

SZAKDOLGOZAT

Nagy Zsolt

Debrecen

2011

Debreceni Egyetem Informatika Kar

Középiskolai programozásoktatás

Témavezető:

Dr. Nyakóné Dr. Juhász Katalin
tudományos főmunkatárs

Készítette:

Nagy Zsolt
informatikatanár
szakvizsgás hallgató

Debrecen

2011

Tartalomjegyzék

Bevezetés	4
Az informatikaoktatás.....	5
Múltja	5
Jelene.....	6
Mi jellemzi jelenleg az informatikaoktatást?	7
Milyen tanítási módszert használunk informatika órán?	7
Informatika érettségi két szinten	9
Jövője	9
Az informatika helye a tantervekben	11
Nemzeti Alaptanterv	12
Informatikai vonatkozásai	12
Kerettanterv.....	14
A számítástechnika oktatásának fő célkitűzései.....	15
A könyvtárhasználat oktatásának célkitűzései	16
A programozás helye a középiskolai informatikaoktatásban	17
A programozás tanítása során alkalmazható oktatási módszerek.....	20
Általános didaktikai megközelítés szerint.....	21
Programozás-tanítási módszerek.....	24
Algoritmizálás, programnyelvek	29
Milyen mélységben algoritmizáljunk?	29
Milyen programozási nyelvet válasszunk?	31
Tanulásra alkalmas programozási nyelvek.....	32
Egy gyakorlati tananyag kidolgozása	38
Free Pascal fejlesztői környezet	39
Gyorstalpaló a Free Pascal IDE használatához	40

Kezdő lépések, szekvencia, szelekció, iteráció	43
Szekvencia	43
Zárszó.....	52
Irodalomjegyzék	53
Melléklet	55
Gyakorlati tananyag folytatása.....	55
Szelekció.....	55
Iteráció.....	61
Alprogramok bevezetése (eljárás, függvény), fájlkezelés.....	67
Eljárás	67
Függvény	70
Fájlkezelés	72
Alapvető algoritmusok	81
Vektor, mátrix feltöltése	81
Megszámlálás, eldöntés	83
Összeg-, átlagképzés.....	85
Kiválasztás.....	86
Kiválogatás	87
Lineáris keresés	89
Maximum- minimumkiválasztás	92
Minimum-kiválasztásos rendezés.....	94
Fejlettebb algoritmusok.....	97
Rendezettség.....	97
Buborék-rendezés	100
Beszúrási rendezés	107
Metszetképzés.....	108
Unióképzés	111

Összefuttatás.....	114
Mátrixműveletek.....	115
VEREM kezelése.....	117
SOR kezelése.....	120
Logaritmikus keresés.....	120
Rekurzió	121
Gyorsrendezés	124
Leszámláló rendezés (Ládarendezés)	126
Dinamikus lista (adatok szétszórt ábrázolása, tárolása)	129
Fabejárás (szélességi, mélységi).....	134
Visszalépéses keresés	134
Megoldandó feladatok.....	135
Szekvencia, szelekció, iteráció	135
Eljárás, függvény, fájlkezelés.....	137
Megszámlálás, eldöntés, összegzés, átlag, kiválogatás, vektor, mátrix	138
Minimum-, maximum-kiválasztás, keresés, rendezés.....	139
Metszetképzés, unióképzés, különbség, összefuttatás.....	140
Rekurzió, dinamikus lista	141

Bevezetés

Sokat gondolkodtam, hogy milyen jellegű szakdolgozatot készítek; rendszerfejlesztés, programkészítés, vagy inkább szakmódszertannal kapcsolatosat. Végül az utóbbi mellett döntöttem, mert a tanári pályán ennek nagyobb hasznát veszem, másrészt érdekes, új területeket lehet megismerni. Viszont az informatika tanítása igen nagy terület, ezért csak egy részének bemutatására vállalkozhattam. Úgy gondolom, az algoritmizálás, a programozásoktatása hangsúlyos része kell, legyen az informatikaoktatásnak. Eltérő mélységben, de nem nélkülözheti sem a szakiskolai, sem a szakközépiskolai és gimnáziumi tanmenet sem. Így esett a választásom a középiskolai programozásoktatás bemutatására.

A szakdolgozatom két egységből áll. Az első részben az informatikaoktatás kialakulásával, helyzet elemzésével foglalkozom. Bemutatom a tantervi háttérrel (NAT, Kerettantervek) majd azt, hogy milyen módszereket lehet alkalmazni a programozás oktatása során. Mivel középiskolában nem kifejezetten programozókat nevelünk, ezért kitérek arra, hogy milyen mélységű legyen az algoritmizálás, és írok a programozási nyelv kiválasztásának szempontjairól is.

A második részben gyakorlati feladatsort állítok össze, részben megoldásokkal, magyarázatokkal. A feladatok összeállításánál a fő hangsúlyt a fokozatosság elvére helyezem. Így a feladatokat feladat csoportokba szervezem.

Az informatikaoktatás

Mint minden más tantárgy esetén, az informatikaoktatásban is jelentős különbség fedezhető fel iskola és iskola között. Tudom, sok minden közrejátszik az eltérő teljesítményhez, de csak egy tényezőt szeretnék kiemelni. A legfontosabb az ország térségei közötti jelentős gazdasági különbség, amely kihatással van az emberek, családok anyagi és morális helyzetére. A munkanélküliség és a vele járó káros szenvedélyek meghatározzák a diákok jövőképét, erkölcsi nézeteit, motivációit.

Így vannak oktatási intézmények, amelyek szelektálni tudnak a hozzájuk jelentkező diákságból (tudás, motiváció, stb. alapján). Ezeknek az intézményeknek természetesen jobb továbbtanulási, verseny, stb. eredményei lesznek. Emiatt megnő a szülők választási hajlandósága, és az eredményeik miatt sikerebben tudnak erőforrásokra pályázni.

Azok a diákok, amelyek különböző okokból kifolyólag már az általános iskolában sem tudtak sikeresen helytállni, csak rosszabb helyzetben lévő intézménybe lesznek képesek bekerülni.

Múltja

A nyolcvanas évek közepén, végén már sok iskolában volt „játékterem”, amelynek a gépparkja többnyire a Commodore, Spectrum illetve a Videoton TV computer egyes változataiból álltak. A diákság tanórán csak ritka esetben fért hozzá a gépekhez, erre leginkább a technika óra¹ volt (lett volna) alkalmas. Jellemzőbb volt az, amikor szakkörön és egyéb tanórán kívüli fakultáció keretén belül próbáltak ráhangolódni a gépek lelki világára. Az oktatás többnyire két dolgot jelentett; a hardver működési alapjainak megismerését és/vagy programozást, többnyire BASIC nyelven. A programozás oktatása nagyon új volt, kialakulatlan tanítási módszerek, tanmenetek és tankönyvek hiánya hátráltatta a tanárok munkáját. Ezt tetézte a számítástechnika szakos tanárok hiánya.

A számítástechnika iskolába történő beszivárgásának első lépése az 1983-as kormány által támogatott iskolaszámítógép-program, ahol középiskolák kaptak – létszámtól függően – legalább egy magyar gyártmányú HT-1080Z típusú számítógépet. Az informatika tantervben előírt – kötelező – oktatása 1988-ban a Művelődési Minisztérium rendeletével lett elindítva, a technika tantárgyban. [-3-] 1988-ra a

¹ Lovassy László Gimnázium – <http://specmat.lovassy.hu>

COCOM lista enyhülésével változás állhatott be a számítógép piacon. Így a nyolcvanas – kilencvenes évek fordulóján megjelentek az iskolákban IBM PC alapú számítógépek, viszont nem olyan mennyiségben, hogy minden számítástechnikát tanulni vágyó diák hozzáférhessen.

A kilencvenes évek elejétől a számítógépek felhasználói oktatása került előtérbe. Vagyis inkább az alkalmazások használatát oktatták, mint a programozást. A kilencvenes évek közepétől indult hódító útjára a WWW, amely már nemcsak szöveges alapú dokumentumok megosztására volt alkalmas, hanem tudta kezelni a képet, egyéb objektumokat, scripteket. 1997-től a SuliNet program keretében minden iskolát igyekeztek bekapcsolni a világhálóba. Tanulóknak és tanároknak is kedvezményesen lehetett számítógépet vásárolni. Az olcsóbbá váló gépek egyre több fiatal életébe behozták a számítógépet, amely kihatott az oktatásra is. A számítástechnika és informatikaképzések slágerré váltak. Persze a tanulói igény az éremnek csak az egyik oldala, a munkáltatói piac, az ipar szükséglete szintén ezt az irányt erősítette.

Jelene

Lassan megbarátkozunk vele, vagy csupán belenyugszunk, hogy életünk minden pillanatában jelen van a számítógép. Mára közhellyé vált, hogy *információs társadalomban* élünk, és az információ minél hamarabbi és minél teljesebb megszerzésére vagyunk kényszerülve. Ha nem mi kapjuk meg elsőként, a versenytársak már le is előztek. Felpörgött világunk információ éhségének legnagyobb kielégítője az internet és szolgáltatásai. Ma már, ha szükségünk van ismeretre valamiről, irány az internet; levél helyett e-mail-ben értesülünk; mindennapos tárgyaink formáját CAD programok adják; programok számolják a statikát; döntéseink meghozatalát mesterséges intelligencia segíti. Ez az irány megadja a lehetőséget, hogy egyre hamarabb elérjük céljainkat, de egyben bonyolulttá is teszi mindennapjainkat. Eligazodni ezekben a technológiákban nem egyszerű. Azok az emberek, akik nem érik el a számítógép és internet nyújtotta előnyöket lemaradnak a versenyben. Itt óriási szerepe van az oktatásnak, tágabban pedig az oktatáspolitikának.

Sok kérdés felvetődik: Hozzá fér-e mindenki a szükséges képzéshez? Összhangban van-e az ipar szükséglete a képzés kibocsátásával? Megfelelően kapcsolódik-e az informatikaoktatás más tudományágak képzéséhez?

Mi jellemzi jelenleg az informatikaoktatást?

Az 1995-ben elfogadott Nemzeti Alaptanterv (NAT) kötelezővé és a minimum meghatározása által valamelyest egységessé tette az addig különböző tartalommal jelen lévő (vagy nem) számítástechnika / informatikaoktatást.

Informatikát oktatni, csak korszerű számítógépekkel és a hozzá társított infrastruktúrával lehet. Évről évre javuló tendenciát mutatva, de még mindig jelentős hiányokat mutatnak az iskolák. Sok helyen, főleg általános iskolákban még mindig nincs megfelelő internet hozzáférés, ergonomikus számítógépasztal, szék, korszerű perifériák. Elavult számítógépeken tanulnak, tíz-tizenöt éves szoftverekkel és ezekből sincs elegendő. [-1-] Viszont szerencsére van ellenpélda is. A szerencsésebb (rangosabb) iskolák pályázat, illetve az ipar szerepvállalásával korszerű multimédiás számítógépteremmel rendelkeznek, fejlett mérőberendezéseket használhatnak.

Adatok a Kőrösné Mikis Márta által vezetett 2004-es informatika tantárgy helyzetelemzése kutatásból [-1-], ahol az egyik kérdéskör a középiskolák számítógép-ellátottságát mérte. Ebből kiderült, hogy átlagosan 55 számítógéppel rendelkeznek. Viszont iskolánként nagy eltérések mutatkoztak: egy iskolában 300 db fölötti a gépszám, 2 iskolában 200 db, 8 iskolában 8-10 db!

Egy OECD INES nemzetközi felmérés alapján hazánkban átlagosan 10,2 diákra jut 1 számítógép, míg Svédországban 3-4 diákra.

A tantárgy oktatását sok helyen nem szakos kollégák végzik és középiskolában is tanítanak főiskolai diplomával. [-1-]

Nehézséget jelent, hogy középiskolában nincs egységes tankönyv. Programozáshoz és emelt szintű oktatáshoz szinte teljes egészében órai jegyzeteket használunk. Az elektronikus dokumentumokat csak kiegészítő jelleggel alkalmazhatjuk, mert nem számíthatunk arra, hogy mindenki tudja olvasni (2011).

Milyen tanítási módszert használunk informatika órán?

Általános és középiskola vonatkozásában is az alkalmazott tanítási módszerek egyformák. Többségében frontális előadásról van szó, tanári magyarázatokkal. [-2-] (Néha a diák kérdéseivel.) Ritka esetekben lehet a szűkös órakeretet projekt módszerre, vagy csoportos foglalkozásokra használni. Differenciált feladatokat alkalmazni még

csoportbontásban, gyakorlati foglalkozásokon is szinte lehetetlen. Erre talán a házi feladatok a legalkalmasabbak, viszont ott nem garantált az önálló munka. (Sok esetben még a megoldás sem.)

Az Európai Unió oktatáspolitikusai egyetértenek abban, hogy az informatikai műveltség elsajátítását már (általános) iskolás korban el kell kezdeni. Abban viszont, hogy ezt milyen formában kell megtenni, már nem ilyen egységes a kép. Például Franciaországban a számítógépek széles körű elterjedése után eltörölték az informatika tantárgyat, és ezzel párhuzamosan az IKT (infokommunikációs technológia) eszközök használatát integrálták más tantárgyakba. A francia oktatáspolitikusok álláspontja az, hogy „*az informatikát nem oktatni, hanem alkalmazni kell*”. Angliában már az 5-7 éves korosztály is elkezd ismerkedni az informatikával, az Alaptantervben leírtak szerint. [-3-]

Így mi is feltehetjük a kérdést, hogy kell-e külön informatika tantárgy?

Úgy gondolom, hogy az informatika tantárgy részbeni eltörlésének (integrálásának) *gondolata* csak akkor vetődhet fel, ha minden iskolás diák számára biztosított az otthoni számítógép használat, internet hozzáféréssel. Továbbá szintén megoldott minden iskolában, oktatási időn túl is az IKT eszközeinek használata. Még ekkor is kérdéses lehet, hogy az informatika fejlődésével lépést tudnak-e tartani a nem szakos tanárok. Másrészt egyre inkább az életünk részévé válik az információtechnológia, mind munkában, mind szabadidőben, így naiv gondolat lenne részünkről, ha ennek pontos és egységes elsajátítását a tanulókra bízánk. A matematikánál is szükséges az alapok lefektetése, a szabályok, definíciók megtanulása ahhoz, hogy később az életben alkalmazni tudjuk. Az informatikának is vannak alapjai, szabályai, fogásai, amelyeket előbb el kell sajátítani, hogy később építhessünk rá.

Ezért az én véleményem az, hogy külön tantárgyként is szerepelnie kell az informatikának, még akkor is, ha minden tanulónak saját számítógépe lesz.

Változásra viszont szükség van! Fel kell erősíteni az IKT használatát más tárgyakban is. Ugyanis informatika órákon, az időkeret szűkösségére tekintettel nincs igazán lehetőség arra, hogy a hardver/szoftver ismereteken túl, integráltabb, alkalmazhatóbb tudást szerezzenek a diákok. (Az érettségi kimenetet is figyelembe véve!) Az alkalmazói tudás egy képesség, amelyet ki kell fejleszteni. Ehhez pedig az is kell, hogy minél szélesebb

körben használjuk a számítástechnika adta lehetőségeket. Ebben segíthet az, hogy a többi tantárgyban tantervileg megköveteljük az IKT használatát. Persze ez nem oldható meg egyik pillanatról a másikra. A saját tapasztalatom az, hogy a nem informatika szakos tanárok többsége nem képes a tanórákon megfelelően használni az IKT eszközeit. A tanárképzésben és a továbbképzéseken kellene IT (információtechnológia) használati módszerekkel, modellekkel megismertetni a pedagógusokat.

Nem szabad elhanyagolni az informatika tárgy folyamatos felülvizsgálatát, témaköreinek aktualizálását. Növelni kell azon ismeretanyagok súlyát, amelyek segítenek a mindennapok ügyeinek intézésében. Például a fontosabb internetes webhelyek alapos megismerése, e-ügyintézés, stb..

Informatika érettségi két szinten

A meglévő közép és emelt szintű érettségi rendszerén változtatni kell. A közép szintet, informatikai alapműveltségi szintnek tartom, ami a fent leírt változtatásokkal alkalmas a feladata ellátására. Az emelt szintet pedig a felsőoktatásba való bejutás alapjának tekinteném, és nem csak államilag finanszírozott képzésben. A vizsga követelmények témaköreinek súlyát a jelenlegi formában megfelelőnek tartom.

Informatikai képzések Magyarországon:

- ECDL (European Computer Driving Licence) – európai számítógép-használó jogosítvány,
- országos képzési jegyzékben (OKJ) megjelölt szakmák és az akkreditált felsőfokú iskolarendszerű szakképzés,
- tanfolyamok,
- szakiskolák, szakközépiskolák,
- felsőfokú szakképzés
- főiskolai vagy egyetemi (alap, mester) képzések,
- informatika doktori képzés.

Jövője

Az informatika, mint tudomány, mint technológia erőteljesen fejlődik. Új elméletek, és gyakorlati megvalósítások látnak napvilágot. Ennek a változásnak bizonyos mértékig a tantervekben/tanmenetekben is meg kell jelennie. Vagyis az informatika tantárgy tartalmának folyamatos megújítása szükséges.

A szoftverek sokszínűsége nem teszi lehetővé azt, hogy minden egyes terméket külön-külön megismerjünk. Ezért az oktatás során arra kell törekedni, hogy a diákokban kialakuljon egy „szoftverhasználati készség”, melynek segítségével eltérő funkcionalitású alkalmazásokat is könnyebben használatba tudnak venni.

Fontos, hogy a tanár, lépést tartson ezzel a fejlődéssel, amelyet részben önállóan kell megtennie szakirodalom felhasználásával, részben központi továbbképzések jelentős fokozásával. Itt nemcsak a technikai ismeretekre gondolok, hanem a szakdidaktikai tudás naprakészen tartására is. Az informatika a kezünkbe ad olyan eszközöket, szoftvereket (projektor, digitális tábla, multimédia), amelyeknek a helyes használatát, az alkalmazásuk módszereit el kell sajátítani.

A közoktatásban is egyre inkább előtérbe kerülnek az e-learning rendszerek. Az internet web2 újdonságai (blog, podcast, e-vizsgáztatás, stb.) szintén sikeresen beépülhetnek. A számítógépeket be lehet vinni a tantermekbe is, így fokozható a diákok órai aktivitása, és felhasználhatóak lennének az internetes tudásbázisok (pl. SDT). Könnyebb lenne a differenciált tanítás megvalósítása. Az eltérő teljesítményszintű feladatokat könnyebb elektronikusan kiosztani és felügyelni. Oktatóprogramokat használhatnánk, és a lassabb tanulóknál, a fogyatékkal élőknel a programozott oktatás is alkalmazható lenne. A számítógép segíthet abban, hogy ne csak egy kommunikációs csatornát használjunk a tanítás/tanulás során, hanem támaszkodjunk a vizualításra is. A diákok ne csak passzív befogadók legyenek, hanem gyakorlással, utánzással, tapasztalással interaktívan vegyenek részt az órákon. A tanár és tanuló között egy *konstruktív* tanulási folyamat alakuljon ki.

Meggondolandó lenne a szoftver nagyhatalmak hegemoniájának csökkentése, alternatív alkalmazások, free szoftverek bevezetése (Linux, OpenOffice, „számítási felhők” kihasználása). Egyes országokban ez már megvalósult.

Az e-book olvasók tömeges elterjedése, illetve az otthoni diák PC-k számának növekedése, lehetővé teszi a digitális tananyagok bevezetését az iskolákban. Ez sok gondot levonne mind az állam, a tanulók és a tanárok válláról (Moodle keretrendszer). Már most is nagy számban vannak jól használható online tananyagok, amelyek naprakészen tartása is könnyebb a tankönyveknél. Az elektronikus tananyagok koordinálását, kezelését, szerintem állami feladatként kellene megvalósítani.

A papír alapú iskolai adminisztrációt is felváltaná az elektronikus osztálynapló, a szülőkkal való online kapcsolattartás. Az érdemjegyek, szülői értesítések – biztonság és hitelesítés mellett – az interneten is elérhetőek lennének.

Még egy lényeges dolgot szeretnék kiemelni. Az internet által elénk tárt ismeretözönben el is kell tudni igazodni. A jövő és talán már a jelen tanárainak az ismeretek közötti szelektálást, szűrést is meg kell tanítania. Álljon itt egy idézet Kemény Jánostól a hetvenes évekből:

"Simon tétele az, hogy egy jó információs rendszer ne a lehető legtöbb információval lásson el bennünket, hanem a lehető legkevesebbel, ami a munkánkhoz kell... Információgazdag világban élünk, a probléma nem az elégtelen információból adódik, hanem éppen abból az információáradatból, amit képtelenek vagyunk feldolgozni. Az informatika feladata, hogy csökkentse, és ne fokozza az információs zűrzavart." [-7-]

Az informatika helye a tantervekben

A nyolcvanas évek végétől a gazdaságban „új szelek” kezdtek fújni, megnyíltak a nyugati piacok, vállalkozási dömping indult el.

Az oktatásban is kezdett lazulni a központi előíró jellegű szabályozás. Egyre több iskola próbált ki új dolgokat. Megjelentek a hagyományos oktatástól gyökeresen eltérő reformpedagógiai irányzatok (Waldorf-, Montessori-, Freinet-pedagógia). Az iskolák új oktatási stratégiákat vezettek be (nyílt-, adaptív-, projekt-, programozott-oktatás). A számítógépek viszonylagos elterjedésével, egyre több oktatási intézmény gondolta úgy – jogosan –, hogy a számítástechnikai ismereteket oktatni is kell. Mindezt nem kiegészítő jelleggel szakkörökön, fakultációkon, hanem a helyi tantervben előírva, kötelezően. Így kényszerűségből minden iskola elkészítette a számítástechnika tanmenetét, amely a koordináció hiánya miatt eléggé változatos képet mutatott. Természetesen ez a szabadság más tantárgyaknál is megvolt. Erre a kialakult helyzetre szükségszerűen választ kellett adnia a kor oktatáspolitikájának. Ennek egyik produktuma volt a Nemzeti Alaptanterv (NAT) megszületése.

Nemzeti Alaptanterv

A Kormány 1995. október elején fogadta el a NAT végső változatát, amely részben oktatáspolitikai, részben pedagógiai dokumentum. Eltért az addigi „hagyományos” tantervektől, mert nem határoz meg tantárgyakat, óraszámokat, évfolyamonkénti felosztást. Alulról korlátos, vagyis azokat a minimumokat írja elő, amelyeket minden ép elméjű diáknak teljesítenie kell. Konkrét tantárgyak helyett 10 műveltségi területet határoz meg, amelyek között fontossági sorrendet is felállít.

Műveltségi terület	Rangsor 1-4. évf.	Rangsor 5-6. évf.	Rangsor 7-8. évf.	Rangsor 9-10. évf.
Anyanyelv és irodalom	1	1-2	2-3-4	4
Élő idegen nyelv	-	4	5-6	5
Matematika	2	1-2	2-3-4	2-3
Ember és társadalom	6-7	7-8	2-3-4	2-3
Ember és természet	5	6	1	1
Földünk és környezetünk	-	-	9-10	9-10
Művészetek	3	3	5-6	6
Informatika	-	9-	9-10	9-10
Életvitel és gyakorlati ism.	6-7	7-8	7-8	7
Testnevelés és sport	4	5	7-8	7

1. Táblázat: A NAT műveltségi területeinek rangsorlistája a javasolt arányok alapján.

A táblázatból is látható, hogy összhangban az európai fejlődéssel az anyanyelvi, irodalmi, művészeti, matematikai, valamint a természettudományos ismeretek kerültek súlyponti helyre. Az idegen nyelv és a testnevelés közepes fontossággal szerepel. Sajnálatos viszont, hogy az informatikaoktatás ennyire háttérbe szorult (9-10-es rangsor). Láthatjuk az is, hogy olyan területeket tett az alpműveltség részévé, amelyek korábban nem szerepeltek benne (emberismeret, vizuális kultúra, életvitel, környezetkultúra).

Informatikai vonatkozásai

A NAT 1997-es bevezetése az informatika területén több akadályba is ütközött. Az első problémát a számítástechnika, informatika tanárok hiánya jelentette. A gazdaság fizetése sok tanárt csábított a pálya elhagyására, illetve a felsőoktatásból újonnan kilépő tanárok száma is kevesebb volt a szükségesnél. A második akadályt az jelentette, hogy a tanításhoz nem volt megfelelő infrastruktúra, kevés és elavult számítógép jellemezte az

iskolai gépparkot. A NAT minimum követelményként írta elő a hálózati hozzáférés megtanítását, viszont kevés helyen volt biztosított a megfelelő internet csatlakozás. Persze, ezeket a hiányokat a SuliNet program nem sokra rá (1998) megoldotta. A harmadik gondot a tankönyvpiac kiforratlansága jelentette. (Itt még mindig akadnak hiányosságok: középiskolás programozás, adatbázis-kezelés, multimédia tankönyvek.)

A NAT 252 informatika órával számol 12 évfolyamra. Ez nem elegendő még a középszintű informatika érettségi letételéhez sem.

Az informatika műveltségi terület részletes követelményeit két csoportra bontja, a tanuló fejlettségi szintjének megfelelően. Mindkét területen megadja a témaköröket.

1-6. évfolyam témakörei:

- Számítógépes ismeretek
- A számítástechnika alapjai
- Az operációs rendszer használata
- Oktatóprogramok futtatása
- Algoritmizálás
- Számítógéppel segített problémamegoldás
- Szöveg- és ábrszerkesztés
- Az adatbázis- és a táblázatkezelés előkészítése.

7-10. évfolyam témakörei:

- Számítógépes ismeretek
- A számítástechnika alapjai
- Az operációs rendszer használata
- Oktatóprogramok futtatása
- Algoritmizálás
- Számítógéppel segített problémamegoldás
- Szöveg- és ábrszerkesztés
- Az adatbázis- és a táblázatkezelés előkészítése.

Kerettanterv

A NAT műveltségi területeinek, és részterületeinek a gyakorlatban a helyi tantervekben kellett realizálódni, konkrét tantárgyak keretében. Az általánosítás és a minimum követelmények megfogalmazása révén, a NAT alapján sok tantervi változat készülhetett. Főleg az „elit” iskolák ragadták meg ezt a lehetőséget, és az önkormányzati finanszírozás erejéig, a saját arculatuknak megfelelően növelni kezdték a tantárgystruktúrájukat. Ez a helyi autonómia oda vezetett, hogy iskola és iskola között óriási különbségek alakultak ki. A verseny miatt nőtt az egyenlőtlenség (Máté-effektus)¹ és így sérültek az 1993-as közoktatási törvényben (ktv) megfogalmazott alapvető állampolgári jogok. Emiatt szükség volt a NAT és a Helyi Tanterv (HT) között egy újabb szabályozó elem elkészítésére. Ezt a funkciót látják el a kerettantervek. Bár a ktv szabályozza az intézmények maximális óraszámát, ennek nem sikerült teljesen érvényt szerezni. A kerettanterv meghatározza a maximális óraszámokat. Ennek indoklására azt hozták fel, hogy kisebbek lehetnek a finanszírozásból adódó iskolai különbségek, illetve az szakmódszertanosokat, pedagógusokat rákényszeríti a gazdaságosabb tudásátadásra.

A kerettanterv tárgyként jeleníti meg az informatikát, ugyanakkor több lehetőséget ad a tantárgyközi alkalmazásokra, elsősorban az önálló ismeretszerzés során, illetve a könyvtár témakörben. A kerettantervi minimum óraszám 167, amely szintén nagyon kevés, a maximum óraszám 317.

A szakközépiskolák 9-12. évfolyamának ajánlott informatika óraszámai

9. évfolyam	37	heti 1 óra
10. évfolyam	37	heti 1 óra
11. évfolyam	74	heti 2 óra
12. évfolyam	64	heti 2 óra
Összesen	212 óra	

A gimnáziumok 9-12. évfolyamának ajánlott informatika óraszámai

9. évfolyam	55,5	heti 1 óra
-------------	------	------------

¹ Máté-effektus: akinek van, annak adatik, akinek nincs, attól elvétetik.

10. évfolyam	37	heti 1 óra
11. évfolyam	55,5	heti 2 óra
12. évfolyam	48	heti 2 óra
Összesen	196 óra	

Véleményem szerint ez az óraszám középszintű érettségire való felkészítés során, jó képességű tanuló esetén is csak kiegészítő gyakorlással elegendő. Ez a pluszgyakorlás általában otthoni számítógép-használatot jelentene, viszont erre még 2011-ben sincs minden diáknak lehetősége. Emelt szintű érettségihez pedig a programozás, HTML, és SQL ismeretek tananyag tartalmának leadásához szükséges teljes óraszám hiányzik. Erre még jelentős otthoni tanulással és gyakorlással is minimum 111 óra szükséges (heti 3 óra).

Az informatika tantárgy legfontosabb területeit a számítástechnika, – beleértve a multimédia és az internet ismereteket – illetve a könyvtárhasználat jelenti.

A számítástechnika oktatásának fő célkitűzései

- *„Korszerű alkalmazói készség kialakítása:* a tanulók képesek legyenek arra, hogy a számítógépeket és az informatikai eszközöket célszerűen használják.
- *Az algoritmikus gondolkodás fejlesztése:* a matematikához hasonló gondolkodásfejlesztő szerep, amely nemcsak az iskolában, hanem a hétköznapi életben is alapvető fontosságú.
- *Önálló munkára nevelés, differenciált tanulás:* a számítógép, mint interaktív eszköz lehetőséget teremt az egyéni ütemű tanulásra és a pontos, kitartó, fegyelmezett munkára.
- *Együttműködésre nevelés, csoportmunka:* nagyobb számítógépes feladatok megoldása megköveteli a csoportmunkát, feladatok részekre osztását, a másokkal való kapcsolattartást, tervszerű, összehangolt munkát.
- *Alkotó munkára nevelés:* akár táblázatot készítünk a számítógéppel, akár szöveges dokumentumot, a végeredmény egy új produktum lesz.
- *Az informatika társadalomban játszott szerepének felismertetése:* az informatika rohamos fejlődése az egész társadalmat gyökeresen átalakítja, s ebben az állandóan változó világban otthon kell éreznie magát a tanulónak.

- *Az informatikai ismeretek rendszeres alkalmazása:* az iskolai élet eseményeihez vagy a tantárgyakhoz kapcsolódó feladatok megoldására a tanulók használjanak informatikai eszközöket (dolgozat, előadás, bemutatás, tantárgyi feladatmegoldás, szervezés, tanulás).
- *Az informatika etikai és jogi szabályainak megismertetése:* tudatosítani kell a tanulóknál az információszerzés, -feldolgozás és -felhasználás etikai és jogi szabályait.
- *Az esztétikai készség fejlesztése:* igény és készség a számítógépes produktum esztétikus formájának kialakítására.” [-15-]

A könyvtárhasználat oktatásának célkitűzései

- *„Felkészítés az információs társadalom kihívásainak fogadására:* az információszerzés bővülő lehetőségeinek felhasználására, az információk elérésére, kritikus szelekciójára, feldolgozására és a folyamat értékelésére.
- *A könyvtárra alapozott önművelés képességének kialakítása,* fejlesztése a könyvtári információs rendszer lehetőségeinek felhasználásával.
- *A forrásokat komplex és alkotó módon alkalmazó tanulási technikák és módszerek* kifejlesztése. Az iskolai és más típusú könyvtárak, könyvtári források, eszközök megismertetésével, valamint a velük végzett tevékenységek elsajátíttatásával tudatos, biztos használói magatartás kialakítása.
- *A könyvtárhasználati tudás eszközjellegű beépítése* a tanulók tantárgyi képzéséhez, iskolai fejlődéséhez és a mindennapi problémák megoldásához szükséges információszerzésbe és -feldolgozásba.
- *A forrásfelhasználás etikai szabályainak elsajátíttatása* és a normakövetés követelményének elfogadtatása. A különböző társadalmi szerepekbe beilleszkedni, azokat szükség szerint változtatni és bennük hasznosan tevékenykedni tudó személyek nevelése.” [-15-]

Amennyiben sikerül a célok eléréséhez szükséges tudást átadni a diákoknak, bizonyára sikeresen elboldogulnak az úgynevezett „információs társadalomban”. Hiszen megismerkednek az infokommunikációs eszközökkel, azok használatával, alkalmazói szoftverekkel, különböző adatbázisokkal, illetve a könyvtárbeli információszerzéssel.

A 2002-es keresttantervi elképzelések, pedig az informatikai ismeretek (IKT eszközhasználat) integrálását segíti más tantárgyakba, ezzel is erősítve az informatikai alkalmazói képesség fejlesztését, megszilárdítását.

A programozás helye a középiskolai informatikaoktatásban

A programozás oktatását a szakközépiskolai, gimnáziumi informatika órák anyagában vizsgálom.

Fentebb láthattuk, hogy a NAT mind 1-6. évfolyamra, mind 7-10. évfolyamra meghatározza az *algoritmizálás* oktatását.

A kerettantervben a digitális kompetencia fejlesztésének részeként jelenik meg az algoritmizálás/programozás. Az algoritmizálási képesség kialakítása segítséget kíván nyújtani a mindennapi életben felmerülő problémák megoldási lépéseinek összeállításában, a számítógépes alkalmazások kezelésében. Az informatikában fontos szerepet kap a *matematikai kompetencia fejlesztése*, a logikus gondolkodás képességének kialakítása.

Azaz a matematikai gondolkodás és problémamegoldás fejlesztése, az algoritmusok alkalmazásának képessége, mindennapi problémák megoldása és a világ rendszereinek, folyamatainak informatikai (számítástechnikai) modellezése során. Mindezeken túlmenően a tevékenység-algoritmusok nagy szerepet játszanak a számítógépes kommunikációban, az alkalmazások és az információk kezelésében is.

A kerettanterv által meghatározott programozással kapcsolatos fejlesztési követelmények: „Legyen képes különféle formákban megfogalmazni a környezetében, az iskolában előforduló tevékenységek algoritmizálható részeit. Helyesen használja a logika bizonyos elemeit (és, vagy, nem, ha ... akkor ...). A problémamegoldás során ismerje fel az adatok közötti összefüggéseket. Ismerje fel az adatok és az eredmények kapcsolatát. Egyszerű feladat megoldásához legyen képes algoritmust készíteni, és az algoritmust megvalósítani számítógépen (a használt fejlesztő rendszerrel). Legyen képes értelmezni a programok által szolgáltatott válaszokat.”¹

¹ Kerettanterv a szakközépiskolák 9-12. évfolyama számára, Magyar Közlöny, 2008/20/II. szám/835. oldal

Szakközépiskola

A programozás tanítása, a kerettantervi felosztás szerint 12. osztályban történik, a heti két órás informatika óra *algoritmizálás és adatmodellezés* témakörén belül. A következő tartalmak rendelődnek hozzá:

- Algoritmusok a gyakorlatban,
- Algoritmusok leírása általános eszközökkel,
- Programozási nyelvek és a programok,
- Egy fejlesztő rendszer használata,
- Elemi adattípusok,
- Konstansok és változók,
- Adatok bevitele, tárolása és megjelenítése, közlése,
- Képletek és függvények használata,
- Feltételes elágazások,
- Ciklusok típusai,
- Összetett adattípusok, tömbök,
- Típusalgoritmusok,
- A programkészítés lépései.

A megfogalmazott belépő tevékenységek:

- Hétköznapi tevékenységsorok, algoritmusok leírása,
- Adott feladat megoldásához algoritmus tervezése,
- A szükséges adatok és eredmények megtervezése,
- Elemi és összetett adatok, képletek és függvények használata,
- Elágazások és ciklusok alkalmazása,
- Fejlesztő rendszer használata,
- Típusalgoritmusok alkalmazása életszerű, gyakorlati feladatok megoldása során (Összegezés, keresés, megszámlálás, rendezés.),
- A kész program tesztelése és alkalmazása.

Az algoritmizálás és adatmodellezés témakör által meghatározott tartalmi elemek, lényegében átfogják az emelt szintű érettségi programozással kapcsolatos vizsgakövetelményeit. A megvalósítás esélytelenségét a hozzá társított órakeret jelenti

(64 óra), figyelembe véve azt, hogy az algoritmizáláson túl még a következő témaköröket is oktatni kell az előbb említett órakeretben:

- Folyamatok és rendszerek modellezése,
- Problémamegoldás,
- Az informatika fejlődéstörténete,
- Az információs társadalomról,
- Etika és jog,
- Adatvédelem. Az információ megbízhatósága,
- Könyvtári informatika, médiainformatika.

Gimnázium

A programozás tanítása, a kerettantervi felosztás szerint 11. osztályban történik, a heti két órás informatika óra *algoritmizálás és adatmodellezés* témakörén belül. A következő tartalmak rendelődnek hozzá:

- Algoritmusok a gyakorlatban,
- Algoritmusok leírása általános eszközökkel,
- Programozási nyelvek és a programok,
- Egy fejlesztő rendszer használata,
- Elemi adattípusok. Konstansok és változók. Adatok bevitele, tárolása és megjelenítése, közlése,
- Képletek és függvények használata,
- Feltételes elágazások. Ciklusok típusai,
- Összetett adattípusok, tömbök,
- Típusalgoritmusok,
- Rekurzió,
- A programkészítés lépései,
- Problémák megoldása számítógépes modellekkel.

A megfogalmazott belépő tevékenységek:

- Hétköznapi tevékenységsorok, algoritmusok leírása. Adott feladat megoldásához algoritmus tervezése,
- A szükséges adatok és eredmények megtervezése,

- Elemi és összetett adatok, képletek és függvények használata,
- Elágazások és ciklusok alkalmazása,
- Fejlesztő rendszer használata. A kész program tesztelése és alkalmazása,
- Típusalgoritmusok alkalmazása életszerű, gyakorlati feladatok megoldása során, (Összegzés, keresés, megszámlálás, rendezés.),
- Egyszerű modellek megismerése, „kísérletezés” a modellekkel,
- Például oldjunk meg, algoritmizáljunk matematikai feladatokat, modellezzünk egyszerű fizikai rendszereket.

Látható, hogy a szakközépiskoláktól való egyetlen eltérést a rekurzió megjelenése jelenti. Az 55,5 órás éves időkeret itt is nagyon kevés főleg, ha figyelembe vesszük, hogy az adatbázis-kezelés témakört is erre az évre tervezik.

A programozás tanítása során alkalmazható oktatási módszerek

Az oktatási módszer egy-egy didaktikai feladat megoldásának módját, menetét jelenti. A tanítás tartalma és a megoldandó didaktikai feladat alapvetően meghatározza az oktatási módszereket, illetve azok kombinációját. A módszerek kiválasztása egyrészt attól függ, hogy az elsajátítandó ismeretek milyen sajátosságokkal rendelkeznek, másrészt attól, hogy az adott ismeretek elsajátításának mely szakaszáról, azaz mely didaktikai fő feladat megoldásáról van szó.

Az oktatási módszerek megválasztása függ a célcsoporttól (kiknek tanítunk), hogy mit tanítunk (programozás – internetes kommunikáció), továbbá a lehetőségektől is (eszközigény).

A programozás oktatása során sikeresen alkalmazható a táblánál történő kikérdezés. Az algoritmus- és programkészítés elsajátítása során fontos a naprakész tudás, az elmélet és a gyakorlat egymásra ható ismerete. Egy programozási feladat megoldásához ismerni kell az alapvető algoritmusokat, és a programozási eszközöket is. A felelőtől kérhetünk elméletet, algoritmust egy adott problémára, és konkrét programot vagy program részletet is.

A házi feladatok a gyakoroltatáson túl alkalmasak a differenciált feladatmegoldásra is. A tanulók viszont sokszor lemásolják egymásról a megoldásokat, amellyel nemcsak a tanárt csapják be, hanem saját magukat is. A programkészítésnél erről árulkodhat, ha a

beolvasást, kiírást ugyanolyan sorrendben, formában oldják meg. Elősegíthetjük az önálló feladatmegoldást azzal, hogy megköveteljük a programok kommentezését. Magyarozatként pedig a fejlesztő általában a saját gondolatait fűzi a program szövegéhez.

Az oktatási módszerek megválasztásánál lényeges szempont a tanulói önellenőrzés, önértékelés fejlesztése. A módszerek olyan kombinációját kell felhasználni az informatika oktatása során, amelyekkel alkalmassá tesszük a tanulót az önálló tanulásra és alkalmazói képességek fejlesztésére. Az informatika tantárgy különösen sok lehetőséget nyújt arra, hogy tudatosítsuk a tanulóknál a rendszeres önképzés szükségességét. A hardverek állandó fejlődése, a szoftverek egyre újabb verzióinak megjelenése minden leendő felhasználótól állandó lépéstartást kíván.

Általános didaktikai megközelítés szerint

A tanár munkáján alapuló módszerek

Szóbeli ismeretközlés

A tanár szakszerűen, az előzetes tudás figyelembevételével, érthetően adja át a tananyagot. Beilleszti a meglévő ismeretek hierarchiájába, fontos a megfelelő kapcsolati háló kialakítása. Főleg kisebb egységek bevezetése, feldolgozása történik. Sose felejtsük el az óra tárgyának megjelölését, és a tanulók motiváltságának fenntartását.

Ilyen például az algoritmus leíró eszközök ismertetése. Bevezetjük, hogy miért fontos ismernünk (modellezés, absztrakció, szemléletes, átlátható). Majd ismertetjük a fontosabb leíró eszközöket, azok előnyeit/hátrányait, melyiket mikor érdemes alkalmazni. Végül elmondjuk milyen előnyöket jelent a tervezés, az átláthatóság az erre épülő programírásnál.

Előadás

Az előadás hosszabb időn át tartó ismeretközlés, amely elemzéseket, általánosításokat is tartalmazhat. Az tartalom hosszúságától függően sok időt, energiát és felkészültséget igényel a tanártól. Pontosan meg kell tervezni a gondolatmenetet, a témák összekapcsolását. Érthető, érdekes és egyben pihentető példákkal kell tarkítani a monoton részeket. Jó, ha felkészülünk a

tanulók által közben feltett kérdésekre. A tanárnak illik ismernie a téma tágabb környezetét, kapcsolódásait is.

Jól alkalmazható az előadás a történeti jellegű témák feldolgozásánál (programozás kialakulása, programnyelvek fejlődése, a jövő lehetőségei).

Magyarázat

A fogalmak kifejtésénél, a kapcsolatok, összefüggések feltárásánál lehet hatékony. Érthetően, a tanulók fejlettségi szintjének megfelelően, lehetőleg nem szakszavakat használva magyarázzunk. Támaszkodjunk a megelőző tudáselemekre, és minél több oldalról közelítsük meg a kifejtendő témát.

Például a tömb adatszerkezet ismertetésénél, magyarázhatjuk úgy a struktúrát, mint egy szálloda, amelyikben minden szoba egyforma nagyságú és minden lakó ugyanarra a szolgáltatásra számíthat. Az egyes szobákat pedig az ajtajukra kitett szobaszámon keresztül tudjuk azonosítani.

Szemléltetés, kísérlet, szimuláció

Alapja az, hogy „kézzelfoghatóvá” tesszük az ismeret tárgyát. Ha az eredeti bemutatására, tanulmányozására nincs lehetőség, akkor modelleket is használhatunk. Vagy gyorsan változó, drága, veszélyes jelenségeket szimulálhatunk is.

A tömb példánál maradva, készíthetünk egy gyufaskatulyákból összeragasztott szemléltető eszközt.

A különböző rendezések szemléltetésére felhasználhatjuk az interneten már több webhelyen is megtalálható animációkat.

A tanár és a tanuló közös munkáját követelő módszerek

Megbeszélés

A megbeszélés a tananyagnak interaktívan, kérdés-felelet formában való feldolgozását jelenti. Nagy előnye, hogy a tanulók nem csak passzív részesei az óráknak. A kérdezés feltétele, hogy a diák bekapcsolódjon a gondolatmenetbe, vagyis aktív részese legyen az órai munkának. Kérdéseivel valamelyest a témát

is irányíthatja. Lehetőleg a távolra vezető kérdések megválaszolását halasszuk óra végére.

A megbeszélés nemcsak a tanár és tanuló közötti párbeszédet jelenti, kialakulhat tanuló és tanuló között is. Ilyenkor hagyjuk, had vitassák meg egymás között. Meglátjuk milyen büszke lesz magára az a diák, aki elmagyaráz dolgokat az osztálytársainak.

Ismétlés

Az ismétlés az ismeretek rögzítésének, elmélyítésének az eszköze. Alkalmazzuk akkor, ha egy olyan új témakört kezdünk, amely kapcsolható egy már régebben feldolgozott anyaghoz. Ilyenkor felelevenítjük, „leporoljuk” az emlékeket, hogy könnyebben adódjanak hozzá az új ismeretek. De alkalmazzuk akkor is, ha végére értünk egy témakörnek, és a számonkérés következik. Az ismétlés is akkor éri el a legjobb eredményt, ha a tanulót is bevonjuk.

A programozás tanítása/tanulása nagyon sok ismétléssel jár. A változó fogalomtól kezdve a deklarációkon át az élettartam, a lokális/globális fogalmak relatív viszonyáig szinte mindent ismételni szükséges.

Gyakorlás

A gyakorlás a megszerzett ismeretek elmélyítéséhez, alkalmazásához nyújt eszközöket. Kifejlődnek azok a készségek, képességek, amelyek lehetővé teszik a könnyebb adaptációt. Megoldási minták, algoritmusok alakulnak ki. Az egyes problémákat könnyebb lesz osztályokba sorolni, és a rá jellemző tulajdonságokat, viselkedésmódokat felismerni.

Ilyenek a különböző programozási feladatok, amelyeket lehet az órán is készíteni, és házi feladatnak is adható. A házi feladatok adásánál ne felejtsük meg az ellenőrzésről.

A tanuló önálló munkáján alapuló módszerek

Gyakorlati feladatok

Itt inkább a nagyobb, átfogóbb, több témakört érintő feladatokra kell gondolni. Ezek megoldása történhet egyénileg, vagy csoportosan, akár projekt jelleggel is.

Az ilyen jellegű feladatok segítenek a témakörök összekapcsolásában, és bepillantást engednek a munka világában meglévő szituációkba (információkeresés, kutatás, egyeztetések, csoporton belüli magatartás, kompromisszum, vezetés, pontosság, felelősség, stb.).

Kiselőadások

A kiselőadás egy rész-, vagy kiegészítő témakör tanuló általi feldolgozását, majd osztálytársainak való prezentációját jelenti. Fejleszti az önállóságot, kifejezőkészséget, előadásmódot, ismeretszerzést, IKT eszközhasználatot. A témák kiosztása után adjunk tanácsokat az induláshoz, anyaggyűjtéshez. Tegyük egyértelművé elvárásainkat, a minőséggel és a feldolgozandó téma határaival kapcsolatban. Tapasztalataim szerint a kiselőadás ideje 10-15 perc legyen. Hívjuk fel a figyelmüket vázlatkészítésre, és a lényeg kiemelésére. Az elkészített prezentációt bemutatás előtt mindig ellenőrizzük, a bemutatás után pedig értékeljük! (Nem feltétlenül érdemjeggyel.)

Az algoritmus folyamatábrával való leírása sokszor fáradságos munka, papír és toll használatával. Kiadhatjuk a feladatot egy-egy diáknak, hogy kutassák fel és mutassák be, hogy milyen alkalmazásokkal lehet segíteni ezt a tevékenységet.

Házi feladat

A tanulók önálló, a tanítási órák között végzett tevékenységén alapuló módszer. A tanár szerepe a feladatok pontos kijelölésére, és ellenőrzésére korlátozódik. Jól alkalmazható a differenciált oktatás megvalósítására. A diákok képességükhöz mérten kaphatnak reprodukív, felfedező, vagy kutató jellegű feladatokat. Projektszerűen, akár csoportos munka is elképzelhető, megkövetelve azt, hogy az egyéni teljesítmény nyomon követhető legyen.

Programozás-tanítási módszerek

Módszeres, algoritmusorientált

A programkészítés teljes életciklusát átfogja: feladat-meghatározás, algoritmus- és adatstruktúra tervezése, algoritmus helyesség, kódolás, tesztelés, minőségvizsgálat, dokumentálás.

Az egyes résztevékenységekkel külön-külön kell foglalkozni, mindegyikben áttekintve a témához kapcsolódó eszközöket, és módszereket. Az elsődleges szempont az algoritmus megtalálása, előállítása. Azaz a tanítás elsődleges feladata, az algoritmikus gondolkodás kialakítása, fejlesztése. Első lépésként általános feladattípusokat, úgynevezett programozási tételeket veszünk. A következő lépés, hogy a konkrét feladatokat (részfeladatokat) besoroljuk feladat osztályokba. Vagyis megkeressük a megoldási lépéssorozatot adó programozási tételt. Majd, ha a különálló algoritmusok megtalálása és elkészítése már megy, akkor jöhetnek az összetettebb feladatok. Olyan bonyolultabb problémákat vizsgálunk, amelyek több részfeladat egyenkénti megoldásából, és végül összeépítéséből állíthatók elő. A programozási nyelv megkötései, csak a kódolási fázisban jelennek meg. Ezért, aki így tanul programozni, nem lesz nyelvhez kötve. A kódolási szakaszban kitérhetünk az algoritmus iteratív, és rekurzív módon való megoldásának tárgyalására. Illetve a programok ergonomikus/felhasználóbarát tulajdonságaira. A program tesztelése során az algoritmust is vizsgáljuk. Megoldja-e a feladatot? Minden inputra ad-e értelmezhető kimenetet? Az eredményt hatékonyan állítja-e elő? Ezen kérdések fontosságára is fel kell hívni a tanulók figyelmét. A hatékonyság vizsgálatnál készíthetünk olyan különböző keresést, vagy rendezést tartalmazó programokat, amelyeknél mérjük a futási időt.

Ez a módszer a teljességre törekszik, éppen ezért alkalmazható az informatikai szakképzésben (informatikai szakmacsoport). [-18-]

Feladattípus-orientált

Ennél a módszernél mindig teljes feladatokat oldunk meg. Amennyiben eddig még nem ismert programozási eszközt kell használni, azt bevezetjük, megismerjük, majd alkalmazzuk is az aktuális feladat megoldásánál. A feladatokat úgy kell csoportosítani, majd a csoportokat sorba rendezni, hogy az egyszerűbb tevékenységekkel el tudjuk végezni az alapozást, majd fokozatosan egyre bonyolultabb feladatokon keresztül eljussunk az összetett problémamegoldásig. Nagyon jól használhatók a matematikai feladatok, de szöveges, vagy grafikai programokat is készíthetünk. Ez a módszer sem nélkülözi az adatszerkezeteket, vagy az algoritmusokat, csupán azok ismertetését akkor teszi, amikor egy konkrét feladat kapcsán szükségünk van rá. Így az algoritmus nem egy

elvont lépéssorozatként jelenik meg, hanem rögtön hozzá is tudjuk kötni valós problémákhoz.

Ez a módszer alkalmas a viszonylag kis óraszámú, közoktatásban megvalósítandó programozás tanítására, ahol az algoritmikus gondolkodás kialakítása/fejlesztése a cél, és nem a teljes programozási eszközrendszer megismertetése. [-18-]

Mintapéldákon keresztül

A tanulók elé mintapéldákat teszünk, vagy algoritmust valamilyen algoritmus leíró eszközzel, vagy konkrét programot. A példák elemzésén, módosításán keresztül ismertetjük meg a diákokat a programozás alapjaival. Ez a módszer nagymértékben támaszkodik az érdeklődésre.

A tanár elkészít egy példaprogramot, majd ezen keresztül elmagyarázza az eszközöket, azok használatának módját, az algoritmust. Megmutatja, hogy programkódon történt változtatások milyen kimeneti változást okoznak. Majd azt nézzük meg, hogy a feladat paramétereinek megváltoztatása, milyen kódolási változtatást követel.

A tanulótól nagyfokú általánosítást, absztrakciót követel, ezért a közoktatásban egyedüli módszerként nem tanácsos alkalmazni. Érdeklődő diákoknak, délutáni fakultáció keretein belül viszont eredményesen alkalmazható. Illetve bonyolultabb algoritmusok bemutatására is alkalmazható. [-18-]

Nyelvorientált

Mindig valamelyik programozási nyelvhez kötődik, és a nyelv utasításain keresztül ismerteti meg a programozási eszközöket. Emiatt lesznek olyan eszközök, amelyeket be sem vezetnek, mert a választott programozási nyelv nem tartalmazza. Például a „C”-nél: logikai típus, eljárás, modul. Vagy helytelen programozási stílust alakíthatnak ki (pl. BASIC-nél az automatikus deklaráció). Ennél a módszernél is teljes feladatokat oldunk meg. Viszont a feladat-típusokat mindig a programozási nyelv adott eszközének ismertetéséhez kötjük. [-18-]

Adatorientált

Ez a módszer az adatstruktúra előállítását tekinti elsődlegesnek. A programot úgy építjük fel, hogy megkeressük a bemeneti és a kimeneti adatokat, majd a fokozatos

közelítés módszerét alkalmazva a megoldás menetét adó belső „átmeneti” adatszerkezetet. (Jackson).

Ennél a módszernél úgynevezett adatfeldolgozási típusfeladatokkal foglalkozunk (adatfelvitel, listázás, összegfokozatos listázás, másolás), amelyek a bemeneti adatstruktúrához, hozzárendelik a kimeneti adatstruktúrát. [-18-]

Specifikációorientált

Középpontban a specifikáció átalakítása van, formalizmusra, absztrakcióra, erős matematikai alapokra épít. [-18-]

Utasításorientált

Nem egy konkrét nyelv eszközein keresztül ismerkedünk a programozással, hanem egy általános nyelvtípus felhasználásával. A módszer meghatározza azokat a nyelvi elemeket, amelyekre építi a mintapéldáit. Például ha eljutunk az értékadó utasításig, akkor készítünk neki megfelelő egyszerű feladatokat. Nyelvi elemek lehetnek: lexikális egységek (többkarakteres szimbólum, szimbolikus nevek, címke, megjegyzés, literál); szintaktikai egységek (kifejezés); utasítások (üres, értékadó, vezérlést átadó, vezérlési szerkezetek, I/O utasítások); programegységek (alprogram, blokk, csomag, taszk); fordítási egységek; program. [-18-]

Matematikaorientált

Ez a módszer nem meglepően a matematikára épít, feladatait veszi alapul, és ezek felhasználásával ismerkedünk a programozás elemeivel. Jól megválasztott feladattípusok kellene, az egyszerűtől a bonyolultig. Csak kiegészítő jelleggel, más módszerekhez kapcsolatosan alkalmazva ajánlható az alkalmazása. [-18-]

Hardverorientált

A programozásoktatásnál nem az algoritmusok, adatmodellek fontosságát, szükségességét hangsúlyozza, így máris ellentmond a NAT és a Kerettantervi elképzeléseknek. Miszerint a közoktatásban az algoritmizálási képesség fejlesztésén van a hangsúly.

A módszer szerint a programozás tanulását a gép irányából kell kezdeni. Előbb ismerjük meg a processzor működését, majd a gépi utasítás fogalmát, a processzor

utasításkészletét, és ezek felhasználásával oldjuk meg a problémát. Így nemcsak a nyelvet köti (assembly), hanem a processzor architektúrát (CISC, RISC), sőt a processzort is. [-18-]

Problémaorientált

Lényege, hogy a tanulókat olyan gyakorlati „problémákkal”, feladatokkal szembesítjük, amelyek megoldása nem magától értetődő. A tanulókat gondolkodásra, kutatásra, csoportban való együttműködésre, felelősségvállalásra készíteti. A tanári szerep túlnyomórészt a feladatok, feladat irányok kijelölésében, illetve az ismeretszerzés lehetőségeinek feltárásában merül ki.

A problémaalapú tanítási módszer fokozza a tanulók teljesítményét az alábbi készségekben:

- Alkalmazkodás és részvétel a változásokban,
- A problémamegoldás alkalmazása új és jövőbeli helyzetekben,
- Kreatív és kritikus gondolkodás,
- A problémákra és helyzetekre irányuló holisztikus megközelítések elfogadása,
- A nézőpontok különbözőségének elismerése,
- Sikeres együttműködés a csoportban,
- A tanulási hiányosságok és erősségek felismerése,
- Az önirányító tanulás elősegítése,
- Hatékony kommunikációs készségek,
- Az alaptudás növekedése,
- Vezetői készségek,
- A különböző források kezelése.¹

A feladat nagyságától függően több órán, akár több hétig is dolgozhatnak rajta. Közben a kutatások részeredményeit kiselőadások formájában bemutatják egymásnak, és a tanárnak.

A programozásoktatásban való alkalmazását meg kell, hogy előzze az alapvető ismereteknek (algoritmusok, programozási nyelv, programtervezés) az elsajátítása. Ezért alkalmazására inkább csak a programozásra szánt idő vége felé van lehetőség.

¹ Probléma alapú tanulás, www.oki.hu/printerFriendly.php?tipus=cikk&kod=matrix-5-Problema

A feladatok kellően motiválóak, érdekesek, izgalmasak, és megoldhatóak legyenek. Építsen a meglévő tudásra, szükséges legyen új ismeretekre szert tenni, és elérhetőek legyenek ezek az ismeretek.

Például titkosító program készítése.

Algoritmizálás, programnyelvek

Mivel középiskolai programozásoktatásról van szó, nem lehet a célunk programozó palánták nevelése. Erre ott vannak a specializált iskolák, mint az OKJ-s programozói tanfolyamok, informatika főiskolák, egyetemek. Középiskolában elsődleges cél az algoritmikus gondolkodás kialakítása, fejlesztése. Itt nem lehet elsődleges szempont egy konkrét programozási nyelv teljes mélységben történő megismerése, slágernyelvek tanulása, bonyolult algoritmusok megtanulása.

A középiskolai programozásoktatáshoz, leginkább a *feladattípus-orientált és a módszeres algoritmusorientált* módszerek vegyítése alkalmazhatók eredményesen. Kevés óraszámnál alkalmazhatjuk tisztán a *feladattípus-orientált* módszert is.

Mivel a programkészítésnek a futtatható program a végterméke, ezért nem kerülhető ki a programozási nyelv és a fejlesztőkörnyezet sem. Másrészt nem szerencsés, ha csupa absztrakt algoritmusokat, feladat-specifikációkat, és pszeudokódokat lebegtetünk a tanulók szeme előtt. Tehát ne nyelvfüggetlenül tanítsuk a programozást!

Milyen mélységben algoritmizáljunk?

Minden programozás a feladat megértésével, pontos specifikálásával kezdődik, majd ez alapján jön a konkrét program létrehozása. Meg kell tervezni, el kell készíteni a feladat megoldását eredményező tevékenységek sorozatát, az *algoritmust*. Az algoritmus, még ha kicsi is, nem nélkülözheti a következő szabályok betartását.

Az algoritmus legyen:

- pontos – a feladat megoldását eredményezze,
- hatékony – ne csak megoldja, hanem optimális eredményt adjon az idő és tárfelhasználás tekintetében,
- egyértelmű – a tevékenységek megoldásánál zárjuk ki a véletlent,
- elronthatatlan – kezelje le a bemeneti értéktartományon kívüli adatokat is,

- felhasználóbarát – algoritmusnál inkább az érthetőségen a hangsúly
- teljes – a teljes feladatot megoldja,
- célorientált.

Vannak alapszintű programozási tételek, amelyeket nem hagyhatunk ki a tanítás során. Más algoritmusokat pedig azért kell beépíteni, hogy az emelt szinten érettségizni szándékozók is megtalálják számításaikat.

Alapvető algoritmusok, programozási tételek a tanítás során:

- vektor feltöltése,
- összegképzés,
- megszámlálás,
- eldöntés,
- kiválasztás,
- kiválogatás,
- csere,
- lineáris keresés,
- maximum- minimum kiválasztás,
- minimum-kiválasztásos rendezés.

Amelyekre érdemes kitérni az emeltszintű érettségi miatt:

- logaritmikus keresés,
- buborék-rendezés,
- beszúrásos rendezés,
- metszet- és unióképzés,
- összefuttatás,
- SOR és VEREM kezelés,
- mátrixműveletek.

Amelyekre érdemes kitérni a programozási versenyek miatt (fakultáció):

- rekurzió,
- gyorsrendezés,
- leszámláló rendezés,

- dinamikus lista,
- fabejárás (szélességi, mélységi),
- visszalépéses keresés.

Milyen programozási nyelvet válasszunk?

Amikor programozási nyelvet választunk programozás tanítása céljából, a követelmények eltérnek attól, amit a professzionális programozók támasztanak vele szemben. Röviden bemutatom milyen szempontrendszer áll a két megközelítés mellett.

Professzionális programozás igényei a nyelvvel, illetve a fejlesztő eszközzel szemben:

- megbízhatóság, hibátlanság: hivatkozási nyelv, implementáció,
- hatékony fejlesztési eszközök: objektum-orientáltság, újrafelhasználható elemek, minták, sablonkészítés, elérhető osztálykönyvtárak, grafikus felülettervezés, adatbázis kapcsolódási felület, webes lehetőségek, algoritmustervezési eszközök, stb.,
- tömörség,
- platformfüggetlenség,
- hardverprogramozás,
- széleskörű támogatottság.

Programozásként használt nyelvvel és fejlesztő eszközzel szembeni elvárások:

- nyelvi egyszerűség: Itt nem cél a nyelv minél teljesebb megismerése, és az sem, hogy egy feladat megoldására több programozási eszközt adjon a kezünkbe.
- érthető kulcsszavak: például az angol nyelv szavaiból képezve,
- szigorú típusosság: A felhasználás előtt vezessük be, deklaráljuk a saját programozási eszközeinket. Ezt követelje is meg a nyelv! Ne legyen automatikus deklaráció.
- átlátható, világos programszerkezet: Tanulás során előnyt jelent, ha a deklarációs- és a végrehajtható utasítások elkülönülnek, és az algoritmust kódoló utasításokat nem szakítják meg deklarációs utasítások.
- a tömörség inkább hátrány (a C többkarakteres operátor szimbólumai),
- megbízhatóság, hibátlanság,
- használhatóság,

Tanulásra alkalmas programozási nyelvek

Logo

A Logo nyelv parancsnyelv, a Lisp nyelv könnyebben olvasható adaptációja, melyet Wally Feurzeig és Seymour Papert készített. Megtalálhatók benne a listakezelés, fájlkezelés, I/O műveletek.¹

A nyelvhez tartozó grafikus fejlesztőeszköz kisiskolások által is érthetően „teknőccel” vezet be az algoritmizálás, programozás világába. Könnyen tanulható parancsokat használ, melyeket paraméterezhetünk is. A parancsokból eljárásokat állíthatunk össze, amelyek együtt alkotják a programot. Inkább általános iskolás korban alkalmazott eszköz.

Példaprogram:

Feladat: Belülről kifelé haladó négyzet rajzolása.²

```
eljárás rekA :h :db
  ha :db > 0 [
    ism 4 [e :h j 90]
    tf h 10 j 90 h 10 b 90 t1
    rekA :h+20 :db-1
  ]
vége
```

Hívása:

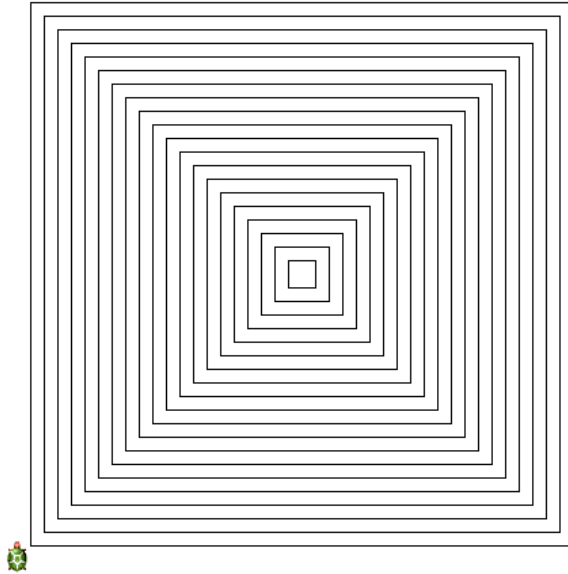
```
rekA 20 20
```

Az elkészített programot rögtön ki is próbálhatjuk.

Eredménye:

¹ [http://hu.wikipedia.org/wiki/Logo_\(programozási_nyelv\)](http://hu.wikipedia.org/wiki/Logo_(programozási_nyelv))

² <http://prog.berzsenyi.hu:8080/prog/View/logo/egyszerurek>



Ábra: A példaprogram futási eredménye.

Pascal

A Pascalt Niklaus Wirth professzor alkotta meg, oktatási célból. Jól megtervezett, strukturált programozást támogató nyelv. Főleg közép- és felsőoktatásban alkalmazták és alkalmazzák napjainkban is.

Az elterjedéséhez hozzájárult a *Borland* cég által fejlesztett *Turbo Pascal* (TP) programozási nyelv és az akkor modernnek számító integrált fejlesztési keretrendszer (IDE). A beépített debugger megkönnyítette a szemantikai hibák feltárását, a szintaktikai hibákat tartalmazó sorokat szintén megmutatta. Használhatósága miatt a professzionális programozásban is használták, ipari nyelvként. A programozási paradigmák változásai a nyelvet is utoléri, előbb megjelenik benne az objektumorientált filozófia. Majd az *Object Pascal*on alapuló vizuális felülettel rendelkező 4. generációs nyelvnek számító *Delphi*.

Manapság az oktatási intézmények egy ingyenes implementációt használhatnak, a Free Pascalt, amely szintén támogatja az objektumorientált programozást.

A nyelv előnyei:

- jól átgondolt felépítés,
- látható programhatárok: hol kezdődik, hol végződik,
- tagoltság: programnév, globális deklarációk, programtörzs, programvég,

- könnyen megjegyezhető szimbolikus nevek: program, begin, end, if, read, write,
- szigorú típusosság: az új neveket mindig deklarálni kell, nincs automatikus deklaráció,
- tagolható programszerkezet: alprogramok (eljárás, függvény), blokk (Begin... End), modulok (Unit),
- érthető vezérlési szerkezetek: feltételes- és ciklusutasítások.

Példaprogram:

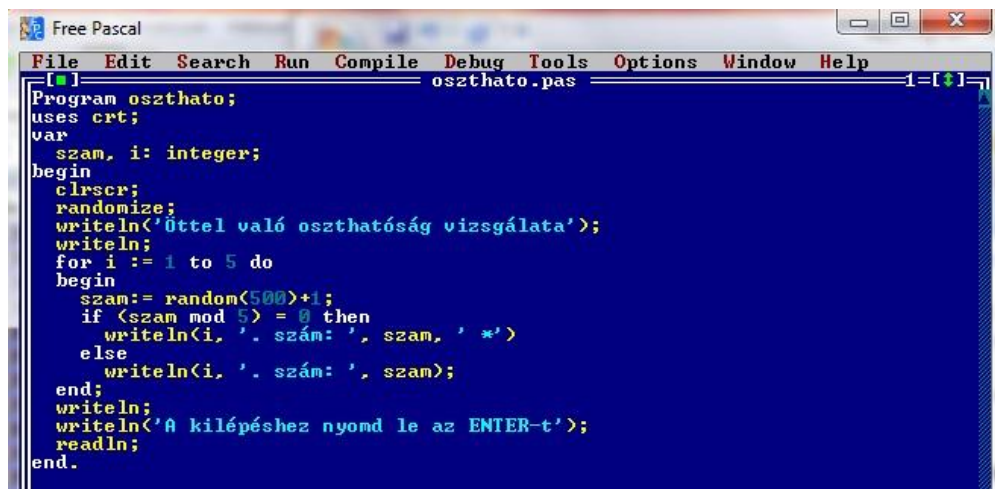
Feladat: Készíts programot, amely előállít öt véletlen számot 1 és 500 között, és az öttel oszthatóaknál kiír egy csillagot.

```

Program oszthato;           {program neve}
uses crt;                  {crt egység beépítése}
var
    szam, i: integer;       {változók deklarálása}
begin                       {program törzs kezdete}
    clrscr;                  {képernyőtörlés}
    randomize; {véletlenszám generátor inicializálása}
    writeln('Öttel való oszthatóság vizsgálata');
    {kiíratás}
    writeln;                 {soremelés}
    for i := 1 to 5 do      {ciklus 1-től 5-ig}
    begin
        szam:= random(500)+1; {véletlen szám előállítás}
        if (szam mod 5) = 0 then {oszthatóság vizsgálat}
            writeln(i, '. szám: ', szam, ' *') {ha osztható}
        else
            writeln(i, '. szám: ', szam,); {ha nem osztható}
        end;
    writeln;
    writeln('A kilépéshez nyomd le az ENTER-t');
    readln;                  {várakozás ENTER-re}
end.                       {program vége}

```

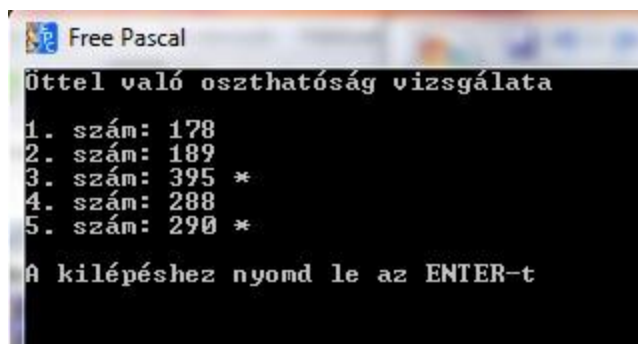
Fejlesztőkörnyezetben:



```
Free Pascal
File Edit Search Run Compile Debug Tools Options Window Help
[1] oszthato.pas 1=[+1]
Program oszthato;
uses crt;
var
  szam, i: integer;
begin
  clrscr;
  randomize;
  writeln('Öttel való oszthatóság vizsgálata');
  writeln;
  for i := 1 to 5 do
  begin
    szam:= random(500)+1;
    if (szam mod 5) = 0 then
      writeln(i, '. szám: ', szam, ' *')
    else
      writeln(i, '. szám: ', szam);
    end;
  writeln;
  writeln('A kilépéshez nyomd le az ENTER-t');
  readln;
end.
```

Ábra: Free Pascal fejlesztőeszköz.

Futási eredmény:



```
Free Pascal
Öttel való oszthatóság vizsgálata
1. szám: 178
2. szám: 189
3. szám: 395 *
4. szám: 288
5. szám: 290 *
A kilépéshez nyomd le az ENTER-t
```

Ábra: Az elkészített program futási képernyője.

Látható, hogy a fejlesztő környezet különböző színek használatával könnyíti meg a nyelv kulcsszavainak és a programozó által bevezetett eszköznevek, valamint szám és szöveg konstansok megkülönböztetését. Szintaktikai hiba esetén kijelzi a nem értelmezhető elemeket, valamint szemantikai hiba esetén a nyomkövetés lehetőségével segít feltárni a problémát. Ezen utóbbi elemek, valamennyi modern programfejlesztő eszközben megtalálhatóak.

Java

A Sun Microsystems, a JavaSoft fejlesztette ki James Gosling vezetésével. A C++-hoz képest sokkal biztonságosabb programozást lehetővé tevő, az objektumorientált paradigmát tisztábban tartalmazó programozási nyelv. Ipari nyelv, amely webes

felületre lett tervezve, de hamar rájöttek, hogy mobil platformon, és PC-s környezetben is kiválóan alkalmazható. A Java virtuális gép miatt hordozható kódot eredményez. A nyelv készítői és a programozók óriási méretű osztálykönyvtárakat hoztak létre, amelyek egy része szabadon hozzáférhető. Támogatja a komponens alapú, a vizuális, a hálózatos, az adatbázis alapú fejlesztéseket. Elterjedt vizuális fejlesztőeszközök: JBuilder (Borland/Inprise), JDeveloper (Oracle), IBM VisualAge for Java, Visual J++ (Microsoft). Jelenleg „slágernyelv”, az iparban széleskörűen alkalmazzák.

Jól átgondolt felépítés, szigorú típusosság, érthető programszerkezet, ingyenes terjesztés miatt az közoktatásban is kiválóan alkalmazható programozási nyelv. A jövőben várhatóan szélesebb körben megjelenik a középiskolákban is.

Példaprogram:

Feladat: A Hello World szöveg kiírása.

```
public class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Menteni kell *HelloWorldApp.java* néven, majd *javac HelloWorldApp.java* paranccsal előállítható belőle egy bájkódú *HelloWorldApp.class* állomány, amelyet a JVM (Java Virtual Machine – Java Virtuális Gép) értelmez és végrehajt a következő parancs által *java HelloWorldApp*.

JavaScript

Webes programozási nyelv. A kliens oldalon futó scriptek elsődleges nyelve. Ingyenesen használható, a hozzá tartozó legegyszerűbb fejlesztő eszköz a *notepad*, a futtató környezet pedig az *internet böngészőnk*. A JavaScript a Netscape Communications Corporation, azaz a kezdetben első számú böngésző (Netscape Communicator) fejlesztői alkották meg. Nyelvtanában hasonlít a „nagytestvér” Java nyelvre. Oktatásban való használatához alap HTML ismeret is szükséges.

A nyelv előnyei:

- jól átgondolt, érthető felépítés,

- könnyen olvasható programkód,
- azonnal kipróbálható a kész program: interpretált végrehajtás,
- grafikus felület, űrlapkészítés
- multimédia beépítés,
- a böngészők támogatják a programkód szintaktikai kiemelését, ellenőrzését (Mozilla Firefox, Opera, Google Chrome).

Példaprogram:

Feladat: A program jelenítse meg az aktuális időt és az univerzális időt egy weblapon.

```

<html>
<head><title>Idő kijelzés</title></head>
<body>
<h1>Aktuális dátum és idő</h1>
<p>
<script language="JavaScript" type="text/javascript">
    most = new Date();
    helyiido = most.toString();
    univerzalisido = most.toGMTString();
    document.write("<b>Helyi idő:</b> " + helyiido +
"<br>");
    document.write("<b>Univerzális idő:</b> " +
univerzalisido);
</script>
</p>
</body>
</html>

```

A program HTML kódba van beágyazva. Létrehoz egy dátum/idő objektumot (new Date()), majd átalakítja karakterlánc típusúra (toString()), és kiírja a weboldalra (document.write()).

Egy gyakorlati tananyag kidolgozása

Ebben a részben arra vállalkozom, hogy a diákoknak, és jómagamnak – a tanároknak – egy feladatgyűjteményt adjak a kezükbe, amelyet remélhetőleg eredményesen tudnak forgatni.

A tananyagot a feladattípus-orientált programozás-tanítási módszer szerint állítom össze, az előzőleg ismertetett előnyei miatt. Azért, hogy ne csak elméleti algoritmizálásról legyen szó egy programozási nyelv segítségével oldjuk meg feladatainkat. Úgy gondolom a Pascal (Free Pascal) nyelv az átgondolt felépítése miatt még mindig alkalmas ennek megvalósítására.

Amikor hozzákezdünk a gyakorlati programozáshoz, a tanulók már ismerik a szoftverfejlesztés életciklusát, az algoritmust, az algoritmus-leíró eszközöket, a programkészítés menetét.

A feladatsor előtt egy rövid ismertetést adok a Free Pascal IDE letöltéséről, telepítéséről, használatáról.

A gyakorlati feladatok csoportjai a következő rendező elv szerint fognak egymásra épülni:

- fejlesztőkörnyezet, programfordítás,
- deklaráció, konstans, változó bevezetésére szolgáló feladatok,
- szekvencia ismertetése egyszerű életből vett, és matematikai példákon, egész, valós, karakter, szöveg adattípusok, csere művelete,
- szelekció, logikai vizsgálatok, logikai típus fogalma, menükészítés,
- iteráció, tömb típus, vektor feltöltése,
- alprogram fogalma (eljárás, függvény),
- fájlkezelés,
- alapvető algoritmusok (eldöntés, megszámlálás, összegképzés, kiválasztás, kiválogatás, sorozatszámítások, szétválogatás, lineáris keresés, maximum- és minimum-kiválasztás),
- fejlettebb algoritmusok (rendezettség, rendezések, logaritmikus keresés, összefuttatás, sor- és veremkezelés, mátrix műveletek, rekurzió, gyorsrendezés, leszámláló rendezés, dinamikus lista).

Free Pascal fejlesztői környezet

Egy magas szintű programozási nyelven megírt *forrásprogramból* elő kell állítani a processzor számára is értelmezhető, *futtatható programot*. A forrásprogram egy szövegfájl, a futtatható program, pedig egy adott platformon (processzoron és operációs rendszeren) végrehajtható, gépi utasításokat tartalmazó állomány.

Erre két megoldás van: fordítóprogramos és az interpreteres. A fordítóprogramos technika teljes egészében vizsgálja a forrásprogramot, majd állítja elő a tárgykódot. Ez a kód még nem futtatható, de már a processzor által is értelmezhető utasításokat tartalmaz. Ez után jön a kapcsolatszerkesztés, amely általában több tárgykódú programból állít elő egy futtatható programot. Az interpreteres technika pedig a forrásprogramot szövegelemenként elemzi, majd állítja elő a gépi kódot, amelyet a processzor végrehajt.

A Pascal fordítóprogramos megoldást alkalmaz. A gépi kód generálására egy olyan integrált alkalmazást használnak, amely tartalmaz szövegszerkesztőt, fordítóprogramot, kapcsolatszerkesztőt, és más programozást segítő modult (pl. a hibák feltárását segítő nyomkövetőt). Ezeket a fejlesztőeszközöket IDE-nek (Integrated Development Environment) nevezik. Az általunk használt programozási eszköz is ilyen.

A Free Pascal fejlesztői környezet letölthető a következő linkről:
<http://ftp.hu.freepascal.org/pub/fpc/dist/2.4.0/i386-win32/fpc-2.4.0.i386-win32.exe>



Ábra: A Free Pascal fejlesztőeszköz (IDE).

Gyorstalpaló a Free Pascal IDE használatához

Érdemes programjainknak egy külön mappát készíteni, és ebbe menteni azért, hogy programjaink ne keveredjenek a Free Pascal IDE fájljaival. Majd állítsuk be a munkakönyvtárunkat (*File/Change dir...*), a mappánkra. Így a megnyitásnál és a mentésnél is ez lesz az alapértelmezett könyvtár.

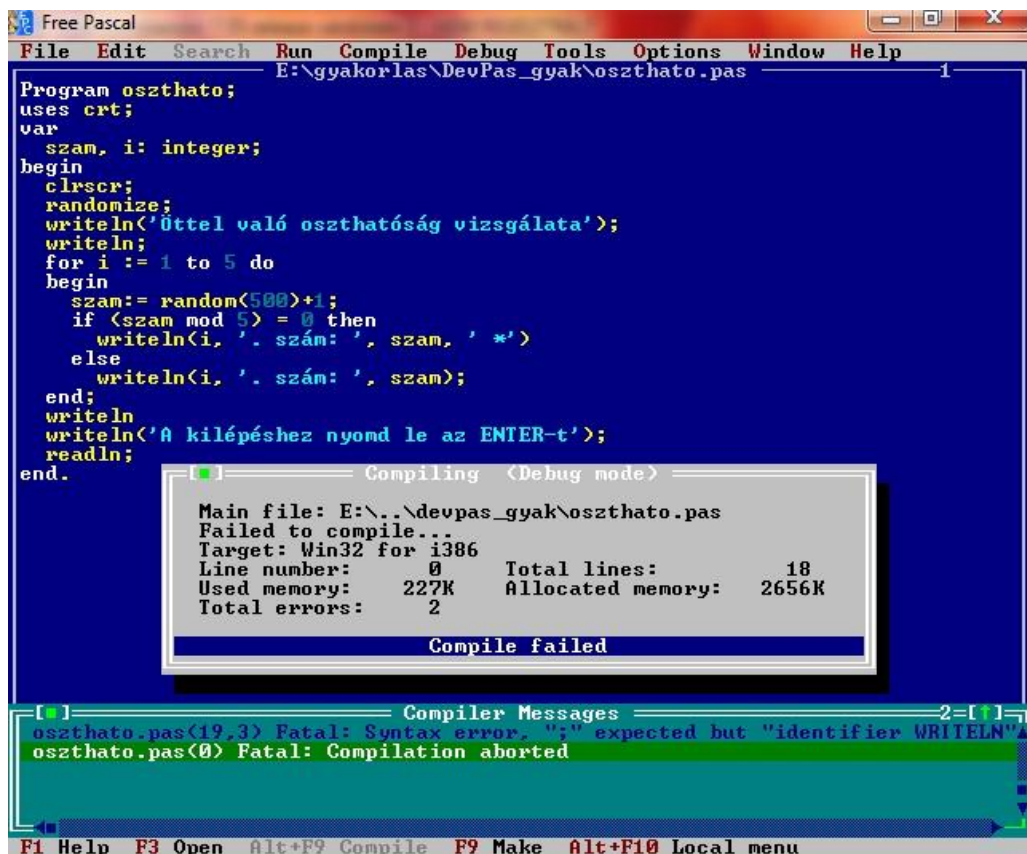
A (*File/New*) menüponttal kérjünk új szövegszerkesztő felületet. Rögtön mentsük is (*File/Save as...*). Egy már elkészített munkánkat a (*File/Open...*) paranccsal nyithatjuk meg.

Az Edit menüben a szokásos szerkesztési funkciók érhetők el. Ha a Windows vágólapjával akarunk kommunikálni, használjuk a (*Copy to Windows*, illetve a *Paste from Windows*) menüpontokat.

Amennyiben szükséges megnövelhetjük az ablak méretét a (*Options/Environment/Preferences*) pontok alatt. Érdemes 80*60-as szerkesztő felületet beállítani. Ugyanezt érhetjük el, ha az ablak címsorán jobb egérgombbal kattintunk, és a tulajdonságlapot választjuk.

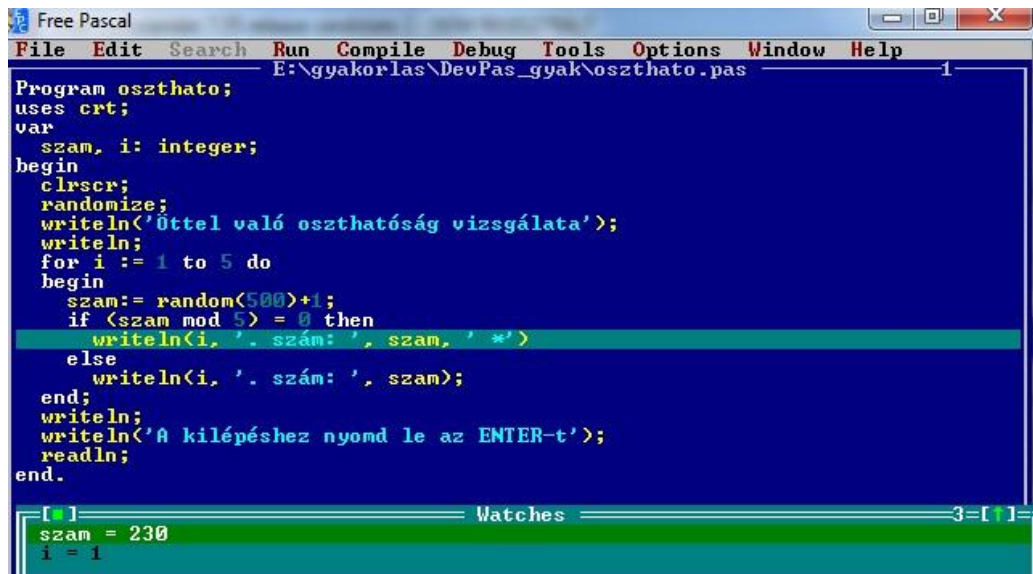
Az elkészült forrásfájlt a (*Run/Run*) menüvel futtathatjuk. Ez fordítást (*Compile*) és szerkesztést (*Linking*) végez.

A fejlesztőeszköz a forrásprogramban vétett *nyelvtani (szintaktikai) hibákat* kijelzi egy kis ablakban. Érdemes az első hibával kezdeni, mert a hibák továbbgyűrűznek. Azaz az első hiba (pl. deklarációs hiba) megszüntetése, megoldhatja a többi problémát is.



Ábra: Fordítási hiba jelzése.

Ha a programunk nem úgy működik, ahogy kéne, helytelen eredményt ad (*szemantikai* – logikai, algoritmusbeli hibát vétettünk), érdemes megnézni, hogy a program futása során hogyan módosul a változók tartalma. Kijelölhetünk változókat figyelésre a (*Debug/Add Watch*) paranccsal. Majd elindítjuk a nyomkövetést (*Run/Step over* vagy *Trace into*). Lehetőség van arra, hogy ne az elejétől lépkedjünk, hanem csak attól a ponttól, ahonnan hibára gyanakszunk. A (*Run/Goto Cursor*) végrehajtja a programot addig, ahol a kurzor áll a forrásprogramban, majd onnan végezhetjük a nyomkövetést. A hibakeresési módból való kilépésre a (*Run/Program Reset*) szolgál.



Ábra: Hibakeresés segítése.

A programozás során több ablak is lehet nyitva, de mindig a legfelül lévő az aktív. Az ablakokat egérrel, a bal felső sarokban lévő négyzetre való kattintással lehet bezárni.

Kezdő lépések, szekvencia, szelekció, iteráció

Minden olyan algoritmus, amelynek egy belépési és egy kilépési pontja van, felépíthető *szekvencia*, *szelekció* és *iteráció* segítségével. Amennyiben a program csak ezeket az elemeket tartalmazza, és nincs benne feltétel nélküli ugró utasítás, *strukturált programot* kapunk (Dijkstra).

Szekvencia

Amennyiben a programunk egymás után végrehajtandó utasítások sorozatából áll, szekvenciális szerkezetről beszélünk.

Feladat: Üres képernyőre írassuk ki a saját nevünket!

Először nézzük az algoritmust *pszeudokóddal*, vagyis mondatszerű leírással.

```
Program:
    képernyőtörlés
    Ki: 'Nagy Zsolt'
Program vége
```

A képernyőtörlést azért vettük fel, mert parancssoros ablakban futtatva a majdani programot tiszta, prompt nélküli felületet szeretnénk látni.

A Pascal program.

```
Program kiirasPr; {A program neve.}
Uses crt;         {Hivatkozunk már megírt kódra (modul).}
Begin             {A főprogram törzsének kezdete.}
    clrscr;       {Képernyőtörlés már elkészített kódja.}
    writeln('Nagy Zsolt'); {Kiíratom a nevem a képernyőre.}
    readln;       {Várakozás egy ENTER billentyűre.}
End.              {A főprogram törzsének vége.}
```

Kapcsos zárójelek között megjegyzést adhatunk meg, amely csak a forrásprogram olvasójának szól, a fordító nem veszi figyelembe.

A *clrscr*, *writeln*, *readln* már előre elkészített program részletek (alprogramok lásd később), amelyeket már csak a feladatuknak megfelelően kell használni.

A *clrscr* eljárás törli a képernyőt.

A *writeln* eljárás a zárójelben szereplő paramétereit kiírja a képernyőre (standard kimenetre). Amennyiben több paramétere van, azokat vesszővel kell elválasztani egymástól. Tudunk vízszintesen balra igazítani úgy, hogy a paraméter után kettősponttal elválasztva megadjuk a mezőszélességet. Például *writeln('Nagy Zsolt': 10)*. A tizedes tört alakú számok kiírásánál megadhatjuk a tizedesek számát a mezőszélesség után. Például *writeln(10.1234:10:2)*.

A *readln* eljárás adatok billentyűzetről (standard input) történő beolvasására való. A beolvasott adato(ka)t a paraméterként megadott változó(k)ban tároljuk. Paraméter nélkül a program készítési fázisában arra használjuk, hogy a lényegi részek lefutása után a futási képernyő ne záruljon be. Csak tesztelési célra!

Feladat: Kérjünk be két számot, és írjuk ki az összegüket!

Ez a feladat a változók és az adattípusok fogalmát vezeti be.

Pszudokód:

```
Program:
  Be: szaml [egész]
  Be: szaml [egész]
  összeg:= szaml + szaml
  Ki: összeg ['Eredmény= 'összeg formában]
Program vége
```

A program = algoritmus + adatok. Minden program adatokkal dolgozik. Ezek az adatok a végrehajtás előtt és után a számítógép operatív memóriájának adott helyén (adatszegmens) tárolódnak. Ezek a helyek címük alapján érhetők el, amely hexadecimális számrendszerben adható meg. Mivel a modern operációs rendszerek a tárkezelést részben saját hatáskörben végzik, érdemes rájuk, és a program működését koordináló futtató rendszerre bízni az adatterületek kijelölését. A *hexadecimális* memóriacímek nem informatívak, nem adnak jelentést a mögöttük lévő adatoknak. Így a magas szintű programozási nyelvekben bevezették a *változó* fogalmat, amely egy nevesített memória részként magyarázható.

Minden saját, névvel rendelkező programozási eszközt deklarálni kell. A deklaráció alapján rendeli hozzá a *fordítóprogram* a memóriaterülethez, a használat módjára vonatkozó előírásokat. A *deklaráció*, deklarációs utasítás segítségével történik.

A név egy *azonosító*, amelyre szabályok vonatkoznak. Csak az angol abc (kis és nagy) betűivel, és az aláhúzás jellel kezdődhet, de az első karakter után számokat is tartalmazhat. A kis- és nagybetűk nincsenek megkülönböztetve. Akármilyen hosszú lehet, de csak az első 63 karakter számít.

Minden memóriában tárolt adatnak, így a változó tárterületének is *deklarált* (meghatározott) *adattípusa* van. Ez a tulajdonság megadja a fordító programnak, hogy milyen feltételekkel lehet használni az adott memóriarészt. Milyen értéktartományból vehet fel adatokat, milyen műveletek végezhetőek vele, és milyen formában (bitkombinációban) tárolódik a memóriában?

Pszudokódban a beviteli adatokra vonatkozó megszorításokat, illetve kivitel formáját szögletes zárójelben adjuk meg.

Pascal kód:

```
Program osszegPr;
Uses crt;
Var
    szam1,    szam2,    osszeg:  Integer;  {Egész típusú
változók.}
Begin
    clrscr;
    write('Első szám: '); {Információs szöveg kiíratása.}
    readln(szam1);        {Bekérjük az első számot.}
    write('Második szám: ');
    readln(szam2);
    osszeg:= szam1 + szam2; {Értékadás.}
    writeln('Eredmény= ', osszeg); {eredmény kiíratása,}
    readln;
End.
```

A *szam1*, *szam2*, *osszeg* nevek a programban változók lesznek. A változó értéke a program futása során többször is megváltozhat. A program adott pontján mindig az aktuális értékével szerepel. Az *Integer* adattípus előjeles egész, amely értéktartománya -32768..32767 közötti egész szám lehet. Összegképzésre a matematikában megszokott + operátor használatos. Kiszámoljuk az összeget, majd a képződött eredményt eltároljuk az *osszeg* változóban. Az érték átadás műveletét az értékadó (:= olvasva legyen egyenlő) operátorral valósítjuk meg. Előbb meghatározódik az értékadó operátor jobb oldalán lévő kifejezés eredménye, majd átadódik a baloldalon álló változónak. Az

eredmény *értékadás szerint kompatibilis* kell legyen az operátor bal oldalán lévő, értéket felvevő változó típusával. Vagyis csak olyan változóba tehetünk be adott értéket, amelynek az *értéktartománya* tartalmazza azt.

A *writeln* eljárásnak most két paramétere van, amelyeket vessző választ el. Az első paraméter egy aposztrófok között megadott szöveg, a második a kiíratandó eredményt tartalmazó változó. Figyeljük meg, hogy a változó nevét (*osszeg*) nem tettük aposztrófok közzé. A változó neve kifejezésben, vagy az értékadó utasítás jobb oldalán használva mindig a memóriabeli értéket (értékkomponens) jelenti. Az értékadó kifejezés bal oldalán használva tárolóként viselkedik (címkomponens).

A *write*('szöveg') és a *writeln*('szöveg') eljárások között az a különbség, hogy a *writeln* a paraméter kiírása után új sort kezd.

Feladat: Készítsünk programot, amely kiszámítja egy felhasználótól bekért sugárú kör kerületét, területét! A bekérést centiméterben, a kiírást milliméterben és mm^2 -ben végezzük.

Ebben a feladatban valós számokat használunk.

Pszeudokód:

```
Program:
  Be: sugár [valós szám]
  sugár := sugár * 10
  kerület:= 2 * sugár * PI
  terület:= sugár * sugár * PI
  Ki: kerület ['Kerület: 'érték formában]
  Ki: terület ['Terület: 'érték formában]
Program vége
```

Az algoritmus harmadik sorában, a centiméterben megadott sugarat átváltjuk milliméterbe. Emlékeztetőül, előbb kiértékelődik az értékadó utasítás jobb oldalán lévő kifejezés (előáll egy érték), majd átadódik az utasítás jobb oldalán álló változónak. Amely ezen túl az új értékével vesz részt a műveletekben (amíg újra meg nem változik).

Pascal kód:

```
Program korPr;
Uses crt;
```

```

Var
    sugar, kerulet, terület: Real; {Valós változók.}
Begin
    clrscr;
    writeln('Számítsuk ki a kör kerületét és
területét!');
    write('Kérem a kör sugarát: ');
    readln(sugar);
    sugar:= sugar * 10;           {Átváltás milliméterbe.}
    kerulet:= 2 * sugar * PI;     {PI függvény.}
    terület:= sugar * sugar * PI;
    writeln('Kerület: ', kerulet:0:2);
    writeln('Terület: ', terület:0:2);
    readln;
End.

```

A *sugar*, *kerulet*, *terulet* változók valós típusúak. A *Real* típusban előjeles tört számokat tárolhatunk 6 bájtton, a legkisebb (0-hoz legközelebb álló) abszolút érték 2.9E-39, a legnagyobb pedig 1.7E38. A maximális pontosság 11-12 jegy, az ennél több tizedes jegyet nem tárolja. A valós számot a *writeln*-ben kiírva lebegőpontos formában jelenik meg, ha ezt nem akarjuk, adjuk meg a tizedesek számát. Amennyiben nem szeretnénk igazítani, mezőszélességnek nullát adjunk. A szám egész része nem csonkul. A példában (kerulet:0:2) két tizedes jeggyel történik a kiírás.

Feladat: Készítsünk programot, amely kiszámítja egy nettó áru termék bruttó, ÁFA-val megnövelt értékét! Az ÁFA fix 20%.

Bevezetjük a konstans fogalmát.

Pszeudokód:

```

Program:
    ÁFA = 20
    Be: nettó ár [valós]
    bruttó ár:= nettó ár + nettó ár * ÁFA / 100
    Ki: bruttó ár ['Bruttó ár: 'érték' Ft' formában]
Program vége

```

Az ÁFA egy konstans érték lesz.

Pascal kód:

```
Program bruttoPr;
Uses crt;
Const
    AFA = 20;          {Konstans, az értéke nem változhat.}
Var
    netto: Real;       {Valós adattípus.}
Begin
    clrscr;
    writeln('Bruttó ár számítása, ÁFA tartalom 20%.');
    write('Add meg a nettó árat: ');
    readln(netto);
    brutto:= netto + netto * AFA / 100;
    writeln('Bruttó ár: ', brutto, '%');
    readln;
End.
```

A *konstans* egy olyan programozási eszköz, amelynek az értéke deklaráláskor (a név bevezetésekor) eldől, és a program futása során változatlan marad. Az adattípus a deklaráláskor az értéknek megfelelően rendelődik hozzá.

A programban az AFA egy konstans, amelynek az értéke 20. Ezt a program teljes ideje alatt megőrzi. Egyezményesen a konstans nevét csupa nagybetűvel írjuk.

Feladat: Készítsünk programot, amely megcseréli két egész típusú változó tartalmát, majd megjeleníti a képernyőn!

A csere művelet bemutatása.

Pszeudokód:

```
Program:
    Be: első [egész]
    Be: második [egész]
    segéd:= első
    első:= második
    második:= segéd
    Ki: első ['Első változó tartalma: 'szám formában]
    Ki:  második ['Második változó tartalma: 'szám
formában.]
Program vége
```

A csere műveletét egy *segéd* változó felhasználásával végezzük. Emlékezzünk rá, hogy a változónak történő értékadás során az előző értéket már nem érjük el. Az véglegesen felülíródott. Ezért az egyik változó tartalmát ideiglenesen kimentjük egy harmadik változóba.

Pascal kód:

```
Program cserePr;
Uses crt;
Var
    elso, masodik, seged: Integer;
Begin
    clrscr;
    writeln('Megcseréljük két változó tartalmát.');
```

{Informatív beolvasások.}

```
    write('Add meg az első változó értékét: ');
    readln(elso);
    write('Add meg a második változó értékét: ');
    readln(masodik);
```

{A csere algoritmusa.}

```
    seged:= elso;
    elso:= masodik;
    masodik:= seged;
```

{Informatív kiírások.}

```
    writeln('Csere után');
    writeln('Első változó tartalma: ', elso);
    writeln('Második változó tartalma: ', masodik);
    readln;
```

```
End.
```

A megjegyzéseket külön sorba is írhatjuk. A program tagolása is segítheti a későbbi olvasást. Adjunk kellő információt a felhasználóknak, hogy mikor mit csinál, mit kér a program.

Feladat: Készítsünk olyan programot, amely bekéri egy felhasználó nevét, születési évét, és kiírja hány éves! Az aktuális év nevesített konstansként legyen megadva.

Bevezetjük a szöveg típusú változó fogalmát.

A szöveg *string* típusú változó egy vagy több karakter (szöveg) tárolására alkalmas. A tárolás sorfolytonosan történik, deklaráció során a típusnév után megadhatjuk szögletes

zárójelben a maximális tárolási hosszát. Amennyiben nem adjuk meg, 255 byte lesz lefoglalva. A nulladik bájton karakter formában a szöveg aktuális mérete van.

Pszudokód:

```
Program:
    AKTUÁLISÉV = 2010
    Be: név [szöveg]
    Be: születésiév [egész]
    éves:= AKTÁLISÉV - születésiév
    Ki: éves [név érték 'éves vagy' formában]
Program vége
```

Az AKTUÁLISÉV nevesített konstans, amely a programban a hozzá rendelt értékkel vesz részt.

Pascal kód:

```
Program evekPr;
Uses crt;
Const
    AKTEV = 2010;
Var
    nev: string[30];
    szulev, éves: word;
Begin
    clrscr;
    writeln('Hány éves vagy?');

    {Adatbekérések}
    write('Neved: ');
    readln(nev);
    write('Születési éved: ');
    readln(szulev);

    {A nevesített konstanst használjuk a számításhoz.}
    éves:= AKTEV - szulev;

    {A kiíratásnál a nev és az éves változók, a többi
    sztring literál.}
    writeln(nev, ' ', éves, ' éves vagy. ');
    readln;
End.
```

A szting típusú *nev* változót ugyanúgy használhatjuk, mint bármely más változót, csak az értéke karakterek sorozata lesz.

Folytatás a mellékletben!

Zárszó

A szakdolgozat első részében az informatikaoktatás, programozásoktatás helyzetét próbáltam bemutatni. Kitértem az oktatás kereteit adó tantervekre is. Részletesen bemutattam, hogy a programozásoktatásánál milyen módszereket választhatunk, milyen mértékben érdemes algoritmizálni.

A második és egyben legnagyobb részben a gyakorlati programozásoktatásához szerettem volna segítséget nyújtani, elsősorban a diákoknak, de nem utolsó sorban a tanároknak is. Úgy érzem, a középiskolában elvárható algoritmusokat, beleértve az emelt szintű érettségi követelményét is, teljes mértékben felöleli. Végül egy feladatsort is összeállítottam, amelyből szemezgethetnek diákok és tanárok egyaránt.

A későbbiek folyamán szeretném átalakítani webes formára (multimédiás elemekkel kiegészítve) a feladatokat, illetve egy-két érdekesebb algoritmussal (fabejárás, visszalépéses keresés) is kibővíteni.

Irodalomjegyzék

1. Kőrösné Mikis Márta: Informatikatanítás a középiskolába – A 2003-as obszervációs felmérés tapasztalatai, <http://www.ofi.hu/tudastar/tantargyak-helyzete/informatikatanitas>, 2010.09.10.
2. Lányi András, Melega Kálmán, Reményi Zoltán, Varga Attila: Betolakodó, vagy várt vendég? – Kerekasztal-beszélgetés az informatika tanításáról, <http://www.ofi.hu/tudastar/betolakodo-vart-vendeg>, 2010.09.11.
3. Kőrösné Mikis Márta: Az informatika helyzete és fejlesztési feladatai, <http://www.ofi.hu/tudastar/tantargyak-helyzete/informatika-helyzete>, 2010.09.14.
4. Páli Judit, Farkas Károly, Theisz György, Kőrösné Mikis Márta: Az informatika tantárgy, vagy szemlélet? – Rendhagyó beszélgetés az informatika oktatási hatásairól, <http://www.ofi.hu/tudastar/informatika-tantargy>, 2010.09.11.
5. Nagy Ádám Ph.D.: Informatikus lesz... s katona, <http://www.ofi.hu/tudastar/informatika-oktatasban/informatikus-leszel>, 2010.09.21.
6. Tompa Klára: Az informatikai műveltség és az informatikaérettségi szakértői megítélése, <http://www.ofi.hu/tudastar/informatikai-muveltseg>, 2010.09.19.
7. Benczúr András: Informatika – oktatás – informatikaoktatás, Természet Világa, 2000 II. különszám, <http://www.kfki.hu/chemonet/TermVil/>, 2010.10.12.
8. Szabó László Tamás: Tantervelmélet, Kossuth Egyetemi Kiadó, Debrecen, 2006
9. Kovács Györgyi, Rozgonyi-Borus Ferenc: Az informatika oktatás története, 2001, <http://www.abax.hu/inlap/t/cikk/inftori.htm>, 2010.08.23.
10. Kerber Zoltán: A kerettanterv hatása a NAT-ra és a helyi tantervre, <http://www.ofi.hu/tudastar/kerettanterv-hatasa-nat>, 2010.10.16.
11. Kőrösné Mikis Márta, Végh András, Bánhidi Sándorné: Gondolatok az informatika-kerettanterv kapcsán, <http://www.ofi.hu/tudastar/gondolatok-informatika>, 2010.10.16.
12. 243/2003. (XII. 17.) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról, <http://www.math.klte.hu/~kovacs/nat.pdf>, 2010.10.23.
13. Informatika kerettantervek – gimnáziumok, szakközépiskolák, szakiskolák részére – 2003, <http://www.nefmi.gov.hu/kozoktatas/tantervek/kerettantervek>, 2010.10.23.

14. Kerettanterv a szakközépiskolák 9-12. évfolyama számára, Magyar Közlöny, 2008/20/II. szám/832-845. oldal, <http://www.ntk.hu/kozepiskola/oktatastamogatas/kerettanterv>, 2010.10.23.
15. Dr. Nyéki Lajos: Az informatika oktatásának módszertana, Széchenyi István Egyetem, Győr, <http://eki.sze.hu/ejegyzet/ejegyzet/infomod/>, 2010.10.29.
16. Szabó László Tamás: Didaktika Szöveggyűjtemény, Kossuth Egyetemi Kiadó, Debrecen, 2006
17. Falus Iván: Didaktika – Elméleti alapok a tanítás tanuláshoz, Nemzeti tankönyvkiadó, Budapest, 2001
18. Szlávi Péter, Zsakó László: Programozás tanítási módszerek, ELTE TTK Informatika Szakmódszertani Csoport
19. Szlávi Péter, Zsakó László: Módszeres programozás, Műszaki Könyvkiadó, Budapest, 1986
20. Probléma alapú tanulás, www.oki.hu/printerFriendly.php?tipus=cikk&kod=matrix-5-Problema, 2010.11.13.
21. Angster Erzsébet: Programozás Tankönyv I-II. – Strukturált tervezés Turbo Pascal, 4KÖR Bt. kiadó, 1995
22. Benkő Tiborné, Tóth Bertalan: Együtt könnyebb a programozás – Free Pascal, ComputerBooks, Budapest, 2006
23. Angster Erzsébet: Objektumorientált Tervezés és Programozás I-II. – Java, 4KÖR Bt. kiadó, 2002
24. Michael Moncur: Tanuljuk meg a JavaScript használatát, Kiskapu, Budapest, 2006
25. Logo nyelv: [http://hu.wikipedia.org/wiki/Logo_\(programozási_nyelv\)](http://hu.wikipedia.org/wiki/Logo_(programozási_nyelv)), 2010.11.20.
26. Logo programozás: <http://prog.berzsenyi.hu:8080/prog/View/logo>, 2010.11.20.

Melléklet

Gyakorlati tananyag folytatása

Szelekció

A szelekció feltételtől függő program elágazást jelent. A feltétel egy logikai kifejezés, amely kiértékelődése után, vagy *igaz*, vagy *hamis* logikai állapotot vesz fel. Az alap, *kétágú* szelekciós utasítás esetén, ha a feltétel *igaz*, az első ágon folytatódik a program, *hamis* esetén a másikon.

Feladat: Készítsünk programot, amely egy felhasználó által megadott egész számról eldönti, hogy páros-e! A program legyen felhasználóbarát!

Egy szám párosságát úgy tudjuk megvizsgálni, hogy elosztjuk kettővel, és megnézzük az osztási maradékát. Amennyiben a maradék nulla, páros a szám, ellenkező esetben páratlan. Pascalban a MOD operátor maradékos osztást tesz lehetővé.

Tehát a feltétel: $(\text{szám} \bmod 2 = 0)$

Egy másik lehetőség, ha az ODD(szám) függvényt használjuk, amely a paraméterként megadott *szám* páratlanságát vizsgálja. Amennyiben letagadjuk, NOT ODD(szám), akkor fog igazat visszaadni, ha páros a *szám*.

Igaz, hogy karakteres felületen dolgozunk, de ilyenkor is ügyelni kell arra, hogy az elkészített program a felhasználó számára könnyen értelmezhető, a használat átlátható, világos legyen. Figyeljünk arra, hogy a program futási képernyője kellően tagolt, és informatív képet adjon.

Pszeudokód:

```
Program:
  Be: szám [egész]
  Ha szám osztható kettővel, akkor
    Ki: 'Páros'
  egyébként
    Ki: 'Páratlan'
  Elágazás vége
Program vége
```

Pascal kód:

```
Program parosPr;
Uses crt;
Var
    szam: Integer;
Begin
    clrscr;

    {Értesítsük a felhasználót a program céljáról.}
    writeln('A program egy szám párosságát vizsgálja.');
```

 {A felhasználó mindig tudja, hogy mikor mit kell megadnia.}

```
    write('Kérem a számot: ');
    readln(szam);

    {A párosság vizsgálat.}
    If szam MOD 2 = 0 Then
        writeln('Páros')
    Else
        writeln('Páratlan');
    readln;
End.
```

A $szam \text{ MOD } 2 = 0$ egy *feltételes kifejezés*. A kifejezés egy olyan szintaktikai eszköz, amelynek feladata, hogy a program adott pontján egy értéket állítson elő. A kifejezésben operátorokat (műveleti jel) használhatunk a műveletek elvégzésére. Amennyiben több operátort is tartalmaz, a kiértékelődés adott szabályok szerint megy végbe. Az operátorok között végrehajtási sorrendet (precedencia sorrendet) határoztak meg. A nyelv definiálja! Az az operátor, amelyiknek nagyobb a precedenciája, előbb van végrehajtva. Abban az esetben, ha két azonos precedenciájú operátor van egy kifejezésben egymás mellett, a kötési sorrend szabálya (asszociativitás szabály) dönt. Pascalban a kötési irány balról-jobbra.

A precedencia sorrend megváltoztatására zárójeleznünk kell, a matematikában megszokott kerek zárójelekkel. Így a műveletek végrehajtása a zárójelen belüli részkifejezésekkel kezdődik.

Jelen esetben azért nem kell zárójeleket használni, mert a MOD operátor erősebb, mint az egyenlőségvizsgáló. Használhatunk redundáns zárójeleket, ha nem vagyunk biztosak az operátorok erőssorrendjében, vagy csak átláthatóbb lesz tőle a kifejezés.

A $(\text{szam} \bmod 2) = 0$, de a $((\text{szam} \bmod 2) = 0)$ kifejezés is helyes.

Vigyázzunk, hogy az *else* előtt nem szerepelhet pontosvessző, különben új utasítást kezdenénk. Ilyen utasítás pedig nincs a Pascal-ban. Ez szintaktikai (nyelvi) hibát jelentene, amelyre a fordítási hibát kapnánk, és nem készülne el a tárgykód.

A program forrásszövegére vonatkozó formai, „nyelvtani” szabályok összességét *szintaktikai szabályoknak* hívjuk. Az ilyen jellegű hibákat *szintaktikai hibáknak*. Ezek a hibák a fordítás során kiderülnek. A fejlesztőeszközök segítenek a hiba helyének lokalizálásában is.

A tartalmi, értelmezési, jelentésbeli szabályok alkotják a *szemantikai szabályokat*. A *szemantikai hibák* csak a program tesztelése során derülhetnek ki.

Feladat: Készítsünk programot, amely bekér két számot, majd kiírja az összegüket, de csak abban az esetben, ha a két szám közül pontosan az egyik páros! A program felhasználóbarát legyen!

Ez a feladat összetett feltételt mutat be. A „két szám közül pontosan az egyik páros” kijelentés azt fejezi ki, hogy vagy a szám1, vagy a szám2 páros, de egyszerre mindkettő nem.

Összetett feltételt kapunk, ha két vagy több feltételt, logikai operátorokkal kapcsolunk össze. A Pascal-ban használható kétoperandusú logikai operátorok: AND, OR, XOR.

A számunkra szükséges feltétel: $(\text{szam1} \bmod 2 = 0) \text{ XOR } (\text{szam2} \bmod 2 = 0)$.

A zárójelezés szükséges, mert a kizáró vagy (XOR) művelet magasabb precedenciájú, mint az egyenlőségvizsgálat (=) operátor. A kifejezés kiértékelése rész-kifejezésenként történik. Előbb meghatározódik a bal oldali részkifejezés, majd az XOR jobb oldalán lévő. Mindkét esetben előáll egy logikai érték (TRUE, FALSE), majd azt követően meghatározódik a teljes kifejezés értéke. A teljes kifejezés abban az esetben lesz igaz (TRUE), ha csak és kizárólag az egyik részkifejezés igaz, a másik pedig hamis (FALSE).

Pszeudokód:

```
Program:
  Be: szám1 [egész]
```

```
Be: szám2 [egész]
Ha csak az egyik páros, akkor
    összeg:= szám1 + szám2
Ki: összeg [Összeg: érték formában]
egyébként
    Ki: 'Az egyedi párosság nem teljesült, ezért nem
számolunk'
Elágazás vége
Program vége
```

Pascal kód:

```
Program egyParosPr;
Uses crt;
Var
    szam1, szam2: Integer;
Begin
    clrscr;
    writeln('A program bekér két számot, és csak abban az
esetben írja ki a két szám összegét, ha kizárólag az
egyik páros.');
```



```
    {A két szám bekérése.}
    write('Első szám: ');
    readln(szam1);
    write('Második szám: ');
    readln(szam2);
```



```
    {Feltétel vizsgálat.}
    If (szam1 MOD 2 = 0) XOR (szam2 MOD 2 = 0) Then
        writeln('Összeg: ', szam1 + szam2)
    Else
        writeln('Az egyedi párosság nem teljesült, ezért
nem számolunk.');
```

```
        readln;
    End.
```

A *writeln* eljárásba kifejezést is írhatunk.

Feladat: Készítsünk programot, amely bekér egy hőmérsékleti értéket és kiírja, hogy a víz ezen a hőmérsékleten milyen halmazállapotú lenne! ($t \leq 0$ esetén szilárd, $0 < t < 100$ esetén folyékony, $t \geq 100$ esetén gőz)

Ez a feladat többágú szelekcióra ad példát.

Pszeudokód:

```
Program:
  Be: hőmérséklet [egész]
  Elágazás
    hőmérséklet <= 0 esetén
      Ki: 'szilárd'
    0 < hőmérséklet < 100 esetén
      Ki: 'folyékony'
    egyébként
      Ki: 'gőz'
  Elágazás vége
Program vége
```

Pascal kód:

```
Program allapotPr;
Uses crt;
Var
  t: Integer;
Begin
  clrscr;
  writeln('Adj meg egy hőmérsékletet, és a program
megmondja, hogy a víz milyen halmazállapotú.');
```



```
  {Adat bekérése.}
  write('Hőmérséklet: ');
  readln(t);

  {Három kimenetű szelekció.}
  If t <= 0 Then
    writeln('Szilárd')
  Else If (t > 0) AND (t < 100) Then
    writeln('Folyadék')
  Else
    writeln('Gőz');
  readln;
End.
```

Ez az *If* szerkezet lényegében egy olyan kétágú szelekció, amelynek a második ágában szintén egy kétágú szelekció van.

Feladat: Készítsünk olyan programot, amely a felhasználó választásától függően elvégzi a négy alapművelet valamelyikét, a felhasználótól bekért két számon!

A feladattal a menükészítést tanuljuk meg.

A „menü” egy választási lehetőséget kínál a felhasználó számára, és a választástól függően hajtja végre az egyik vagy másik lehetőséget.

Pszedokód:

```
Program:
  Be: szaml, szam2 [egész]
  Ki: Felkínált lehetőségek megjelenítése
  Be: Választás
  Választott művelet végrehajtása
  Ki: Eredmény
Program vége
```

Pascal kód:

```
Program menuPr;
Uses crt;
Var
  szaml, szam2: Integer;
  muvelet: Char;
Begin
  clrscr;
  writeln('A program bekér két számot, és a felhasználó
választásától függően végrehajt egy műveletet.');
```



```
  {Menükészítés}
  writeln;
  writeln('Összeadás esetén +');
  writeln('Kivonás esetén -');
  writeln('Szorzás esetén *');
  writeln('Osztás esetén /');
```



```
  {Adatok bekérése}
  writeln;
  write('Első szám: ');
  readln(szaml);
  write('Második szám: ');
  readln(szam2);
```



```
  {Választott művelet bekérése}
  writeln;
  write('Válassz egy műveletet: ');
  readln(muvelet);
```



```
  {Választás}
  writeln;
  Case muvelet Of
    '+': writeln('Összege: ', szaml + szam2:10);
```

```
'-': writeln('Különbség: ', szam1 - szam2:10);
'*': writeln('Szorzata: ', szam1 * szam2:10);
'/': writeln('Hányados: ', szam1 / szam2:10:2);
Else
    writeln('Helytelen karaktert választott!');
End;
readln;
End.
```

A programban az üres *writeln* utasításokkal a kimeneti képernyőn egy üres sort jelenítünk meg. A *writeln*-ben számítjuk és írjuk ki a kifejezés eredményét. A kifejezés után megadtuk a mezőszélességet (10), és a tizedesek számát (2) is.

A *Char* típus egy (alap esetben ASCII) karakter tárolására alkalmas. A karaktert megadhatjuk aposztrófok között ('+'), illetve a karakter kódtáblában elfoglalt sorszámával (#43).

A *Case* utasítás egy érték (muvelet) alapján fog választani, és a választáshoz tartozó utasításokat végrehajtani.

Iteráció

Az iteráció olyan algoritmusforma, amely utasítás(ok) feltételtől függő ismételt végrehajtását teszi lehetővé.

Fajtái:

- előfeltételes,
- végfeltételes,
- számláló, vagy növekményes.

Előfeltételes ciklus esetén a program még a ciklusba való belépés előtt megvizsgál egy feltételt, majd annak teljesülése esetén végrehajtja a *ciklus törzsének* utasításait, ellenkező esetben kilép a ciklusból. A feltételt *belépési feltételnek* nevezzük. A ciklusnak képesnek kell lennie arra, hogy a feltétel eredményét megváltoztassa, különben *végtelen ciklust* kapunk. Az előfeltételes ciklust olyan feladatok megoldására használjuk, amelyeknél az induló feltétel határozza meg a ciklusmag végrehajtásának szükségességét.

Végfeltételes ciklus esetén a ciklus magja egyszer mindenképpen végrehajtódik. Majd a ciklusmag lefutása után történik a feltétel kiértékelése, ami eldönti, hogy kilépünk-e (igaz eset) a ciklusból, vagy nem (hamis eset). Végfeltételes ciklust olyan esetekben használunk, amelyeknél a ciklus törzsének legalább egyszeri végrehajtása biztos.

A *számláló ciklus* egy speciális előfeltételes ciklus, amelynél az ismételt végrehajtás számát a programozó határozza meg. A Pascal-ban egy *ciklusváltozó* kezdeti és végértéke közötti egyel történő inkrementálás, vagy dekrementálás száma határozza meg az ismétlések számát.

Feladat: Készítsünk programot, amely egy pozitív tízes számrendszerbeli számot átszámít kettes számrendszerbe, és mutatja a számítás menetét is!

A feladatban tömböt használunk a kettes számrendszerbeli szám jegyeinek tárolására. A tömb egy összetett adatszerkezet, amelynek több azonos tulajdonságokkal rendelkező rekesze van. Ezekre a rekeszekre egy névvel hivatkozunk, az egyes elemeket pedig a sorszámozott indexeiken keresztül érhetjük el.

A kiírás képernyőre pl.:

```
55 | :2
-----
27 | 1
13 | 1
 6 | 1
 3 | 0
 1 | 1
 0 | 1
```

A kettes számrendszerbeli szám: 110111

Pszeudokód:

```
Program:
  Ki: Feladat leírása.
  Be: szám [egész]
  Fejléckészítés
  Ciklus amíg szám > 0
    hányados := szám DIV 2
    maradék := szám MOD 2
    maradék tárolása
  Ki: hányados, maradék [hányados | maradék formában]
```

```
    szám := hányados  
    ciklus vége  
    Ki: kettes alakú szám  
Program vége
```

Pascal kód:

```
Program kettesszamPr;  
Uses crt;  
Var  
  
    {A szám és a hányados 0..65535 közötti szám lehet.}  
    szam, hanyados: Word;  
  
    {A maradéknak elég 1 bájt 0..255 is.}  
    maradek: Byte;  
  
    {A kettesszam változó egy 30 kis egész tárolására  
    alkalmas tömb.}  
    kettesszam: array[1..30] of Byte;  
    i: Byte;  
Begin  
    clrscr;  
    writeln('Kérek egy pozitív egész számot, és  
    átalakítom kettes számrendszerbe.');
```

```
    writeln;  
    write('Szám = ');  
    readln(szam);  
    writeln;  
  
    {Fejléc elkészítése.}  
    writeln(szam:5, ' | :2');  
    writeln(' -----');
```

```
    {Az i változó fogja az előállt kettes szám, tömbben  
    eltárolt utolsó (legnagyobb helyi értékű)  
    számjegypozícióját megadni. Kezdeti értéke nulla.}  
    If szam = 0 Then  
        Begin  
            i:= 1;  
            kettesszam[i]:= 0;  
        End  
    Else  
        Begin  
            i:=0;  
  
            {A ciklust addig folytatjuk, amíg az osztandó  
            nulla nem lesz.}  
            While szam > 0 Do
```

```

        Begin
            hanyados:= szam DIV 2;
            maradek:= szam MOD 2;
            inc(i);
            kettesszam[i]:= maradek;
            writeln(hanyados:5, ' | ', maradek:2);
            szam:= hanyados;
        End;
    End;
    writeln;
    write('A kettes számrendszerbeli szám: ');

    {A kettes szám kiírását a tárolással ellentétes
    irányban végezzük.}
    While i > 0 Do
        Begin
            write(kettesszam[i]);
            dec(i);
        End;
    readln;
End.

```

A DIV egész osztás, a MOD maradékképzés, az inc és a dec inkrementáló és dekrementáló eljárások.

Feladat: Az előző programot egészítsük ki azzal, hogy csak olyan számot fogadunk el, ami 0 és 10000 közé esik! A számokat addig kérjük be, amíg nem megfelelő!

Ez a részfeladat tipikusan végfeltételes ciklust igényel, ugyanis legalább egy számot be kell kérni.

Pszeudokód:

```

Program:
...
Ciklus
    Be: szám [egész]
    Mígnem (szám >= 0) ÉS (szám <= 10000)
    Ciklus vége
...
Program vége

```

Pascal kód:

```

Program kettesszamPr;

```

```

Uses crt;
Var
    szam, hanyados: Word;
    maradek: Byte;
    kettesszam: array[1..30] of Byte;
    i: Byte;
Begin
    clrscr;
    writeln('Kérek egy pozitív egész számot 0 és 10000
között,');
    writeln(' és átalakítom kettes számrendszerbe.');
```

{A végfeltételes ciklus addig kéri be a számot, amíg helyes értéket nem adunk meg.}

```

    Repeat
        write('Szám = ');
        readln(szam);
    Until (szam >= 0) AND (szam <= 10000);
    writeln;
    writeln(szam:5, ' | :2');
    writeln(' -----');
```

If szam = 0 Then

```

    Begin
        i:=1;
        kettesszam[i]:= 0;
    End
Else
    Begin
        i:=0;
        While szam > 0 Do
            Begin
                hanyados:= szam DIV 2;
                maradek:= szam MOD 2;
                inc(i);
                kettesszam[i]:= maradek;
                writeln(hanyados:5, ' | ', maradek:2);
                szam:= hanyados;
            End;
        End;
        writeln;
        write('A kettes számrendszerbeli szám: ');
        While i > 0 Do
            Begin
                write(kettesszam[i]);
                dec(i);
            End;
        readln;
    End.
```

Feladat: Írassuk ki a felhasználó választása szerinti szorzótáblát!

Mivel előre tudjuk, hogy a ciklusnak hányszor kell lefutnia érdemes számláló ciklust használni.

Pszeudokód:

```
Program:
  Ki: Feladat leírása.
  Be: Szám [1..10 egész]
  Ciklusváltozó := 1-től 10-ig
    Szorzat kiírása
  Ciklus vége
Program vége
```

Pascal kód:

```
Program szorzotablaPr;
Uses crt;
Var
  szam, i: word;
Begin
  clrscr;
  writeln('Szorzótábla készítés');
  writeln;
  write('Adj meg egy számot 1 és 10 között: ');
  readln(szam);
  writeln;
  writeln('A ', szam, '-es szorzótábla');
  writeln;

  {A ciklus a ciklusváltozó kezdő és végértéke közötti
  lépéseken keresztül fog végrehajtódni.}
  For i:= 1 To 10 Do
    Begin
      writeln(i:2, ' * ', szam:2, ' = ', i*szam:3);
    End;
  readln;
End.
```

Alprogramok bevezetése (eljárás, függvény), fájlkezelés

Az alprogramok az eljárásorientált nyelvekben a procedurális absztrakció első megjelenési formája. Egy olyan utasításcsoport, amelyre egy névvel hivatkozunk. Ez az utasításcsoport, egy bemenő adatcsoportot képez le egy kimenő adatcsoportra, úgy, hogy az alprogram specifikációján kívül a konkrét megvalósítást (implementációt) nem ismerjük. Azért nevezzük absztrakt eszköznek, mert az alprogram implementálása során egy formális paraméterlistát használunk az adatcsoport helyettesítésére. Majd a programban ahol szükség van erre az utasításcsoportra, konkretizáljuk a paraméterlistát adatokkal (aktuális paraméterlista). Ez a programkomponens sokszor felhasználható, akár más és más aktuális paraméterlistával is.

A formális paramétereket a törzsben, mint helyi (lokális) változókat használhatjuk. Elérhetőségük (hatókörük) az alprogramra korlátozódik.

Eljárás

Az *eljárás* olyan alprogram, amely valamilyen tevékenységet hajt végre. A hívás helyén e tevékenység eredményeit használhatjuk fel. Az eljárás a hatását a paramétereinek vagy a környezetének megváltoztatásával, illetve a törzsben elhelyezett végrehajtható utasítások által meghatározott tevékenység elvégzésével fejt ki.¹

Az eljárást meghívni utasításszerűen lehet.

Feladat: Készítsünk programot, amely egy decimális számot átalakít tizenhatos számrendszerbe! Az átalakítandó szám 0 és 60000 közé eshet, erről tájékoztassa a felhasználót.

A kiírás képernyőterve pl.:

```
550 | :16
-----
34 | 6
2 | 2
0 | 2
```

A tizenhatos számrendszerbeli szám: 226

¹ Dr. Juhász István előadásai alapján

Pszeudokód:

```
Program:
  Értesítő kiírása
  Szám bekérése [eljárás]
  Fejléc kiírása [eljárás]
  Átváltás elvégzése [eljárás]
  Szám kiírása [eljárás]
Program vége
```

A Pszeudokóddal az algoritmust tervezzük, ezért a bontást olyan mélységig kell elvégezni, ahonnan már egy választott programozási nyelven le tudjuk programozni. Itt az egyes tevékenységeknek alprogramok (eljárások) fognak megfelelni. Amennyiben szükség van rá, az eljárásokat is kifejthetjük.

Pascal kód:

```
Program tizenhatosPr;
Uses crt;

{Tömb típus létrehozása.}
Type
  Ttarolo= array[1..20] of byte;
Var
  szam: word;
  i: byte;
  tarolo: Ttarolo;

{Az eljárás addig kéri be a számot, amíg helyes értéket
nem adnak meg. Az sz változó cím szerinti paraméter.}
Procedure szambeker(Var sz: word);
Begin
  Repeat
    Write('Kérem a számot: ');
    readln(sz);
  Until (sz >= 0) AND (sz <= 30000);
End;

{Paraméter nélküli eljárás elkészíti a fejlécet.}
Procedure fejléc;
Begin
  writeln;
  writeln(szam:5, ' | :16');
  writeln(' -----');
End;
```

{Ez egy háromparaméteres eljárás, amely átalakítja az sz paraméterben átvett számot és elhelyezi a t tömbbe. Az i változó jelzi, hogy a t tömb hányadik indexű eleme tartalmazza az utolsó értékes számot.}

Procedure atalakit(Var sz: word; Var i: byte; Var t: Ttarolo);

```
    Var
        hanyados: word;
        maradek: byte;
    Begin
        If sz = 0 Then
            Begin
                i:=1;
                t[i]:= 0;
            End
        Else
            Begin
                i:=0;
                While sz > 0 Do
                    Begin
                        hanyados:= sz DIV 16;
                        maradek:= sz MOD 16;
                        inc(i);
                        t[i]:= maradek;
                        writeln(hanyados:5, ' | ', maradek:2);
                        szam:= hanyados;
                    End;
                End;
            End;
        End;
```

{Kiíratja a számot tizenhatos számrendszerben. Az esetszétválasztás (CASE) csréli ki a számot a megfelelő betűre.}

Procedure szamkiir(Var t: Ttarolo; Var i: byte);

```
    Begin
        writeln;
        Write('A tizenhatos számrendszerbeli szám: ');
        While i > 0 Do
            Begin
                Case t[i] of
                    10: Write('A');
                    11: Write('B');
                    12: Write('C');
                    13: Write('D');
                    14: Write('E');
                    15: Write('F');
                    Else Write(t[i]);
                End;
                dec(i);
            End;
        End;
```

```
End;

{A program törzse sokkal átláthatóbb.}
Begin
  clrscr;
  writeln('A program egy 0 és 60000 közötti decimális
számot ');
  writeln('átalakít tizenhatos számrendszerbe!');
  writeln;
  szambeker(szam);
  fejlec;
  atalakit(szam, i, tarolo);
  szamkiir(tarolo, i);
  readln;
End.
```

Az eljárás paramétereinél biztosítani kell az *aktuális* (konkrét érték) és a *formális* (deklaráció) *paraméterek* közötti *típus kompatibilitást*. Ezt egyszerűen megtehetjük, ha az összetett típusok esetén típusnevet hozunk létre. A *Ttarolo* nevű tömb deklarációja során is így történt.

A deklaráció/definíció során a *Var* kulcsszó használata során cím szerinti paraméterátadás történik. Ilyenkor az aktuális paraméternek csak a címkomponense adódik át a formális paraméter részére. Ebben az esetben az aktuális paraméter csak változó lehet. Az eljárás az aktuális paraméter (változó) memóriaterületén dolgozik, módosíthatja annak tartalmát.

Függvény

A *függvény* olyan alprogram, amelynek az a feladata, hogy egyetlen értéket határozzon meg. Ez az érték általában tetszőleges típusú lehet. A specifikáció része a visszatérési típus is. A függvény visszatérési értékét mindig a neve hordozza. A függvény törzsének végrehajtható utasításai a visszatérési érték meghatározását szolgálják. Azt a szituációt, amikor a függvény megváltoztatja paramétereit vagy környezetét, a függvény mellékhatásának nevezzük. Ezt általában károsnak tartjuk.¹

A függvény hívása olyan kifejezésben történhet, amelynek operandusa kompatibilis a függvénnyel.

¹ Dr. Juhász István előadásai alapján

Feladat: Készítsünk olyan programot, amely kiszámolja két szám legnagyobb közös osztóját!

Két szám legnagyobb közös osztóját úgy is meghatározhatjuk, hogy a nagyobb számot elosztjuk a kisebbel, s ha az osztási maradék 0, akkor a kisebb szám egyúttal az *lnko* is. Amennyiben a maradék nullától különböző, akkor elosztjuk vele a volt osztót, az új maradékkal ismét a volt osztót. Ezt mindaddig folytatjuk, míg a maradék 0 nem lesz. Az *lnko* az utolsó osztó. (*Euklideszi algoritmus.*)

Pszeudokód:

```
Program:
  Adatok bekérése
  Legnagyobb közös osztó kiszámítása [függvény]
  Adat kiírása
Program vége
```

Pascal kód:

```
Program lnkoPr;
Uses crt;
Var
  a,b, hanyados, maradek: word;

Function lnko(x,y:word):word;
Begin
  If x < y Then
    Begin
      {Megcseréljük a két változó tartalmát.}
      x:= x XOR y;
      y:= x XOR y;
      x:= x XOR y;
    End;

    {Most x nagyobb vagy egyenlő y.}
    maradek:= x MOD y;
    While maradek <> 0 Do
      Begin
        x:= y;
        y:= maradek;
        maradek:= x MOD y;
      End;
    If maradek = 0 Then
      lnko:=y;
```

```
End;

Begin
  clrscr;
  writeln('A program kiszámolja két szám legnagyobb
közös osztóját. ');
  writeln;
  Write('Kérem az első számot: ');
  readln(a);
  Write('Kérem a második számot: ');
  readln(b);
  writeln('-----');

  {Az lnko függvényt a kifejezésben hívjuk meg.}
  writeln('A legnagyobb közös osztó: ',lnko(a,b));
  readln;
End.
```

Az *lnko* függvény két érték szerinti paramétert vár. Az érték szerinti paraméterátadás során a formális paraméter az aktuális paraméter értékkomponensét kapja meg.

Két változó tartalmát a kizáró vagy (XOR) művelet segítségével is megcserélhetünk.

Fájlkezelés

Fizikai állománynak nevezzük a másodlagos tárolón elhelyezett adatok önálló névvel (állományspecifikáció) ellátott halmazát. A számítógépeken az operációs rendszer feladatai közé tartozik az állománykezelés (eszközfájlok, adatállományok). Az állományok elérését az operációs rendszeren keresztül a programozási nyelvek is támogatják.

Az állományműveletekhez a következő fő lépéseket kell elvégezni:

- a fájlváltozó hozzárendelése a fizikai állományhoz – assign(fájlváltozó, fájlnev),
- a fájl megnyitása – reset(fájlváltozó), rewrite(fájlváltozó), append(fájlváltozó),
- fájlműveletek – read, readln, write, writeln, blockread, blockwrite, seek, stb.,
- a fájl lezárása – close(fájlváltozó)

Az utolsó lépés aktualizálja a fájl tartalmát, üríti az írási puffer tartalmát. Az állomány lezárása után megmarad a kapcsolat a fájlváltozó és a fájlnev között, ezért újbóli megnyitás esetén nem kell az összerendelést ismét elvégezni.

Feladat: Titkosítsunk egy szövegfájlt (ASCII kódrendszer)!

A feladat szövegfájl kezelésre mutat példát.

A titkosítást kizáró vagy (XOR) művelettel végezzük. Ez egy szimmetrikus titkosító eljárás, amelynél ugyanazt a kulcsot használjuk kódolásra, mint dekódolásra. A szövegfájlunk ASCII kódrendszert használ karakterkódolásra. Ezt felhasználva a szöveget karakterenként, annak ASCII kódja alapján kódoljuk. A kódolt szöveget kimentjük fájlba, majd onnan visszaolvasva dekódoljuk.

A fájlok:

nyers.txt – a titkosítandó fájl,
titkos.txt – a titkosított fájl,
dekod.txt – a dekódolt fájl.

A használt Pascal függvények:

ord(karakter) – a karakter ASCII kódját adja,
chr(egész) – karaktert ad vissza.

Pszudokód:

```
Program:
  Fájllösszerendelések elvégzése.
  A titkosítandó fájl kiírása.
  Titkosítás elvégzése.
  Titkosított fájl kiírása.
  Dekódolás elvégzése.
  Dekódolt fájl kiírása.
Program vége.
```

Pascal kód:

```
program szov_tit;
uses crt;
type
  tpuffer = array[1..4096] of char;
  tstr = string[30];
  ttomb= array[1..10] of tstr;

{Képernyőre írja a paraméterként megadott fájlt.}
```

```

procedure kiir_kepernyo(var f: text);
var
    lv: tstr;
begin
{Hibakezelés.}
    {$i-}
    reset(f);
    if ioresult <> 0 then
        begin
            writeln('Hiba! - read');
            readln;
            exit;
        end;
    {$i+}
    while not eof(f) do
        begin
            readln(f, lv);
            write(lv, ', ');
        end;
    writeln;
    close(f);
end;

{Titkosítja az f-ből beolvasott szöveget és kiírja az
ft-vel jelölt fájlba.}

procedure titkosit(var f, ft:text);
var
    kulcs, i, n: byte;
    lv, tv: tstr;
begin
    {$i-}
    reset(f);
    if ioresult <> 0 then
        begin
            writeln('Hiba! - read');
            readln;
            exit;
        end;
    rewrite(ft);
    if ioresult <> 0 then
        begin
            writeln('Hiba! - write');
            readln;
            exit;
        end;
    {$i+}
    if not eof(f) then
        begin

```

```

        write('adja meg a kulcsot /0..255/: ');
readln(kulcs);
        while not eof(f) do
            begin
                readln(f, lv);
                n:= ord(lv[0]);
                tv[0]:= chr(n);
                if n>0 then
                    begin
                        for i:=1 to n do
                            begin
                                tv[i]:= chr(ord(lv[i]) xor kulcs);
                            end;
                        writeln(ft, tv);
                    end;
            end;
        end
    else
        writeln('Üres a fájl!');
        close(f);
        close(ft);
    end;
end;

{Dekódolja ft-t, dt-vel jelölt fájlba.}

procedure dekodol(var ft, dt:text);
var
    kulcs, i, n: byte;
    lv, tv: tstr;
begin
    {$i-}
    reset(ft);
    if ioresult <> 0 then
        begin
            writeln('Hiba! - read');
            readln;
            exit;
        end;
    rewrite(dt);
    if ioresult <> 0 then
        begin
            writeln('Hiba! - write');
            readln;
            exit;
        end;
    {$i+}
    if not eof(ft) then
        begin
            write('adja meg a kulcsot /0..255/: ');
readln(kulcs);

```

```

        while not eof(ft) do
            begin
                readln(ft, lv);
                n:= ord(lv[0]);
                tv[0]:= chr(n);
                if n>0 then
                    begin
                        for i:=1 to n do
                            begin
                                tv[i]:= chr(ord(lv[i]) xor kulcs);
                            end;
                        writeln(dt, tv);
                    end;
            end;
        else
            writeln('Üres a fájl!');
            close(ft);
            close(dt);
        end;

var
    f,ft,fd: text;
    puffer, puffert, pufferd: tpuffer;
begin
    clrscr;

    {Összerendelések elvégzése.}
    assign(f, '..\tp_gyak\nyers.txt');
    assign(ft, '..\tp_gyak\titkos.txt');
    assign(fd, '..\tp_gyak\dekod.txt');
    settextbuf(f, puffer);
    settextbuf(ft, puffert);
    settextbuf(fd, pufferd);
    kiir_kepernyo(f);
    titkosit(f, ft);
    kiir_kepernyo(ft);
    dekodol(ft, fd);
    kiir_kepernyo(fd);
    readln;
end.

```

A Free Pascal lehetővé teszi, hogy a fájl nyitáskor (hibás fájlnev) fellépő hibákat lekezeljük. Ha a kritikus programrészeket a {\$I-} és a {\$I+} *fordítási direktívák* közé tesszük, akkor az *ioresoutl* függvény által visszaadott értékből következtethetünk a hiba okára. A nulla visszatérési érték azt jelzi, hogy nem történt hiba, minden más érték hibát jelez.

A nyers.txt fájl ASCII kódú szöveget tartalmaz, már léteznie kell. A titkos.txt és a dekód.txt fájlt létrehozza. A titkosító kulcsot a felhasználótól kérjük be, értéke 0 és 255 között legyen.

Feladat: Készítsünk olyan programot, amely eltárol neveket és kódokat tartalmazó adatokat.

Ez a feladat a típusos fájl használatát mutatja be.

A típusos fájlban azonos felépítésű rekordok tárolódnak. Itt nevek és hozzá tartozó kódok jelentik a rekordot.

A program bekéri a fájl nevét, majd létrehozza. A létrehozás után bekéri a neveket a hozzá tartozó kódokkal együtt. A nevek 30, a kódok 3 karakter hosszúak lehetnek. Miután feltöltöttük a fájlt, kereshetünk benne név, vagy kód alapján.

Pszeudokód:

```
Program:
  Fájlleltrehozás.
  Fájl feltöltés.
  Tartalom kiírás.
  Keresés név vagy kód alapján.
Program vége.
```

Pascal kód:

```
program tipusosfile;
uses crt;
type
  str30= string[30];
  str3= string[3];
  telem= record
    nev: str30;
    kod: str3;
  end;
  tfiletip= file of telem;
var
  f: tfiletip;

{Létrehozza a típusos fájlt.}
procedure letrehoz;
var
  fnev: str30;
```

```

begin
    write('Létrehozandó file neve: '); readln(fnev);
    assign(f, fnev);
    {$i-} rewrite(f); {$i+}
    if ioresult <> 0 then
        begin
            writeln('Hiba!'); exit;
        end
    else
        writeln(fnev, ' létrehozva');
    end;

{Megnyitja a fájlt.}
procedure megnyit;
var
    fnev: str30;
begin
    write('Megnyitás, file név: '); readln(fnev);
    assign(f, fnev);
    {$i-} reset(f); {$i+}
    if ioresult <> 0 then
        begin
            writeln('Nincs ilyen file!'); exit;
        end
    else
        writeln(fnev, ' megnyitva.');
```

```

end;

{Feltölti a neveket és kódokat tartalmazó fájlt.}
procedure feltolt( var ft: tfiletip);
var
    lv: telem;
begin
    writeln('Adatok felvitele név= üres karakterig');
    repeat
        write('Név: '); readln(lv.nev);
        if lv.nev <> '' then
            begin
                write('Kód: '); readln(lv.kod);
                write(ft, lv);
            end;
    until lv.nev = '';
end;

{Kiírja a fájl tartalmát.}
procedure kiir(var ft: tfiletip);
var
    lv:telem;
    i: byte;
begin
```

```

i:=0;
while not eof(ft) do
begin
    read(ft, lv);
    inc(i);
    writeln(i, '. rek.: ',lv.nev,', ', ',lv.kod);
end;
end;

{Név vagy kód alapján keresést végez.}
procedure keres;
var
    n: str30;
    k: str3;
    e:telem;
    m: char;
begin
    writeln('n_név-re vagy k_kód-ra keres');
    repeat
        readln(m);
        m:= upcase(m);
    until m in ['N','K'];
    if m = 'N' then
        begin
            write('Keresendő név: '); readln(n);
            while not eof(f) do
                begin
                    read(f,e);
                    if e.nev = n then
                        begin
                            writeln('Megvan! A kódja: ', e.kod);
readln; exit;
                        end;
                end;
            writeln('Nincs benne!'); readln;
        end
    else
        begin
            write('Keresendő kód: '); readln(k);
            while not eof(f) do
                begin
                    read(f, e);
                    if e.kod = k then
                        begin
                            writeln('Megvan! A neve: ', e.nev);
readln; exit;
                        end;
                end;
            end;
        end;
    close(f);

```

```
end;

{A főprogram kezdete.}
begin
    clrscr;
    letrehoz;
    feltolt(f);
    close(f);
    megnyit;
    kiir(f);
    seek(f,0);
    writeln('Keresés!');
    keres;
    close(f);
end.
```

Alapvető algoritmusok

Vektor, mátrix feltöltése

Feladat: Egy adott (N) elemszámú vektor (tömb) feltöltése véletlen értékekkel!

Létrehozunk egy 0 és 255 közötti számok tárolására alkalmas vektort. A vektor N elemszámú. Majd feltöltjük véletlen számokkal.

Pascal kód:

```
Program vektorPr;
Uses crt;
Const

{Konstansban tároljuk a maximális elemszámot.}
    N = 50;
Var

{Létrehozunk egy tömböt.}
    tomb: Array[1..N] of byte;
    i: byte;
Begin
    clrscr;
    writeln('Tömb feltöltése véletlen számokkal');

{Véletlenszám generátor inicializálása.}
    randomize;

{A tömb elemein egy ciklus segítségével haladunk végig.}
    For i:=1 To N Do
        tomb[i]:= random(N+1);
        writeln;

{Kiíratjuk a tömb elemeit, sorfolytonosan.}
    For i:=1 To N Do
        write(tomb[i], ', ');
        readln;
End.
```

Feladat: Írjunk programot, amely feltölt 0 és 500 közötti véletlen páros számokkal egy 20*10-as mátrixot, majd kiírja a képernyőre! A feltöltést eljárással valósítsuk meg!

A mátrix egy kétdimenziós tömb, amelynek két kiterjedése van (sor*oszlop). A mátrix elemeire a *matrix[i,j]*, vagy *matrix[i][j]* formák valamelyikével hivatkozunk, ahol az *i*

és a j a mátrix sorát és oszlopát jelöli ki. Az elemeket elérhetjük sor- és oszlop-folytonosan is. Legegyszerűbben két ciklus egymásba ágyazásával. Sorfolytonos elérés esetén az első (külső) ciklus a sorokon fog lépkedni, a második (belső) pedig a soron belül az elemeket (oszlopokat) éri el.

Pascal kód:

```
Program matrixPr;
Uses crt;
Type
  {Létrehozunk egy 20*10-as mátrix típust.}
  Tmatrix = Array[1..20, 1..10] of Integer;

{Mátrix feltöltése véletlen számmal, az indexek 1-ről
indulnak.}
Procedure feltolt_matrix(Var m:Tmatrix; k, l: byte);
  Var
    i, j: byte;
  Begin

    {Véletlenszám generálása.}
    randomize;

    {Sorok elérése.}
    For i:=1 To k Do

      {Oszlopok (elemek) elérése.}
      For j:= 1 To l Do

        {Csak páros számokat fogadunk el.}
        Repeat
          m[i,j]:= random(501);
        Until (m[i,j] MOD 2) = 0;

  End;

{Mátrix kiírása a képernyőre.}
Procedure kiir_matrix(Var m:Tmatrix; k, l: byte);
  Var
    i, j: byte;
  Begin

    {Sorok elérése.}
    For i:= 1 To k Do
      Begin

        {Oszlopok (elemek) elérése.}
        For j:= 1 To l Do
```

```

        {Öt mezőszélességen jobbra igazítjuk a
számokat.}
        write(m[i,j]:5);

        {Egy sor kiírása után új sort kezdünk.}
        writeln;
    End;
End;

Var
    matrix: Tmatrix;
Begin
    clrscr;
    Writeln('20*10-es matrix feltöltése véletlen páros
számokkal. ');
    writeln;
    feltolt_matrix(matrix, 20, 10);
    kiir_matrix(matrix, 20, 10);
    readln;
End.

```

Megszámlálás, eldöntés

A *megszámlálás* olyan művelet, amely egy sorozatban megszámlolja, hogy egy adott tulajdonságú elem hányszor szerepel. Ha a számlálást előről kezdjük, ne felejtsük nullázni a tárolásra szánt változót, majd minden meg talált érték után egyel növelni.

Az *eldöntés* tétele megmondja, hogy egy adott tulajdonságú elem szerepel-e a vizsgált sorozatban.

Egy változó értékének növelésére használható az *inc(v, n)* eljárás, amely a *v* változót, *n* értékkel növeli. Amennyiben *n* hiányzik, a lépésköz 1.

Feladat: Az előző feladatot egészítsük ki azzal, hogy megszámloljuk egy adott szám előfordulását! A számot a felhasználótól kérjük be, végezzünk értékhatár ellenőrzést (0..500)! Készítsük el a szükséges eljárást, és a főprogram kiegészítését!

Pascal kód (részlet):

```

...
{Megszámlolja egy adott tulajdonságú elem
előfordulását.}
Procedure szamol_matrix(Var m: Tmatrix; k, l: byte);
Var
    i, j: byte;

```

```

        szam, db: Integer;
Begin
    writeln;
    writeln('Adj meg egy számot 0 és 500 között, ');
    writeln('megmondom      hányszor      szerepel      a
mátrixban.');
```

{Addig kérjük be a számot, amíg 0 és 500 közötti
nem lesz.}

```

    Repeat
        write('Szám: ');
        readln(szam);
    Until (szam >= 0) AND (szam <= 500);

    {Tároló nullázása.}
    db:= 0;

    {Előfordulás megszámlálása.}
    For i:=1 To k Do
        For j:=1 To l Do
            If m[i,j] = szam Then
                inc(db);

    {Darabszám kiírása.}
    writeln('A keresett szám előfordulása: ', db);
End;
```

...

```

Begin
    clrscr;
    Writeln('20*10-es matrix feltöltése véletlen páros
számokkal.');
```

{Mátrix feltöltése}

```

    feltolt_matrix(matrix, 20, 10);

    {Mátrix kiírása}
    kiir_matrix(matrix, 20, 10);

    {Megszámlálás}
    szamol_matrix(matrix, 20, 10);
    readln;
End.
```

Összeg-, átlagképzés

Összeg meghatározása során egy számsorozat elemeinek összegét, átlag meghatározása során pedig átlagát állítjuk elő. Az összeget és az átlagot tartalmazó változókat nullázni kell.

Feladat: Az előző feladatokat folytatva határozzuk meg a véletlen páros számokat tartalmazó mátrix elemeinek összegét, és átlagát! A megoldásra eljárást készítsünk!

Pascal kód:

```
...
{Összeg és átlagképzés.}
Procedure  összeg_atlag_matrix(Var  m:Tmatrix;  k,  l:
Byte);
  Var
    i,j: Byte;
    összeg: Longint;
    atlag: Real;
  Begin

    {Változók nullázása.}
    összeg:=0;
    atlag:=0;

    {Összeg meghatározása.}
    For i:=1 To k Do
      For j:=1 To l Do
        inc(összeg, m[i,j]);

    {Kiírás}
    writeln;
    writeln('Az elemek összege: ', összeg);

    {Átlag meghatározása.}
    atlag:= összeg /(k*l);  {k*l elemszám}
    writeln;
    writeln('Az elemek átlaga: ', atlag:0:2);    {2
tizedesre igazítjuk}
  End;
...
Begin
  clrscr;
  Writeln('20*10-es matrix feltöltése véletlen páros
számokkal.');
```

{Mátrix feltöltése}

```

feltolt_matrix(matrix, 20, 10);

{Mátrix kiírása}
kiir_matrix(matrix, 20, 10);

{Megszámlálás}
szamol_matrix(matrix, 20, 10);

{Összeg és átlag számítás}
osszeg_atlag_matrix(matrix, 20, 10);
readln;
End.

```

Kiválasztás

A kiválasztás során adott tulajdonságú elemet, vagy elemeket keresünk egy sorozatban. Kíváncsiak vagyunk a sorozatban elfoglalt helyére (sorszámára, indexére). Kereshetjük a legelső, a legutolsó előfordulását, vagy az összeset is.

Feladat: A mátrix feladatot folytatva készítsünk olyan eljárást, amely kiírja a tizenhatallosztható számok sor*oszlop indexeit a képernyőre!

Pascal kód:

```

...
{16-tal osztható számok indexei}
Procedure oszthatol6_matrix(Var m: Tmatrix; k, l:
Byte);
Var
    i, j: Byte;
Begin
    writeln;
    writeln('Tizenhatallosztható számok helyei a
táblázatban:');
    writeln;
    For i:=1 To k Do
        For j:=1 To l Do

            {Tizenhatallosztható számok meghatározása.}
            If m[i,j] MOD 16 = 0 Then
                writeln(m[i,j]:3, ' sor: ', i:2, ' - oszlop:
', j:2);
        End;
    End;
...
Begin
    clrscr;

```

```

Writeln('20*10-es matrix feltöltése véletlen páros
számokkal.');
```

```

writeln;

{Mátrix feltöltése}
feltolt_matrix(matrix, 20, 10);

{Mátrix kiírása}
kiir_matrix(matrix, 20, 10);

{Megszámlálás}
szamol_matrix(matrix, 20, 10);

{Összeg és átlag számítás}
osszeg_atlag_matrix(matrix, 20, 10);

{Tizenhattal osztható számok indexei}
oszthatol6_matrix(matrix, 20, 10);
readln;
End.
```

Kiválogatás

A kiválogatás tétele egy sorozat adott tulajdonságokkal rendelkező elemeit átmásolja egy másik sorozatba. A másolás során fel kell készülni a szélsőségekre is. Azaz lehet, hogy minden elem megfelel a feltételnek.

Abban az esetben, ha a feltétel bonyolult, érdemes logikai értéket visszaadó függvény formájában megvalósítani.

Feladat: Az előző feladatokat folytatva készítsünk olyan eljárást, amely átmásolja egy másik mátrixba a hárommal osztható és kettőre végződő számokat!

Feltétel: $(\text{szám} \bmod 3 = 0) \text{ AND } (\text{szám} \bmod 10 = 2)$.

A MOD operátor a szám osztási maradékát adja eredményül.

Pascal kód:

```

...
{A hárommal osztható és kettőre végződő számok
kiválogatása.}
Procedure három_ketto_matrix(Var m1, m2: Tmatrix; k1,
l1: Byte);
    Function fgv(szam: Integer): Boolean;
        Begin
```

```

        If (szam MOD 3 = 0) AND (szam MOD 10 = 2) Then
            fgv:= TRUE
        Else
            fgv:= FALSE;
        End;
    Var
        i, j, k2, l2: Byte;
    Begin
        writeln;
        writeln('Meghatározzuk a hárommal osztható kettőre
végződő számokat.');
```

 writeln;

 {Tároló mátrix indexeinek beállítása.}

```

        k2:=1;
        l2:=0;
        For i:= 1 To k1 Do
            For j:= 1 To l1 Do

                {Adott tulajdonságú elem meghatározása.}
                If fgv(m1[i,j]) Then
                    Begin

                        {Sorszámot csak akkor növelünk, ha az
oszlopszám elérte a 10-et.}
                        If l2 = 10 Then
                            Begin
                                inc(k2);
                                l2:=1;
                            End
                        Else

                            {egyébként csak az oszlopszámot növeljük.}
                            inc(l2);
                            m2[k2,l2]:= m1[i,j];
                        End;

                    {Elemek kiírása}
                    {Figyelembe vesszük, hogy egy vagy több sorunk van-
e.}
                    If k2 > 1 Then
                        Begin
                            For i:=1 To k2-1 Do
                                Begin
                                    For j:=1 To l1 Do
                                        write(m2[i,j]:5);
                                    writeln;
                                End;
                            For i:=1 To l2 Do
                                write(m2[k2,i]:5);

```

```

End

    {Ha csak egy sor van.}
    Else If k1 = 1 Then
        For i:= 1 To l2 Do
            write(m2[k1,l2]:5)
        Else

            {Ha nincs ilyen elem azt is jelezzük.}
            writeln('Nincs ilyen elem.');
```

End;

...

```

Begin
    clrscr;
    Writeln('20*10-es matrix feltöltése véletlen páros
számokkal.');
```

writeln;

```

    {Mátrix feltöltése}
    feltolt_matrix(matrix, 20, 10);

    {Mátrix kiírása}
    kiir_matrix(matrix, 20, 10);

    {Megszámlálás}
    szamol_matrix(matrix, 20, 10);

    {Összeg és átlag számítás}
    osszeg_atlag_matrix(matrix, 20, 10);

    {Tizenhattal osztható számok indexei}
    oszthatol6_matrix(matrix, 20, 10);

    {Hárommal osztható kettőre végződő számok.}
    harom_ketto_matrix(matrix, matrix2, 20, 10);
    readln;
End.
```

Lineáris keresés

A keresések során, egy adott tulajdonságú elemről meg szeretnénk tudni, hogy része-e egy sorozatnak, esetleg a sorozatban elfoglalt helyére is kíváncsiak vagyunk.

A lineáris keresés során a sorozatot az elejéről vizsgáljuk addig, amíg meg nem találtuk a kérdéses elemet (elemeket), vagy el nem értük a sorozat végét. Lassú keresési módszer, amelynél átlagosan a sorozat feléig kell keresnünk.

Rendezett sorozatban való keresésnél a keresett elem megtalálásáig, illetve ha nincs benne, az elemet a rendezettségben elhagyó első elemig keresünk. Például növekvő rendezettség esetén, a keresett elemünket nagyság szerint elhagyó elemhez érünk, akkor abbahagyjuk a keresést.

Rendezett sorozatban való keresésre, van egy sokkal hatékonyabb módszer: a bináris, vagy logaritmikus keresés. Egy N elemű sorozatban ($\log_2 N$) lépésben eldönti, hogy a kérdéses elem benne van-e a sorozatban.

Inkább rendezetlen sorozatban való keresésre használjuk. Amennyiben tudjuk, hogy egy sorozatban hány elem van, akkor a keresést csak az utolsó elem megtalálásáig folytatjuk, majd megállunk.

Ha nem tudjuk, hogy a keresett elem benne van-e a sorozatban, vagy hány előfordulása van, akkor mindenképpen a sorozat végéig kell keresni (teljes keresés).

A keresés során egyenként lépkedünk a sorozat elemein és figyeljük, hogy az aktuális elem az keresett-e és, hogy elértük-e a sorozat végét. A két feltétel vizsgálat növeli a végrehajtási időt. Ciklusonként egy feltételvizsgálatot megspórolhatunk, ha a sorozat végére elhelyezzük a keresett elemet (strázsaelem). Ilyenkor nem kell vizsgálni ciklusonként a sorozat végét, mert az utolsó elem úgyis leállítja a ciklust. A ciklusból kilépve meg kell nézni, hogy a megtalált elem a strázsaelem-e, mert ekkor a keresett elem nem része a sorozatnak, ellenkező esetben része.

Feladat: Készítsünk olyan programot, amely feltölt egy 20000 elemű vektort véletlen számokkal, majd a felhasználótól bekért számot megkeresi, és ha benne van, kiírja a sorszámát, ha nincs benne, arról tájékoztatást ad!

Pascal kód:

```
Program linearisPr;
Uses crt, dos;
Type
    Tvektor = Array[1..20001] Of Integer;

{Feltölt egy 20000 elemű vektort véletlen (1..1000)
értékekkel.}
Procedure feltolt_vektor(Var v: Tvektor);
    Var
        i: word;
```

```

Begin
  {Véletlenszám generátor inicializálása.}
  randomize;
  For i:= 1 To 20000 Do
    v[i]:= random(1001);
  End;

  {Kiírja egy 20000 elemű tömb értékeit.}
  Procedure kiir_vektor(Var v: Tvektor);
  Var
    i: word;
  Begin
    writeln('A vektor elemei:');
    writeln;
    For i:= 1 To 20000 Do
      Begin
        write(v[i], ', ');
        If (i MOD 10 = 0) Then
          writeln;
      End;
    End;
  End;

  {Lineáris keresés (első előfordulás) a felhasználó
  kérése alapján.}
  Procedure keres_linearis(Var v: Tvektor; ker: Integer);
  Var
    i: word;
  Begin
    i:= 1;

    {A keresendő elemet betesszük a végére strázsának.}
    v[20001]:= ker;

    {Ilyenkor nem kell a tömb végét vizsgálni.}
    While v[i] <> ker Do
      inc(i);

    {Megvizsgáljunk mikor léptünk ki.}
    If i <= 20000 Then
      writeln('A keresett elem benne van, helye: ', i)
    else
      writeln('A keresett elem nincs benne.');
```

End;

```

  Var
    vektor: Tvektor;
    keresett: Integer;
  Begin
    clrscr;
```

```
feltolt_vektor(vektor);  
{kiir_vektor(vektor);}  
writeln;  
write('Kérem a keresett számot: '); readln(keresett);  
keres_linearis(vektor, keresett);  
readln;  
End.
```

Maximum- minimumkiválasztás

Maximumkiválasztás lényege, hogy a sorozat elemeit sorban megvizsgáljuk, és mindig megjegyezzük az addigi legnagyobb elemet. Ha egy ennél nagyobb elem érkezik, akkor lecseréljük erre az eddigi legnagyobbnak tartott elemet. Induló értéknek, vagy a sorozat első elemét választjuk, vagy egy olyan elem, amelyet a sorozat minden eleme lehalad. A kiválasztás végén, megkapjuk a maximális elemet, ha volt legalább egy eleme a sorozatnak.

Minimumkiválasztás esetén, a sorozat legkisebb elemét keressük. Induló értéknek vagy a sorozat első elemét, vagy egy minden értéknél nagyobbat választunk. Haladva a sorozatban mindig megjegyezzük az addigi legkisebb értéket.

Maximumkiválasztás ismert elemszám esetén:

```
Program:  
  maxindex := 1  
  maxelem := sorozat[első eleme]  
  Ciklus első elemtől utolsóig  
    Ha sorozat[aktuális eleme] > maxelem  
      maxelem := sorozat[aktuális eleme]  
      maxindex := aktuális elem  
  Ha vége  
  Ciklus vége  
Program vége
```

Maximumkiválasztás ismeretlen elemszám esetén:

```
Program:  
  maxindex := 1  
  maxelem := sorozat[első eleme]  
  Ciklus második elemtől amíg van elem  
    Ha sorozat[aktuális eleme] > maxelem  
      maxelem := sorozat[aktuális eleme]  
      maxindex := aktuális elem
```

```
    aktuális elem := következő elem
  Ha vége
    Ciklus vége
  Program vége
```

Feladat: Készítsünk olyan eljárást, amely az előző feladatban előállított véletlen elemű vektorban megkeresi a legnagyobb és a legkisebb elemet!

Pascal kód:

```
...
{Minimális, maximális elemet kiválasztó eljárás.}
Procedure min_max(Var v: Tvektor);
  Var
    i: word;
    min, max: Integer;
  Begin

    {Legyen a minimális és maximális elem az első.}
    min:= v[1];
    max:= v[1];

    {Keresés a második elemtől.}
    For i:= 2 To 200 Do
      Begin

        {Ha az aktuális kisebb mint az eddigi
        legkisebb.}
        If v[i] < min Then
          min:= v[i];

        {Ha az aktuális nagyobb mint az eddigi
        legnagyobb.}
        If v[i] > max Then
          max:= v[i];

        End;
      writeln;
      writeln('A vektor legkisebb eleme: ', min);
      writeln('A vektor legnagyobb eleme: ', max);
    End;

  ...
Begin
  clrscr;
  feltolt_vektor(vektor);
  {kiir_vektor(vektor);}
  writeln;
  {Lineáris keresés.}
  write('Kérem a keresett számot: '); readln(keresett);
```

```

keres_linearis(vektor, keresett);

{Minimális, maximális elem kiválasztása.}
min_max(vektor);
readln;
End.

```

Minimum-kiválasztásos rendezés

A rendezés egy sorozat elemeinek egymáshoz viszonyított helyét állítja be. A minimum-kiválasztásos rendezésnél a rendező elv a nagyság szerinti sorrend.

A minimum-kiválasztásos rendezés lényege:

- Első lépésben megkeressük a sorozat legkisebb elemét. Majd, ha ez nem az első elem, kicseréljük az első elemmel. Így az első helyen a sorozat legkisebb eleme lesz.
- A következő lépésben, a második legkisebb elemet keressük, a még nem rendezett (második elemtől az utolsóig) részben. A megtalált elemet, ha ez nem egyenlő a második helyen lévővel, megcseréljük a második helyen lévővel. Ekkor a sorozatunk, a második elemig rendezett.
- A további lépések az előző kettőhöz hasonlóan futnak le, mindig megkeresve a sorozat következő legkisebb elemét. Amely ez által addig rendezetté válik.
- Utolsó lépésként az utolsó előtti helyre kell kiválasztani a legkisebb elemet. Ezt követően a sorozat rendezetté válik.

Pszeudokód:

```

Ciklus i := 1-től a sorozat vége -1-ig
  minindex := i
  Ciklus j := i + 1-től a sorozat végéig
    Ha sorozat[j] < sorozat[minindex] akkor
      minindex := j
    Elágazás vége
  Cilus vége
  Ha i <> minindex akkor
    Csere(sorozat[i], sorozat[minindex])
  Elágazás vége
Ciklus vége

```

Példa:

3, 6, 2, 7, 4, 9, 1, 8, 5	← rendezetlen sorozat
<u>1</u> , 6, 2, 7, 4, 9, 3, 8, 5	
<u>1</u> , <u>2</u> , 6, 7, 4, 9, 3, 8, 5	
<u>1</u> , <u>2</u> , <u>3</u> , 7, 4, 9, 6, 8, 5	
<u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , 7, 9, 6, 8, 5	
<u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , 9, 6, 8, 7	
<u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , <u>6</u> , 9, 8, 7	
<u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , <u>6</u> , <u>7</u> , 8, 9	
<u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , <u>6</u> , <u>7</u> , <u>8</u> , <u>9</u>	← rendezett sorozat

Ciklusok száma: 8

Feladat: Az előző feladatban előállított véletlen elemszámú vektort rendezzük minimum-kiválasztásos rendezéssel! Írjuk meg a rendező eljárást!

Pascal kód:

```
...
{Csere eljárás.}
Procedure csere(Var a, b: Integer);
Begin
    {Csere kizáró vagy segítségével.}
    a:= a XOR b;
    b:= a XOR b;
    a:= a XOR b;
End;

{Minimum-kiválasztásos rendezés}
Procedure minimum_rendez(Var v: Tvektor);
Var
    minindex, i, j: word;
Begin
    For i:= 1 To 200-1 Do
        Begin
            minindex:= i;
            For j:= i+1 To 200 Do
                If v[j] < v[minindex] Then
                    minindex:= j;
            If i <> minindex Then
                csere(v[i], v[minindex]);
        End;
    End;
...
Begin
    clrscr;
    feltolt_vektor(vektor);
```

```
kiir_vektor(vektor);
writeln;
write('Kérem a keresett számot: '); readln(keresett);
keres_linearis(vektor, keresett);

{Minimális, maximális elem kiválasztása.}
min_max(vektor);

{Minimum-kiválasztás rendezés}
minimum_rendez(vektor);
writeln;
writeln('Rendezés. ');
kiir_vektor(vektor);
readln;
End.
```

Fejlettebb algoritmusok

Rendezettség

A rendezettség azt jelenti, hogy egy sorozat *egymást követő két eleméről* (pl. A és B) egyértelműen meg lehet állapítani, hogy melyikük az előrébb való. A rendező elv lehet bonyolultabb is, mint a kisebb/nagyobb reláció. Amikor egy összetett feltétel szerint kell meghatározni az elemek egymáshoz viszonyított helyét, érdemes egy rendező függvényt készíteni. Ez a függvény, akkor fog igazat visszaadni, ha a két rendezendő elem egymáshoz képest jó helyen van.

Feladat: Töltsünk fel egy 20 elemű vektort véletlen értékekkel, és rendezzük úgy, hogy elől legyenek a páratlan számok növekvőleg, majd a párosak szintén növekvő sorrendben!

Páratlanság vizsgálata: $(A \bmod 2 \neq 0)$ – a szám kettővel való osztása nem nulla osztási maradékot ad.

Párosság vizsgálata: $(A \bmod 2 = 0)$ – a szám kettővel való osztása esetén nincs maradék.

Elől legyenek a páratlan számok, hátul a párosak: $(A \bmod 2 \neq 0) \text{ AND } (B \bmod 2 = 0)$.

Két páratlan szám esetén az első kisebb legyen, mint a második: $(A \bmod 2 \neq 0) \text{ AND } (B \bmod 2 \neq 0) \text{ AND } (A \leq B)$.

Két páros szám esetén az első kisebb legyen, mint a második: $(A \bmod 2 = 0) \text{ AND } (B \bmod 2 = 0) \text{ AND } (A \leq B)$.

Tehát a szükséges feltétel: **$((A \bmod 2 \neq 0) \text{ AND } (B \bmod 2 = 0)) \text{ OR } ((A \bmod 2 \neq 0) \text{ AND } (B \bmod 2 \neq 0) \text{ AND } (A \leq B)) \text{ OR } ((A \bmod 2 = 0) \text{ AND } (B \bmod 2 = 0) \text{ AND } (A \leq B))$.**

Az AND erősebb prioritású művelet, mint az OR, ezért a külső zárójelek elhagyhatók. A MOD szintén erősebb, mint a relációjel, ezért oda nem lett téve zárójel.

Pascal kód:

```

Program rendezettsegPr;
Uses crt;
Type
    Tvektor= Array[1..20] Of Integer;

{Vektor feltöltése 0..1000 közötti véletlen
értékekkel.}
Procedure feltolt_vektor(Var v: Tvektor);
    Var
        i: Byte;
    Begin
        randomize;
        For i:= 1 To 20 Do
            v[i]:= random(1001);
        End;

{Elemek kiírása.}
Procedure kiir_vektor(Var v: Tvektor);
    Var
        i: Byte;
    Begin
        For i:= 1 To 20 Do
            Begin
                write(v[i], ', ');

                {Tíz szám kiírása után sortörés.}
                If i MOD 10 = 0 Then
                    writeln;
            End;
        End;

{Rendezés.}
Procedure rendez(Var v: Tvektor);
    Function rendezett(a, b: Integer): Boolean; {Érték
szerinti paraméter.}
        Begin
            {A rendezettség adó feltétel.}
            rendezett:= (a MOD 2 <> 0) AND (b MOD 2 = 0) OR
                (a MOD 2 <> 0) AND (b MOD 2 <> 0) AND (a <= b)
OR
                (a MOD 2 = 0) AND (b MOD 2 = 0) AND (a <= b);
        End;

    {Két változó értékét felcserélő eljárás.}
    Procedure csere(Var a, b: Integer); {Cím szerinti
paraméter.}
        Var
            seged: Integer;
        Begin

```

```

        {Csere segédváltozóval.}
        seged:= a;
        a:= b;
        b:= seged;
    End;

    Var {rendez}
        i, j, minindex: Byte;
    Begin {rendez}
        For i:= 1 To 20-1 Do
            Begin
                minindex:= i;
                For j:= i+1 To 20 Do
                    If Not rendezett (v[minindex], v[j]) Then
                        minindex:= j;
                If minindex <> i Then
                    csere(v[i], v[minindex]);
            End;
        End; {rendez}

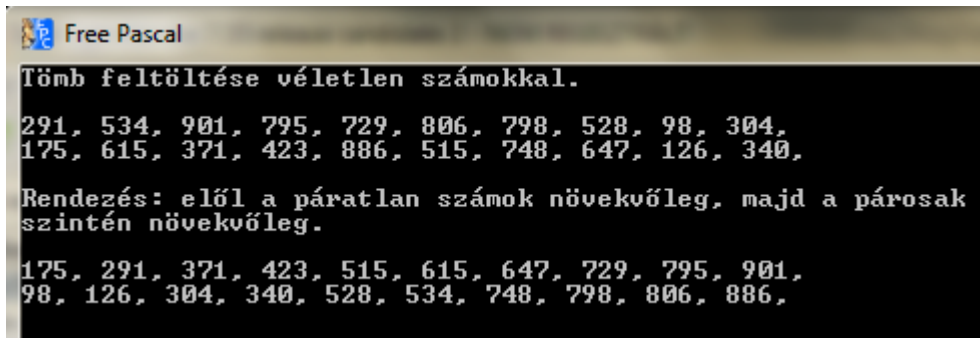
    Var {Főprogram}
        vektor: Tvektor;
    Begin {Főprogram}
        clrscr;
        writeln('Tömb feltöltése véletlen számokkal.');
```

{Vektor feltöltése véletlen számokkal.}
 feltolt_vektor(vektor);
 writeln;

{Rendezetlen vektor kiírása.}
 kiir_vektor(vektor);
 writeln;

{Rendezés.}
 rendez(vektor);
 writeln('Rendezés: elől a páratlan számok növekvőleg,
 majd a párosak ');
 writeln('szintén növekvőleg.');

{Rendezett vektor kiírása.}
 kiir_vektor(vektor);
 readln;
 End.



```
Free Pascal
Tömb feltöltése véletlen számokkal.
291, 534, 901, 795, 729, 806, 798, 528, 98, 304,
175, 615, 371, 423, 886, 515, 748, 647, 126, 340,

Rendezés: elől a páratlan számok növekvőleg, majd a párosak
szintén növekvőleg.
175, 291, 371, 423, 515, 615, 647, 729, 795, 901,
98, 126, 304, 340, 528, 534, 748, 798, 806, 886,
```

Ábra: Rendezettség

Buborék-rendezés

A buborékos rendezés a szomszédos elemeket cseréli, ha a sorrend nem megfelelő. Kezdetben az első két elemet hasonlítjuk össze, és szükség esetén felcseréljük őket. A második lépésben a 2. és a 3. elemet hasonlítjuk, és ha kell, cseréljük. Ezt így folytatjuk a sorozat végéig. Ekkor növekvő rendezettséget feltételezve a sorozat végén lesz a legnagyobb elem. A második ciklusban, már csak az utolsó előtti elemig végezzük a hasonlítás és szükség esetén a cserét. Ha a második ciklus lefutott, a sorozat utolsó két eleme már a helyén van. A harmadik, és az azt követő ciklusokban egyre kisebb része marad rendezetlenül a sorozatnak. Utolsó ciklus az első két elemet rendezi.

Pszeudokód:

```
Buborék(T: tömb[1..N] egész)
Deklaráció
    i, j: egész
Buborék_kezd
    Ciklus i:= N-1-től 1-ig -1-esével
        Ciklus j:= 1-től i-ig 1-esével
            Ha T[j] > T[j+1] akkor
                csere(T[j], T[j+1])
            Elágazás vége
        Ciklus vége
    Ciklus vége
Buborék vége
```

Példa:

```
A rendezetlen vektor:
3, 6, 2, 7, 4, 9, 1, 8, 5,

Buborék rendezés első változat
3, 6, 2, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 1, 9, 8, 5,
3, 2, 6, 4, 7, 1, 8, 9, 5,
3, 2, 6, 4, 7, 1, 8, 5, 9,
2, 3, 6, 4, 7, 1, 8, 5, 9,
2, 3, 4, 6, 7, 1, 8, 5, 9,
2, 3, 4, 6, 1, 7, 8, 5, 9,
2, 3, 4, 6, 1, 7, 8, 5, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 1, 6, 7, 5, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 1, 4, 6, 5, 7, 8, 9,
2, 3, 1, 4, 5, 6, 7, 8, 9,
2, 1, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
Cserék száma: 15
Ciklusok száma: 36
```

Ábra: Buborék első

Ez a megvalósítás a legegyszerűbb, viszont egy rendezett tömbön is a (külső ciklus) * (belső ciklus) számszor fut le, és végzi az összehasonlításokat. Javíthatunk az algoritmuson, ha figyeljük, hogy a belső ciklusban történt-e csere. Ha nem, akkor a következő külső ciklusban sem fog. Így abba lehet hagyni a rendezést. Ezt mutatja be a következő megvalósítás.

Pszudokód:

```
Buborék2(T: tömb[1..N] egész)
Deklaráció
    i, j: egész
    vége: logikai
Buborék2_kezd
    i := N-1
```

```

vége:= HAMIS
Ciklus amíg i >= 1 ÉS NEM vége
    vége:= IGAZ
    Ciklus j:= 1-től i-ig 1-esével
        Ha T[j] > T[j+1] akkor
            csere(T[j], T[j+1])
            vége:= HAMIS
        Elágazás vége
    Ciklus vége
    i:= i-1
Ciklus vége
Buborék2 vége

```

Példa:

```

A rendezetlen vektor:
3, 6, 2, 7, 4, 9, 1, 8, 5,

Buborék rendezés második változat
3, 6, 2, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 1, 9, 8, 5,
3, 2, 6, 4, 7, 1, 8, 9, 5,
3, 2, 6, 4, 7, 1, 8, 5, 9,
2, 3, 6, 4, 7, 1, 8, 5, 9,
2, 3, 4, 6, 7, 1, 8, 5, 9,
2, 3, 4, 6, 7, 1, 8, 5, 9,
2, 3, 4, 6, 1, 7, 8, 5, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 1, 6, 7, 5, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 1, 4, 6, 5, 7, 8, 9,
2, 3, 1, 4, 5, 6, 7, 8, 9,
2, 1, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
Cserék száma: 15
Ciklusok száma: 35

```

Ábra: Buborék második

Ha a belső ciklusban kiderül, hogy a sorozat már rendezett, akkor kilép. Viszont a külső ciklus csak egyet lép, pedig a rendezettség többet is megengedne. Ezen segíthetünk, ha

megjegyezzük azt a sorszámot, ahol utoljára kellett cserélni. E fölött a sorozat rendezett, így felesleges újra végigjárni. A harmadik algoritmus ezt mutatja be.

Pszudokód:

```
Buborék3(T: tömb[1..N] egész)
Deklaráció
    i, j: egész
    utolsó_csere
Buborék3_kezd
    i := N-1
    Ciklus amíg i >= 1
        utolsó_csere := 0
        Ciklus j := 1-től i-ig 1-esével
            Ha T[j] > T[j+1] akkor
                csere(T[j], T[j+1])
                utolsó_csere := j
            Elágazás vége
        Ciklus vége
    i := utolsó_csere-1
    Ciklus vége
Buborék3 vége
```

Példa:

```
A rendezetlen vektor:
3, 6, 2, 7, 4, 9, 1, 8, 5,

Buborék rendezés második változat
3, 6, 2, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 7, 4, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 9, 1, 8, 5,
3, 2, 6, 4, 7, 1, 9, 8, 5,
3, 2, 6, 4, 7, 1, 8, 9, 5,
3, 2, 6, 4, 7, 1, 8, 5, 9,
2, 3, 6, 4, 7, 1, 8, 5, 9,
2, 3, 4, 6, 7, 1, 8, 5, 9,
2, 3, 4, 6, 1, 7, 8, 5, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 6, 1, 7, 5, 8, 9,
2, 3, 4, 1, 6, 7, 5, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 4, 1, 6, 5, 7, 8, 9,
2, 3, 1, 4, 6, 5, 7, 8, 9,
2, 3, 1, 4, 5, 6, 7, 8, 9,
2, 1, 3, 4, 5, 6, 7, 8, 9,
2, 1, 3, 4, 5, 6, 7, 8, 9,
1, 2, 3, 4, 5, 6, 7, 8, 9,
Cserék száma: 15
Ciklusok száma: 31
```

Ábra: Buborék harmadik

A második és a harmadik buborékrendezés akkor hatékony, ha a sorozat nagy része rendezett.

Pascal kód:

```
Program rendPr;
Uses crt;
Type
  Tvektor= Array[1..9] Of Byte;
Const
  vektor: Tvektor = (3, 6, 2, 7, 4, 9, 1, 8, 5);

Procedure csere(Var a, b: Byte);
Var
  s: Byte;
Begin
  s:=a; a:=b; b:=s;
End;
```

```

Procedure kiir_vektor(Var v: Tvektor);
Var
    i: Byte;
Begin
    For i:= 1 To 9 Do
        write(v[i], ', ');
        writeln;
    End;

Procedure kiir_vektor2(Var v: Tvektor; k, l: Byte);
Var
    i, oldtextattr: Byte;
Begin
    For i:= 1 To 9 Do
        If (i=k) OR (i=l) Then
            Begin
                oldtextattr:= textattr;
                textcolor(4);
                write(v[i], ', ');
                textattr:= oldtextattr;
            End
        Else
            write(v[i], ', ');
        writeln;
    End;

Procedure buborek_rendez(Var v: Tvektor);
Var
    i, j, n, m: Byte;
Begin
    n:= 0;
    m:= 0;
    For i:=9-1 Downto 1 Do
        For j:= 1 To i Do
            Begin
                If v[j] > v[j+1] Then
                    Begin
                        csere(v[j], v[j+1]);
                        kiir_vektor2(v, j, j+1);
                        inc(m)
                    End
                Else
                    kiir_vektor(v);
                inc(n);
            End;
        writeln('Cserék száma: ', m);
        writeln('Ciklusok száma: ', n);
    End;

Procedure buborek2_rendez(Var v: Tvektor);
Var

```

```

    i, j, n, m: Byte;
    vege: Boolean;
Begin
    n:= 0;
    m:= 0;
    i:= 9-1;
    vege:= FALSE;
    While (i >= 1) AND NOT vege Do
        Begin
            vege:= TRUE;
            For j:= 1 To i Do
                Begin
                    If v[j] > v[j+1] Then
                        Begin
                            csere(v[j], v[j+1]);
                            vege:= FALSE;
                            kiir_vektor2(v, j, j+1);
                            inc(m);
                        End
                    Else
                        kiir_vektor(v);
                        inc(n);
                    End;
                dec(i);
            End;
            writeln('Cserék száma: ', m);
            writeln('Ciklusok száma: ', n);
        End;

Procedure buborek3_rendez(Var v: Tvektor);
Var
    i, j, n, m: Byte;
    utolsocsere: Byte;
Begin
    n:= 0;
    m:= 0;
    i:= 9-1;
    While i >= 1 Do
        Begin
            utolsocsere:= 0;
            For j:= 1 To i Do
                Begin
                    If v[j] > v[j+1] Then
                        Begin
                            csere(v[j], v[j+1]);
                            utolsocsere:= j;
                            kiir_vektor2(v, j, j+1);
                            inc(m);
                        End
                    Else
                        kiir_vektor(v);
                End
            End;
            i:= utolsocsere;
        End;
    End;

```

```

        inc(n);
    End;
    i:= utolsocsere-1;
End;
writeln('Cserék száma: ', m);
writeln('Ciklusok száma: ', n);
End;

Begin
    clrscr;
    writeln('A rendezetlen vektor:');
    kiir_vektor(vektor);
    {writeln; writeln('Buborék rendezés első változat');
    buborek_rendez(vektor);}
    {writeln;      writeln('Buborék      rendezés      második
változat');
    buborek2_rendez(vektor);}
    writeln;      writeln('Buborék      rendezés      második
változat');
    buborek3_rendez(vektor);
    readln;
End.

```

Beszúrásos rendezés

A beszúrásos, vagy pókerrendezés hasonlít arra, ahogy az ember kártyaosztás után elrendezi a lapokat a kezében. Felvesszük az első lapot, majd a másodikat, és helyére tesszük. A helyét, egy adott rendező elv határozza meg. Minden újabb lapnál megkeressük és elkészítjük a helyét, majd betesszük a lapot.

Pszeudokód:

```

Beszúrásos(T: tömb[1..N] egész)
Deklaráció
    x, i, j: egész
Beszúrásos_kezd
    Ciklus i:= 2-től N-ig
        j:= i-1
        x:= T[i]
        Ciklus amíg j >= 1 ÉS x < T[j]
            T[j+1]:= T[j]
            j:= j-1
        Ciklus vége
        T[j+1]:= x
    Ciklus vége
Beszúrásos vége

```

Metszetképzés

A metszet, vagy közös rész meghatározása során két vagy több sorozat azon elemeit válogatjuk ki, amelyek minden sorozatban benne vannak. A sorozatok nincsenek rendezve, és feltételezzük, hogy minden elem csak egyszer fordul elő az egyes sorozatokban. Az képződő sorozat elemszáma kisebb vagy egyenlő, mint a legkisebb elemszámú bemenő sorozat elemszáma.

A feladatot úgy oldjuk meg, hogy végigmegyünk a legkisebb sorozaton, és az aktuális elemet keressük a többi sorozatban. Ha mindegyikben benne van, akkor betesszük a közös rész sorozatába.

Feladat: Három bemenő sorozatból válogassuk ki a közös részt egy új sorozatba!

Pascal kód:

```
Program metszetPr;
Uses crt;
Const
  N1=20; N2=30; N3=40;
Type
  Tv1=Array[0..N1] Of Byte;
  Tv2=Array[0..N2] Of Byte;
  Tv3=Array[0..N3] Of Byte;
  Tmetszet=Array[0..N1] Of Byte;

{Függvény ellenőrzi, hogy egy elemet tartalmaz-e egy
tömb.}
Function bennevan(Var v: Array of Byte; elem: Byte):
Boolean;
  Var
    i, n: Byte;
  Begin
    i:=0;
    n:= High(v); {Vektor utolsó elemének indexe.}
    While (i <= n) AND (v[i] <> elem) Do
      inc(i);
    bennevan:= i <= n;
  End;

{Eljárás, mely véletlen egyedi számokkal feltölti a
vektorokat.}
Procedure feltolt_vektor(Var v: Array of Byte);
  Var
    i,e: Byte;
    n: Byte; {Elemszáma a tömbnek.}
  Begin
```

```

        n:= High(v);
        For i:=0 To n Do
            Begin
                Repeat
                    e:= random(51); {0 és 50 közötti véletlen
szám}
                    Until NOT bennevan(v,e);
                    v[i]:= e;
                End;
            End;
        End;

{Kiírja a vektor tartalmát, tízesével.}
Procedure kiir_vektor(Var v: Array Of Byte);
    Var
        i, n: Byte;
    Begin
        n:= High(v);
        For i:= 0 To n Do
            Begin
                If i MOD 10 = 0 Then {Tízesével sort emelünk.}
                    writeln;
                    write(v[i], ', ');
                End;
            End;
        writeln;
    End;

{Kiírja a metszet vektor tartalmát, tízesével, ha van
közös elem.}
Procedure kiir_vektor2(Var v: Array Of Byte; im:
Integer);
    Var
        i: Byte;
    Begin
        If im > 0 Then
            Begin
                writeln('Közös elemek:');
                For i:= 0 To im Do
                    Begin
                        If (i MOD 10 = 0) AND (i<>0) Then
{Tízesével sort emelünk.}
                            writeln;
                            write(v[i], ', ');
                        End;
                    End
                Else
                    writeln('Nincs közös elemük!');
                    writeln;
                End;
            End;
        End;
    End;

{Három vektor metszetének meghatározása.}

```

```

Procedure metszet_vektorok(Var v1, v2, v3, m: Array Of
Byte; Var im: Integer);
  Var
    i: Byte;
  Begin
    i:= 0;
    im:= -1; {A -1 miatt kell Integernek deklarálni.}
    While i <= High(v1) Do {Az első vektor végéig
megyünk.}
      Begin
        {Ha az első vektor aktuális eleme benne van a
másik két vektorban.}
        If bennevan(v2, v1[i]) AND bennevan(v3, v1[i])
Then
          Begin
            inc(im);
            m[im]:= v1[i];
          End;
          inc(i);
        End;
      End;
    End;

  Var
    vektor1: Tv1; vektor2: Tv2; vektor3: Tv3; metszet:
Tmetszet;
    im: Integer;
  Begin
    clrscr;
    writeln('A program három vektor metszetét határozza
meg. ');
    writeln;
    {Vektorok feltöltése.}
    feltolt_vektor(vektor1);
    feltolt_vektor(vektor2);
    feltolt_vektor(vektor3);
    {Vektorok kiíratása.}
    RandSeed:= 4432512; {A véletlenszám változó
beállítása külön-külön.}
    writeln('Első vektor: ');
    kiir_vektor(vektor1);
    writeln;
    RandSeed:= 457200234;
    writeln('Második vektor: ');
    kiir_vektor(vektor2);
    writeln;
    RandSeed:= 603;
    writeln('Harmadik vektor');
    kiir_vektor(vektor3);
    writeln;
    {Metszet meghatározása.}

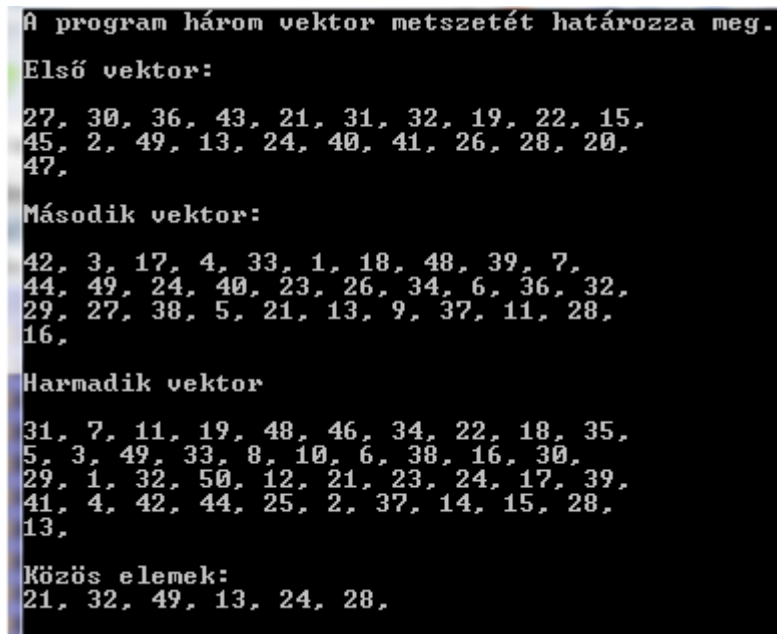
```

```

    metszet_vektorok(vektor1, vektor2, vektor3, metszet,
im);
    kiir_vektor2(metszet, im);
    writeln;
    readln;
End.

```

Futási kép:



```

A program három vektor metszetét határozza meg.
Első vektor:
27, 30, 36, 43, 21, 31, 32, 19, 22, 15,
45, 2, 49, 13, 24, 40, 41, 26, 28, 20,
47,
Második vektor:
42, 3, 17, 4, 33, 1, 18, 48, 39, 7,
44, 49, 24, 40, 23, 26, 34, 6, 36, 32,
29, 27, 38, 5, 21, 13, 9, 37, 11, 28,
16,
Harmadik vektor
31, 7, 11, 19, 48, 46, 34, 22, 18, 35,
5, 3, 49, 33, 8, 10, 6, 38, 16, 30,
29, 1, 32, 50, 12, 21, 23, 24, 17, 39,
41, 4, 42, 44, 25, 2, 37, 14, 15, 28,
13,
Közös elemek:
21, 32, 49, 13, 24, 28,

```

Ábra: Metszet képzés

A `feltolt_vektor()` eljárás nyitott tömb paramétert (Array Of elemtípus) használ. Amellyel olyan alprogramokat alakíthatunk ki, melyek hívása nem függ a paramétertömb méretétől. A tömb 0-tól indexelt, és az utolsó elemének az indexét a `High()` függvénnyel kérdezhetjük le. A `RandSeed` system egységbeli változót külön állítom be, hogy a három vektor elemei ne essenek teljesen egybe. Ha a `randomize()` eljárást használnám, akkor a rendszeridő alapján állítaná be a `RandSeed` változót, és a három vektor első tíz eleme teljesen megegyezne.

Unióképzés

A bemenő sorozatokból készítünk egy kimenő sorozatot. Itt a bemenő sorozatok egyesítését (unióját) keressük. Az unióba bekerül az elem, ha része valamelyik kiindulási sorozatnak. A unió elemszáma maximálisan a kiindulási sorozatok

elemszámainak az összege lehet. Itt is követelmény, hogy az egyes vektorokban minden elem csak egyszer szerepel.

A feladatot úgy oldjuk meg, hogy az első sorozatot teljes egészében beletesszük a kimenő sorozatba. Majd a többi sorozatot megvizsgáljuk, hogy van-e olyan elemük, amely még nincs benne és, ha igen, akkor beletesszük.

Feladat: Az előző feladatot bővítve, készítsük el a három vektor unióját!

Pascal kód:

```
...
Type
...
    Tunio=Array[0..UN] Of Byte;
...
{Kiírja az unió vektor tartalmát, tízesével.}
Procedure kiir_vektor3(Var v: Array Of Byte; iu:
Integer);
    Var
        i: Byte;
    Begin
        writeln('Unió tartalma:');
        For i:= 0 To iu Do
            Begin
                If (i MOD 10 = 0) AND (i <> 0) Then {Tízesével
sort emelünk.}
                    writeln;
                write(v[i], ', ');
            End;
        writeln;
    End;
...
{Három vektor uniója.}
Procedure unio_vektorok(Var v1, v2, v3, u: Array Of
Byte; Var iu: Integer);
    Var
        i: Byte;
    Begin
        iu:= High(v3);
        {A legnagyobb vektor bemásolása a kimenő vektorba.}
        For i:= 0 To iu Do
            u[i]:= v3[i];
        {Egyesítés}
        For i:=0 To High(v1) Do
            If NOT bennevan(u, v1[i]) Then
                Begin
                    inc(iu);
```

```

        u[iu]:= v1[i];
    End;
For i:=0 To High(v2) Do
    If NOT bennevan(u, v2[i]) Then
        Begin
            inc(iu);
            u[iu]:= v2[i];
        End;
    End;
End;

Var
    vektor1: Tv1; vektor2: Tv2; vektor3: Tv3;
    unio: Tunio;
    iu: Integer;
Begin
    clrscr;
    writeln('A program három vektor metszetét határozza
meg. ');
    writeln;
    {Vektorok feltöltése.}
    feltolt_vektor(vektor1);
    feltolt_vektor(vektor2);
    feltolt_vektor(vektor3);
    {Vektorok kiíratása.}
    RandSeed:= 4432512; {A véletlenszám változó
beállítása külön-külön.}
    writeln('Első vektor:');
    kiir_vektor(vektor1);
    writeln;
    RandSeed:= 457200234;
    writeln('Második vektor:');
    kiir_vektor(vektor2);
    writeln;
    RandSeed:= 603;
    writeln('Harmadik vektor');
    kiir_vektor(vektor3);
    writeln;
    {Unió meghatározása.}
    unio_vektorok(vektor1, vektor2, vektor3, unio, iu);
    kiir_vektor3(unio, iu);
    readln;
End.

```

Futási kép:

```
A program három vektor metszetét határozza meg.  
Első vektor:  
27, 30, 36, 43, 21, 31, 32, 19, 22, 15,  
45, 2, 49, 13, 24, 40, 41, 26, 28, 20,  
47,  
Második vektor:  
42, 3, 17, 4, 33, 1, 18, 48, 39, 7,  
44, 49, 24, 40, 23, 26, 34, 6, 36, 32,  
29, 27, 38, 5, 21, 13, 9, 37, 11, 28,  
16,  
Harmadik vektor  
31, 7, 11, 19, 48, 46, 34, 22, 18, 35,  
5, 3, 49, 33, 8, 10, 6, 38, 16, 30,  
29, 1, 32, 50, 12, 21, 23, 24, 17, 39,  
41, 4, 42, 44, 25, 2, 37, 14, 15, 28,  
13,  
Unió tartalma:  
31, 7, 11, 19, 48, 46, 34, 22, 18, 35,  
5, 3, 49, 33, 8, 10, 6, 38, 16, 30,  
29, 1, 32, 50, 12, 21, 23, 24, 17, 39,  
41, 4, 42, 44, 25, 2, 37, 14, 15, 28,  
13, 27, 36, 43, 45, 40, 26, 20, 47, 9,
```

Ábra: Unióképzés

Összefuttatás

Az összefuttatásnál több bemenő sorozatból készítünk egy kimenő sorozatot. A bemenő sorozatok rendezettek, és a kimenő sorozat is rendezett lesz. A bemenő sorozatokat egyszerre dolgozzuk fel úgy, hogy elérhetővé tesszük az első elemeket. Amelyik kisebb, azt áttesszük a kimenő sorozatba, majd a feldolgozott sorozat indexét eggyel növeljük. A sorozatok rendezettsége miatt az egyenlő elemek egyszerre lesznek elérhetők. Az összes elem feldolgozása után, előáll a rendezett kimeneti sorozat.

A bemenő sorozatok végeire egy-egy ütközőt teszünk. Ez olyan elem, amely mindegyik sorozatbeli elemnél nagyobb. Így a feldolgozásnak akkor van vége, ha minden sorozat indexe az ütközőn áll.

Pszeudokód:

```
Összefésül:  
  Ütközők elhelyezése.  
  Ráállítás minden bemenő sorozat első elemére.  
  Ciklus amíg van elem valamelyik bemenő sorozatban.  
    Elemek összehasonlítása  
    A legkisebb átírása a kimenő sorozatba.
```

Indexének növelése.
Ciklus vége
Összefésül vége

Mátrixműveletek

Mátrix feltöltése

Két egymásba ágyazott ciklussal.

Mátrixok összeadása

A tömböknek azonosaknak kell lenniük. Két mátrix összegét úgy kapjuk, hogy a megfelelő elemeket rendre összeadjuk: $C[m,n] := A[m,n] + B[m,n]$.

Mátrixok szorzata

Két mátrix szorzatának meghatározásához a mátrixoknak kompatibiliseknek kell lenniük. Legyen adottak az $A[m,k]$ és a $B[k,n]$ mátrixok, ezek szorzata egy $C[m,n]$ -es mátrix lesz. Vagyis a baloldali mátrixnak ugyanannyi oszlopának kell lennie, mint ahány sora a jobboldali mátrixnak van. A C mátrix $[i,j]$ eleme úgy áll elő, hogy az A mátrix i . sorának és a B mátrix j . oszlopának képezzük a szorzatösszegüket.

Mátrix transzponáltja

Négyzetes mátrix elemeinek tükrözése a főátlóra.

Feladat: Készítsük el két 3×4 -es és 4×2 -as mátrix szorzatát!

Pascal kód:

```
Program matrix2Pr;
Uses crt;
Const
  M=3; K=4; N=2;
Type
  TA= Array[1..M, 1..K] Of Byte; {3*4}
  TB= Array[1..K, 1..N] Of Byte; {4*2}
  TC= Array[1..M, 1..N] Of Byte; {3*2}

  {Mátrix kiíratása.}
Procedure kiir_matrixC (Var c:TC);
Var
  i,j: Byte;
```

```

Begin
  For i:= 1 To M Do
    Begin
      For j:= 1 To N Do
        write(c[i,j]:4);
        writeln;
      End;
    End;
  End;

Procedure kiir_matrixB(Var b:TB);
Var
  i,j: Byte;
Begin
  For i:= 1 To K Do
    Begin
      For j:= 1 To N Do
        write(b[i,j]:4);
        writeln;
      End;
    End;
  End;

Procedure kiir_matrixA(Var a:TA);
Var
  i,j: Byte;
Begin
  For i:= 1 To M Do
    Begin
      For j:= 1 To K Do
        write(a[i,j]:4);
        writeln;
      End;
    End;
  End;

{Két mátrix szorzata.}
Procedure szorzat_matrix(Var a: TA; Var b: TB; Var c:
TC);
Var
  i,j,l: Byte;
  sz: Word;
Begin
  For i:= 1 To M Do
    For j:= 1 To N Do
      Begin
        sz:= 0;
        For l:= 1 To K Do
          sz:= sz + A[i,l] * B[l,j];
        c[i,j]:= sz;
      End;
    End;
  End;

```

```

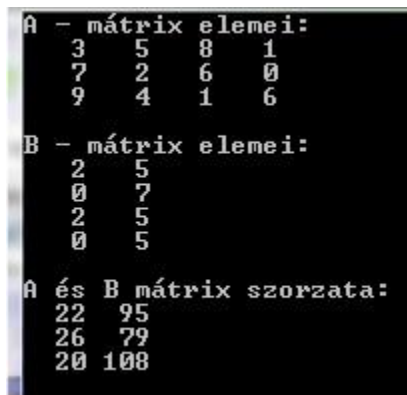
{A, B mátrix kezdőértékekkel való deklarálása.}
Const
  A: TA= ((3,5,8,1),(7,2,6,0),(9,4,1,6));
  B: TB= ((2,5),(0,7),(2,5),(0,5));

Var
  C: TC;

Begin
  clrscr;
  writeln('A - mátrix elemei:');
  kiir_matrixA(A);
  writeln;
  writeln('B - mátrix elemei:');
  kiir_matrixB(B);
  writeln;
  writeln('A és B mátrix szorzata:');
  szorzat_matrix(A, B, C);
  kiir_matrixC(c);
  readln;
End.

```

Futási kép:



```

A - mátrix elemei:
 3  5  8  1
 7  2  6  0
 9  4  1  6

B - mátrix elemei:
 2  5
 0  7
 2  5
 0  5

A és B mátrix szorzata:
22  95
26  79
20 108

```

Ábra: Mátrixok szorzata

VEREM kezelése

A VEREM (STACK) egy szekvenciális adatszerkezet, amely a földbe ásott „verem” elvén működik. A verembe betenni mindig csak a tetejére, kivenni mindig csak a tetejéről lehet. Szokásos elnevezése még LIFO (Last In First Out – Utolsó Be Első Ki) adatszerkezet. A vermet tömb segítségével valósítjuk meg.

Szokásos verem műveletek:

INIT – verem inicializálása, a verem mutató (stack pointer) SP kezdő állapotba állítása.

TELE – verem lekérdezése, hogy tele van-e.

ÜRES – verem lekérdezése, hogy üres-e.

PUSH – egy elem betétele a verembe, mindig a tetejére.

POP – egy elem kivétele a veremből, mindig a tetejéről.

Feladat: Készítsünk egy navigációs szoftvert, amely megjegyzi azokat a helyeket, ahol jártunk, és segít abban, hogy ugyanazon az útvonalon vissza is térjünk!

A feladat megoldása során, fel kell jegyeznünk azokat a helyeket, amelyeket érintettünk, majd a vissza utat generálnunk kell. Erre építünk egy VEREM szerkezetet. Ahogy haladunk, beletesszük a helységek nevét, majd az út végén kilistázzuk a tartalmát. Mivel az elemeket fordított sorrendben tudjuk kivenni, így előáll a megoldás.

Pascal kód:

```
Program veremPr;
Uses crt;
Const
  N=30;
Type
  Telem = String[40];
  Tverem = Array[1..N] Of Telem;

{Verem inicializálása.}
Procedure init(Var sp: Byte);
Begin
  sp:= 1;
End;

{Üres verem lekérdezése.}
Function ures(Var sp: Byte): Boolean;
Begin
  ures:= sp = 1;
End;

{Tele verem lekérdezése.}
Function tele(Var sp: Byte): Boolean;
Begin
  tele:= sp = N;
End;
```

```

{Elem betétele.}
Procedure push(Var v: Tverem; Var sp: Byte; elem:
Telem);
Begin
  If NOT tele(sp) Then
    Begin
      v[sp]:= elem;
      inc(sp);
    End;
End;

{Elem kivétele.}
Procedure pop(Var v: Tverem; Var sp: Byte; Var elem:
Telem);
Begin
  If NOT ures(sp) Then
    Begin
      dec(sp);
      elem:= v[sp];
    End;
End;

Var
  verem: Tverem;
  elem: Telem;
  sp: Byte;
Begin
  clrscr;
  writeln('Útvonal nyilvántartó');
  writeln('Kérem adja meg a meglátogatott helyeket
(üres karakterig)');
  {Verem inicializálása.}
  init(sp);
  {Verem feltöltése.}
  write(sp, '. hely: ');
  readln(elem);
  While elem <> '' Do
    Begin
      push(verem, sp, elem);
      write(sp, '. hely: ');
      readln(elem);
    End;
  {Elemek kiírása.}
  writeln;
  writeln('Visszaút: ');
  While NOT ures(sp) Do
    Begin
      pop(verem, sp, elem);

```

```
writeln(elem);  
End;  
readln;  
End.
```

SOR kezelése

A sor egy szekvenciális adatszerkezet, amelyből mindig a legelsőnek betett elemet lehet kivenni. Szokásos elnevezése még FIFO (First In First Out – Első Be Első Ki).

Szokásos műveletek:

PUT – Elem betétele a sorba, mindig a végére.

GET – Elem kivétele a sorból, mindig az elejéről.

A SOR adatszerkezettel olyan feladatokat oldunk meg, amelyeknél az adatok feldolgozása érkezési sorrendben történik.

Logaritmikus keresés

Rendezett sorozatban kereshetünk egy elemet. A keresés maximális időigénye $\log_2(N)$, N az elemszám, ezért nevezik a keresést logaritmikus keresésnek. Minél nagyobb az elemszám, annál jobban megmutatkozik az algoritmus hatékonysága. Például 1024 elemszámnál $\log_2(1024)=10$, azaz 10 maximum 10 lépésen belül megvan az elem vagy kiderül, hogy nincs benne.

Pszeudokód:

```
Logaritmikus (be:tömb[1..N], be:keresett, ki: sorszám)  
Deklaráció  
    alsó, felső: egész  
Logaritmus kezdet  
    alsó:=1  
    felső:= N  
Ciklus  
    sorszám:= (alsó + felső) DIV 2  
    Ha tömb[sorszám] < keresett Akkor  
        alsó:= sorszám + 1  
    Ha vége  
    Ha tömb[sorszám] > keresett Akkor  
        felső:= sorszám - 1  
    Ha vége  
Mígnem (tömb[sorszám] = keresett) VAGY (alsó > felső)  
Ciklus vége  
Ha alsó > felső Akkor
```

```
sorszám := -1
Ha vége
Logaritmikus vége
```

Rekurzió

A rekurzió során a feladat megoldásához úgy jutunk el, hogy találunk egy ismételt egyszerűsítési folyamatot. Ennek a folyamatnak olyannak kell lennie, hogy véges sok lépésben eljussunk egy olyan állapotba, amelynél a megoldás már magától értetődő. Tehát, a problémát visszavezetjük egy egyszerűbb problémára, majd ezt az egyszerűsítést addig végezzük, amíg eljutunk a legegyszerűbb (triviális) esethez. Ezt megoldva, már az eggyel bonyolultabb eset is megoldható, míg végül eljutunk a teljes probléma megoldásához.

Például az első N természetes szám összegének kiszámítása, visszavezethető az $N-1$ összegének meghatározására, amelyhez még hozzáadjuk az N értékét.

$$\begin{aligned}\text{összeg}(N) &= \text{összeg}(N-1) + N \\ \text{összeg}(N-1) &= \text{összeg}(N-2) + (N-1)\end{aligned}$$

Ezt az egyszerűsítést addig folytatjuk, amíg $N=1$ legegyszerűbb esetet meg nem találjuk. Ha ezt ismerjük, akkor meg tudjuk oldani az $N=2$ esetet is.

$$\text{összeg}(2) = \text{összeg}(1) + 2$$

Nézzük meg egy példán keresztül, legyen $N = 5$:

$$\begin{aligned}\text{összeg}(5) &= \text{összeg}(4) + 5 \\ \text{összeg}(4) &= \text{összeg}(3) + 4 \\ \text{összeg}(3) &= \text{összeg}(2) + 3 \\ \text{összeg}(2) &= \text{összeg}(1) + 2 \\ \text{összeg}(1) &= 1\end{aligned}$$

Ez a triviális eset, innen fokozatosan megoldhatók a bonyolultabb esetek is.

$$\begin{aligned}\text{összeg}(2) &= \underline{1+2} \\ \text{összeg}(3) &= \underline{\quad 3 \quad} + 3 \\ \text{összeg}(4) &= \underline{\quad 6 \quad} + 4 \\ \text{összeg}(5) &= \quad 10 \quad + 5 = \underline{15}\end{aligned}$$

A rekurzív feladatokat, rekurzív alprogramokkal (eljárás, függvény) oldjuk meg. Beszélhetünk,

- közvetlen rekurzióról, amikor az alprogram saját magát hívja meg, illetve
- közvetett rekurzióról, amikor az alprogram, egy másik alprogram által hívódik meg. Az „A” alprogram hívja „B” alprogramot, és „B” hívja „A”-t.

A rekurzív alprogramban szerepelnie kell:

- egy egyszerűsítési folyamatnak, amely lépésről-lépésre változik és a triviális esetben végződik,
- egy küszöbfeltételnek, amely leállítja az egyszerűsítést.

Az alprogramok hívása során felépül egy hívási lánc, a triviális eset megtalálásáig. Majd ezután a lánc utolsó tagja (triviális eset) visszaadja a vezérlést a hívási lánc utolsó előtti tagjának. Miután megoldotta a saját részproblémáját, az ezt megelőző alprogramra kerül a vezérlés. A részproblémák megoldása a hívási lánc első eleméig folytatódik, amelynek végrehajtása során a teljes feladat megoldódik.

Minden alprogram hívása során a verembe kerülnek a lokális változók, a paraméterek, és a visszatérési cím. A hívási láncon való visszajutást ez az adatszerkezet segíti. A rekurzió hátránya, hogy a verem egy véges adatszerkezet, ezért a rekurzió mélysége korlátozott. Illetve az alprogramok hívása lassítja a feldolgozást. Előnye viszont az átláthatóbb programszerkezet.

Minden rekurzív feladatnak, van iteratív párja.

Feladat: Határozzuk meg az első N természetes szám összegét!

```
Program rekurzioPr;
Uses crt;

{Rekurzív függvény az első N természetes szám
összegére.}
Function osszeg(N: Word): Longint;
Begin
    If N = 1 Then
        osszeg:= 1 {A triviális eset.}
    Else
        osszeg:=  osszeg(N-1)    +    N;    {A rekurzív
hivatkozás.}
```

```

End;

Var
  N: Word;
Begin
  clrscr;
  writeln('A program kiszámolja az első N természetes
szám összegét.');
```

```

  writeln;
  write('N = ');
  readln(N);
  writeln('Összeg = ', összeg(N));
  readln;
End.
```

Az összeg() függvény, rekurzív módon hívódik meg.

Feladat: Készítsünk programot, amely a képernyőre rajzol egy karakterből felépülő oszlopot, és ez az oszlop hol növekszik, hol zsugorodik! A pulzálást kísérje hangjelenség.

A feladatot rekurzív eljárással oldjuk meg.

Pascal kód:

```

Program pulzalPr;
Uses crt;

{A pulzal 0-tól 15 sor magasságig készít téglalapot,
majd vissza.}
Procedure pulzal(n: word);

  {Kiír egymás mellé két karaktert, és megszólaltatja a
hangszórót.}
  Procedure kiir(kar: Char);
  Begin
    gotoXY(40, 25-n); {A kurzort adott X,Y pozícióba
viszi.}
    write(kar);
    write(kar);
    sound(500); {Hangszóró megszólaltatása adott Hz-
en.}
    delay(200); {Késleltetés ezredmásodpercig.}
    nosound;    {Hangszóró leállítás.}
  End;

  Begin
```

```

        kiir('=');
        If n < 15 Then
            pulzal(n+1);
        kiir(' ');
        If (n = 0) AND NOT keyPressed Then
            pulzal(0);
        End;

Begin
    clrscr;
    pulzal(0);
End.

```

A keyPressed logikai függvény, amely igazat (TRUE) ad vissza, ha van várakozó karakter a billentyűzetpufferben, és hamis (FALSE), a üres a puffer. A delay(ezredmásodperc) eljárás a paraméterében megadott ezredmásodpercig késlelteti a végrehajtást. A sound(frekvencia) eljárás adott frekvenciájú hangot állít elő, amit a noSound eljárással lehet abbahagyni.

Gyorsrendezés

A gyorsrendezés (quicksort) az egyik leghatékonyabb rendező algoritmus, amely a rekurzióra épül. A rendezés alapja, hogy a sorozatot az elemek cserélgetésével két olyan részre osztjuk, amelynek elemei a sorozat egy kitüntetett eleménél kisebbek, illetve nagyobbak. Majd ezt a módszert alkalmazva a két részsorozatra, végül 1 elemű részsorozatokat kapunk, amelyek már önmagukban rendezettek. A kitüntetett elem lehet például a sorozat középső eleme.

Nézzünk egy példát:

```

rendezendő:
    1. 6, 2, 5, 8, 4, 9, 3

rendezés:
    2. 6, 2, 5, 8, 4, 9, 3
    3. 6, 2, 5, 3, 4, 9, 8
    4. 4, 2, 5, 3, 6, 8, 9
    5. 4, 2, 3, 5, 6
    6. 2, 4, 3
    7. 2, 3, 4
a rendezett sorozat:
    8. 2, 3, 4, 5, 6, 8, 9

```

A sárgával kiemelt elemek felcserélődnek. Aláhúzással lett jelölve az a kezdeti sorozat, illetve később a részsorozatok, amelyeken belül történik az elemek felcserélése. Az első sorban látható a rendezetlen sorozat. A második sorban a kitüntetett (középső) elem a 8-as. Megvizsgáljuk, hogy tőle balra csak kisebbek vannak-e, illetve jobbra csak nagyobbak. A nem megfelelő elemeket kicseréljük csere(8, 3). A harmadik sorban a 4, 9 egymáshoz képest jó helyen van. Ezután, mivel $i > j$ kilépünk a ciklusból. Két részsorozat (6, 2, 5, 3, 4) és (9, 8) képződik, amelyet az első alapján elkezdünk rendezni. A rendezett számoknál nincs aláhúzás. A 6. sorban újra két sorozat van (2), és (4, 3), a 2 önmagában rendezett, így csak a (4, 3) sorozatot kell rendezni. Az utóbbi két számot felcserélve a sorozat rendezetté válik. A rendezéshez 6 cserére volt szükség.

Pszeudokód:

```

Gyorsrendezés(Cím T: tömb[1..N] egész, alsó, felső:
egész)
    i, j: egész
    középső: egész
Gy_kezdet
    i:= alsó
    j:= felső
    középső:= (alsó + felső) DIV 2
    Ciklus
        Ciklus Amíg T[i] < T[középső]
            i:= i + 1
        Ciklus vége
        Ciklus Amíg T[j] > T[középső]
            j:= j - 1
        Ciklus vége
        Ha i < j Akkor
            Csere(T[i], T[j])
        Ha vége
        Ha i <= j Akkor
            i:= i + 1
            j:= j - 1
        Ha vége
    Ciklus vége Ha i > j
    Ha alsó < j Akkor
        Gyorsrendezés(T, alsó, j)
    Ha vége
    Ha felső > i Akkor
        Gyorsrendezés(T, i, felső)
    Ha vége
Gy_vége

```

Leszámláló rendezés (Ládarendezés)

Bizonyos esetekben a gyorsrendezésnél hatékonyabb rendezést kapunk a most ismertetendő algoritmussal. Akkor alkalmazzuk, ha a rendezendő „n” elem közül, az elemek csak 1 és „k” között helyezkednek el. Az algoritmus meghatározza, hogy egy adott elem előtt hány elem helyezkedik el. Ezzel pedig az adott elemet, rögtön a saját pozíciójába lehet tenni.

Az algoritmus felhasznál három tömböt:

- „A” tömb a rendezendő elemek tömbje → bemenet,
- „B” tömb a rendezett elemek tömbje → kimenet,
- „C” tömb a rendezés során felhasznált segédtömb.

Pszeudokód:

```
Leszámláló_rendezés(A, B, k)
  Bemenet: „A” és „B” számtömbök, k a maximális elem
érték
  Eredmény: a rendezetlen „A” tömb elemeit a „B” tömbbe
rendezi nagyság szerint.
Leszámláló_kezd
//A számláló tömb „C” törlése
  for i:=1 to k do
    C[i]:=0
  endfor

//Az „A” tömb értékei hányszor fordulnak elő.
  for i:=0 to A.hossz do
    C[A[i]]:= C[A[i]]+1
  endfor

//Hány kisebb vagy egyenlő kulcsú elem van a
sorozatban. A C[i]-C[i-1] különbség meghatározza, hogy
az „i.” index által meghatározott „A” tömbelem hányszor
fordul elő az „A” tömbben.
  for i:=2 to k do
    C[i]:= C[i]+C[i-1]
  endfor

//A „B” tömb rendezett feltöltése a „C” tömb alapján. A
C[i] értéke megadja az elem leendő helyét a „B”
tömbben. Minden egyes elem elhelyezése után a C[i]
értékét csökkentjük eggyel.
  for i:= 1 to A.hossz do
    B[C[A[i]]]:= A[i]
```

```
    C[A[i]]:= C[A[i]]-1
endfor
Leszámláló_vége
```

Nézzük meg az algoritmust egy példán keresztül.

Legyen egy 9 elemű rendezendő „A” tömbünk:

1, 2, 3, 4, 5, 6, 7, 8, 9 → a tömb indexei
1, 5, 2, 1, 4, 2, 1, 4, 5 → a tömb elemei

k = 1..5

Az 5 elemszámú „C” tömb nullázása:

1, 2, 3, 4, 5 → a tömb indexei
0, 0, 0, 0, 0 → a tömb elemei

Melyik kulcs hányszor fordul elő:

A „C” tömb indexei jelentik az „A” tömb elemeit. Végigmegyünk az „A” tömbön, és az A[i] által kijelölt „C” tömb értékét megnöveljük eggyel.

3, 2, 0, 2, 2 → a „C” tömb elemei

Hány kisebb vagy egyenlő kulcsú elem van a sorozatban:

Meghatározza az A[i] elhelyezkedését a „B” tömbben.

3, 5, 5, 7, 9 → a „C” tömb elemei

Rendezés:

Első iteráció.

1, 5, 2, 1, 4, 2, 1, 4, 5 → „A” első eleme kiválasztja a „C” első indexét.

3, 5, 5, 7, 9 → „C” első eleme 3, amely meghatározza az „A” tömb első elemének a helyét a „B” tömbben.

0, 0, 1, 0, 0, 0, 0, 0, 0 → „B” tömbben az „A” első eleme a helyére került.

2, 5, 5, 7, 9 → „C” tömb első elemét csökkentjük eggyel.

Második iteráció.

1, 5, 2, 1, 4, 2, 1, 4, 5 → „A” második eleme kiválasztja a „C” ötödik elemét.

2, 5, 5, 7, 9 → „C” ötödik indexű eleme a 9, amely a „B” tömb azon indexét határozza meg, ahova az A[i] elemet helyezzük.

0, 0, 1, 0, 0, 0, 0, 0, 5 → „B” tömbben helyére került az „A” második eleme az 5-ös.

2, 5, 5, 7, 8 → „C” tömb ötödik elemét csökkentjük eggyel.

Harmadik iteráció.

1, 5, 2, 1, 4, 2, 1, 4, 5 → „A” harmadik eleme kiválasztja a „C” második elemét.

2, 5, 5, 7, 9 → „C” második indexű eleme az 5, amely a „B” tömb azon indexét határozza meg, ahova az A[i] elemet helyezzük.

0, 0, 1, 0, 2, 0, 0, 0, 5 → „B” tömbben helyére került az „A” harmadik eleme az 2-es.

2, 4, 5, 7, 9 → „C” tömb második elemét csökkentjük eggyel.

Negyedik iteráció.

1, 5, 2, 1, 4, 2, 1, 4, 5 → „A” negyedik eleme kiválasztja a „C” első elemét.

2, 5, 5, 7, 9 → „C” első indexű eleme a 2, amely a „B” tömb 2. indexét határozza meg, ahova az A[i] elemet (1-es) helyezzük.

0, 1, 1, 0, 2, 0, 0, 0, 5 → „B” tömbben helyére került az „A” negyedik eleme az 1-es.

1, 4, 5, 7, 9 → „C” tömb első elemét csökkentjük eggyel.

.
. .
.

Kilencedik iteráció:

1, 1, 1, 2, 2, 4, 4, 5, 5 → „B” tömbben helyére került az „A” kilencedik eleme az 5-ös.

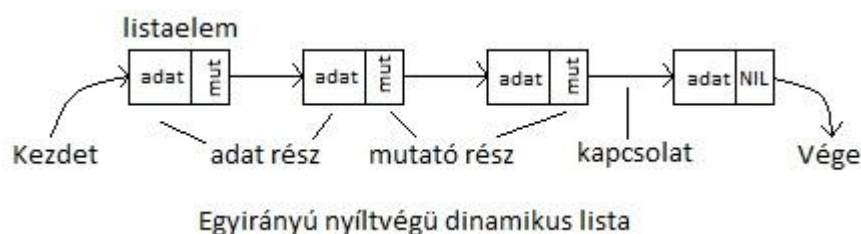
0, 3, 5, 5, 7 → „C” tömb ötödik elemét csökkentjük eggyel.

Az i . legnagyobb/legkisebb elem meghatározására felhasználhatjuk a „C” tömböt. Addig lépkedünk a „C” tömbben, amíg a $C[j] < i$, ($j=1..k$). A „C” indexe lesz a keresett elem.

Dinamikus lista (adatok szétszórt ábrázolása, tárolása)

A *listás* tárolás, az elemek egymásutániságára utal. Vagyis arra, hogy minden elemnek, kivéve az elsőt, van megelőző, és kivéve az utolsót, van rákövetkező társa. Ha a tárolást a tömb adatszerkezettel valósítjuk meg, akkor *statikus listát* kapunk. Ennél a listánál előre el kell döntenünk, hogy maximálisan mekkora adathalmazt akarunk kezelni. Hátránya, hogy kis elemszámnál az előre lefoglalt hely miatt pazarol, illetve a maximális méreténél több elemet nem tudunk eltárolni. Előnye a gyors elérés. A *dinamikus lista*, egy olyan adatszerkezet, amelynél a maximális méretet a memória (heap, halom terület) mérete korlátozza. Hátránya a lassabb adatelérés.

A dinamikus lista, *listaelemekből* épül fel. A listaelemek olyan összetett adatszerkezetek, amelyeknek van egy feladattól függő *adattároló részük*, és egy a lista megvalósításától függő *mutató részük*. Ezért a listaelemeket rekorddal (struktúrával) valósítjuk meg. A mutatóból a megvalósítástól függően lehet több is, de mindegyikre jellemző, hogy a következő (megelőző) listaelemre mutat. A listát az első eleme egyértelműen azonosítja.



Ábra: Dinamikus lista

Dinamikus lista típusok:

- *Egyirányú nyíltvégű dinamikus lista:* A lista kezdetét jelölni kell, egy mutatóval. A lista végét az utolsó elem mutatójának NIL értéke jelzi. A NIL egy olyan érték, amely nem mutat érvényes memóriacímre, azaz jelzi a lista végét. A listaelemeket egy irányban a kezdetétől a végéig tudjuk elérni, és feldolgozni. Az egyes elemek mutató része mindig a következő elemet hivatkozza.
- *Kétirányú nyíltvégű dinamikus lista:* A listaelemeknek két mutatójuk van, amelyekkel a listát mindkét irányban elérhetjük. A lista mindkét végén van egy kezdő elem (fej).
- *Cirkuláris lista:* A lista utolsó eleme az elsőre mutat (nem NIL), vagyis a lista körkörösén elérhető.
- *Rendezett, rendezetlen lista:* A rendezés/rendezetlenség az adatelemek rendezettségét/rendezetlenségét jelenti.

Feladat: Egész számok tárolása és karbantartása egyirányú nyíltvégű dinamikus listában. A számok (≥ 0 és ≤ 1000) közöttiek legyenek! Nincs rendezettség.

Pascal kód:

```
Program dinamikusListaPr;
Uses crt;
{Listaelem és a mutató típusának létrehozása.}
Type
  Pelem = ^Telem;      {Előzetes mutató deklarálás a
listaelemre.}
  Telem = Record
    adat: Integer;
    kov: Pelem; {Hivatkozás a következő listaelemre.}
  End;

{Lista feltöltése, mindig a lista végére. szám  $\geq 0$  és
szám  $\leq 1000$ }
Procedure feltolt_vege(Var lista: Pelem);
Var
  uj: Pelem; {Az új elem mutatója.}
  akt: Pelem; {A feldolgozás alatti elem mutatója.}
  szam: Integer; {A számot tartalmazó változó.}
Begin
  {Lista létrehozása.}
  lista:= NIL;
  akt:= lista;      {Az aktuális a lista kezdetére
mutat.}
```

```

write('Szám: ');
readln(szam);
While (szam >= 0) AND (szam <= 1000) Do
  Begin
    {Helyfoglalás az új elemnek.}
    New(uj);
    {Adatrész kitöltése.}
    uj^.adat:= szam;
    {Listaelem elhelyezése a lista végére.}
    If lista = NIL Then {Ha üres a lista.}
      lista := uj
    else
      akt^.kov:= uj;
      akt:= uj;

      {Mivel a lista végre tesszük az új elemet,
ezért a kov mutatója NIL.}
      uj^.kov:= NIL;

      {Új elem bekérése.}
      write('Szám: ');
      readln(szam);
    End;
  End;

{Lista kiírása.}
Procedure kiir_lista(Var lista: Pelem);
  Var
    akt: Pelem;
  Begin
    {Az aktuális mutatót a listára állítjuk.}
    akt:= lista;
    writeln;
    writeln('A lista tartalma:');
    {Listaelemek elérése, amíg a következő mutató nem
NIL értékű.}
    While akt <> NIL Do
      Begin
        write(akt^.adat, ', ');
        {A mutatót egyel előreleptetjük.}
        akt:= akt^.kov;
      End;
    End;

    {A lista által lefoglalt memóriaterület felszabadítása.}
    Procedure torol_lista(Var lista: Pelem);
      Var
        akt, torlendo: Pelem;
      Begin
        akt:= lista;

```

```

        {Addig megyünk, amíg az aktuális nem NIL.}
        while akt <> NIL Do
            Begin
                torlendo:= akt; {A törlendő elemet mutatja.}
                akt:= akt^.kov; {Az aktuálisat tovább
léptetjük.}
                Dispose(torlendo); {Töröljük a Dispose()
eljárással.}
            End;
            {Első elem mutatóját is NIL-re állítjuk.}
            lista:= NIL;
            writeln;
            writeln('Lista törölve!');
        End;

{Elem hozzáadása.}
Procedure elem_felfuz(Var lista: Pelem);
    Var
        akt, uj: Pelem;
        szam: Integer;
    Begin
        {Adat bekérése.}
        writeln; writeln;
        Writeln('Egy elem felvitele a lista végére!');
        write('Szám: ');
        readln(szam);

        {Az új listaelem elkészítése.}
        New(uj); {Memória lefoglalása.}
        uj^.adat:= szam; {adatrész kitöltése.}
        uj^.kov:= NIL; {Mivel az utolsó elem lesz, ez
mindíg NIL.}

        akt:= lista; {Az aktuális elem ráállítása a
listára.}
        While akt^.kov <> NIL Do {Megkeressük az utolsó
elemet.}
            akt:= akt^.kov;
            {Felfűzés.}
            akt^.kov:= uj;
        End;

{Egy elem megkeresése.}
Procedure elem_keres(Var lista: Pelem);
    Var
        akt: Pelem;
        szam: Integer;
    Begin
        writeln; writeln;
        Writeln('Megkeres egy számot a listában.');
```

```

        {Keresendő szám bekérése.}
        write('Keresendő szám: ');
        readln(szam);
        writeln;
        akt:= lista; {Az aktuális inicializálása.}
        {Keresés}
        While (akt <> NIL) AND (akt^.adat <> szam) Do
            akt:= akt^.kov;
        If akt = NIL Then
            writeln('Nincs ilyen szám!')
        Else
            writeln('Van ilyen szám!');
        End;

{Elem törlése.}
Procedure elem_torol(Var lista: Pelem);
    Var
        akt, elozo: Pelem;
        szam: Integer;
    Begin
        writeln;
        writeln('Egy szám törlése a listából!');
        write('Törlendő szám: ');
        readln(szam);
        {Szám megkeresése.}
        akt:= lista;
        elozo:= NIL;
        While (akt <> NIL) AND (akt^.adat <> szam) Do
            Begin
                elozo:= akt;
                akt:= akt^.kov;
            End;
        If akt = NIL Then
            writeln('Nincs benne ilyen szám!')
        Else {Az aktuális a törlendő, az előző a törlendő
        előtti elemen van.}
            Begin
                {Az aktuális elem lekapcsolása a listáról.}
                elozo^.kov:= akt^.kov;
                dispose(akt);
            End;
        End;

{Főprogram változói.}
Var
    lista: Pelem;

{Főprogram}
Begin
    clrscr;

```

```
{Lista létrehozása.}
feltolt_vege(lista);
kiir_lista(lista);

{Karbantartási műveletek.}
{Új elem felvitele.}
elem_felfuz(lista);
kiir_lista(lista);

{Elem keresése.}
elem_keres(lista);

{Elem törlése a listából, ha benne van.}
elem_torol(lista);
kiir_lista(lista);

{Teljes lista törlése.}
torol_lista(lista);

readln;
End.
```

A Dispose(pointer) eljárás felszabadítja a pointer által mutatott memóriaterületet.

Fabejárás (szélességi, mélységi)

Tervezett fejlesztés.

Visszalépéses keresés

Tervezett fejlesztés.

Megoldandó feladatok

Szekvencia, szelekció, iteráció

1. feladat: Határozzuk meg két különböző sugarú gömb térfogatának egymáshoz viszonyított arányát.
2. feladat: Kérjünk be öt számot, és írjuk ki a szorzatukat!
3. feladat: Számoljuk ki három felhasználótól bekért szám átlagát!
4. feladat: Kérjük be a pontos időt (óra, perc, másodperc) és írjuk ki, hogy másodpercben mennyi!
5. feladat: Írassuk ki a nevünket a képernyő közepére! (GotoXY())
6. feladat: Olvassunk be egy egész számot 0 és 255 között, és írjuk ki a karakter megfelelőjét!
7. feladat: Milyen karakter van a kis „a” és a nagy „A” betűk előtt az ASCII kódtáblában? Ne a karakter kódtáblából nézzük meg!
8. feladat: Olvassuk be egy osztálytársunk születési dátumát (év, hó, nap), és írjuk ki jelenleg hány éves!
9. feladat: Olvassunk be egy egész számot, és vizsgáljuk meg, hogy pozitív-e!
10. feladat: Határozzuk meg egy háromszög területét, az oldalak ismeretében! Figyeljünk a szerkeszthetőségre!
11. feladat: Döntsük el, hogy két bekért szám azonos előjelű-e!
12. feladat: Olvassuk be valakinek a születési évét, számoljuk ki hány éves, és írjuk ki, hogy melyik csoportba tartozik! (0..14 – gyerek, 15..24 – ifjú, 25..70 – felnőtt, 71.. – idős)
13. feladat: Kérjünk be két dátumot (év, hónap, nap), és határozzuk meg a közöttük lévő különbséget napokban!
14. feladat: Készítsük el a másodfokú egyenlet (ax^2+bx+c) megoldását! A program minden értékre adjon üzenetet.
15. feladat: Olvassunk be három számot, és írjuk ki a legnagyobbat!
16. feladat: A felhasználó választásától függően számoljuk ki egy háromszög, vagy egy négyszög területét!
17. feladat: Az A, B, C változókat rakjuk sorrendbe a memóriában! A write(A,B,C) eljárás a számokat nagyság szerint csökkenő sorrendben írja ki.
18. feladat: Kérjünk be három számot a felhasználótól és írjuk ki nagyság szerint!

19. feladat: Számoljuk ki, mennyi pénzünk lenne, ha a pénzünket bizonyos ideig a bankban kamatoztatnánk (kamatos kamat)! Az összeget, a futamidőt és a kamatot a felhasználótól kérje be.
20. feladat: Kérjünk be karaktereket a kis 'a' betű leütéséig, majd írjuk ki hány karaktert ütöttünk le!
21. feladat: Írassuk ki 10-szer egymás alá a nevünket!
22. feladat: Írjuk ki az angol ABC nagybetűit egymás mellé!
23. feladat: Írjuk ki az angol ABC nagybetűit visszafelé!
24. feladat: Írjuk ki egy ciklusban a számokat 1-től 10-ig (számonként négy karakter mezőszélességben), majd alá az egyes számok, adott számmal megnövelt értékét! A növekményt a felhasználótól kérjük be (maximum két jegyű).
25. feladat: Írjuk ki a felhasználótól bekért első N szám (0..100) összegét!
26. feladat: Olvassunk be egy karakterláncot, és írjuk ki visszafelé!
27. feladat: Kérjünk be egy karakterláncot, és alakítsuk kisbetűssé!
28. feladat: Írjunk ki a képernyő közepére egy felhasználótól bekért szöveget!
29. feladat: Tikosítsunk egy szöveget a kizáró vagy művelettel, majd alakítsuk vissza!
30. feladat: Olvassunk be két szöveget és állapítsuk meg melyik hosszabb!
31. feladat: Olvassunk be egy nevet, (vezetéknév, keresztnév) és írjuk ki a monogramját!
32. feladat: Olvassunk be két karakterláncot és keressük meg a második előfordulását az elsőben! Ha benne van, piros színnel írjuk ki az előfordulást.
33. feladat: Kérjünk be egy szöveget, és határozzuk meg hány szóból áll!
34. feladat: Írjunk programot, amely bekéri a banki egyenlegünk értékét, és kiírja hány hónap múlva lesz 10000000 forintunk, ha a havi kamat (kamatos kamat) 10%!
35. feladat: Rajzoljunk ki a képernyőre egy csillag karakterekből álló egyenlő szárú háromszöget!
36. feladat: Kockát dobunk 200-szor. Határozzuk meg, hogy az egyes számokat hányszor dobtuk!
37. feladat: Olvassunk be számokat '*' végjelig, és írjuk ki az összegüket! Ha nem számot ütünk be, akkor adjunk egy figyelmeztető hangot. val(karlánc, szám, kód)

38. feladat: Olvassunk be egy mondatot, és cseréljük ki a '@' karakterre az összes szóközt!
39. feladat: Írjunk programot, amely bekér egy egész számot, majd eldönti, hogy az 1-en és önmagán kívül van-e osztója!
40. feladat: Határozzuk meg egy adott kezdőelemű (A_0) és differenciájú (D) számtani sorozat N . elemét, és eddig az elemig az összegét! ($A_n = A_0 + (N - 1) * D$)
41. feladat: Határozzuk meg, hogy egy felhasználótól bekért természetes szám prímszám-e!
42. feladat: Írassuk ki az N -nél kisebb prímszámokat!
43. feladat: Határozzuk meg az N -nél kisebb iker-prímeket! (Olyan prímek, amelyek különbsége 2)

Eljárás, függvény, fájlkezelés

44. feladat: Írjunk egy eljárást, amely 1 és 10 közötti szorzótáblát jelenít meg a képernyőn, táblázatos formában! A számot a felhasználótól kérje be.
45. feladat: Készítsünk oktatóprogramot, amely a négy alpműveletet gyakoroltatja! A négy alpműveletet függvénnyel, és eljárással is írjuk meg.
46. feladat: Készítsünk függvényt, amely a felhasználótól bekért karakterlánc hosszát adja meg!
47. feladat: Készítsünk eljárást, amely átállítja a képernyő háttér- és karakterszínét az eljárás paraméterében megadottakra!
48. feladat: Készítsünk eljárást, amely a paraméterében megkapott három számot rendezi (növekvően, csökkenően), amelyet szintén a paraméterben adunk meg!
49. feladat: Írjunk eljárást, amely egy paraméterben megkapott tízes számrendszerbeli számot átalakít kettes számrendszerbe! Akasztófa módszer mintájára az átalakítás menete is látható legyen.
50. feladat: írjunk függvényt, amely paraméterként megkap egy számot, és kiírja az első tőle nagyobb prímszámot!
51. feladat: Írjunk függvényt, amely egy szóról megvizsgálja, hogy tükörszó-e!
52. feladat: Írjunk függvényt, amely paraméterként megkap egy tetszőleges méretű sztringeket tartalmazó egydimenziós tömböt, és visszaadja egy olyan tömb mutatóját, amelyben a sztringek hosszai vannak!

53. feladat: Írjunk függvényt, amely meghatározza egy számról, hogy prím-e!

Megszámlálás, eldöntés, összegzés, átlag, kiválogatás, vektor, mátrix

54. feladat: Nullázzunk ki egy háromdimenziós tömböt!

55. feladat: Töltsünk fel véletlen értékekkel egy kétdimenziós tömböt!

56. feladat: Töltsünk fel véletlen páros számokkal egy vektort!

57. feladat: Olvassunk be számokat nulla végjelig, majd írjuk ki hány darabot olvastunk be, mennyi az összegük, illetve az átlaguk!

58. feladat: Addig olvassunk be számokat, amíg az átlaguk el nem éri az 50-et! a számok -10 és +10 közöttiek lehetnek. A végén írjuk ki hány darabot olvastunk be, és mennyi volt az összegük!

59. feladat: Töltsünk fel egy mátrixot véletlen értékekkel (0..10000), és számoljuk meg hány páros és páratlan számot tartalmaz!

60. feladat: Készítsünk eljárást, amely eldönti, hogy egy tízelemű véletlen egészeket tartalmazó tömbben van-e hárommal osztható szám!

61. feladat: Készítsünk eljárást, amely két kompatibilis (azonos elemszám, elemtípus) tömbről eldönti, hogy legalább ötven százalékban vannak-e azonos elemeik! Az indexegyezés nem fontos.

62. feladat: Készítsünk eljárást, amely az első paraméterben megadott tömbből, átmásolja a második paraméterben megadott tömbbe, a harmadik paraméterrel osztható elemeket!

63. feladat: Írjunk ki 100-tól minden 6. számot, összesen 10 darabot!

64. feladat: Írjunk ki a képernyőre egy 40 karakter széles és 10 sor magas csillagokból álló téglalapot!

65. feladat: Készítsünk fényreklámot, amelyben a „Helló” szó fut a képernyőn felfelé és lefelé! Billentyűzetről kérjük be az ismétlések számát. A program egy billentyű leütéséig folytatódjon! (gotoXY, delay, delLine)

66. feladat: Írjunk programot, amely billentyűzetről beolvas egy pozitív egész számot (max 100), majd a billentyűzetről beolvas ennyi darab valós számot, és közülük képernyőre írja azokat, amelyek értékének a beolvasott számok átlagától való eltérése az átlag felénél kisebb!

67. feladat: Adott egy 30*10-as mátrix, amelyet feltöltünk véletlen (0..500) értékekkel. Írjuk ki az eredeti, majd a kétszeresére növelt értékeket!

Minimum-, maximum-kiválasztás, keresés, rendezés

68. feladat: Határozzuk meg a legnagyobb és a legkisebb elemét, egy véletlen számokat tartalmazó kétdimenziós tömbnek!
69. feladat: Határozzuk meg a legnagyobb páros számot egy véletlen értékekkel feltöltött háromdimenziós tömbben!
70. feladat: Egy véletlen értékeket (0..1000) tartalmazó mátrixban keressük meg a 200 és 300 közötti kettővel és hárommal is osztható számokat!
71. feladat: Készítsünk eljárást, amely rendez nagyság szerint egy véletlen értékeket tartalmazó vektort!
72. feladat: Készítsünk eljárást, amely rendez egy karaktereket tartalmazó tömböt! (ASCII szerint)
73. feladat: Írjunk olyan rendező eljárást, amely egy 20 elemű egészeket tartalmazó tömböt rendez úgy, hogy a legkisebb elem az 1. helyre, a következő a 20. helyre, a harmadik elem a 2. helyre, majd a következő a 19. helyre, ... kerül!
74. feladat: Kérjük be egy osztály magasság adatait, és határozzuk meg a legnagyobbat!
75. feladat: Készítsünk olyan függvényt, amely két szám rendezettségét vizsgálja logikai visszatérési értékkel! A rendező elv legyen: elől a páros számok növekvően, majd a páratlan számok csökkenően.
76. feladat: Készítsünk eljárást, amely a második paramétereként megadott karakterláncot megkeresi az első paraméterében!
77. feladat: Írjunk függvényt, amely egy paraméterként megkapott rendezetlen vektorban megkeres egy szintén paraméterként megkapott számot, és kiírja az indexét, vagy -1-et, ha nincs benne!
78. feladat: Írjunk függvényt, amely egy paraméterként megkapott rendezett vektorban megkeres egy szintén paraméterként megkapott számot, és kiírja az indexét, vagy -1-et, ha nincs benne!
79. feladat: Készítsünk programot, amely az iskolai sportnap eredményeit tárolja és dolgozza fel! A sportnapon egyéni számok vannak (futás, gerelyhajítás, kislabdadobás, kosárlabdadobás). Vigyük fel az adatokat 5 diákra, és listázzuk ki csökkenő sorrendben az eredményeket számonként, és

összesítésben is. Az összesítésben a sorrendet az határozza meg, hogy ki nyert többször.

80. feladat: Olvassunk be egy osztálynévsort, és tantárgyanként a tanulók félévi osztályzatait. Menüből lehessen kérni a következőket: tanuló adatai (osztályzatai, átlaga), tantárgyankénti osztályátlag, legjobb tanulók tantárgyanként, bukott tanulók!
81. feladat: Tároljuk el egy hónap napi középhőmérsékleti értékeit! Majd menüből engedje meg az adatok módosítását, adjon listát a bevitt adatokról akár nap szerinti sorrendben, akár hőméréskelt szerint növekvően, vagy csökkenően. A naphoz írjuk ki a hőmérsékletet, és a hőmérséklethez a napot.
82. feladat: Írjunk programot, amely a billentyűzetről pozitív egész számokat olvas mindaddig, míg 0-t nem adunk. A program válassza ki a számok közül a legkisebbet és a legnagyobbat (lehetnek azonos értékűek is, de legfeljebb 3-3), írjuk azok értékét a képernyőre, és adja meg, hogy hányadikként olvastuk be őket!

Metszetképzés, unióképzés, különbség, összefuttatás

83. feladat: Adott három tömb: az első tartalmazza az áruk neveit, a második az egységárait, a harmadik a raktári készletet. Készítsünk egy tömböt, amely a raktári áruk cikkenkénti értékeit tartalmazza! Listázzuk ki az árukészlet adatait!
84. feladat: Az osztály ebben a hónapban kirándulást szeretne szervezni. Tároljuk el, hogy ki mikor ér rá, és határozzuk meg azt a napot (napokat), amelyik mindenkinek jó!
85. feladat: Megkérdezzük az osztálytársainkat, hogy milyen ételleket tudnak készíteni. Határozzuk meg azokat az ételleket, amelyiket mindenki el tud készíteni!
86. feladat: Egy osztálykiránduláson a szakácsok felírják, hogy milyen ételleket tudnak készíteni. Határozzuk meg, hogy hány féle ételből lehet választani!
87. feladat: Éva és Peti összeírja, hogy külön-külön milyen városokban voltak kirándulni. Írjuk ki azokat a városokat, ahol csak Éva volt, illetve azokat, amelyeket csak Peti látott!

Rekurzió, dinamikus lista

88. feladat: Írjuk ki rekurzív alprogramot használva $n!$ (n faktoriálisát)!
89. feladat: Írjuk ki az első N természetes szám összegét!
90. feladat: Készítsünk függvényt, amely az R valós szám pozitív egész hatványát adja meg!
91. feladat: Készítsünk függvényt, amely az R valós szám egész hatványát adja meg!
92. feladat: Készítsünk programot, mely egy szöveges állomány sorait sztringlistába szervezi!