Programozási alapismeretek jegyzet

Barhács OktatóKözpont
2002.

Történeti áttekintés

Előzmények

A számítástechnika kezdetén elterjedt (egyeduralkodó) volt a mérnökpult használata, a gép és az ember kommunikációja bináris nyelven zajlott.

A gépi kódú programozás nem tette lehetővé komplex, általános jellegű programok készítését (a programok csak egy megadott feladatra születnek), megjelennek az alacsonyszintű (assembly) nyelvek (a gépi utasításkódokat az angol nyelvi megfelelőik rövidítésével - mnemonikokkal - jelzik, a bináris adatszerkezeteket felváltja a hexadecimális ábrázolásmód).

A hardver fejlődése bonyolultabb programok készítését tette lehetővé, kialakul a "szoftverkrízis": a tesztelt programok az üzemeltetés során javíthatatlan hibákat tartalmaznak (nem volt javítási lehetőség). A hardver gyorsabban fejlődik, mint a szoftver, mivel a termékek gyorsan elavultak, keményen ellenálltak a módosításoknak.

Programozási nyelvek fejlődése

Az '50-es évek elején az IBM-nél megkonstruálják a **Fortrant**. Létrehozásának oka a gépi kód alkalmazásának kényelmetlenné válása, valamint az automatizálás lehetőségének felmerülése. A Fortran tudományos, műszaki nyelv, szövegfeldolgozásra alkalmatlan, viszont alkalmas matematikai műveletek végzésére. Újabb verziói miatt 2002-ben is élő nyelv, a szociológiai, kémiai, fizikai statisztikai számításoknak kb. felét ma is Fortrannal végzik.

Az '50-es évek második felében az amerikai hadügyminisztérium benyújtja kérését az IBM-nek: készítsenek a feltételeiknek megfelelő magas szintű programnyelvet. Megszületik a **Cobol**, amely adatfeldolgozásra, pénzügyi, gazdasági, nyilvántartási feladatokra alkalmas. Innentől az IBM Fortrant és Cobolt szállít a gépeihez.

1960-ban az IBM európai központjában létrehozzák az **Algol60**-t. Ez az első matematikai értelemben definiált nyelv. Hivatkozási nyelve is van, rendezett fogalomrendszerrel rendelkezik. Az IBM nem támogatja, ennek ellenére nagy jelentőségű, konstrukciója ma is hat. Ekkortól kezdve az Algol60 az algoritmus-leíró szabvány, a későbbi nyelvek vagy belőle indulnak ki, vagy tagadják. Az Algol60 tagadására épülő nyelvek: pl. **LISP** (a mesterséges intelligencia támogatására fejlesztették ki), **APL** (eszközrendszere még ma is egyedülálló).

'60-as évek első fele: Mindenki nyelvet gyárt. Több száz nyelv születik, általában az Algol60-ra hivatkoznak. (pl. **BASIC**, de csak a '80-as években válik világméretűvé)

'60-as évek közepe: Az IBM elhatározza, hogy a nyelvek kavalkádjában rendet teremt. Olyan nyelvet akarnak alkotni, amely egyesíti az eddigi nyelvek előnyeit, és operációs rendszer írására is alkalmas. Így jön létre a **PL/1**. A nyelv túl bonyolultra sikerült, sok mindent megpróbáltak belezsúfolni. Új fogalmak (köztük egyedülállóak is) jelentek meg, pl. az alnyelv (subset). Az alnyelvek (A-tól F-ig) eszközrendszere halmazszerűen épül egymásra. Legszűkebb az A halmaz, ez a minimális, az F a teljes eszközrendszert tartalmazza. A szakma a PL/1-t alkalmatlannak nyilvánítja operációs rendszer írására, és az eddigi nyelveket használja fel.

1967: SIMULA67. Egy teljesen eltérő filozófiát vezet be, mert objektumorientált.

1968: ALGOL68. Olyan bonyolultra tervezték, hogy egy ideig fordítóprogramja sem volt. Elméleti jelentőségű.

1971: Wirth, aki más jelentős nyelvek létrehozásában is segédkezett, megkonstruálja a **PASCAL**-t. Standard algoritmus-leíró nyelv, átveszi az Algol60 szerepét. Eszközrendszere szűkös, rengeteg implementációja létezik, rengeteg eltéréssel. A gyakorlatban is jelentős, pl. a közoktatásban az első számú nyelv.

A **PROLOG** Európában, Franciaországban születik meg, a mesterséges intelligencia kutatás támogatására.

'70-es évek közepe: Az amerikai hadügyminisztérium újabb kívánságlistát nyújt be, melynek eredménye az **ADA**. Ez a nyelv saját kategóriájában a csúcs, minden benne van, ami kell. Bonyolult, az elméletben kulcsnyelv, de gyakorlatban is jelentős. (pl. a NATO-ban alkalmazzák)

'70-es évek vége: C nyelv. Az ősnyelvek mellett új szerepet játszik, bizonyos verziói a '80-as évektől kezdve a szakma nyelvének számítanak. Az első magas szintű nyelv, amin operációs rendszert valósítanak meg.

'80-as évek: Jelentős nyelvet nem konstruálnak.

- **SMALLTALK**: A '80-as években fejlődik ki. Az első vizuális kezelőfelületű nyelv.
- **EIFFEL**: Csak az elméleti szerepe jelentős.
- **C++**: Ma alapvető szerepe van.

'90-es évek: JAVA. Ma a Java "a" nyelv.

Programozási nyelvek osztályozási szempontjai[1]

Generációk szerint

Első generáció:
- manuális programozástechnika
- gépi kód, alacsonyszintű nyelvek használata

Második generáció:
- magas szintű nyelvek használata
- strukturált programozástechnika megjelenése

Harmadik generáció:
- objektumorientált megközelítés kialakulása
- objektumorientált nyelvek használata, kiegészítések, "ráépülő" nyelvek kifejlődése

Negyedik generáció (4GL):
- vizuális kezelőfelületű nyelvek megjelenése
- eseményvezéreltség kialakulása
- az objektumorientáltság alapvető követelmény

Működés szerint

Procedurális nyelv, amelyben a programozó adja meg az utasítások végrehajtásának sorrendjét és mikéntjét. Bármely adatszerkezeten bármely nyelvi utasítás végrehajtható a szintaktikai szabályoknak megfelelően.

Objektumorientált nyelv, amelyben az adatokat és a rajtuk végrehajtható utasításokat egy egységként kezeljük. A programozó korlátozva van abban, hogy milyen utasításokat hajthat végre az adatokon.

Eseményvezérelt nyelv, amely már nem lineáris szerkezetű, az utasítás végrehajtása az objektumhoz tartozó eseményhez kötődik (az utasítás végrehajtás során a programozó korlátozva van abban, hogy hogyan hajtathatja végre azt).

Szerkezet szerint

Az imperatív (utasításszerkezetű) nyelvek alapeszközei az utasítások és a változók. A program szövege utasítássorozat, minden utasítás mögött gépi kód áll. Minden program utasítássorozat, amely mögött több gépi utasítás áll. Kötődnek a Neumann-architektúrához, általában fordítóprogramosak. Algoritmikus nyelvek, a programban azt az algoritmust írom le, amelyet a gép végrehajt, és a probléma megoldása így születik meg. A program a hatását a tár egyes területein lévő értékeken fejti ki.

A deklaratív (leírásjellegű) nyelveknél nincs utasításfogalom, a Neumann-architektúrától távol áll. Nem algoritmikusak, a programban csak a problémát fogalmazom meg, a megoldást nem, az algoritmus a rendszerbe van beépítve. A tárhoz a programozónak kevés köze van, nem feladata a tár egyes részeinek módosítása.

[1] ***Melléklet: pralap_I.ppt***

Alapdefiníciók

- **Információ**: értelmezett adat, értesülés, új adat, mely összefüggéseivel együtt kerül be ismereteinkbe. Az információ mértékegysége a **bit**, ami vagy 1 vagy 0 (igaz vagy hamis, magasabb vagy alacsonyabb elektromos feszültségi szint) értéket vehet fel.
- **Adat**: A számítógépben jelsorozat formájában tárolt, kódolt információ. A bennünket körülvevő világ objektumainak (tárgyak, dolgok) mérhető és nem mérhető jellemzői. Adat egy tárgy kilogrammban kifejezett értéke, egy ember neve, a ruha színe. Mindegyik egy tulajdonságot jellemez, de tartalmukat tekintve különbözőek. Az adatok jellemzésének egyik módja, hogy megadjuk milyen értékeket vehetnek fel az adott szituációban, és ezekkel milyen műveleteket lehet elvégezni. Egy adat lehetséges értékeinek halmazát ***típusérték-halmaznak*** nevezzük. Egy adat típusát három dolog határozza meg. Egyrészt azok az értékek, amelyeket az adat felvehet, a ***típusérték-halmaz.*** Másodszor az a ***szerkezet***, ahogyan egy ilyen érték egyszerűbb típusok értékeiből felépül. Harmadszor azoknak a ***műveleteknek*** *az összessége*, amit az adott halmazon el lehet végezni.
- **Parancs**: a számítógép számára adott közvetlen utasítás.
- **Upgrade**: átdolgozott verzió, továbbfejlesztett, bővített funkciókkal.
- **Update:** frissített verzió, hibák javításával, program újabb adatokkal való feltöltése.
- **Szintaktika:** A szöveg összeállítására vonatkozó szabályok összessége.
- **Szemantika:** A program működésére vonatkozó szabályok összessége.
- **Hivatkozási nyelv:** Egy magas szintű nyelvnek definíciója van, ez általában szabvány. A hivatkozási nyelv a szintaktikai és szemantikai szabályokat adja meg, definiálja a nyelvet. Csak egy darab van belőle.
- **Implementáció:** A nyelv adott rendszeren belüli konkrét megvalósítása, általában nem kompatibilis a hivatkozási nyelvvel.
- **Compiler (fordító)**: a magas szintű programozási nyelven írt programot lefordítja a gép számára érthető formára. (Pascal, ADA, Clipper).
- **Interpreter (értelmező)**: egy magas szintű programozási nyelven írt programot értelmez a gép számára. (BASIC, LOGO).
- **Absztrakció**: azon adatok és tulajdonságok kiválasztása, melyek egy feladat végrehajtásához szükségesek
- **Dekompozíció**: a feladat részekre bontása.
- **Alacsony szintű nyelv**: az ember által nehezebben megfogalmazható, nagyobb programozói munkát igénylő, gépközeli programozási nyelv. Az assembly nyelv utasításai 3 részre oszthatóak:
  - **cím**: az a memóriarekesz, amiben az adott utasítás található
  - **utasításkód**: mnemonikok: a nyelv alapszavainak rövid, könnyen megjegyezhető formái
  - **operandus**: művelet elvégzéséhez szükséges adatot vagy címet tartalmazza
- **Assembler**: program, mely az assembly-ben írt programot lefordítja a gép számára érthető formára.

- **Magas szintű nyelv**: a programozó számára könnyebben megfogalmazható, emberközelibb, bővebb utasításkészlettel rendelkező programnyelv. Hordozható, viszonylag gépfüggetlen programok. Sok utasítással rendelkeznek, összetettebb feladatok megvalósítására is képesek.(Pl. Pascal, C, Basic, Delphi, Clipper, LOGO)
- **Forrásprogram**: maga a program egy adott programozási nyelven kódolva.
- **Tárgyprogram**: félig fordított kód (célkód), kisebb helyet foglal, a hardver utasítások gépi kódját végleges formában tartalmazza, de a címek átcímezhetők. Az object könyvtárból másolódnak hozzá programmodulok.
- **Gépi kódú program**: gépi kódú utasítás: a gép számára végrehajtható utasítások sorozata.
- **Szintaktikai hiba**: egy programnyelv azon szabályainak megsértése, amelyek az utasítások és adatok leírására vonatkoznak. Általában elgépelés okozza.
- **Szemantikai hiba**: logikailag értelmetlen művelet, az adatok tartalmi hibájából és/vagy helytelen csoportosításából, hibás összefüggéseiből következő hiba.(tömbön túli indexelés).
- **Programfejlesztés**: a program elkészítésének munkafolyamata a felmerülő problémától a kész, eladható termékig.
- **Analízis**: felmérjük a helyzetet: mire van szükség, mik a lehetőségek?
- **Feladatspecifikáció**: a feladat pontos megfogalmazása, az esetleges képernyő- és listatervekkel együtt.
- **Fejlesztői dokumentáció**: a program fejlesztését végigkísérő dokumentációk összessége.
- **Felhasználói dokumentáció**: a program használatával kapcsolatos tudnivalókat tartalmazza.
- **Kódolás**: a forrásprogram elkészítése.
- **Tesztelés**: a hibák felderítése.
- **Szárazteszt**: a programterv gondolatban való ellenőrzése, kipróbálása.
- **Felhasználóbarát**: a program szép, nem idegesítő, teljes mértékben szolgálja a felhasználót.

Az algoritmus

Az algoritmus egy feladat megoldására szolgáló egyértelműen előírt módon és sorrendben végrehajtandó véges tevékenységsorozat, mely véges idő alatt befejeződik. A tevékenység matematikai művelettől kezdve tetszőleges számítási, gyártási vagy technológiai művelet lehet.

Az algoritmusokkal szemben támasztott követelmények

- Lépésekből áll. Végrehajtása lépésenként történik (folyamat).
- Minden lépésnek egyértelműen végrehajthatónak kell lennie.
- Részletezés, dekompozíció.
- A végrehajtás tárgya az adat.
- A végrehajtandó instrukciónak valamilyen célja van.
- Vannak bemenő adatai, melyeket felhasznál.
- Legalább egy kimenő adatot produkálnia kell.
- Véges számú lépésben megoldhatónak kell lennie.
- Legyen hatékony és elronthatatlan!
- Legyen az algoritmus felhasználóbarát!

Az algoritmus állapottere

Az ***előfeltételben*** leírjuk a változók segítségével azt a feltételt, ami a kezdőállapotokat jellemzi, az ***utófeltételben*** pedig a végállapotok jellemzőit.

A feladat ***specifikációja (deklarációja)*** során megadjuk a feladat szempontjából lényeges típusérték-halmazokat a hozzájuk tartozó változókkal együtt, a változók segítségével leírjuk az elő- és utófeltételt.

A program I.

A program előre megadott utasítások logikus sorozata, amely közli a számítógéppel, hogy mit tegyen a betáplált adatokkal az adott feladat elvégzése érdekében. A program = algoritmus + adatszerkezetek (Wirth).

A programfejlesztés lépései

Compileres technika

1. Forrásprogram megírása, szerkesztése egy ***szövegszerkesztővel***
2. Forrásprogram lefordítása a ***fordítóprogram*** segítségével eredménye a tárgykód
3. Gépi kódú futtatható állomány létrehozása a ***kapcsolatszerkesztő*** segítségével
4. Program futtatása a ***betöltő*** segítségével, hibakeresés

Ahhoz, hogy a számítógépet céljainknak megfelelően tudjuk használni, programot kell írnunk. Ehhez általában valamilyen szövegszerkesztőt használunk. A megírt programszöveget ***forrásprogramnak*** nevezzük. Ahhoz, hogy a programot megértessük a géppel, a forrásprogramot le kell fordítani a gép számára érthető formára. A ***fordítóprogram*** a forrásprogramot ún. ***tárgyprogrammá*** vagy ***tárgykóddá*** alakítja át, amely már gépi kódú program. Szükség lehet egyéb programrészekkel való szerkesztésre, illetve el kell látni a futtatáshoz szükséges információkkal. A szintaktikai hibák a fordításnál, a szemantikai hibák a program futása alatt (vagy egyáltalán nem) derülnek ki. Csak szintaktikailag helyes szövegnek van tárgykódja. A szerkesztőprogram készíti a futtatható programot, a betöltő feladata a futtatható program betöltése a tárba és elindítása.

Interpreteres technika

Néhány nyelvhez használható ***interpreter*** (értelmezőprogram). Ez nem állít elő tárgykódot, hanem magát a forrásprogramot hajtja végre utasításonként. Nem képződik tárgykód, nincs szükség kapcsolatszerkesztésre, de minden futtatáskor újra kell értelmezni a forrásprogramot, utasításonként. A forrásszöveget elemei szerint értelmezi, az értelmezés után rögtön az eredményt szolgáltatja. Hátránya, hogy tárgykód hiányában állandóan kiértékel (pl. ciklusoknál), és előfordulhat, hogy az esetleges hibák helyére a tesztelés ideje alatt egyáltalán nem kerül a vezérlés (pl. elágazásoknál), tehát egyes szintaktikai hibák rejtve maradhatnak.
Jellemzői:

- Az értelmezőnek mindig benn kell lennie a memóriában >> rossz memória-kihasználás.
- A programokat utasításonként értelmezi és hajtja végre >> lassú végrehajtás.
- Párbeszédes munka >> könnyebb programfejlesztés.

Programmal szembeni elvárások

- Megszokott alkalmazói felület
- Gyorsaság
- Hatékonyság
- Egyszerűség
- Olcsóság
- Rugalmasság
- Korlátlan továbbfejleszthetőség
- Könnyű kezelhetőség
- Érthetőség
- Biztonság
- Tudja a felhasználó, hogy épp mit csinál a gép

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Mikorra tehető az első magas szintű programnyelvek létrejötte?
   a., 1940-es évek vége
   b., 1950-es évek
   c., 1962-től
2. A programozási nyelvek hányadik generációja alakította ki az objektumorientált megközelítést?
   a., első
   b., harmadik
   c., negyedik
3. Mi a compiler?
   a., fordító
   b., értelmező
   c., szerkesztő
4. Az assembler
   a., egy nyelv
   b., egy fordító
   c., egy operációs rendszer
5. Mi a hivatkozási nyelv?
   a., magas szintű nyelvek összefoglaló neve
   b., egy magas szintű nyelv szintaktikai és szemantikai szabályainak leírása
   c., a nyelv egy adott rendszeren belüli megvalósítása
6. Hány generációját ismerjük a programozási nyelveknek?
   a., három
   b., négy
   c., öt
7. Mi a tárgykód?
   a., szerkesztés előtti bináris kód
   b., a leírt programszöveg
   c., a futtatható program

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A FORTRANT az IBM dolgozta ki.
   igaz
   hamis
2. A második generációs nyelvek magas szintűek.
   igaz
   hamis
3. Az eseményvezéreltség a programozási nyelvek harmadik generációjában jelenik meg.
   igaz
   hamis
4. Az imperatív szerkezetű nyelvek nem algoritmikusak.
   igaz
   hamis
5. A szintaktika a szöveg összeállítására vonatkozó szabályok összessége.
   igaz
   hamis
6. Az implementáció azon adatok és tulajdonságok kiválasztása, melyek egy feladat végrehajtásához szükségesek.
   igaz
   hamis
7. Az algoritmusok mindegyike véges időn belül befejeződik.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mi a mnemonik?
2. Mi a jelentősége az ALGOL programozási nyelvnek?
3. Melyik volt az első objektumorientált nyelv?
4. Milyen programozási nyelv generációkat ismer?
5. Hogyan osztályozná a nyelveket működés szerint?
6. Mi a különbség az imperatív és a deklaratív nyelvek között?
7. Mi az adat?
8. Mitől tekintünk egy nyelvet magas szintűnek?
9. Mi az algoritmus?
10. Mik a programfejlesztés lépései compileres technika esetén?

A program II.

A program életútja[2]

A program elkészítése egy gyártási folyamat, melynek több fázisát különböztetjük meg. A fázisokon való végighaladást a program életútjának szokás nevezni.

Feladatmegfogalmazás

A program megrendelője megfogalmazza elvárásait, igényét köznapi vagy a megrendelő szakmai nyelvén. Jobb helyeken a megrendelővel a rendszerszervező tárgyal, akinek feladata a megrendelő szakmájának olyan szintű ismerete, hogy eldönthesse a feladatról, hogy az mennyire számítógépesíthető, milyen hardverigényű, létezik-e a probléma megoldására már kész szoftver. Továbbá milyen átalakításokat igényel a hagyományos folyamat a gépesítéshez. A rendszerszervező feladata az is, hogy közelítőleg megbecsüli a szoftver elkészítési idejét és költségét.
A feladat megfogalmazása legyen pontos, egyértelmű, teljes; rövid, tömör formalizált; szemléletes, érthető, tagolt formájú. Egyértelműen és pontosan azt a feladatot és úgy oldja meg, ahogy azt a feladat kitűzője elvárja.

Specifikáció, algoritmustervezés

A rendszerszervező, a programozó és a megrendelő pontosan meghatározzák az alkalmazás feltételeit és határait, a lehetséges továbbfejlesztési irányokat. A feladatot a lehető legpontosabban leírják, ez lesz a programozó iránytűje a továbbiakban. Az elkészült specifikáció a szakmai nyelv mellett tartalmazza a programozói szóhasználatot is a megrendelő, felhasználó számára is érthető és ellenőrizhető formában. A program legfőbb részeit valamilyen módon meg kell terveznünk. E terv leírása különösen fontos a későbbi teszteléshez, a dokumentáláshoz. A program tervezője ekkor állítja össze a menüszerkezetet, elkészíti a képernyőterveket, meghatározza az adatábrázolási módokat. Különböző algoritmus-leíró eszközöket használhat a programozó (mondatszerű leírás, folyamatábra, struktogram, struktúra diagram).

[2] ***Melléklet: pralap_II.ppt***

Kódolás

A megtervezett program megvalósítása valamilyen programozási nyelv felhasználásával. A programnyelv kiválasztásakor figyelembe kell venni a programozási feladat jellegét, a már rendelkezésre álló programrészleteket, és a programot futtató rendszer jellemzőit. Ha a kódolás során elakadunk, akkor a változtatásokat, új megoldási utakat az algoritmustervbe is fel kell vennünk. Ha nagyobb problémával kerülünk szembe, akkor akár a specifikációt is meg kell változtatnunk, és újra kell terveznünk algoritmusaink egy részét. A kódolás folyamata alatt újabb tesztadatokat gyűjthetünk össze. A későbbi könnyebb módosítás és továbbfejlesztés érdekében célszerű megjegyzéseket is tenni a kódba.

Tesztelés

A program nyelvi helyességét általában az adott nyelv végzi. A szélső és extrém, kifejezetten abnormális adatbevitelek, normális esetek vizsgálata nagyon fontos! Teszteléskor a cél az, hogy minél több hibát felderítsünk. Az első lépés, hogy megvizsgáljuk a program szövegét, tényleg azt írtuk-e le, amit akartunk. Következő lépés: működés közben vizsgáljuk a programot. Különböző bemenő adatokra a megfelelő kimenetet adja-e? Fontos, hogy minden lényeges esetet megvizsgáljunk. Pl. Ha egy számról el kell dönteni, hogy prímszám-e, akkor próbáljuk ki egy prímszámra, egy nem prímre, az 1-re és a 2-re is. Olyan bemenő adatot is meg kell vizsgálni, ami a feladat szempontjából határesetnek minősül (pl. nullával való osztás). Vizsgálni kell azt is, hogy a program az előforduló hibalehetőségeket hogyan kezeli (rossz típusú adat esetén mi történik).

Programozási típushibák

- Gépelési hiba
- Azonosító hibás megadása
- Vezérlésátadási hiba (ciklusoknál rossz feltétel, case szerkezet hibás, eljárások függvények hívás, elágazás szervezési hibák, utasítások maradnak ki)
- Végtelen ciklus
- Változó hibái: pl.: nem kap kezdőértéket, értékhatáron kívüli értéket kap, változószaporítás stb.
- Inputadat hiba
- Aritmetikai és logikai kifejezések hibái: pl.: zárójelezés, precedencia sorrend figyelmen kívül hagyása, operátorok megkeverése
- Tömbökkel kapcsolatos hibák: pl.: méretezés, túlindexelés
- Eljárásokkal kapcsolatos hibák: pl.: paraméterátadási hiba, változók lokalitásának figyelmen kívül hagyása, értékátadás
- File-kezelési hibák: pl.: nyitás, zárás, kimarad az utolsó rekord, a file-mutatót nem jól kezeljük

"Nincs tökéletes program, csak olyan, amelyiknek még nem találták meg a hibáit!"
(Wirth)

Statikus tesztelési módszer

- szárazteszt: a program számítógép nélküli formális ellenőrzése (kódellenőrzés, egyeztetés az algoritmussal, szemantikai vagy logikai ellenőrzés: nem követtünk-e el logikai hibát; a szintaktikai ellenőrzés fordításkor kiderül)

Dinamikus tesztelés

- fekete doboz tesztelés: logikailag ellenőrizzük a programot a program input-output adatain keresztül a forrásprogram ismerete nélkül (csak specifikáció alapján - nem ismert a működés)
- az adatokat olyan csoportok szerint vizsgáljuk, ahol a csoport minden tagja egyformán viselkedik (ekvivalens állítások)
- határesetek vizsgálata
- fehér doboz tesztelés: a forrásprogram tételes ellenőrzése különböző szempontok, stratégiák szerint
- kipróbálási stratégiát készítünk (globális változók nyomkövetése, eljárások tesztelésének sorrendje, paraméterátadás figyelése)
- tesztadatok generálása
- útvonalak tesztelése: minden útvonalat be kell járni a programban (elágazásnál !)
- csomópont tesztelés: a program minden csomópontján a logikai kifejezések minden részkifejezése vegye fel az összes lehetséges értéket

Speciális tesztek

- a konkrét esetvizsgálat nem a program helyességéről győz meg, hanem konkrét szituációt elemez
- a funkcióteszt azt vizsgálja, hogy az adott funkciót megoldja-e a program
- biztonsági teszt: a program biztonságos működését vizsgálja, I/O hibákat figyeli, ellenőrzi

Hibakeresési eszközök, tesztelési eszközök

- modulvégrehajtás: a program egészétől függetlenül a paraméterek változtatásával hajtjuk végre
- kiíratjuk a változók értékét
- nyomkövetés
  - utasításonként
  - programsoronként
- logikai egységenként, modulonként
- változó figyelés
  - folyamatos
  - adott feltételtől, szituációtól függő
- töréspont elhelyezése
- állapotellenőrzés a program futása közben az adatok megjelenítésére részfeltételeket lehet hárítani (pl. rendezés)

Hibakeresés, javítás

Program szövegének vizsgálata. Előfordul, hogy egy hiba kijavítása után több hiba keletkezik. A hibakeresés a teszteléssel szorosan összekapcsolódik. A korszerűbb fejlesztőrendszerekben különböző hibakereső eszközök is rendelkezésre állnak:

- az egyes változókat nyomon követhetjük a futás során, láthatjuk az aktuális értéküket
- lépésenkénti végrehajtás: a programot a program szövege alapján utasításonként végrehajthatjuk
- töréspontok elhelyezése: a program futása a programban elhelyezett töréspontig tart, majd innen folytathatjuk a végrehajtást

Hatékonyságvizsgálat

Az elkészült programot vizsgálni kell a futási idő és a tárfelhasználás szempontjából, ezeknek a minimalizálására kell törekedni.

Dokumentálás

A programot érdemes magyarázó megjegyzésekkel ellátni a későbbi fejlesztési munkák megkönnyítéséhez, illetve működésének megértéséhez.

Üzembehelyezés, karbantartás

A felhasználó megismeri a végleges verziót. A programozó kiképezi a felhasználót a program működtetésére. A felhasználó ellenőrizheti, hogy a program a kívánt feladatokat oldja-e meg. Valódi üzemi körülmények között megtörténik egy átfogó tesztelés. Amennyiben a program használata során rendellenességek tapasztalhatók, akkor a fejlesztőnek azt kötelessége a garanciális időn belül kijavítani.

A dokumentáció

Felhasználói kézikönyv (User Guide)

A dokumentációk közül talán a legfontosabb. Részletesen tartalmaznia kell a program telepítését, indítását, használatát. Ki kell terjednie a program által elvégezhető összes funkcióra. Rendszerint képernyőmintákkal, példákkal illusztrált. Tartalmazza az esetleges hibaüzeneteket és a kapcsolódó hibák elkerülésének, kijavításának lehetőségeit. A dokumentációnak olyan részletesnek kell lennie, hogy a legkevesebb hozzáértéssel is használni lehessen, hiszen ez nem szakembereknek készül elsősorban. Általában kinyomtatott formában megvásárolható, ill. mai követelményeknek megfelelően HTML vagy PDF formátumban olvashatóak.

Tartalma:

- futtatáshoz szükséges géptípus, konfiguráció, hardverkörnyezet
- a futáshoz szükséges szoftverek, operációs rendszer, szoftverkörnyezet
- program közérthető nyelvű specifikációja
- kész program fontosabb paraméterei
- speciális elnevezések
- telepítés, installálás menete
- program használatának részletes leírása
- mintahasználat
- hibajelzések leírása
- hibák kezelésének leírása
- program képességei, alkalmazási köre
- menürendszer leírása
- képernyőképek

Fejlesztői kézikönyv (Programming Guide)

Ha egy program úgy kerül eladásra, hogy annak fejlesztési jogát is megvásárolják, akkor annak elengedhetetlenül tartalmaznia kell egy olyan dokumentációt, melynek birtokában egy másik programozó szükség esetén elvégezheti a módosításokat, hibajavításokat.

Tartalma:

- algoritmus részletes leírása
- programban szereplő adatok leírása, változótáblák
- program fejlesztési lehetőségei
- program teljes listája
- futás időeredmények

Operátori kézikönyv (Installing Guide)

Egyfelhasználós környezetben általában a felhasználói kézikönyv tartalmazza az üzemeltetési, operátori teendőket, mivel ott az nem válik szét. Többfelhasználós környezetben azonban az operátori teendőket leíró dokumentációnak tartalmaznia kell a program telepítésének menetét, szükséges konfigurációját, beállításokat. Leírja a program indítását, paraméterezését. Előírja a szükséges mentések idejét, módját menetét. Az előforduló hibaüzenetek részletes leírását, valamint a hibák megszüntetésének módját.

Programtervezési módszerek

Frontális feladatmegoldás

A feladatot egy egységként kezeli, egyszerre akarja megoldani minden előzetes felmérés, részekre bontás, átgondolás nélkül. A kész programnak nincs szerkezete, áttekinthetetlen, módosítani, javítani, fejleszteni szinte lehetetlen, nincs dokumentációja, használata bizonytalan, teljes anarchia. Kis programoknál használható jól.

Felülről lefelé (top-down) programozás

Legfontosabb megvalósítási elve a lépésenkénti finomítás elve: a feladat megoldását először csak átfogóan végezzük el. Ezután az egészet rész-feladatokra bontjuk, és a továbbiakban ezeket a részfeladatokat oldjuk meg. Így az egyes részeket egymástól függetlenül (de illesztve egymáshoz) írhatjuk, tesztelhetjük, javíthatjuk. Az egyes részeket tovább kell finomítani, amíg elemi részfeladatokig nem jutunk.
Lépések:
- bemenő adatok meghatározása
- szükséges műveletek elvégzése
- eredmények megjelenítése, kiírása, rögzítése

Alulról felfelé (bottom-up) programozás

Nagy előrelátás, gyakorlat szükséges hozzá. Először a legelemibb részeket és ezek algoritmusát készítjük el, utána ezeknek a segítségével bonyolultabb rész-feladatokat oldunk meg.

Párhuzamos finomítás elve

A finomítást az adott szint minden részfeladatára végezzük el: finomítjuk az adatokat, az egyes részfeladatokat, meghatározzuk az adatok kapcsolatait, és ezeket rögzítjük is.

Döntések elhalasztásának elve

Egyszerre csak kevés dologról, de azokról következetesen döntünk. A felbontás után keletkező részek lehetőleg egyenlő súlyúak legyenek! Azokat a döntéseket, amelyek az adott szinten nagyon bonyolultnak látszanak, próbáljuk későbbre halasztani, elképzelhető, hogy később egyszerűen megoldhatók lesznek.

Vissza az ősökhöz elv

Ha zsákutcába jutottunk, nem elegendő az adott szint újbóli végiggondolása, vissza kell lépnünk az előző szintre, és azt végiggondolnunk, stb.

Nyílt rendszerű felépítés

Általánosan fogalmazzuk meg a feladatot, általános algoritmust és programot készítsünk, így az szélesebb körben, hosszú ideig alkalmazható lesz. A konkrét feladat megfogalmazását kell általánosítani, az adatait pedig paraméterként kezelni.

Döntések kimondásának (dokumentálásának) elve

A kimondott, de nem leírt döntések rengeteg bajt okozhatnak. Általában arról feledkezünk meg, hogy egy adat nem lehet nulla. Az algoritmus készítésénél ez még nyilvánvaló, de ha nem mondjuk ki, nem rögzítjük, akkor a kódolásnál már el is felejtjük ezt ellenőrizni!

Adatok elszigetelésének elve

Az egyes programrészekhez tartozó adatokat ki kell jelölni, és szigorúan el kell különíteni más programrészektől. Beszélhetünk globális adatokról, melyekhez a program összes részegysége hozzáférhet, módosíthat; illetve lokális (helyi) adatokról, amelyeket nem minden programrész módosíthat, érhet el. A bemenő adatokat input adatoknak, a kimeneti adatokat output adatoknak nevezzük.

Moduláris programozás

A modul önálló, névvel rendelkező, önmagában értelmezhető programegység. Önállóan tervezhető, kódolható, tesztelhető. A programot nagyobb egységekre, modulokra bontjuk, a kész program a modulok összeillesztéséből jön létre. Ezáltal áttekinthetővé válik a program. Meg kell határozni, hogy az adott modul milyen módon tart kapcsolatot a többi modullal, milyen bemenő és kimenő adatai vannak. (eljárások, függvények, unitok).

A modulok fajtái

- adatmodulok
- eljárásmodulok
- vezérlőmodulok
- I/O modulok

Strukturált programozás

Felülről lefelé elv. Bármilyen algoritmus felépíthető elágazásokból, ciklusokból és részlépések egymásutánjából (szekvenciákból).

A strukturált programozás lényege

- Felülről lefelé történő lépésenkénti problémamegoldás.
- Minden szinten csak a közvetlenül odatartozó döntések folyamata.
- Kevés, de jól meghatározott vezérlési és adatszerkezeti elemek használata.
- Tiltott a feltételes vagy feltétel nélküli vezérlésátadás (GOTO).

A strukturált programozás elveinek meghatározása

- A feladat meghatározását modellsorozattal végezzük.
- A modell olyan program, amely a célprogram működését egy magasabb intelligenciájú " absztrakt " gépen szimulálja.
- Az utolsó változat utasításai egybeesnek egy létező gép vagy nyelv utasításkészletével. A modell elemei hierarchikus viszonyban állnak egymással.
- Alapelv a lépésenkénti finomítás elve, amely az egyes absztrakciós szintek kialakítására ad módszert.
- A megoldandó feladatokat több, kevésbé összetett részfeladatra bontjuk, de a rész-feladatok megfelelő illesztése adja az összprogramot; egyes részei általánosak, de jól körülhatárolt, önálló feladatokat oldanak meg.

Objektumorientált programozás

Az objektumorientált megközelítés az objektumok mint programegységek bevezetésével megszünteti a program kódjának és adatainak szétválasztását. Objektumok használatával egyszerű szerkezetű, jól kézben tartható programok készíthetők. Az objektumorientált programozás középpontjában az egymással kapcsolatban álló programegységek hierarchiájának megtervezése áll.

Az objektumosztályok felépítése

Az adatokat és az adatokon végrehajtható műveleteket egyenrangúan, zárt egységben kezeljük. Ezeket az egységeket objektumoknak nevezzük, mely nem más, mint az adatok és az adatokat kezelő alprogramok (metódusok) egységbezárása. Az objektum felhasználói típusként ***(class, osztály)*** jelenik meg, mellyel változókat, ***objektumpéldányokat (instance)*** hozhatunk létre. Maga az osztály nem más, mint egy ún. ősobjektum, mely az adatmezőiben nem kötelezően tartalmaz adatokat, de az adattípus definíciókat már igen.

Az objektumok lehetnek önálló (nem származtatott) és statikus helyfoglalású objektumok is, a class típus példányai azonban dinamikusan jönnek létre és minden új típusnak van elődje. Az ***öröklődés*** az jelenti, hogy már meglévő osztályból kiindulva újabb osztályokat építhetünk fel, amelyek öröklik a felhasznált osztály minden tulajdonságait. Az objektum tulajdonságait és metódusait is lehet örökíteni. Minden osztály ***adatmezőket***, ***metódusokat*** és ***jellemzőket*** tartalmaz.

Az adatmezők olyan adatelemek, amelyek az osztály minden objektumpéldányában megtalálhatóak. A metódusok az objektumon elvégzendő műveleteket definiáló eljárások és függvények. A ***konstruktor*** egy olyan metódus, mely segítségével megadhatjuk az objektum létrehozásával és inicializálásával kapcsolatos műveleteket. A ***destruktor*** egy olyan metódus, mellyel az objektum megszüntetésével kapcsolatos műveleteket gyűjthetjük egy csoportba. Meghívásakor a destruktor felszabadítja az objektumpéldány számára dinamikusan lefoglalt memóriaterületet. Azokat a metódusokat, amelyek az objektumpéldány helyett magán az osztályon fejtik ki hatásukat osztálymetódusnak nevezzük.

Az objektumorientáltság három fő ismérve

- ***Egységbezárás (encapsulation)***
  - Azt takarja, hogy az adatstruktúrákat és az adott struktúrájú adatokat kezelő függvényeket (metódusokat) egy egységként kezelve, az alapelemeket elzárjuk a világ elől. Az így kapott egységek az objektumok.
- ***Öröklődés (inheritance)***
  - Azt jelenti, hogy az adott meglévő osztályokból levezetett újabb osztályok öröklik a definiálásukhoz használt alaposztályok már létező adatstruktúráit és metódusait.
- ***Többrétűség (polimorfizmus)***
  - Azt értjük ezalatt, hogy egy örökölt metódus az adott objektumpéldányban felüldefiniálódik.

Az adatrejtés elve az objektumorientált programozásban

Az objektum adatmezői és metódusai alaphelyzetben korlátozás nélkül elérhetőek. Az objektum adatmezőit csak metódusok felhasználásával érjük el. Kulcsszavak (pl. public és private) segítségével kijelölhetjük az objektum belső és kívülről is elérhető részeit. A védetten deklarált része az objektumnak belső elérésű a külvilág számára. A publikált részben elhelyezkedő adatmezőkhöz és jellemzőkhöz futási idejű típus-információkat kapcsol a rendszer. Ezzel a megoldással ismeretlen típusú osztály adatmezői és jellemzői is elérhetők.

Objektumok közti kapcsolat

- semmi kapcsolat nincs a két objektum között.
- ***o1 IS A o2***: vagyis az egyik objektumtípus leszármazottja a másik objektumtípusnak, tehát örökli annak tulajdonságait és metódusait, illetve ezeket bővítheti is.
- ***o1 HAS A o2:*** vagyis a két objektum között birtokos viszony van.

A polimorfizmus megvalósítása

A metódusok átdefiniálása a virtuális metódusok segítségével történik Ez valójában azt jelenti, hogy azonos hivatkozás esetén más-más művelet (metódus) kerül végrehajtásra. A program futása közben dől el, hogy végül is melyik metódust kell aktivizálni. Ezt a jelenséget ***késői kötésnek (late binding)*** nevezzük.

Adatszerkezetek

Matematikai

Természetes számok: a számlálás útján nyert számok, mely sorszámnévként is használhatóak, {0,1,2,3…} jele **N.**

A természetes számok halmazán értelmezhető az összeadás és a szorzás. Ahhoz azonban, hogy az összeadás fordított (inverz) műveletét, a kivonást is elvégezhessük, ezt a halmazt ki kell bővítenünk a negatív egész számok halmazával.

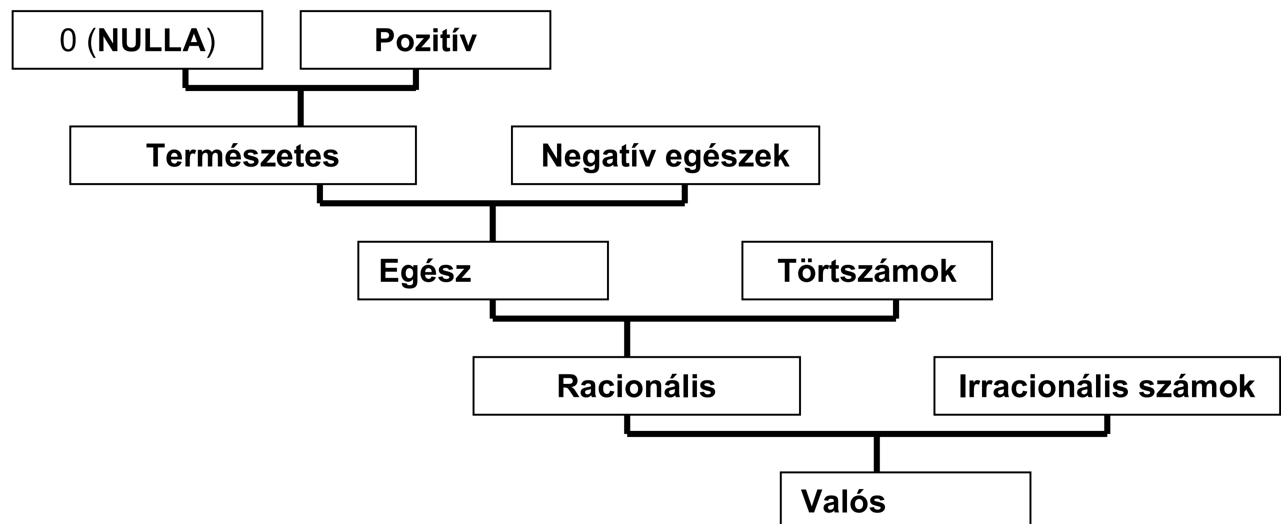
Egész számok: a pozitív, a negatív egész számok és a nulla, {…-1,0,1,…} jele **Z**.

Az egész számok halmazán a kivonás is értelmezhető minden esetben. Ahhoz azonban, hogy a szorzás inverze az osztás is elvégezhető legyen további bővítésre van szükségünk.

Racionális számok: véges vagy szakaszos tizedes törtek, minden olyan szám, mely felírható két egész szám hányadosaként, {m/n ahol $m \in Z, n \in Z$, n<>0) jele **Q**.

A hatványozás inverzének, a gyökvonásnak elvégzéséhez tovább kell bővíteni a számokat, ugyanis léteznek olyan számok melyek végtelen tizedes törtalakúak, mivel nem írhatóak fel két egész szám hányadosaként (pl. $\sqrt{2}$). Ezek az **irracionális számok**.

Valós számok: a racionális és az irracionális számokat együttesen valós számoknak nevezzük, jelezve ezzel, hogy ezen számtípussal minden előforduló számot fel lehet írni, jele **R**.



Programozási

Egyszerű adattípusok

Az adott szituációban nincs értelme további részekre bontani, részeit nem tudjuk külön kezelni.

Numerikus

Egész			
Byte	0 - 255	előjel nélkül	8 bites
Shortint	-128 - 127	előjeles	8 bites
Word	0 - 65536	előjel nélkül	16 bites
Integer	-32768 - 32767	előjeles	16 bites
Longint	-2147483648 - 2147483647	előjeles	32 bites (-)

Valós			
Real	$-2,9*10^{-39}$ - $1,7*10^{38}$	11-12 számjegy pontosság	48 bites,
Single	$-1,5*10^{-45}$ $-3,4*10^{38}$*	7-8 számjegy pontosság	32 bites,
Double	$-5*10^{-324}$ - $1,7*10^{308}$*	15-16 számjegy pontosság	32 bites,
Extended	$-3,4*10^{-4932}$ - $1,1*10^{4932}$*	19-20 számjegy pontosság	80 bites,
Comp	-2^{-63} + 1..2^{63}-1*	19-20 számjegy pontosság	64 bites,

*csak aritmetikai koprocesszorral

Logikai:

True - False
1 - 0
Igaz - Hamis

Karakter

A karakter típusú változóban egy bármilyen ASCII kódú karaktert tárolhatunk. A karakter típusú változónak adhatunk értékül karakter konstanst (1 hosszú szövegkonstanst), illetve karakter típusú változót is. Tárolása 1 Byte-on történik.

Mutató

A mutató egy olyan adatszerkezet, amely egy memóriaterületre mutat. Egy dinamikus változót a program futása közben hozunk létre, illetve szüntetünk meg. A létrehozott változóra mutatóval hivatkozunk, mely mutató egy címet tartalmaz: a mutatott változó fizikai címét. Programunk bármely részén deklarálhatunk mutató típusú változót, mely értékadás után egy címet fog tartalmazni. Ha p egy mutató típusú változó, akkor a mutatott változó p^ (vagy *p). Kétféle mutató létezik: típusos és típus nélküli. Típusos mutató esetén a mutatott változónak jól meghatározott típusa van, míg egy típus nélküli mutató által mutatott memóriaterületre nem jellemző a típus. Mutatókkal nem csak dinamikus változókra lehet mutatni, hanem bármely memóriaterületre, például egy már létező statikus változóra. A mutatókat értékadással át lehet irányítani egyik memóriacímről a másikra. Mutató típusú változót sem beolvasni, sem pedig kiírni nem lehet.

Összetett adattípusok

Részei külön kezelhetők, a hozzá tartozó adatelemek között valamilyen sorrendi, szerkezeti összefüggés van. Az adatoknak ezen osztályozás szerinti besorolása erősen szituációfüggő.

Az összetett adat elemi adatokból háromféleképpen építhető fel:

- azonos típusú elemek sokasága (iterált vagy sorozat adattípus)
- különböző típusú elemek sokasága, ezek a rekord mezői (direkt szorzat vagy rekord típus)
- különböző típusú részekből állhat, de a részekre osztás is többféleképpen történhet egy feltételtől függően (alternatív vagy unió típus)

Sorozat adattípusok

Halmaz

A halmaz olyan adatszerkezet, melyben egyforma típusú, de különböző, rendezetlen elemek találhatók. A halmaz bármilyen fajtájú, jól meghatározott, egymástól megkülönböztethető dolgok összessége. A halmazban lévő dolgokat a halmaz elemeinek nevezzük. Egy halmazban egy elem csak egyszer szerepel, és az elemeknek nincs sorrendje. Az elemek száma lehet véges vagy végtelen. Halmazt megadhatunk úgy, hogy felsoroljuk az elemeit: H1 = {Pascal, Eiffel, C++, Prolog } és H2 = {1,2,3,4,5,6,7,8,9}.

Sorozat

Nem feltétlenül rögzített az elemek száma. Legismertebb sorozat a számsorozat (intervallum típus) ahol a lépésköz rögzített és amely mindkét irányban végtelen. Természetesen megállapíthatunk zárt intervallumokat is, ilyenkor a sorozat "kvázi-tömbként" viselkedik, annyi eltéréssel, hogy a tömböt explicit módon fel kell tölteni, a számsorozat tagjai viszont adottak. A sorozat nagyon rugalmas, léteznek más típusú sorozatok pl. nevek sorozata.

Tömb
Típusérték-halmaza konstans hosszúságú elemeket tartalmaz, az egyes elemekre indexeléssel lehet hivatkozni. Egy tömb megadásakor meg kell adni az egyes dimenziók irányába eső maximális komponensek számát, tehát a tömb mérete rögzített. Az egydimenziós tömböt vektornak (pl. lottószámok), a kétdimenziós tömböt mátrixnak (órarend, sakktábla) is szokás nevezni. Háromdimenziós tömb pl. az iskola összes órarendje. A tömb bármely elemére hivatkozhatunk úgy, hogy megadjuk az elem sorszámát. A sorszámot a változó neve után szögletes zárójelbe kell tenni. Ezt a sorszámot indexnek, a hivatkozási módszert indexelésnek nevezzük. Arra kell vigyázni, hogy az elemre való hivatkozáskor az indexnek olyan értéke legyen, mely egy létező tömbelemre hivatkozik. Ellenkező esetben a tömbön túli indexelés megállítja a program futását. Léteznek ún. asszociatív vagy "hash" tömbök, ahol az indexet nem a sorszám, hanem egy érték adja, ezek a tömbök kulcs-érték párokból épülnek fel.

String vagy text
Karaktertömb. Két fajtája van a "normál" string azokban a nyelvekben, ahol ez az adattípus külön definiált (Pascal), ilyenkor a hosszát külön tároljuk, ill. a nullvégű string, ami karaktervektor, a végén egy nulla értékű byte-al (C). A karakterlánc típusú változónak illetve konstansnak bármelyik karakterére külön hivatkozhatunk úgy, hogy megadjuk annak sorszámát, ugyanúgy ahogy a normál tömbnél.

Rekord típusok

Rekord
Pl. Tanulók adatai: név, cím, telefonszám, életkor, anyja neve, tanulmányi átlaga.
A rekordban különböző típusú, de összetartozó adatokat tárolunk. Legnagyobb előnye, hogy ezeket az adatcsoportokat egyszerre tudjuk kezelni és mozgatni. A rekord a különböző típusú, de összetartozó adatokat összefogja, azokat egy adatként kezeli. Az ilyen adatcsoportok a memóriában egymás mellett helyezkednek el, és az egész adatcsoportra egy névvel lehet hivatkozni. A rekord adatait mezőknek nevezzük. A rekord típus típusérték-halmaza a felépítésében részt vevő típusok értékhalmazainak direkt szorzata. Egyfajta művelete van: egyenként ki lehet választani az egyes komponenseket.

File
A lemezre vitt adatokat egy másik számítógépre is átvihetjük, ott felhasználhatjuk, módosíthatjuk. A file-okra azért van szükség, mert a memóriában tárolt adatok a program futásának végeztével mindenképpen elvesznek, de ugyanez következhet be valamilyen nem várt esemény (áramszünet, programhiba, számítógép kikapcsolása) hatására is. Az adatok tárolásához szükséges memória nagysága általában többszörösen meghaladja a rendelkezésre álló memória nagyságát. A bevitt adatokra később is szükségünk lehet - pl. az áruházban forgalmazott áruk adatait nem csak a bevitel napján, hanem esetleg évekig használni szeretnénk. Az adatok tárolását biztonsággal kell megoldanunk. Fizikai file-nak nevezzük a másodlagos tárolón (floppy, winchester) elhelyezett adatok önálló névvel ellátott halmazát. Ez az önálló név a lemezen az állományspecifikáció.

Elérését tekintve a file lehet:
- szekvenciális input file - sorozat első elemének olvasása.
- szekvenciális output file - sorozat végére írás.
- direkt file - megengedett a pozícionálás valamely elemére, így lehetővé válik bármely elemének olvasása, felülírása, ill. a file végére írás művelete.

A logikai file egyed-előfordulások önálló névvel ellátott halmaza, mely olyan tulajdonságtípusok előfordulásait tartalmazza, mely egy adott feladat szempontjából lényeges. Például az Étel logikai file konkrét ételeket tartalmazhat. Elképzelhető, hogy egy étterem számítógépen szeretné nyilvántartani ételeinek receptjeit, jellemzőit. A szakács szerint az ételek legfontosabb tulajdonságtípusai többek között a következők: Étel neve, Hozzávalók, Elkészítési idő, Elkészítés pontos leírása, Előállítási ár, Eladási ár, stb.

Szerkezetét tekintve a file lehet:
- **Típusos file:**
  - Direkt szervezésű állomány. A kiírás illetve a beolvasás egysége a komponens. A komponensek egyforma típusúak, mely típus az állományra jellemző. A komponens hosszát a típus határozza meg. A komponensek sorszámozva vannak 0-tól kezdve. Bármelyik elemre hivatkozhatunk közvetlenül a sorszámával, azt beolvashatjuk, illetve kiírathatjuk. A komponens helyét a rendszer a sorszám és a komponens hossza alapján meg tudja határozni. A típusos állományt szekvenciálisan is feldolgozhatjuk a komponensek fizikai sorrendjében, vagyis a sorszámok szerint.
- **Típusnélküli file:**
  - Direkt szervezésű állomány. Abban különbözik a típusos állománytól, hogy itt a komponensek hossza tetszőlegesen megadható, azt nem a típus határozza meg. Általában akkor használjuk, amikor csak az számít, hogy hány byte-ot írunk ki, illetve olvasunk be egyszerre - az adatok típusa érdektelen.
- **Szöveges file**
  - Soros szervezésű állomány. A szöveges állomány sorokból, a sorok karakterekből és egy "sor vége" jelből állnak. A kiírt illetve beolvasott adatok változó hosszúságúak, azok fizikai címét nem lehet sorszám alapján megállapítani. Ezért a szöveges állományban található adatokat nem lehet direkt módon elérni.

<u>Unió típusok</u>

Unió

Pl. ha nő, akkor a szülések száma, ha férfi, akkor a katonaigazolvány száma a nyilvántartott adat. Típusművelete segítségével meg tudjuk kérdezni, hogy egy érték egy elemi típushoz tartozik-e.

Objektum

Hasonlít a rekord adattípushoz, de nem csak a logikailag összefüggő adatokat tároljuk egy szerkezetben, hanem az adatokon elvégezhető műveleteket is. Jellemzője az öröklődés, vagyis hogy a származtatott objektum a szülő objektum adatstruktúráit és eljárásait is megkapja, bár át is definiálhatja azokat.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. A program életútja
   a., a program gyártási folyamata
   b., az elavulásig eltelt idő
   c., az értékelés változásai
2. Mi a tesztelés szerepe?
   a., megmutatni a vevőnek, hogy a program megfelelő
   b., hibakeresés
   c., a lehető legjobb eredmények elérése
3. A dokumentáció részei:
   a., fejlesztői és üzemeltetői kézikönyv
   b., felhasználói és kereskedelmi kézikönyv
   c., fejlesztői, felhasználói és operátori kézikönyv
4. A strukturált programozás
   a., egy top-down módszer
   b., egy bottom-up módszer
   c., egy moduláris programozási technika
5. Melyik nem elemi adattípus?
   a., byte
   b., mutató
   c., szöveg

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. Teszteléskor a cél az, hogy minél több hibát felderítsünk.
   igaz
   hamis
2. A szárazteszt egy statikus tesztelési módszer.
   igaz
   hamis
3. Fejlesztői Kézikönyv minden eladott programhoz jár.
   igaz
   hamis
4. A frontális feladatmegoldás csak kis programoknál alkalmazható jól.
   igaz
   hamis
5. Az objektumorientált programozás egy felülről-lefelé történő dekompozíción alapuló elv.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Ismertesse a kódolás szerepét a program életútjában.
2. Mik a Felhasználói Kézikönyv legfontosabb részei?
3. Mi a jelentősége az Operátori Kézikönyvnek?
4. Mikor kötelező a programhoz Fejlesztői Kézikönyvet is mellékelni?
5. Mi a döntések elhalasztásának elve?
6. Mire jó a nyílt rendszerű felépítés?
7. Mik a modulok fajtái?
8. Mi az objektumorientált programozás lényege?
9. Mi az objektumorientáltság három fő ismérve?
10. Milyen sorozat adattípusokat ismer?

Az algoritmusok alapelemei

Változók

Olyan programozási eszközök, amelynek négy komponense van:

- **Név**
  - Egyedi azonosító, a program szövegében a változó mindig a nevével jelenik meg, ez hordozza a komponenseket.
- **Attribútumok**
  - A változó futás közbeni viselkedését, az általa felvehető értékeket határozzák meg. Az eljárás-orientált nyelvekben a legfontosabb attribútum a típus, nem típusos nyelvekben ilyen komponens nincs, de más attribútum lehetséges. Változóhoz attribútum rendelés deklaráció segítségével történhet.
- **Cím**
  - A tár azon területének a címe, ahol az adott változó értéke elhelyezkedik.
- **Érték**
  - Az adott tárrészen elhelyezkedő bitkombináció. A típus eldönti, hogy hány byte-on, milyen ábrázolási móddal van ábrázolva a változó, és meghatározza az értékhatárokat.

I/O műveletek

Az adat be- és kiviteli műveletek azok az utasítások, amik alapján a program helyzetfüggő információkat kaphat.

Az input-output az az eszközrendszer, amit a programban akkor használunk, ha a perifériákkal akarunk kommunikálni. Hardverfüggő, operációs rendszer-függő, a leginkompatibilisebb része a nyelveknek. A nyelvek egy részében nincs I/O utasítás (pl. Algol60, az implementációra bízza), más nyelvekben van. Az I/O alapja az állomány. A nyelvek gyakran kihagyják, a kommunikációt "a" perifériával képzelik el (implicit/standard állomány). Ezt az állományt a nyelv nem kezeli explicit módon, de a rendszer igen. Nem kell deklarálnom, megnyitnom, összerendelnem fizikai állománnyal, lezárnom. Ilyenkor a program a szabvány bemenetről (alaphelyzetben a billentyűzet) olvas és a szabvány rendszerkimeneti perifériára (alaphelyzetben a monitor) ír. A kimenet és a bemenet átdefiniálható.

Utasítások

Az utasítások a program szövegének azon egységei, amelyeket a fordítóprogramnak elsősorban fel kell ismernie, mert a fordítóprogram ezekkel az utasításokkal dolgozik.

Deklarációs utasítások

A fordítóprogramnak szólnak, a működését befolyásolják, szolgáltatást kérnek, üzemmódot váltanak, információval látják el, amelyet a fordítóprogram felhasznál kód generálásánál. Nem áll mögöttük kód, a fordítóprogram nem fordítja le őket.

Végrehajtható utasítások

Kód áll mögöttük, a fordítóprogram lefordítja, és ezekből generálja a tárgyprogramot.

- **értékadó utasítások:** Szerepük, hogy egy változó értékkomponensét a program futásának bármely pillanatában be tudjuk állítani, változtatni.
- **üres utasítások:** A legtöbb magas szintű nyelvben van, néhány nyelvben elengedhetetlen, bizonyos szituációkban a nyelvek előírják. Külön gépi kódja van, jele vagy van (pl. ;) vagy nincs. Hatására a program nem csinál semmit.
- **elágaztató utasítások**
- **ciklusszervező utasítások**
- **ugró utasítások**
- **hívó utasítások**

Ezek minden eljárás-orientált nyelvben megtalálhatóak. Az elágaztató, ciklusszervező, ugró és hívó utasításokat vezérlő utasításoknak hívjuk, a program vezérlési szerkezetének felírására szolgálnak.

- **input-output utasítások:** Az adatmozgatást vezérlik a perifériák és a tár között valamelyik irányban.
- **egyéb utasítások:** A nyelvek között a legnagyobb eltérés az egyéb utasításoknál van, pl. **Pascal:** csak egy van: WITH (a minősítést segíti (record)), **PL/1:** több egyéb utasítás van, mint más utasítás.

Kifejezések

A kifejezés a programnyelvek szintaktikai egysége, az eddigi fogalmak jelennek meg benne. Már ismert értékek alapján új értéket határozunk meg. Olyan objektum, amelynek két komponense van: érték és típus. Típussal csak a típusos nyelvekben rendelkezik. A kifejezések lehetnek matematikaiak vagy logikaiak. Formálisan operandusokból, operátorokból és kerek zárójelekből áll.

Operandusok: Az értéket képviselik, egy operandus önmagában is kifejezést alkot (a legegyszerűbb kifejezés). Operandus lehet: konstans, nevesített konstans, változó vagy függvényhívás.

Operátorok: Minden nyelv definiálja saját operátorait, néhol a programozó is definiálhat sajátot. Típusai:

Műveleti jelek: -, +, *, / stb.
Relációs operátorok: >,<,>=,<=,<>,=
Logikai operátorok: NOT, AND, OR, XOR stb.

Vezérlési szerkezetek

Az elágaztató, ciklusszervező, ugró-hívó utasításokat együttesen vezérlési szerkezeteknek nevezzük.

Utasítás-végrehajtási sorozat (szekvencia)

Előírt utasítások lineáris végrehajtása a legegyszerűbb vezérlési szerkezet.

Elágazás (szelekció)

Egyágú szelekció: ha igaz a megadott feltétel, akkor a hozzá kapcsolódó tevékenységet végre kell hajtani, egyébként azt ki kell kerülni, és a programot az azt követő közös tevékenységgel kell folytatni.
Kétágú szelekció: ha a kiértékelődés után a kifejezés értéke igaz, akkor a feltétel utáni tevékenység hajtódik végre. Ha az értéke hamis akkor a különben ágban lévő utasításokat hajtja végre. Ezután a program a feltételes utasítás utáni utasításon folytatódik.
Többirányú szelekció: feladata, hogy a program egy adott pontján akárhány tevékenység közül tudjunkk egyet választani. A választás általában egy kifejezés (szelektor) értékei szerint történik, lényeges a kifejezés típusa. A kifejezés kiértékelődik, az értékét a konstanslistához hasonlítja. Ha talál megfelelő ágat, végrehajtja az utasítás(oka)t és kilép az elágazásból. Ha nincs megfelelő ág és van különben ág, a különben ágban lévő utasítást végzi el és kilép, ha nincs különben ág, akkor üres utasítást hajt végre.

Ciklusszervezés (iteráció)

Az algoritmusok vezérlőszerkezetei közé tartozik az **iteráció**, más néven ciklus: egy vagy több utasítás ismételt végrehajtása. Akkor van rá szükség, ha egy adatcsoport valamennyi elemén ugyanazt a műveletet kell elvégezni. Ciklus használatánál a műveletet ciklikusan kell megismételni az összes adattal. A ciklikus műveletek végét valamilyen feltétel határozza meg. A ciklus kezdete előtt állhatnak műveletek, amelyeket csak egyszer kell ugyan végrehajtani, de a ciklushoz kapcsolódnak: a változók értékeinek beállítása. A műveletsorozatot, amelyet ismételten végrehajtunk, **ciklusmag**nak nevezzük. A ciklus folytatásával vagy befejezésével kapcsolatos vizsgálatot **ciklusfeltétel-vizsgálatnak** hívjuk. Az az adat, amelynek értéke meghatározza a ciklus folytatását, vagy befejezését, a **ciklus változója**. A ciklusműveletek addig hajtódnak végre, amíg a **végrehajtási feltétel** teljesül (van olyan ciklus, ahol a fordítottja igaz). Fontos, hogy a feltétel ne teljesüljön mindig, mert akkor a végrehajtások száma végtelen lesz.

A ciklusnak formálisan van:

Fej
Mag
Vég

fej, vég: Külön utasításokkal adjuk meg, az ismétlődésre vonatkozó információt tartalmazzák.
mag: Az ismétlendő tevékenységet írja le.

Típusai:

- **előírt lépésszámú (növekményes) ciklus (FOR)**
  - Az ismétlések száma a **ciklusba való belépés előtt már ismeretes**, vagy kiszámítható. Egy utasítás ismételt végrehajtását írja elő, miközben egy változó monoton növekvő vagy csökkenő értéket vesz fel. A ciklusváltozó az ismétléseket számolja az első kifejezés által megadott értéktől kezdve a második kifejezés által megadott értékig. A ciklusváltozó sorszámozott típusú (de tömb eleme nem lehet), a két kifejezésnek pedig azonos típusúnak kell lennie és kompatibilisnek a ciklusváltozó típusával.
- **feltételes ciklus:**
  - **elöltesztelő ciklus (WHILE)**
    - A feltételvizsgálat a **ciklusmag** végrehajtása **előtt** történik meg. A ciklusmagot mindaddig végre kell hajtani, amíg a végrehajtási feltétel fennáll, ezt belépési feltételnek hívjuk. Ha a feltétel már nem teljesül, akkor a ciklus befejeződik és a vezérlés a ciklus utáni következő utasításra kerül.
  - **hátultesztelő ciklus (REPEAT UNTIL)**
    - A feltételvizsgálat a **ciklusmag után** megy végbe. Azt kell vizsgálni, hogy a kilépés feltétele fennáll-e. Ha nem áll fenn, akkor a ciklust folytatni kell. Ellenkező esetben a ciklus befejeződött.
- **végtelen ciklus**
  - A megszakítási feltétel a törzsben található, ezáltal a vezérlőszerkezet nem tartalmaz közvetlen információt a befejeződés feltételéről.

Ugró utasítások

Ha a címke létezik, a program ott folytatódik. Az ősnyelvekben nem lehetett GOTO nélkül programozni (pl. Fortran, PL/1). A későbbi nyelvek némelyikében van GOTO (hagyománytiszteletből), de lehet nélküle programot írni (pl. Pascal). Van olyan nyelv is, amelyben egyáltalán nincs, pl. JAVA.

Az algoritmus alapjelei, építőelemei

- Fenntartott szavak: tulajdonképpen a nyelv szavai, amelyeket a programozás során használhatunk.
  - Például: begin, end, for, procedure, stb. Az alapszavak összességét felfoghatjuk úgy, mint a nyelv szótárát.
- Szimbólumok. Például: , ; ? :=
- Azonosítók: a programozó munkája során rengeteg azonosítót használ.
  - Például azonosítót rendel a változókhoz, a saját eljárásaihoz, stb.
- Konstansok: a legtöbb programban szükség van konstans értékekre, melyek értéke a program egészében állandó.
  - Pl.: PI 3.14, stb.
- Határoló- vagy elválasztójelek.
  - Például szóköz, =, (,), /, +, , , stb.

Algoritmus leíró eszközök[3]

Folyamatábra

Az algoritmus részlépéseit különböző geometriai szimbólumokkal szemlélteti. Az egyes szerkezeti elemek között nyilakkal jelöljük a végrehajtási sorrendet. Az értékadó utasítás illetve az eljárások téglalapba, az elágazások rombuszba vagy lapos hatszögbe, az adatáramlás paralelogrammába, a vezérlő utasítások körbe kerülnek.

Struktogram

Az algoritmust egy téglalapba írjuk be. Ebbe a téglalapba további téglalapokat illesztünk, és a végrehajtandó utasításokat ezekbe írjuk be. Az egyes szerkezeti elemek jól elkülönülnek, a szekvencia az egymásutánisággal, a szelekció az egymásmellé kerüléssel, az iteráció a visszatérési út kijelölésével ábrázolható. Ezek a szerkezetek egymásba ágyazhatóak.

Pszeudokód

Egy megadott programnyelvhez hasonló, de szintaktikailag szabadabb algoritmus leírás. Mondatszerű elemekkel bővíti a nyelv utasításkészletét.

Funkcionális leírás

Az adott algoritmus funkcióinak és ezek hierarchiájának szöveges leírása.

Jackson ábra

Top-down dekompozíciós diagram, ahol az algoritmus feltételei ún. feltételjegyzékbe, az általa végrehajtott tevékenységek pedig tevékenységjegyzékbe kerülnek.

Mondatszerű leírás

Az algoritmust egymás után következő mondatokkal írjuk le. Ma már a pszeudokód és a mondatszerű leírás összemosódott, a fő különbség mégis az, hogy a mondatszerű leírásban a programszerkezetet magyarázó részek a szövegbe kerülnek, míg a pszeudokódnál a magyarázat a használatos nyelv kommentezési szokásai szerint van megjelölve.

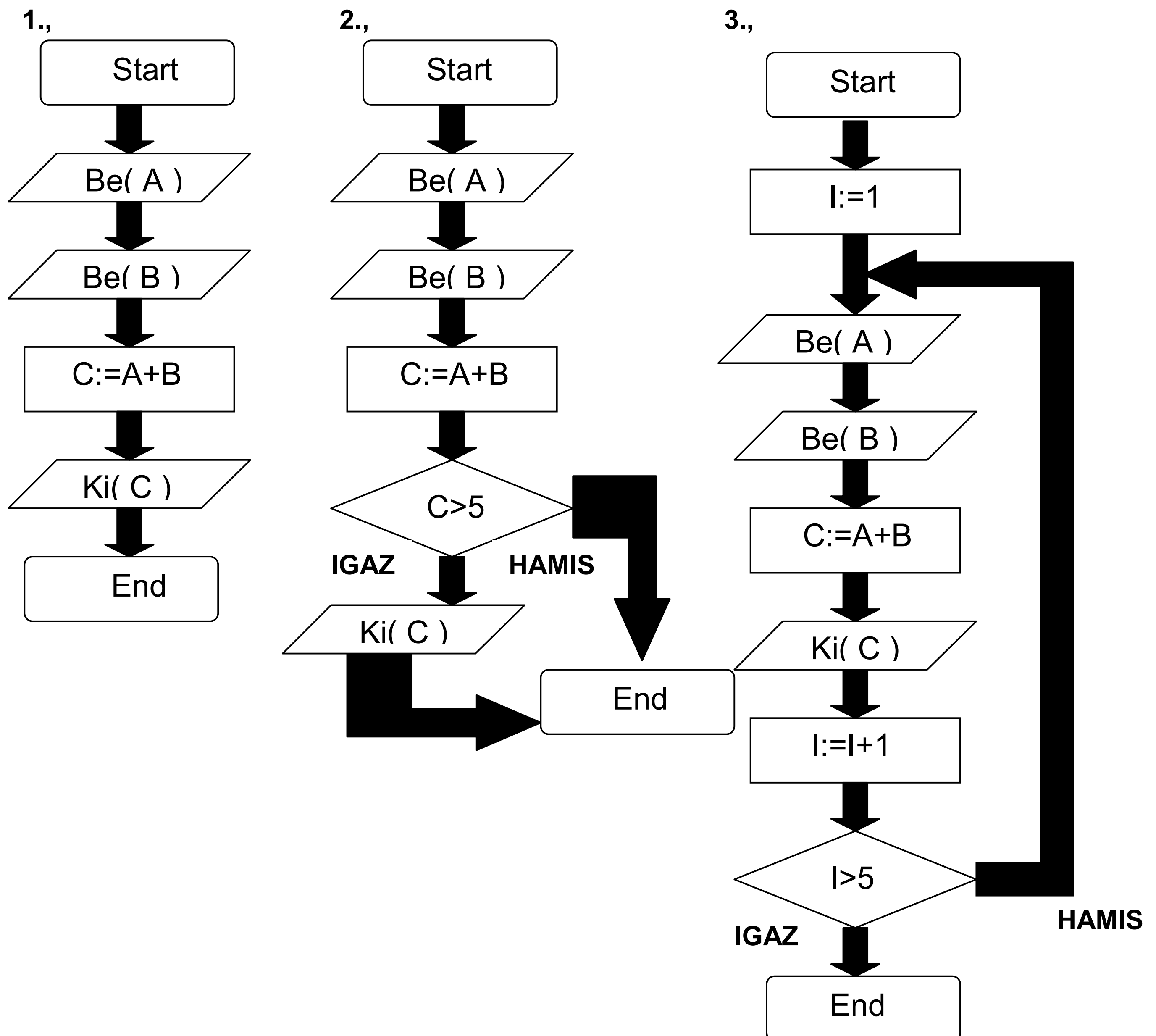
[3] ***Melléklet: pralap_III.ppt***

Példák

Algoritmusleírás
Jelöljük a következő alapelemeket különböző algoritmus-leíró eszközökkel!

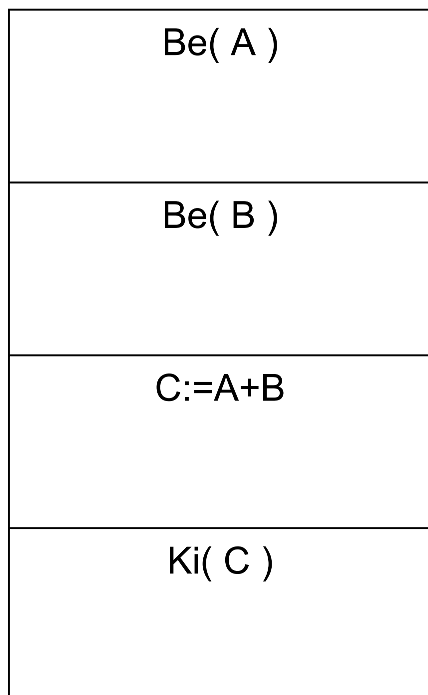
1., Szekvencia
2., Szelekció
3., Iteráció

Folyamatábra

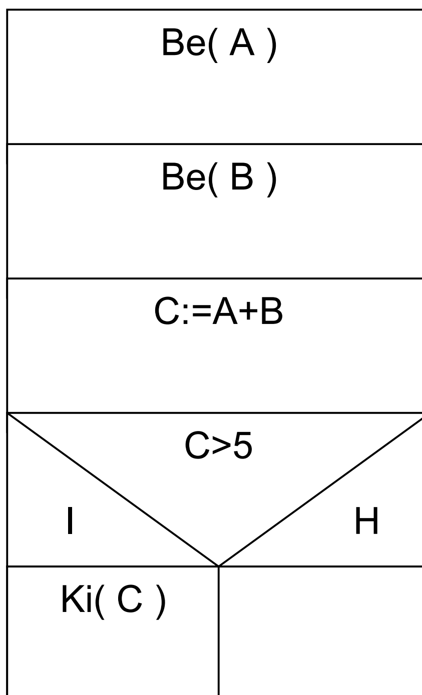


Struktogram

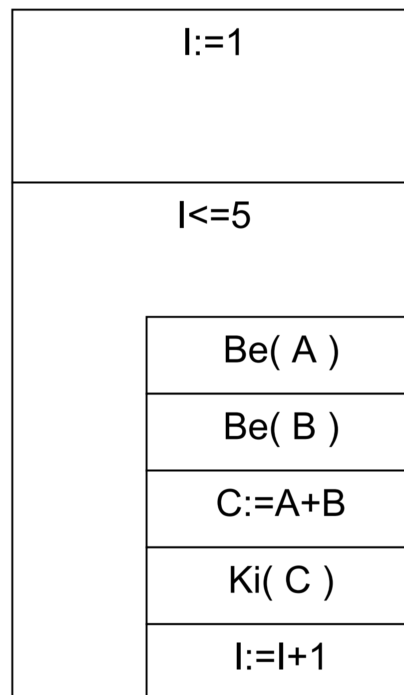
1.,



2.,



3.,



Pszeudokód (Pascal)

1.,

```
Be( A );
Be( B );
C:=A+B;
Ki( C );
```

2.,

```
Be( A );
Be( B );
C:=A+B;
Ha (C>5) akkor
        Ki( C );
Különben
        Semmi;
Elágazás vége
```

3.,

```
Ciklus i:=1-től 5-ig
        Be( A );
        Be( B );
        C:=A+B;
        Ki( C );
Ciklus vége
```

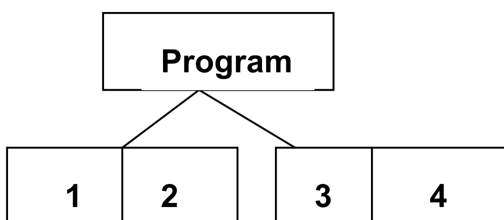
Jackson jelölés

A szekvencia jele

A szelekció jele

Az iteráció jele

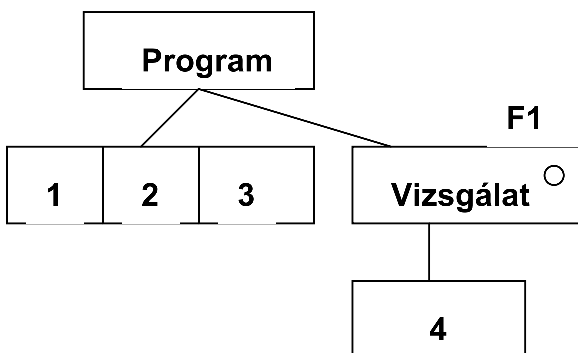
1.



Tevékenységjegyzék:

1. Be(A)
2. Be(B)
3. C:=A+B
4.Ki(C)

2.



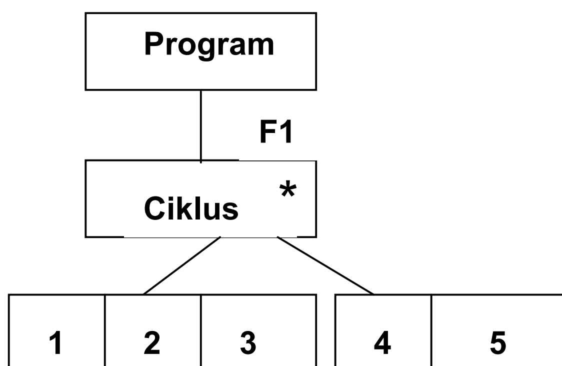
Tevékenységjegyzék:

1. Be(A)
2. Be(B)
3. C:=A+B
4.Ki(C)

Feltételjegyzék:

F1. C > 5

3.



Tevékenységjegyzék:

1. Be(A)
2. Be(B)
3. C:=A+B
4.Ki(C)
5. I:=I+1

Feltételjegyzék:

F1. I <= 5

Ellenőrző kérdések

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mi a változó?
2. Milyen szabványos programozási kifejezéseket ismer?
3. Mutassa be az elágazást (szelekciót)!
4. Mutassa be a ciklust (iterációt)!
5. Milyen algoritmus leíró eszközöket ismer?

Alprogramok

Az újrafelhasználhatóság és a procedurális absztrakció eszköze.

Újrafelhasználhatóság: Ha a program különböző részein ugyanaz az utasítássorozat fordul elő, akkor ki lehet emelni. A kiemelés helyéről hivatkozva rá aktivizálható.

Procedurális absztrakció: Lehetőség van a kiemelt szövegrész paraméterezésére, így nem csak egy tevékenység, hanem egy tevékenységcsoport végrehajtására képes.

Az alprogramok egy problémaosztályt oldanak meg. A probléma akkor konkretizálható, amikor az alprogramot az aktuális paraméterek megadásával meghívom. A fejben található, általában kerek zárójelek között. A nyelvek egy része azt mondja, hogy nem kell a zárójel, ha az alprogramnak nincs formális paramétere (pl. Pascalban), más része szerint pedig a zárójel nem a paraméterekhez, hanem az alprogramhoz tartozik, tehát paraméterek nélkül is ki kell tenni (pl. C). Nyelvfüggő, hogy a paraméterek mivel vannak elválasztva. A paraméterlistán szereplő nevek a törzsben különféle objektumok lehetnek: változók, nevesített konstansok, állománynevek, más alprogramok nevei. A korábbi nyelvekben a listán csak a paraméterek nevei szerepeltek, később a név mellet megjelent a típusmegjelölés, de lehetnek plusz információk is, amelyek a formális paraméterek futás közbeni viselkedését szabályozzák. Számuk bármennyi lehet, a nulla paraméterrel rendelkező alprogramot paraméter nélküli alprogramnak hívjuk.

Eljárás (PROCEDURE)

Hatását paramétereinek, környezetének vagy mindkettőnek megváltoztatásával fejti ki. Adat-transzformációt hajt végre vagy tevékenységet végez. Hívása utasításszerűen történik, végrehajtandó utasításnak tekinthető. A program szövegébe bárhol elhelyezhető eljáráshívás, ahol végrehajtandó utasítás lehet. Egyes nyelvekben külön alapszóval (általában CALL) hívható, más nyelvekben nincs rá alapszó.

Befejezése:

1. Az utasítások elfogynak, a vezérlést visszaadja a hívást követő utasításra.
2. Külön utasításra. Általában ez az utasítás a RETURN. Bárhol kiadható, a program a hívást követő utasításra tér vissza. Ez a szabályos befejeztetés. GOTO-val is ki lehet lépni, de nem szabályos. A megadott címkére kerül a vezérlés, ha a címke létezik. Veszélyes, mert a közbensőkkel bármi történhet, főleg ha egy hívási láncban vagyunk.
3. Befejeztető utasításokkal vagy eljárásokkal. A teljes programot befejeztetik. Nem teljesen szabályosak, de nem is veszélyesek. (pl. HALT, STOP)

Függvény (FUNCTION)

A matematikai fogalmat hozza át, feladata 1 db érték meghatározása. Nyelvfüggő, hogy ez az érték mennyire bonyolult struktúrájú lehet. Még egy komponense van, a függvény által visszaadott érték (visszatérési érték) típusa. Ezt a függvény neve hordozza, a fejben szerepel, a specifikáció része. Csak kifejezésben hívható meg, mert a függvény neve által hordozott értéket fel kell használni.

Befejezése és érték-hozzárendelés:

1. A függvény neve az eljárás törzsén belül mint változó használható (értékadó utasítás bal és jobb oldalán is), befejezéskor az értéke a legutoljára kapott érték. A függvény befejeződik, ha elértük a végét (a logikai sorrend számít, nem a felírásé, pl. Fortran)
2. A függvény nevének az eljárás törzsén belül értéket kell kapnia, értékadó utasítás jobb oldalán a függvény neve önmagában nem szerepelhet. Befejezéskor az értéke a legutoljára kapott érték. A függvény befejeződik, ha elértük a végét (a logikai sorrend számít, nem a felírásé, pl. Pascal).
3. Külön utasításra rendeli hozzá az értéket a függvény nevéhez a függvény befejeztekor.
   3.1. pl. RETURN[(kifejezés)]: Bárhol, akárhányszor kiadható, a hozzárendelés és a vezérlés-visszaadás az első RETUN-nél történik. A kifejezés elhagyható. (pl. C-ben)
   3.2. STOP
   3.3. GOTO: Végképp szabálytalan, de engedett. A függvényt és az eredeti kódot is otthagyja.

A függvény feladata 1 db visszatérési érték meghatározása. Ha a függvény megváltoztatja paramétereit vagy környezetét, akkor ezt a tevékenységet mellékhatásnak hívjuk. A nyelvek általában nem javasolják, de megengedett.

Paraméterek, paraméterátadási módok

Formális paraméter listában kell definiálni azt az adatszerkezetet, amely révén a függvény vagy eljárás bemeneti információt kap. A definiált adatszerkezet fiktív, a tényleges adatokat csak a függvény vagy eljárás hívásakor fogjuk előírni. A formális paramétereket a hívó program nem név szerint, hanem a felsorolásban elfoglalt pozíció szerint azonosítja: az első aktuális paraméter az első formális paraméternek, a második aktuális paraméter a második formális paraméternek, a harmadik aktuális paraméter a harmadik formális paraméternek és így tovább felel meg.

Aktuális paramétereknek azokat a paramétereket nevezzük, amelyeket a függvény vagy eljárás meghívásakor adunk meg a függvény vagy eljárás neve után. Ezeken hajtódnak végre a függvény vagy eljárás utasításai.

Változó vagy cím szerinti paraméterátadás

A változóparaméter értékét a hívott program (eljárás, függvény) megváltoztathatja, ekkor a hívó programban használt aktuális paraméter értéke is megváltozik.

A folyamat:

Meghatározódik a rendszer által az aktuális paraméter címkomponense, ez kerül átadásra a hívótól a hívotthoz. Az aktuális paraméter értéke ezen a tárcímen helyezkedik el, az alprogram itt dolgozik. Az információátadás kétirányú. A hívott alprogram tudja hol van a hívó egység, ha nem vigyázunk, szabadon mozoghat a hívó területén. Az aktuális paraméter változó is lehet.

Érték szerinti paraméterátadás

A formális paraméter értékének változása nem hat vissza az aktuális paraméter értékére. A kifejezés, utasítás az eljárás vagy függvény aktivizálásakor értékelődik ki, és a megfelelő formális paraméter ezt az értéket kapja meg.

A folyamat:

Meghatározódik a rendszer által az aktuális paraméter értéke, átkerül a hívótól a hívotthoz (átmásolódik a címkomponensre). Bizonyos kezdőérték-adásnak is tekinthető. Az információátadás egyirányú, a hívótól a hívott felé. A hívott alprogram nem tud semmit a hívóról, a saját területén dolgozik. Az aktuális paraméter ebben az esetben kifejezés lehet.

A formális paraméterlista változó deklarációk listája. Az aktuális paraméterlista pedig kifejezések listája.

Rekurzió

Rekurzió: olyan programtevékenység, ahol az eljárások, függvények önmagukat hívják meg. A feladatot esetekre bontjuk, és van olyan eset, amely eset önmagával van megfogalmazva, egy másik n értékkel. Ha ez a folyamat többször ismétlődik, hívási láncról beszélünk. A lánc alaphelyzetben dinamikusan változik, eleje mindig a főprogram. Függetlenül attól, hogy működik-e, minden eleme aktív. Aktív alprogram újra meghívását rekurzív hívásnak (rekurziónak) nevezzük.

Közvetlen rekurzió: A rekurzió egyszerű esete, az alprogram saját magát hívja meg.
Közvetett rekurzió: Az alprogram egy másik, a hívási láncban szereplő alprogramot hív meg.

Faktoriális:
n! = n*(n-1)*...*1
0! = 1
tehát ha n > 0, akkor n! = n*(n-1)!
ha n = 0, akkor n! = 1

Algoritmizálva:

```
Függvény Fakt(n:int):int
        Ha (n=0) akkor
                Fakt:=1;
        Különben
                Fakt:=Fakt(n-1)*n;
        Elágazás vége
Függvény vége
```

Hatványozás:
2^3 = 8
n^m = n^(m-1)*n
n^0 = 1 és 0^m = 0

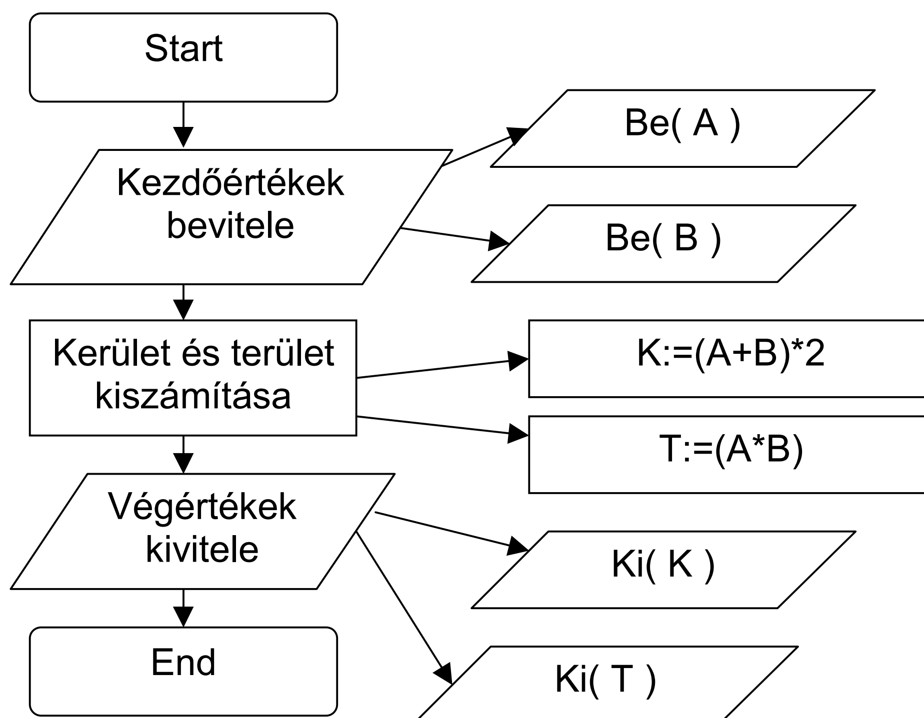
Algoritmizálva:

```
Függvény Hatvany(n,m:int):int
        Ha (n=0) akkor
                Hatvany:=0;
        Különben Ha (m=0) akkor
                        Hatvany:=1;
                Különben
        Hatvany:=Hatvany(n,m-1)*n;
                Elágazás vége
        Elágazás vége
Függvény vége
```

Programozási technikák

Írjunk algoritmust téglalap kerületének és területének kiszámítására!

Strukturált programozástechnika



A művelet amit itt bemutatunk a dekompozíció, vagyis a feladat egyre alapvetőbb részekre bontása. A dekompozíció az alapja a top-down módszernek, amely végeredménye olyan feladatok megfogalmazása, melyet már megvalósíthatunk az adott nyelv szintaktikájának megfelelően.

Objektumorientált programozástechnika

```
Elso:int;
Masodik:int;
Objektum Téglalap
        a: int;
        b: int;
Konstruktor Létrehoz;
Destruktor Megszüntet;
Függvény Kerület(a,b:int):int;
Függvény Terület(a,b:int):int;
Objektumdefiníció vége;
```

```
Konstruktor Létrehoz
        Új(Téglalap);
Eljárás vége

Destruktor Megszüntet
        Töröl(Téglalap);
Eljárás vége

Függvény Kerület(a,b):int
        Kerület:=(a+b)*2;
Függvény vége

Fügvény Terület(a,b):int
        Terület:=(a*b);
Függvény vége
```

```
Program kezdete
        Be(Elso);
        Be(Masodik);
        Téglalap.Létrehoz;
        Téglalap.A:=Elso;
        Téglalap.B:=Masodik;
        Ki(Téglalap.Kerület());
        Ki(Téglalap.Terület());
        Téglalap.Megszüntet;
Program vége
```

A példában az egységbezárásra látunk egyfajta megoldást, a téglalap oldalainak adatai és az adatokon elvégzendő műveletek egy objektumon belül találhatóak. Külön definiáljuk a metódusokat, közöttük a **konstruktort**, ami az objektumpéldány létrehozásáért felel és a **destruktort**, ami a megszüntetést végzi. A programegységek definiálása után a programtörzs egyszerű értékadásokat és függvényhívásokat tartalmazva áttekinthetővé, könnyen fejleszthetővé válik. Más példa (a birtokos viszony bemutatására):

```
Osztály Objektum
        {függvény Konstruktor()
                …                               /memóriafoglalás
        függvény vége
        függvény Destruktor()
                …                               /memória-felszabadítás
        függvény vége}
Osztálydefiníció vége

Osztály Pont: objektum
        {x:Z;                                   /koordináták
        y:Z;
        függvény Megjelenít()
                …                               /megjeleníti a pontot
        függvény vége}
Osztálydefiníció vége

Osztály Szakasz: objektum
        {a:pont;                                /pont objektum típus!
        b:pont;                                 /has a reláció!
        függvény Meghúz()
                …                               /meghúzza a szakaszt
        függvény vége}
Osztálydefiníció vége

Osztály Irányított_Szakasz: szakasz
        {függvény Meghúz(kezdőpont)
                …                               /virtuális metódus, megadott
        függvény vége}                          /kezdőpontból húzza a szakaszt
Osztálydefiníció vége
```

Ellenőrző kérdések

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mi az alprogramok szerepe a programozásban?
2. Ismertesse az ELJÁRÁS fogalmát!
3. Ismertesse a FÜGGVÉNY fogalmát!
4. Mi a paraméter? Milyen típusait ismeri?
5. Mi a különbség a cím és az érték szerinti paraméterátadás között?
6. Mi a rekurzió?

Elemi algoritmusok I.[4]

Az elemi algoritmusok jelentik a programtervek alapjait, olyan általánosan előforduló problémákra tartalmaznak megoldásokat, melyek a leggyakrabban jelentkeznek. A legtöbb elemi algoritmus matematikai elveken nyugszik, így "egy-az-egyben" való átültetésük nem mindig biztosítja a legjobb hatékonyságot, ám a megoldandó feladat kidolgozásához, a probléma felismeréséhez ismeretük nélkülözhetetlen.

Összegzés tétele

Adott egy **n** elemű **e** sorozat, vagy tömb. Határozzuk meg az elemek összegét! A végeredményt **s** tartalmazza.

```
s:=0;                          /s-nek értéket adunk, mivel értéktelen változóval nem lehet
                               / matematikai műveletet végezni;
ciklus i:=m - től n - ig       /majd elindítjuk a ciklust a sorozat vagy tömb első elemétől és
                               / haladunk az utolsó elemig (növekményes ciklus - tudjuk hol a vége);
        s:=s + e[i];           /s változóba beletöltjük s eddigi értékét, hozzáadva az aktuális elem
                               /értékét (a bal oldalon álló változónak adunk értéket a jobb
                               /oldalon álló kifejezéssel, a kiértékelés tehát jobb >> bal irányú);
ciklus vége                    /a ciklus vége, innen vagy a ciklus elejére ugrunk, vagy továbblépünk;
Ki(s);                         /kiírjuk s értékét, a ciklus véget ért, hiszen itt a vezérlés;
```

Példa

Határozza meg az [1, 100] intervallumba eső páros számok összegét!

```
Osszeg:=0;
Ciklus i:=1 - től 100 - ig
                                     /a mod a maradékképzés operátora, a feltétel arra a matematikai
    Ha (i mod 2 = 0) akkor           / igazságra utal, hogy a páros számok 2-vel maradék nélkül
        osszeg:=osszeg + i;          /oszthatóak;
    Különben
        semmi;                       /üres utasítás;
    Elágazás vége

Ciklus vége
Ki(osszeg);
```

Feladatok

1. Határozza meg az első n természetes szám összegét!
2. Adott az egész számok egy intervalluma. Határozza meg az intervallumba eső egész számok köbeinek összegét!
3. Feladat: hiányzási statisztika készítése.
   Egy tanulóról tudjuk, hogy a hónap egyes napjain hány órát mulasztott, Határozza meg, hogy ezen hónapban mennyi mulasztott óráinak a száma.
4. Egy osztály tanulóinak félévi matematika osztályzatai ismertek. Számítsa ki a matematika félévi átlageredményét!

A feladatokhoz készítse el az állapotteret és az algoritmust!

---

[4] ***Melléklet: pralap_IV.ppt***

Kiválasztás tétele

Adott egy **n** elemű **e** számsorozat, és az elemein értelmezett **T** tulajdonság. Tudjuk, hogy valamely elem(ek) a sorozatban **T** tulajdonságú(ak). Határozzuk meg az első ilyen elem sorszámát!

```
i:=1;                                    /kezdőérték beállítása;
Ciklus amíg (e[i] nem T tulajdonságú)    /ciklus indítása a ciklus addig tart amíg nem
                                         /találunk egy a tulajdonságnak megfelelő elemet
                                         /(elöltesztelő feltételfüggő ciklus);
        i:=i + 1;                        /ciklusérték növelése (a FOR ciklusnál automatikus,
                                         /a WHILE ciklusnál azonban a
                                         /programozóra van bízva);
Ciklus vége                              /ciklusláb;
Ki(i);                                   /kiírjuk az értéket vagy a tömbindexet
                                         /(sorozat vagy tömb);
```

Példa

Válasszuk ki a 25-nél nagyobb számok közül az első héttel oszthatót!

```
i:=25;                   /kezdőérték 25-re állítása;
Ciklus amíg (i mod 7 <> 0)   /ciklus amíg nem találunk olyan számot mely maradék nélkül osztható 7-tel;
     i:=i + 1;           /a ciklusmagban az inkerementálás;
Ciklus vége
Ki(i);                   /érték kiírása (sorozat!);
```

Feladatok

1. Adott egy tetszőleges természetes szám. Keressük a nála nem kisebb prímszámok közül a legkisebbet!
2. Adott egy tetszőleges természetes szám. Keressük a nála nem nagyobb prímszámok közül a legnagyobbat!
3. Döntsük el, hogy Kiss Ibolya hányadik az osztálynévsorban. Tudjuk, hogy van ilyen nevű tanuló az osztályban. Ha több is van, akkor az első előfordulását adjuk meg!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Megszámlálás tétele

Adott egy intervallumon vagy tömbön értelmezett **T** tulajdonság. Határozzuk meg, hogy hány **T** tulajdonságú elem van a sorozatban vagy a tömbben. A ciklusmag minden végrehajtása után **s** tartalmazni fogja az [**n, i**] intervallumban levő **T** tulajdonságú elemek számát.

```
db:=0;                                  /a kezdésnél a darabszám még 0;
Ciklus i: = m - től n - ig              /ciklus indítása 1-től (tömb!) vagy m-től (sorozat!)
        Ha (e[i] T tulajdonságú) akkor  /ha E i-edik eleme T tulajdonságú
                db:=db + 1;             /akkor a darabszámot növeljük eggyel;
        Különben
                semmi;                  /az egyirányú elágazásunk üres utasítása;
        Elágazás vége
Ciklus vége
Ki (db);                                /darabszám kiírása;
```

Példa

Számoljuk meg, hogy a [25, 68] intervallumban hány darab öttel osztható szám található!

```
db:=0;                          /darabszám a kezdetkor 0;
Ciklus i:=25-től 68-ig          /ciklus amíg tart a sorozat;
        Ha (i mod 5 = 0) akkor  /ha i maradék nélkül osztható 5-tel
                db:=db + 1;     /növeljük eggyel a darabszámot;
        Különben
                semmi;          /különben üres utasítással folytatjuk;
        Elágazás vége
Ciklus vége                     /ciklusláb;
Ki(db);                         /darabszám kiírás;
```

Feladatok

1. Hány prímszám található a természetes számok egy adott intervallumában?
2. Adjuk meg, hogy egy adott iskola nyilvántartásában hány Nagy Gábor nevű tanuló szerepel!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Elemi algoritmusok II.[5]

Eldöntés tétele

A kiválasztás hatékony akkor, ha biztosak vagyunk abban, hogy a tömbünk tartalmaz **T** tulajdonságú elemet. Azonban ha ez az állítás nem igaz, a kiválasztás tömbön túli indexelést okoz. Ha nem vagyunk biztosak abban, hogy egy tömb vagy zárt intervallum tartalmaz-e adott tulajdonságú elemet az eldöntést kell használnunk.

Döntsük el, hogy egy adott intervallum vagy tömb tartalmaz-e adott tulajdonságú elemet!

```
i:=m - 1;                               /i értékének beállítása, az első elem elé eggyel, hogy az első elemet
                                        / is vizsgálni tudjuk;
talált:=hamis;                          /a hamis egy logikai érték amit azért állítunk hamisra
                                        / hogy be tudjunk lépni a ciklusba;
Ciklus amíg ((talált = hamis) ÉS ( i < n ))  /ciklus amíg nem találunk T tulajdonságú elemet
                                        /ÉS nincs vége a tömbnek;
        i:=i + 1;                       /inkrementálás (WHILE!);
        talált:=e[i] T tulajdonságú;    /talált értékének beállítása, a jobb oldali kifejezés logikai:
                                        /E i-edik eleme vagy T tulajdonságú (igaz) vagy nem (hamis);
Ciklus vége
Ki(talált);                             /az eldöntés végeredménye: van-e T tulajdonságú elem?
```

Példa

Egy megadott intervallum tartalmaz-e nyolccal osztható számot?

```
Be(m);                                  /bekérjük az intervallum alsó (m) és felső (n) értékét;
Be(n);
i:=m-1;                                 /i értékének beállítása;
talált:=hamis;                          /talált értéke hamis, hogy be tudjunk lépni a ciklusba;
Ciklus amíg ((talált = hamis) ÉS (i < n))  /ciklus indítása az ismert feltételekkel;
        i:= i + 1;                      /inkrementálás;
        talált:= (i mod 8) = 0;         /talált értéke akkor igaz ha a sorozat adott eleme
Ciklus vége                             /maradék nélkül osztható 8-cal;
Ki(talált);                             /kiírjuk hogy a kérdésre adott válasz igaz-e vagy hamis?
```

Feladatok

1. Adott természetes számról döntsük el, hogy van-e valódi osztója!
2. Egy adott névről döntsük el, hogy szerepel-e az adott névsorban!
3. Természetes számok adott intervallumában van-e prímszám?

A feladatokhoz készítse el az állapotteret és az algoritmust!

[5] ***Melléklet: pralap_V.ppt***

Kiválogatás tétele

Adott egy **n** elemű **e** sorozat vagy tömb és egy **T** tulajdonság. Válogassuk ki a **T** tulajdonságú elemeket egy új vektorba!

```
j :=0;                                   /céltömb index inicializálása;
Ciklus i:=1-től n-ig                     /növekményes ciklus indítása;
        Ha (i T tulajdonságú) akkor      /ha a sorozat i-edik eleme T tulajdonságú;
                j:=j + 1;                /inkrementáljuk a tömbindexet;
                a[j]:=i;                 /hogy a következő tömbhelyre rakjuk a T tulajdonságú elemet;
        Különben
                semmi;                   /üres utasítás;
        Elágazás vége
Ciklus vége
```

Példa

Válogassuk ki az A tömbből a B tömbbe a prímszámokat!

```
j:=0;                                    /tömbindex 0-ra állítása (alaphelyzet);
Ciklus i:=1-től n-ig                     /ciklus az első tömbelemtől az utolsóig;
        Ha (A[i] prímszám ) akkor        /ha A i-edik eleme prím;
                j:=j + 1;                /növeljük B tömb tömbindexének értékét;
                B[j]:=A[i];              /B j-edik eleme legyen A i-edik eleme;
        Különben
                semmi;                   /üres utasítás;
        Elágazás vége
Ciklus vége
```

Feladatok

1. Egy osztálynévsorból válogassuk ki a K betűvel kezdődő nevű tanulókat!
2. Egy orvosi névsorból válogassuk ki a megadott magasságú betegeket!
3. Megadott sorozatból válogassuk ki a megadott számmal oszthatóakat!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Maximum-minimum kiválasztás tétele

Adott egy **n** elemű **e** tömb vagy sorozat. Keressük meg a legnagyobb (legkisebb) értéket!

```
i:=1;                              /tömbindex a tömb első elemindexe mivel a ciklusban már nem
                                   /hasonlítjuk össze az első elemet saját magával;
ind:=1;                            /ind változónak az első elem indexértékét adjuk;
max:=e[1];                         /a max értéke legyen az első elem értéke;
Ciklus amíg (i < n)                /ciklus indítása amíg i kisebb mint n;
      i:=i + 1;                    /ciklusindex inkrementálása;
      Ha (max < e[i]) akkor        /ha a jelenlegi max kisebb mint E i-edik eleme
                  max:=e[i];       /akkor max legyen e i-edik eleme;
                  ind:=i;          /ind változó értéke legyen az aktuális maximum érték indexe;
      Különben
                  semmi;           /üres utasítás;
      Elágazás vége
Ciklus vége
Ki (ind);                          /kiírjuk a maximum értéket és indexét;
Ki (max);
```

Mivel itt a ciklusfejben csak egy - a ciklusváltozó vizsgálatára vonatkozó - feltétel szerepel, így ezt a ciklust feltételes helyett növekményes módon is megadhatjuk.

Példa

Keressük meg egy intervallumon a legkisebb függvényértéket!

```
i:=m;                              /i értéke az intervallum alsó határa;
ind:=m;                            /ind értéke az első elem indexe;
min:=f(m);                         /min az első sorozatelemre értelmezett függvényérték;
      Ciklus amíg (i < n)          /ciklus indítása amíg i értéke kisebb/mint a sorozat felső határa;
            i:=i + 1;              /ciklusindex inkrementálása;
            Ha (min > f(i)) akkor  /ha min nagyobb mint az i-edik elemre értelmezett függvényérték
                  min:=f(i);       /akkor min legyen az;
                  ind:=i;          /ind pedig mindig a legkisebb érték indexét tárolja;
            Különben
                  semmi;           /ha min nem nagyobb (kisebb vagy egyenlő) akkor
            Elágazás vége          /ne változzon semmi;
Ciklus vége
Ki(ind);                           /output;
Ki(min);
```

Feladatok

1. Az $f(x) = 2x^3 - 3x^2 + 4x - 9$ függvényt a [-2; 15] egészintervallumon értelmezzük. Határozza meg az intervallumnak azt az elemét, ahol a függvény helyettesítési értéke a legnagyobb! Mekkora ez az érték?
2. Egy osztály orvosi lapjairól nyilvántartást készítettünk a tanulók fogainak számáról. Válassza ki a nyilvántartásból azt a gyereket, akinek a legtöbb foga van!
3. Az építőtábor 18 brigádja között hat napig tartó versenyt hirdettek meg. Egy táblázatban rögzítették naponta a brigádteljesítményeket, Ki lett a győztes brigád?

A feladatokhoz készítse el az állapotteret és az algoritmust!

Keresési tételek

Adott egy **n** elemű **e** sorozat vagy tömb és az elemein értelmezett **T** tulajdonság. Döntsük el, hogy van-e **T** tulajdonságú elem a sorozatban és mi annak a sorszáma. Észrevehető, hogy a keresés tétele alapesetben tulajdonképpen az eldöntés és a kiválasztás tételének kombinációja.

Lineáris keresés tétele

A sorozat elemei nem rendezettek. Adott egy **n** elemű **e** tömb, és egy **T** tulajdonság. Döntsük el, hogy **e**-nek van-e **T** tulajdonságú eleme, és ha van, akkor hányadik?

```
i:=1;                                              /indexváltozó állítása az első elemre;
Ciklus amíg ((i <= n) ÉS (e[i] nem T tulajdonságú))  /ciklus indítása amíg nem érjük el a tömb végét
                                                   /és nem találunk megfelelő elemet;
        i:=i + 1;                                  /inkrementálás a ciklustörzsben (while ciklus!);

Ciklus vége                                        /ciklusláb;

talált:=(i <= n);                                  /értékadás a talált változónak, ha túlmentünk a
                                                   /tömbön akkor nincs ilyen elem;
Ha (talált = igaz) akkor                           /ha talált értéke igaz akkor
        Ki(i);                                     /kiíratjuk a megtalált elem indexét;
Különben                                           /ha hamis
        Ki('Nincs ilyen elem');                    /akkor csak egy üzenetet;
Elágazás vége
```

Feladatok

1. Egy számokat tartalmazó tömbben hányadik a legelső 100-nál nagyobb szám (ha van)?
2. Egy névsorban található-e Szabó István nevű tanuló, és ha igen hányadik?
3. Adott a tanulók év végi matematika eredménye. Állapítsuk meg, hogy van-e bukott közöttük!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Logaritmikus keresés tétele

Az intervallum egyik határától kiindulva nem egyesével növeljük a már megvizsgált elemek számát, hanem a még kérdéses szakasz felével. Hogy melyik felével, azt a középső elemhez tartozó érték és az adott korlát nagyságviszonya dönti el. **H** ebben az esetben a keresett értéket jelenti.

```
alsó:=1;                               /kezdéskor az alsó értéke legyen az intervallum kezdőértéke;
felső:=n;                              /a felső értéke pedig a legnagyobb érték;
talált:=hamis;                         /talált értéke hamis, hogy be tudjunk lépni a ciklusba;
Ciklus amíg ((talált = hamis) és (alsó <= felső))
                                       /ciklust indítunk amíg nem találjuk meg a
                                       /keresett elemet és még van vizsgálandó tartomány;
        közép:=(alsó + felső) div 2;   /közép értéke legyen alsó és felső összege felének egészrésze;
        Ha (közép = h) akkor           /vizsgálatok: ha közép a keresett érték
                talált:= igaz;         /akkor talált értéke igaz lesz (megtaláltuk!);

        Különben Ha (közép < h) akkor  /ha közép kisebb mint a keresett érték
                alsó:=közép + 1;       /akkor az eddigi középnél eggyel nagyobb érték
                                       /lesz az intervallum alsó határa;
        Különben Ha (közép > h) akkor  /ha közép nagyobb mint a keresett érték
                felső:=közép - 1;      /akkor az eddigi középnél eggyel kisebb érték
                                       /legyen az intervallum felső határa;
        Különben
                semmi;                 /üres utasítás;
        Elágazás vége
        Elágazás vége
        Elágazás vége
Ciklus vége
Ha (talált = igaz) akkor               /ha a bináris keresés folyamán talált igaz értéket kap;
        Ki(közép);                     /ott megáll a ciklus és közép fogja tartalmazni a keresett értéket;
Különben
        Ki('Nincs ilyen elem');        /ha talált hamis maradt a ciklus lefutása után akkor nem találtuk
Elágazás vége                          /meg az intervallumon belül a keresett értéket;
```

N elemű tömbben való bináris keresés esetén a maximális lépések száma $Log_2(N)+1$, az átlagos keresés lépésszáma pedig $Log_2(N)$. Ezért hívják ezt a keresési formát logaritmikus keresésnek. Ismert még a bináris vagy felezéses keresés elnevezés is.

Feladat

Készítsünk algoritmust, amely kitalálja a felhasználó által gondolt számot (1-100), és kiírja azt is, hogy hány lépésben sikerült megtalálni azt.

A feladathoz készítse el az állapotteret és az algoritmust!

Visszalépéses keresés tétele

Adott n darab változó hosszúságú sorozat, mely sorozatok mindegyikéből ki kell választani úgy egy elemet, hogy az a megadott szabályoknak eleget tegyen. Alapesetben elemekhez tartozó többszörös értékek közül kell kiválasztani egyet oly módon, hogy minden elemhez tartozzon érték, de egyetlen érték se tartozzon több elemhez.

Előfeltétel: Adott az elemek tömbje és a hozzájuk tartozó sorozatérték tömbök
Utófeltétel: A program írja ki, hogy milyen párosításban lehet az elemekhez legalább egy sorozatértéket rendelni
Deklaráció: Megoldás[1..n], i: N;
Darab[1.n],Hozzárendelés[1..n,1..m], index, j : Z+

```
Függvény Jómegoldás( i, index ):L          /akkor jó a megoldás, ha nem egyezik meg egyik előzővel sem;
    j := 1;
    Ciklus amíg ((j < i) ÉS (Hozzárendelés[j, Megoldás[j]] <> Hozzárendelés[i,index]))
        j := j + 1;                        / ciklusfejben a vizsgálat. a törzsben csak inkrementálunk;
    Ciklus vége
    Jómegoldás := ( j >= i );
Függvény vége

Függvény Talált( i ):N                                /visszaadja a jó megoldás indexét;
    index := Megoldás[i]+1;                           /ha visszalépünk a tömbelem értéke az előzőleg
    Ciklus amíg (( index <= Darab[i] ) ÉS ( Jómegoldás(i, index) = hamis ))
                                                      /kiválasztott, ha a Hozzárendelés indexedik eleme
        index := index + 1;                           /már foglalt ( hozzárendeltük egy másik elemhez),
    Ciklus vége                                       /lépünk;
    Ha ( index <= Darab[i] ) akkor
        Talált := index;                              /Talált értéke a megtalált (elemhez tartozó)
    Különben                                          /Hozzárendelés indexe;
        Talált := 0;                                  /nulla ha nincs ilyen érték az elemhez tartozó
    Elágazás vége                                     /Darab intervallumon belül;
Függvény vége

Ciklus i:=1 - n-ig
    Megoldás[i]:=0;                                   /kezdéskor nullázzuk a Megoldás tömböt (még
Ciklus vége                                           /nincs megoldás);
i := 1;
Ciklus amíg ( i >= 1 ) ÉS ( i <= n ))                 /lépkedünk az értékeken;
    Megoldás[i] := Talált( i );                       /a megoldás a megtalált index;
    Ha ( Megoldás[i] <> 0 ) akkor                     /ha van megoldás továbblépünk;
        i := i + 1;
    Különben
        i := i - 1;                                   /ha nincs visszalépünk;
    Elágazás vége
Ciklus vége
Ha ( i > n ) akkor
    Ciklus i:= 1 - n-ig
        Ki( i, Megoldás[i] );                          megoldó kulcs kiíratása;
    Ciklus vége
Különben
    Ki( 'Nincs megoldás!' );                          /ha a feladat nem megoldható (nem mindegyik
Elágazás vége                                         /elemhez tudunk értéket rendelni);
```

Példa

Adottak munkák és dolgozók, azonos számban. Mindegyik dolgozó elmondja, hogy milyen munkát képes elvégezni. A feladat a munkák elosztása a dolgozók között oly módon, hogy minden dolgozóhoz tartozzon munka, és minden munkát elvégezzen valaki.

Munkák:

1. Festés
2. Tapétázás
3. Parkettázás
4. Villanyszerelés
5. Fűtésszerelés

Dolgozók	**Hány munkát vállalnak?**	**Milyen munkákat vállalnak?**	**Munkák elosztása**
	(Darab)	(Hozzárendelés)	(Megoldás (index!))
Tihamér	2	1, 3	1: Festés
Gusztáv	3	1, 3, 5	3: Fűtésszerelés
Vazul	1	3	1: Parkettázás
Huba	3	3, 4, 5	2: Villanyszerelés
Kázmér	1	2	1: Tapétázás

A fenti példában az első visszalépés a harmadik embernél (Vazul) történik, hiszen az elsőhöz hozzárendeljük az első munkát, a másodikhoz (az elsőt már foglalt!) a harmadikat, azonban a harmadik embernél evvel a módszerrel már zsákutcába jutottunk, hiszen Vazul csak a hármas munkát vállalja, amit már kiosztottunk. Ezért vissza kell lépni az előző dolgozóhoz, és másik munkát hozzárendelni. Ha nem lenne ilyen szabad munka, még egyet kellene visszalépnünk és az elsőnek új munkát kiosztani. Ez jelen esetben nem okoz problémát, hiszen már a második emberhez is képesek vagyunk új munkát hozzárendelni. A fenti példa optimális eset, előfordulhat, hogy nem tudjuk a párosítást megoldani úgy, hogy minden elemhez tartozzon érték. Ebben az esetben értékeljük ki úgy a feladatot, hogy arra megoldás nem található.

A fenti példa és a hozzá tartozó algoritmus alapeset, gyakorta előfordul, hogy más szempontokat is figyelembe kell venni, mint pl.: a kiválasztott elemek darabszáma ne haladja meg a valós értéket, a kiválasztott értékek rekurzív vizsgálata az elemekre (lehetséges párok képzése), az algoritmus "motorja", azonban mindig a fent látható visszalépéses keresésben található meg.

Rendezések I.[6]

A rendezés klasszikus számítástechnikai feladat. A problémakör lényege, hogy egy adathalmaz elemeit meghatározott sorrendbe rakjuk. Általánosan azt mondhatjuk, hogy adott egy N elemű E tömb, s a tömbelemeket kell növekvő vagy csökkenő sorrend szerint rendezni. Igen sokféle rendezési algoritmus létezik, különböző hatékonysággal, az hogy egy adott probléma magoldására melyiket választjuk, a megoldandó feladattól függ.

Rendezés közvetlen kiválasztással

A rendezendő számok legyenek az **A** vektor elemei. Az első menetben kiválasztjuk a vektor legkisebb elemét úgy, hogy az **A(1)**-et összehasonlítjuk **A(2)**, …, **A(n)** mindegyikével. Ha **A(1)**-nél kisebb elemet találunk, felcseréljük őket, vagyis ezt a kisebbet tesszük **A(1)**-be. Így a menet végére **A(1)** biztosan a vektor legkisebb elemét tartalmazza majd. Az eljárást **A(2)**-vel folytatjuk, ezt hasonlítjuk össze az **A(3)**, …, **A(n)** elemekkel, és így tovább, menetenként a soron következő legkisebb elem kiválasztásával. n-1 menet után a vektor rendezett lesz.

```
Ciklus i:=1 - től n - 1-ig              /külső ciklus ez határozza meg, hogy mit hasonlítunk
                                        /össze, az utolsó előtti elemig tart;
    Ciklus j:=i + 1-től n-ig            /belső ciklus, határozza meg, hogy az előző értéket
                                        /mivel hasonlítjuk össze;
        Ha (A[j] < A[i]) akkor          /ha a vektor nagyobb indexű eleme értéke mint a kisebb indexűé
            segéd:=A[j];                /akkor csere;
            A[j]:=A[i];
            A[i]:=segéd;
        Különben
            semmi;                      /ha nem akkor nem történik semmi;
        Elágazás vége

    Ciklus vége

Ciklus vége
```

[6] *Melléklet: pralap_VI.ppt*

Rendezés minimumkiválasztással

A felesleges cserék kiküszöbölése érdekében két segédváltozó bevezetésére van szükségünk. Az **érték** nevű változó tartalmazza az adott menetben addig megtalált legkisebb elemet, **index** pedig annak vektorbeli sorszámát, indexét. Az **A** vektor elemét mindig **érték** változó tartalmával hasonlítjuk össze. Ha értéknél kisebb elemet találunk, azt betesszük az **érték** nevű változóba és az **indexben** megjegyezzük a szóban forgó elem indexét. A menet végére az **érték** a vektor soron következő legkisebb elemét tartalmazza, **index** pedig azt a sorszámot, ahol ezt az elemet találtuk. Csak a menet utolsó lépésében van szükségünk cserére, amikor az **értékben** lévő legkisebb elemet helyére tesszük.

```
Ciklus i:=1 - től n - 1 - ig                /ciklusindítás az elsőtől az utolsó előtti elemig;
    index:=i;                               /index változó legyen az épp vizsgált elem indexe;
    min:=A[i];                              /a minimum pedig az értéke (egyelőre ez a legkisebb);
    Ciklus j:=i + 1-től n-ig                /belső ciklus az i-edik elemet követő elemtől az utolsóig;
        Ha (min > A[j]) akkor               /ha a j-edik elem kisebb mint az eddigi legkisebb;
            min:=A[j];                      /akkor ez az elem legyen a legkisebb;
            index:=j;                       /az index pedig ennek az indexe;
        Elágazás vége                       /vége az elágazásnak, különben ág nélkül (egyirányú
                                            /elágazás!);
    Ciklus vége                             /belső ciklus vége;
    A[index]:=A[i];                         /ha kisebbet találtunk akkor annak a helyére rakjuk
    A[i]:=min;                              /az i-edik elemet, az i-edik helyére pedig a legkisebbet;
Ciklus vége
```

Buborékos rendezés

Az első menetben a rendezendő **A** vektor végéről indulva minden elemet összehasonlítunk az előtte lévővel. Amennyiben rossz sorrendben vannak, felcseréljük őket. Az első menet végére a legkisebb elem biztosan a helyére kerül. Minden további menetben ismét a vektor végéről indulunk, de egyre kevesebb összehasonlításra van szükségünk, hiszen a vektor eleje fokozatosan rendezetté válik. Végeredményben a teljes vektor rendezéséhez n-1 menetre van szükségünk.

```
Ciklus i :=2 - től n - ig                   /ciklus indítása 2-től, mivel hátulról megyünk előre
                                            /így a másodikat hasonlítjuk össze az elsővel a végén;
    Ciklus j:=n - től i - ig                /belső ciklus a végétől az éppen összehasonlítandó elemig;
        Ha (A[j-1] > A[j]) akkor            /ha j-edik elem előtti elem értéke nagyobb mint a j-ediké;
            segéd:=A[j-1];                  /akkor csere;
            A[j-1]:=A[j];
            A[j]:=segéd;
        Elágazás vége                       /egyirányú elágazás különben ág nélkül, hisz abban
    Ciklus vége                             /úgyis csak egy üres utasítás lenne;

Ciklus vége
```

Beszúrásos rendezés

A rendezés során sorrendbeli hibát keresünk. A kialakult sorrendtől eltérő helyet megjegyezzük, az ott levő elemet elmentjük, majd addig keresünk attól visszafelé, amíg nála nagyobbat nem találunk, hiszen ez elé kell majd beszúrnunk. Amikor a helyet megtaláltuk, akkor a közbeeső elemeket feljebb tolva, az imént kiemelt elemet a felszabaduló helyre illesztjük. A következő ciklusban mindig eggyel magasabb pozícióról indulva ismételten elvégezzük a vizsgálatot, amíg ki nem alakul a megfelelő sorrend.

```
Ciklus i:=2 - től n - ig                  /ciklus kettőtől hiszen j eggyel kisebb mint i, ezért i nem
                                          /lehet 1, mert az tömbön túli indexelést jelentene;
    segéd:=A[i];                          /segédváltozó értéke az i-edik elem
                                          /(amit vizsgálunk);
    j:=i - 1;                             /j értéke eggyel kisebb mint i-é;
    Ciklus amíg ((j > 0) ÉS (segéd < A[j]))  /ciklus amibe csak akkor lépünk be, ha visszafele nem értük
                                          /el a tömb elejét, és a vizsgálandó elem kisebb mint a j-edik;
        A[j+1]:=A[j];                     / ilyenkor j értékét/i-be töltjük (i értéke=segéd!);
        j:=j - 1;                         /dekrementálás, visszafele haladunk;

    Ciklus vége
    A[j+1]:=segéd;                        /segéd értékét visszatöltjük;

Ciklus vége
```

Feladatok

1. Adott egy számokat tartalmazó tömb. Válasszon egy rendezési algoritmust és rendezze a tömböt csökkenő sorrendbe!
2. Adott egy Hőmérséklet tömb, mely három hónap, napi hőmérsékleteit tartalmazza. Válasszon egy rendezési algoritmust és rendezze a tömb elemeit növekvő sorrendbe!
3. Adott egy névsor. Rendezze a névsor elemeit születési idő szerint növekvő sorrendbe beszúrásos rendezéssel.

A feladatokhoz készítse el az állapotteret és az algoritmust!

Rendezések II.[7]

Gyorsrendezés

Tekintsük a tömb középső elemét (felezzük a tömböt). Balról keressük meg azt az első elemet ami ennél nem kisebb, jobbról ami ennél nem nagyobb. Cseréljük ki a két elemet, s folytassuk a cserélgetést egészen addig, amíg a baloldalon a középső elemnél (mely természetesen cserélődhet menet közben) csupa nem nagyobb, jobboldalon pedig csupa nem kisebb elem áll. Rekurzív hívással most rendezzük a tömb alsó és felső felét, stb.

Előfeltétel: adott a tömb.
Utófeltétel: a program rendezze a tömböt gyorsrendezéssel.
Deklaráció:

Bal, Jobb, i, j:Z+;
Tömb[1..n], Közép, Segéd:N;

```
Eljárás Gyorsrendezés(Bal:Z+,Jobb:Z+)
        Közép:=Tömb[(Bal+Jobb) Div 2];          /középérték meghatározása;
        i:=Bal;
        j:=Jobb;
        Ciklus amíg (i<=j)                       /külső ciklus;
                Ciklus amíg (Tömb[i] < Közép)    /ciklus amíg nem találunk a középértéknél
                        i:=i+1;                  /nagyobbat (i=bal!);
                Ciklus vége
                Ciklus amíg (Tömb[j] > Közép)    /ciklus amíg nem találunk a középértéknél
                        j:=j-1;                  /kisebbet (j=jobb!);
                Ciklus vége
                Ha (i <= j) akkor                /ha még nem szaladtunk túl a középen akkor
                        Segéd:=Tömb[i];          /csere;
                        Tömb[i]:=Tömb[j];
                        Tömb[j]:=Segéd;
                        i:=i+1;
                        j:=j-1;
                Elágazás vége
        Ciklus vége
        Ha (i < Jobb) akkor                      /ha a kisebb érték és Jobb között még van intervallum,
                Gyorsrendezés(i, Jobb);          /rekurzív hívással meghívjuk erre az intervallumra a
        Elágazás vége                            /gyorsrendezést;
        Ha (j > Bal) akkor                       /a közép másik oldalán lévő intervallumra is ugyanez;
                Gyorsrendezés(Bal, J);
        Elágazás vége
Eljárás vége
```

[7] **Melléklet: Algoritmusok\sortexe\sortdemo.exe**

Összefésülés

Az összefésülés és változatai is gyakori problémakört alkotnak. Alapesetben két rendezett tömböt (sorozatot, file-t stb.) kell összefésülni egy harmadikba, olyan módon hogy a rendezettség megmaradjon.

```
i:=0;                                      /az új tömb indexe;
j:=1;                                      /az egyik összefésülendő tömb indexe;
k:=1;                                      /a másik összefésülendő tömb indexe;
Ciklus amíg (i < m + n)                    /ciklus amíg i kisebb mint a két tömb elemeinek
                                           /összege (ekkora lesz az új tömb);
    i:=i + 1;                              /inkrementálás;
                                           /feltétel: ha nem futottunk ki egyik tömbből sem
    Ha ((k > n) VAGY ((j <= m) ÉS (A[j] < B[k]))) akkor
                                           /ÉS A j-edik eleme kisebb mint B k-adik eleme VAGY
        C[i]:=A[j];                        /vége B tömbnek akkor az A tömb elemét rakjuk C-be;
        j:=j + 1;                          /léptetjük A tömböt;

    Különben Ha ((j > m) VAGY ((k <= n) ÉS (B[k] < A[j]))) akkor
                                           /az előbbi feltétel B tömbre értelmezve;
            C[i]:=B[k];                    /B elemét rakjuk C-be, hisz az a kisebb;
            k:=k + 1;                      /és B-t léptetjük;

        Különben Ha (A[j] = B[k]) akkor    /mindkét tömb még "él" hiszen ezen az ágon
                                           /vagyunk, és egyenlők az elemek;
                C[i]:=A[j];                /akkor megállapodás szerint A-val kezdjük;
                j:=j + 1;
                i:=i+1;                    /két elem kerül C-be, így i-t is léptetni kell;
                C[i]:=B[k];                /B-vel folytatjuk;
                k:=k + 1;

            Elágazás vége                  /a három egymásba ágyazott elágazás vége;

        Elágazás vége

    Elágazás vége

Ciklus vége                                /összefésülési eljárás vége;
```

Feladatok

1. Adott két egész számokat tartalmazó rendezett tömb, melyben minden elem csak egyszer szerepel, s így halmazként is felfoghatóak. Írjon algoritmust, mely elkészíti a két tömb unióját!
2. Adott két egész számokat tartalmazó rendezett tömb, melyben minden elem csak egyszer szerepel, s így halmazként is felfoghatóak. Írjon algoritmust, mely elkészíti a két tömb metszetét!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Vegyes Feladatok

Emil és Ödön nyáron három héten át dolgoztak, málnát szedtek. A málna nem egyenletesen érett, ezért minden nap más és más mennyiséget sikerült leszedniük. A munkabérből utána nyaralást terveztek, Balaton körüli biciklitúrára készültek. A túra szervezéséhez gondosan tanulmányozták a térképet, a parti települések neveit és a megteendő távolságokat.

<u>Adottak:</u>
Emil napi málnaadagjai MálnaE[1..21]:N
Ödön napi málnaadagjai MálnaÖ[1..21]:N
Balaton körüli települések nevei Település[1..n]:string
Települések közötti távolságok (az i.-től az i+1. Településig) Távolság [1..n]:N

1. Emil vagy Ödön szedett több málnát a három hét alatt?
2. Mennyi volt Emil napi átlaga?
3. Hányadik napon kezdett jól teremni a málna, vagyis mikor szedtek először ketten együtt 20 kg-nál többet?
4. Hányadik naptól csökkent a málna mennyisége Emil szedése alapján?
5. Hány olyan nap volt, mikor Ödön 5 kg-nál több málnát szedett?
6. Lustálkodott-e a második hét közben Emil, volt-e olyan nap amikor nem szedett semmit?
7. Mely napokon szedtek fejenként 10 kg-nál többet?
8. Melyik volt Emil legjobb napja, mennyi málnát szedett aznap?
9. Melyik volt Ödön legrosszabb napja, mennyi málnát szedett aznap?
10. Rendezze sorba a napokat (közvetlen kiválasztás) Emil növekvő teljesítménye szerint!
11. Rendezze sorba a napokat (buborékos) Ödön növekvő teljesítménye szerint!
12. Fésülje össze a kapott tömböket egy harmadikba!
13. Rendezze csökkenő sorrendbe a településtávolságokat (maximum kiválasztás)!
14. Melyik a leghosszabb nevű település?
15. Mely településektől vannak a szomszédos települések 5 km-nél közelebb?
16. Hány település neve kezdődik "Balaton"-nal?
17. Van-e olyan két szomszédos település, melyek között a távolság 10 és 12 km közötti?
18. Hány 20 km-nél hosszabb távolság van a települések között?
19. Balatonfüredtől indulva hányadik település Balatonszemes?
20. Hány km-t kell átlagosan megtenniük, ha 3 hét alatt körbe akarják járni a tavat?

Adattárolási módszerek I.[8]

Mátrix

A mátrix a kétdimenziós tömb elnevezése a számítástechnikában. Nagyon sok olyan adattárolási forma van, amit hatékonyan csak mátrixszal lehet megoldani. A mátrixok beolvasása, a bennük való keresés könnyű, az alapelv az, hogy két egymásba ágyazott ciklust indítunk, az egyik a sorokon lépdel végig a másik az oszlopokon. Mindig a sorokon való végighaladással kezdjük, mivel a gép memóriájában a kétdimenziós tömb is vektorként van tárolva (a sorok egymás után való felfűzésével).

Bármely mátrix leképezhető vektorra az alábbi egyszerű szabály ismeretében:

k:= oszlopszám*(i-1)+j

ahol k a vektorindex, i a mátrix sorindexe, j pedig az oszlopindex. Könnyű belátni, hogy

Mátrix[2,3]=3*1+3=6

vagyis a mátrix 2. sorának 3. oszlopában szereplő érték a memóriavektor hatodik eleme (feltéve, ha a mátrix oszlopainak száma három).

Példa

Összegezzük A[1..m,1..n] mátrix elemeit!

```
összeg:=0;                              /összeg változó inicializálása;
Ciklus i:=1-től m-ig                    /külső ciklus, a sorokon lépdel;
        Ciklus j:=1-től n-ig            /belső ciklus, az adott sor celláin (oszlopokon) lépdel;
                összeg:=összeg + A[i,j]; /összeghez adjuk hozzá az A mátrix i-edik sorában
        Ciklus vége                     /j-edik oszlopában lévő cella összegét;
Ciklus vége
```

Feladatok

1. Készítsünk algoritmust mely egy m*n-es mátrixnak megadja a legnagyobb és legkisebb elemét!
2. Válaszuk ki egy m*n-es pozitív egész számokat tartalmazó mátrix legnagyobb összegű sorát!
3. Adott egy n*n-es mátrix. Írjunk algoritmust, ami megadja a mátrix átlóiban lévő elemek összegét!

A feladatokhoz készítse el az állapotteret és az algoritmust!

[8] ***Melléklet: pralap_VII.ppt***

Szöveg (string)

A szöveg valójában egy karaktertömb, így hivatkozhatunk a szövegen belül lévő karakterekre, a tömbnél megszokott módon (**Szöveg[1]**). Segédfüggvényként gyakran alkalmaznak Hossz értékű alprogramot, mely visszaadja a karaktertömb elemeinek számát (a szöveg hosszát).

Példa

Adjuk meg egy szövegben lévő mássalhangzók számát!

```
Be(szöveg);                                   /bekérjük a szöveget;
db:=0;                                        /még nem tudjuk, hány mássalhangzó lesz;
Ciklus i:=1 től Hossz(szöveg)-ig              /ciklus amíg el nem érjük a karaktertömb végét;
        Ha (szöveg[i] mássalhangzó) akkor     /vizsgálat;
                db:=db + 1;                   /ha mássalhangzó növeljük a darabszámot eggyel;
        Elágazás vége
Ciklus vége
Ki(db);                                       /írassuk ki a darabszámot;
```

Feladatok

1. Írjunk algoritmust, ami eldönti, hogy egy adott hosszúságú szövegben, megtalálható-e a "kutya" szó!
2. Adott egy szöveg, készítsünk algoritmust arra, amely kiszámítja a szövegben előforduló karakterek relatív gyakoriságát!
3. Számoljuk meg egy szövegben a szavak számát!

A feladatokhoz készítse el az állapotteret és az algoritmust!

File

A file adatszerkezet lehet tömbszerű (szekvenciális állomány), vagy rekordalapú (direkt állomány). A direkt eléréshez indexeket használunk, amelyek megmutatják, hogy a keresett rekord, hol helyezkedik el az állományon belül. Indexelt állományról akkor beszélünk, ha ezeket az indexeket külön adatszerkezetbe (indextáblába) gyűjtjük. Ha az indexek mutatókat tartalmaznak a következő index(ek)re, indexlista jön létre, ekkor beszélünk indexelt szekvenciális állományról (a lista szekvenciális). Ha a rekordok azonos méretűek, akkor a rekordok mezőit önállóan is kezelhetjük, azonban ha a rekordok változó hosszúságúak (ez a gyakoribb), akkor csak teljes rekordokat tudunk felvinni ill. kiolvasni.

A file-ok alapműveletei

- **megnyitás (open):** beolvassuk az állomány és azonosítót (handle) rendel hozzá valamint beállítja a mutatót, ami jelzi hogy hol tartunk az állományban.
- **lezárás (close):**
  - ***mentéssel:*** kiírjuk a változásokat a lemezre és töröljük a mutatót, a file-változó összerendelés megmarad (a handle törlődik).
  - ***mentés nélkül:*** csak a mutatót töröljük, valamint felszabadítjuk a memóriaterületet.
- **olvasás (read):** a mutató által meghatározott helyről beolvassuk az adatokat.
- **írás (write):** a mutató által meghatározott pozícióba írjuk az adatot (ha volt ott adat felülírjuk).
- **felülírás (rewrite):** töröljük a file tartalmát (ha volt) és beleírjuk az adatokat, a mutató a file elejére mutat.
- **hozzáfűzés (append):** nem töröljük a file tartalmát, a file végére állítjuk a mutatót és oda írjuk az adatainkat.
- **áthelyezés (seek):** a mutató mozgatása a file-on belül.

Példa

Adott két szekvenciális file **x** és **y** , mindkettő rendezett, **z** állományba fésüljük össze a két állomány tartalmát, úgy hogy a rendezettség megmaradjon!

```
MEGNYIT(z);                                       /megnyitjuk (létrehozzuk) z-t;
MEGNYIT(x);                                       /megnyitjuk x-et;
MEGNYIT(y);                                       /megnyitjuk y-t;
Ciklus amíg ((nem eof(x)) VAGY (nem eof(y)))      /ciklus amíg bármely állományban van még adat;
        Ha ((eof(y)) VAGY ((nem eof(x)) ÉS (ex < ey))) akkor
                HOZZÁFŰZ(z, ex);                  /x adott rekordja kisebb mint y-é;
                OLVAS(x, ex, eofx);               /olvasással léptetünk a következő rekordra;
        Különben Ha ((eof(x)) VAGY ((nem eof(y)) ÉS (ey < ex))) akkor
                        HOZZÁFŰZ(z, ey);
                        OLVAS(y, ey, eofy);       /léptetés, amíg nincs file vége;
                Különben Ha (ex = ey) akkor
                                HOZZÁFŰZ(z, ex);
                                OLVAS(x, ex, eofx);
                                LÉPTET(z);        /mutató mozgatása direkt módon a következő rekordra;
                                HOZZÁFŰZ(z, ey);
                                OLVAS(y, ey, eofy);
                        Elágazás vége
                Elágazás vége
        Elágazás vége
        LÉPTET(z);
Ciklus vége
LEZÁR(x, nem ment);
LEZÁR(y, nem ment);
LEZÁR(z, ment);                                   /elmentjük a beleírt adatokat;
```

Adattárolási módszerek II.[9]

Tábla

A tábla egy asszociatív adatszerkezet, melynek elemei kulcs és adat párok, ahol a kulcsok egyediek és bármely elem a kulcsán keresztül érhető el. A táblával kapcsolatos műveletek központi kérdése az adott kulcshoz tartozó adatok, minél rövidebb idő alatt történő megkeresése, és a tábla karbantartása.

A tábla egy olyan adatszerkezet, amelyen a következő műveletek vannak értelmezve:

- Keresés
- Beszúrás
- Törlés
- Szekvenciális elérés

A táblaműveletek hatékonysága a fizikai tárolástól, vagyis a tábla szervezésétől függ. Szinte minden művelet tartalmaz keresést, hiszen nem vihetünk fel például olyan adatot, melynek kulcsa már szerepel a listában. A tábla valójában az egydimenziós tömb (a vektor) általánosítása - ott a tárolt adatokat indexeken keresztül érjük el, vagyis a kulcsok maguk az indexek.

A táblák lehetnek:

- Sorosak
- Önátrendezők
- Rendezettek
- Kulcstranszformációsak

Mi ez utóbbival foglalkozunk behatóbban.

Mi a kulcstranszformáció lényege? Minden táblatípushoz tartozik egy algoritmus, mely meghatározza az új elem beszúrásának helyét. Kulcstranszformációs tábla esetében a kulcshoz tartozó transzformáció (függvény) adja meg az elem helyét (indexét). A transzformációs függvényt hash (hasító) függvénynek is szokás nevezni, feladata pedig az ún. kulcsütközések feloldása, vagyis annak elkerülése, hogy két kulcshoz, ugyanaz az index rendelődjön.

Ha a kulcsok egész számok a transzformációs függvények e következők lehetnek:

- Maradékképzés (hasítás): ha az indexek 1 és m közé esnek, a kulcshoz rendelés módja lehet a (Kulcs mod m) + 1.
- Szorzás: a kulcsot összeszorozzuk saját magával és vesszük az eredmény középső jegyeit.
- Randomizálás: egy olyan véletlenszámgenerátor használata, ami egy adott értékhez mindig ugyanazt a "véletlen" értéket rendeli.

[9] ***Melléklet: Algoritmusok\index.html***

Hash algoritmus

Hash keresést - rendezést (mint a neve is mutatja) asszociatív (hash) tömbön lehet végrehajtani. Az asszociatív tömb jellegzetessége, hogy kulcs-érték párokból áll, ahol a kulcsot és az értéket is a felhasználó adja meg. A Hash táblák kezelése azon alapszik, hogy a kulcsként meghatározott adathoz, az algoritmus mindig ugyanazt a táblaindex értéket rendeli hozzá, ezáltal meghatározható, hogy egy adott érték hol található a táblában (a kulcsa alapján). Ha a hozzárendelt táblaindex foglalt, akkor a következő indexérték vesszük, ha az is foglalt , akkor az azt követőt, és így tovább. Ha elérjük a tábla végét az elejére pakolunk. A legfontosabb itt annak a módszernek a meghatározása amivel egy kulcshoz táblaindexet rendelünk. A legismertebb módszer a

Kulcs mod táblahossz + 1

mivel természetesen biztosítanunk kell azt, hogy a hozzárendelt táblaindex az **1..táblahossz** tartományban legyen, valamint azt is, hogy elég egyenletes eloszlású maradjon az indexek kiosztása, hiszen a kijelölés itt nem lineáris módon történik mint az előző algoritmusokban. Az előző maradékképzés eredménye úgy is tekinthető, mintha a kulcsot **táblahossz** alapú számrendszerben írtuk volna fel, és az indexhez csak az utolsó számjegyet használtuk volna fel (hozzáadva egyet), ebből származik a csonkításos - hash - algoritmus neve. A keresésnél meghatározzuk a kulcshoz az indexet, majd onnan kezdünk keresni egészen addig amíg meg nem találjuk az értéket vagy üres táblabejegyzést nem találunk.

Példa

Adott egy **n** elemű **e** tömb. Keressünk meg a tömbben egy adott értéket!

```
Be(érték);                          /bekérjük az értéket;
kulcs:=Kulcs(érték);                /a Kulcs függvény visszaadja az értékhez tartozó kulcsot;
index:=kulcs mod n + 1;             /a kulcs ismeretében meghatározzuk az indexet;
nincs:=hamis;                       /logikai segédváltozó, abban bízunk, hogy van ilyen elem;
Ciklus amig ((e[index] <> érték) ÉS (e[index] <> NULL) ÉS (nincs = hamis))     /ciklus amíg e indexedik
                                    /eleme nem a keresett elem, nem üres és nincs értéke hamis;
        index:= index + 1;          /index inkrementálása;

        Ha (index > n) akkor        /ha túlmegyünk a táblahosszon

                index:=1;           /kezdjük az elejéről;

        Elágazás vége

        Ha (index = kulcs mod n + 1) akkor     /ha az index a legelőször meghatározott index,
                                    /az azt jelenti, hogy "körbejártunk";
                        nincs:=igaz;    /vagyis nincs ilyen értékű elem;

                Elágazás vége       /nincs különben ág, hiszen ott úgyis üres utasítás lenne;

Ciklus vége

Ha ((nincs = hamis) ÉS (e[index] <> NULL)) akkor
        Ki(index);                  / ha megtaláltuk akkor kiírjuk;
Különben                            /különben ha "körbejártunk" vagy kiléptünk a ciklusból
        Ki('Nincs ilyen elem');     /mert üres elemet találtunk, kiírjuk, hogy nem találtuk meg;
Elágazás vége
```

Verem adatszerkezet

A verem a legáltalánosabb és legelterjedtebb adatszerkezet. Minden olyan helyzetben alkalmazható, ahol az adatokhoz a beírásukhoz képest fordított sorrendben kívánunk hozzáférni. Az egymásba ágyazott függvények változói nagyon kényelmesen kezelhetőek ezzel a szerkezettel, hiszen a függvények belülről kifelé hajtódnak végre, vagyis előbb azok amiket később hívtunk meg.

A verem fő jellemzője, hogy a tetejére pakolunk, és onnan is veszünk ki, hiszen egy elemet csak a felette lévő elemek eltávolítása után érhetünk el. Azon helyzetekben alkalmazható jól, ahol az adatok felhasználása **LIFO** (Last in, first out - a később jött előbb távozik) rendszerben történik

Globális változók:

ADATOK [1..n]:string /a "valódi" vektor amit veremként kezelünk;
szabad:Z+ /az első szabad hely a veremben (tömbben);

```
Eljárás VTÖRÖL                          /a verem alaphelyzetbe állítása (törlése);
        szabad:=1;                      /üressé válik a verem, ha a  szabad helyre mutató változót
Eljárás vége                            /az első elemre állítjuk;

Függvény VÜRES:L                        /verem ürességének vizsgálata;
        VÜRES:=(szabad = 1);            /üres a verem, ha az szabad hely változója az első elemre
Függvény vége                           /mutat;

Függvény VTELE:L                        /verem feltöltöttségének vizsgálata;
        VTELE:=(szabad> n);             /tele a verem, ha az szabad hely változója nagyobb értékű
Függvény vége                           /mint az elemek maximális száma;

Függvény VEREMBŐL:string                      /elem kivétele a veremből (POP);
        Ha (VÜRES) akkor                      /ha a VÜRES függvény igaz értéket ad vissza;
          Ki('Nincs elem, a verem üres');     /üzenet kiírása;
          VEREMBŐL:=NULL;                     /a függvény által visszaadott érték üres;
        Különben                              /ha nem üres a verem;
          szabad:= szabad-1;                  /akkor - mivel kiveszünk egy elemet - felszabadul egy hely
          VEREMBŐL:= ADATOK[szabad];          /vagyis a szabad hely változója dekrementálódik;
        Elágazás vége                         /a kivett adat pedig legyen az ADAT tömb beállított
Függvény vége                                 /"szabad"-ik eleme (ami a kivétel miatt üressé válik;

Eljárás VEREMBE(adat:string)            /elem betétele a verembe (PUSH);
        Ha (VTELE) akkor                /ha a szabad változó értéke nagyobb mint a tömb utolsó indexe;
          Ki('Tele a verem');           /akkor nincs már szabad hely;
        Különben                        /ha van még hely a veremben;
          ADATOK[szabad]:=adat;         /ADAT-ok tömb "szabad"-ik eleme legyen a paraméterként
                szabad:=szabad + 1;     /megadott érték, a szabad változó értéke inkrementálódik
        Elágazás vége/                  (eggyel csökken a szabad helyek száma);
Eljárás vége
```

Sor adatszerkezet

A sor adatszerkezet fő jellemzője, hogy mindig a sor végére pakolunk és a sor elejéről veszünk el elemeket. Azon helyzetekben alkalmazható jól, ahol az adatok felhasználása **FIFO** (First in, first out - az előbb jött előbb távozik) rendszerben történik. Ha a tömb végére érünk, és az eleje üres, akkor a sor elejére pakolunk.

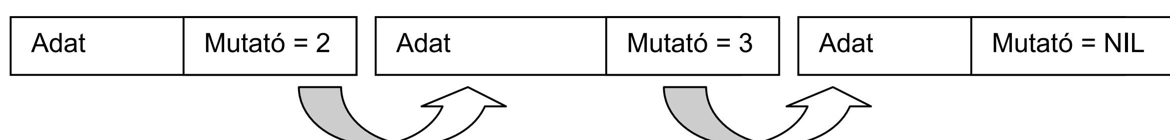
Globális változók:

```
ADATOK[1..n]:string
honnan:Z+                              /sor eleje;
hova:Z+                                /utolsó elem utáni első szabad hely;
db:N                                   /sorban lévő elemek száma;

Eljárás STÖRÖL                         /sor inicializálása (törlése);
    db:=0;                             /nincs elem;
    hova:=1;                           /az első helyre pakolhatunk;
    honnan:=1;                         /az első helyről vehetünk ki;
Eljárás vége

Függvény SÜRES:L                       /sor ürességvizsgálata;
    SÜRES:=(db=0);                     /nincs elem a sorban;
Függvény vége

Függvény STELE:L                       /sor feltöltöttségének vizsgálata;
    STELE:=(db=n);                     /a darabszám megegyezik a maximális elemszámmal;
Függvény vége

Függvény SORBÓL:string                 /elem kivétele a sorból;
    Ha (SÜRES) akkor                   /ha nincsenek elemek a sorban
        Ki('Nincs elem, üres a sor');  /akkor a sor üres;
        SORBÓL:=NULL;                  /a visszaadott érték üres;
    Különben                           /ha vannak
        SORBÓL:=ADATOK[honnan];        /akkor a "honnan"-adik helyen lévő elemet vesszük ki;
        honnan:=honnan + 1;            /és "honnan"-t inkrementáljuk;
        Ha (honnan > n) akkor          /ha elérjük a sor végét
            honnan:=1;                 /kezdjük az elejétől;
        Elágazás vége
            db:=db - 1;                /kivettünk egy elemet, a darabszám csökken;
    Elágazás vége
Függvény vége

Eljárás SORBA(adat:string)             /elem betétele a sorba;
    Ha (STELE) akkor                   /ha a darabszám megegyezik az indexszámmal
        Ki('Tele a sor');              /akkor nincs hely a sorban;
    Különben                           /ha van
        ADATOK[hova]:=adat;            /akkor a "hova"-dik helyre berakjuk az adatot;
        hova:=hova + 1;                /inkrementáljuk a "hova" változót;
        Ha (hova > n) akkor            /ha elérjük a sor végét
            hova:=1;                   /az elejére pakolunk;
        Elágazás vége
    db:=db + 1;                        /betettünk egy elemet, a darabszám nő eggyel;
    Elágazás vége
Eljárás vége
```

Adattárolási módszerek III.

Lista

A lista egy olyan szerkezet, amely megmondja, hogy egy adott adatelem után melyik a (logikailag) következő. Nevezik ezt a tárolási formát láncolt ábrázolásnak is. Könnyű az elemek kiolvasása, új elem beszúrása, de a műveletekhez szekvenciálisan végig kell olvasni az egész listát. Ez az adatszerkezet akkor is előnyös, amikor két adatelem közé kell beszúrni egy újat, vagy az adatsor közepéről kell törölni egy elemet.
A láncolt adatszerkezet így ábrázolható:



Listatípusok

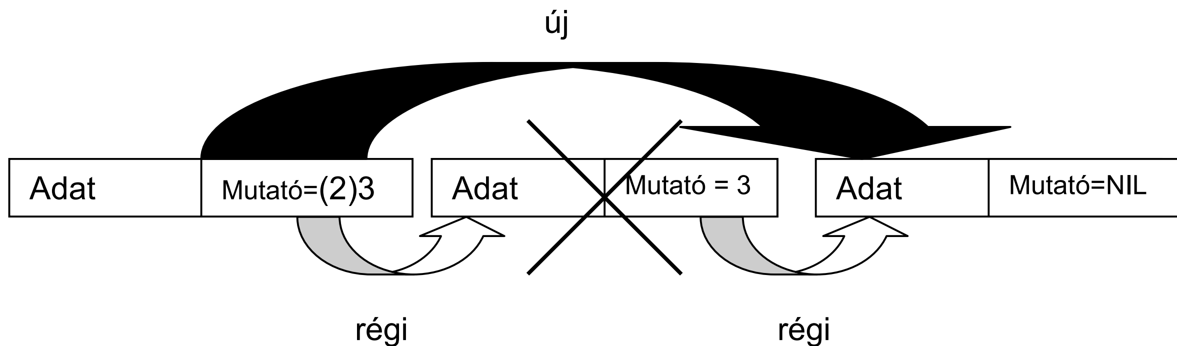
- Egyirányú: minden elem csak egy mutatót tartalmaz az őt követő elemre.
- Kétirányú vagy szimmetrikus lista: minden elem két mutatót tartalmaz egyet az őt megelőzőre, egyet pedig az őt követőre.
- Nyíltvégű: a lista utolsó elemének mutatója értéktelen (NIL).
- Cirkuláris: a lista utolsó elemének mutatója az első elemre mutat.
- Multilista: minden listaelem egy újabb lista kiindulópontja lehet.
- Statikus: előre meghatározott számú és lefoglalt listaelemet tartalmaz.
- Dinamikus: tetszés szerint bővíthető új elemek lefoglalásával.

A példánkban egy egyirányú, nyíltvégű, dinamikus lista karbantartó programját adjuk meg.

<u>**Globális változók:**</u>

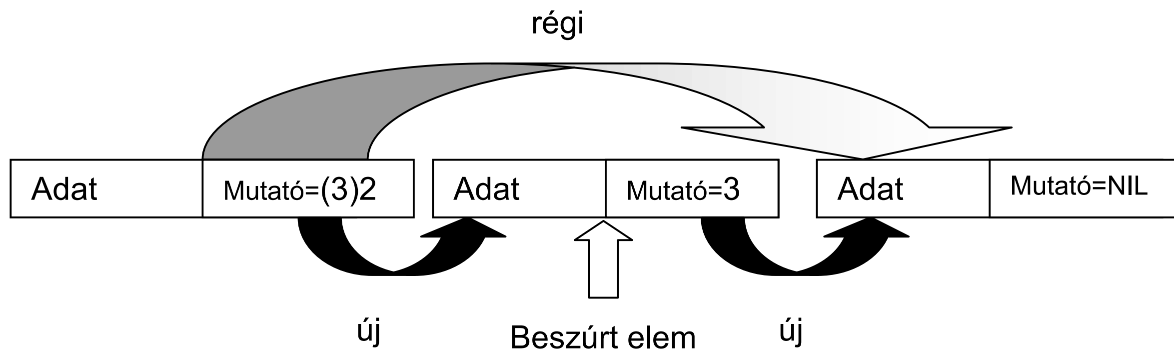
```
Rekord Listaelem                /önhivatkozó struktúra;
        adat:string
        köv:mutató              /egy rekordra mutató mutatót tartalmazó rekord;
Rekord vége
Lista[1..n]:Listaelem           / a dinamikusan kezelt tömb;

ELEJE:mutató                    /lista elejére mutat;
AKTUÁLIS:mutató                 /az aktuális elemre mutat;
KÖVETKEZŐ:mutató                /az aktuális elem köv mezőjében szereplő elemre mutat;
ELŐZŐ:mutató                    /az aktuális megelőzőre mutat;

MEMFOG(Byteok:Z+):mutató        /lefoglalja a memória egy részét (akkorát amekkora a
                                /tömbelem típusának szükséges) és visszaadja a lefoglalt
Például:                        /memóriaterület címét egy mutatóban;
P:=MEMFOG(20)
Ha (P = NIL) akkor              /a NIL egy speciális mutatóérték nem mutat sehova;
        ('Hiba, nincs elég memória a listához')
Elágazás vége
```

```
MEMSZAB(P)                                    /felszabadítja a memória mutató által
                                              /meghatározott részét;

Függvény ÜRES:L
        ÜRES:=(ELEJE = NIL);                  /üres a lista, ha az első elem értéktelen;
Függvény vége

Függvény VÉGE:L
        VÉGE:=(KÖVETKEZŐ = NIL)               /vége van a listának, ha NIL az következő elem értéke,
Függvény vége                                 /vagyis az utolsó elemen állunk;

Eljárás ELSŐRE
        AKTUÁLIS:= ELEJE;                     /az első elem lesz az aktuális;
        KÖVETKEZŐ:= (*AKTUÁLIS).köv;          /a következő pedig az őt követő elem;
        ELŐZŐ:=NIL;                           /ha az első elem az AKTUÁLIS, akkor az előző NIL lesz
Eljárás vége                                  / nincs megelőző elem;

Eljárás ALAP                                  /alapállapotba hozza (inicializálja) a listát;
        ELSŐRE;
        Ciklus amíg (AKTUÁLIS <> NIL)         /amíg el nem érjük a lista végét;
                MEMSZAB(AKTUÁLIS) ;           /töröljük az aktuális elemet;
                AKTUÁLIS:=KÖVETKEZŐ;          /az aktuális legyen a következő elem;
                KÖVETKEZŐ:=(*AKTUÁLIS).köv;   /következő mutasson az aktuális elemet
        Ciklus vége                           /követőre;
        ELEJE:=NIL;                           /üres a lista, nincs kezdő elem;
        ELŐZŐ:=NIL;                           /ha az eleje NIL, akkor előtte sincs semmi;
Eljárás vége


Eljárás LÉPTET
        Ha (nem VÉGE) akkor                   /ha nem az utolsó elemen állunk(van következő);
                ELŐZŐ:=AKTUÁLIS;              /a most aktuális lesz az előző elem;
                AKTUÁLIS:=KÖVETKEZŐ;          /az aktuális pedig az őt követő elem lesz;
                KÖVETKEZŐ:= (*AKTUÁLIS).köv;
        Elágazás vége
Eljárás vége

Eljárás BEOLVAS                               /a teljes lista beolvasása;
        ELSŐRE;                               /mivel az első elemtől kezdünk;
        Ciklus amíg (nem VÉGE)                /megyünk a legutolsó listaelemig;
                ELŐZŐ:=AKTUÁLIS;              /a most aktuális lesz az előző elem;
                AKTUÁLIS:=KÖVETKEZŐ;          /az aktuális pedig az őt követő elem lesz;
                KÖVETKEZŐ:= (*AKTUÁLIS).köv;  /(ua. mint a léptetésnél, csak itt nem egyet
        Ciklus vége                           /lépünk);
Eljárás cége

Függvény ÉRTÉK:string
        Ha (AKTUÁLIS <> NIL) akkor            /ha van mutatónk
                ÉRTÉK:=(*AKTUÁLIS).adat;      /akkor az aktuális listaelem értékét
        Különben                              /vesszük ki, különben az Érték függvény
                ÉRTÉK:=NULL;                  /adjon vissza üres értéket;
        Elágazás vége
Függvény vége
```




```
Eljárás TÖRÖL
    segéd:=AKTUÁLIS;                       /a segéd változót a memória felszabadításához használjuk;
    Ha (nem VÉGE) akkor                    /ha nem az utolsó elemen állunk
        Ha (AKTUÁLIS = ELEJE) akkor        /ha az első elemet töröljük;
            ELEJE:=KÖVETKEZŐ;              /a lista eleje a következő elem lesz;
        Különben                           /ha nem az első elem;
            (*ELŐZŐ).köv:=(*AKTUÁLIS).köv; /előző elem következő mezője mutasson
        Elágazás vége                      /a törlendő elem utánira, majd az aktuális
        AKTUÁLIS:=KÖVETKEZŐ;               /is legyen az, majd a következő legyen a
        KÖVETKEZŐ:= (*AKTUÁLIS).köv;       /mostani aktuális követő;
    Különben                               /ha az utolsó elemen állunk;
        Ha (ELŐZŐ = AKTUÁLIS) akkor        /vizsgálat: ha már töröltük az utolsót, és újra
            BEOLVAS;                       /meghívtuk a törlési eljárást, beolvassuk a listát,
        Elágazás vége                      hogy ELŐZŐNEK legyen használható értéke;
        AKTUÁLIS:=ELŐZŐ;                   /visszalépünk egyet;
        (*AKTUÁLIS).köv:=NIL;              /és az aktuális elem mutatóját töröljük,
    Elágazás vége                          /így most ez az utolsó;
    MEMSZAB(segéd);                        /a segéd változóba tett utolsó elemet
Eljárás vége                               /töröljük, a "következő" változót
                                           /nem bántjuk, értéke úgyis NIL;
```

Problémát jelent a törlésnél az, hogy esetleg többször is meghívják a törlési eljárást egymás után. Ez az ELŐZŐ mutató rossz értéke miatt (ELŐZŐ = AKTUÁLIS!), hibát okozhat. Ez az eset rávilágít a lista legfontosabb tulajdonságára, hogy egyirányú, ezért szekvenciálisan olvasható csak be. Újraolvasással segíthetünk a gondon.




```
Függvény BESZÚR(adat:string):L                        /adat felvitele a listába;
PÚJ:mutató                                            /függvényváltozó deklarálása;
PÚJ:=MEMFOG(listaelemméret);                          /le kell foglalni a helyet az új adat részére;
Ha (PÚJ <> NIL) akkor                                 /ha sikerült a lefoglalás;
    Ha ((ÜRES) VAGY (AKTUÁLIS = ELEJE)) akkor         /ha üres a lista vagy az
        ELEJE:=PÚJ;                                   /elején állunk, az elejére szúrunk be;
    Különben
        Ha (ELŐZŐ = AKTUÁLIS) akkor                   /ha töröltük az utolsót, és visszaléptünk egyet
            BEOLVAS;                                  /újra beolvassuk a listát, hogy ELŐZŐNEK
        Elágazás vége                                 /legyen használható értéke;
    (*ELŐZŐ).köv:=PÚJ;                                /az aktuális elé szúrunk be;
    Elágazás vége                                     /az új elem következő mezője mutasson az
    (*PÚJ).köv:=AKTUÁLIS;                             /aktuálisra;
    KÖVETKEZŐ:= AKTUÁLIS                              /következő legyen a mostani aktuális elem;
    AKTUÁLIS:=PÚJ;                                    /aktuálissá tesszük az új elemet;
    (*AKTUÁLIS).adat:=adat;                           /beletesszük az adat részt;
    BESZÚR:=igaz;                                     /a beszúr függvény igaz értéket ad vissza;
Különben
    BESZÚR:=hamis;                                    /nem sikerült a lefoglalás, hamis a visszatérési
Elágazás vége                                         /érték;
Függény vége
```

```
Eljárás TÖRÖLAH(honnan:Z+)              /adott helyről való törlés, a honnan tartalmazza a törlendő indexét;
van:logikai                             /logikai változó annak vizsgálatára hogy van-e ilyen elem;
van:=igaz;                              /abban bízunk, hogy van;
Ha (honnan > n) akkor                   /ha nagyobb indexet adunk meg mint a tömbelem száma,
    van:=hamis;                         /az tömbön túli indexelés, nincs ilyen elem;
Különben
    ELSŐRE;                                         /a lista elejére állunk;
    i:=1;                                           /index legyen az első elemindex;
    Ciklus amíg ((i < honnan) ÉS (van = igaz))      /ciklus amíg el nem érjük a keresett indexet;
        Ha ((VÉGE) ÉS (i <> honnan)) akkor          /ha elértük a lista végét és nem találtuk meg
            van:=hamis;                             /a keresett indexet, nincs ilyen listaelem;
        Különben
            LÉPTET;         /léptetünk;
            i:=i + 1;       /ciklusváltozó inkrementálása;
        Elágazás vége
    Ciklus vége
Elágazás vége
Ha (van = hamis) akkor                  /ha nem találtunk ilyen indexet;
    Ki('Nincs ilyen elemindex');        /kiírás;
Különben                                /ha megtaláltuk a keresett elemet (honnan);
    TÖRÖL;                              /törölhetünk, hiszen az i változóval pontosan beállítottuk,
Elágazás vége                           /hogy honnan szeretnénk törölni;
Eljárás vége
```

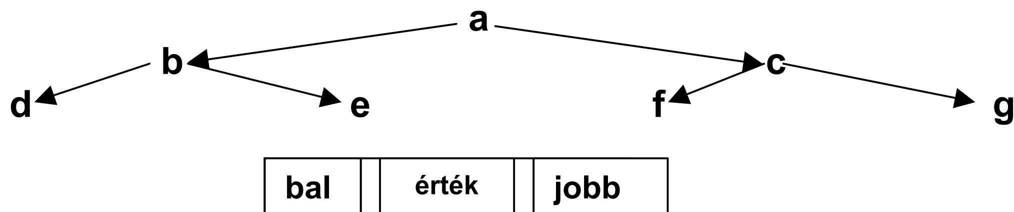
```
Eljárás BESZÚRAH(hova:Z+, adat:string)     /adott helyre való beszúrás;
van:logikai                                /logikai változó annak vizsgálatára hogy van-e ilyen elem;
van:=igaz;                                 /abban bízunk, hogy van;
Ha (hova > n) akkor                        /ha nincs ilyen tömbelemindex (túlindexelünk a tömbön);
        van:=hamis;                        /van értéke legyen hamis;
Különben
        ELSŐRE;                            /a lista elejére állunk;
        i:=1;                              /ciklusváltozó inicializálása;
        Ciklus amíg ((i < hova) ÉS (van = igaz))  /ciklus amíg meg nem találjuk, ahova be akarunk szúrni;
          Ha ((VÉGE) ÉS (i <> hova)) akkor /ha elértük a lista végét és nem találtuk meg
                        van:=hamis;        /a keresett indexet, nincs ilyen listaelem;
          Különben
                        LÉPTET;            /léptetünk;
                        i:=i + 1;          /inkrementáljuk a ciklusváltozót (WHILE!);
          Elágazás vége
        Ciklus vége
Elágazás vége
Ha (van = hamis) akkor                     /ha nincs ilyen elem (hova);
        Ki('Nincs ilyen elemindex');       /kiírás;
Különben                                   /ha megtaláltuk, beállítottuk a helyet, tehát beszúrunk,
        HA (nem BESZÚR(adat)) akkor        /az adat a hívó eljárás "adat" paramétere;
          Ki('Nem sikerült a memóriafoglalás');  /ha a BESZÚR függvény hamis értéket ad vissza
        Elágazás vége                      /a többi feltétel teljesült, de nem tudtuk lefoglalni a
Elágazás vége                              /szükséges memóriaterületet;
Eljárás vége
```

Mind az adott helyről való törlésnél, mind a beszúrásnál le kell fednünk azt a problémát, hogy esetleg olyan listaelemre hivatkozunk, ami nem létezik. **N** a tömbelemek maximális száma, ami azonban nem egyezik meg a listaelemek tényleges számával (dinamikus memóriafoglalás!), így vizsgálnunk kell, hogy nem adnak-e meg ezeknél nagyobb indexet. A beszúrásnál még egy vizsgálatot kell végeznünk, mégpedig, hogy sikerült-e a memóriafoglalás az új elemnek.

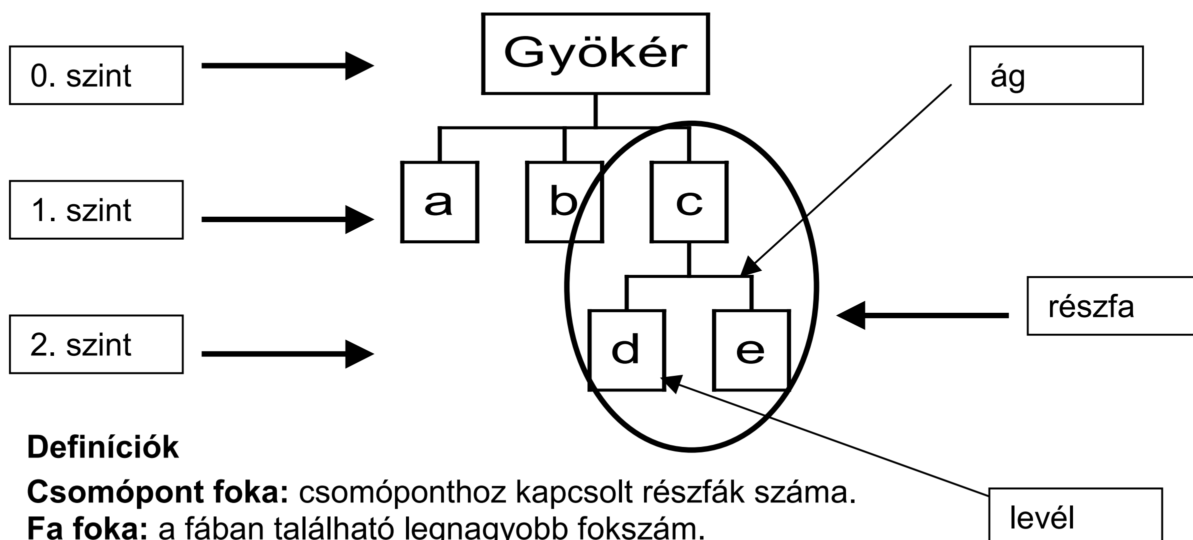
Adattárolási módszerek IV.

Bináris fa

A bináris fa a számítástechnikában igen gyakran használt szerkezet, hiszen hierarchikus összefüggéseket tudunk a segítségével tárolni. Olyan multilistaként képzelhető el, melyben minden elemhez két mutató tartozik, a jobboldali alsó és a baloldali alsó kapcsolódó elemhez tartozó.



A fa egy hierarchikus adatszerkezet amely véges számú csomópontból áll, és igazak rá a következők:
- Két csomópont között a kapcsolat egyirányú, az egyik a kezdőpont a másik a végpont
- Van a fának egy kitüntetett csomópontja, mely nem lehet végpont. Ez a fa gyökere.
- Az összes többi csomópont pontosan egyszer végpont.
- A gyökérhez 0 vagy több diszjunkt fa kapcsolódik. Ezek a gyökérhez tartozó részfák.



Definíciók

Csomópont foka: csomóponthoz kapcsolt részfák száma.
Fa foka: a fában található legnagyobb fokszám.
Levél: 0 fokú csomópont.
Szülő (ős): kapcsolat kezdőpontja (csak a levelek nem szülők).
Gyerek: kapcsolat végpontja (csak a gyökér nem gyerek).
Szintszám: gyökértől mért távolság. A gyökér szintszáma 0.
Fa magassága: a levelekhez vezető utak közül a leghosszabb, a maximális szintszám.
Bináris fa: fokszáma 2, gyökeréből maximálisan két részfa ágazik el: bal és jobboldali részfa.

Rendezetlen bináris fa

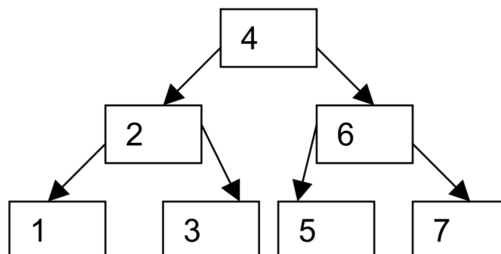
A rendezetlen bináris fa egy egyszerű multilista. Az elemek a fában nem rendezettek, semmilyen szabály szerint nem állapítható meg, hogy egy adott elem, a fában hol található. Kezelése (módosítás, törlés, beszúrás) nagyon gyors, de a keresése, "bejárása", lassú főleg nagy elemszám esetén.

Rendezett bináris fa (keresőfa)

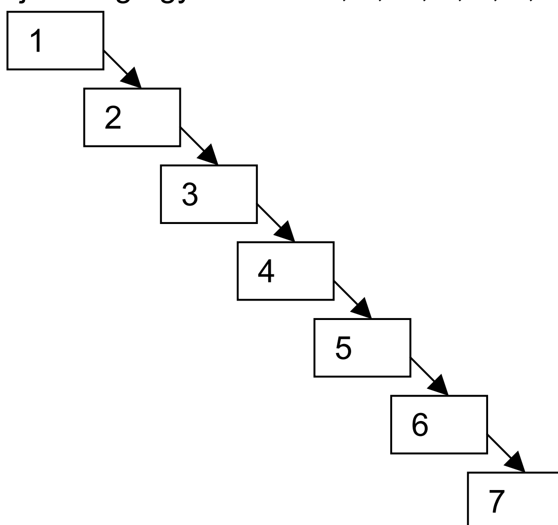
A rendezett bináris fa megfelel a következő szabályoknak:
- A baloldali részfa összes eleme kisebb, mint a szülő.
- A jobboldali részfa összes eleme nagyobb vagy egyenlő, mint a szülő.
- Egy n szintű fa elemeinek száma maximum 2^{n+1}-1.
- Egy elemet maximum n+1 lépésben megtalálunk.

Mindezt úgy érjük el, hogy már az elemek felfűzésekor ("beszúrásakor") figyelünk arra, hogy az elem jó helyre kerüljön. Minden esetben levélként szúrjuk be, oly módon, hogy megkeressük az elem helyét, vagyis a gyökértől kezdve végiglépdelünk a csomópontokon, oly módon, hogy ha az adott csomópont értéke kisebb, mint a beszúrandó elemé balra megyünk, ha nem jobbra. Ezt a stratégiát folytatva érjük el a fa alját, a leveleket, amihez hozzáfűzhetjük az elemünket.

Példa
Fűzzük fel a 4, 2, 6, 3, 1, 7, 5 számokat egy bináris fára!



Próbáljuk meg ugyanezt az 1, 2, 3 ,4 ,5 ,6 ,7 sorrenddel!

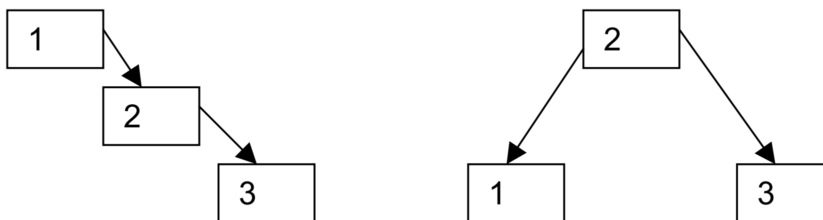


A második esetben a fa egyszerű lineáris listává fajult, degenerált fa vált belőle. Az ilyen típusú, ún. kiegyensúlyozatlan fában történő keresés még a listában történő keresésnél is rosszabb hatásfokú. Míg a kiegyensúlyozott fában maximum Log_2N lépésben megtaláljuk a keresett elemet, addig a kiegyensúlyozatlan fában a legrosszabb esethez tartozó lépésszám maga N (az elemek száma). Vagyis ha az elemek száma 1 milliárd, kiegyensúlyozott keresőfa esetén bármely elemet megtalálunk 30 lépésben (2^{30}=1 073 741 824), míg a legrosszabb eset az elemek száma vagyis: 1 000 000 000 lépés.

Kiegyensúlyozott bináris fák (AVL-fák, vörös-fekete fák)

Az általános bináris keresőfák hátránya az, hogy elfajulhatnak. Mi a megoldás? A kiegyensúlyozás. A magasság szerinti kiegyensúlyozás középút a kiegyensúlyozatlan és a tökéletesen kiegyensúlyozott fák között. Csak annyira van kiegyensúlyozva, hogy a karbantartás még elviselhető, de a keresés már hatékony legyen. E fák elméletét Adelson-Velskii és Landis vezették be, ezért nevezik ezeket a fákat **AVL-fák**nak.

Egy bináris fa akkor kiegyensúlyozott magasság szerint, ha bármely csomópontra igaz, hogy jobb és a baloldali részfájának magassága közötti különbség <=1. Az ilyen fákat forgatással tudjuk egyensúlyban tartani.



Ez egy egyszeres balra forgatás, vagyis minden elem balra mozdult el az eredeti helyéről. Attól függően forgatunk balra vagy jobbra, hogy melyik részfa magassága a nagyobb. Sok esetben egy forgatással nem tudjuk kiegyensúlyozni a részfát ilyenkor kettős forgatással rotáljuk az elemeket.

Az AVL-fák magassága legfeljebb 1,45-szerese az optimális bináris keresőfa magasságának (max. 45%-al magasabb). Viszont a forgatások száma elvileg elérheti a fa magasságát is.

A **vörös-fekete fák** szintén kiegyensúlyozott bináris keresőfák. Mindegyik pontot ki kell egészíteni egy bitnyi kísérő információval, ami az adott pont színét adja meg. Az előre magadott szabályok szerinti adatfelvitelkor a fa kiegyensúlyozottként jön létre. A vörös-fekete tulajdonság a következő:

- Minden pont színe vagy vörös vagy fekete.
- Minden NIL levél színe fekete.
- Minden vörös pontnak mindkét fia fekete.
- Bármely két, azonos pontból induló és levélig vezető úton ugyanannyi fekete pont van.

A szabályok betartásához itt is szükség van a lokális forgatásokra.

A továbbiakban egy kiegyensúlyozatlan, bináris keresőfa kezeléséhez szükséges algoritmusokat adjuk meg.

Globális változók

```
Rekord Faelem                     /önhivatkozó struktúra
    adat:string
    jobb:mutató                   /rekordra mutató mutató
    bal:mutató
Rekord vége
Adatok[1..n]:Faelem               /dinamikusan kezelt tömb

ELEJE: mutató                     /a legelső elemre mutat (gyökérre);
AKTUÁLIS: mutató                  /aktuális elemre mutat;
SZÜLŐ: mutató                     /az aktuális elem szülőjére mutat;

Eljárás GYÖKÉR
    AKTUÁLIS:=ELEJE;              /a legelső elem lesz az aktuális;
    SZÜLŐ:=NIL;                   /az első elemnek nincs szülője;
Eljárás vége

Függvény VÉGE:L                                                  /visszatérési érték logikai;
  VÉGE:=((*AKTUÁLIS).bal=NIL és (*AKTUÁLIS).jobb=NIL);  /nincs elem amire aktuális mutatna;
Függvény vége

Eljárás BEJÁRÁS
    GYÖKÉR;
    Eljárás BEJÁR(AKTUÁLIS:mutató)
        Ha (nem VÉGE) akkor                  /ha nem a végén vagyunk;
            Ki((*AKTUÁLIS).adat);            /adatkiírás;
            BEJÁR((*AKTUÁLIS).bal);          /rekurzívan meghívjuk a függvényt, addig
            BEJÁR((*AKTUÁLIS).jobb);         /megyünk balra míg el nem érjük a fa
        Különben                             /végét, majd jobbra folytatjuk;
            Ki((*AKTUÁLIS).adat);
        Elágazás vége
    Eljárás vége
Eljárás vége
(A gyökértől kezdve a kiírás sorrendje: a; b; d; e; c; f; g)
```

A bináris fa bejárási stratégiái:

- Gyökérkezdő (preorder): gyökér, bal részfa, jobb részfa.
    - (A gyökértől kezdve a kiírás sorrendje: a; b; d; e; c; f; g)
- Gyökérközepű (inorder): bal részfa, gyökér, jobb részfa.
    - (A gyökértől kezdve a kiírás sorrendje: d; b; e; a; f; c; g)
- Gyökérvégző (postorder): bal részfa, jobb részfa, gyökér.
    - (A gyökértől kezdve a kiírás sorrendje: d; e; b; f; g; c; a)

```
Eljárás LÉPTET
    Ha (nem VÉGE) akkor
        SZÜLŐ:=AKTUÁLIS;                        /a mostani aktuális legyen a szülő;
        Ha( (*AKTUÁLIS).bal <> NIL) akkor       /ha van balra elem,
            AKTUÁLIS:=(*AKTUÁLIS).bal;          /legyen ő az aktuális;
        Különben
            AKTUÁLIS:=(*AKTUÁLIS).jobb;         /ha nincs, a jobb lesz az;
        Elágazás vége
    Különben
        semmi;
    Elágazás vége
Eljárás vége
```

```
Eljárás TÖRÖL
TÖRLENDŐ:mutató                              /segédváltozó deklarációja;
        GYÖKÉR;
        Ciklus amíg (nem VÉGE)               /léptetünk, a végéig;
                LÉPTET;
        Ciklus vége
        TÖRLENDŐ:=AKTUÁLIS;                  /a törlendő változóba betesszük az aktuálisat;
        AKTUÁLIS:=SZÜLŐ;                     /visszalépünk egy szintet;
        Ha (TÖRLENDŐ=(*AKTUÁLIS).bal) akkor  /megvizsgáljuk, hogy a törlendő a szülő melyik
                (*AKTUÁLIS).bal:=NIL;        /mutatójához kapcsolódik, majd azt a mutatót
        Különben                             /töröljük;
                (*AKTUÁLIS).jobb:=NIL;
        Elágazás vége
        MEMSZAB(TÖRLENDŐ)                    /felszabadítjuk az elem által lefoglalt memóriát;
Eljárás vége
```

Levél törlésénél nem kell foglalkozni a rendezettség megtartásával, mindegy hogy melyik levelet töröljük. Így nem is érdekel minket, hogy milyen érték volt a törölt elemben. Ha a fenti eljárást ciklikusan meghívjuk (amíg **ELEJE <> NIL**), le tudjuk törölni a teljes fát.

```
Eljárás BESZÚR(adat:String)                  /a fa végére történő elemfelvitel;
PÚJ:mutató
PÚJ:=MEMFOG(elemméret)                       /a listából ismert memóriafoglalás;
Ha (PÚJ <> NIL) akkor                        /ha sikerült lefoglalni;
        Ha (ELEJE=NIL) akkor                 /ha üres a fa,
                ELEJE:=PÚJ;                  /az első elem lesz az új ( a gyökér);
                GYÖKÉR;                      /majd aktuálissá tesszük;
        Különben                             /ha a fa nem üres;
                GYÖKÉR;                      /az elejére lépünk;
                Ciklus amíg (AKTUÁLIS <> NIL) /amíg az aktuális nem értéktelen;
                        SZÜLŐ:=AKTUÁLIS;     /léptetés:
                        HA (adat < (*AKTUÁLIS).adat)     /ha az adat kisebb mint az akt.adat,
                           AKTUÁLIS:=(*AKTUÁLIS).bal;    /akkor balra lépünk,
                        Különben
                           AKTUÁLIS:=(*AKTUÁLIS).jobb;   /különben jobbra;
                        Elágazás vége
                Ciklus vége                  /ha vége a ciklusnak, leértünk a fa aljára;
                Ha (adat < (*SZÜLŐ).adat) akkor /AKTUÁLIS=NIL, szülőt vizsgáljuk;
                        (*SZÜLŐ).bal:=PÚJ;   /ha az adat kisebb mint a szülő adata,
                Különben                     /balról csatlakozik;
                        (*SZÜLŐ).jobb:=PÚJ;  /ha nem, jobbról;
                Elágazás vége
        Elágazás vége
                AKTUÁLIS:=PÚJ;               /az aktuális legyen az új elem;
                (*AKTUÁLIS).adat:=adat;      /és rakjuk bele az adatot;
                (*AKTUÁLIS).jobb:=NIL;       /értéktelenné tesszük az újdonsült levél mutatóit
                (*AKTUÁLIS).bal:=NIL;
        Különben
                Ki('Nem sikerült a memóriafoglalás'); /nem sikerült a memóriafoglalás (pl. nincs
        Elágazás vége                        /elég memória);
Eljárás vége
```

Beszúrásnál mindig a fa végére szúrunk be (levélként), új elemet. Látható, hogy beszúrásnál azt a szabályt követjük, hogy a baloldalon lévő elem mindig kisebb, mint a szülő, míg jobboldali nagyobb vagy egyenlő mint a szülő. Így egy n szintű fa elemeinek a száma maximum 2^{n+1}-1, és egy elemet n lépésben biztos megtalálunk.

```
Függvény KERES(adat:string):L                          /elem keresése a bináris fában
van: logikai                                           /logikai segédváltozó;
        van:=hamis;                                    /logikai változó beállítása (inicializálás);
        GYÖKÉR;                                        /a fa elejére lépünk;
        Ciklus amíg ((van = hamis) ÉS (AKTUÁLIS <> NIL)) /keresünk amíg meg nem találjuk és
                Ha (adat = (*AKTUÁLIS).adat) akkor     /nem érünk a végére, ha megtaláltuk
                        van:=igaz;                     /akkor igazra állítjuk a segédváltozót;
                Különben                               /különben léptetünk;
                        SZÜLŐ:=AKTUÁLIS;
                        Ha (adat < (*AKTUÁLIS).adat) akkor   /az adat értékétől függően;
                           AKTUÁLIS:=(*AKTUÁLIS).bal;   /balra
                        Különben
                           AKTUÁLIS:=(*AKTUÁLIS).jobb;  /vagy jobbra;
                        Elágazás vége
                Elágazás vége
        Ciklus vége
        KERES:=van;                                    /KERES függvény adja vissza a
Függvény vége                                          /segédváltozó értékét;
```

Ha a fa rendezett a keresés hatékony, hiszen ugyanazzal a módszerrel keresünk, mint ahogy feltöltöttük az adatszerkezetet.

```
Eljárás BESZÚRAH(adat:string, keresett:string)     /adott helyre történő beszúrás;
PÚJ:mutató                                         /beszúrandó elem mutatója;
SEGÉD:mutató                                       /segédváltozó;
PÚJ:=MEMFOG(elemméret);                            /memóriafoglalás;
Ha (PÚJ <> NIL) akkor                              /ha sikeres
        Ha (KERES(keresett) = hamis) akkor         /elkezdjük a keresést, ami ha hamis
                Ki('Nincs ilyen elem');            /értéket ad vissza, akkor nincs ilyen elem;
        Különben                                   /különben az aktuális a keresett elem;
                Ha (adat < (*AKTUÁLIS).adat) akkor /az aktuális után szúrunk be megtartva a
                        SEGÉD:=(*AKTUÁLIS).bal;    /"rendezettséget", elmentjük azt a mutatót
                        (*AKTUÁLIS).bal:=PÚJ;      /amit elfoglalunk, majd felkapcsoljuk
                Különben                           /az új elemet;
                        SEGÉD:=(*AKTUÁLIS).jobb;
                        (*AKTUÁLIS).jobb:=PÚJ;
                Elágazás vége
                SZÜLŐ:=AKTUÁLIS;
                AKTUÁLIS:=PÚJ;                             /az aktuális legyen az új elem
                (*AKTUÁLIS).adat:=adat;                    /rakjuk bele az adatot;
                Ha ((*SEGÉD).adat < (*AKTUÁLIS).adat)      /kapcsoljuk fel a segédbe lementett
                        (*AKTUÁLIS).bal:=SEGÉD;            /elemet, értéke szerint az új elemre;
                Különben
                        (*AKTUÁLIS).jobb:=SEGÉD;
                Elágazás vége
        Elágazás vége
Különben
        Ki('Nem sikerült a memóriafoglalás');
Elágazás vége
Eljárás vége
```

Az adott helyre való beszúrásra illik az "erőltetett beszúrás" jelző, mivel eltér az eddig alkalmazott beszúrási sémától. Valójában az adott helyre történő beszúrást a bináris fánál nem alkalmazzák, mivel nem hatékony, hiszen annak eldöntése, hogy egy adatot az kiválasztott helyre be lehet-e szúrni anélkül, hogy megsértenénk a rendezettséget, csak az adott fa ismeretében lehetséges. Emiatt először be kellene olvasni a teljes fát, és csak ezután lehetne eldönteni azt, hogy érdemes-e (növeli-e a hatékonyságot) egy adott (és melyik) helyre beszúrni az új elemet, vagy jobban járunk, ha levélként fűzzük hozzá az adatszerkezethez. Jelen jegyzetben azért kapott helyet az algoritmus, hogy össze lehessen hasonlítani a lista hasonló eljárásával.

```
Eljárás TÖRÖLAH(adat:string)                    /adott helyről történő törlés;
HELY:mutató                                     /segédváltozók deklarálása;
MAX:mutató
Ha(KERES(adat) = hamis) akkor                   /ha nincs ilyen elem
      Ki('Nincs ilyen elem');                   /ezt közöljük;
Különben                                        /különben a keresett az aktuális;
      Ha ((*AKTUÁLIS).bal = NIL) akkor          /ha nincs balra elem,
            HELY:=(*AKTUÁLIS).jobb;             /a törlés után a jobboldali lép előre;
      Különben
            HELY:=(*AKTUÁLIS).bal;              /különben a bal;
            MAX:=HELY;
            Ciklus amíg ((*MAX).jobb <> NIL)    /ilyenkor lemegyünk a baloldali ág aljára;
                  MAX:=(*MAX).jobb;
            Ciklus vége
            (*MAX).jobb:=(*AKTUÁLIS).jobb;      /erre kapcsoljuk fel a jobboldali ágat;
      Elágazás vége
      Ha (AKTUÁLIS = ELEJE) akkor               /ha a keresett elem a gyökér,
            ELEJE:=HELY;                        /a hely változóba elmentett legyen az;
      Különben Ha ((*SZÜLŐ).bal = AKTUÁLIS) akkor
                  (*SZÜLŐ).bal=HELY;            /különben elfoglalja az aktuális helyét
            Különben                            /a szülőnél;
                  (*SZÜLŐ).jobb:=HELY;
            Elágazás vége
      Elágazás vége
      MEMSZAB(AKTUÁLIS);                        /felszabadítjuk a memóriaterületet;
      AKTUÁLIS:=HELY;                           /aktuálissá tesszük az előrelépőt;
Elágazás vége
Eljárás vége
```

Az elemek adott helyről való törlése igen gyakori probléma a bináris fánál. Míg az nem okoz gondot, hogy egy új elemet levélként az adatszerkezet végéhez fűzünk, hisz a rendezettség megmarad és az új elem is bekerül a fába, addig nem mindig leveleket akarunk törölni. Törlésnél ha a törlendő elemnek van baloldali ága, akkor a baloldali ág lép a törlendő helyére, egyébként a jobboldali. Ha a baloldali ág lép eggyel feljebb, akkor ez az ág örökli a jobboldali ágat, vagyis a legnagyobb eleméhez - a jobboldali legalsó levélhez - fog kapcsolódni az egész jobboldali ág.

Algoritmustervezési elvek

Az algoritmustervezési elvek, olyan módszerek, melyek bonyolultabb feladatok, problémák megoldásához, algoritmizálásához nyújtanak segítséget. Használatukkal kiválaszthatjuk az adott probléma megoldásának legoptimálisabb (legjobb munka/teljesítmény arányú) megoldási módozatát.

Elemenkénti feldolgozás

Elemenkénti feldolgozásról beszélünk, ha egy vagy több halmazból kell egy vagy több halmazt előállítani. Mindegyik esetben az előállítandó halmazok elemeit olyan szempontból kell kiválogatnunk (előállítanunk), melynél a kiindulási halmaz(ok) egy elem elegendő annak eldöntésére, hogy az elem bekerül-e az eredményhalmazba, vagy sem. Ez független a halmaz(ok) többi elemétől. Az ilyen feladatokat elemenként feldolgozható feladatoknak nevezzük.

Ezeket több csoportba sorolhatjuk:

- Egy halmazból egy halmazt állítunk elő
- Egy halmazból két vagy több halmazt állítunk elő
- Két vagy több halmazból egy halmazt állítunk elő
- Két vagy több halmazból két vagy több halmazt állítunk elő

Egy halmazból egy halmaz előállítása

Sorravesszük x halmaz elemeit, és a meghatározott tulajdonságúakat hozzávesszük y halmazhoz és elhagyjuk x halmazból. Így x mindig csak azokat az elemek fogja tartalmazni amiket még nem dolgoztunk fel. A megoldó algoritmus ciklus lesz. Kezdetben amikor még egy elemet sem dolgoztunk fel y üres. A feldolgozás végére x üres, y pedig tartalmazza a meghatározott tulajdonságú elemeit x-nek.
Példa: Egy osztálynévsorból válogassuk ki a lánytanulókat!

Egy halmazból több halmaz előállítása

Az x halmazból kiválasztott elemet mindkét feltétel szerint fel kell dolgozni, és a megfelelő (y vagy z) halmazba kell elhelyezni. A feldolgozás végére x üres, a részhalmazok pedig tartalmazzák a feltételüknek megfelelő elemeket.
Példa: Egy osztálynévsorból válogassuk külön a lányokat és a fiúkat!

Két vagy több halmazból egy halmaz előállítása

A két halmaz uniójának ($x \cup y$) elemeit kell sorravenni (ha egy elem mindkét halmazban szerepel, csak egyszer kell vizsgálni), és a feltételnek megfelelően elhelyezni z halmazban. A feldolgozás végére $x \cup y$ üres z pedig tartalmazza a feltételnek megfelelő elemeket.
Példa: Egy iskola osztályainak névsoraiból válogassuk ki a lányokat!

Két vagy több halmazból két vagy több halmaz előállítása

A két halmaz uniójának ($x \cup y$) elemeit mindkét feltétel szerint fel kell dolgozni, és a megfelelő (z vagy v) halmazba kell elhelyezni. A feldolgozás végére($x \cup y$) üres, a részhalmazok pedig tartalmazzák a feltételüknek megfelelő elemeket.
Példa: Egy iskola osztályainak névsoraiból válogassuk külön a lányokat és a fiúkat!

Mohó algoritmusok

A mohó algoritmusok egy adott lépésnél mindig az optimálisnak tűnő megoldást választják, abban a reményben, hogy ez az optimális megoldáshoz fog vezetni. Abban az esetben használható, ha a lokális optimum választása a globális optimumhoz fog vezetni. Nem mindig ad optimális megoldást, de sok probléma megoldható vele.

Esemény-kiválasztás

Adott események egy S halmaza, amik egy közös erőforrást (pl.: épületet, autót, stb.) kívánnak használni. Minden eseményhez adott egy kezdő és egy befejező időpont (kezdés<=befejezés). Két, i és j esemény kompatibilis, ha az i esemény vége<=a j esemény kezdőidőpontjánál, vagy a j esemény vége<=az i esemény kezdőidőpontjánál. Az esemény-kiválasztási probléma azt jelenti, hogy kiválasztandó a kölcsönösen kompatibilis eseményeknek egy legnagyobb elemszámú halmaza.

Megoldás

1. Az eseményeket a befejező időpont szerint sorba rendezzük
2. Kiválasztjuk a legelső (legelsőnek befejeződő) eseményt.
3. Mindig a leghamarabb befejeződő eseményt választjuk úgy, hogy az eddig választottakkal kompatibilis legyen.

Pénzváltás

Adott egy összeg, és adottak különböző nagyságú címletek. Váltsuk fel az összeget a lehető legkevesebb címlet felhasználásával.

Megoldás

A mohó algoritmus ebben az esetben nem minden bemenő értékre ad helyes megoldást. Ha a mindennapi életben használt címleteket használjuk (pl.: 1,2,5,10,20 Ft), akkor jó működik, de könnyen hozhatunk ellenpéldát is (pl: 10Ft-ot váltsunk fel, ha 6,5,2 Ft-os címleteink vannak. A mohó megoldás 6+2+2 módon váltja fel, holott 5+5 felváltás is lehetséges, amihez csak 2 érme szükséges.) Megfigyelhető, hogy optimális megoldást kapunk, ha a címletek növekvő sorrendben vannak.

Elágazás és korlátozás

A módszer angol neve branch-and-bound. Olyan esetekben szokták alkalmazni, amikor a megoldandó feladat természetes módon vezet egy óriási gyökeres irányított fa teljes, vagy részleges bejárásához. A fát gyakran nevezik a feladat fázisterének. A fa általában olyan hatalmas, hogy nem érdemes (olykor nem is lehetne) minden csúcsát tárolni. Vannak viszont szabályok, amivel egy csúcs leszármazottjait elő tudjuk állítani. Ezt szokásos kifejezéssel generálásnak nevezzük.

A feladat általában azt követeli, hogy a fa igen sok csúcsáról rendelkezzünk információval. Az elágazás és korlátozás módszere ezt az információgyűjtést igyekszik gazdaságosan megszervezni. Ilyen feladat például egy sakkállás értékelése. A cél annak a meghatározása, hogy melyik fél - világos vagy sötét- áll jobban, és mi(k) a helyes lépés(ek). Képzeljünk el ehhez egy fát, aminek a csúcsai a sakkállások, megjelölve azzal, hogy melyik félen van a lépés joga. Egy csúcsból (állásból) és vezet a belőle egyetlen lépéssel megkapható állásokba. Az állás értékének megállapításához az alatta lévő részfa csúcsait kellene értékelni. Ez azonban túl nagy, időigényes vállalkozás lenne. Az elágazás azt jelenti, hogy generáljuk az éppen vizsgált csúcs fiait, bizonyos esetekben korlátos mélységig a további leszármazottait is. A generált csúcsok vizsgálata alapján döntjük el, hogy merre menjünk tovább lefelé a fában.

Ebben fontos szerepet játszanak a korlátozó heurisztikák. Ezek a feladat tulajdonságaiból eredő megfontolások, amelyekkel igyekszünk minél több csúcsot és egyszerre azok leszármazottait is kizárni a további keresésből. A sakk esetén az elágazás azt jelenti, hogy generáljuk a néhány lépésben elérhető állásokat, és azokat valahogy értékeljük. Korlátozó heurisztika lehet, hogy nem nézzük azokat az ágakat, ahol vezért vesztünk. Természetesen a komoly sakkprogramok többféle, a fentinél sokkal finomabb és bonyolultabb korlátozó heurisztikát használnak. A korlátozó heurisztikák alkalmazásával ágakat tudunk levágni a fáról, és csak az esélyes, túlélő irányokkal foglalkozunk tovább.

Dinamikus programozás

A dinamikus programozás éppúgy, mint az oszd meg és uralkodj módszer, a feladatot részfeladatokra való osztással oldja meg. A feladatot felosztja részfeladatokra, és a részfeladat eredményét egy táblázatban tárolja el, és ezáltal elkerüli az ismételt számítást, ha a részfeladat megint felmerül. A dinamikus programozást optimalizálási feladatok megoldására használjuk.

A dinamikus programozás lépései

1. részproblémákra bontás
2. a részproblémák közötti összefüggések meghatározása (rekurzív képlettel)
3. a részproblémák optimális értékének kiszámítása alulról felfelé haladva (táblázat kitöltése)
4. az optimális megoldás megalkotása

Közelítő algoritmusok

Előfordul, hogy egy nehéz algoritmikus problémával kerülünk szembe, de elegendő céljainkhoz az optimum elég jó közelítése is. Bizonyos esetekben igen egyszerű és hatékony módszerekkel kaphatunk az optimálishoz közeli eredményt. Érdemes lehet közelítő módszert használni akkor is, ha a problémára van ugyan gyors módszer, de egy a céloknak megfelelő közelítést még gyorsabban ki lehet számítani.

Ládapakolás

Inputként adottak racionális súlyok, melyek értéke >=0 és <=1. A cél a súlyok elhelyezése minél kevesebb a súlykapacitású ládába.

<u>*Megoldás*</u>

1. megoldás:
FF-módszer (first fit). Vegyünk először üres ládákat, és számozzuk meg őket 1,2,...,k egészekkel. Tegyük fel, hogy 1-től i-1-ig a súlyokat már elhelyeztük. Ekkor az i. súly kerüljön az első olyan ládába, amibe még belefér. Az FF igen gyors és egyszerű módszer, amely eléri az optimum 10/17-ét.

2.megoldás:
Az FFD (first fit decreasing) módszer egy kicsit továbbfejlesztett változat. Először rendezzük a súlyokat csökkenő sorrendbe, majd alkalmazzuk a FF módszert. Ezzel a módszerrel elérjük az optimum 9/11-ét.

Léteznek ezeknél valamivel jobb eredményt adó módszerek is, ezek azonban egyszersmind jóval bonyolultabbak is.

Véletlent használó algoritmusok

Az ilyen algoritmusok furcsa sajátossága, hogy a bemenet és az algoritmus maga nem határozzák meg egyértelműen a tényleges lépéseket, a számítási időt, olykor a végeredményt sem. Mindezekért a bizonytalanságokért azonban kárpótol a véletlent használó módszerek hatékonysága. Egy példával már találkoztunk a gyorsrendezés kapcsán, ahol a gyorsrendezés ideje nagymértékben függ attól, hogy a particionáló elemek mennyire vágják egyforma darabokra a tömböt. Gyakorlati szempontból a gyorsrendezés randomizált változata a legjobb az ismert rendező módszerek közül.

A randomizált módszerekkel kapcsolatban két érvet szokás felhozni. Az egyik a bizonytalanság, a hiba lehetősége, amely mintegy bele van tervezve ezekbe a módszerekbe. A másik érv jóval súlyosabb. A randomizált módszerek elemzésénél feltesszük, hogy valamiféle "igazán véletlen" bitek kellő hosszú sorozatával rendelkezünk, mégpedig általában nulla költséggel. Valójában nem világos, hogy mit tekinthetünk véletlen sorozatnak, és a még kevésbé, hogy nyerhetünk-e ilyet egyáltalán valamilyen természeti folyamatból. Ezek azonban filozófiába nyúló kérdések, amelyek izgalmasak ugyan, de kevés algoritmikus tanulsággal szolgálnak

Vegyes Feladatok

1. Írjunk algoritmust, ami megadja két szám legnagyobb közös osztóját!
2. Írjunk algoritmust, ami megadja két szám legkisebb közös többszörösét!
3. Készítsünk algoritmust, ami megadja két négyjegyű szám maszkjának értékét (azt, hogy a két szám hány helyi értéken egyezik meg)!
4. Írjunk programot, ami évenkénti bontásban kiszámítja és kiírja a betét kamatos kamattal megnövelt összegét. A betét , a kamat és az évek száma bemenő adat!
5. Írjunk algoritmust, ami feltölti a szorzótáblát!
6. Állapítsuk meg, hogy egy n*n-es mátrix szimmetrikus-e? A szimmetria feltétele minden mátrixelemre: Tömb[i,j] = Tömb[j,i].
7. Írjunk algoritmust, ami kiszámolja megadott értékekre a permutáció és a variáció értékét! A feladatot alprogramokkal oldjuk meg!
8. Rendezzük egy m*n-es pozitív egész számokat tartalmazó mátrix elemeit növekvő sorrendbe!
9. Adott A[1..m] tömb és B[1..n] tömb, melyek nem tartalmaznak ismétlődő elemeket, így halmazként is felfoghatóak. Adjuk meg azt az adatszerkezetet és az adatszerkezethez tartozó algoritmust, amelyik a két tömbnek mint halmaznak a Descartes-szorzatát tárolja!
10. Adott egy szöveg. Írjunk algoritmust, ami kiírja azokat a szavakat, melyek egynél többször fordulnak elő!

A feladatokhoz készítse el az állapotteret és az algoritmust!

Megoldások

1. fejezet

I.

1. b.
2. b.
3. a.
4. b.
5. b.
6. b.
7. a.

II.

1. igaz
2. igaz
3. hamis
4. hamis
5. igaz
6. hamis
7. igaz

III.

1. Az alacsonyszintű nyelvekben (assembly) a gépi utasításkódokat az angol nyelvi megfelelőik rövidítésével - mnemonikokkal - jelzik.
2. 1960-ban az IBM európai központjában létrehozzák az Algol60-t. Ez az első matematikai értelemben definiált nyelv. Hivatkozási nyelve is van, rendezett fogalomrendszerrel rendelkezik. Az IBM nem támogatja, ennek ellenére nagy jelentőségű, konstrukciója ma is hat. Ekkortól kezdve az Algol60 az algoritmus-leíró szabvány, a későbbi nyelvek belőle indulnak ki, vagy tagadják. Az Algol60 tagadására épülő nyelvek: pl. LISP (a mesterséges intelligencia támogatására fejlesztették ki), APL (eszközrendszere még ma is egyedülálló).
3. Simula67.
4.
    - Első generáció:
        - manuális programozástechnika
        - gépi kód, alacsonyszintű nyelvek használata
    - Második generáció:
        - magas szintű nyelvek használata
        - strukturált programozástechnika megjelenése
    - Harmadik generáció:
        - objektumorientált megközelítés kialakulása
        - objektumorientált nyelvek használata, kiegészítések, "ráépülő" nyelvek kifejlődése
    - Negyedik generáció (4GL):
        - vizuális kezelőfelületű nyelvek megjelenése
        - eseményvezéreltség kialakulása
        - az objektumorientáltság alapvető követelmény

5. Procedurális nyelv, amelyben a programozó adja meg az utasítások végrehajtásának sorrendjét és mikéntjét. Bármely adatszerkezeten bármely nyelvi utasítás végrehajtható a szintaktikai szabályoknak megfelelően. Objektumorientált nyelv, amelyben az adatokat és a rajtuk végrehajtható utasításokat egy egységként kezeljük. A programozó korlátozva van abban, hogy milyen utasításokat hajthat végre az adatokon. Eseményvezérelt nyelv, amely már nem lineáris szerkezetű, az utasítás végrehajtása az objektumhoz tartozó eseményhez kötődik (az utasítás végrehajtás során a programozó korlátozva van abban, hogy hogyan hajtathatja végre azt).
6. Az imperatív (utasításszerkezetű) nyelvek alapeszközei az utasítások és a változók. A program szövege utasítássorozat, minden utasítás mögött gépi kód áll. Minden program utasítássorozat, amely mögött több gépi utasítás áll. Kötődnek a Neumann-architektúrához, általában fordítóprogramosak. Algoritmikus nyelvek, a programban azt az algoritmust írom le, amelyet a gép végrehajt, és a probléma megoldása így születik meg. A program a hatását a tár egyes területein lévő értékeken fejti ki.

   A deklaratív (leírásjellegű) nyelveknél nincs utasításfogalom, a Neumann-architektúrától távol áll. Nem algoritmikusak, a programban csak a problémát fogalmazom meg, a megoldást nem, az algoritmus a rendszerbe van beépítve. A tárhoz a programozónak kevés köze van, nem feladata a tár egyes részeinek módosítása.
7. Adat: A számítógépben jelsorozat formájában tárolt, kódolt információ. A bennünket körülvevő világ objektumainak (tárgyak, dolgok) mérhető és nem mérhető jellemzői. Adat egy tárgy kilogrammban kifejezett értéke, egy ember neve, a ruha színe. Mindegyik egy tulajdonságot jellemez, de tartalmukat tekintve különbözőek. Az adatok jellemzésének egyik módja, hogy megadjuk milyen értékeket vehetnek fel az adott szituációban, és ezekkel milyen műveleteket lehet elvégezni. Egy adat lehetséges értékeinek halmazát típusérték-halmaznak nevezzük. Egy adat típusát három dolog határozza meg. Egyrészt azok az értékek, amelyeket az adat felvehet, a típusérték-halmaz. Másodszor az a szerkezet, ahogyan egy ilyen érték egyszerűbb típusok értékeiből felépül. Harmadszor azoknak a műveteknek az összessége, amit az adott halmazon el lehet végezni.
8. Magas szintű nyelv: a programozó számára könnyebben megfogalmazható, emberközelibb, bővebb utasításkészlettel rendelkező programnyelv. Hordozható, viszonylag gépfüggetlen programok. Sok utasítással rendelkeznek, összetettebb feladatok megvalósítására is képesek.(Pl. Pascal, C, Basic, Delphi, Clipper, LOGO).
9. Az algoritmus egy feladat megoldására szolgáló egyértelműen előírt módon és sorrendben végrehajtandó véges tevékenységsorozat, mely véges idő alatt befejeződik. A tevékenység matematikai művelettől kezdve tetszőleges számítási, gyártási vagy technológiai művelet lehet.
10. A. Forrásprogram megírása, szerkesztése egy szövegszerkesztővel

B. Forrásprogram lefordítása a fordítóprogram segítségével eredménye a tárgykód
C. Gépi kódú futtatható állomány létrehozása a kapcsolatszerkesztő segítségével
D. Program futtatása a betöltő segítségével, hibakeresés

2. fejezet

I.

1. a.
2. b.
3. c.
4. a.
5. c.

II.

1. igaz
2. igaz
3. hamis
4. igaz
5. hamis

III.

1. A megtervezett program megvalósítása valamilyen programozási nyelv felhasználásával. A programnyelv kiválasztásakor figyelembe kell venni a programozási feladat jellegét, a már rendelkezésre álló programrészleteket, és a programot futtató rendszer jellemzőit. Ha a kódolás során elakadunk, akkor a változtatásokat, új megoldási utakat az algoritmustervbe is fel kell vennünk. Ha nagyobb problémával kerülünk szembe, akkor akár a specifikációt is meg kell változtatnunk, és újra kell terveznünk algoritmusaink egy részét. A kódolás folyamata alatt újabb tesztadatokat gyűjthetünk össze. A későbbi könnyebb módosítás és továbbfejlesztés érdekében célszerű megjegyzéseket is tenni a kódba.
2. A dokumentációk közül talán a legfontosabb. Részletesen tartalmaznia kell a program telepítését, indítását, használatát. Ki kell terjednie a program által elvégezhető összes funkcióra. Rendszerint képernyőmintákkal, példákkal illusztrált. Tartalmazza az esetleges hibaüzeneteket és a kapcsolódó hibák elkerülésének, kijavításának lehetőségeit. A dokumentációnak olyan részletesnek kell lennie, hogy a legkevesebb hozzáértéssel is használni lehessen, hiszen ez nem szakembereknek készül elsősorban. Általában kinyomtatott formában megvásárolható, ill. mai követelményeknek megfelelően HTML vagy PDF formátumban olvashatóak.
3. Egyfelhasználós környezetben általában a felhasználói kézikönyv tartalmazza az üzemeltetési, operátori teendőket, mivel ott az nem válik szét. Többfelhasználós környezetben azonban az operátori teendőket leíró dokumentációnak tartalmaznia kell a program telepítésének menetét, szükséges konfigurációját, beállításokat. Leírja a program indítását, paraméterezését. Előírja a szükséges mentések idejét, módját menetét. Az előforduló hibaüzenetek részletes leírását, valamint a hibák megszüntetésének módját.
4. Ha egy program úgy kerül eladásra, hogy annak fejlesztési jogát is megvásárolják, akkor annak elengedhetetlenül tartalmaznia kell egy olyan dokumentációt, melynek birtokában egy másik programozó szükség esetén elvégezheti a módosításokat, hibajavításokat.

5. Egyszerre csak kevés dologról, de azokról következetesen döntünk. A felbontás után keletkező részek lehetőleg egyenlő súlyúak legyenek! Azokat a döntéseket, amelyek az adott szinten nagyon bonyolultnak látszanak, próbáljuk későbbre halasztani, elképzelhető, hogy később egyszerűen megoldhatók lesznek.
6. Általánosan fogalmazzuk meg a feladatot, általános algoritmust és programot készítsünk, így az szélesebb körben, hosszú ideig alkalmazható lesz. A konkrét feladat megfogalmazását kell általánosítani, az adatait pedig paraméterként kezelni.
7. A modulok fajtái
   - adatmodulok
   - eljárásmodulok
   - vezérlőmodulok
   - I/O modulok
8. Az objektumorientált megközelítés az objektumok mint programegységek bevezetésével megszünteti a program kódjának és adatainak szétválasztását. Objektumok használatával egyszerű szerkezetű, jól kézben tartható programok készíthetők. Az objektumorientált programozás középpontjában az egymással kapcsolatban álló programegységek hierarchiájának megtervezése áll.
9. Az objektumorientáltság három fő ismérve

   Egységbezárás (encapsulation)

   Azt takarja, hogy az adatstruktúrákat és az adott struktúrájú adatokat kezelő függvényeket (metódusokat) egy egységként kezelve, az alapelemeket elzárjuk a világ elől. Az így kapott egységek az objektumok.

   Öröklődés (inheritance)

   Azt jelenti, hogy az adott meglévő osztályokból levezetett újabb osztályok öröklik a definiálásukhoz használt alaposztályok már létező adatstruktúráit és metódusait.

   Többrétűség (polimorfizmus)

   Azt értjük ezalatt, hogy egy örökölt metódus az adott objektumpéldányban felüldefiniálódik.
10. Sorozat adattípusok

Halmaz

A halmaz olyan adatszerkezet, melyben egyforma típusú, de különböző, rendezetlen elemek találhatók. A halmaz bármilyen fajtájú, jól meghatározott, egymástól megkülönböztethető dolgok összessége. A halmazban lévő dolgokat a halmaz elemeinek nevezzük. Egy halmazban egy elem csak egyszer szerepel, és az elemeknek nincs sorrendje. Az elemek száma lehet véges vagy végtelen. Halmazt megadhatunk úgy, hogy felsoroljuk az elemeit: H1 = {Pascal, Eiffel, C++, Prolog } és H2 = {1,2,3,4,5,6,7,8,9}.

Sorozat

Nem feltétlenül rögzített az elemek száma. Legismertebb sorozat a számsorozat (intervallum típus) ahol a lépésköz rögzített és amely mindkét irányban végtelen. Természetesen megállapíthatunk zárt intervallumokat is, ilyenkor a sorozat "kvázi-tömbként" viselkedik, annyi eltéréssel, hogy a tömböt explicit módón fel kell tölteni a számsorozat tagjai viszont adottak. A sorozat nagyon rugalmas, léteznek más típusú sorozatok pl. nevek sorozata.

Tömb

Típusérték-halmaza konstans hosszúságú elemeket tartalmaz, az egyes elemekre indexeléssel lehet hivatkozni. Egy tömb megadásakor meg kell adni az egyes

dimenziók irányába eső maximális komponensek számát, tehát a tömb mérete rögzített. Az egydimenziós tömböt vektornak (pl. lottószámok), a kétdimenziós tömböt mátrixnak (órarend, sakktábla) is szokás nevezni. Háromdimenziós tömb pl. az iskola összes órarendje. A tömb bármely elemére hivatkozhatunk úgy, hogy megadjuk az elem sorszámát. A sorszámot a változó neve után szögletes zárójelbe kell tenni. Ezt a sorszámot indexnek, a hivatkozási módszert indexelésnek nevezzük. Arra kell vigyázni, hogy az elemre való hivatkozáskor az indexnek olyan értéke legyen, mely egy létező tömbelemre hivatkozik. Ellenkező esetben a tömbön túli indexelés megállítja a program futását. Léteznek ún. asszociatív vagy "hash" tömbök, ahol az indexet nem a sorszám, hanem egy érték adja, ezek a tömbök kulcs-érték párokból épülnek fel.
String vagy text
Karaktertömb. Két fajtája van a "normál" string azokban a nyelvekben, ahol ez az adattípus külön definiált (Pascal), ilyenkor a hosszát külön tároljuk, ill. a nullvégű string, ami karaktervektor, a végén egy nulla értékű byte-al (C). A karakterlánc típusú változónak illetve konstansnak bármelyik karakterére külön hivatkozhatunk úgy, hogy megadjuk annak sorszámát, ugyanúgy ahogy a normál tömbnél.

3. fejezet

1. Olyan programozási eszközök, amelynek négy komponense van:
    - Név
    - Egyedi azonosító, a program szövegében a változó mindig a nevével jelenik meg, ez hordozza a komponenseket.
    - Attribútumok
    - A változó futás közbeni viselkedését, az általa felvehető értékeket határozzák meg. Az eljárás-orientált nyelvekben a legfontosabb attribútum a típus, nem típusos nyelvekben ilyen komponens nincs, de más attribútum lehetséges. Változóhoz attribútum rendelés deklaráció segítségével történhet.
    - Cím
    - A tár azon területének a címe, ahol az adott változó értéke elhelyezkedik.
    - Érték
    - Az adott tárrészen elhelyezkedő bitkombináció. A típus eldönti, hogy hány byte-on, milyen ábrázolási móddal van ábrázolva a változó, és meghatározza az értékhatárokat.
2. A kifejezés a programnyelvek szintaktikai egysége, az eddigi fogalmak jelennek meg benne. Már ismert értékek alapján új értéket határozunk meg. Olyan objektum, amelynek két komponense van: érték és típus. Típussal csak a típusos nyelvekben rendelkezik. A kifejezések lehetnek matematikaiak vagy logikaiak. Formálisan operandusokból, operátorokból és kerek zárójelekből áll.
3. Egyágú szelekció: ha igaz a megadott feltétel, akkor a hozzá kapcsolódó tevékenységet végre kell hajtani, egyébként azt ki kell kerülni, és a programot az azt követő közös tevékenységgel kell folytatni.
   Kétágú szelekció: ha a kiértékelődés után a kifejezés értéke igaz, akkor a feltétel utáni tevékenység hajtódik végre. Ha az értéke hamis akkor a különben ágban lévő utasításokat hajtja végre. Ezután a program a feltételes utasítás utáni utasításon folytatódik.
   Többirányú szelekció: feladata, hogy a program egy adott pontján akárhány tevékenység közül tudjak egyet választani. A választás általában egy kifejezés

értékei szerint történik, lényeges a kifejezés típusa. A kifejezés kiértékelődik, az értékét a konstanslistához hasonlítja. Ha talál megfelelő ágat, végrehajtja az utasítás(oka)t és kilép az elágazásból. Ha nincs megfelelő ág és van különben ág, a különben ágban lévő utasítást végzi el és kilép, ha nincs különben ág, akkor üres utasítás.

4. Feladata a program adott pontján egy tevékenység egymás utáni többszörös végrehajtása.
   Típusai:
   - előírt lépésszámú (növekményes) ciklus (FOR)
     - Előre tudjuk, hogy hányszor akarjuk elfuttatni a ciklusmagban lévő utasításokat. A ciklusváltozó automatikusan vesz fel értékeket, lépésköz és irány megadása kötelező
   - feltételes ciklus:
     - elöltesztelő ciklus (WHILE)
       - A feltételtől függ, hogy belépünk-e a ciklusba vagy sem.
     - hátultesztelő ciklus (REPEAT UNTIL)
     - A ciklusmag egyszer mindenféleképpen végrehajtódik
   - összetett ciklus
   - végtelen ciklus
     - A megszakítási feltétel a törzsben található.
5. Folyamatábra

Az algoritmus részlépéseit különböző geometriai szimbólumokkal szemlélteti. Az egyes szerkezeti elemek között nyilakkal jelöljük a végrehajtási sorrendet. Az értékadó utasítás illetve az eljárások téglalapba, az elágazások rombuszba vagy lapos hatszögbe, az adatáramlás paralelogrammába, a vezérlő utasítások körbe kerülnek.

Struktogram

Az algoritmust egy téglalapba írjuk be. Ebbe a téglalapba további téglalapokat illesztünk, és a végrehajtandó utasításokat ezekbe írjuk be. Az egyes szerkezeti elemek jól elkülönülnek, a szekvencia az egymásutánisággal, a szelekció az egymásmellé kerüléssel, az iteráció a visszatérési út kijelölésével ábrázolható. Ezek a szerkezetek egymásba ágyazhatóak.

Pszeudokód

Egy megadott programnyelvhez hasonló, de szintaktikailag szabadabb algoritmus leírás. Mondatszerű elemekkel bővíti a nyelv utasításkészletét.

Funkcionális leírás

Az adott algoritmus funkcióinak és ezek hierarchiájának szöveges leírása.

Jackson ábra

Top-down dekompozíciós diagram, ahol az algoritmus feltételei ún. feltételjegyzékbe, az általa végrehajtott tevékenységek pedig tevékenységjegyzékbe kerülnek.

Mondatszerű leírás

Az algoritmust egymás után következő mondatokkal írjuk le. Ma már a pszeudokód és a mondatszerű leírás összemosódott, a fő különbség mégis az, hogy a mondatszerű leírásban a programszerkezetet magyarázó részek a szövegbe kerülnek, míg a pszeudokódnál a magyarázat a használatos nyelv kommentezési szokásai szerint van megjelölve.

4. fejezet

1. Az alprogramok egy problémaosztályt oldanak meg. A probléma akkor konkretizálható, amikor az alprogramot az aktuális paraméterek megadásával meghívom. A fejben található, általában kerek zárójelek között. A nyelvek egy része azt mondja, hogy nem kell a zárójel, ha az alprogramnak nincs formális paramétere (pl. Pascalban), más része szerint pedig a zárójel nem a paraméterekhez, hanem az alprogramhoz tartozik, tehát paraméterek nélkül is ki kell tenni (pl. C). Nyelvfüggő, hogy a paraméterek mivel vannak elválasztva. A paraméterlistán szereplő nevek a törzsben különféle objektumok lehetnek: változók, nevesített konstansok, állománynevek, más alprogramok nevei. Számuk bármennyi lehet, a nulla paraméterrel rendelkező alprogramot paraméter nélküli alprogramnak hívjuk.
2. Hatását paramétereinek, környezetének vagy mindkettőnek megváltoztatásával fejti ki. Adat-transzformációt hajt végre vagy tevékenységet végez. Hívása utasításszerűen történik, végrehajtandó utasításnak tekinthető. A program szövegébe bárhol elhelyezhető eljáráshívás, ahol végrehajtandó utasítás lehet. Egyes nyelvekben külön alapszóval (általában CALL) hívható, más nyelvekben nincs rá alapszó.
3. A matematikai fogalmat hozza át, feladata 1 db érték meghatározása. Nyelvfüggő, hogy ez az érték mennyire bonyolult struktúrájú lehet. Még egy komponense van, a függvény által visszaadott érték (visszatérési érték) típusa. Ezt a függvény neve hordozza, a fejben szerepel, a specifikáció része. Csak kifejezésben hívható meg, mert a függvény neve által hordozott értéket fel kell használni.
4. Formális paraméter listában kell definiálni azt az adatszerkezetet, amely révén a függvény vagy eljárás bemeneti információt kap. A definiált adatszerkezet fiktív, a tényleges adatokat csak a függvény vagy eljárás hívásakor fogjuk előírni. A formális paramétereket a hívó program nem név szerint, hanem a felsorolásban elfoglalt pozíció szerint azonosítja: az első aktuális paraméter az első formális paraméternek, a második aktuális paraméter a második formális paraméternek, a harmadik aktuális paraméter a harmadik formális paraméternek és így tovább felel meg. Aktuális paramétereknek azokat a paramétereket nevezzük, amelyeket a függvény vagy eljárás meghívásakor adunk meg a függvény vagy eljárás neve után. Ezeken hajtódnak végre a függvény vagy eljárás utasításai
5. Változó vagy cím szerinti paraméterátadás: A változóparaméter értékét a hívott program (eljárás, függvény) megváltoztathatja, ekkor a hívó programban használt aktuális paraméter értéke is megváltozik.
   Érték szerinti paraméterátadás: A formális paraméter értékének változása nem hat vissza az aktuális paraméter értékére. A ki-fejezés, utasítás az eljárás vagy függvény aktivizálásakor értékelődik ki, és a megfelelő formális paraméter ezt az értéket kapja meg.
6. Rekurzió: olyan programtevékenység, ahol az eljárások, függvények önmagukat hívják meg. A feladatot esetekre bontjuk, és van olyan eset, amely eset önmagával van megfogalmazva, egy másik n értékkel. Ha ez a folyamat többször ismétlődik, hívási láncról beszélünk. A lánc alaphelyzetben dinamikusan változik, eleje mindig a főprogram. Függetlenül attól, hogy működik-e, minden eleme aktív. Aktív alprogram újra meghívását rekurzív hívásnak (rekurziónak) nevezzük.

5. fejezet

Összegzés

1. *Előfeltétel:* *adott az intervallum 1-től n-1-ig.*
Utófeltétel: *a program írja ki az intervallumba eső számok összegét.*
Deklaráció: *s:N*
i,n:Z+

```
Be(n);
s:=0;
Ciklus i:=1-től n-1-ig
        s:=s + i;
Ciklus vége
Ki(s);
```

2. *Előfeltétel:* *adott az intervallum m-től n-ig.*
Utófeltétel: *a program írja ki az intervallumba eső számok köbeinek összegét.*
Deklaráció: *s:N*
i,m,n:Z

```
Be(m);
Be(n);
s:=0;
Ciklus i:=m-től n-ig
        s:=s + i^3;
Ciklus vége
Ki(s);
```

3. *Előfeltétel:* *adott a 31 elemű Hiányzás tömb.*
Utófeltétel: *a program írja ki a hiányzott órák számát (a Hiányzás tömb elemeinek összegét).*
Deklaráció: *s:N*
i,n:Z+
Hiányzás[1..31]:N

```
s:=0;
Ciklus i:=1-től 31-ig
        s:=s + Hiányzás[i];
Ciklus vége
Ki(s);
```

4. *Előfeltétel:* *adott a Matek tömb.*
Utófeltétel: *a program írja ki a tömb elemeinek átlagértékét.*
Deklaráció: *s:Z+*
i,n:Z+
Matek[1..n]:Z+

```
s:=0;
Ciklus i:=1-től n-ig
        s:=s + Matek[i];
Ciklus vége
Átlag:=s / n;
Ki(átlag);
```

Kiválasztás

1. ***Előfeltétel:*** ***adott N szám.***
Utófeltétel: ***a program adja meg az n-nél nem kisebb (nagyobb vagy egyenlő számok közül a legkisebb prímet.***
Deklaráció: ***n:Z+***
i:Z+

Be(n);
i:=n;
Ciklus amíg (i nem prím)
i:=i + 1;
Ciklus vége
Ki(i);

2. ***Előfeltétel:*** ***adott N szám.***
Utófeltétel: ***a program adja meg az n-nél nem nagyobb (kisebb vagy egyenlő számok közül a legnagyobb prímet.***
Deklaráció: ***n:Z+***
i:Z+

Be(n);
i:=n;
Ciklus amíg (i nem prím)
i:=i - 1;
Ciklus vége
Ki(i);

3. ***Előfeltétel:*** ***adott az osztálynévsor.***
Utófeltétel: ***a program döntse el, hogy Kiss Ibolya hányadik az osztálynévsorban.***
Deklaráció: ***Névsor[1..n]:string***
i:Z+

i:=1;
Ciklus amíg (Névsor[i] <> "Kiss Ibolya")
i:=i + 1;
Ciklus vége
Ki(i);

Megszámlálás

1. ***Előfeltétel:*** ***adott az egész számok egy intervalluma.***
Utófeltétel: ***a program írja ki, hogy az adott intervallumban hány prímszám van.***
Deklaráció: ***m,n,i,db:N***

Be(m);
Be(n);
db:=0;
Ciklus(i:=m-től n-ig
Ha (i prím) akkor
db:=db + 1;
Elágazás vége
Ciklus vége
Ki(db);

2. *Előfeltétel:* *adott a névsor.*
Utófeltétel: *a program adja ki, hogy a névsorban hány Nagy Gábor nevű tanuló szerepel.*
Deklaráció: *i,db:N*
Névsor[1..n]:string

```
db:=0;
Ciklus i:=1-től n-ig
        Ha (Névsor[i] = "Nagy Gábor") akkor
                db:=db + 1;
        Elágazás vége
Ciklus vége
Ki(db);
```

6. fejezet

Eldöntés

1. *Előfeltétel:* *adott a szám.*
Utófeltétel: *a program írja ki, hogy az adott számnak van-e valódi osztója*
Deklaráció: *i,n:Z+*
talált:L

```
Be(n);
talált:=hamis;
i:=1;
Ciklus amíg ((nem talált) ÉS (i < n/2))
        i:=i + 1;
        Ha (n mod i) = 0 akkor
                talált:=igaz;
        Elágazás vége
Ciklus vége
Ha (talált) akkor
        Ki("Van");
Különben
        Ki("Nincs);
Elágazás vége
```

2. *Előfeltétel:* *adott a név és a névsor.*
Utófeltétel: *a program írja ki, hogy az adott név szerepel-e az adott névsorban*
Deklaráció: *i:N*
talált:L
név, Névsor[1..n]: string

```
Be(név);
talált:=hamis;
i:=0;
Ciklus amíg ((nem talált) ÉS (i < n))
        i:=i + 1;
        talált:= (Névsor[i] = név);
Ciklus vége
Ha (talált) akkor
        Ki("Szerepel");
Különben
        Ki("Nem szerepel");
Elágazás vége
```

3. **Előfeltétel:** adott az intervallum.
Utófeltétel: a program írja ki, hogy az intervallumban van-e prímszám.
Deklaráció: i,n:Z+
talált:L

```
Be(m);
Be(n);
talált:=hamis;
i:=m-1;
Ciklus amíg ((nem talált) ÉS (i < n))
        i:=i + 1;
        talált:=i prím;
Ciklus vége
Ha (talált) akkor
        Ki("Van");
Különben
        Ki("Nincs);
Elágazás vége
```

Kiválogatás

1. **Előfeltétel:** adott a névsor.
Utófeltétel: a program válogassa ki a "K" betűvel kezdődő tanulókat.
Deklaráció: j:N
i:Z+
Szöveg, Névsor[1..n], K[1..n]:string

```
j:=0;
Ciklus i:=1-től n-ig
        Szöveg:=Névsor[i];
        Ha (Szöveg[1] = "K") akkor
                j:=j + 1;
                K[j]:=Névsor[i];
        Elágazás vége
Ciklus vége
```

2. **Előfeltétel:** adott a névsor.
Utófeltétel: a program válogassa ki a megadott magasságú személyeket.
Deklaráció: j:N
i:Z+
Rekord Személy
    név:string
    magasság:string
Rekord vége
Névsor[1..n],Magasság[1..n]:Személy

```
Be(magasság);
j:=0;
Ciklus i:=1-től n-ig
        Ha (Névsor[i].magasság = magasság) akkor
                j:=j + 1;
                Magasság[j]:=Névsor[i];
        Elágazás vége
Ciklus vége
```

3. Előfeltétel: adott az intervallum és a szám.
Utófeltétel: a program válogassa ki a megadott számmal oszthatóak.
Deklaráció: j:Z+
m,n:Z
i:Z
Oszt[1..n]:Z

```
Be(m);
Be(n);
Be(szam)
j:=0;
Ciklus i:=m-től n-ig
        Ha (i mod szam = 0) akkor
                j:=j + 1;
                Oszt[j]:=i;
        Elágazás vége
Ciklus vége
```

Maximum-minimum kiválasztás

1. Előfeltétel: adott az intervallum és a függvény.
Utófeltétel: a program határozza meg az intervallumnak azt az elemét, ahol a függvény helyettesítési értéke a legnagyobb.
Deklaráció: i,ind:Z
max:Z

```
i:=-2;
ind:=-2;
max:=f(-2);
        Ciklus amíg (i < 15)
                i:=i + 1;
                Ha (max < f(i)) akkor
                        max:=f(i);
                        ind:=i;
                Különben
                        semmi;
                Elágazás vége
        Ciklus vége
Ki(ind);
Ki(max);
```

2. Előfeltétel: adott az osztály tömb.
Utófeltétel: a program határozza meg annak a gyereknek a nevét, akinek a legtöbb foga van.
Deklaráció: i,ind:N
Rekord Tanuló
név:string
fog:N
Rekord vége
Max,Osztály[1..n]:Tanuló

```
i:=1;
ind:=1;
max:=Osztály[i].fog;
Ciklus amíg (i < n)
        i:=i + 1;
        Ha (max < Osztály[i].fog) akkor
                max:=Osztály[i].fog;
                ind:=i;
        Elágazás vége
Ciklus vége
Ki(Osztály[ind].név);
```

3. **Előfeltétel:** adottak a brigádok és a teljesítmények.
Utófeltétel: a program adja meg annak a brigádnak a sorszámát, amelynek a teljesítménye a legnagyobb volt.
Deklaráció: i,ind:N
Max,Brigád[1..18,1..6]:N

```
i:=1;
ind:=1;
max:=Brigád[i,1]+ Brigád[i,2]+ Brigád[i,3]+Brigád[i,4]+Brigád[i,5]+ Brigád[i,6];
Ciklus amíg (i < 18)
        i:=i + 1;
        Ha (max < Brigád[i,1]+ Brigád[i,2]+ Brigád[i,3]+Brigád[i,4]+Brigád[i,5]+ Brigád[i,6]) akkor
                max:= Brigád[i,1]+ Brigád[i,2]+ Brigád[i,3]+Brigád[i,4]+Brigád[i,5]+ Brigád[i,6];
                ind:=i;
        Elágazás vége
Ciklus vége
Ki(i);
```

7. fejezet

Lineáris keresés

1. **Előfeltétel:** adott a számokat tartalmazó tömb.
Utófeltétel: a program adja ki az első olyan szám indexét(ha van) amelyik nagyobb mint 100.
Deklaráció: i:Z+
tömb[1..n]:Z
talált:L

```
i:=1;
Ciklus amíg ((i <= n) ÉS (tömb[i] < 100))
        i:=i + 1;
Ciklus vége
Talált:=(i <= n);
Ha (talált) akkor
        Ki(i);
Különben
        Ki("Nincs ilyen szám!");
Elágazás vége
```

2. **Előfeltétel:** adott a neveket tartalmazó tömb.
Utófeltétel: a program adja ki annak a tömbelemnek az indexét(ha van) melynek értéke Szabó István..
Deklaráció: i:Z+
tömb[1..n]:string
talált:L

```
i:=1;
Ciklus amíg ((i <= n) ÉS (tömb[i] <> "Szabó István"))
        i:=i + 1;
Ciklus vége
Talált:=(i <= n);
Ha (talált) akkor
        Ki(i);
Különben
        Ki("Nincs ilyen név!");
Elágazás vége
```

3. ***Előfeltétel:*** ***adott a tanulók matematika osztályzatainak tömbje.***
Utófeltétel: ***a program adja ki hogy volt e bukott tanuló közöttük.***
Deklaráció: ***i:Z+***
tömb[1..n]:Z
talált:L

```
i:=1;
Ciklus amíg ((i <= n) ÉS (tömb[i] <> 1))
        i:=i + 1;
Ciklus vége
Talált:=(i <= n);
Ha (talált) akkor
        Ki("Van");
Különben
        Ki("Nincs");
Elágazás vége
```

Logaritmikus keresés

1. ***Előfeltétel:*** ***adott a tanulók matematika osztályzatainak tömbje.***
Utófeltétel: ***a program adja ki hogy volt e bukott tanuló közöttük.***
Deklaráció: ***i:Z+***
tömb[1..n]:Z
talált:L

```
lépés:=0;
alsó:=1;
felső:=100;
talált:=hamis;
ciklus amíg ((nem talált) ÉS (alsó<=felső))
        közép:=(alsó+felső) div 2;
        lépés:=lépés+1;
        Ha (közép = gondolt) akkor
                talált:=igaz;
        Különben Ha (közép < gondolt) akkor
                        alsó:=közép + 1;
                Különben
                        felső:=közép - 1;
                Elágazás vége
        Elágazás vége
Ciklus vége
Ki(közép);
Ki(lépés);
```

A feladat megoldásából hiányoznak az inputok.

8. fejezet

Rendezések

1. ***Előfeltétel:*** ***adott a számokat tartalmazó tömb.***
Utófeltétel: ***a program rendezze a tömb elemeit csökkenő sorrendbe.***
Deklaráció: ***i:Z+***
segéd;A[1..n]:N

```
Ciklus i :=2 - től n - ig
        Ciklus j:=n - től i - ig
                Ha (A[j-1] < A[j]) akkor
                        segéd:=A[j-1];
                        A[j-1]:=A[j];
                        A[j]:=segéd;
                Elágazás vége
        Ciklus vége
Ciklus vége
```

2. ***Előfeltétel:*** ***adott a hőmérsékleteteket tartalmazó tömb.***
Utófeltétel: ***a program rendezze a tömb elemeit növekvő sorrendbe.***
Deklaráció: ***i:Z+***
segéd;Hőmérséklet[1..n]:N

```
Ciklus i :=2 - től n - ig
        Ciklus j:=n - től i - ig
                Ha (Hőmérséklet[j-1] > Hőmérséklet[j]) akkor
                        segéd:=Hőmérséklet[j-1];
                        Hőmérséklet[j-1]:=Hőmérséklet[j];
                        Hőmérséklet[j]:=segéd;
                Elágazás vége
        Ciklus vége
Ciklus vége
```

3. ***Előfeltétel:*** ***adott a névsor.***
Utófeltétel: ***a program rendezze a névsor elemeit születési idő mező szerint növekvő sorrendbe.***
Deklaráció: ***i:Z+***

```
j,szül:N
Rekord Személy
        Név:string
        Szül:N
Rekord vége
Névsor[1..n],segéd:Személy
```

```
Ciklus i:=2 - től n - ig
        szül:=Névsor[i].szül;
        segéd:=Névsor[i]
        j:=i - 1;
        Ciklus amíg ((j > 0) ÉS (szül < Névsor[j].szül))
                Névsor[j+1]:=Névsor[j];
                j:=j - 1;
        Ciklus vége
        Névsor[j+1]:=segéd;
Ciklus vége
```

9. fejezet

Összefésülés

1. Előfeltétel: adott a két számokat tartalmazó tömb.
Utófeltétel: a program hozza létre a két tömb unióját (a megegyező elemek csak egyszer szerepeljenek az új tömbben).
Deklaráció: j,k:Z+
A[1..n],B[1..m],c[1..m+n]:N

```
i:=0;
j:=1;
k:=1;
Ciklus amíg (i < m + n)
        i:=i + 1;
        Ha ((k > n) VAGY ((j <= m) ÉS (A[j] < B[k]))) akkor
                C[i]:=A[j];
                j:=j + 1;
        Különben Ha ((j > m) VAGY ((k <= n) ÉS (B[k] < A[j]))) akkor
                        C[i]:=B[k];
                        k:=k + 1;
                Különben Ha (A[j] = B[k]) akkor
                                C[i]:=A[j];
                                j:=j + 1;
                                k:=k + 1;
                        Elágazás vége
                Elágazás vége
        Elágazás vége
Ciklus vége
```

2. Előfeltétel: adott a két számokat tartalmazó tömb.
Utófeltétel: a program hozza létre a két tömb metszetét (csak a a megegyező elemek kerüljenek az új tömbbe).
Deklaráció: j,k:Z+
A[1..n],B[1..m],c[1..m+n]:N

```
i:=0;
j:=1;
k:=1;
Ciklus amíg (i < m + n)
        Ha ((k > n) VAGY ((j <= m) ÉS (A[j] < B[k]))) akkor
                j:=j + 1;
        Különben Ha ((j > m) VAGY ((k <= n) ÉS (B[k] < A[j]))) akkor
                        k:=k + 1;
                Különben Ha (A[j] = B[k]) akkor
                                C[i]:=A[j];
                                j:=j + 1;
                                k:=k + 1;
                                i:=i + 1;
                        Elágazás vége
                Elágazás vége
        Elágazás vége
Ciklus vége
```

Vegyes feladatok

```
1.
sE:=0;
sÖ:=0;
Ciklus i:=1-től 21-ig
        sE:=sE+MálnaE[i];
        sÖ:=sÖ+MálnaÖ[i];
Ciklus vége
Ha (sE>sÖ) akkor
        Ki('Emil');
Különben Ha (sÖ>sE) akkor
        Ki('Ödön');
Különben
        Ki('Ugyanannyi');
Elágazás vége
Elágazás vége

2.
s:=0;
Ciklus i:=1-től 21-ig
        s:=s+MálnaE[i];
Ciklus vége
átlag:=s/21;
Ki(átlag);

3.
i:=1;
Ciklus amíg (MálnaE[i]+MálnaÖ[i]>=20)
        i:=i+1;
Ciklus vége
Ki(i);

4.
i:=1;
Ciklus amíg (MálnaE[i+1]>MálnaE[i])
        i:=i+1;
Ciklus vége
Ki(i);

5.
db:=0;
Ciklus i:=1-től 21-ig
        Ha (MálnaÖ[i]>5) akkor
                db:=db+1;
        Elágazás vége
Ciklus vége
Ki(db);
```

6.

```
lusta:=hamis;
i:=13;
Ciklus amíg ((nem lusta) ÉS (i <21))
        i:=i+1;
        Ha (MálnaE[i]=0) akkor
                lusta:=igaz;
        Elágazás vége
Ciklus vége
Ha (lusta) akkor
        Ki('Lustálkodott!');
Különben
        Ki('Nem lustálkodott');
Elágazás vége
```

7.

```
j:=0;
Ciklus i:=1-től 21-ig
        Ha ((MálnaE[i]>10) ÉS (MálnaÖ[i]>10)) akkor
                j:=j+1;
                Jonapok[j]:=i;
        Elágazás vége
Ciklus vége
```

8.

```
i:=1;
ind:=1;
max:=MálnaE[1];
Ciklus i:=2-től 21-ig
        Ha (MálnaE[i]>max) akkor
                max:=MálnaE[i];
                ind:=i;
        Elágazás vége
Ciklus vége
Ki(ind);
Ki(max);
```

9.

```
i:=1;
ind:=1;
min:=MálnaÖ[1];
Ciklus i:=2-től 21-ig
        Ha (MálnaÖ[i]<min) akkor
                min:=MálnaE[i];
                ind:=i;
        Elágazás vége
Ciklus vége
Ki(ind);
Ki(min);
```

```
10.
Ciklus i:=1-től 20-ig
        Ciklus j:=i+1-től 21-ig
                Ha (MálnaE[j]<MálnaE[i]) akkor
                        segéd:=MálnaE[j];
                        MálnaE[j]:=MálnaE[i];
                        MálnaE[i]:=segéd;
                Elágazás vége
        Ciklus vége
Ciklus vége
```

```
11.
Ciklus i:=2-től 21-ig
        Ciklus j:=21-től i-ig
                Ha (MálnaÖ[j-1]>MálnaÖ[j]) akkor
                        segéd:=MálnaÖ[j-1];
                        MálnaÖ[j-1]:=MálnaÖ[j];
                        MálnaÖ[j]:=segéd;
                Elágazás vége
        Ciklus vége
Ciklus vége
```

```
12.
i:=0;
j:=1;
k:=1;
Ciklus amíg (i<42)
        i:=i+1;
        Ha ((k > 21) VAGY ((j <= 21) ÉS (MálnaE[j] < MálnaÖ[k]))) akkor
                Együtt[i]:=MálnaE[j];
                j:=j+1;
        Különben Ha ((j > 21) VAGY ((k <= 21) ÉS (MálnaÖ[k] < MálnaE[j]))) akkor
                Együtt[i]:=MálnaÖ[k];
                k:=k+1;
        Különben
                Együtt[i]:=MálnaE[j];
                j:=j+1;
                i:=i+1;
                Együtt[i]:=MálnaÖ[k];
                k:=k+1;
        Elágazás vége
        Elágazás vége
Ciklus vége
```

```
13.
Ciklus i:=1-től n-1-ig
        index:=i;
        max:=Távolság[i];
        Ciklus j:=i+1-től n-ig
                Ha (max<Távolság[j]) akkor
                        max:=Távolság[j];
                        index:=j;
                Elágazás vége
        Ciklus vége
        Távolság[index]:=Távolság[i];
        Távolság[i]:=max;
Ciklus vége
```

14.
```
i:=1;
ind:=1;
max:=Hossz(Település[1]);
Ciklus amíg (i<n)
	i:=i+1;
	Ha (Hossz(Település[i])>max) akkor
		max:=Település[i];
		ind:=i;
	Elágazás vége
Ciklus vége
Ki(Település[ind]);
```

15.
```
j:=0;
Ciklus i:=1-től n-ig
	Ha (Távolság[i]<5) akkor
		j:=j+1;
		Sűrű[j]:=Település[i];
	Elágazás vége
Ciklus vége
```

16.
```
db:=0;
Ciklus i:=1-től n-ig
	Ha (Település[i]='Balaton*') akkor
		db:=db+1;
	Elágazás vége
Ciklus vége
Ki(db);
```

17.
```
talált:=hamis;
i:=0;
Ciklus amíg (nem talált) és (i<n)
	i:=i+1;
	Ha (Távolság[i]>=10 ÉS Távolság[i]<=12) akkor
		talált:=igaz;
	Elágazás vége
Ciklus vége
Ha (talált)
	Ki('Van);
Különben
	KI('Nincs);
Elágazás vége
```

18.
```
db:=0;
Ciklus i:=1-től n-ig
	Ha (Távolság[i]>20) akkor
		db:=db+1;
	Elágazás vége
Ciklus vége
Ki(db);
```

19.
i:=1;
j:=1;
Ciklus amíg (Település[i]<>'Balatonfüred')
    i:=i+1;
Ciklus vége
Ciklus amíg (Település[j]<>'Balatonszemes')
    j:=j+1;
Ciklus vége
Ki(j-i);

20.
s:=0;
Ciklus i:=1-től n-ig
    s:=s+Távolság[i];
Ciklus vége
átlag:=s/21;
Ki(átlag);

A feladatokból hiányoznak az állapottér leírások.

10. fejezet

Mátrix

***1.** Előfeltétel: adott a mátrix.*
Utófeltétel: a program írja ki a mátrix legnagyobb és legkisebb elemét.
Deklaráció: i,j:Z+
max, min, Mátrix[1..m,1..n]:Z

```
max:=Mátrix[1,1];
min:=Mátrix[1,1];
Ciklus i:=1-től m-ig
        Ciklus j:=1-től n ig
                Ha (max < Mátrix[i,j]) akkor
                        max:=Mátrix[i,j];
                Különben Ha (min > Mátrix[i,j]) akkor
                                min:=Mátrix[i,j];
                        Elágazás vége
                Elágazás vége
        Ciklus vége
Ciklus vége
Ki(max, min);
```

***2.** Előfeltétel: adott a mátrix.*
Utófeltétel: a program írja ki a mátrix legnagyobb összegű sorának indexét.
Deklaráció: i, j, ind, max, sorszum, Mátrix[1..m,1..n]:Z+

```
max:=0;
ind:=1;
Ciklus i:=1-től m-ig
        sorszum:=0;
        Ciklus j:=1-től n ig
                sorszum:=sorszum+Mátrix[i,j];
        Ciklus vége
        Ha (max < sorszum) akkor
                max:=sorszum;
                ind:=i;
Ciklus vége
Ki(ind);
```

***3.** Előfeltétel: adott a mátrix*
Utófeltétel: a program írja ki a mátrix átlóiban lévő elemek összegét
Deklaráció: i,:Z+
j:N
Mátrix[1..n,1..n], összeg:Z

```
j:=n+1;
összeg:=0;
Ciklus i:=1-től n ig
        j:=j-1;
        Ha (i=j) akkor
                összeg:=összeg + Mátrix[i,j];
        Különben
                összeg:=összeg + Mátrix[i,i] + Mátrix[i,j];
        Elágazás vége
Ciklus vége
Ki(összeg);
```

Szöveg

1. Előfeltétel: adott a szöveg.
Utófeltétel: a program írja ki, hogy a szövegben szerepel-e a "kutya" szó.
Deklaráció: i:N
van:L
szöveg:string

```
Be(szöveg)
i:=0;
van:=hamis
Ciklus amíg ((i < Hossz(szöveg)-4) ÉS (van=hamis))
        i:=i + 1;
        Ha ((szöveg[i]="k") ÉS (szöveg[i+1]="u")
        ÉS (szöveg[i+2]="t") ÉS (szöveg[i+3]="y") ÉS (szöveg[i+4]="a")) akkor
                van:=igaz
        Elágazás vége
Ciklus vége
Ha (van) akkor
        Ki("Megtalálható");
Különben
        Ki("Nem található meg");
Elágazás vége
```

2. Előfeltétel: adott a szöveg.
Utófeltétel: B tömb tartalmazza a szövegben előforduló karakterek relatív gyakoriságát.
Deklaráció: i, j, db:N
van:L
szöveg, c:String
B[1..n]:R

```
Be(szöveg);
i:=0;
Ciklus amíg (i< Hossz(szöveg))
        i:=i + 1;
        j:=0;
        c:=szöveg[i];
        db:=0;
        Ciklus amíg (j < Hossz(szöveg))
                j:=j + 1;
                Ha (c = szöveg[j]) akkor
                        db:=db + 1;
                Elágazás vége
        Ciklus vége
        B[i]:=db / Hossz(szöveg)
Ciklus vége
```

3. Előfeltétel: adott a szöveg.
Utófeltétel: a program írja ki a szövegben előforduló szavak számát.
Deklaráció: i, j, db:N

```
Be(szöveg);
db:=0;
Ciklus i:=1-től Hossz(szöveg)-ig
        Ha (szöveg[i] = " ") akkor
                db:=db + 1;
        Elágazás vége
Ciklus vége
db:=db+1;
Ki(db);
```

14. fejezet

Vegyes feladatok

1. ***Előfeltétel:*** ***adott a két szám.***
Utófeltétel: ***a program írja ki a két szám legnagyobb közös osztóját.***
Deklaráció: ***a, b:N***

```
Be(a);
Be(b);
Ciklus amíg (a <> b)
        Ha (a < b) akkor
                b:=b-a;
        Különben
                a:=a-b;
        Elágazás vége
Ciklus vége
Ki(a);
```

2. ***Előfeltétel:*** ***adott a két szám.***
Utófeltétel: ***a program írja ki a két szám legkisebb közös többszörösét.***
Deklaráció: ***a, b, lktöbb, osztó:Z+***

```
Be(a);
Be(b);
lktöbb:=1;
Ciklus amíg ((a <> 1) VAGY (b <> 1))
        Ciklus amíg (a <> 1)
                osztó:=2;
                Ciklus amíg (a mod osztó <> 0)
                        osztó:=osztó+1;
                Ciklus vége
                Ha (b mod osztó = 0) akkor
                        b:=b div osztó;
                Elágazás vége
                lktöbb:=lktöbb*osztó;
                a:=a div osztó;
        Ciklus vége
        Ciklus amíg (b <> 1)
                osztó:=2;
                Ciklus amíg (b mod osztó <> 0)
                        osztó:=osztó+1;
                Ciklus vége;
                b:= b div osztó;
                lktöbb:=lktöbb*osztó;
        Ciklus vége
Ciklus vége
Ki(lktöbb);
```

```
3.      Előfeltétel:    adott a két négyjegyű szám szám.
        Utófeltétel:    a program írja ki a két szám maszkjának értékét.
        Deklaráció:     a, b:Z+
                        db:N
                        i:Q

Be(a);
Be(b);
i:=1000;
db:=0;
Ciklus amíg (i >= 1)
        Ha ((a div i) = (b div i)) akkor
                db:=db + 1;
        Elágzás vége
        a:=a mod i;
        b:=b mod i;
        i:=i/10;
Ciklus vége
Ki(db);
```

```
4.      Előfeltétel:    adott az évek száma, a kamatláb és a betét összege.
        Utófeltétel:    a program írja ki a betét kamatos kamattal növelt összegét évenként.
        Deklaráció:     betét, kamat:R
                        i, év:Z+

Be(betét, kamat, év);
Ciklusi:=1-től év-ig
        betét:=betét*(1+kamat/100);
        Ki(betét);
Ciklus vége
```

```
5.      Előfeltétel:    ismerjük a szorzótáblát :).
        Utófeltétel:    a program töltse fel a szorzótáblát oszlop és sorfejléccel együtt.
        Deklaráció:     i, j:N
                        Tábla[1..11,1..11]:N

Ciklus i:=1-től 11-ig
        Tábla[1,i]:=i-1;
        Tábla[i,1]:=i-1;
Ciklus vége
Ciklus i:=2-től 11-ig
        Ciklus j:=2-től 11-ig
                Tábla[i,j]:=(i-1)*(j-1);
        Ciklus vége
Ciklus vége
```

```
6.      Előfeltétel:    adott az n*n-es mátrix.
        Utófeltétel:    a program döntse el, hogy a mátrix szimmetrikus-e.
        Deklaráció:     i, j:N
                        Tömb[1..n,1..n]:Z
                        szimmetrikus:logikai
szimmetrikus:=igaz;
Ciklus i:=1-től n-ig
        Ciklus j:=1-től n-ig
                Ha (Tömb[i,j] <> Tömb[j,i]) akkor
                        szimmetrikus:=hamis;
                Elágazás vége
Ciklus vége
Ki(szimmetrikus);
```

7. ***Előfeltétel:*** ***adottak az értékek.***
Utófeltétel: ***a program írja ki a permutáció és variáció értékét.***
Deklaráció: ***counter:N***
total:Z+
a, b:R

```
Függvény Factorial(Num:Z+):Z+
total = 1;
Ciklus counter:=2-től Num-ig
        total = total * counter;
Ciklus vége
Factorial = total;
Függvény vége

Függvény P(N:Z+, R:Z+):R
        P = Factorial(N) / Factorial(N - R);
Függvény vége

Függvény C(N:Z+, R:Z+):R
   C = Factorial(N) / (Factorial(N - R) * Factorial(R));
Függvény vége

Eljárás Permutáció(N:Z+, Z:+)
a = P(N, Z);
b = C(N, Z);
Ki(a);
Ki(b);
Eljárás vége
```

8. ***Előfeltétel:*** ***adott az m*n-es mátrix.***
Utófeltétel: ***a program rendezze a mátrixot növekvő sorrendbe minimumkiválasztásos rendezéssel.***
Deklaráció: ***i, j, k, l, indi, indj:Z+***
Mátrix[1..m,1..n], min:N

```
Ciklus i:=1-től m-ig
        Ciklus j:=1-től n-ig
                min:=Mátrix[i,j];
                Ciklus k:=m-től i-ig
                        Ciklus l:=n től 1-ig
                                Ha ((i = k) ÉS (j >= l)) akkor
                                Különben
                                        Ha (min > Mátrix[k,l])
                                                min:=Mátrix[k,l];
                                                indi:=k;
                                                indj:=l;
                                        Elágazás vége
                                Elágazás vége
                        Ciklus vége
                Ciklus vége
                Mátrix[indi,indj]:=Mátrix[i,j];
                Mátrix[i,j]:=min;
        Ciklus vége
Ciklus vége
```

9. ***Előfeltétel:*** ***adott a két tömb***
Utófeltétel: ***a program hozza létre a két tömb Descartes-féle szorzatát.***
Deklaráció: ***i, j, k:Z+***
A[1..m],B[1..n]:N
Mátrix[1..m*n,1..2]:N

```
k:=1;
Ciklus i:=1-től m-ig
        Ciklus j:=1-től n-ig
                Mátrix[k,1]:=A[i];
                Mátrix[k,2]:=B[j];
                k:=k + 1;
        Ciklus vége
Ciklus vége
```

10. ***Előfeltétel:*** ***adott a szöveg***
Utófeltétel: ***a program adja ki azokat a szavakat, melyek egynél többször fordulnak elő.***
Deklaráció: ***i, j, k, l , db:N***
szöveg, Tömb[1..n]:string

```
Be(szöveg)
i:=0;
j:=0;
Ciklus amíg (i < Hossz(szöveg));
        i:=i + 1;
        j:=j + 1;
        Ciklus amíg ((szöveg[i] <> " ") VAGY (i < Hossz(szöveg)))
                Tömb[j]:=Tömb[j] & szöveg[i];
                i:=i+1;
        Ciklus vége
Ciklus vége
Ciklus k:=1-től j-ig
        db:=0;
        Ciklus l:=1-től j-ig
                Ha (Tömb[k]=Tömb[l]) akkor
                        db:=db+1;
                Elágazás vége
        Ciklus vége
        Ha (db > 1) akkor
                Ki(Tömb[k]);
                Ki(db);
        Elágazás vége
Ciklus vége
```

Tartalomjegyzék

TÖRTÉNETI ÁTTEKINTÉS 2
ELŐZMÉNYEK 2
PROGRAMOZÁSI NYELVEK FEJLŐDÉSE 2
PROGRAMOZÁSI NYELVEK OSZTÁLYOZÁSI SZEMPONTJAI 4
GENERÁCIÓK SZERINT 4
MŰKÖDÉS SZERINT 4
SZERKEZET SZERINT 4
ALAPDEFINÍCIÓK 5
AZ ALGORITMUS 7
AZ ALGORITMUSOKKAL SZEMBEN TÁMASZTOTT KÖVETELMÉNYEK 7
AZ ALGORITMUS ÁLLAPOTTERE 7
A PROGRAM I. 8
A PROGRAMFEJLESZTÉS LÉPÉSEI 8
Compileres technika 8
Interpreteres technika 8
PROGRAMMAL SZEMBENI ELVÁRÁSOK 9
ELLENŐRZŐ KÉRDÉSEK 10
I. 10
II. 11
III. 11
A PROGRAM II. 12
A PROGRAM ÉLETÚTJA 12
Feladatmegfogalmazás 12
Specifikáció, algoritmustervezés 12
Kódolás 13
Tesztelés 13
Hibakeresés, javítás 14
Hatékonyságvizsgálat 15
Dokumentálás 15
Üzembehelyezés, karbantartás 15
A DOKUMENTÁCIÓ 16
Felhasználói kézikönyv (User Guide) 16
Fejlesztői kézikönyv (Programming Guide) tartalma 16
Operátori kézikönyv (Installing Guide) 16

PROGRAMTERVEZÉSI MÓDSZEREK **17**
Frontális feladatmegoldás *17*
Felülről lefelé (top-down) programozás *17*
Alulról felfelé (bottom-up) programozás *17*
Párhuzamos finomítás elve *17*
Döntések elhalasztásának elve *17*
Vissza az ősökhöz elv *17*
Nyílt rendszerű felépítés *18*
Döntések kimondásának elve *18*
Adatok elszigetelésének elve *18*
Moduláris programozás *18*
Strukturált programozás *19*
Objektumorientált programozás *19*
ADATSZERKEZETEK **21**
MATEMATIKAI 21
PROGRAMOZÁSI 22
Egyszerű adattípusok *22*
Összetett adattípusok *23*
ELLENŐRZŐ KÉRDÉSEK **26**
I. 26
II. 26
III. 27
AZ ALGORITMUSOK ALAPELEMEI **28**
VÁLTOZÓK 28
I/O MŰVELETEK 28
UTASÍTÁSOK 28
Deklarációs utasítások *28*
Végrehajtható utasítások *29*
KIFEJEZÉSEK 29
VEZÉRLÉSI SZERKEZETEK 30
Utasítás-végrehajtási sorozat (szekvencia) *30*
Elágazás (szelekció) *30*
Ciklusszervezés (iteráció) *30*
Ugró utasítások *31*
AZ ALGORITMUS ALAPJELEI, ÉPÍTŐELEMEI 31
ALGORITMUS LEÍRÓ ESZKÖZÖK **32**
FOLYAMATÁBRA 32
STRUKTOGRAM 32
PSZEUDOKÓD 32
FUNKCIONÁLIS LEÍRÁS 32
JACKSON ÁBRA 32
MONDATSZERŰ LEÍRÁS 32
PÉLDÁK 33
Algoritmusleírás *33*
Folyamatábra *33*
Struktogram *34*
Pszeudokód (Pascal) *34*
Jackson jelölés *35*
ELLENŐRZŐ KÉRDÉSEK **36**

ALPROGRAMOK ... **37**

ELJÁRÁS (PROCEDURE) ... 37
FÜGGVÉNY (FUNCTION) ... 38
PARAMÉTEREK, PARAMÉTERÁTADÁSI MÓDOK ... 39
Változó vagy cím szerinti paraméterátadás ... 39
Érték szerinti paraméterátadás ... 39
REKURZIÓ ... 40
PROGRAMOZÁSI TECHNIKÁK ... 41
Strukturált programozástechnika ... 41
Objektumorientált programozástechnika ... 42

ELLENŐRZŐ KÉRDÉSEK ... **43**

ELEMI ALGORITMUSOK I. ... **44**

ÖSSZEGZÉS TÉTELE ... 44
Példa ... 44
Feladatok ... 44
KIVÁLASZTÁS TÉTELE ... 45
Példa ... 45
Feladatok ... 45
MEGSZÁMLÁLÁS TÉTELE ... 46
Példa ... 46
Feladatok ... 46

ELEMI ALGORITMUSOK II. ... **47**

ELDÖNTÉS TÉTELE ... 47
Példa ... 47
Feladatok ... 47
KIVÁLOGATÁS TÉTELE ... 48
Példa ... 48
Feladatok ... 48
MAXIMUM-MINIMUM KIVÁLASZTÁS TÉTELE ... 49
Példa ... 49
Feladatok ... 49

KERESÉSI TÉTELEK ... **50**

Lineáris keresés tétele ... 50
Logaritmikus keresés tétele ... 51
Visszalépéses keresés tétele ... 52

RENDEZÉSEK I. ... **54**

RENDEZÉS KÖZVETLEN KIVÁLASZTÁSSAL ... 54
RENDEZÉS MINIMUMKIVÁLASZTÁSSAL ... 55
BUBORÉKOS RENDEZÉS ... 55
BESZÚRÁSOS RENDEZÉS ... 56
Feladatok ... 56

RENDEZÉSEK II. ... **57**

GYORSRENDEZÉS ... 57

ÖSSZEFÉSÜLÉS ... **58**

Feladatok ... 58

VEGYES FELADATOK ... **59**

ADATTÁROLÁSI MÓDSZEREK I. **60**
MÁTRIX 60
Példa *60*
Feladatok *60*
SZÖVEG (STRING) 61
Példa *61*
Feladatok *61*
FILE 62
A file-ok alapműveletei *62*
Példa *63*
ADATTÁROLÁSI MÓDSZEREK II. **64**
TÁBLA 64
Hash algoritmus *65*
VEREM ADATSZERKEZET 66
SOR ADATSZERKEZET 67
ADATTÁROLÁSI MÓDSZEREK III. **68**
LISTA 68
Listatípusok *68*
ADATTÁROLÁSI MÓDSZEREK IV. **73**
BINÁRIS FA 73
Definíciók *73*
Rendezetlen bináris fa *74*
Rendezett bináris fa (keresőfa) *74*
Kiegyensúlyozott bináris fák (AVL-fák, vörös-fekete fák) *75*
ALGORITMUSTERVEZÉSI ELVEK **80**
ELEMENKÉNTI FELDOLGOZÁS 80
Egy halmazból egy halmaz előállítása *80*
Egy halmazból több halmaz előállítása *80*
Két vagy több halmazból egy halmaz előállítása *80*
Két vagy több halmazból két vagy több halmaz előállítása *80*
MOHÓ ALGORITMUSOK 81
Esemény-kiválasztás *81*
Pénzváltás *81*
Elágazás és korlátozás *82*
DINAMIKUS PROGRAMOZÁS 82
A dinamikus programozás lépései *82*
KÖZELÍTŐ ALGORITMUSOK 83
Ládapakolás *83*
VÉLETLENT HASZNÁLÓ ALGORITMUSOK 83
VEGYES FELADATOK **84**

MEGOLDÁSOK **85**
1. FEJEZET 85
I. *85*
II. *85*
III. *85*
2. FEJEZET 87
I. *87*
II. *87*
III. *87*
3. FEJEZET 89
4. FEJEZET 91
5. FEJEZET 92
Összegzés *92*
Kiválasztás *93*
Megszámlálás *93*
6. FEJEZET 94
Eldöntés *94*
Kiválogatás *95*
Maximum-minimum kiválasztás *96*
7. FEJEZET 97
Lineáris keresés *97*
Logaritmikus keresés *98*
8. FEJEZET 99
Rendezések *99*
9. FEJEZET 100
Összefésülés *100*
Vegyes feladatok *101*
10. FEJEZET 106
Mátrix *106*
Szöveg *107*
14. FEJEZET 108
Vegyes feladatok *108*

TARTALOMJEGYZÉK **112**