + 5 \leqslant m^4 \ , \quad \text{vagyis} \quad m^4 - 3m^2 - 5 \geqslant 0 \ ,$$

tehát

$$m^2 \geqslant 4{,}19 \quad \text{és} \quad m \geqslant 2{,}04$$

tehát $m = 3$ a legkisebb érték, azaz bináris kód nem konstruálható. Egy

lehetséges megvalósítás:

00	1000	2000
01	1001	2001
02	1002	

3.5. Feladatok a kódolásra

1. Egy adó az $A_1, A_2, A_3, A_4, A_5, A_6$ jeleket adja le $^1/_2,\ ^1/_4,\ ^1/_{16},\ ^1/_{16},\ ^1/_{16},\ ^1/_{16},\ ^1/_{16}$, valószínűségekkel. Adottak a következő kódrendszerek.

	K1	K2	K3	K4	K5	K6
A_1	0	1	0	111	1	0
A_2	10	011	10	110	01	01
$A3$	110	010	110	101	0011	011
A_4	1110	001	1110	100	0010	0111
A_5	1011	000	11110	011	0001	01111
A_6	1101	110	111110	010	0000	011111

   a) Határozzuk meg, melyek az egyértelműen megfejthető kódrendszerek.

   b) Melyek a prefix tulajdonságúak?

   c) Határozzuk meg az egyértelműen megfejthető kódok átlagos hosszát és hatásfokát.

   d) Eredményez-e a fenti kódok valamelyike minimális átlagos hosszat?

2. Adjuk meg az összes lehetséges olyan prefix tulajdonságú bináris kódot, amely az A_1, A_2, A_3 jeleket legfeljebb 3 jegy hosszúságú szavakban kódolja.

3. Állapítsuk meg, hogy a következő gyakoriságú szóhosszak megfelelnek-e egy egyértelműen megfejthető bináris kódnak?

   a) $s_1 = 0;\ s_2 = 2;\ s_3 = 3;\ s_4 = 2.$

   b) $s_1 = 0;\ s_2 = 2;\ s_3 = 2;\ s_4 = 2;\ s_5 = 5.$

4. Adjuk meg az összes lehetséges prefix tulajdonságú bináris kódot, amely legfeljebb 3 jegy hosszúságú szavakban kódolja az

   a) A_1, A_2, A_3, A_4;

   b) $A_1, A_2, A_3, A_4, A_5, A_6$ jeleket.

5. Adjunk meg olyan kódrendszereket, amelyeknél az átlagos hossz minimuma elérhető bináris kód esetén.

6. Írjuk fel a következő karakterekhez tartozó bináris kódokat (ASCII és EBCDIC).

   a) A, **b**) Z, **c**) 0, **d**) 9, **e**) +, **f**) /.

7. Olvassuk el a következő hexadecimálisan (ASCII-ben) kódolt üzeneteket:

   a) 455A542049532054414E554C4E49204B454C4C;

   b) 4946582B593D395448454E593D3130.

8. Írjuk át a következő közleményeket hexadecimális ASCII kódba.

   a) ADATOK;

   b) Adatok;

   c) HA MAX < A[I] AKKOR MAX = A[I];

   d) Y = LOG(5 + ABS(SIN(2))).

4. ADATOK, ADATSTRUKTÚRÁK

4.1. Az adat fogalma

Az előző fejezetekben megismerkedtünk azokkal a karakterekkel, amelyek segítségével az információkat megadhatjuk és rögzíthetjük a számítógép számára.

Definíció. A rögzített, megjelenített információt adatnak nevezzük.

Az adat elnevezéssel a mindennapi életben is gyakran találkozunk:

– béradat, felvételi létszám adatok, statisztikai adatok, mérési adatok stb.

– adatfeldolgozás, adattárolás, adatátvitel stb.

Az információ és adat megjelölést általában azonos jelentésűnek fogadják el:

információ-feldolgozás – adatfeldolgozás

információhordozó – adathordozó stb.

szóhasználatban. Ne felejtsük el azonban, hogy az adat csak az információ hordozója, ami mindenki számára létező adat, de nem biztos, hogy információ is, mivel annak jelentése is van. Tehát az adat és információ nem azonos fogalmak.

A mindennapi életben általában

1. *numerikus* (számszerű mennyiségek) *adatok* pl. a dolgozó bére, a hallgató ösztöndíja, az osztály létszáma, tanulmányi átlaga, a tanuló születési évszáma, a beteg vérsüllyedése, az iskola telefonszáma stb. és

2. *alfabetikus,* illetve *alfanumerikus* adatok pl. a tanuló neve, a tanuló címe, a dolgozó szakképzettsége, iskolai végzettsége, családi állapota stb. fordulnak elő.

Ezek az adatok sok mindenben különböznek egymástól. Az egyik adat lehet mért, a másik lehet számított más adatokból, de lehet az adatot örökölni, vagy másoktól kapni, vagy megszerezni stb. Bennünket azonban itt elsősorban az érdekel, hogy miben hasonlóak. A felsorolásból kitűnik, hogy minden adat valaminek *vagy valakinek* a valamije *vagy valakije*. Pontosabban az adat az *egyed vagy objektum tulajdonsága, attribútuma* rögzített formában. *Egy egyednek több tulajdonsága* lehet, például a tanuló neve, születési helye, születési éve, lakhelye stb. és *egy tulajdonság több egyedhez* is tartozhat, pl. ugyanaz a születési év több tanulóhoz is tartozhat.

Mind az egyedek száma, mind a tulajdonságok száma elvileg végtelen lehet. Fontos feladat tehát egy *adatfeldolgozási feladat* megoldásával kapcsolatban

- a feldolgozásban résztvevő egyedek véges halmazának kiválasztása és
- a feldolgozáshoz szükséges tulajdonságok ugyancsak véges halmazának meghatározása.

Ezek figyelembevételével tehát az úgynevezett *adatfeldolgozási feladatoknál* mindig kialakítható egy táblázat, amelynek oszlopaiban az egyedek azonos tulajdonságaira vonatkozó adatok szerepelnek, soraiban pedig egy-egy egyed figyelembe vett összes tulajdonságainak az adatai. Például a tanulókra vonatkozó alábbi adatok:

Kód	Név	Szül. idő	Lakhely	Átlag	Elt. fogl.
100	Ács Ferenc	1999.01.22	Eger	4,1	tanár
101	Balla Béla	1999.05.07	Maklár	3,5	szabó
102	Csende Károly	1998.11.12	Eger	3,5	eladó

A fenti csoportosítás összefüggést, relációt fejez ki az adatelemek között. Ez a struktúra az alapja az ún. *relációs adatbáziskezelő rendszereknek* is, amelyekkel a későbbiek során foglalkozunk majd.

4.2. Elemi adattípusok

A táblázatunkban szereplő KÓD tulajdonsághoz tartozó adatok *egész típusúak,* az ÁTLAG-hoz tartozók *valós típusúak,* a SZÜLETÉSI IDŐ *dátum típusúak,* a NÉV, LAKHELY, ELTARTÓ FOGLALKOZÁSA tulajdonságokhoz tartozó adatok pedig *karakterlánc* típusúak. A típus a számítógépes feldolgozásban résztvevő adat legfontosabb jellemzője.

Definíció. Az adattípus megadása meghatározza a szóban forgó típus értékkészletét, a rajta végzett műveleteket és a tárban való ábrázolását.

Az adattípusok kétfélék lehetnek:

1. *elemi adattípusok,* amelyeknek felhasználói szempontból nincs belső szerkezetük,
2. *összetett adattípusok,* vagy adatstruktúrák, amelyek elemi adattípusokból épülnek fel.

Az adat típusát meghatározhatjuk a formája, vagy a programbeli leírása alapján. Adattípust definiálni is tudunk pl. szín, hónap, nap típus stb., de ezek alkalmazhatósága a programnyelvektől függ.

Mi itt csak azokat az adattípusokat említjük meg, amelyeket a legtöbb programnyelv kezelni tud.

Elemi adattípusok:

1. *Egész (integer)* típusú adat: Mindig csak egész szám lehet. Az egész típusú adatokkal minden numerikus művelet elvégezhető. Pl. $-5; 255; 0$
2. *Valós (real, double)* típusú adat: Mindig csak tört szám lehet. Ezekkel az adatokkal is a műveletek sokaságát végezhetjük, amelyek a matematikából már közismertek. Pl. $-12{,}5; 0{,}0125$. Meg kell jegyeznünk, hogy a táblázatkezelők követik a nemzeti beállításokat, de a programozási nyelvek általában nem, azaz a tizedespontot használnak tizedes vessző helyett.

3. *Logikai (logical, Boolean)* típusú adat: Mindig csak két érték lehet: igaz vagy hamis, amelyet jelölhetünk *true,* vagy *false* formában, vagy egyéb módon is. Legfontosabb műveletek: a negáció (not), a konjunkció (and), a diszjunkció (or) és az antivalencia (xor).

4. *Karakter (char)* típusú adat:

   a) Mindig csak egy karakter lehet. Ábrázolásuk ASCII kód esetén 1, UNICODE esetén 2 bájton a belső kódjuk alapján történik. Értelmezhető műveletek az összehasonlítás, kódmegadás, előző karakter, következő karakter képzése. Pl. *A*; 8; $; %.

   b) A másik értelmezés szerint egyes programnyelvek elemi adattípusnak tekintik a *karakterlánc* vagy *szöveg* típust is, mivel a műveletek ebben az esetben is az adatok teljes egészén kerülnek végrehajtásra és nem azok egy részén. Ilyen értelmezésben a karakter egy karakterből álló szöveg. Tehát nincs külön jelentősége a karakter típus **a)** változatának. Értelmezhető műveletek az összehasonlítás és az összefűzés (konkatenáció) művelete. Pl. EGER; ALGOL-60; NAGY IMRE.

A felsorolt elemi típusok tárban való ábrázolását a későbbiek során ismertetjük.

4.3. Összetett adattípusok, adatstruktúrák

Az elemi adattípusokból különböző adatstruktúrákat állíthatunk össze. Ha a táblázatunk oszlopait tekintjük, azokban azonos típusú adatok szerepelnek egy meghatározott sorrendben.

Definíció. Az azonos típusú adatokból álló véges adatsort **tömbnek** nevezzük.

A legegyszerűbb tömb a matematikából ismert vektor, pl.

$$(a_1, a_2, \ldots, a_n) \ ,$$

amelyet a számítástechnikában

$$A(1), A(2), \ldots, A(n),$$

vagy

$$A[1], A[2], \ldots, A[n]$$

módon jelölünk. Az A a tömb neve, minden elemre azonos, és bármely elemére az indexével hivatkozunk. $A(J)$ a J. tömbelem.

Az adatok sorozatának elrendezése, ha több azonos típusú tulajdonságot foglalunk egybe (pl. a hét öt munkanapján elért termelési eredmények dolgozónként), két dimenzióban történhet. Ilyen a matematikából ismert mátrix elrendezés.

$$\begin{matrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{matrix}$$

amelynek m sora és n oszlopa van. Az ilyen struktúrát kétdimenziós tömbnek nevezzük és az i. sor j. elemére az

$$A(i,j), \text{vagy } A[i,j]$$

jelöléssel hivatkozhatunk.

A tömbök dimenziószáma tovább is növelhető, de a gyakorlatban a legtöbb feladat megoldható egy és kétdimenziós tömbök használatával.

Itt is megemlíthetjük, hogy ha csak az egy karaktert tekintjük elemi típusnak, akkor a *karakterlánc* vagy *szöveg* is összetett típus és a tömbhöz hasonló struktúrával rendelkezik: olyan karaktersorozat, amelynek az elemszáma változhat.

Tekintsük most a táblázatunk egy sorát. Ezek az adatok különböző típusúak, de logikailag egybefűzi őket az, hogy egy tanulóra vonatkoznak.

Definíció. Tetszőleges típusú, logikailag összefüggő adatok egy egységgé való összekapcsolása adja a **rekordot**.

(A definíció nem zárja ki az egyenlő típusokból történő felépítést sem.) A rekordot is, mint a tömböt egy névvel azonosítjuk. A rekord részeinek (mezőinek) is neve van. Így lehetőség van a rekord elemeinek elérésére is, és a rekord együttes kezelésére is. Példánkban legyen a rekord neve: TANULOK az elemek (mezők) neve pedig KOD, NEV, SZIDO, LAKHELY, ATLAG, FOGL. Ilyen megnevezések mellett például a tanuló nevére a

TANULOK.NEV

lakhelyére a

TANULOK.LAKHELY

jelöléssel hivatkozhatunk.

Az adatfeldolgozási feladatoknál az is szükséges, hogy az egyedet, vagyis a hozzátartozó rekordot azonosítani tudjuk, illetve megkülönböztessük a

többitől. Ezt úgy érjük el, ha a rekordba beépítünk egy ilyen azonosítót. A NEV táblázatunkban lehetne ilyen azonosító, de a valóságban azonos nevű tanulók létezhetnek, így ezt elvetjük. Könnyen belátható, hogy azonosítónak egyedül a KOD választható, amelyről feltesszük, hogy sorszám és csak egyszer fordul elő.

Definíció. A record C nyelvben való megvalósulása a **struktúra**.

A következő példában .Net-ben, c#-ban mutatjuk be a struktúra használatát:

```
class Program
    {
        struct student
        {
            public string nev;
            public int kod;
            public double jegy;
        };
        static void Main(string[] args)
        {
            student[] s = new student[10];
            int i;
            Console.WriteLine("Adjuk meg a hallgatók adatait:");
            for (i = 0; i < s.Length; ++i)
            {
                s[i].kod = i + 1;
                Console.WriteLine("Azonosító {0}", s[i].kod);
                Console.Write("Név: ");
                s[i].nev = Console.ReadLine();
                Console.Write("Jegy: ");
                s[i].jegy = Convert.ToDouble(Console.ReadLine());
            }

            Console.WriteLine("A hallgatók adatati:");
            for (i = 0; i < s.Length; ++i)
            {
                Console.WriteLine("Azonosító: {0}", i + 1);
                Console.WriteLine("Név: {0}", s[i].nev);
                Console.WriteLine("Jegy: {0}", s[i].jegy);
            }
            Console.ReadLine();
        }
    }
```

Definíció. Az **azonosító** vagy **kulcs** az egyednek olyan tulajdonsága, vagy tulajdonságcsoportja, amely adott konkrét esetben csak egy egyednél fordul elő.

A népesség-nyilvántartásban az állampolgárt a személyi szám azonosítja, tehát ez a rekord kulcsa, amellyel az állampolgár adatait gyorsan „el tudják érni". Ha valamilyen külön tárolón pl. lemezen vagy szalagon, vagy háttértárolón valamennyi egyed rekordját összegyűjtjük, akkor a táblázathoz jutunk.

Definíció. Többnyire háttértárolón tárolt olyan adatszerkezetet, amelynek az elemei rekordok, **állománynak** vagy **fájlnak** (file) nevezzük.

Szemléletesen az adatok hierarchiája:

a *rekord* adatokból: Al,A2,...,

a *fájl* rekordokból: Rl, R2, ...,

az *adatbázis* fájlokból: F1, F2, ... épül fel.

5. ADATOK ÁBRÁZOLÁSA SZÁMÍTÓGÉPBEN

Mivel az adatok karakterekből épülnek fel és ezek kódjai 8 vagy 16 bites bitsorozatok, és a számítógépes tárak logikailag legkisebb egysége is 8 bitből álló bájt, ezért durva megfogalmazásban minden adat egy vagy több egymásutáni bájton elhelyezett bitsorozat. Az adatok típusával kapcsolatban már arra is utaltunk, hogy az adattípus megadása egyértelműen meghatározza a tárban való ábrázolásának a módját is. Az ismertetett típusok ábrázolása, tárolása alapvetően kétféle lehet. Az egyik a numerikus típusú (egész és valós) számok műveletvégzésre alkalmas formájú tárolása (gépi számábrázolás), a másik pedig a karakterlánc, szöveg kódolt ábrázolása. Az egész típusú szám úgy is interpetálható, hogy a törtpont (tizedespont, bináris pont) fixen a szám után következik. Ezt a törtpontot máshová is elképzelhetjük (fixálhatjuk), de külön nem jelölhetjük, a gép erről nem vesz tudomást. Ezt a számábrázolást fixpontos ábrázolásnak nevezzük, ellentétben a valós típusú számok ábrázolásával, ahol a törtpont helye nem fix, hanem „lebeghet" és lebegőpontos ábrázolásnak nevezzük.

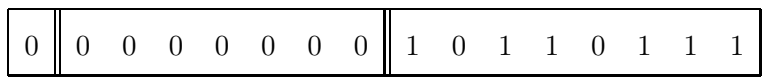
5.1. A fixpontos számábrázolás

A fixpontos számokat általában egy, két , négy vagy nyolc bájton ábrázoljuk, tehát 8, 16, 32 illetve 64 bit hosszú bitsorozaton. A fixpontot a bitsorozat után képzeljük el és csak az egész típusú számokkal foglalkozunk. Ez a megkötés semmilyen korlátozást nem jelent. A számok ábrázolását nem a karakterenkénti kódolásával valósítjuk meg, mert pl. 2 bájton csak kétjegyű számot ábrázolhatnánk, és még a számok előjelét sem vettük figyelembe. Ha a számokat kettes számrendszerbe konvertálva adjuk meg, akkor „gazdaságosabb" megoldást kapunk. Az előjel megadásához sem szükséges 1 bájt, ha erre a célra a bitsorozat első bitét jelöljük ki.

Definíció. A számok bináris ábrázolásánál az első bitet előjelbitnek nevezzük, amelynek értéke 1 ha a szám pozitív, és 0 ha a szám negatív.

Az előjelbit értéke a műveletvégzésben is részt vesz mint érték.

1. A pozitív számok ábrázolása tehát két bájton úgy történik, hogy az első bit 0, az ezt követő 15 biten pedig a szám kettes számrendszerbeli alakja következik. Pl. mivel $+183 = (10110111)2$, ezért ennek két bájton a



   bitsorozat felel meg. A két bájton ábrázolható legnagyobb pozitív szám: $2^{15} - 1 = 32767$, amelynek a

   $$0111111111111111 \text{ bitsorozat}, (7FFF)_{16} \text{ felel meg.}$$

2. A negatív számok ábrázolásánál 16 biten a szám abszolút értéke kettes számrendszerbeli alakjának kettes komplementere van, amely biztosítja, hogy az 1. biten 1 legyen. Vizsgáljuk meg mit jelent ez?

Mivel a fixpontos ábrázolást elsősorban műveletvégzés céljából hozzuk létre, ezért ha a negatív számoknál is csak egyszerűen beírnánk a szám kettes számrendszerbeli alakját, és a két számot összeadnánk, nem kapnánk megfelelő eredményt. Például:

+1		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
−1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

a $+1$ ábrázolásánál az utolsó biten van 1. A -1 ábrázolása olyan szám kell hogy legyen, amelyet bitenként hozzáadva a $+1$ bitjeihez 0-t kapunk. Ezt a követelményt a tiszta 1-es bitsorozat elégíti ki. Az összeadást elvégezve ugyanis a 17. helyiérték kivételével tiszta 0 sorozatot kapunk, ami megfelel a 0-nak, mert a 17. helyiértéknek már nincs hely, azt mondjuk hogy *kicsordul* a tárból.

A továbbiakban vizsgáljuk meg, hogyan történik a komplementer képzése és milyen jelentősége van.

Definíció. Egy p alapú számrendszerben egy szám $(p-1)$-es komplementere az a szám, amely minden helyiértéken $(p-1)$-re egészíti ki a számot.

Pl. 354 kilences komplementere 645,
$(10110111)_2$ egyes komplementere $(01001000)_2$.

Vegyük észre, hogy ez az eljárás a kettes számrendszerben egy bitenkénti negáció, vagyis nagyon egyszerű eljárás:

1 helyett 0, 0 helyett 1

írandó minden biten.

Definíció. A p alapú számrendszerben, a p-s komplementert a $(p-1)$-es komplementerből 1 hozzáadásával kapjuk.

Pl. 354 tizes komplementere $645 + 1 = 646$,
$(10110111)_2$ kettes komplementere $(01001001)_2$.
Mivel +183 ábrázolása

0000000010110111

ennek 1-es komplementere

1|111111101001000

\+ 1

2-es komplementere, vagyis a –183 ábrázolása

1|111111101001001

MEGJEGYZÉS. A kettes komplementerképzést egy lépésben úgy is megoldhatjuk, hogy a bitsorozatban a végéről előre haladva az első 1-esig bezárólag

változatlanul leírjuk a biteket, majd innen tovább minden bitet ellenkezőre változtatunk.

Például:

$$\begin{array}{lll} 000011011000 & 1000 & \\ & \mid \longleftarrow & \\ 111100100111 & 1000 & \text{kettes komplementere} \end{array}$$

A negatív számok tartománya 2 bájton az

$$1000000000000000 \qquad -32768$$

negatív számig terjedhet, amelynek 1-es komplementere

$$(0111111111111111)_2 \ ,$$

ami 32767, 2-es komplementere 1-gyel nagyobb, vagyis 32768.

Összefoglalva az N egész szám két bájton

$$-32768 > N > 32767$$

lehet.

Az hogy a negatív számokat komplementereikkel ábrázoljuk, szükségtelenné teszi a kivonást mint műveletet a számítógépnél, mert az

$$a - 6 = a + (-6)$$

módon kezelhető.

PÉLDA. $5 - 18 = 5 + (-18)$:

+18	0	0...010010
-18	1	1...101110

5	0	0...000101
-18	1	1...101110
	1	1...110011

Az eredmény negatív szám, hogy mennyi azt a kettes komplementere,

$$0|0...001101$$

adja, ami 13 és valóban $5 - 18 = -13$.

A fixpontos ábrázolásnak azonban előnyei mellett van egy hátránya is, nevezetesen, hogy két szám összege nem számítható ki minden esetben, mert az összeadás eredménye sem lehet nagyobb pl. 32767-től, különben *túlcsordulás* következik be. Például

	0	110...01	> 0 szám
+	0	110...10	> 0 szám
	1	000...11	negatív szám lesz

ami nyilván nem igaz. Az ilyen túlcsordulásra a felhasználónak ügyelni kell.

A következő táblázat a C# egész típusú változóit mutatja be.

Típus	**Értékkészlet**	**Méret**
sbyte	-128 ... 127	Signed 8-bit
byte	0 ... 255	Unsigned 8-bit
char	U+0000 ... U+ffff	Unicode 16-bit
short	-32 768 ... 32 767	Signed 16-bit
ushort	0 ... 65 535	Unsigned 16-bit
int	-2 147 483 648 ... 2 147 483 647	Signed 32-bit
uint	0 ... 4 294 967 295	Unsigned 32-bit
long	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807	Signed 64-bit
ulong	0 ... 18 446 744 073 709 551 615	Unsigned 64-bit

Érdemes a programozásnál figyelembe vennünk az egész számok ábrázolási tartományát.

Általánosan, ha *előjel nélküli* (unsigned) egész számunk van n biten, akkor a tartomány

$$0 \dots 2^n - 1$$

Ha *előjeles* (signed) egész számunk van n biten, akkor a tartomány

$$-2^{n-1} \dots 2^{n-1} - 1$$

Például 1 bájtos számok esetén:

	decimálisan	binárisan	hexadecimálisan
előjel nélküli	$0 \dots 2^8 - 1 = 0 \dots 255$	$00000000 \dots 11111111$	$00 \dots FF$
előjeles	$-2^7 \dots 2^7 - 1 = -128 \dots 127$	$10000000 \dots 01111111$	$80 \dots 7F$

5.2. Lebegőpontos számábrázolás

A lebegőpontos *(floating point)* számábrázolást a valós számok reprezentálására használjuk. Tulajdonképpen egy olyan eljárás, amely a valós számokat – *mivel a számítógépeink csak véges hosszúságú számjegysorozatokkal képesek dolgozni* - egy közelítő értékkel helyettesítve racionális számokká alakítja. Míg fixpontos ábrázolásnál, mint arról az előzőekben eset szó, a kettedes pont *"fixen"* marad, addig lebegőpontosnál ez *"lebeghet"*.

5.2.1. FLOPS

A processzorok műveletvégzési sebességének mértékegysége a FLOPS *(**FL**oating point **OP**erations **P**er **S**econd)*, mellyel a másodpercenként elvégezhető lebegőpontos műveletek számát jellemezhetjük. Az első általános célú számítógép az ENIAC volt, amelynek tervezéséhez 1943-ban kezdtek hozzá, elsősorban hadászati célokra *(hidrogénbombához szükséges számítások elvégzése)* készült, számítási teljesítménye 5 Kiloflop/s volt *(5000 művelet másodpercenként)*. Hogy érzékeltessük a fejlődést, a jelenlegi (2013) leggyorsabb szuperszámítógép a Titan *(560 640 db processzor, 261 632 db NVIDIA K20x mag)*, melynek teljesítménye 17,59 Petaflop/s.

Megjegyzés: Amennyiben érdeklődik a szuperszámítógépek iránt és szeretne több információhoz jutni róluk. Érdemes felkeresni a *www.top500.org* oldalt.

5.2.2. Lebegőpontos szám

A lebegőpontos ábrázolási mód értelmében minden számot szorzat alakban adunk meg. Egy lebegőpontos szám *(floating point number)* ebben az esetben a következő alakban írható fel:

$N = m \cdot A^k$, *ahol* N a lebegőpontos szám, m: a mantissza, A: az alkalmazott számrendszer alapja, k: a kitevő *(karakterisztika)*.
Megjegyzés: A mantissza és a kitevő is lehet negatív szám.

5.2.3. Normalizálás

Mielőtt továbbmennénk, fontosnak tartjuk felhívni a figyelmet, hogy a matematikai értelemben illetve a számítástechnikai értelemben vett normálalak között különbséget teszünk.

Egészre normalizálás

Matematikai értelemben a mantissza (m) értéke: $1 \leqslant m < A$, ahol $m \neq 0$ és A a számrendszer alapja. Ebben az esetben egészre normalizálunk, tehát a bináris számrendszerben vett mantissza értéke egy $1 \leqslant m < 2$ közötti szám lesz.

5.1. példa. Tízes számrendszerbeli szám esetében:

$$13{,}625 = 1{,}3625 \cdot 10^1$$

5.2. példa. Kettes számrendszerbeli szám esetében:

$$1001{,}1001_2 = 1{,}0011001_2 \cdot 2^3$$

Megfigyelhető, hogy az egészek helyén mindig 1-*es* áll, ezért ennek tárolása szükségtelen. Ezt ***implicit bit***-nek hívjuk. Így a tárolt mantissza *(m)* értéke:

m: 0011001

Törtre normalizálás

Törtre normalizálás esetében a bináris pontot addig toljuk el, amíg a mantissza értéke $1/2 \leqslant m < 1$ között nem lesz.

5.3. példa. Tízes számrendszerbeli szám esetében:

$$13{,}625 = 0{,}13625 \cdot 10^2$$

5.4. példa. Kettes számrendszerbeli szám esetében:

$$1001{,}1001_2 = 0{,}10011001_2 \cdot 2^4$$

Ebben az esetben a 2^{-1} helyi értéken lévő bit mindig 1-*es* értékű, ezért ennek eltárolása is felesleges. A tárolt mantissza értéke természetesen megegyezik az előzővel:

m: 0011001

Megjegyzés: A későbbi műveletvégzés előtt mindkét esetben a nem tárolt *(implicit)* biteket vissza kell helyezni, ellenkező esetben hibás eredményt fogunk kapni.

5.2.4. Eltolt karakterisztika

Az ábrázolás során a számítógépnek nem kell tárolnia a számrendszer alapját, hiszen minden számítás azonos alapú számrendszerben ábrázolt számokkal történik. A számítógép meghatározott számú tárolóhelyet biztosít mind a mantissza, mind pedig a kitevő számára, ezeket már tárolni kell.
A mantissza, mint ahogy az előzőekben már volt róla szó, egy valódi tört, melynek ábrázolása történhet kettes komplemens alapján is.

A kitevő ábrázolása azonban legtöbbször feszített módban *(többletes kód segítségével)* történik. Ezt a megoldást *eltolt* vagy *ofszet* karakterisztikának is hívják. Ekkor k-t a lebegőpontos szám karakterisztikájának nevezzük. A feszített módú ábrázolás értelmében a karakterisztikát eltoljuk a pozitív számok tartományába, így a negatív kitevő is ábrázolhatóvá válik anélkül, hogy előjelét külön kellene ábrázolni.

Az eltolás mértékére *(d)* két megoldás használatos:
$d = 2^{n-1} - 1$ *pl. Egészre normalizálás esetén*

$d = 2^{n-1}$ *pl. Törtre normalizálás esetén*
ahol n a karakterisztika ábrázolására szánt bitek száma.
Így az eltolt karakterisztika *(k)* a

$$k{=}e + d$$

összefüggéssel számítható ki *(ahol e az ábrázolni kívánt kitevő).*

Amennyiben a karakterisztika ábrázolására *8* bit áll rendelkezésünkre, az eltolás mértéke:

$$d = 128 - 1 = 127$$

A feszített módban ábrázolandó kitevő az ***4.*** példa esetében:

$$k = 3 + 127 = 130$$

Bináris alakban:

$$k = 10000010_2$$

5.2.5. Lebegőpontos számok reprezentálása

További vizsgálatokat végezve az N lebegőpontos számon megfigyelhető, hogy van legnagyobb eleme (N_∞). Ez abban az esetben áll elő, amikor a mantissza és karakterisztika is a legnagyobb értéket veszi fel, míg a legkisebb pozitív számot, N_0 -t úgy kapjuk, hogy a legkisebb normalizált mantisszát és a karakterisztikát vesszük. Az előzőekből következik, hogy a legkisebb ábrázolható szám a $-N_\infty$, és a legnagyobb negatív szám pedig a $-N_0$. Mindezekből megállapítható, hogy az N tulajdonképpen egy 0-ra szimmetrikus korlátos halmaz, amely része a Q-nak, és csak véges sok eleme van.

Szemléltetéshez nézzük a következő példát. Ábrázoljuk az N lebegőpontos számot 1 byte-on úgy, hogy az első bit jelölje az előjelet, a következő 3 bit a karakterisztikát és a maradék 4 bit pedig a mantisszát. Normalizált ábrázolási mód értelmében a mantissza *(első értékén csak egyes állhat)* 1000_2 és

1111_2 között, a karakterisztika *(eltolt)* pedig 000_2 (-3) és 111_2 $(+3)$ között vehet fel értékeket.

A legnagyobb ábrázolható számot (N_∞) abban az esetben kapjuk meg, amikor a karakterisztika az 111_2, a mantissza pedig az 1111_2 értéket veszi fel.

$$N_\infty = \left(1\frac{1}{2} + 1\frac{1}{4} + 1\frac{1}{8} + 1\frac{1}{16}\right) \cdot 2^3 = 7{,}5$$

A legkisebb pozitív szám (N_0) pedig akkor áll elő, amikor a mantissza értéke 1000_2 a karakterisztika értéke pedig 000_2 (-3)

$$N_0 = 1\frac{1}{2}2^{-3} = \frac{1}{16}$$

Megjegyzés: Az egy byte-on ábrázolható lebegőpontos számok darabszáma a 8 különböző mantisszához hozzávéve a 7 lehetséges karakterisztikát a negatív számokkal és a nullával együtt $2 \cdot 8 \cdot 7 + 1 = 113$.

		1	2	3	4	5	6	7	8
	m	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
		0,50000	0,56250	0,62500	0,68750	0,75000	0,81250	0,87500	0,93750
$k^{(-)}$	-3	0,06250000	0,07031250	0,07812500	0,08593750	0,09375000	0,10156250	0,10937500	0,11718750
	-2	0,12500000	0,14062500	0,15625000	0,17187500	0,18750000	0,20312500	0,21875000	0,23437500
	-1	0,25000000	0,28125000	0,31250000	0,34375000	0,37500000	0,40625000	0,43750000	0,46875000
	0	0,50000000	0,56250000	0,62500000	0,68750000	0,75000000	0,81250000	0,87500000	0,93750000
	1	1,00000000	1,12500000	1,25000000	1,37500000	1,50000000	1,62500000	1,75000000	1,87500000
	2	2,00000000	2,25000000	2,50000000	2,75000000	3,00000000	3,25000000	3,50000000	3,75000000
$k^{(+)}$	3	4,00000000	4,50000000	5,00000000	5,50000000	6,00000000	6,50000000	7,00000000	7,50000000

5.1. ábra. Lebegőpontos számok reprezentálása

5.2.6. ANSI/IEEE 754-es szabvány

A lebegőpontos számok ábrázolásának egységesítését 1977-ben kezdték meg. Cél a különböző architektúrák között az adatszintű kompatibilitás meg-

teremtése. Minden architektúrából összegyűjtötték a legjobb megoldásokat, az első szabvány, amelyet a nagy processzorgyártók (Intel, AMD, Motorola, stb. . .) is elfogadtak, 1985-re született meg. Az IEEE 754-es szabvány a lebegőpontos művelet végrehajtásához kétfajta pontosságot definiált 2-es és 10-es számrendszerben:
- egyszeres pontosság (single precision): 32 bit,
- dupla pontosság (double precision): 64 bit.

A jelenlegi verzió, az **IEEE 754-2008**, amely tartalmazza az eredeti IEEE 754-és az IEEE 854-1987-es *(alaptól független lebegőpontos ábrázolás)* szabványokat, öt alapvető formátumot definiál.
Három bináris lebegőpontos alapvető formátum *(kódolt 32, 64 vagy 128 bit)* és két decimális lebegőpontos alapvető formátum *(kódolt 64 vagy 128 bit).*

Név	**Ismert neve**	**Alap**	**Számjegyek**	**k min.**	**k max.**	**Decimális számjegyek**	**Dec. k max.**
binary16	Feles pontosság	2	10+1	-14	+15	3,31	4,51
binary32	Egyszeres pontosság	2	23+1	-126	+127	7,22	38,23
binary64	Dupla pontosság	2	52+1	-1022	+1023	15,95	307,95
binary128	Négyszeres pontosság	2	112+1	-16382	+16383	34,02	4931,77
decimal32		10	7	-95	+ 96	7	96
decimal64		10	16	-383	+ 384	16	384
decimal28		10	34	-6143	+6143	34	6144

Mielőtt egy konkrét példát megnéznénk, nézzük meg, hogyan épül fel egy bináris szám 32 biten.

Az első bit az előjel bit, amely az ábrázolandó valós szám előjele (e), az ezt követő 8 biten ábrázoljuk az eltolt karakterisztikát (k), végül az utolsó 23 biten pedig a mantisszát (m).

Fontos megjegyezni, hogy az előjel bit értéke 0, ha a szám pozitív és 1, ha negatív. Továbbá azt, hogy a mantisszában levő fixpontos szám egészre

normalizáltan értendő, tehát az eltolás mértéke 127. Az előjel esetében az $e = 0$ és az e = 255 különleges esetekre vannak fenntartva, mint a ± 0, $\pm\infty$ és a NaN *(Not a Number).*

A következő táblázatban összefoglalva megtalálhatóak az egyes pontosságokhoz tartozó előjel, karakterisztika és mantissza bitértékek.

Pontosság	**Előjel**	**Karakterisztika**	**Mantissza**
egyszeres (32 bit)	1 bit	8 bit	23 bit
dupla (64 bit)	1 bit	11 bit	52 bit
négyszeres (128 bit)	1 bit	15 bit	112 bit

Megjegyzés: Dupla pontosság esetében az ábrázolható számtartomány:

$$4,19 \cdot 10^{-307} < |N| < 1,67 \cdot 10^{308}$$

között van.

5.5. példa. Ábrázoljuk lebegőpontosan 32 biten a $38{,}29_{10}$ számot! *Egészrész:*

$$38_{10} = 100110_2$$

Törtrész:

$$0,29_{10} = 0{,}01001010001111010111000 0_2$$

Kettes számrendszerbeli alak:

$$38{,}29_{10} = 100110{,}01001010001111010111000 0_2$$

Egészre normalizált alak:

$$N = 1{,}00110010010100011110101110000_2 \cdot 2^5$$

Mantissza (23 bit):

$$m = 00110010010100011110101_2$$

Karakterisztika (8 bit):

$$k = 5 + 127 = 132_{10} = 10000100_2$$

Előjelbit (1 bit):

$$e = 0$$

Táblázatba foglalva:

0	**10000100**	**00110010010100011110101**
Előjel	Eltolt karakterisztika	Mantissza

Hexadecimális alak:

0100	0010	0001	1001	0010	1000	1111	0101
4	**2**	**1**	**9**	**2**	**8**	**F**	**5**

$$N_{16} : 421928F5$$

Megjegyzés: Természetesen az N_{16} nem azonos az N szám tizenhatos számrendszerbeli értékével, ezért nem használtunk egyenlőségjelet.

5.6. példa. Melyik tízes számrendszerbeli szám hexadecimális alakja a következő $N_{16} : C16A0000$?
Bináris alak:

1100	0001	0110	1010	0000	0000	0000	0000

Csoportosítva:

Előjel	Eltolt karakterisztika	Mantissza
1	10000010	11010100000000000000000

Előjel bit:

$$e = 1$$

Tehát negatív számról van szó.
Karakterisztika:

$$k = 10000010_2$$

$$k = 10000010_2 = 130$$

$$k = 130 - 127 = 3$$

Kiolvasható, hogy a kitevő értéke **3**.
Mantissza:

$$m = 11010100000000000000000_2$$

Megjegyzés: He felejtsük el, hogy az implicit bit-et nem tároljuk!
Implicit bitet visszatéve:

$$1{,}11010100000000000000000_2$$

A szám normálalakja:

$$N = -1{,}11010100000000000000000_2 \cdot 2^3$$

Kettes számrendszerbeli alak:

$$-1110{,}101_2$$

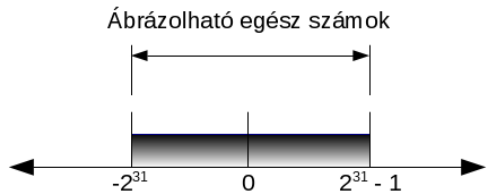
Tízes számrendszerbeli alak:

$$-14{,}625$$

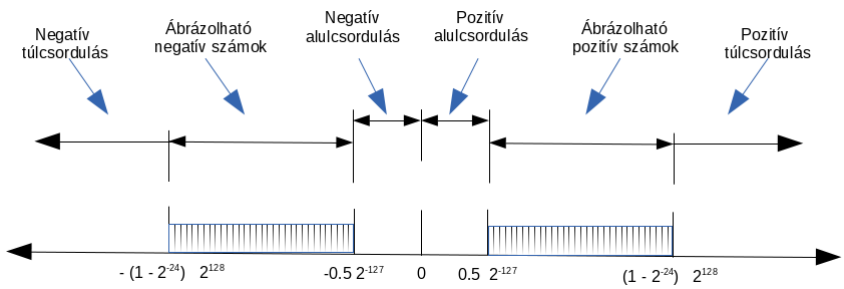
5.2.7. Alulcsordulás, túlcsordulás

Az eddigi példákból is kiderült, hogy a valós számokat sok esetben nem tudjuk pontosan ábrázolni. A 38,29 valós számot 128 bites pontosság mellett sem tudjuk pontosan ábrázolni. Továbbá nem tudjuk ábrázolni a túl nagy, illetve a túl kis számokat sem. 32 biten a legkisebb ábrázolható szám a 2^{-126} $\approx$ 1,2 x 10^{-38}, amelynél kisebb szám esetében alulcsordulás *(underflow)* hibaüzenetet kapunk.

Túlcsordulás *(overflow)* hibaüzenetet kapunk, amennyiben $(2–2^{-23})$ x 2^{127} $\approx$ 3,4 x 10^{38}nál nagyobb számot szeretnénk ábrázolni.



5.2. ábra. a, ábrázolható egész számok



5.3. ábra. b, ábrázolható egész számok

5.2.8. Speciális értékek

Előjeles nulla

Az IEEE 754 szabvány a nullát előjelesen kezeli. Ez azt jelenti, hogy létezik pozitív nulla "+0" és negatív nulla "-0" is. Ezek műveletvégzéskor sok esetben egyenlően viselkednek. Előfordul néhány eset azonban, amikor eltérnek. Például, a "1/-0" a negatív végtelen (pontosan), míg az "1/+0" a pozitív végtelent jelenti (pontosan). A "+0" és "-0" közötti különbség leginkább összetett műveletek esetében jelentkezik.

Megjegyzés: A legtöbb aritmetikai művelet, amelynek eredménye 0, a "+0" értéket kapja.

Végtelen ábrázolása

A szabvány ugyanúgy, mint az előjeles nulla esetében, a végtelen ábrázolására is két értéket különböztet meg ($+\infty$ és $-\infty$). 32 bites bináris ábrázolás mellett a mínusz végtelen hexadecimális alakja FF800000, a pozitív végtelen alakja pedig a 7F800000. A végtelent úgy kezeli, mint egy nagyon nagy számot, így értelmezett a végtelennel való műveletvégzés is, tehát a következő műveletek elvégezhetőek.

5.7. példa. Műveletek a végtelennel $(+\infty) + x = (+\infty)$
$(+\infty) \cdot x = (+\infty)$
$(+\infty)/x = (+\infty)$
$x/(+\infty) = +0$,
ahol $(0 < x < \infty)$

5.8. példa. Konkrét értékekkel $(+\infty) + (+7) = (+\infty)$
$(+\infty) \cdot (-2) = (-\infty)$
$(+\infty) \cdot 0 = NaN$

NaN – Not a Number. Már az előző példában is találkozhattunk a NAN ("nem szám") értékkel, amely egy speciális érték, amelyet érvénytelen műveletek, mint a $0/0, \infty \cdot 0$, vagy $sqrt(-1)$ visszatérési értékére használnak.
IEEE 754/1985 szabvány speciális lebegőpontos számformái A következő táblázatban megtalálhatóak az IEEE 754/1985 szabvány speciális

lebegőpontos számformái.

Megnevezés	**Előjel**	**Kitevőrész**	**Mantisszarész**
Normalizált érték	s	Tetszőleges kitevő	Tetszőleges számérték
Denormalizált érték	s	000...000	Nem nulla számérték
Nulla	s	000...000	0
Negatív végtelen	1	111...111	0
Pozitív végtelen	0	111...111	0
Jelző 'nem szám' (*NaN*)	x	111...111	Nem nulla számérték
Egyszerű 'nem szám' (*NaN*)	x	111...111	Nem nulla számérték

5.2.9. Kerekítés

Gyakran előfordul, hogy a pontos eredményt még 128-biten sem tudjuk ábrázolni, ezért a kerekítés korrekt elvégzését követeli meg a szabvány. A kerekítés alapvetően négyféleképpen történhet:

1. **Csonkolás** *(Round toward zero)* Levágjuk az n-ik számjegy után az összes számjegyet.

2. **Bankár kerekítés** *(Round to the Nearest Value -tie to even-)* Az $n+$ $+$ 1-ik számjegy alapján kerekítünk: ha ez a számjegy nagyobb, mint 5, felfelé kerekítünk, ha kisebb, lefelé; ha 5, akkor az n-ik számjegy alapján döntünk *(például felfelé kerekítünk ha az páros, és lefelé, ha páratlan).*

3. **Kerekítés a pozitív végtelenhez** *(Round toward Positive Infinity)* Ebben az esetben egyszerűen felfelé kerekítünk.

4. **Kerekítés a negatív végtelenhez** *(Round toward Negative Infinity)* Ebben az esetben egyszerűen lefelé kerekítünk.

5.2.10. Feladatok

1. Ábrázolja lebegőpontosan (IEEE 754 -32 bit) az alábbi számokat!
   a, 532,18 b, 0,0035 c, -0,0123 d, 2013,1010

2. Alakítsa át tízes számrendszerbe az alábbi hexadecimális alakban megadott számokat!
   a, C2C90000 b, 42F02000 c, 3DCCCCCD d, 7F800000

Megjegyzés: A feladatok megoldásait az alábbi oldalon ellenőrizheti: http://babbage.cs.qc.cuny.edu/IEEE-754/

5.3. Műveletek lebegőpontos számokkal

Ebben a fejezetben a lebegőpontos számokkal történő műveletvégzési lehet?ségeket nézzük át. Elsődlegesen a következő: $+, -, *, /$ aritmetikai operátorokkal foglalkozunk.

Természetesen további műveletek is elvégezhetők lebegőpontos számokkal *(összehasonlítás, gyökvonás, ...)* de ezekkel majd a későbbi tanulmányaik alatt (*pl. Numerikus módszerek*) ismerkedhetnek meg részletesebben.

Lebegőpontos számokkal történő műveletvégzés esetén mindenek előtt figyelembe kell vennünk, hogy az eredmény általában nem ábrázolható pontosan. Két 32 bites lebegőpontos szám szorzata egy 64 bites szám. Ezekben az esetekben az helyesen kerekített értékét (adott szabvány által definiált) tároljuk.

5.3.1. Relatív kerekítési hiba

Legyen x és y két lebegőpontos szám, továbbá $\oplus, \ominus, \otimes, \oslash$, rendre a következő műveleteknek $+, -, \cdot, /$ számítógépeken értelmezett implementációik. Az $x \oplus y$ tehát nem más, mint az $x + y$ egy közelítő értéke.
További műveletek esetében:

$$x \oplus y = kerekites\,(x + y)\,,$$

$$x \ominus y = kerekites\,(x - y)\,,$$

$$x \otimes y = kerekites\,(x \cdot y)\,,$$

$$x \oslash y = kerekites\,(x/y)\,.$$

Relatív kerekítési hiba, amennyiben $x + y$ egy normalizált szám:

$$x \oplus y = (x + y)(1 + \delta),$$

$$|\delta| \leqslant \varepsilon.$$

Megjegyzés: Hasonlóan értelmezhető a többi műveletre is.

5.3.2. Egyszerű algoritmusok aritmetikai műveletekre

A műveletek elvégzése előtt természetesen feltételezzük, hogy az operandusok már IEEE 754-es lebegőpontos formátumban vannak.

Összeadás

$$x + y = (x_m \cdot 2^{xe}) + (y_m \cdot 2^{ye})$$

1. lépés Bináris pont igazítása:

1. Megvizsgáljuk, hogy melyik szám kitevője a nagyobb (xe, ye). Ez lesz kezdetben a kitevő értéke.
2. Kiszámoljuk a két kitevő különbségét, pl. $xe - ye$.
3. Ha $ye > xe$, akkor az x_m-et eltoljuk jobbra, $(x_m \cdot 2^{xe-ye})$.
4. Ha $xe > ye$, akkor pedig az y_m-et toljuk el jobbra, $(y_m \cdot 2^{ye-xe})$.

2. lépés Kiszámítjuk az igazított mantisszák összegét:

$$x_m \cdot 2^{xe-ye} + y_m \; vagy \; x_m + y_m \cdot 2^{ye-xe}$$

3. lépés Amennyiben szükséges, normalizáljuk az eredményt.
A normalizálás lépései:

1. Balra eltoljuk az eredményt, csökkentjük az eredmény kitevőjét *(pl. ha az eredmény 0,001xx . . .)*,

vagy

1. Jobbra eltoljuk az eredményt, növeljük az eredmény kitevőjének értékét *(pl. ha az eredmény 10,1xx ...)*

Ezt addig folytatjuk, amíg a legmagasabb helyi értékű jegy 1-es nem lesz.

4. lépés Ellenőrizzük az eredmény kitevőjét:

1. Amennyiben nagyobb, mint a maximálisan megengedett, kitevő túlcsordulás.

2. Amennyiben kisebb, mint a minimálisan megengedett, kitevő alulcsordulás.

5. lépés Ha az eredmény mantisszája *0*, a kitevőt is nullára kell állítanunk.

5.9. példa. Összeadás

$$x = 2345{,}125$$

Lebegőpontos alakban: *0 10001010 00100101001001000000000*

$$y = 0{,}75$$

Lebegőpontos alakban: *0 01111110 11000000000000000000000.*

1. lépés Bináris pont igazítása: Mivel $xe > ye$ kezdetben az eredmény kitevője tehát:

$$xe - ye = 10001010_2 - 01111110_2 = 00001100_2 = 12$$

Eltoljuk y_m-et 12 pozíciót jobbra:

$$y_m \cdot 2^{ye-xe} = y_m \cdot 2^{-12} = 0{,}000000000001100000000000$$

2. lépés Összeadjuk a mantisszákat:

$$x_m \cdot + y_m 2^{-12}$$

	1,00100101001001000000000
+	0,00000000000110000000000
	1,00100101001111000000000

3. lépés

1. Megvizsgáljuk, hogy az eredmény normalizált-e? *Igen.*

4. lépés

1. Megvizsgáljuk, hogy a kitevő túlcsordult-e? *Nem.*

2. Megvizsgáljuk, hogy a kitevő alulcsordult-e? *Nem.*

5. lépés

1. Megvizsgáljuk, hogy az eredmény nulla-e? *Nem.*

Az összeg:	0	10001010	00100101001111000000000

Kivonás

$$x - y = (x_m \cdot 2^{xe}) - (y_m \cdot 2^{ye})$$

1. lépés Bináris pont igazítása:

1. Megvizsgáljuk, hogy melyik szám kitevője a nagyobb (xe, ye). Ez lesz kezdetben a kitevő értéke.
2. Kiszámoljuk a két kitevő különbségét, pl. $xe - ye$.
3. Ha $ye > xe$, akkor az x_m-et eltoljuk jobbra, $(x_m \cdot 2^{xe-ye})$,
4. Ha $xe > ye$, akkor pedig az y_m-et toljuk el jobbra, $(y_m \cdot 2^{ye-xe})$.

2. lépés Kiszámítjuk az igazított mantisszák különbségét:

$$x_m \cdot 2^{xe-ye} + y_m \; vagy \; x_m + y_m \cdot 2^{ye-xe}$$

3. lépés Amennyiben szükséges, normalizáljuk az eredményt. A normalizálás lépései:

1. Balra eltoljuk az eredményt, csökkentjük az eredmény kitevőjét *(pl. ha az eredmény 0,001xx . . .)*,
   vagy
2. Jobbra eltoljuk az eredményt, növeljük az eredmény kitevőjének értékét *(pl. ha az eredmény 10,1xx ...)*

Ezt addig folytatjuk, amíg a legmagasabb helyi értékű jegy 1-es nem lesz.

4. lépés Ellenőrizzük az eredmény kitevőjét:

1. Amennyiben nagyobb, mint a maximálisan megengedett, kitevő túlcsordulás.

2. Amennyiben kisebb, mint a minimálisan megengedett, kitevő alulcsordulás.

5. lépés Ha az eredmény mantisszája *0*, a kitevőt is nullára kell állítanunk.

Szorzás

$$x \cdot y = (-1)^{xs}(x_m \cdot 2^{xe}) \cdot (-1)^{ys}(y_m \cdot 2^{ye})$$

1. lépés Amennyiben valamelyik operandus értéke nulla, az eredményt nullára állítjuk.

2. lépés Az eredmény előjelének meghatározása:

$$xs\ XOR\ ys$$

3. lépés Mantisszák szorzása:

$$x_m \cdot y_m$$

Kerekítsük az eredményt a mantisszáknak fenntartott bitekre *(Az egyszerűség kedvéért végezzünk csonkolást).*

4. lépés Kitevő kiszámítása:
nem szabályos (x) + nem szabályos (y) – eltérés

5. lépés Amennyiben szükséges, normalizáljuk az eredményt.
A normalizálás lépései:

1. Balra eltoljuk az eredményt, csökkentjük az eredmény kitevőjét *(pl. ha az eredmény 0,001xx . . .)*,
   vagy

2. Jobbra eltoljuk az eredményt, növeljük az eredmény kitevőjének értékét *(pl. ha az eredmény 10,1xx ...)*

Ezt addig folytatjuk, amíg a legmagasabb helyi értékű jegy 1-es nem lesz.

6. lépés Ellenőrizzük az eredmény kitevőjét:

1. Amennyiben nagyobb, mint a maximálisan megengedett, kitevő túlcsordulás.

2. Amennyiben kisebb, mint a minimálisan megengedett, kitevő alulcsordulás.

5.10. példa. Szorzás

$$x = -18$$

Lebegőpontos alakban: *1 10000011 00100000000000000000000*

$$y = 9{,}5$$

Lebegőpontos alakban: *0 10000010 00110000000000000000000*
1. lépés Valamelyik operandus nulla? *Nem.*

2. lépés Az előjel meghatározása:

$$s = xs\ XOR\ ys = 1\ XOR\ 0 = 1$$

3. lépés Mantisszák szorzása: Két 24 bites mantissza szorzata 48 biten keletkezik.

0101011000000. . . .000000

Kerekítenünk kell a mantisszák szorzatát 24 bitre *(csonkolás)*.
Ekkor :

010101100000000000000000

4. lépés Kitevő kiszámítása:

$$xe + ye - 127 = 10000011_2 + 10000010_2 - 0111111_2 = 10000110_2$$

5. lépés Megvizsgáljuk, hogy az eredmény normalizált-e? *Igen.*

6. lépés
Megvizsgáljuk, hogy a kitevő túlcsordult-e? *Nem.*
Megvizsgáljuk, hogy a kitevő alulcsordult-e? *Nem.*

A szorzat:	1	10000110	01010101100000000000000

Osztás

$$x/y = (-1)^{xs}(x_m \cdot 2^{xe})/(-1)^{ys}(y_m \cdot 2^{ye})$$

1. lépés Ha y értéke nulla, akkor az eredmény *"Infinity"*. Ha mindkettő nulla, az eredmény *NaN*.
2. lépés Az eredmény előjelének meghatározása:

$$xs\ XOR\ ys$$

3. lépés Mantisszák szorzása:

$$xm \cdot ym$$

Kerekítsük az eredményt a mantisszáknak fenntartott bitekre *(Az egyszerűség kedvéért végezzünk csonkolást).*

4. lépés Kitevő kiszámítása:
nem szabályos (x) - nem szabályos (y) + eltérés
5. lépés Amennyiben szükséges, normalizáljuk az eredményt.

Normalizálás lépései:

1. Balra eltoljuk az eredményt, csökkentjük az eredmény kitevőjét *(pl.*

ha az eredmény 0,001xx . . .),
vagy

2. Jobbra eltoljuk az eredményt, növeljük az eredmény kitevőjének értékét *(pl. ha az eredmény 10,1xx ...)*

Ezt addig folytatjuk, amíg a legmagasabb helyi értékű jegy 1-es nem lesz.

6. lépés Ellenőrizzük az eredmény kitevőjét:

1. Amennyiben nagyobb, mint a maximálisan megengedett, kitevő túlcsordulás.

2. Amennyiben kisebb, mint a minimálisan megengedett, kitevő alulcsordulás.

5.3.3. Feladatok

Végezze el a következő műveleteket:
a, $1234{,}56_{10}$ (+,−, ∗, /) $191{,}31_{10}$
b, 201_{10} (+,−,∗, /) $3200{,}99_{10}$

5.4. Decimális számok ábrázolása

A BCD *(Binary Coded Decimal)* olyan kódolási eljárás, amelyet decimális számok *(számjegyek)* tárolására használunk. Ismert, hogy négy biten *(nibble)* 16 különböző kódot tárolhatunk, és mivel tízes alapú számrendszerben 0-tól 9-ig vannak számjegyek, ez nekünk bőven elég is. A kódolás olyan értelemben fixpontos, hogy tudjuk előre, hogy hol helyezkedik el a tizedespont, az eltérő ábrázolási mód miatt mégsem sorolhatjuk ebbe a csoportba.

Joggal merülhet fel a kérdés, hogy miért van erre a tárolási módra szükségünk? Egyrészt nagyon jól használható olyan esetekben, amikor az adatainkat kijelzésre alkalmassá, az ember számára könnyen "olvashatóvá" kell tennünk. Tehát olyan feladatoknál, ahol viszonylag kevés az aritmetikai, ellenben sok az *I/O* művelet. Másrészt tudjuk, hogy a törtszámok kettes számrendszerbe való átváltásakor a legritkább esetben kapunk pontos eredményt. Amennyiben az előbb tárgyalt módokon tároljuk a számokat, ez a nagy pontosságot igénylő számításoknál *(pl. Bankok, Kutatóközpontok, stb. . . .)* jelentős hibákat eredményezhet. Akár a huszadik helyen álló törtjegy értéke is fontos lehet számunkra, miközben lehetséges, hogy már az első tizedes jegy értéke sem pontos. Mivel ennél a tárolási módnál nem a számot, hanem a számjegyeket tároljuk, könnyen megoldható, hogy a kívánt pontosság érdekében az egész rész és a törtrész tárolására tetszőleges számú nibble-t biztosítsunk.

BCD kód esetében a bitek súlyozása 8-4-2-1, de léteznek ettől eltérő, elsősorban a méréstechnikában, automatizálásban használatos megoldások is *(pl. BCD Aiken kód - 2 4 2 1)*.

5.11. példa. BCD kód

A 13907 decimális szám megfelelője BCD kódban:

13907 –> 0001 0011 1001 0000 0111.

Megjegyzés: a 1001_2-nél nagyobb számok tiltottak, megjelenésük sok rendszerben hibát eredményezhet.

5.4.1. Számábrázolási módok, ASCII és EBCDIC kódtáblák

Ezen számábrázolási módnak több variánsa is van attól függően, hogy az ASCII vagy az EBCDIC kódtáblára alapul, továbbá mindkettőn belül van még zónás és pakolt tömörített forma is *(ez utóbbiban megegyezik a két alak)*.

ASCII

Az ASCII *(American Standard Code for Information Interchange - amerikai szabványos kód az információ kölcsönös cseréjére)* egy egységesített kódrendszer, amelyben a különféle karakterekhez *(betűk, számok, vezérlőkarakterek, írásjelek)* bináris kódokat rendelnek. A kódot az American Standard Institute dolgozta ki, amelyet az 1977-ben az Amerikai Szabványügyi Hivatal megerősítése és jóváhagyása után a Nemzetközi Szabványügy Hivatal *(ISO)* is átvett *(ISO646)*.

Kezdetben az ASCII egy 7 bites kód volt, amely $2^7 = 128$ különféle bitsorozatot tartalmazott 0-tól 127-ig sorszámozva. Ez az ún. alap vagy standard karakterkészlet (az angol ábécé kis- és nagybetűi, számjegyek, írásjelek). Később az IBM kezdeményezésére hozzáadott 1 bites kiterjesztéssel újabb 128 karakter használatát szabványosította, amely kódrendszer Latin1 néven vált ismerté. Ez a nyolcbites kóddá való kiegészítés tette lehetővé a nemzeti sajátosságokat is figyelembe vevő karakterkészlet kialakítását: 128-255 között az ún. kiegészítő karakterkészlet *(számos európai nyelv – pl. francia, német spanyol, stb. – speciális nemzeti karakterei, ékezetes betűk, az angolban nem létező egyéb betűtípusok, vonalrajzoló, a görög ABC betűi, táblázatrajzoló karakterek, stb.)* elemei találhatók. Ezeket nevezik kódlapoknak is, pl. Latin1, Latin2, 852-es kódlap, stb. Ez utóbbi, a 852-es a Magyar Szabványügyi Hivatal által is elfogadott kódlap, amely a teljes magyar karakterkészletet tartalmazza.
A teljes kódtábla az alábbi címen érhető el és tekinthető meg:
http://www.asciitable.com/

ASCII kódtábla részlete

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
DEC	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**
HEX	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**

EBCDIC

A BCD kód kiterjesztett változata, elnevezése *(Extended Binary Coded Decimal Interchange Code = kiterjesztett BCD kód)* is erre utal. Az EBCDIC 38 tulajdonképpen egy olyan 8 bites kódkészlet, amely szöveg, grafika és vezérlőkarakterek ábrázolását teszi lehetővé a számítógépeken. Érdekessége a kódképzés szabálya, nem egy egyszerű hozzárendelés alapján kódolódnak a karakterek, hanem a sorszámuknak megfelelően.

EBCDIC kódtábla részlete

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
DEC	**240**	**241**	**242**	**243**	**244**	**245**	**246**	**247**	**248**	**249**
HEX	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**

Előjeles ábrázolás

Előjeles ábrázolási mód értelmében az előjelet és az egyes számjegyeket is külön bájtokon ábrázoljuk. A számjegyet a bájt alsó részében tároljuk, a felső részét pedig a kódtáblától függően (3 vagy F) töltjük fel.

ASCII

1. bájt az előjel:

$2B_{16}$ +

$2D_{16}$ −

A további bájtokon ASCII kódban a számjegyek.

$$3i_{16}, i = 0,1,2,..,9$$

2	B+ / D−	3	i	...		3	i

0010	1011 1101	0011	i_{16}	...		0011	i

5.12. példa. $+-309$

+309

2B	33	30	39
0010 1011	0011 0011	0011 0000	0011 1001

−309

2D	33	30	39
0010 1101	0011 0011	0011 0000	0011 1001

EBCDIC

1. bájt előjel:

$4E_{16}$ +

60_{16} −

A további bájtokon ASCII kódban a számjegyek.

$$F_{i16}$$

$$i = 0,1,2,..,9$$

4/ 6	E+ / 0−	F	i	...		F	i
0100/ 0110	1110 0000	1111	i_{16}	...		1111	i

5.13. példa. $+-516$

+516

4E	F5	F1	F6
0100 1110	1111 0101	1111 0001	1111 0110

−516

60	F5	F1	F6
0110 0000	1111 0101	1111 0001	1111 0110

Zónás ábrázolás

A zónás ábrázolásnál is minden egyes számjegyet külön bájtban tárolunk. A számjegyet a bájt alsó részében tároljuk, a felső részét pedig a kódtáblától függően (3 vagy F) töltjük fel. A szám előjelét itt már nem külön bájt-on, hanem az első bájt zónajelében tároljuk.

ASCII

Az utolsó bájton lévő előjel és számjegy:

ha $i >= 0$ akkor $3i_{16}$,

különben $7i_{16}$.

3	i	3	i	...	...	3 + 7 −	i

5.14. példa. $+-418$

+418

34	31	38
0011 0100	0011 0001	0011 1000

−418

34	31	78
0011 0100	0011 0001	0111 1000

EBCDIC

A pozitív előjelnek a **C**, a negatívnak pedig a **D** értékek felelnek meg.

F	i	F	i	...	...	C + D −	i

5.15. példa. +−418

+418

F4	F1	C8
1111 0100	1111 0001	1100 1000

−418

F4	F1	D8
1111 0100	1111 0001	1101 1000

Pakolt tömörített forma

A pakolt ábrázolási módnál elhagyjuk a zónajeleket, és a számjegyek tárolását nem bájtonként, hanem fél bájtonként végezzük. Így egy bájton két számjegyet tudunk ábrázolni. Az előjel az utolsó fél bájtra kerül. Ezzel a formával már aritmetikai műveletek is végezhetőek.

Előjel

\+ helyett C

− helyett D

Ha a számjegyek száma páros, akkor a szám elé egy 0-át kell írnunk, hogy az ábrázolás az előjellel együtt egész bájtokon képződjön.

5.16. példa. +− 31 052

\+ 31 052 ==> 31 05 2C

− 31 052 ==> 31 05 2D

5.17. példa. − 453107

− 453107 ==> − 04 53 10 7D

0000	0100	0101	0011	0001	0000	0111	1101
0	4	5	3	1	0	7	D

5.4.2. Feladatok

Ábrázolja a következő számokat *(előjeles, zónás, pakolt)* alakokban:

a, +− 2347

b, +− 596

6. UTASÍTÁSOK, ALGORITMUSOK, PROGRAMOK

6.1. Az utasítás fogalma

Az előző fejezetben megismerkedtünk a számítógéppel feldolgozható információk, adatok tárolásával és ábrázolásával. Ezen adatok feldolgozásához, mint azt már jeleztük is, a számítógépben tárolni kell azokat az utasításokat is, amelyek alapján a kijelölt feladat automatikusan, lépésről-lépésre haladva megoldást eredményez, vagyis kimenő adatokat (output), információkat eredményez. Szükséges tehát annak tisztázása is, hogy az utasítások ábrázolása és tárolása hogyan valósítható meg. Először azonban tisztázni kell milyen utasításokat értelmezhetünk a számítógépnél.

Definíció. Az utasítás olyan tevékenység pontos megfogalmazása, amelyet tovább részletezni nem tudunk, vagy nem akarunk, és a végrehajtó számára egyértelmű jelentéssel bír.

A számítógépes utasítás nyilván szűkebb területre vonatkozik, a gép belső működéséhez, állapotváltozásához nyújt információt.

A számítógép mint automata az utasítások hatására a kiinduló, kezdeti állapotból több közbülső állapoton keresztül eljut egy végső állapotba, melynek hatására az input adatokból eredményadatok állnak elő.

Egy-egy számítógépes utasítás alapvetően

- műveletek elvégzését kezdeményezi, vagy
- vezérlési feladatot lát el.

Kétfajta elven működő számítógépeken különböztethetünk meg, bár manapság már ötvözik a két technológiát.

1. A CISC (a Complex Instruction Set Computer, vagyis "összetett utasításkészlettel rendelkező számítógép"). Az ilyen processzoroknak utasításkészlete több és komlexebb mint a RISC processzoroké. A CISC processzorok utasításai sokszor több elemi műveletet végeznek egyidőben,

a gépi kódú programjaik rövidebbek, jobban átláthatóak, ami egyszerűsíti a fordítóprogramok működését. Viszont a a bonyolultabb utasítások megnövelhetik a végrehajtási időt. Ilyen bonyolúlt processzorok pl a X86-os architektúrák, amit az Intel és az AMD gyárt.

2. A RISC (Reduced Instruction Set Computing), csökkentett utasításkészletű számítástechnika, vagy erre az elvre épülő számítógépi processzor tervezés. Egyszerűség miatt a RISC elvet a mikrokontrollerek tervezésénél is kihasználják. RISC technológiát alkalmaznak például az Oracle (korábban Sun Microsystems) SPARC számítógépei vagy az IBM Power Architecture típusú szuperszámítógépei. A RISC technológia alkalmazásának új területe az ARM architektúrájú hordozható eszközök, tabletek és ultrabookok piaca, illetve a szintén főleg ARM-re készített Microsoft Windows CE (beágyazott Windows) technológia.

6.2. A program és algoritmus fogalma

Egy számítógéppel megoldandó feladat, probléma megoldása pontosan megfogalmazott és megfelelő sorrendben leírt utasítások sorozata, amely alapján a gép a szükséges adatok ismeretében előállítja a probléma megoldásához szükséges információt.

Definíció. A program olyan utasítássorozat, amely egy kijelölt (változtatható) input adatstruktúrából egy meghatározott output adatstruktúrába való átmenet lépéseit írja elő. A program számítógéppel történő végrehajtását dinamikus programnak nevezzük.

A program általában két részből áll, az adatdefiníciós és a vezérlésleíró részből. Az adatdefiníciós részben az előforduló adatok típusait adjuk meg, a vezérlésleíró részben pedig az eredményt előállító utasítások sorrendjét és tartalmát rögzíthetjük.

Egy-egy probléma megoldási lépéseit, vagy egy folyamat, tevékenységsorozat elvégzését írásban rögzítjük.

Definíció. Az algoritmus egy probléma általános érvényű megoldásának véges számú részlépésben történő egyértelmű és teljes leírása.

Általános érvényű olyan értelemben, hogy segítségükkel több, esetleg végtelen sok egymástól csak bemenő adatokban különböző feladat megoldható. Ebből következően minden algoritmus értelmezési tartománnyal rendelkezik.

Egyértelműségen azt értjük, hogy minden részlépés után egyértelműen meghatározható a soronkövetkező, a *teljességen* pedig azt, hogy az utolsó részlépés kivételével, amely az algoritmus vége, minden részlépésnek van rákövetkezője.

Az algoritmus és az utasítás fogalma is független a számítógéptől, azaz általánosabb érvényű. Az algoritmusok segítségével olyan tevékenységek végrehajtási sorrendje is leírható, amelyek elvégzése számítógéppel nem lehetséges, de a gyakorlatban hasznosak. Sok mindennapi tevékenységünk is algoritmizálható.

Az algoritmusok leírási módjára nincsenek külön megkötések. Több algoritmusleíró eszköz ismeretes. Ezek célja a megoldás menetének, a program vezérlési részének géptől és nyelvtől független, szemléletes, a logikai gondolatmenetet, a szerkezeti egységeket világosan tükröző leírása.

Algoritmuselíró eszközök:

a) folyamatábra,

b) struktogram,

c) mondatszerű leírás.

6.3. Feladatok algoritmusokra

1. Írjuk le az alábbi képletek behelyettesítési értékének kiszámítási algoritmusát feltételezve, hogy a gép egy utasítás során csak egy művelet elvégzésére képes. Értékeljük ki ezeket a kifejezésket táblázatkezelővel is.

   a)
   $$\frac{(5a+1)a}{2a+1},$$

   b)
   $$\left(\frac{a^2+b^2}{15}-(a+b)^3\right)^2,$$

   c)
   $$\frac{x^2+y^2}{(x+y)^2},$$

   d)
   $$1+x+\frac{x^2}{2}+\frac{x^3}{6}.$$

2. Írjunk algoritmust a létező háromszögek halmazából egy elem kiválasztása esetén a pontos megnevezésének meghatározására.

3. Írjunk algoritmust adott n db egész szám összegének meghatározására.

4. Végezzük el az összegzést, addig amíg az összeg utolsó tagja nem lesz kisebb egy adott p pontosságnál. Írjuk le az algoritmusát.

   a)
   $$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots,$$

   b)
   $$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots,$$

   c)
   $$\frac{1}{1\cdot 2}+\frac{1}{3\cdot 4}+\frac{1}{5\cdot 6}+\frac{1}{7\cdot 8}+\cdots.$$

5. Írjunk algoritmust adott K kereset után levonandó adóelőleg kiszámítására a jelenlegi érvényes előírások alapján.

6. Adott egy évszám, egy hónap és egy nap. Készítsünk algoritmust, amely meghatározza, hogy ez a dátum az év hányadik napja.

7. Adott egy osztály fiú tanulóinak névsora és külön a leányok névsora ábécé sorrendben. Írjunk algoritmust, amely előállítja az osztály ábécés névsorát.

8. Készítsünk algoritmust az alábbi titkosírás kódolásához és dekódolásához: minden betű helyett a ciklikusan rákövetkezőt írjuk (Z helyett A) és a nem betűjeleket nem változtatjuk meg a kódolásnál.

9. Készítsünk algoritmust a Horner-féle elrendezésre.

7. KIFEJEZÉSEK KIÉRTÉKELÉSE

7.1. Kifejezés

A *kifejezés* egyetlen operandus vagy operandusok és operátorok sorozatából álló nyelvi elem, ahol az *operátorok* különféle műveleti jelek, melyek összekapcsolják a kifejezésekben szereplő operandusokat. Egy operandust, amennyiben például egy konstans érték, azonosító, sztring, metódushívás, tömbindex vagy tagkiválasztó operátor, *elsődleges kifejezésnek* nevezzük. Természetesen egy operandus lehet egy zárójelezett vagy zárójel nélküli operátorokkal összekapcsolt további operandusokból álló összetett kifejezés is.

A *művelet* egy olyan tevékenység *(sorozat)*, amit az operátorok előírnak. A *kifejezés kiértékelése* nem más, mint a benne szereplő összes művelet elvégzése. Elsőbbségi *(precedencia)* szabályok a műveletek során a kifejezések kiértékelési sorrendjét meghatározó szabályok, amelyek által precedencia szintenként csoportosíthatóak a műveletek. A sorrend zárójelezés segítségével testre szabható, mert először mindig a zárójelben lévő műveletek hajtódnak végre.
Megjegyzés: Az operátorok többsége túlterhelhető, amennyiben az egyik vagy mindkét operandusa felhasználó által definiált osztály vagy típus.

7.1.1. Asszociativitás

Amennyiben több azonos precedencia szinten lévő operátorral van dolgunk, akkor a kiértékelési sorrendje általában balról jobbra haladva történik, kivételt képeznek az egy operandusú, értékadással egybekötött két operandusú, illetve a három operandusú műveletek, melyek kiértékelése jobbról balra történik. A sorrend zárójelezés segítségével testre szabható, mert először mindig a zárójelben lévő műveletek hajtódnak végre.

7.1. példa. Jobbról-balra szabály
A $a = b = c = 5$; azonos az $a = (b = (c = 5))$; kifejezéssel, mivel az értékadó utasításoknál a kiértékelés jobbról-balra történik.

7.2. példa. Balról-jobbra szabály

Az alábbi kifejezésben / és $*$, illetve a $+$ és a $-$ azonos precedencia szinten vannak.

$$a + b/c * d - e;$$

A kiértékelés balról-jobbra történik, először a /, majd a $*$, ezt követően az $+$ és végül a $-$ kerül kiértékelésre.

Konkrét értékekkel vizsgálva, a műveletvégrehajtás sorrendje a következő:

```
1 double a = 7, b = 10, c = 2, d = 3, e = 4;
2
3 1. b / c = 5
4 2. b / c * d = 5 * d = 15
5 3. a + b / c * d = a + 15 = 22
6 4. a + b / c * d - e = 22 - e = 18
```

7.3. példa. Zárójelezés

Mint arról az előzőekben volt szó, amennyiben egy kifejezésben különböző precedencia szinteken lévő műveletek vannak, a kiértékelést mindig a magasabb precedenciájú műveleteket tartalmazó részkifejezés kiértékelésével kell kezdenünk. A zárójelezéssel a kiértékelés sorrendjét változtathatjuk meg.

Amennyiben $a + \frac{b}{c} \cdot d - e$ helyett a $\frac{a+b}{c} \cdot d - e$ kifejezés értékét szeretnénk kiszámolni, a $+$ művelet precedenciáját zárójelezéssel előrébb kell hozni,

$$(a + b)/c * d - e.$$

Konkrét értékekkel vizsgálva, a műveletvégrehajtás sorrendje a következő:

```
1 double a = 7, b = 10, c = 2, d = 3, e = 4;
2
3 1. (a + b) = 17;
4 2. (a + b ) / c = 17 / c = 8,5;
5 3. (a + b ) / c * d = 8,5 * d =  25,5;
6 4. (a + b ) / c * d - e = 25,5 - e = 21,5;
```

7.1.2. Mellékhatás (side effect)

Bizonyos műveletek pl. függvényhívás, többszörös értékadás, léptetés (++, −−) feldolgozásakor jelentkező jelenség, melynek során a kifejezés értékének megjelenése mellett bizonyos változók is megváltoztathatják értékeiket. Kiértékelésük sorrendjét nem határozza meg a C# szabvány, így ügyelni kell rájuk, el kell kerülni az olyan utasításokat, melyek kiértékelése függ a precedenciától.

7.1.3. Rövidzár (short circuit)

Az a kiértékelési mód, amely során nem szükséges kiértékelni a teljes kifejezést ahhoz, hogy egyértelműen meghatározzuk az értékét. Például ha egy && bal oldali operandusa 0, a jobb oldalit már szükségtelen kiértékelni, a kifejezés értéke egyértelműen 0 lesz.

7.2. C# nyelv

A *C#* nyelvben használható operátorok **precedencia sorrendje** a következő:

1. **Elsődleges kifejezések:** x.y, f(x), a[x], x++, x−−, new, typeof, checked, unchecked, sizeof, − >;
2. **Egy operandusú (unáris) operátorok:** +, −, !, ∼, ++x, −−x, (T)x;
3. **Multiplikatív operátorok:** ∗, /, %;
4. **Additív operátorok:** +, −;
5. **Biteltoló operátorok**: <<, >>;
6. **Összehasonlító (relációs) és típus operátorok:** <, >, <=, >=, is, as;
7. **Egyenlőségvizsgáló operátorok:** ==, ! =;;

8. **Logikai, feltételes és Null operátorok:** &, ^, |, &&, ||, ??, ? :;

9. **Értékadó és anonim operátorok:** =, ∗ =, /=, %=, +=, − =, <<=, >>=, &=, ^=, |=;

Mi most csak a kifejezések kiértékelésének megértéséhez szorosan köthető, precedencia szintenként a fontosabb operátorokkal ismerkedünk meg.

7.2.1. Egyoperandusú műveletek

Prefix és postfix alak

Egyoperandusú műveletek esetén a kifejezés vagy az *operátor operandus* vagy *operandus operátor* alakot veszi fel. Az első esetben *prefix,* míg a második esetben *postfix* alakról beszélünk.

7.4. példa. Prefix és postfix alak
int i = 2;
$++i$; prefix alak, i értékének növelése kettővel;
$i--$; postfix alak, i értékének csökkentése kettővel.

Egyoperandusú műveletek közé tartoznak az elsődleges kifejezések és az unáris operátorok.

Elsődleges kifejezések

.	tagkiválasztó operátor
()	zárójel operátor
[]	szögletes zárójel
op++	utólagos inkrementálás
op−−	utólagos dekrementálás
new	dinamikus helyfoglalás, objektumok létrehozása, stb.
typeof	típus meghatározó operátor
checked	túlcsordulás ellenőrzés
unchecked	túlcsordulás ellenőrzés mellőzése
sizeof	típus méretének lekérdezése
− >	komb. a pointerek hivatkozásának feloldását és tagkiválasztást

Unáris operátorok

+	változatlanul hagyás jele
−	kettes komplemens;
++op	elsődleges inkrementálás
−−op	elsődleges dekrementálás
!x	logikai tagadás
∼x	bitenkénti tagadás
(T)x	explicit típuskonverzió

Léptető operátorok (inkrementáló/dekrementáló)

Feladatuk a numerikus típusú változók értékének eggyel történő növelése vagy csökkentése.

prefixes alakok: $++i$, $--i$;

postfixes alakok: $++i$, $--i$.

Megjegyzés: Az i++ -t és az ++i -t, az $i = i+1$ és az $i+ = 1$, míg az i−− -t és az −−i -t, az $i = i-1$ és az $i- = 1$ kifejezések helyett célszerű alkalmazni.

7.5. példa. Léptető operátorok

```
1  int q=6, p=7, r=8, s=9, eredmeny;
2  eredmeny = ++q - p--;
3  Console.WriteLine("A művelet után a q értéke: "+ q + ", p értéke: " +
       p + " az eredmény: " + eredmeny);
4  eredmeny = p++ + r--;
5  Console.WriteLine("A művelet után a p értéke: "+ p + ", r értéke: " +
       r + " az eredmény: " + eredmeny);
6  eredmeny = --r - s++;
7  Console.WriteLine("A művelet után a r értéke: " + r + ", s értéke: "
       + s + " azeredmény: " + eredmeny);
8  eredmeny = s-- + ++r;
9  Console.WriteLine("A művelet után a s értéke: " + s + ", r értéke: "
       + r + " az eredmény: " + eredmeny);
10 Console.ReadKey();
```

A program futásának eredménye:

```
A művelet után a q értéke: 7, p értéke: 6 az eredmény: 0
A művelet után a p értéke: 7, r értéke: 7 az eredmény: 14
A művelet után a r értéke: 6, s értéke: 10 azeredmény: -3
A művelet után a s értéke: 9, r értéke: 7 az eredmény: 17
```

7.1. ábra. Léptető operátorok

Megfigyelhető, hogy az első (++q , $p--$) művelet elvégzése után az eredmény 0, holott kiíratásnál q értéke 7, a p értéke pedig 6. Ennek oka, hogy postfixes alak esetén a léptetés csak a kifejezés kiértékelése után következik be. Kiértékeléskor még a q és a p értéke is 7 volt, a p értéke ezt követően dekrementálódott.

typeof

A typeof(T) típus meghatározó operátor.

7.6. példa. typeof

```
1  Type t1 = typeof(bool);
2  Console.WriteLine(t1.Name);
3  Type t2 = typeof(byte);
4  Console.WriteLine(t2.Name);
5  Type t3 = typeof(int);
6  Console.WriteLine(t3.Name);
7  Type t4 = typeof(float);
8  Console.WriteLine(t4.Name);
9  Type t5 = typeof(double);
10 Console.WriteLine(t5.Name);
11 Console.ReadLine();
```

A program futásának eredménye:

```
Boolean
Byte
Int32
Single
Double
```

7.2. ábra. typeof

Megjegyzés: A float egy 32 bites, míg a double egy 64 bites, IEEE 754-es szabványnak megfelelő lebegőpontos szám tárolására alkalmas típus.

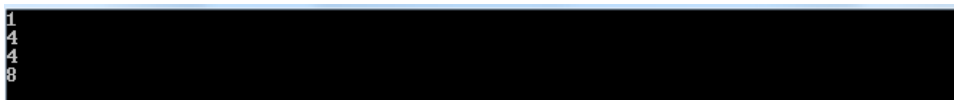
sizeof

A sizeof egy adott típus méretének lekérdezésére alkalmas.

7.7. példa. sizeof

```
1 int byteSize = sizeof(byte);
2 Console.WriteLine(byteSize);
3 int intSize = sizeof(int);
4 Console.WriteLine(intSize);
5 int floatSize = sizeof(float);
6 Console.WriteLine(floatSize);
7 int doubleSize = sizeof(double);
8 Console.WriteLine(doubleSize);
9 Console.ReadLine();
```

A program futásának eredménye:



7.3. ábra. sizeof

Túlcsordulás ellenőrzés

Túlcsordulás (overflow): Mind fixpontos, mind lebegőpontos számábrázolás esetén fellépő számolási hiba, amely akkor fordul elő, amikor egy számot nagyobbra növelünk *(pl. összeadás vagy szorzás)*, mint amekkora maximális értéket tárolni tud az adott számábrázolás. Ilyenkor nagyobb helyértékek elvesznek. Természetesen ennek ellenkezője is előfordulhat *(pl. osztásnál vagy kivonásnál)*, ekkor alulcsordulásról *(underflow)* beszélünk.

1. *checked* túlcsordulás ellenőrzés;
2. *unchecked* túlcsordulás ellenőrzés mellőzése.

7.8. példa. Túlcsordulás ellenőrzés

```
1  class Tulcsordulas
2  {
3      const int x = 1000000;
4      const int y = 1000000;
5      static int F()
6      {
7          return checked(x * y);     // Fordítási hiba, túlcsordulás
8      }
9      static int G()
10     {
11         return unchecked(x * y);  // Visszatérési érték: -727379968
12     }
13     static int H()
14     {
15         return x * y;              // Fordítási hiba, túlcsordulás
16      }
17 }
```

Az F() és H() metódusok esetében fordítási idő alatti hibát jelez (túlcsordulás), viszont a G() esetében, mivel ott kikapcsoltuk az ellenőrzést, nem jelez hibát, elvégzi a műveletet és az eredményt csonkolja.

Unáris műveletek

+	változatlanul hagyás
−	logikai tagadás
!	logikai tagadás
∼	bitenkénti tagadás

7.9. példa. Unáris műveletek

```
1  int x = 3;
2  Console.WriteLine(x);
3  Console.WriteLine(-x);
4  Console.WriteLine(!false);
5  Console.WriteLine(!true);
6  int[] ertekek = { 0, 0x111, 0xfffff, 0x8888, 0x22000022 };
7
8  foreach (int e in ertekek)
9  {
10     Console.WriteLine("\~{}0x\{0:x8\} = 0x\{1:x8\}", e, \~{}e);
11 }
12 Console.ReadLine();
```

A program futásának eredménye:

```
3
-3
True
False
~0x00000000 = 0xffffffff
~0x00000111 = 0xfffffeee
~0x000fffff = 0xfff00000
~0x00008888 = 0xffff7777
~0x22000022 = 0xddfffffd
```

7.4. ábra. Unáris műveletek

7.2.2. Kétoperandusú műveletek

A legtöbb operátor két operandussal rendelkezik, ezeket kétoperandusú (bináris) operátoroknak nevezzük. Általános felépítésük a következő:

operandus1 operátor operandus2

Amennyiben több azonos precedencia szinten lévő kétoperandusú operátorral van dolgunk, a kiértékelés általában balról jobbra haladva történik, kivételt képeznek az értékadással egybekötött két operandusú műveletek, melyek kiértékelése jobbról balra történik.

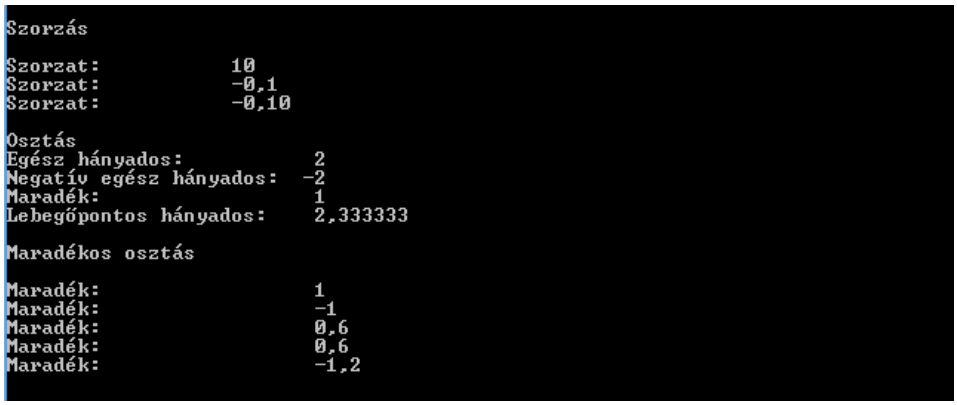
Multiplikatív operátorok

*	szorzás
/	osztás
%	maradékképzés

7.10. példa. Multiplikatív operátorok

```
1  //szorzás
2  Console.WriteLine("\nSzorzás\n");
3  Console.WriteLine("Szorzat: {0}", 5 * 2);
4  Console.WriteLine("Szorzat: {0}", -.5 * .2);
5  Console.WriteLine("Szorzat: {0}", -.5m * .2m); // decimális típus
6  Console.WriteLine("\nOsztás");
7  Console.WriteLine("Egész hányados:          {0}", 7 / 3);
8  Console.WriteLine("Negatív egész hányados:  {0}", -7 / 3);
9  Console.WriteLine("Maradék:                 {0}", 7 % 3);
10 float osztando = 7;
11 Console.WriteLine("Lebegőpontos hányados:{0}", osztando / 3);
12 Console.WriteLine("\nMaradékképzés \n");
13 Console.WriteLine("Maradék:    {0}", 5 % 2);        // int
14 Console.WriteLine("Maradék:    {0}", -5 % 2);       // int
15 Console.WriteLine("Maradék:    {0}", 5.0 % 2.2);    // double
16 Console.WriteLine("Maradék:    {0}", 5.0m % 2.2m); // decimal
17 Console.WriteLine("Maradék:    {0}", -5.2 % 2.0);   // double
18 Console.ReadLine();
```

A program futásának eredménye:



7.5. ábra. Multiplikatív operátorok

Additív operátorok

x + y	összeadás; szöveg esetén, azok összefűzése
x − y	kivonás

7.11. példa. Additív operátorok

```
1  Console.WriteLine(+5);          // unáris plusz
2  Console.WriteLine(5 + 5);       // összeadás
3  Console.WriteLine(5 + .5);      // addition
4  Console.WriteLine("5" + "5"); // szöveg konkatenáció
5  Console.WriteLine(5.0 + "5"); // szöveg konkatenáció
6  //automatikus típuskonverzió (double-ből string-be)
7  int a = 5;
8  Console.WriteLine(-a);  // kettes komplemens képzés
9  Console.WriteLine(a - 1); //kivonás
10 Console.WriteLine(a - .5); //kivonás
11 Console.ReadLine();
```

A program futásának eredménye:

7.6. ábra. Additív operátorok

Biteltoló operátorok

Egy szám bitjeinek jobbra illetve balra történő eltolása n bittel, ami tulajdonképpen 2^n-en értékkel történő osztásnak, illetve szorzásnak felel meg.

x << y	eltolás balra
x << y	eltolás jobbra

7.12. példa. Biteltoló operátorok

```
1  SByte si = 2; //előjeles egész
2  Byte usi = 12; //előjel nélküli egész
3  Console.WriteLine(si << 1); //szorzás kett?vel
4  Console.WriteLine(si << 32); // 32 bináris alak 100000, nem változik
       az eredmény
5
6  Console.WriteLine(si << 33); // 33 bináris alakja 100001, szorzás
       kettővel
7
8  Console.WriteLine(si >> 1); //osztás kettővel
9  Console.WriteLine(si >> 32);    // 32 bináris alak 100000, nem
       változik az eredmény
10 Console.WriteLine(si >> 33);// 33 bináris alakja 100001, osztás
       kettővel
11 Console.WriteLine( usi << 2); //szorzás néggyel
12 Console.WriteLine( usi >> 2); //osztás néggyel
```

A program futásának eredménye:

7.7. ábra. Biteltoló operátorok

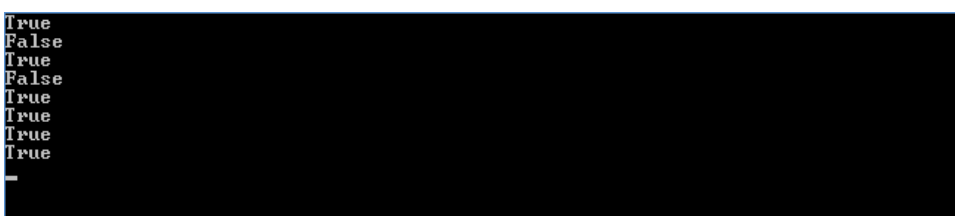
Összehasonlító (relációs) és típus operátorok

<	kisebb, mint
>	nagyobb, mint
<=	kisebb egyenlő
>=	nagyobb egyenlő

7.13. példa. Összehasonlító (relációs) és típus operátorok

```
1 Console.WriteLine(1 < 1.1);
2 Console.WriteLine(1.1 < 1.1);
3 Console.WriteLine(1.1 > 1);
4 Console.WriteLine(1.1 > 1.1);
5 Console.WriteLine(1 <= 1.1);
6 Console.WriteLine(1.1 <= 1.1);
7 Console.WriteLine(1.1 >= 1);
8 Console.WriteLine(1.1 >= 1.1);
9 Console.ReadLine();
```

A program futásának eredménye:



7.8. ábra. Összehasonlító (relációs) és típus operátorok

is

Az is egy objektumról megmondja, hogy a bal oldali operandus a jobb oldali típusnak egy változója-e.

as

Kétoperandusú típuskényszerítés. A bal oldali változót a jobb oldali referencia típusra alakítja, ha tudja. Ha sikertelen az átalakítás, akkor eredményül a null értéket adja.

Egyenlőségvizsgálat operátorai

x == y	egyenlő
x != y	nem egyenlő

7.14. példa. Egyenlőségvizsgálat operátorai

```
1  // Numerikus értékek egyenlőségének vizsgálata
2  Console.WriteLine((2 + 2) == 4);
3
4  // referencia egyenlőség: különböző objektumok
5  object s = 1;
6  object t = 1;
7  Console.WriteLine(s == t);
8
9  string a = "hello";
10 string b = String.Copy(a);
11 string c = "hello";
12
13 // Stringek egyenlőségének vizsgálata
14 Console.WriteLine(a == b);
15
16 // Stringek referenciáinak egyenlőségének vizsgálata
17 Console.WriteLine((object)a == (object)b);
18 Console.WriteLine((object)a == (object)c);
19 Console.ReadLine();
```

A program futásának eredménye:

```
True
False
True
False
True
```

7.9. ábra. Egyenlőségvizsgálat operátorai

Logikai, feltételes és Null operátorok

Logikai műveletek

x & y	bitenkénti *és* művelet
x ^ y	bitenkénti *kizáró vagy* művelet
x |y	bitenkénti *vagy* művelet

7.15. példa. Logikai műveletek

```
1  int a = 7, b = 4, c = 0;
2  // és művelet
3  Console.WriteLine("a és b = {0}", a & b);
4  Console.WriteLine("a és b és c = {0}", a & b & c);
5  //vagy m?velet
6  Console.WriteLine("b vagy c = {0}", b | c);
7  Console.WriteLine("a vagy b vagy c = {0}", a | b | c);
8  //
9  Console.WriteLine("b kizáró vagy c = {0}", b ^ c);
10 Console.WriteLine("a kizáró vagy b kizáró vagy c = {0}", a ^ b ^ c);
11 Console.ReadLine();
```

A program futásának eredménye:

```
a és b = 4
a és b és c = 0
b vagy c = 4
a vagy b vagy c = 7
b kizáró vagy c = 4
a kizáró vagy b kizáró vagy c = 3
```

7.10. ábra. Logikai műveletek

Feltételvizsgálat

x && y	logikai *és* művelet

Az y-t csak abban az esetben értékeli ki, ha az x igaz.

x ||y	logikai *vagy* művelet művelet

Az y-t csak abban az esetben értékeli ki, ha az x hamis.

x ?? y	kiértékeli az y-t ha x nulla, egyébként x

Értékadó és anonim operátorok

=	értékadás
*op*erátor=	összetett értékadás
+=	összeadó értékadás; *pl.* a += 3;
− =	kivonó értékadás; *pl.* a -= 3;
∗ =	szorzó értékadás; egyenértékű az a = a * 5;-el
/=	osztó értékadás; *pl.* a /= 2;
%=	maradék osztó értékadás ; *pl.* a %= 3;
&=	bitenkénti és műveletet végző értékadás; *pl.* a &= 3;
|=	bitenkénti vagy műveletet végző értékadás; *pl.* a |= 3;
!=	bitenkénti nem műveletet végző értékadás
<<=	bitenkénti balra léptető értékadás
>>=	bitenkénti jobbra léptető értékadás

7.16. példa. Értékadó és anonim operátorok

```
int a = 5;
a += 3;
Console.WriteLine("a értéke:" + a);
a *= 3;
Console.WriteLine("a értéke:" + a);
a -= 3;
Console.WriteLine("a értéke:" + a);
a /= 3;
Console.WriteLine("a értéke:" + a);
a %= 3;
Console.WriteLine("a értéke:" + a);
a &= 3;
Console.WriteLine("a értéke:" + a);
a |= 3;
Console.WriteLine("a értéke:" + a); a <<= 3;
Console.WriteLine("a értéke:" + a);
a >>= 3;
Console.WriteLine("a értéke:" + a);
```

A program futásának eredménye:

```
a értéke:8
a értéke:24
a értéke:21
a értéke:7
a értéke:1
a értéke:1
a értéke:3
a értéke:24
a értéke:3
```

7.11. ábra. Értékadó és anonim operátorok

7.2.3. Háromoperandusú művelet

x ? y : z	kiértékeli az y-t , ha x igaz, z-t ha x hamis

7.17. példa. Háromoperandusú művelet

```
char c='6';
int a;
a = ((c >= '0' && c <= '9') ? c - '0' : -1);
Console.WriteLine("a értéke:" + a);
c = 'f';
a = ((c >= '0' && c <= '9') ? c - '0' : -1);
```

```
7 Console.WriteLine("a értéke:" + a);
8 Console.ReadLine();
```

A program futásának eredménye:

```
a értéke:6
a értéke:-1
```

7.12. ábra. Háromoperandusú művelet

Megjegyzés: A háromoperandusú operátor valójában egy logikai elágazás utasítás, mellyel hatékonyabbá *(tömörebbé)* tehető a kifejezéskészítés.

7.3. Excel

Ebben a fejezetben a táblázatkezelőknél jól ismert képletekkel, és azok kiértékelésével foglalkozunk. Hogy miért van erre szükség? A képletek kiértékelése eltér az előzőekben ismertetett kifejezések kiértékelésétől. Csak egy példát kiragadva, itt az *és* művelet nem operátor, hanem egy függvény, ami jelentősen módosítja a kiértékelés sorrendjét.

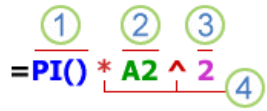
7.3.1. Képletek

Amennyiben egy cellába műveleti utasításokat írunk, akkor azt képletnek vagy kifejezésnek nevezzük. Egy képlet maximálisan 8192 karakterből állhat és = jellel kell kezdeni. Ebben az esetben az Excel értelmezés szerint tudja, hogy nem adatot kell a cellában tárolnia, hanem műveleteket kell elvégeznie. Egy képlet tartalmazhat számokat, hivatkozásokat (neveket), függvényeket és műveleti jeleket. Kifejezések kiértékelése a matematikában jól ismert precedencia szabály szerint történik.

Képletek kiértékelésének precedencia sorrendje

1. Zárójelek
2. Függvények
3. Egy operandusú műveletek
4. Százalékszámítás
5. Hatványozás
6. Multiplikatív műveletek
7. Additív műveletek
8. Relációk
9. Balról-jobbra szabály

7.18. példa. Képletek kiértékelése



7.13. ábra. Képletek kiértékelése

1. lépés - függvények kiértékelése: A `PI()` függvény visszatér a pi értékével 3.142...
2. lépés - hivatkozások kiértékelése: `A2` visszatér az `A2` cella értékével
3. lépés - konstansok kiértékelése: A numerikus és karakteres értékek közvetlenül használhatóak a képletben, mint például a 2.
4. lépés - műveletek elvégzése: A `^` operátor elvégzi a hatványozást, majd az eredményt megszorozza a *pi* értékével.

7.3.2. Adattípusok

A cellákban különböző típusú adatokat tárolhatunk. Egy cella tartalmazhat:

1. szöveges adatot,
2. numerikus adatot,
3. logikai adatot,
4. dátum adatot,
5. kifejezést *(képletet, függvényt).*

A szöveges adat betűk, számok és egyéb karakterek tetszőleges kombinációja lehet. pl. `1qwerty00, 06/30 123 4567`. A szöveges adatok a cella bal széléhez igazodnak.
Az Excel csak azokat az adatokat tekinti numerikusnak, amelyek megfelelnek az alábbi szintaktikai szabályoknak.

Egy szám tartalmazhat:

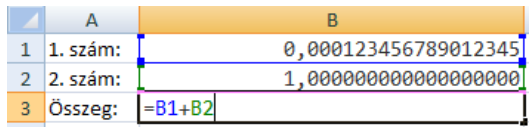
1. számjegyet,
2. tizedesjelet,
3. előjelet,
4. százalékjelet,
5. normál alakban megadott szám kitevőjének jeleit (`e` vagy `E`, pl. `23000` normál alakja: `2,3E+4`),
6. pénznem jelét.

A helyesen bevitt numerikus értékek alapértelmezés szerint a cella jobb oldalához vannak igazítva. Az Excel az IEEE 754-es specifikációnak megfelelően tárolja és számítja a lebegőpontos számokat. Mint arról a lebegőpontos számábrázolásnál már volt szó, a IEEE 754-es specifikációnak vannak korlátai, amelyek az alábbi három általános kategóriába sorolhatóak:

1. maximális/minimális korlátok,
2. pontosság,
3. szakaszos bináris számok.

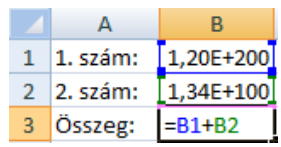
A következőekben olvasható két példa nagyon nagy és nagyon kicsi számokkal végzett műveletre.

7.19. példa. Műveletvégzés nagyon nagy számok esetében.
Az ábrán látható két számot összeadva az összeg a `B3`-as cellában `1,20E+200` lesz, azonos a `B1` cella értékével. Ennek az oka, hogy az IEEE 754 –es specifikációban a tárolás csak 15 értékes számjegy pontosságú. A fenti számítás tárolásához az Excel alkalmazásnak legalább 100 számjegy pontosságúnak kellene lennie.



	A	B
1	1. szám:	0,000123456789012345
2	2. szám:	1,000000000000000000
3	Összeg:	=B1+B2

7.15. ábra. Műveletvégzés nagyon kis számok esetében



	A	B
1	1. szám:	1,20E+200
2	2. szám:	1,34E+100
3	Összeg:	=B1+B2

7.14. ábra. Műveletvégzés nagyon nagy számok esetében

7.20. példa. Műveletvégzés nagyon kis számok esetében.
Az ábrán látható két számot összeadva az összeg a `B3`-as cellában `1,0001 23456789012345` helyett `1,00012345678901` lesz. Az ok ebben az esetben is ugyanaz, mint az előző esetben. Ahhoz, hogy pontos eredményt kapjunk, itt az alkalmazásnak legalább 19 számjegy pontosságúnak kellene lennie.

7.3.3. Függvények

A függvények a táblázatkezelő rendszerek egyik legfontosabb eszközei. Függvények használatával egyszerű vagy összetett számításokat végezhetünk. Egyszerű példán keresztül szemléltetve az `=ÁTLAG(A1:A5)` függvény e- gyenértékű az `=(A1+A2+A3+A4+A5)/5` képlettel. A függvényeket a képletekhez hasonlóan egyenlőségjellel kell kezdeni, amelyet csak az első függvénynév előtt kell feltüntetni, a kifejezésen belül már nem. A függvények két fő részből állnak: a függvény nevéből és az argumentumok listájából. Az argumentumok lehetnek: számok, nevek, szövegek, tartományok, cellahivatkozások, nevek, képletek, dátum, idő, függvénynév, adatbázisnév stb. Több argumentum esetén az argumentumokat pontosvesszővel kell egymástól elválasztani. A szöveges adatokat általában idézőjelek közé kell rakni.

```
=függvénynév(argumentum1;argumentum2;stb.)
```

Vannak olyan függvények, amelyekhez üres argumentum tartozik, a zárójeleket ebben az esetben is kötelező kirakni.

```
=függvénynév()
```

Ilyen függvények többek között a `=PI()` és a `=MA()`.
Az egymásba ágyazott függvények használatánál egyik probléma a helytelen zárójelezésből és paraméterezésből adódik. Abban az esetben, ha valamelyik függvényargumentum hiányzik, az eredménycellában a `#HIÁNYZIK` hibaüzenet jelenik meg. Az alábbi logaritmus érték

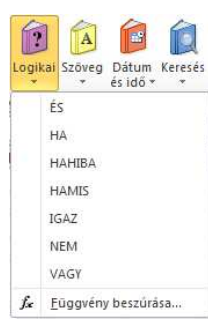
$$log_2\sqrt[3]{16}$$

kiszámolásához alkalmazható képlet:

```
=LOG(HATVÁNY(16;1/3);2)
```

Logikai függvények

A logikai függvényeket a Képletek szalag Függvénytár mezőjében találjuk. Logikai kifejezések kiértékelésénél használható összehasonlító operátorokról már az előzőekben volt szó. Ebben a részben az alapvető logikai függvényekről lesz szó.



7.16. ábra. Logikai függvények

`HAMIS()`

A `HAMIS` logikai értéket adja eredményül. A függvény alkalmazása helyett egyszerűen beírhatjuk a `HAMIS` szót.

`IGAZ()`

Az `IGAZ` logikai értéket adja eredményül. Az `IGAZ` logikai érték a függvény használata nélkül is bevihető a cellákba és képletekbe, egyszerűen az `IGAZ` érték beírásával.

`ÉS(logikai1;logikai2;...)`

`IGAZ` értéket ad vissza, ha az összes argumentuma `IGAZ`; `HAMIS` értéket ad vissza, ha egy vagy több argumentuma `HAMIS`.

	A	B	C
1	És művelet igazságtáblázata		
2	A	B	A ÉS B
3	0	0	=ÉS(A3;B3)
4	0	1	IGAZ
5	1	0	IGAZ
6	1	1	HAMIS

7.17. ábra. És művelet igazságtáblázata

`VAGY(logikai1;logikai2;...)`

Az `IGAZ` értéket adja eredményül, ha legalább egy argumentumának értéke `IGAZ`; a visszatérési érték `HAMIS`, ha az összes argumentum értéke `HAMIS`.

	A	B	C
1	VAGY művelet igazságtáblázata		
2	A	B	A VAGY B
3	0	0	=VAGY(A3;B3)
4	0	1	IGAZ
5	1	0	IGAZ
6	1	1	IGAZ

7.18. ábra. Vagy művelet igazságtáblázata

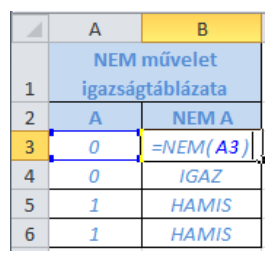
`XVAGY(logikai1, logikai2, ...)`

Az argumentumokból az `XVAGY` művelettel képzett kifejezés eredményét adja vissza. Azaz `IGAZ` értéket adja eredményül, ha pontosan egy argumentumának értéke `IGAZ`; a visszatérési érték `HAMIS` az összes többi esetben.

7.19. ábra. Kizáró vagy művelet igazságtáblázata

`NEM(logikai)`

Az argumentum értékének ellentettjét adja eredményül. A `NEM` függvényt akkor használjuk, amikor biztosítani szeretnénk, hogy egy érték egy megadott értékkel ne egyezzen meg.



7.20. ábra. Nem művelet igazságtáblázata

De Morgan-azonosságok

Az azonosságok a halmazelmélet matematikai logika, illetve a matematikai logika két alapvető tételét fogalmazzák meg. A De Morgan-féle azonosságok felírására a matematikában számos különböző jelölés használatos.
Halmazelmélet formalizmusával leírva:

$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

$$\overline{A \cup B} = \bar{A} \cap \bar{B}$$

ahol A az A komplementerhalmaza, ∩ jelöli két halmaz metszetét és ∪ jelöli két halmaz unióját.
Ítéletkalkulus formalizmusával leírva:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

$$\neg(a \vee b) = \neg a \wedge \neg b$$

A De Morgan-azonosságokat logikai formulákat a következőképpen is leírhatjuk:

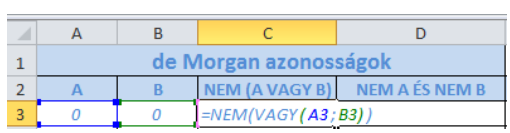
```
NEM(a ÉS b) = (NEM a) VAGY (NEM b)
```

```
NEM(a VAGY b) = (NEM a) ÉS (NEM b)
```

Ez utóbbi leírás hasonlít legjobban az Excel formalizmusához.

7.21. példa. Ebben a példában a De Morgan azonosságokat vizsgáljuk az Excel logikai függvényei segítségével.
A `NEM(A VAGY B)` illetve a `NEM A ÉS NEM B` kifejezések Excel-beli megfelelői:

```
=NEM(VAGY(A3;B3))
```



7.21. ábra. a, De Morgan azonosságok

illetve

```
=ÉS(NEM(A3);NEM(B3))
```

	A	B	C	D	
1	de Morgan azonosságok				
2	A	B	NEM (A VAGY B)	NEM A ÉS NEM B	Ha
3	0	0	IGAZ	=ÉS(NEM(A3);NEM(B3))	

7.22. ábra. b, De Morgan azonosságok

`HA(logikai vizsgálat;érték ha igaz;érték ha hamis)`

A `HA` függvény feltételes vizsgálatok elvégzésére használható értékeken és képleteken. Ha a logikai feltétel által szolgáltatott érték `IGAZ`, akkor a kiértékelést az igaz ágon folytatjuk, különben a Hamis ágon.

7.23. ábra. c, De Morgan azonosságok

	A	B	C	D	E
1	de Morgan azonosságok				
2	A	B	NEM (A VAGY B)	NEM A ÉS NEM B	Ha függvénnyel
3	0	0	IGAZ	IGAZ	Teljesül
4	0	1	HAMIS	HAMIS	Teljesül
5	1	0	HAMIS	HAMIS	Teljesül
6	1	1	HAMIS	HAMIS	Teljesül
7	A	B	NEM (A ÉS B)	NEM A VAGY NEM B	
8	0	0	IGAZ	IGAZ	Teljesül
9	0	1	IGAZ	IGAZ	Teljesül
10	1	0	IGAZ	IGAZ	Teljesül
11	1	1	HAMIS	HAMIS	Teljesül

7.24. ábra. d, De Morgan azonosságok

A Képletek mutatása funkció Képletek Szalag Képletvizsgálat mezőjében a Képletek ikonra kattintva, az eredmények helyett minden cellában a képletet jeleníti meg.

7.3.4. Hivatkozások

A képletekben számokon, műveleti jeleken és függvényeken kívül cellahivatkozások is szerepelnek. Függetlenül a cella aktuális értékétől, mindig a cellában tárolt adatra hivatkozunk. Ezt mint a táblázatkezelő programok egyik legnagyobb előnyét tartjuk számon.

Relatív hivatkozás

A hivatkozásainkat legtöbbször relatív módon adjuk meg. Mit is jelent ez? Relatív címzés esetében a táblázatkezelő a hivatkozott celláról nem azt jegyzi meg, hogy melyik oszlop hányadik sorában található, hanem azt, milyen irányba és hány cellányira van attól a cellától, amibe beírtuk. Ezzel válik lehetővé, hogy a cella másolásakor a formula az ugyanolyan irányban és távolságra lévő cella tartalmával hajtsa végre az előírt műveletet. A megértéshez tekintsük meg a következő példát:

7.22. példa. Írjuk fel az alábbi sorozatok első 5 tagját:

$$a_n = \frac{3^n - 2}{2n - 10}$$

$$b_n = |3 - 2n|$$

Az `A` oszlopot töltsük fel egytől ötig.
A `B2`-es cellába a következő képletet: `=(3^A2-2)/(2*A2-10)`
Ezek után a `B2`-es cella tartalmát másoljuk át `B3`-tól a `B5`-ös cellákba, ekkor

```
B3: =(3^A3-2)/(2*A2-10)

B4: =(3^A4-2)/(2*A4-10)

...
```

lesz.
Mint látható, az összes sorban automatikusan kiszámításra került.

=(3^A4-2)/(2*A4-10)

	A	B
1	n	a_n
2	1	- 1/8
3	2	-1 1/6
4	3	=(3^A4-2)/(2*A4-10)
5	4	-39 1/2
6	5	#ZÉRÓOSZTÓ!
7		

7.25. ábra. Relatív hivatkozás

Abszolút hivatkozás

Gyakran szükséges egy adott képletben megkövetelnünk azt, hogy a cella másolásakor is tartsa meg a címét, ebben az esetben abszolút cellahivatkozásról beszélünk. Ha egy cella címére szeretnénk hivatkozni, akkor az oszlopazonosító és a sorazonosító elé egy "$" jelet kell beírnunk. A "$" jeleket az F4 funkcióbillentyű egyszeri lenyomásával is bevihetjük.

7.23. példa. Egy repülőgép sebessége 3800 km/h. Mennyi idő alatt tesz meg 60, 100, 350, 800 km távolságot?

=A7/$C$2

	A	B	C
1			
2		A repülőgép sebessége (v):	3800 km/h
3			
4	s	t	
5	600 km	1,58E-01 h	
6	1000 km	2,63E-01 h	
7	3500 km	=A7/$C$2	
8	8000 km	2,11E+00 h	

7.26. ábra. Abszolút hivatkozás

Megoldás során figyelnünk kell arra, hogy a repülőgép sebessége egyszer, a `C2`-es cellában van tárolva, ezért a `B5`-ös cellába beírt képlet a következőképpen alakul: `=A5/$C$2`. A képlet másolásakor az `A5`-ös cella értéke mindig az aktuális A oszlopbeli értéket veszi fel, míg a `$C$2` változatlan marad. Abban az esetben, ha egy másik repülőgép esetében is szeretnénk számolni

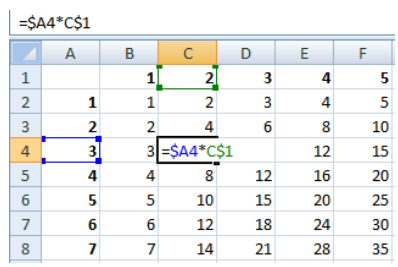
ezeket az értékeket, csak a C2-es cella tartalmát kell módosítanunk, egyéb változtatásokat nem kell végeznünk a táblázatban.

Vegyes hivatkozás

Amennyiben az F4 –es funkcióbillentyűt többször is lenyomtuk, akkor már megfigyelhettük, hogy nem csak relatív és abszolút hivatkozásról beszélhetünk.

Ha a cellakoordináták egyikét abszolút, a másikat relatív címzéssel adjuk meg, akkor ezt a címzési módot vegyes hivatkozásnak nevezzük. Ilyenkor a sort vagy az oszlopot rögzítjük, a többit hagyjuk relatív módon változtatni.

7.24. példa. Egy klasszikus példa a vegyes hivatkozás alkalmazására a szorzótábla. Készítsünk egy 10 x 10-es szorzótáblát. Az A oszlopot és az 1. sort töltsük fel egytől tízig az ábra alapján.



	A	B	C	D	E	F
1		1	2	3	4	5
2	1	1	2	3	4	5
3	2	2	4	6	8	10
4	3	3	=$A4*C$1		12	15
5	4	4	8	12	16	20
6	5	5	10	15	20	25
7	6	6	12	18	24	30
8	7	7	14	21	28	35

7.27. ábra. Vegyes hivatkozás

A B2-es cellába írjuk be a következő képletet: =$A2*B$1

A képletbe írt $A2 azt jelenti, hogy a képletet bárhová is másoljuk be, az adatot mindig az A oszlopból veszi, a B$1 pedig azt, hogy a másik adatot mindig az első sorból. Így az átmásolt cellákba a fejlécként beírt számok szorzata kerül.

S1O1 Speciális hivatkozási stílus

Speciális hivatkozásokat ritkán használunk, szinte csak makróknál. Használat előtt engedélyeznünk kell, amit megtehetünk a Képletekkel végzett munka részben. Ezzel a beállítással lehetőségünk nyílik arra, hogy az osz-

lopok nevei és sorok sorszámai helyett egyszerűbben, számokkal jelöljük az oszlopokat és a sorokat.

7.3.5. Operátorok

Az Excelben használható az operátorokat négy különböző csoportba sorolhatjuk *(aritmetikai, összehasonlító, szövegösszefűző és referencia).*

Aritmetikai operátorok

Matematikai alapműveleteket hajtanak végre.

Aritmetikai operátor	**Jelentése**	**példa**
+	Összeadás	3+3
-	Kivonás, Negáció	3–1–1
*	Szorzás	3*3
/	Osztás	3/3
%	Százalékszámítás	20%
^	Hatványozás	3^2

Összehasonlító operátorok

Két értéket hasonlítanak össze.

Összehasonlító OPERÁTOR	**Jelentése**	**példa**
=	Egyenlő	A1=B1
>	Nagyobb, mint	A1>B1
<	Kisebb, mint	A1<B1
>=	Nagyobb vagy egyenlő, mint	A1>=B1
<=	Kisebb vagy egyenlő, mint	A1<=B1
<>	Nem egyenlő	A1<>B1

Szövegösszefűző operátor

Két vagy több szöveges értékből egyetlen, egyesített szöveges értéket állít elő.

SZÖVEGES OPERÁTOR	Jelentése	Példa
&	Szöveg konkatenáció	"Buda"&"pest"

Hivatkozások

REFEREN-CIA OPERÁTOR	jelentése	példa
:	Tartomány operátor	B5:B15
,	Unió operátor, egyesíti a különböző hivatkozásokat.	SUM(B5:B15, D5:D15)
szóköz	Metszet operátor, a két hivatkozás közös részének celláira hivatkozik.	B7:D7 C6:C8

Operátorok precedencia sorrendje

OPERÁTOR	Leírás
: szóköz ,	Hivatkozások operátorai
-	Negáció, kettes komplemens
%	Százalékszámítás
^	Hatványozás
* és /	Szorzás és osztás
+ és -	Összeadás és kivonás
&	Szöveg konkatenáció
=< ><=>=<>	Feltételes operátorok

Asszociativitás

Az Excel az azonos precedencia szinten lévő műveleteket balról jobbra haladva hajtja végre. Amennyiben ettől a kiértékelési sorrendtől el szeretnénk térni, zárójeleket kell használnunk. Az Excel először a zárójelbe tett kifejezéseket értékeli ki, és a további számításokhoz ezek eredményét használja.

7.3.6. Képletkiértékelő

A képletkiértékelő egy könnyen és jól használható segédeszköz kifejezések kiértékeléséhez. A képletkiértékelő a Képletek szalag Képletvizsgálat mező-

jében található.

7.28. ábra. Képletvizsgálat

7.25. példa. =HA(ÉS(NEM(A5);B5)=VAGY(NEM(A5);
XVAGY(A5;B5));IGAZ;HAMIS)

1. lépés

=HA(ÉS(NEM(*A5*);B5) =
VAGY(NEM(A5);XVAGY(A5;B5));IGAZ;HAMIS)

=HA(ÉS(*NEM*;B5) = VAGY(NEM(A5);XVAGY(A5;B5));IGAZ;HAMIS)

2. lépés

=HA(*ÉS(HAMIS;B5*) =
VAGY(NEM(A5);XVAGY(A5;B5));IGAZ;HAMIS)

=HA(*HAMIS* = VAGY(NEM(A5);XVAGY(A5;B5));IGAZ;HAMIS)

3. lépés

=HA(HAMIS = VAGY(NEM(*A5*);XVAGY(A5;B5));IGAZ;HAMIS)

=HA(HAMIS = VAGY(*NEM(IGAZ);XVAGY(A5;B5));IGAZ;HAMIS)*

4. lépés

=HA(HAMIS = VAGY(HAMIS;*XVAGY(A5;B5)*);IGAZ;HAMIS)

=HA(HAMIS = VAGY(HAMIS;*HAMIS*);IGAZ;HAMIS)

5. lépés

=HA(HAMIS = *VAGY(HAMIS;HAMIS)*;IGAZ;HAMIS)

=HA(HAMIS =*HAMIS*;IGAZ;HAMIS)

6. lépés

=HA(*HAMIS =HAMIS*;IGAZ;HAMIS)

=HA(*IGAZ*;IGAZ;HAMIS)

7. lépés

=HA(IGAZ;IGAZ;HAMIS)

IGAZ

7.3.7. Képleteink hatékony kihasználása

A képleteink hatékony kihasználása érdekében három fontos funkcióval kell megismerkednünk

Számítás: Számítások során alkalmazott képletek és függvények paraméterei lehetnek állandóak, de mutathatnak változó cellatartalmakra is. Alapértelmezés szerint az Excel csak azon képletek kiértékelését végzi el újra, amelyek alapjául szolgáló cellatartalmak módosultak. Nagyobb számításigényű számítások esetében ezt a funkciót kikapcsolhatjuk. Ezek után csak az F9 billentyű lenyomásával, vagy a Képletek szalag Számítás csoportjában lévő számológépre kattintva értékeli ki újra a képleteket.

Pontosság: Az Excel pontosságáról már az előzőekben volt szó, mint ismeretes, 15 számjegy pontossággal végzi el a számításokat.

Közelítés: Közelítéssel tulajdonképpen az aktuális munkafüzet ismételt újraszámításának számát adhatjuk meg. Azt az esetet, amikor egy képletben – közvetlenül vagy közvetve – hivatkozunk a képlet eredményét számító cellára, körkörös hivatkozásnak nevezzük. Ebben az esetben az eredményt az Excel nem tudja automatikusan kiszámítani, de manuálisan beállíthatjuk az ilyen jellegű számítások esetében a közelítések maximális számát illetve az elfogadható változás mértékét.

7.3.8. Hibaértékek képleteknél

Előfordulhat, hogy képleteink értékeit az Excel nem tudja kiszámolni. Tipikus példa erre a nullával való osztás, amely matematikailag nincs értelmezve. Ha a program az eredményt nem tudja kiszámolni, akkor minden esetben `(#)` kettős kereszttel kezdődő hibaüzenetet ad. A szöveges hibaüzenetek, amelyek mindig csupa nagybetűvel jelennek meg, az alábbi kategóriákba sorolhatóak:

`#SZÁM!` Abban az esetben kapunk ilyen hibaértéket, ha a kapott eredmény túl nagy vagy túl kicsi szám, amit már az Excel nem tud értelmezni.

`#ÉRTÉK!` Ezzel a hibaüzenettel akkor találkozhatunk, amikor egy numerikus számot igénylő művelet esetén szöveges értéket adunk meg vagy szöveges értéket tartalmazó cellára hivatkozunk.

`#ZÉRÓOSZTÓ!` Erről már a bevezetőben volt szó, akkor kapunk ilyen hibaüzenetet, ha a képletben osztóként nullát vagy üres cellára történő hivatkozást adunk meg.

`#HIV!` Ilyen hibaüzenettel akkor találkozhatunk, ha a beírt képlet érvénytelen cellahivatkozást tartalmaz. Ez leggyakrabban cellák másolásánál, áthelyezésénél, illetve törlésénél fordul elő.

`#NÉV?` Ezzel a hibaüzenettel akkor találkozhatunk, amikor is a képletben használt cellahivatkozást, függvénynevét vagy cellatartomány nevét a program (általában gépelési hiba miatt) nem ismeri fel.

8. VISUAL BASIC ÉS MAKRÓK

A Visual Basic *(VB)* első verziója *1991*-ben jelent meg *Microsoft Windows 3.0*-ás platformon. Egy eseményvezérelt, nem teljesen objektumorientált (*hiányzik a polimorfizmus és az öröklődés)* programozási nyelv, amely szintaxisát tekintve a *BASIC* nyelvet veszi alapul. Megjelenését követően hamar nagy népszerűségre tett szert, mivel a fejlesztőknek hatékony és gyors eszközt adott a *Windows* alkalmazások vizuális fejlesztéséhez. A cég a kezdeti siker hatására a nyelvet folyamatosan bővítette, az *"utolsó"* verziót *(Visual Basic 6.0) 1998*-ban dobta piacra.

A technológia fejlesztések mozgatórugója a *VBA (Visual Basic for Application)* és a *VBScript (Visual Basic Scripting Edition)* volt. Előbbi az *Office* programcsomag makrónyelve, míg utóbbi a *Windows* operációs rendszer scriptnyelve.

A *Visual Basic-et 2002*-ben felváltotta a *Visual Basic .NET*. A *VB.NET* a *VB*-nek egy teljesen újratervezett, szinte teljesen új változata. A *.NET* keretrendszerhez szorosan igazodva a *VB.NET* is teljesen objektumorientált, szembetűnő a hasonlóság a *C#*-al. Sok kritika érte a nyelvet, mivel alapjaiban változtatták meg, ezért kompatibilitási problémák léptek fel a klasszikus *Visual Basic* nyelvvel. A probléma orvoslására a *Microsoft* készített egy konvertálóprogramot, mely program csak a legegyszerűbb projektek esetén működik.
Előnye, hogy a széleskörű komponenspalettával már kezdő programozók is használhatják a legbonyolultabb *Windows* alatti megoldásokat. A Visual Basic legújabb verziója a *Visual Basic 2010.*

A *Visual Basic for Applications* a *Visual Basic* egyszerűsített változata, tartalmaz egy integrált fejlesztői környezetet, amely be van építve a legtöbb *Microsoft Office* alkalmazásba. A nyelvet alapvetően arra tervezték, hogy más alkalmazásokhoz kiegészítő funkcionalitást biztosítson, például makrók rögzítése és futtatása valamint varázslók készítése.

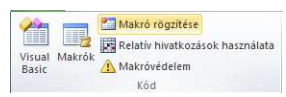
VBA fejlesztők számára lehetővé teszi folyamatok automatizálását és *DLL*-eken keresztül hozzáférést a *Win32* és egyéb alacsony szintű funkciókhoz.

8.1. Makrók

Egy makró nem más, mint műveleteknek rögzített tevékenységsorozatát tartalmazó készlet, melyek *Visual Basic* nyelven kerülnek tárolásra. A makrók segítséget nyújtanak gyakran végrehajtandó feladatok automatikus végrehajtására, de makrók segítségével pl. az Excel funkcionalitását is szélesíthetjük *(saját függvények létrehozása)*.
A makrókat létrehozhatjuk tevékenységsorozatunk rögzítésével vagy megírhatjuk *VBA* nyelven. Természetesen az előbbi két módszert kombinálva is használhatjuk.

8.1.1. Makrók rögzítése

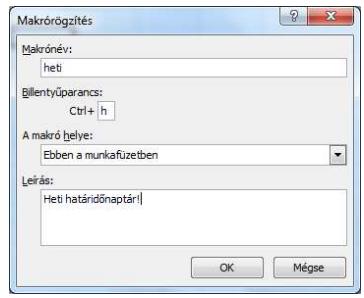
Makrók rögzítésének elindítására vagy a **Fejlesztőeszközök** szalag **Kód** mezőjében vagy a **Nézet** szalag **Makró** mezőjében vagy a Státuszsoron elhelyezkedő **Makró rögzítése** ikonra kattintva van lehetőségünk.



8.1. ábra. Makrók rögzítése

Fontos megjegyezni, hogy makró rögzítésekor csak a kiválasztott funkciók kerülnek rögzítésre. Így nem kerül rögzítésre a különböző szalagokon, menükön történő mozgás.

8.1. példa. Makró rögzítésével készítsünk egy heti határidőnaptárt, amely minden új munkafüzet beszúrásakor lefut és az új munkalapra elkészíti a határidőnaptárt.
Rögzítésekor a makró nevét, billentyűparancsát, helyét és leírását adhatjuk meg.



8.2. ábra. Makrórögzítés

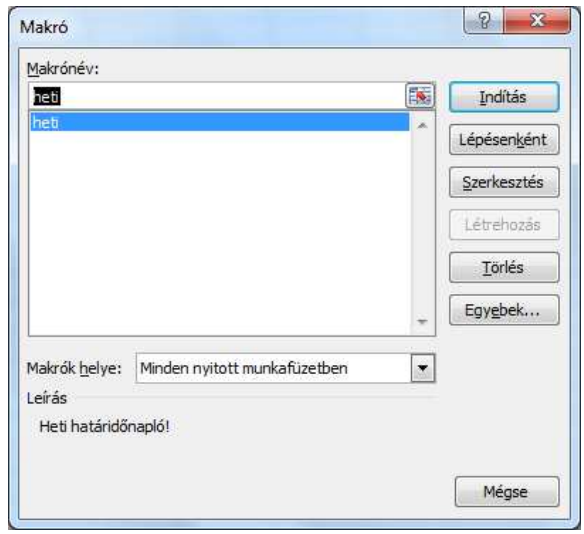
Rögzítés alatt készítsük el az alábbi táblázatot. Az *A1*-es cellába mindig az aktuális hét sorszámát jelenítsük meg.

A1 =HÉT.SZÁMA(MA();21)

	A	B	C	D	E	F
1	31	Hétfő	Kedd	Szerda	Csütörtök	Péntek
2	8:00					
3	8:30					
4	9:00					
5	9:30					
6	10:00					
7	10:30					
8	11:00					
9	11:30					
10	12:00					
11	12:30					
12	13:00					
13	13:30					
14	14:00					
15	14:30					
16	15:00					
17	15:30					
18	16:00					
19	16:30					
20	17:00					
21	17:30					
22	18:00					

8.3. ábra. Heti naptár

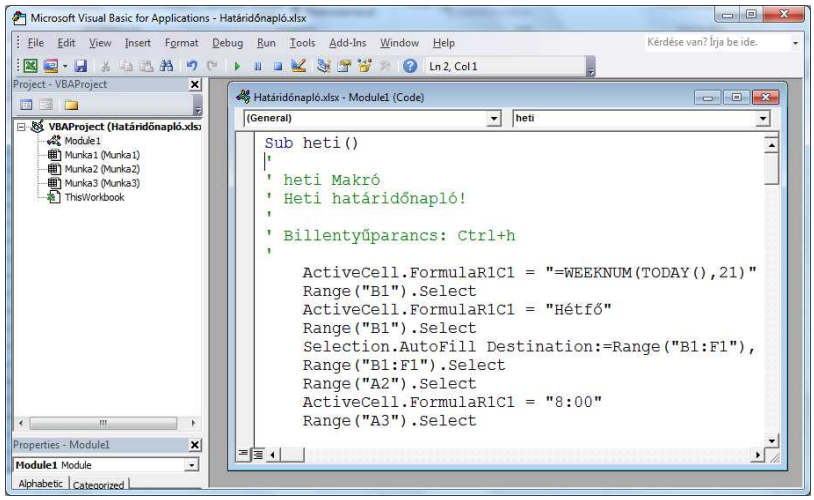
Miután a rögzítést befejeztük, indítsuk el a készített makrót! Látható, hogy gond nélkül lefut, és elkészíti a táblázatot. Ezzel a feladat első részét teljesítettük is, viszont van még egy pont, mégpedig, hogy új munkalap megnyitásakor a makró automatikusan induljon el. Ahhoz, hogy ezt elérjük, az elkészített makrónkon módosítanunk kell. A **Nézet** szalag **Makrók** mezőjében válasszuk a **Makrók** megjelenítése menüpontot. A megjelenő Makró párbeszédpanelen az alábbi menüpontok között választhatunk:



8.4. ábra. Makró párbeszédpanel

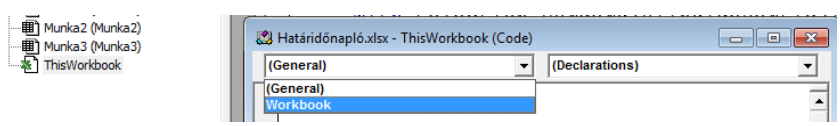
1. Indítás: lefuttathatjuk a kiválasztott makrót;
2. Lépésenként: a kiválasztott makrót soronként futtathatjuk le;
3. Szerkesztés: megjeleníthetjük a Visual Basic szerkesztő ablakot, ahol a makrót módosíthatjuk;
4. Létrehozás: új makrót készíthetünk, az új makrónév beírásakor válik aktívvá a Létrehozás gomb, aminek hatására a Visual Basic szerkesztőablak nyílik meg;
5. Törlés: a kiválasztott makrót törölhetjük;
6. Egyebek: korábban már létrehozott makróhoz billentyűkombinációt rendelhetünk és megadhatunk hozzá egy leírást;
7. Mégse: bezárhatjuk a párbeszédpanelt.

Amennyiben a **Szerkesztés** gombra kattintunk, a *Visual Basic* szerkesztőben megjelenik a makrónk *VBA* kódja. Váltás szerkesztőnézetbe (*Visual Basic Editor*-ba) az *Alt + F11* billentyűkombinációval is történhet.

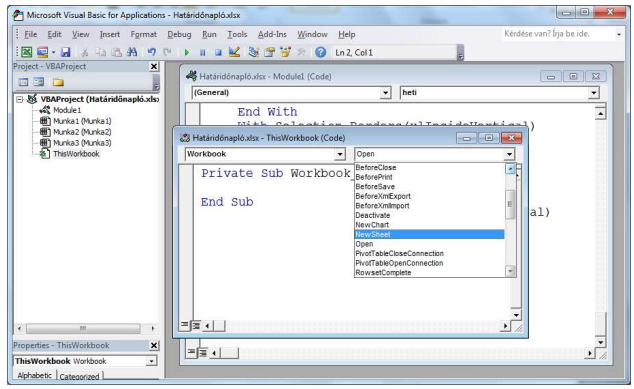


8.5. ábra. VBA Editor

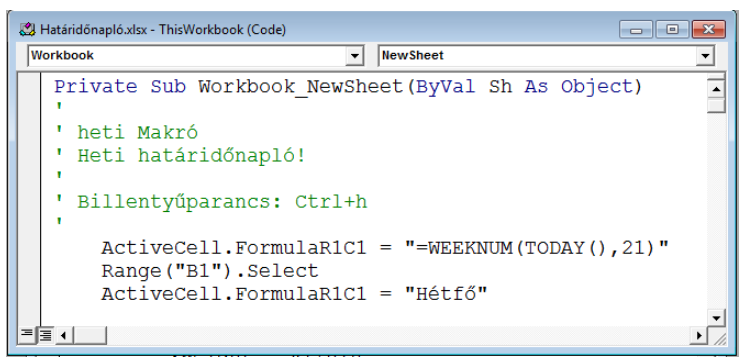
A *VBA* kódról továbbiakban még részletesebben lesz szó, most csak annyi ismeretet közlünk, amennyi a feladat megoldásához szükséges.



8.6. ábra. ThisWorkbook- Workbook



8.7. ábra. a, Newsheet



8.8. ábra. b, Newsheet

8.2. Visual Basic Editor kezelőfelülete

Sajnos a *Visual Basic* kezelőfelülete angol nyelvű magyar nyelvű *Office* esetén is. A kezelőfelület részei:

1. Project ablak: A projekt ablakban találhatjuk a megnyitott dokumentumaink *(Excel esetében munkafüzetek)* elemeit. Az elemek tárolása fa szerkezetű. Minden egyes *VBA Projekt (munkafüzetnév)* egy-egy excel munkafüzetet jelent, amelyen belül megtaláljuk a hozzátartozó munkalapokat és a

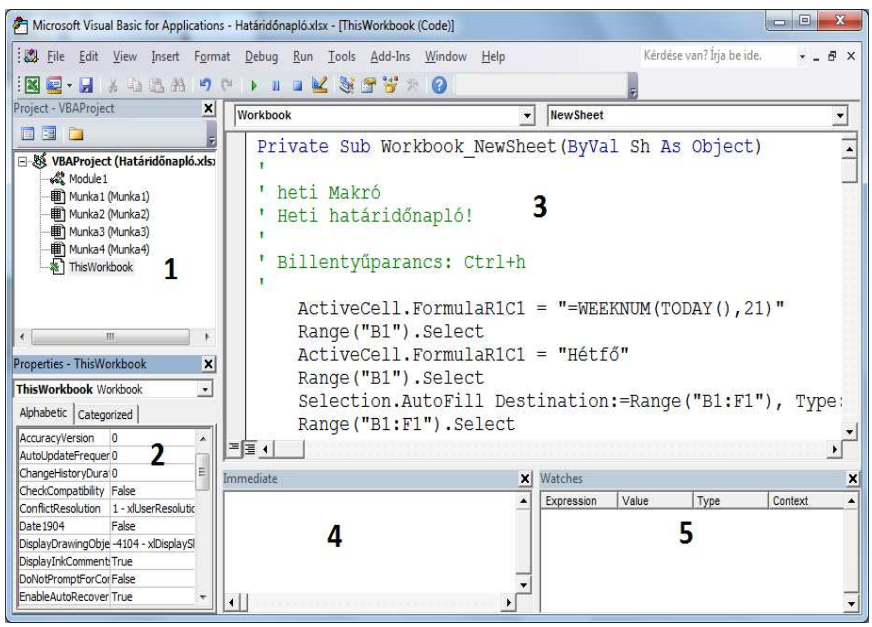
ThisWorkbook objektumot, ami az egész munkafüzetet együtt jelenti.

2. Properties ablak: A Properties ablakon a kiválasztott objektumok tulajdonságait állíthatjuk be.

3. Szövegszerkesztő *(Code)* ablak: A szövegszerkesztő ablakon belül készíthetjük el a programunk kódját.

4. Futtatási *(Intermediate)* ablak: Az intermediate ablakban a programunk futási eredményeit vizsgálhatjuk meg.

5. Watch ablak: A watch ablakban programunk nyomkövetésére és hibakeresésére van lehetőségünk.



8.9. ábra. Visual Basic for Applications Editor kezelőfelülete

8.3. VBA

8.3.1. Láthatósági körök

A *VBA*-ban kétféle érvényességi *(láthatósági)* kört különböztethetünk meg:

1. *eljárásszintű*

2. *modulszintű*

Eljárásszintű láthatóság esetében a változót kívülről nem érhetjük el, annak a metódusnak a sajátja, amelyben deklarálva van. A deklarációs kulcsszó a `Dim` vagy `Static`. A két deklaráció között a különbség a memóriahasználatnál van, míg `Static` kulcsszóval deklarált változó értéke az alprogram lefutása után a memóriában marad, addig a `Dim` kulcsszóval deklarált változó értéke megszűnik.
Modulszintű érvényességi kör esetében `Private` és `Protected` és `Public` érvényességi szintű változókat deklarálhatunk. `Private` esetben a változó csak az őt tartalmazó modulból érhető el, `Public` esetben pedig bárhonnan.

Megjegyzés: A típusok deklarálása sem eljárások, sem függvények esetében nem kötelező, az első értékadás határozza meg a változók típusát.

8.3.2. Paraméterátadás

Paramétereket VBA-ban cím szerint `(ByRef)` és érték `(ByVal)` szerint tudunk átadni.

1. Érték szerinti `(ByVal)` paraméterátadással tudjuk megadni az ún. bemenő paramétereket. A híváskor hely foglalódik le a memóriában a változó számára, ahova a paraméterként kapott értéket bemásolja. Ha a függvény vagy eljárás megváltoztatja ennek értékét, akkor a hívó programba visszatérve ez a megváltoztatott érték elveszik, marad a régi érték. Az eljárás vagy függvény lefutása után a paraméter számára lefoglalt memóriaterületek felszabadulnak.

2. Cím szerinti `(ByRef)` paraméterátadással tudjuk megadni az ún. kimenő paramétereket. A híváskor a paraméter címe kerül átadásra. A cím alapján a függvény vagy eljárás megváltoztathatja ennek értékét úgy, hogy a hívó programba visszatérve ez a megváltoztatott érték megmarad. Ilyen típusú paraméter az eljárás törzsében, legalább egyszer az értékadó utasítás bal oldalán foglal helyet.

Megjegyzés: Amennyiben elhagyjuk a paraméter típusát, `ByRef` az alapértelmezés.

8.3.3. Alprogramok

Eljárás

Mivel a *VBA* egy objektumorientált nyelv, utasításainkat eljárásokba és függvényekbe szervezhetjük. Minden egyes eljárást a `Sub` utasítással kell kezdenünk, majd a makró nevét kell megadnunk, és végül zárójelek között a futáshoz szükséges paraméterlistát. Az eljárásokat az `End Sub` utasítással zárjuk.

Eljárások szintaxisa:

```
[Private|Public] Sub <eljárásnév>([paraméterlista]) [As típus]
    [<deklarációk>]
   <utasítások>
End Sub
```

Eljárás hívása:

```
[Call] <eljárásnév> ([aktuális paraméterlista])
```

8.2. példa. Csere

```
Sub csere(a As Double, b As Double)  két érték cseréje
   Dim w As Double
   w = b: b = a: a = w
End Sub
```

Függvény

Excelben könnyedén készíthetünk saját függvényeket. Minden egyes függvényt a `Function` utasítással kell kezdenünk, majd a függvény nevét kell megadnunk, végül zárójelek között a futáshoz szükséges paraméterlistát. A függvényeket az `End Function` utasítással zárjuk. Minden függvénynek kötelezően tartalmazni kell legalább egy olyan értékadó utasítást, amelynek bal oldalán a függvény neve szerepel.

Függvények szintaxisa:

```
[Private|Public] Function <függvénynév>
([paraméterlista])[As típus]
    [<deklarációk>]
    <utasítások>
    függvénynév = (legalább egy értékadó utasítás)
End Function
```

Függvény hívása:
`<függvénynév> ([aktuális paraméterlista])`

8.3. példa. Faktoriális

```
Function faktorialis(ByVal n As Integer) As Integer
Dim i As Integer, s As Integer
s = 1
   For i = 1 To n
    s = s * i
   Next
faktorialis = s
End Function
```

8.4. példa. Készítsünk egy függvényt és egy eljárást a fajlagos ellenállás kiszámítására.
Fajlagos ellenállás

$$\rho = R\frac{A}{l}$$

ahol ρ a fajlagos ellenállás ohmméterben, l a vezető hossza méterben, A a vezető keresztmetszete m^2-ben és R a vezető ellenállása ohmban.
Függvény

A függvény neve: Fajlagos
Bemenő (`ByVal` – érték szerinti) paraméterei:

`l` – a vezető hossza,

`a`– a vezető keresztmetszete,

`r` – a vezető ellenállása.

Mivel a függvény a nevében adja vissza az eredményt, azaz a fajlagos ellenállást, a függvény neve típusának meg kell egyeznie az eredmény típusával. A bemenő paraméterek és a fajlagos ellenállás eredménye legyen `Double` típusú, a függvény deklarációja:

```
Function Fajlagos(ByVal l As Double, ByVal a As Double,
ByVal r As Double) As Double
```

A függvény törzsében a Fajlagos függvény nevének egyszer az értékadó utasítás bal oldalán kell szerepelni, hogy átvegye az eredmény értékét:

```
Fajlagos = r * a / l
```

A függvény utolsó utasítása az `End Function`.

Eljárás

Az eljárás neve: Fajlagos
Bemenő (`ByVal` – érték szerinti) paraméterei:

`l` – a vezető hossza,

`a` – a vezető keresztmetszete,

`r` – a vezető ellenállása.

Kimenő (`ByRef` – cím szerinti) paramétere:

`FajlEll` – fajlagos ellenállás.

eljárás deklarációja:

```
Sub Fajlagos(ByVal l As Double, ByVal a As Double,
ByVal r As Double, ByRef FajlEll As Double)
```

Az eljárás törzsében a kimenő paraméter az értékadó utasítás bal oldalán átveszi a fajlagos ellenállás értékét:

```
FajlEll = r * a / l
```

Az eljárást az `End Sub` zárja.

8.3.4. Konstansok és Változók

A konstansok létrehozásával egy nevet rendelhetünk konkrét értékekhez. A konstansok értékét nem lehet megváltoztatni.

Konstansok deklarációja
`[Private | Public] Const` ***`név`*** `As` ***`típus`*** `=` ***`érték`***

A változók olyan programelemek, amelyek a program futása során többször változtathatják értéküket. A deklarációkban megadott típusnak megfelelő értékkészlet halmazból újabb és újabb értéket vehetnek fel. A változók a megadott az alaphalmazból vehetnek fel értékeket.
Változók deklarációja
`{Dim|Public|Private|Protected|Static} vá1tozó1 <adattípus>, változó2 As <adattípus>`
Néhány adattípus:

- `Integer(%)` *egész szám*
- `Single(!)` *lebegőpontos szám*
- `Double(#)` *duplapontosságú lebegőpontos szám*
- `Decimal` *decimális szám*
- `String($)` *karaktersorozat*
- `Boolean` *logikai (*true, false*)*
- `Date` *dátum*
- `Variant` *szöveg, szám vagy dátum (automatikus típus)*
- `Object` *objektum*

8.3.5. Elágazások

Programunk készítése során gyakran előfordul, hogy az utasítások végrehajtását egy feltételhez kötjük. Tehát egy vagy több feltételvizsgálat után döntjük el, hogy mely utasítást vagy utasításokat hajtjuk végre.

Elágazás szintaxisa:

```
If 1. feltétel Then
   1. utasítások (ha a 1. feltétel igaz)
[ElseIf 2. feltétel
   2. utasítások] (amennyiben a 2. feltétel igaz)
[Else
   3. utasítások] (minden egyéb esetben)
End If
```

Feltételek összeállításához használhat operátorok:

relációs operátorok: < > = <= >= <>

logikai operátorok: `And, Or, Not, Xor`

Értéktől függő elágazás

Abban az esetben használhatjuk, amikor egy változó értékétől függően kell különböző utasításokat végrehajtani.

Értéktől függő elágazás szintaxisa:

```
Select Case <kifejezés>
   [Case <1. kifejezés>
     <1. utasítások>]
    [Case <2. kifejezés>
    <2. utasítások>]
     ...
   [Case Else
     <különben utasítások>]
End Select
```

A `Case Else` ágban adhatjuk meg azokat az utasításokat, amelyek abban az esetben hajtódnak végre, amikor egyik `Case` ágnak sem volt lehetősége lefutni.

8.3.6. Ciklusok

Ciklusok használatával ugyanazt az utasítássorozatot többször is megismételtetjük a programmal. A ciklus szervezése szempontjából megkülönböztethetünk elöltesztelő, hátultesztelő és számláló ciklust.
Amennyiben előre tudjuk, hogy hányszor szeretnénk futtatni az utasítássorozatot, akkor a számláló ciklust célszerű használnunk.
A számláló ciklus szintaxisa:

```
For ciklusváltozó =ettől To eddig [Step +/-lépésköz]
[utasítások]
Next[ciklusváltozó]
```

Amennyiben nem tudjuk előre megmondani, hogy hányszor kell lefuttatnunk az utasítássorozatot, akkor a feltételes ciklusok valamelyikét kell használnunk. Egy feltételes ciklus lehet elől- illetve hátul-tesztelő.
Elöltesztelő ciklusnál a feltétel kiértékelése a ciklus elején a `Do While` kulcsszavak után történik. Elől-tesztelő ciklus esetében a feltétel teljesülése esetén futnak le a ciklusmagban szereplő utasítások.
Elöltesztelő ciklusok szintaxisa:

```
Do While feltétel
   [utasítások]
   Loop
```

Hátultesztelő ciklusnál a feltétel kiértékelése a ciklus végén a `Loop Until` kulcsszavak után történik. Mindaddig történik az ismétlés, amíg a feltétel hamis. A hátultesztelő ciklus esetében a ciklusmagban lévő utasítások legalább egyszer biztosan lefutnak.

Hátultesztelő tesztelő ciklusok szintaxisa:

```
Do
   [utasítások]
Loop Until feltétel
```

8.3.7. Tömbök

Az eddig megismert változók csak egyetlen értéket voltak képesek tárolni. Mivel Excellel dolgozunk, különösen felmerül az igény arra, hogy egyszerre nagy mennyiségű adatot tudjunk kezelni *(egy egész sort vagy oszlopot, esetleg egy tartományt).* A tömb nem más, mint azonos típusú adatok összetartozó sorozata.

Tömb deklarálása:

```
{Dim |Public |Private |Protected |Static} tömbnév
[(tömbindex)] [As <adattípus>] [ = kifejezés]
```

Tömb egy elemére az indexének megadásával hivatkozhatunk. Pl. egy x tömb esetében x(2) utasítással a tömb 3. elemére hivatkozhatunk. Az előző példából is látható, hogy a VBA-ban tömbök indexelése 0-tól kezdődik.

`Dim a(9) As Double` Egy 10 elemű `Double` típusú tömböt hoz létre.

Többdimenzs tömbök deklarációja:

```
Dim tömbnév(maxindex1, [maxindex2], ..., [maxindex60])
```

`Dim pont(999,999) As Integer` Egy Egész típusú elemekből áll kétdimenziós tömböt hoz létre, amely egy *1000 x 1000* –es táblázat kezelésére szolgál.

`Dim abc()` Egy dinamikus tömböt hoz létre.

Tömbök létrehozására értékadásnál is van lehetőség.

```
honapok= Array(" Január ", " Február ", " Március ",
" Április ", " Május ", " Június ", " Július ", " Augusztus ",
" Szeptember ", " Október ", " November ", " December ")
```

8.3.8. Megjegyzések

Megjegyzéseket a program kódjába a sor elején vagy akár sorban elhelyezett aposztróffal *(')* tehetünk. A *Visual Basic* zöld színnel jelöli a megjegyzéseket. Célszerű minél több megjegyzést elhelyezni a programkódban, hiszen ezzel egyrészt áttekinthetőbbé tehetjük, másrészt megkönnyíthetjük a kód későbbi esetleges újraértelmezését.

8.3.9. Üzenőablakok

A program futása során a felhasználóval történő kapcsolattartásnak a legpraktikusabb módja üzenőablakokon keresztül történik.
Adatbekérésre az `InputBox` függvényt, míg üzenetek, válaszok küldésére a `MsgBox` függvényt használhatjuk.
Az `InputBox` szintaxisa:
`változó = InputBox("szöveg","címke")`
A `MsgBox` szintaxisa:
`változó = MsgBox ("üzenet", paneltípus)`

8.3.10. Paneltípusok

1. `vbOKOnly` OK gomb
2. `vbOKCancel` Ok és Mégse gombok
3. `vbAbortRetryIgnore` Leállítás, Ismét és Kihagyás gombok
4. `vbYesNoCancel` Igen, Nem és Mégse gombok
5. `vbYesNo` Igen és Nem gombok

8.4. Az Excel objektumainak metódusai és tulajdonságai

Az Excel objektumai *(munkalap, cella, tartomány, stb. . .)* rendelkeznek tulajdonságokkal és metódusokkal, ezeket úgy tudjuk megadni, hogy az objektumra vonatkozó parancs után ponttal elválasztva írjuk a tulajdonságot, illetve a metódust.
Tulajdonság
Az `A1`-es cellában félkövér betűstílus beállítása: `Cells(1,1).Font.Bold = True`
Metódus
Az `A1`-es cella kijelölése: `Cells(1,1).Select`

8.4.1. Munkalapok, tartományok, cellák

Tartomány *(cella)* azonosítására a `Cells(sorszám, oszlopszám)` vagy a `Range (cellaazonosító)` parancsokkal van lehetőségünk.
Az `A9`-es cella a `Range("A9")` formulával azonosíthat. Az `A9`-es cellát a `Cells` paranccsal a `Cells(9,1)` módon tehetjük meg.
Aktuális cellára az `ActiveCell` paranccsal hivatkozhatunk, amennyiben az aktív cellát módosítani szeretnénk `Range(cellaazonosító).Select` formulát kell használnunk.

```
Range("B10").Select
```

Ha egy egész tartományt szeretnénk kijelölni, akkor azt a
`Range("A1:B10").Select` formulával tehetjük meg. Természetesen az egyes parancsokat egymásba is ágyazhatjuk. Az előző tartománykijelöléssel teljesen ekvivalens a
`Range(Cells(1,1),Cells(10,2)).Select` kifejezés.
Teljes sorra vagy oszlopra a `Rows(sorazonosító)`
illetve a `Columns (oszlopazonosító)` parancsokkal hivatkozhatunk.

Sorok és oszlopok átméretezésére is van lehetőségünk. Oszlopok esetében a `Columns(oszlopazonosító).ColumnWidth = új_méret` illetve sorok esetében a `Rows(sorazonosító).RowHeight = új_méret` utasításokkal.
A `B` oszlop szélesség értékének `24`-re történő beállítása a `Columns("B") .ColumnWidth = 24`, míg az első három sor magasságának `20`-as értékre történő beállítása a `Rows(1:3).RowHeight =20` kifejezéssel történhet.
A `B` oszlop kijelölése a `Columns(2).Select` vagy a `Columns("B").Select` utasításokkal érhető el. Az `A,C,E` oszlopokból álló tartományt pedig a `Range (Columns(1), Columns("C"), Columns(5)` kifejezéssel jelölhetjük ki. Sorok esetében a `Rows` parancs argumentumába csak sorszámok kerülhetnek. A `4.` sort a `Rows(4). Select` paranccsal tudjuk kijelölni. Egy munkalap összes celláját a `Cells.Select` paranccsal jelölhetjük ki.

Sorok és oszlopok beszúrására is szükségünk lehet. Ilyenkor figyelnünk kell

arra, hogy sor beszúrás esetén az aktuális cella fölé, míg oszlop beszúrásakor az aktuális cella elé történik a beszúrás.
A `C5`-ös cella fölé új sor beszúrásához a `Range("C5").EntireRow.Insert` utasítást, míg a cella elé egy új oszlop beszúrásához a `Range("C5").EntireColumn.Insert` utasítást használhatjuk.

Adott sor illetve oszlop törlésére a `Range("cellaazonosító"). EntireRow. Delete` illetve a `Range("cellaazonosító ").EntireColumn. Delete` utasításokat használhatjuk.

Más munkalapokon lévő cellára a `Range("munkalapnév!cellaazonosító")` kifejezéssel tudunk hivatkozni. pl. `Range("Munka3!C3")`
Munkalapokra a `Worksheets(munkalap sorszáma vagy neve)` paranccsal hivatkozhatunk. A `Worksheets(3).Cells(4,1)` formulával a harmadik munkalap `A4`-es cellájára hivatkozhatunk. Természetesen a munkalap nevével is hivatkozhatunk, ebben az esetben a `Worksheets("Bevétel").Cells(4,1)` kifejezéssel a `Bevétel` munkalap `A4`-es cellájára hivatkozhatunk.

A makróknál eddig használt hivatkozások mindegyike abszolút, amennyiben relatív hivatkozásra van szükségünk a `Selection.Offset(sor, oszlop)` formulát kell használnunk. A `Selection.Offset(5, 2)` paranccsal az 5 sorral lejjebb és 2 oszloppal jobbra lévő cellára hivatkozhatunk.
Munkalapot a `Sheets("munkalapnév" vagy sorszám).Select` utasítással tudunk váltani.

Cellának értéket a `Value` tulajdonsággal lehet adni, mivel azonban ez az alapértelmezett tulajdonság, ezért nem kötelező kiírni. A következő négy értékadás egymással teljesen ekvivalens.

```
Cells(1,1).Value = "Excel 2010"
Cells(1,1) = "Excel 2010 "
Range("A1") = "Excel 2010 "
ActiveCell = "Excel 2010 "
```

***Megjegyzés**: Abban az esetben, ha* `A1` az aktív cella.

8.4.2. Formázások

Természetesen makróból is van lehetőségünk cellaformázások elvégzésére. A formázás gyorsabb és pontosabb lehet, ha makrórögzítéssel végezzük el, és azt követően illesszük a kódba.
Formázást a formázandó tartomány előzetes kijelölésével vagy a konkrét tartományhivatkozás megadásával is elvégezhetjük. Első esetben a tartományhivatkozás helyett a `Selection` utasítást is használhatjuk.

`Range("A1:B3").Select`, majd
`Selection.Font.Size = 18`
vagy
`Range("A1:B3").Font.Size = 18`

A Font tulajdonsággal a cella betűtípusát módosíthatjuk. Újabb ponttal elválasztva lehet megadni stílust, típust, színt, stb. ...

```
.Name = "Betűtípus_neve"
.Size = betűméret
.Bold = True/False
.Italic = True/False
.Shadow = True/False
.Underline = aláhúzástípus
```

Amennyiben több formázást is szeretnénk egyszerre alkalmazni, akkor `With` és `End With` utasítások közé kell tenni a formázó sorokat.

```
With Selection.Font
   .Bold = True
   .Size = 14
End With
```

A háttérszín beállításához az `Interior` tulajdonságot használhatjuk.
`Interior.Color` esetében a cella hátterének színét szövegesen vagy színkóddal adhatjuk meg. Szöveges megadás esetében csak angol színelnevezéseket használhatunk úgy, hogy a színek elé kell írni, hogy vb.

`Cells(1,1).Interior.Color = vbGreen`

`Interior.ColorIndex` esetében a cella hátterének színét számmal adhatjuk meg. `Cells(1,1).Interior.ColorIndex = 3`

Cellák tartalmának vízszintes igazításának beállítására a `HorizontalAlignment`, függőleges igazítására a `VerticalAlignment` tulajdonságokat használhatjuk.

```
Cells(1,2). HorizontalAlignment = xlCenter
Cells(1,2).VerticalAlignment = xlCenter
```

A vízszintes igazításnál használható további értékek:

1. `xlLeft` igazítás balra
2. `xlRight` igazítás balra
3. `xlGeneral` igazítás általánosan

A függőleges igazításnál használható további értékek:

1. `xlTop` igazítás felülre
2. `xlBottom` igazítás alulra

Számformátum beállítására a `NumberFormat` tulajdonságot használhatjuk. A `B1`-es cellába három tizedesjegyet tartalmazó számok formátumát a
`Cells(1,2).NumberFormat = "0.000"`
kifejezéssel állíthatjuk be.
Az idézőjelek közé a számformátumnál megismert formátumkódot kell beírnunk.

8.4.3. Fájlműveletek

Makrók segítségével a fájlműveletek is biztonságosan elvégezhetők. Munkalap mentése az `ActiveWorkbook.Save`, mentés másként az `ActiveWorkbook.SaveAs`, bezárása az `ActiveWorkbook.Close` utasítással végezhető.
Munkafüzet megnyitása a `WorkBooks.Open Filename:= "[elérési_út\]fájlnév"` utasítással nyitható meg.

```
WorkBooks.Open Filename:= "C:\excel\makro.xlsx"
```

Munkalap nyomatása az `ActiveWindow.SelectedSheets.PrintOut` utasítással végezhető el. Amennyiben több példányszámban szeretnénk nyomtatni, meg kell adnunk a másolatok számát is, ezt a `Copies:=példányszám` utasítással tehetjük meg.

```
ActiveWindow.SelectedSheets.PrintOut Copies:=10
```

8.4.4. Diagramok készítése

Makrók segítségével a könnyedén készíthetünk diagramokat is.

8.5. példa. Kukoricatermelés Magyarországon

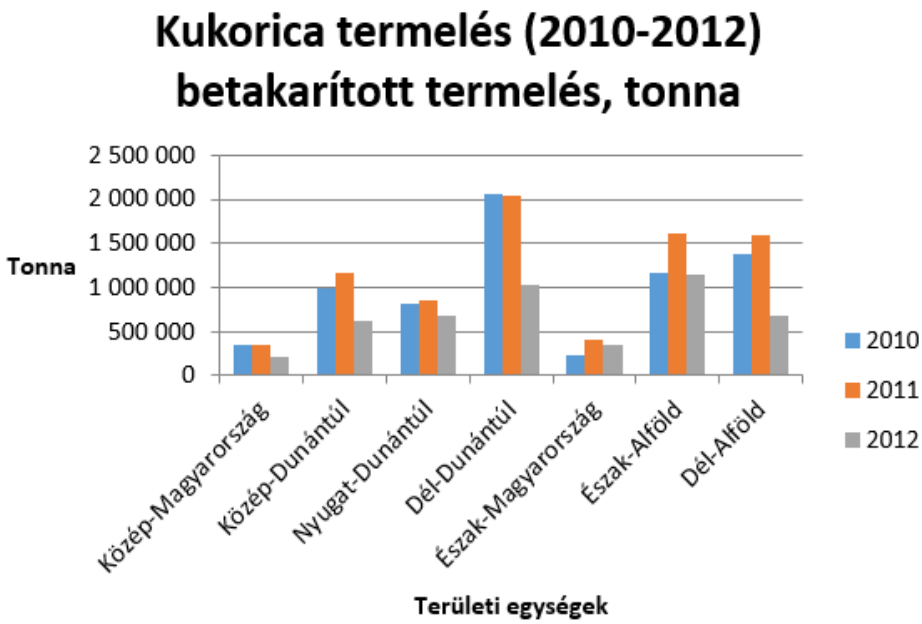
Kukorica termelés Magyarországon (2010-2012) *beta-karított termelés, tonna*			
Területi egység	**2010**	**2011**	**2012**
Közép - Magyarország	**351 292**	**353 198**	**217 607**
Közép-Dunántúl	**998 838**	**1 155 604**	**619 314**
Nyugat-Dunántúl	**805 996**	**851 923**	**683 301**
Dél-Dunántúl	**2 066 106**	**2 033 049**	**1 036 953**
Észak- Magyarország	**225 348**	**405 554**	**357 427**
Észak-Alföld	**1 156 680**	**1 604 040**	**1 145 353**
Dél-Alföld	**1 380 612**	**1 589 075**	**681 538**

Forrás www.ksh.hu

VBA kód

```
Sub Diagram()
ActiveSheet.Shapes.AddChart.Select 'aktív munkalap kijelölése grafikon hozzáadásához
```

```
With ActiveChart
 .SetSourceData Source:=Range(Cells(3,2), Cells(9,4))
'adatforrás
 .ChartType = xlColumnClustered 'Diagram típusának megadása -
oszlopdiagram
 .SeriesCollection=(1).XValues = ActiveSheet.Range(Cells(3, 1),
Cells(9, 1)) 'X tengely értékei
 .SeriesCollection=(1).Name = ActiveSheet.Cells(2, 2)
 .SeriesCollection=(2).Name = ActiveSheet.Cells(2, 3)
 .SeriesCollection=(3).Name = ActiveSheet.Cells(2, 4)
 .SetElement (msoElementChartTitleAboveChart) 'A diagram címé-
nek elhelyezkedése - címe a grafikon felett legyen
 .SetElement (msoElementPrimaryCategoryAxisTitleAdjacentToAxis)
'közel legyen a tengelyhez
.SetElement (msoElementPrimaryCategoryAxisTitleHorizontal) ' víz-
szintesen helyezkedjen el
 .Axes(xlCategory).AxisTitle.Text = "Területi egységek"
 .SetElement (msoElementPrimaryValueAxisTitleHorizontal)
 .Axes(xlValue).AxisTitle.Text = "Tonna"
 .ChartTitle.Text = ActiveSheet.Cells(1, 1) 'diagram címe
 .Location Where:=xlLocationAsNewSheet 'diagram helye
 End With
 End Sub
```



8.10. ábra. Diagram

9. FÜGGELÉK

Kód megfejthetősége (félreérthetősége) történelmi példán keresztül[9]

Forrás: Jókai Mór: A magyar nemzet története regényes rajzokban, Gertrúd

Soha sem tesz annyi kárt egy gyönge népben egy erős zsarnok, mint egy erőteljes népben egy gyönge király.

Jeruzsálemi András egész tizenhét évi uralkodása alatt nem tett egyebet a magyar nemzet, mint saját sírját ásta.

A király pazar, a nemzet koldus, kívül szükségtelen harc, belül pártháború.

Az önakarattalan királyt majd büszke, nagyravágyó nők, majd önző, alacsony lelkű tanácsosok kormányozzák, s ha mindeniktől megszabadult – saját önállástalan lelke.

Első neje, meráni Gertrud, kit a természet nem királynénak, hanem királynak teremte, de semmi esetre sem a magyar számára: büszkesége, pazarlása s idegen udvara által ellenségévé tette trónjának az ország minden rendjeit.

A főpapokat és nemeseket bosszantá, hogy minden rokonát, tanítóját, udvarmesterét, gyóntatóját érseki, báni, főispáni hivatalokba rakta, s a köznép nyögött a vérét sajtoló adó terhe alatt.

A magyar zúgolódva látta magát mindenében megraboltatni: hivatalaiban, rangbüszkeségében, vagyonában és életében, csak egy csepp kelle még a bosszú poharához, hogy kicsorduljon.

E csepp volt a női erény könnye.

Ami a Tarquiniusokat megbuktatá, az lőn Gertrudnak veszte is.

Az akkori nádor Bór Benkének, kit ismertebb néven Bánk bánnak neveznek, csudaszép neje volt a királyné udvarában, ki iránt Ottó, Gertrud testvére, tiltott szerelmet kezde érzeni.

A szép nő szebb volt erényei miatt. A magyar nők egyik főtulajdona volt

[9] Az ötlet a Vassányi István: Információelmélet című jegyzete alapján született.

eleitől fogva a hűség, szűziesség, és itt a választás nem volt nehéz a délceg, daliás nádor s az idétlen meráni herceg között.

Ottót Németországból az igazság keze üldözé nénje udvarába. Fülöp király orgyilkosai közt őt is megismerék. S aki értett az orgyilokhoz, értett a méregkeveréshez is. Egy este, midőn a királynéval s a szép Melindával együtt vacsorált, a nádor nejének poharába szerelemitalt vegyíte, s a királyné saját szobájában egyedül hagyta a herceget Melindával.

A nádor, ki éppen akkor tért vissza ítéletosztó körútjából, a kétségbeesés könnyei közt, félőrülten találta hitvesét, s míg szemei kiolvasták e könnyekből a helyrehozásra nem, csak megtorlásra váró eseményt, nejének rokonai, Mikhál, Simon és Petur bánok ujjal mutattak a bosszú tárgyára.

Ez Gertrud volt.

Rég el volt határozva a királyné halála az összeesküvők által az általuk felszólított esztergomi érsek, János, kérdésükre e dodonai kétértelműségű feleletet adta:

„Reginam occidere nolite timere bonum est; si omnes consentiunt ego non contradico."

Melyet a különböző megszakítás szerint így is lehet érteni:

„A királynét megölni nem kell – félnetek jó lesz; ha mindenki beleegyez – én nem – ellenzem."

De emígy is lehet magyarázni:

„A királynét megölni nem kell félnetek – jó lesz; ha mindenki beleegyez, én nem ellenzem."

De a megsértett férj bosszúja nem kérd és nem hallgat meg tanácsot. Midőn a király éppen Halicsban volt hadat viselni s országát azalatt Bánk bánra bízta, ez a királynét saját palotájában meggyilkolá. Ottó megmenekült, meggyilkolt testvére kincseit is magával elrabolva.

A visszatérő király az összeesküvőket családostul kiirtatá; egyedül Bánkot, neje gyilkosát nem volt bátorsága megöletni. Érzé: hogy a meggyalázott nő miatti keserv nagyobb, mint a megölt miatti. (Bánk bán történetét örökíté meg Katona József hasonló című drámájában, mely elsőrendű remekműve a magyar irodalomnak.)

Forrás: http://www.mek.iif.hu/porta/szint/human/szepirod/magyar/jokai/osszes/ht

Irodalomjegyzék

[1] Richard B. Wells: Applied Coding and Information Theory for Engineers, Prentice Hall, 1999.

[2] Linder Tamás, Lugosi Gábor: Bevezetés az információelméletbe, Műegyetemi Kiadó. 1997, jegyzetszám: 51445. Precízebb matematikai alapok.

[3] Steven Roman: Introduction to Coding and Information Theory, Springer, 1997. Kiegészítő anyag.

[4] Perge Imre: Bevezetés az informatikába, Eszterházy Károly Főiskola jegyzete, 1993.

[5] Vassányi István: Információelmélet, Kivonatos jegyzet, Pannon Egyetem, http://www.irt.vein.hu/~vassanyi/info/infojegyzet.pdf, 2002-2005.

[6] Tanenbaum, A.S. and Woodhull, A.S.: Operating Systems: Design and Implementation; Pearson Education International, 2009.

[7] Roger Seeck: BinaryEssence, http://www.binaryessence.com/axf/ About.htm, 2012.

[8] Frigg, R. and Werndl, C. "Entropy – A Guide for the Perplexed". In Probabilities in Physics; Beisbart C. and Hartmann, S. Eds; Oxford University Press, Oxford, 2010.

[9] Rich Baldwin: Classical Information Theory (Shannon) – Talk Origins Archive,
http://www.talkorigins.org/faqs/information/shannon.html#Entropy, 2005.

[10] Shannon, Claude E. (July/October 1948). "A Mathematical Theory of Communication". Bell System Technical Journal 27 (3): 379–423.

[11] Kovács Emőd: A magyarországi informatikus BSc programmok az ACM 2005-ös informatikai oktatási programjai tükrében, AgriaMedia konferencia kötet, 196-205 o., 2006.

[12] Biró Csaba, Kovács Emőd: Alkalmazói ismeretek,EKF-TTK, 2011.

[13] Perge Imre: A számítástechnika alkalmazása I. EKF Líceum Kiadó, 2000.

[14] Bártfai Barnabás: Microsoft Office 2010 BBS-INFO Kiadó, 2011 ISBN 978-963-9425-72-9

[15] Péterfy Kristóf: Táblázatkezelés EXCEL 2002 Kossuth Kiadó, 2003 ISBN 9789630944144

[16] Bártfai Barnabás: Makróhasználat Excelben BBS-INFO Kiadó, 2010 ISBN 978-963-9425-40-8

[17] Reményi Zoltán-Siegler Gábor-Szalayné Tahy Tünde: Érettségire felkészítő feladatgyűjtemény Nemzeti Tankönyvkiadó, 2004 ISBN 963 19 54 07 2

[18] Holczer-Farkas-Takács: Windows és Office feladatgyűjtemény Jedlik Oktatási Stúdió, 2005 ISBN 963 86514 7 4

[19] Somogyi Edit: Gazdasági számítások Excel 2007-ben Jedlik Oktatási Stúdió, 2008 ISBN 978 963 87629 5 5

[20] Benkő Tiborné -Tóth Bertalan: Együtt könnyebb a programozás C# Computerbooks, 2008 ISBN: 9789636183578

[21] Illés Zoltán: Programozás C# nyelven, Jedlik Oktatási Stúdió, 2005. ISBN: 963 86514 1 5