Programozási alapfogalmak

Alapfogalmak

Algoritmus: lépések sorozata, amellyel egy feladat kiindulásától a megoldásáig jutunk.
Program: Az algoritmus megfogalmazása a számítógépek, vagy a fordító programok számára érthető nyelven.

A programozás szintjei:
- gépi kódú: a számítógép számára közvetlenül érthető nyelv a gépi kód. A nyelv elemei közvetlenül a processzor számára ismert gépi kódú utasítások. Mind az utasítások, mind az adatok bináris kódúak. A memóriában tárolt adatok elhelyezkedésének és helyes felhasználásának összes gondja a programozót terheli.
- assembly nyelv: A nyelv utasításkészlete szinte teljesen egyezik a gépi kódú programozásával, de az utasításokat könnyebben megjegyezhető angol nevekkel látták el, valamint az adatokat és memóriacímeket sem kell binárisan megadni, hanem használható a sokkal kényelmesebb hexadecimális számrendszer is. Természetesen az assembly nyelvű programok közvetlenül nem értelmezhetőek a processzor számára, hanem egy fordító program segítségével gépi kódúvá kell lefordítani ahhoz, hogy futtatható legyen.
- magas szintű programozási nyelvek: Magasabb szintű utasításkészlet, amely már nem a processzor utasításkészletéhez áll közel, hanem inkább az emberi gondolkodáshoz. Az adatok kezelésével kapcsolatos gondok nagy részét átveszi a fordítóprogram. Ezekkel a nyelvekkel fogunk részletesebben foglalkozni.
- újabb generációs nyelvek: Az emberi gondolkodáshoz még közelebb álló programozási nyelvek.

A továbbiak a magas szintű programozási nyelvekre vonatkoznak:

Két később használatos fogalom:
Szintaktika: a nyelvtan, szintaktikailag helyes valami, ha helyesen van leírva.
Szemantika: a jelentés, szemantikailag helyes valami, ha helyesen működik.
A programokat szintaktikailag tudja ellenőrizni a fordító, szemantikailag nem.

Compiler-es és Interpreter-es nyelvek
A programozási nyelvek lehetnek:
- compiler-es
- interpreter-es

Fordító program (compiler): Egy program, amely valamilyen programozási nyelven megírt programot gépi kódúvá fordít. Minél magasabb szintű egy nyelv annál bonyolultabb a fordítás.

Értelmező (interpreter): Egy program, amely valamilyen programozási nyelven megírt programot utasításonként értelmez és végre is hajt.

A programkészítés és futtatás menete compileres nyelvek (pl C, C++, Pascal, Delphi, stb) esetén:
1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program lefordítása a fordító program segítségével gépi kódú programmá
3. a gépi kódú program futtatása

A programkészítés és futtatás menete interpreteres nyelvek esetén (Basic, script nyelvek:
1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program futtatása egy futtató program (interpreter) segítségével

vagy (pl. JAVA):
1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program lefordítása egy köztes kódra (ezt a fordítót is szokás compiler-nek nevezni)
3. köztes kód futtatása egy futtató program segítségével

A compiler-es nyelvek előnye: gyorsabb, hátránya: a lefordított kód csak ugyanolyan architectúrájú gépen futtatható, mint amelyiken fordították.

Az interpreter-es nyelvek előnye: a program hordozható minden olyan helyre, ahol van interpreter, hátránya: lassabb.

Case-sensitive, case-insensitive nyelvek
A case-sensitive nyelvek megkülönböztetik a kisbetűket és a nagybetűket, a case-insensitive nyelvek nem, azaz mindegy, hogy valamit kisbetűvel vagy nagybetűvel írunk.

Ellenőrző kérdések

1. Algoritmus, program fogalma?
2. Mi a szintaktika és szemantika?
3. Melyek a programozás szintjei?
4. Milyen könnyebbséget jelent az assembly programozás a gépi kódú programozáshoz képest?
5. Milyen könnyebbséget jelent a magas szintű programozási nyelvek használata a gépi kódú programozáshoz képest?
6. Mi a programkészítés és futtatás menete compileres nyelvek esetén? Soroljon fel néhány ilyen nyelvet!
7. Mi a programkészítés és futtatás menete interpreteres nyelvek esetén? Soroljon fel néhány ilyen nyelvet!
8. Melyek a compileres és interpreteres nyelvek előnyei, hátrányai?
9. Mit jelentenek a case-sensitive, case-insensitive jelzők?

Adatok kezelése

Az utasításokban használhatunk
- **konstansok**at, azaz az utasításba közvetlenül beírhatjuk az adatot.
- **elnevezett konstansok**at, azaz lehetőségünk van arra, hogy névvel lássunk el konstans adatokat és ahelyett, hogy magát az adatot írnánk az utasításba, az adat nevével hivatkozunk rá. Ennek jelentősége a program megértésében, valamint módosíthatóságában van. Pl. mennyivel könnyebb megjegyezni és megérteni egy képletet, amelyben a PI szerepel, mintha 3.14….. szerepelne.
- **változók**at: Az utasításokban leggyakrabban a memóriában már korábban eltárolt adatokat használunk. Ezekre a gépi kódú programokban, illetve az assembly nyelvben az eltárolt adat memóriacímével kellett hivatkozni. A magas szintű programozási nyelvekben ezek helyett a változókat használjuk.

A változó

A változó egy elnevezett memória terület. Jellemzői: a neve, memóriacíme, típusa, pillanatnyi értéke. A változókat első használat előtt deklarálni kell, azaz meg kell adni a változó nevét és típusát. Ennek hatására a fordító automatikusan kijelöl egy (virtuális)memóriacímet ennek a változónak. Arról. hogy melyik cím lett lefoglalva a változó számára a programozónak nem kell tudnia, hiszen a programban mindenütt a változó nevével hivatkozunk a területre. Erre a területre eltárolhatunk adatot (ha már volt ott adat, akkor felülírva azt), azaz értéket adunk a változónak, illetve hivatkozhatunk a pillanatnyilag éppen ott található értékre.
- **A változó neve**: nevek képzésének vannak szabályai, ezek természetesen programozási nyelvenként eltérőek lehetnek. Legtöbb nyelvben a szabályok a következők: a nevek betűkből és számokból állhatnak, betűvel kell kezdődjenek (semmiképpen sem lehetnek bennük írásjelek vagy szünet!!).
- **A változó memóriacíme**: nem kell ismernünk az értékét, de bizonyos nyelvekben felhasználjuk, lásd később.
- **A változó pillanatnyi értéke**: egy változónak a definiálás pillanatától kezdve van pillanatnyi értéke. Még akkor is ha még nem adtunk neki értéket (persze ez az érték nem valódi eltárolt érték), de ilyenkor nem célszerű felhasználni a pillanatnyi értéket. A definiált, de még értéket nem kapott változókat inicializálatlan változóknak szokás nevezni. Egyes nyelvek fordító programjai nem engedik felhasználni az inicializálatlan változók értékét, míg más nyelveknél a programozó felelőssége, hogy ne használja.
- **A változó típusa**: a típus
  egyrészt meghatározza a lefoglalt memóriaterület nagyságát, van 1 byte-os, 2 byte-os, 4 byte-os, stb.
  másrészt meghatározza, hogy az adatot hogyan lehet kezelni, egész számként, valós számként, karakterkódként, stb.
  harmadrészt meghatározza, hogy milyen műveletek végezhetők az adattal.

Definíció, deklaráció

A definíció és deklaráció szinte teljesen azonos fogalmak. Ha változó deklarálásáról beszélünk, akkor azt akarjuk kifejezni, hogy megadjuk a fordító számára egy használni kívánt változó nevét és típusát. Ha változó definiálását emlegetünk változó deklarálása helyett, akkor azzal azt hangsúlyozzuk ki, hogy lefoglalódik egy hely a változónak a memóriában. Egy változót egyszer lehet definiálni, de bizonyos körülmények között előfordul, hogy többször deklaráljuk. Vagyis egyszer történhet egy változó számára helyfoglalás, de lehetséges, hogy többször is tudatnunk kell a fordítóval a változó nevét és típusát. (Például többmodulos programok esetén.)

Inicializálás

Inicializálásnak nevezzük, amikor egy változó definiálásával egyidejűleg értéket is adunk a változónak. (Nem összetévesztendő az inicializált változó fogalmával, lásd változó pillanatnyi értéke.)

Gyengén és erősen típusos nyelvek

Azokat a nyelveket, amelyekben kötelező minden használni kívánt változót deklarálni és minden adattípussal szigorúan csak olyan műveletet enged végezni, amely a típusra lehetséges, erősen típusos nyelvnek nevezzük. Ezzel szemben a gyengén típusos nyelvekben nem kötelező a változók deklarálása. A fordító az első értékadáskor állapítja meg , hogy használni kívánjuk a változót, a kapott értékből állapítja meg a típust és a definiálást automatikusan elvégzi. A műveletek ellenőrzésekor is kevésbé szigorú, ugyanis, ha találkozik egy olyan művelettel, amely az adott típussal nem elvégezhető (pl. karakterek szorzása), akkor megpróbálja az adatokat olyan típusúvá átalakítani, amelyre értelmezve van a művelet.
A gyengén típusos nyelvek nagy hátránya, hogy a hibák felderítése hihetetlenül megnehezedik. Például, ha elírjuk egy változó nevét, akkor azt hiszi a fordító, hogy egy új változót szeretnék használni. Az erősen típusos nyelvek esetén, ha elírjuk a változó nevét egy helyen, akkor szól a fordító, hogy ilyen nevű változót nem deklaráltunk. Előnyük viszont, hogy nem kell "vacakolni" a deklarációval. Hosszabb programok esetén megéri az erősen típusos nyelvek használata.
Gyengén típusos nyelv pl. a Visual Basic, JavaScript.
Erősen típusos nyelv pl. a C++, JAVA, Delphi(Pascal)

Adattípusok

Csoportosítás: egyszerű, összetett
előre definiált, felhasználó által definiált

Egyszerű adattípusok:

- logikai: két értéke lehetséges az igaz vagy a hamis
- karakter: egyes nyelvekben 1 byte-os kódolású, míg az újabb nyelvekben már az unicode karakterkódolást használnak
- egészek: lehet 1 byte-os, 2 byte-os, 4 byte-os, 8 byte-os, lehet előjeles és előjeltelen. Pontosan kiszámítható tehát a tároló kapacitásuk. Pl.
  1byte-os előjeles: -128:127 előjeltelen: 0:255
  2 byte-os előjeles: -32268:32267 előjeltelen: 0:65535
- valósak: van egyszeres pontosságú, dupla pontosságú, esetleg van még nagyobb pontosságú.
- mutató típus: egy mutató típusú változó értékül egy memória címet képes felvenni (azaz mutat valahova). A mutatókat ismerő nyelveknek van olyan operátora (művelete), amellyel lehet hivatkozni a mutató által mutatott területre.
- referencia típus: egy referencia változó képes felvenni értékül egy másik változó referenciáit, azaz olyanná válik, mintha a másik változó egy alias neve lenne (egy újabb elnevezés ugyanarra a memóriaterülethez).
- felsorolás típus (felhasználó által definiálható): Létrehozhatunk egy új típust úgy, hogy felsoroljuk az összes lehetséges értékét (pl. redőny, reluxa, függöny). Az ilyen típusú változó csak ezen értékek valamelyikét veheti fel. Tárolás szempontjából a felsorolás típusú adatok egészeknek számítanak. A fordító egész kódokat rendel az értékekhez és ezek tárolódnak, viszont a programozó nem ezeket a kódokat használja, hanem a felsorolt értékeket, ami sokkal kényelmesebb és a hibázás lehetőségét is csökkenti.
- résztartomány típus (felhasználó által definiálható): Létrehozhatunk új típust úgy is, hogy egy létező sorszámozható típus egy résztartományát adjuk meg. Sorszámozható típusok: egész, karakter, felsorolás. Pl. a jegyek tárolásához létre lehetne hozni egy új típust, amely az egészek 1..5 résztartománya lenne. Az ilyen típusú adatok ugyanúgy egészként tárolódnának, viszont a fordító letudná ellenőrizni, hogy valóban csak egy és öt közötti értéket vesz-e fel.

Összetett adattípusok:

- tömb: több azonos típusú elemből áll, az elemekre egy sorszám (index) segítségével hivatkozhatunk. Léteznek statikus és dinamikus tömbök. A statikus tömb deklarációjakor konstansként meg kell adni a maximális elemszámot és ennek megfelelő méretű helyet foglal neki, függetlenül attól, hogy esetleg nem is használunk, ennyi elemet. Statikus tömb esetén a tömbben nem lehet a maximális elemszámtól több elemet tárolni. Dinamikus tömbből létezik változó méretű tömb, amelynél nem csak konstanssal adhatjuk meg a tömb elemszámát, hanem egy változó segítségével is, valamint létezik dinamikus tömb, amely képes szükség szerint növelni a helyfoglalását.
  Létezhetnek többdimenziós tömbök is, ahol nem egy, hanem több index azonosít egy elemet.
- karakterlánc: szöveg tárolására. Megvalósítása karakter típusú tömbökkel történik, csak kiegészítve néhány plussz funkcióval, pl. amellyel eltárolódik az is, hogy hány karakterből áll a karakterlánc, valamint pl. megjelenítési funkcióval.
- rekord vagy struktúra (felhasználó által definiálható): több, különböző típusú elemből áll, az elemekre névvel hivatkozunk. Minden elemének (mezőknek) saját neve van. Használatának előnye, hogy egy egységbe foglalja össze az összetartozó adatokat. Pl. egy könyv címe, ára. Egy rekord(struktúra) használatához előbb egy új típus létrehozását igényli, hiszen ahhoz, hogy a fordító típusként tudja használni (azaz tudjon helyet foglalni neki, valamint le tudja ellenőrizni az elemeivel végzett műveleteket) le kell írni a típust pontosabban. A típus definícióban minden egyes mezőnek meg kell adni a nevét és a típusát.

A tömbökre illetve a rekordokra (struktúrákra) legtöbb nyelvben nincs definiálva művelet, azaz semmilyen műveletet nem végezhetünk velük (nem lehet pl. egy tömböt megszorozni kettővel). Természetesen az elemeikkel végezhetünk a típusuknak megfelelő műveletet. A karakterlánc típusú adattal viszont lehet néhány műveletet végezni, pl kiíratni, értéket adni neki.
A tömbök illetve rekordok (struktúrák) elemei szintén lehetnek összetett típusúak. Lehet pl. egy tanuló típus, amelynek elemei a tanuló neve (karakterlánc), a tanuló jegyei (egész elemű tömb), iskolai végzettség (iskola rekord, amelynek elemei: az iskola megnevezése (karakterlánc), végzettség megnevezése (karakterlánc)).

Ellenőrző kérdések, feladatok

1. Mi az elnevezett konstans? Mi használatuk előnye?
2. Ismertesse a változó fogalmát, jellemzőit?
3. Melyek az általános azonosító képzési szabályok?
4. Mit jelent az, hogy a változó inicializálatlan?
5. Mit értünk inicializálás alatt?
6. Mit határoz meg a változó típusa?
7. Mit jelent a definíció, és a deklaráció?
8. Mit jelent, hogy egy nyelv erősen típusos? Miben különböznek ettől a gyengén típusos nyelvek?
9. Ismertesse az erősen típusos és gyengén típusos nyelvek előnyeit, hátrányait!
10. Ismertesse az előre definiált egyszerű típusokat!
11. Mit nevezünk tömbnek? Mi a rekord (struktúra)?
12. Milyen felhasználó által definiálható típusok vannak?

Kifejezés, kifejezés kiértékelés

Kifejezés: kifejezés elemek összekapcsolva operátorokkal, ahol a kifejezés elem lehet egy konstans, egy változó, függvényhívás (lásd később), vagy egy zárójelezett kifejezés. Pl

```
5*(alma+2)-korte
```

kifejezés tartalmaz egy konstanst (5), egy zárójelezett kifejezést (alma+2), egy változót (korte), valamint két operátort a szorzást (*), és a kivonást (-). A zárójelezett kifejezés tartalmaz egy változót (alma), egy konstanst (2) és egy operátort az összeadást (+).

Kifejezés kiértékelés: Ha egy kifejezés több operátort is tartalmaz, akkor a kiértékelésük sorrendjét a precedencia szabályok határozzák meg. A zárójelezett kifejezésben szereplő operátorok a teljes kifejezés kiértékelése szempontjából nem számítanak, hiszen a zárójelezett kifejezés külön értékelődik ki.

Precedencia szintek: Minden operátor be van sorolva valamilyen precedencia szintre. Egy precedencia szinten lehet több operátor is.

Precedencia szabályok:

(a) A különböző precedencia szintű operátorok esetén mindig a magasabb precedenciájú hajtódik végre először (tehát nem leírás sorrendjében!!).

(b) Az azonos precedencia szintű operátorok közül a leírás sorrendisége dönt. A legtöbb precedencia szint esetén a leírás sorrendjében, azaz balról-jobbra hajtódnak végre. Egyes precedencia szinteknél előfordul, hogy jobbról-balra.

Operátorok

Léteznek:

- egy operandusú operátorok: pl. logikai tagadás operátor, előjel operátor, növelő operátor, stb.
- két operandusú operátorok: pl. összeadás, szorzás, stb.
- három operandusú operátor: a C szerű nyelvekben létezik egy feltételes operátor, amely három operandusú

A leggyakrabban használt operátorok (az első szint a legmagasabb):

Szint 1: előjel operátor:-, logikai tagadás operátor: ! (C szerű nyelvekben)
Szint 2: szorzás: *, osztás: /
Szint 3: összeadás: +, kivonás: -
Szint 4: relációk: <, >, <=, >=
Szint 5: értékadó operátorok (C szerű nyelvek esetén): =, +=, -=, *=, /=, stb. (jobbról-balra!!!)

Természetesen az egyes nyelvek ettől sokkal több operátort ismernek, és több precedencia szint is létezik. Általánosságban elmondható, hogy az egyoperandusú operátorok magasabb precedenciájúak, mint a kétoperandusúak. Valamint a legtöbb szint balról-jobbra értékel ki, kivételek az értékadó operátorok, valamint legtöbb egyoperandusú operátor (hiszen ezeket elé szokás írni és az értékelődik ki hamarabb, amely közelebb áll az operandushoz.

1. Példa:

`3+2*4-4/2*3`

lépés 1.: legmagasabb precedenciájú a *, /, , balról-jobbra értékelődik tehát első lépésben a 2*4 értékelődik ki. Ezután marad a:
3+8-4/2*3
lépés 2.: 4/2. Marad a
3+8-2*3
lépés 3.: 2*3. Marad a
3+8-6
lépés 4: 3+8. Marad a
11-6
lépés 5: 11-6
Eredmény: 5

2. Példa:

`2+3-((5+3)/(2*4)+2)-1  kiértékelése:`

lépés 1: A kifejezésben levő operátorok +, -, -, azaz mindegyik azonos precedenciájú, kiértékelésük balról-jobbra történik: 2+3, marad a
5-((5+3)/(2*4)+2)-1
lépés 2: A kiértékelésben következő operátor jobb oldalán egy zárójelezett kifejezés áll, így a kivonás kiértékelése előtt ki kell értékelni a zárójelezett kifejezést.

`(5+3)/(2*4)+2 kiértékelése:`

lépés 1: két operátor van: /, +. Magasabb precedenciájú a / ennek baloldalán egy zárójelezett kifejezés áll, tehát ki kell értékelni:

`5+3 kiértékelése:`

lépés 1: csak 1 operátort tartalmaz: 5+3

a / operátor jobboldali operandusa is zárójelezett kifejezés:

`2*4 kiértékelése:`

lépés 2: csak 1 operátort tartalmaza: 2*4

tehát az osztás: 8/8 és maradt a kifejezésből az

`1+2`

lépés 2: 1+2, eredménye tehát: 3

ezek után a kifejezés a következőképpen néz ki: 5-3-1 és először az 5-3 hajtódik végre vagyis az eredmény: 2
lépés 3: 2-1, az eredmény 1

Típuskonverziók

Létezik:
- automatikus típuskonverzió
- kikényszerített típuskonverzió

Automatikus típuskonverzió

Ha egy műveletben két különböző típusú adat vesz részt, akkor a művelet elvégzése előtt megpróbálja azonos típusúvá alakítani őket. A két operandus közül a kisebb kapacitásút fogja átalakítani a másiknak a típusára. Például egy 1 byte-os és egy 2 byte-os egész szám összeadásakor az 1 byte-ost alakítja 2 byte-ossá. (Természetesen nem az eredeti adatot fogja átalakítani, hanem létrehoz egy 2 byte-os területet, ahova beíródik az 1 byte-on tárolt szám.)
Szabályok:
- az egészek képesek nagyobb kapacitásúvá konvertálódni
- az egészek képesek valóssá konvertálódni
- a valósak nagyobb pontosságú valóssá
- értékadás operátor esetén mindig a jobb oldali operandus konvertálódik a baloldali típusára

Egyes nyelvekben, illetve a gyengén típusos nyelvekben egy adat képes kisebb kapacitásúvá konvertálódni, ha az értéke belefér.
Példa:
`valt1+valt2*valt3+valt2 kiértékelése: (feltéve, hogy valt1 valós, valt2 1 byte-os egész, valt3 2 byte-os egész)`

lépés 1: valt2*valt3, ennek végrehajtása során a valt2 értéke 2 byte-ossá alakul. Nevezzük az eredményt ered1-nek (hogy tudjak rá hivatkozni valahogy a következő mondatomban). Az ered1 típusa 2 byte-os egész.
lépés 2: valt1+ered1, azaz egy valós és egy 2 byte-os egész. Végrehajtása során az ered1 értéke valóssá alakul, mivel valt1 valós. Az eredmény szintén valós lesz, nevezzük ered2-nek.
lépés 3: ered2+valt2, azaz egy valós és egy 1byte-os egész. Végrehatás során a valt2 értéke valóssá alakul, mivel ered2 valós. Az eredmény szintén valós lesz.

Kényszerített típuskonverzió

A típuskényszerítési operátorral rákényszeríthetjük az adatot olyan típusátalakulásra is, amelyre automatikusan nem lenne képes. Pl. egy valósból egészet csinálhatunk.

Ellenőrző kérdések, feladatok

1. Kifejezés fogalma?
2. Ismertesse a kifejezés kiértékelés szabályait!
3. Mikor történik automatikus típuskonverzió?
4. Ismertesse az automatikus típuskonverzió szabályait!
5. Mit jelent és hogyan érhető el a kényszerített típuskonverzió?

Feladatok:
1. Írja le a következő képletet, az ismertetett operátorok segítségével:
   $$\frac{a+b}{a*b} - \frac{c}{a+2}$$
2. Írja le a másodfokú egyenlet megoldó képletét, ha a gyökvonás művelete `sqrt(kifejezés)`, ahol `kifejezés` az amiből a gyököt vonjuk!
3. Írja le a végrehajtás lépéseit a következő képlet kiértékelése során: `5+6*2-3*(7-2)*2`
4. Írja le a végrehajtás lépéseit és a lezajló típuskonverziókat az 1. feladatban szereplő kifejezés kiértékelése során, ha A egy byte-os egész, B valós, C két byte-os egész.

Algoritmusok, algoritmus leírások

A jegyzet elején már tisztáztuk az algoritmus fogalmát. Elemi lépések sorozata, amellyel eljuthatunk a megoldásig. Ha megvizsgáljuk az elemi lépések végrehajtását, akkor láthatjuk, hogy létezhetnek csak feltételesen végrehajtandó lépések, illetve ismétlődően végrehajtandó lépések is.
Feltételesen végrehajtandó lépésekre lehet példa egy másodfokú egyenlet valós számok halmazán történő megoldása: ha ugyanis a gyökjel alatt (b*b-4*a*c) negatív van akkor nincs megoldás, ha viszont nem negatív van, akkor számolhatjuk a gyököket. Vagyis a gyökök számítása csak abban az esetben hajtandó végre, ha teljesül egy feltétel.
Ismétlődően (ciklikusan) végrehajtandó lépésekre lehet példa egy közelítési feladat: kiszámolva egy közelítő eredményt, ha még nem elég pontos, akkor ebből kiindulva újra kezdjük a közelítés számítást, ezt addig folytatjuk, amíg eléggé pontos eredményt nem kapunk.

Algoritmus leírások
Ha egy feladat algoritmusát másokkal is szeretnénk tudatni. Az algoritmust le kell írnunk valahogyan. Többféle algoritmus leíró módszer létezik:

- szöveges: a legkevésbé pontos, és a legterjengősebb, de mindenki számára érthető
- metanyelven: az algoritmusok leírására kifejlesztett nyelven, ez pontos, és rövid, de ismerni kell a nyelvet
- grafikus metanyelven: pontos, rövid, könnyebben megtanulható nyelv
- programnyelven: a programok is tekinthetők egy algoritmus leírásának

Folyamatábra
Az egyik leggyakrabban használt grafikus leírónyelv a folyamatábra.
Jelölésrendszere:

nyíl: jelzi a lépések végrehajtási sorrendjét
elemi lépés: egy téglalapba beleírt lépés
összetett lépés: egy téglalapba beleírt lépés
elágazás: egy rombusz, amelyből több irányba indul ki nyíl, a rombuszba kell írni az elágazás feltételét, a nyilakra az egyes ágak megnevezését
adat beolvasás, kiírás: egy paralerogrammába írva a kiírandó vagy beolvasandó
folyamat ábra megszakítása, folytatása: karikába írt szám

Ellenőrző kérdések, feladatok

1. Milyen algoritmus leíró módszerek léteznek?
2. Ismertesse a folyamatábra jelölésrendszerét?

Utasítások

A program utasítások sorozata. Az utasítások szekvenciálisan, azaz sorrendben hajtódnak végre. Az utasítások között lehetnek vezérlő utasítások, amelyekkel az utasítások végrehajtási sorrendjét tudjuk befolyásolni.

Utasítások általános szintaktikája:
Az utasítást egy utasításvég jel zárja le, vagy utasítás elválasztójel választja el őket. Emiatt több utasítást is írhatunk egy sorba, vagy akár egy utasítást több sorba is írhatunk. A szokás az, hogy egy sorba nem írunk több utasítást, és ha lehet, akkor nem is írjuk több sorba. A hosszú utasításokat viszont, főleg a vezérlési szerkezeteket több sorba illik írni.

Utasításblokk (összetett utasítás)

Mindenhova, ahova egy utasítást írhatunk, írhatunk helyette utasításblokkot is. Az utasításblokk szintaktikailag egy utasításnak számít. Alakja:

```
Utasításblokk kezdetét jelző kulcsszó
      Utasítások (bármennyi)
```

`Utasításblokk végét jelző kulcsszó`
A blokkok egymásba ágyazhatóak, azaz blokkon belül lehet újabb blokk.
A szokás az, hogy a blokkon belül levő utasításokat kicsit bentebb kezdjük, hogy jobban látszódjanak a blokk határai.
Egyes nyelvekben a vezérlő utasítások (elágazások, ciklusok) belsejében csak egy utasítás lehet, így ha szeretném, hogy több utasításra is vonatkozzon a vezérlő utasítás, akkor utasításblokkot kell használnunk (ami egy utasításnak számít, de benne több utasítás lehet).

Utasításfajták

Üres utasítás

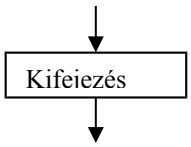
Akkor használjuk, ha valahova kell utasítást írni, de nem akarunk.

Kifejezés utasítás

A legegyszerűbb utasítás a kifejezés utasítás.
Alakja:

```
kifejezés;
```

Működése:
Kiértékelődik a kifejezés.
Egyes nyelvekben, csak olyan kifejezések lehetnek utasítások, amelyekben megváltozik egy memória terület tartalma, vagy függvényhívás (lásd köv fejezet) van. Eszerint nem lehet utasítás pl. az a kifejezés, hogy `5*alma`.
Folyamatábra megfelelője:



Értékadó utasítás:
Leggyakrabban használt kifejezés utasítás az értékadó utasítás. Az értékadó utasítás segítségével lehet egy változónak értéket adni.
Szintaktikája:

```
Változó értékadóoperátor kifejezés
```

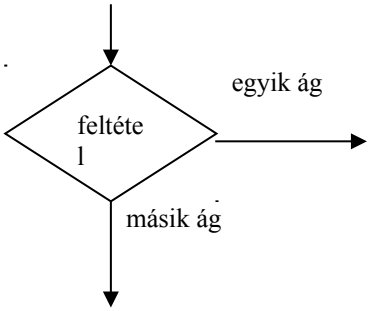
Pl. `változó = kifejezés`
Szemantikája (jelentése): a kifejezés kiértékelődik, majd a változó felveszi értékül a kifejezés eredményét. Mivel előbb értékelődik ki a kifejezés és csak utána változik meg a változó értéke, ezért a változó szerepelhet a kifejezésben és ott még a változó korábbi értékével számol.
Pl.

```
Alma=3
Alma=Alma+2
```

Előbb kiértékelődik a kifejezés, azaz Alma+2, azaz 3+2 egyenlő 5, majd az alma felveszi az 5 értéket.

Elágazásos utasítások

Létezik kétirányú elágazás és többirányú elágazás.
A kétirányú elágazás
Szerkezete:
Vezérlő rész, amelyben egy feltétel (logikai értékű kifejezés) található
A feltétel igaz értéke esetén végrehajtandó utasítások
A feltétel hamis értéke esetén végrehajtandó utasítások
Működése:
Kiértékelődik a feltétel, majd ha a feltétel igaz, akkor az igaz ági utasítások hajtódnak végre, ha nem igaz, akkor pedig a hamis ági utasítások.
Folyamatábra megfelelője:



A többirányú elágazás

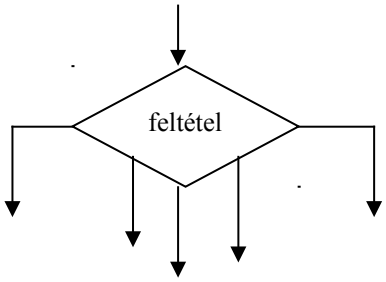
Szerkezete:

vezérlő rész, amelyben egy egész értékű kifejezés található
esetblokk kezdete
esetek és a hozzájuk tartozó utasítások
esetblokk vége

Működése:

Kiértékelődik az egészértékű kifejezés, majd sorban összehasonlítja az értékét a megadott esetekkel, az első egyezésnél végrehajtódnak az esethez tartozó utasítások. Utolsó esetként lehet egy alapértelmezett eset, amelyhez tartozó utasítások, akkor hajtódnak végre, ha egyik esettel sem egyezett meg a kifejezés értéke.

Folyamatábra jelölése:



Ciklusok

Utasítások ismételt végrehajtására.

Léteznek elöltesztelő és hátultesztelő, valamint léteznek iteratív és taxatív ciklusok. A nyelvek többségében három fajta ciklust használnak: elöltesztelő iteratív, hátultesztelő iteratív, és egy elöltesztelő taxatív ciklust.

Elöltesztelő iteratív ciklus:

Szerkezete:

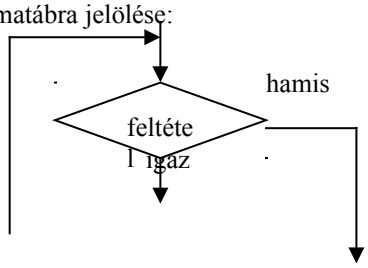
vezérlő rész, amelyben egy feltétel (logikai értékű kifejezés) található
ciklus mag, azaz az ismétlendő utasítások

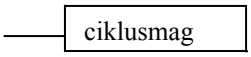
Működése:

kiértékeli a feltételt
ha igaz, akkor végrehajtja a ciklusmagot, majd ugrik vissza feltétel kiértékeléshez
ha hamis akkor az utasítás befejeződik

Tehát ez az utasítás addig ismételgeti a ciklusmagot, amíg igaz a feltétel.

Folyamatábra jelölése:





Hátultesztelő iteratív ciklus:

Szerkezete:

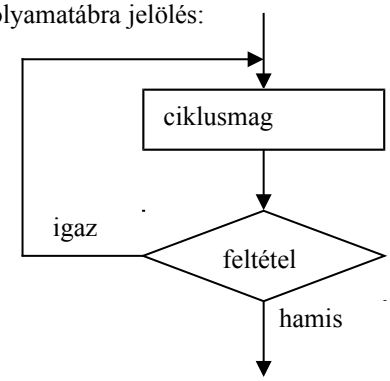
utasítás kezdetét jelző kulcsszó
ciklusmag, azaz az ismétlendő utasítások
vezérlő rész, amelyben egy feltétel található

Működése:

végrehajtja a ciklusmagot
kiértékeli a feltételt
ha igaz, akkor ugrik vissza a ciklusmag végrehajtásához, ha hamis, akkor befejeződik az utasítás.

Tehát ez az utasítás is addig ismételgeti a ciklusmagot, amíg igaz a feltétel.

Folyamatábra jelölés:



A különbség a két utasítás között abban van, hogy az elöltesztelő ciklus rögtön a feltétel kiértékeléssel kezdi, ezért ha a feltétel már kezdetben hamis, akkor egyszer sem hajtja végre a ciklusmagot. A hátultesztelő ciklus mivel az ciklusmag végrehajtásával kezdi, ezért legalább egyszer biztosan végrehajtja a ciklusmagot. A hátultesztelő ciklus egy gyakori alkalmazása, mikor a feltétel olyan változót tartalmaz, amelyik csak a ciklusmagon belül kap értéket.

Pl. Olvassunk be egy 10 és 20 közötti számot egy változóba.

a. beolvassuk a számot
b. ha a beolvasott szám nem esik 10 és 20 közé akkor ugrás újra a beolvasásra

Ezt a feladatot elöltesztelő ciklussal a következő képpen lehetne megoldani:

1. szám=1 (bármilyen nem 10 és 20 közé eső számot adhatunk értékül a számnak)
2. kiértékeli a feltételt azaz azt hogy a szám nem esik 10 és 20 közé
3. ha igaz, akkor beolvassuk a számot és ugrik vissza a feltétel kiértékeléshez, ha hamis, akkor befejeződik az utasítás

Az **iteratív** ciklusok közös jellemzője, hogy előre nem tudjuk, hogy hányszor fog végrehajtódni a ciklusmag, hogy végrefog-e még egyszer hajtódni az csak akkor fog kiderülni, amikor előzőleg lefutott a ciklusmag.

A **taxatív** ciklusok esetén előre tudhatjuk (a vezérlőrészből látható), hogy hányszor fog lefutni a ciklusmag.

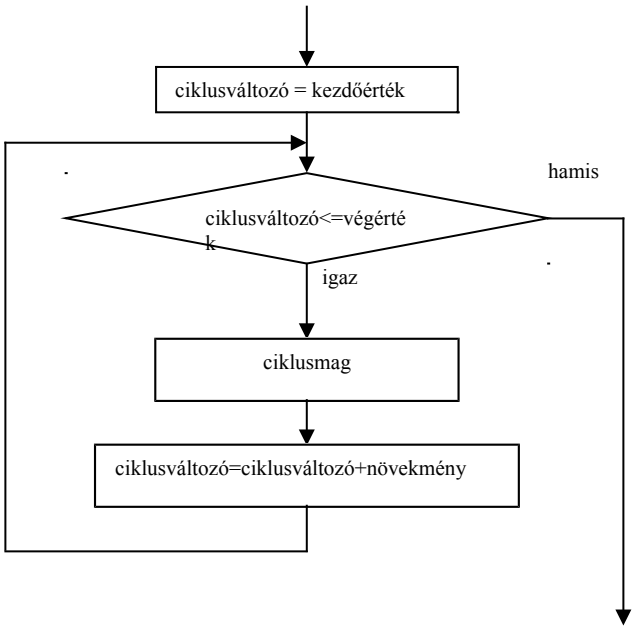
Előltesztelő taxatív ciklus:

Szerkezete:

Vezérlő rész, amelyben található egy változó (un. ciklus változó), valamint meg van adva egy kezdőérték, végérték, és a lépésköz
Ciklusmag

Működés:

A ciklusváltozó felveszi a kezdőértéket.
Kiértékelődik a feltétel, azaz hogy a ciklusváltozó kisebb vagy egyenlő a végértékkel
Ha igaz, akkor
    végrehajtódik a ciklusmag
    megnövelődik a ciklusváltozó értéke a lépésközzel
    ugrik vissza a feltétel kiértékelésre
Ha hamis, akkor vége a ciklusnak.



Taxatívként használható iteratív ciklus:
A C-szerű nyelvekben nincsen taxatív ciklus, viszont az elöltesztelő iteratív ciklusnak van egy olyan fajtája, amely taxatívként is használható:

Szerkezete:

Vezérlő rész, amelyben három kifejezés található. Az első kifejezés végrehajtódik a ciklus kezdete előtt, a második kifejezés a ciklus feltétele, a harmadik kifejezés végrehajtódik minden ciklusmag végrehajtás után.

Ciklusmag

Működése:

Végrehajtódik az első kifejezés
Kiértékelődik a második kifejezés
Ha igaz, akkor
végrehajtódik a ciklusmag
végrehajtódik a harmadik kifejezés
ugrik vissza a feltétel kiértékelésre
Ha hamis, akkor vége a ciklusnak

Ebben az utasításban, ha az első kifejezésben egy változót inicializálok, a második kifejezésbe egy olyan feltételt írok, amely ennek a változónak az értékét hasonlítja össze egy végértékkel, valamint a harmadik kifejezésbe a változó értékének a növelését írom, akkor ez az utasítás taxatív jellegű.

Ugró utasítások

A végrehajtás nem a következő, hanem egy másik utasításon fog folytatódni.
Létezik strukturált, és nem strukturált ugró utasítás.
A nem strukturált ugró utasítások használatával a program végrehajtás menete teljesen áttekinthetetlenné válik, így használatuk nem javasolt, sőt tilos. Az újabb nyelvekből már el is hagyták ezeket az utasításokat.
A strukturált ugró utasítás szerkezete olyan, hogy a program jól olvasható, érthető marad használata esetén is.
Strukturált ugró utasítások:

Alprogram hívás és abból visszatérés: lásd alprogramok fejezet.
Ciklusból kiugrás: ez az utasítás csak ciklusmagban fordulhat elő, hatására a ciklus befejeződik és a ciklus utáni utasításon folyatódik a végrehajtás.
Ciklus feltétel kiértékelésére ugrás: szintén csak ciklusmagban fordulhat elő, hatására azonnal visszaugrik a ciklus feltétel kiértékelésére.

Ellenőrző kérdések

1. Mi az utasításblokk? Mi a szerepe?
2. Ismertesse a kifejezés utasítást!
3. Mutassa be az elágazásos utasításokat!
4. Milyen fajta ciklusok léteznek?
5. Mikor célszerűbb a hátultesztelő ciklus alkalmazása?
6. Ismertesse az egyes ciklusfajtákat!
7. Milyen ugró utasítások léteznek?
8. Miért veszélyes a strukturálatlan ugró utasítások használata?

Feladatok

Készítse el a folyamatábráját a következő feladatoknak:

1. Beolvasva egy háromszög három oldalát, ha alkothat háromszöget kiszámolja és kiírja a kerületét, ha nem alkothat háromszöget, akkor kiírja, hogy hibás adatok!
2. Addig olvassa be az oldalértékeket, amíg helyes értékeket nem adnak meg és akkor kiszámolja és kiírja a háromszög kerületét!
3. Számkitaláló játék: előállít egy véletlenszerű számot (egy elemi lépés), bekér egy tippet, és kiírja, hogy kisebbet, nagyobb tippelt-e vagy eltalálta!

Hatáskör, élettartam

Hatáskör

Minél nagyobb egy program annál hátrányosabb, ha egy létrehozott azonosító (pl. változó neve) az egész programban ismert. Egyre nehezebb lesz megjegyezni mit mire használtunk és egyre könnyebb lesz eltéveszteni. Csapatmunka esetén pedig szinte lehetetlen összeegyeztetni a munkát.
Célszerű tehát egy azonosító hatáskörét minél kisebb területre korlátozni. A hatáskör alatt azon programrészt értjük, ahol az azonosító látható, használható.
A legtöbb programozási nyelvben a változók hatáskörét meghatározza a deklaráció helye, illetve a deklarálási kulcsszó.
Egy utasításblokk belsejében deklarált változó hatásköre a deklaráció helyétől az utasításblokk végéig tart, és kiterjed a blokkban levő blokkokra is. Nem látható viszont a blokkon kívűl.
Pl.

```
A blokk kezdete
      utasítás1
      utasítás2
blokk vége
B blokk kezdete
      utasítás3
      Alma változó deklarálása
      utasítás4
      C blokk kezdete
            utasítás5
            utasítás6
      blokk vége
      utasítás7
blokk vége

Az Alma változó nem látható az A blokkban valamint a B blokkban a
deklaráció előtt, azaz az utasítás3-ban. Látható viszont az utasítás4-ben,
a teljes C blokkban, utasítás7-ben.
```

Ha egy blokkban látható egy blokkon kívül deklarált változó, akkor szokás mondani, hogy a változó arra a blokkra nézve globális. (Az Alma a C blokkra globális.)
Ha a program összes blokkjában látható, akkor a programra globális. Az ilyen változók használata veszélyes és nem nagyon ajánlott.
Ha egy változó hatásköre egy blokkon kívülre nem terjed ki, akkor a változó a blokkra lokális. (Az Alma a B blokkra lokális.)

Élettartam

A változók élettartamának azt nevezzük, amíg foglalja a memóriaterületet. Élettartam szempontjából léteznek statikus változók, amelyek a program során végig élnek, azaz foglalják a területet, és léteznek dinamikus változók, amelyek még a program befejeződése előtt elpusztulnak, azaz nem foglalják tovább a területet.
Azok a változók, amelyek hatásköre egy blokkra lokális legtöbbször dinamikusak is, azaz a blokkban a deklarációkor jönnek létre és a blokk végén el is pusztulnak.
Léteznek blokkra lokális statikus változók is. Ezek a blokkot elhagyva nem használhatók, de nem pusztulnak el, a blokkba visszatérve, újra használhatóak lesznek, korábbi értékük megmarad.

Ellenőrző kérdések

1. Mi a hatáskör szerepe?
2. Hogyan adhatjuk meg a hatáskörét egy változónak?
3. Mit jelent az, hogy egy változó egy blokkra lokális?
4. Mit jelent az, hogy egy változó egy blokkra globális?
5. Mit jelent az, hogy egy változó (programra) globális?
6. Mit nevezünk élettartamnak? Élettartam szempontjából milyen fajta változók léteznek?

Alprogramok (szubrutinok)

Elnevezett, paraméterezhető utasítássorozat.

A szubrutin definiálása: meg kell adni a fejrészt és az utasításblokkot.
Fejrész: tartalmazza a szubrutin nevét, paramétereit, valamint egyéb szubrutinra vonatkozó információkat.
Szubrutin utasítás blokkja: tartalmazza a szubrutinhoz tartozó utasításokat (lehetnek deklarációs utasítások is közöttük).
Szubrutin hívása:
A szubrutinra lehet hivatkozni akárhányszor a program más részein a neve és aktuális paraméterei megadásával, ezt nevezzük a szubrutin meghívásának.
Ha a program végrehajtáskor egy szubrutin hívásához ér, akkor először végrehajtódik egy paraméterátadás, azaz a szubrutin paraméterei felveszik a szubrutin hívásában megadott aktuális értékeiket (lásd még paraméterátadás), majd a szubrutin utasításai hajtódnak végre.
A szubrutin hívása lehet önálló utasítás, ilyenkor a szubrutin utasításainak végrehajtása után a szubrutin hívását követő utasításon folytatódik a végrehajtás. A szubrutin hívása lehet egy kifejezésben is, ilyenkor a szubrutin utasításainak végrehajtása után folytatja a kifejezés kiértékelését.

A szubrutinok használatának előnyei:

- az ismételten végrehajtandó kódrészletet elég egyszer megírni (leellenőrizni) és akárhányszor meghívható,
- olvashatóbbá válik a program,
- a feladat önállóan megvalósítható részfeladatokra bontható, amelyet esetleg különböző személyek valósítanak meg,
- szubrutin könyvtárak hozhatók létre, ezzel felhasználhatjuk más programokban is a már jól bevált kódunkat.

Szubrutin definíciós rész

A programok tehát tartalmaznak szubrutin definíciós részt, valamint úgynevezett főprogramot, aminek utasításain elindul a végrehajtás.
Bizonyos nyelvekben (pl. Pascal) nem csak egy szubrutin definíciós rész lehet, hanem a szubrutinoknak lehet saját szubrutin definíciós része, amelyben csak általuk látható szubrutinok definiálhatók.
A főprogram hívhat szubrutinokat, de természetesen a szubrutinok is hívhatnak szubrutinokat.
Egyes nyelvekben a szubrutin definiálásának sorrendje nem mindegy, hiszen egy szubrutin csak attól kezdve létezik, ahol definiáltunk. Azaz egy szubrutin nem tud meghívni egy olyan szubrutint, amely később lett

definiálva. Az egymásra hivatkozó szubrutinok esetén nem lehetne megfelelő sorrendet kialakítani, ezért lehet hivatkozni olyan szubrutinra is, amelyet még nem definiáltak, de már deklaráltak.
Szubrutinok deklarálása: A szubrutinnak csak a fejrészét adjuk meg, törzsét (az utasításblokkot) nem. Természetesen később a programban valahol még definiálni is kell a szubrutint.
Például:

```
BB szubrutin fejrésze (BB szubrutin deklarálása)

AA szubrutin fejrésze
Utasításblokk kezdete
      …..
      BB szubrutin hívása
      …..
Utasításblokk vége

BB szubrutin fejrésze
Utasításblokk kezdete
      …….
Utasításblokk vége
```

Szubrutinok fajtái

A szubrutinoknak két fajtája van a függvény és az eljárás.

Függvény

A függvény egy értéket állít elő, amely behelyettesítődik a függvényhívás helyére. Tehát a függvényt meg lehet hívni egy kifejezés belsejében. A kifejezés kiértékelése közben talál rá a függvényhívásra, ekkor végrehajtódik a függvény, majd az előállított érték behelyettesítődik a kifejezésbe a függvényhívás helyére és folytatja a kifejezés kiértékelését. A függvény meghívható önálló utasításként is (kivéve néhány programozási nyelvet), ilyenkor a visszatérési érték nem kerül felhasználásra. A függvény fejlécében mindig jelezni kell a visszatérő érték típusát, hogy a fordító megállapíthassa megfelelő műveleteket végzünk-e a visszatérési értékkel.
Egy fiktív nyelven bemutatva a függvényhívás (Az egyszerűség kedvéért a kifejezés egy értékadó utasítás lesz. Természetesen bármilyen kifejezésben meghívható függvény. Pl elágazások feltételeiben, vagy ciklusok vezérlő részében is.):

```
Függvény narancs visszatérési értéke egy egész szám
Utasításblokk kezdete
      Utasítás1
      Utasítás2
      (Az utasítások valamelyike meg kell adja a visszatérési értéket.)
Utasításblokk vége

Főprogram
Utasításblokk kezdete
      Utasítás3
      Utasítás4:A=5*8+narancshívása/2
      Utasítás5
Utasításblokk vége
```

Végrehajtás menete:

```
Utasítás3
Utasítás4 végrehajtása: a kifejezés kiértékelési sorrendből rögtön látható
      Végrehajtja az 5*8
      Majd meghívja a narancs-ot,
            Utasítás1
            Utasítás2
```

A végrehajtás után a narancs helyére behelyettesítődik a visszatérési értéke

```
      Végrehajtja a narancsvisszatérésiértéke/2
      Végrehajtja az összeadást
      Végül az értékadást
```

```
Utasítás5
```

A függvényt tehát akkor célszerű használni, ha a szubrutin célja egy érték előállítása.

Eljárás

Az eljárás nem állít elő olyan értéket, amely behelyettesítődne a hívás helyére, így az eljáráshívás csak önálló utasítás lehet. Az eljárás végrehajtása után az eljáráshívást követő utasításon folytatódik a végrehajtás. Egy példa fiktív nyelven az eljárás jellegű szubrutinok végrehajtására:

```
Eljárás Alma
Utasításblokk kezdete
      Utasítás1
      Utasítás2
      Utasítás3
Utasításblokk vége

Eljárás Korte
Utasításblokk kezdete
      Utasítás4
      Alma szubrutin hívása
      Utasítás5
      Utasítás6
Utasításblokk vége

Főprogram
Utasításblokk kezdete
      Utasítás7
      Alma eljárás hívása
      Korte eljárás hívása
      Utasítás8
      Utasítás9
Utasításblokk vége
```

Végrehatási sorrend:

```
Utasítás7
Utasítás1
Utasítás2
Utasítás3
Utasítás4
Utasítás1
Utasítás2
Utasítás3
Utasítás5
Utasítás6
Utasítás8
Utasítás9
```

Paraméterek, paraméterátadás

A szubrutinok a paramétereken keresztül képesek kommunikálni az őt meghívóval. Egy szubrutin nem láthatja az őt hívó szubrutin változóit, hiszen azok hatásköre csak saját utasításblokkjukra korlátozódik. Ha tehát szüksége van a hívott szubrutinnak valami kiinduló adatra, vagy esetleg a hívó szubrutin valamelyik változójának értékét kellene megváltoztatnia a hívott szubrutinnak (pl. eredményvisszaadás), akkor ezt a paramétereken keresztül teheti meg (vagy olyan változókkal, amelyek mindkettőjük számára látható).
A szubrutin paramétereit a szubrutin neve után kell zárójelben deklarálni.

```
Szubrutin Szilva(param1 deklarációja, param2 deklarációja)
Utasításblokk kezdete
…….
```

```
Utasításblokk vége
```

Ha olyan szubrutint hívunk meg, amelynek vannak paraméterei, akkor a híváskor meg kell adni a paraméterek aktuális értékeit:

```
Szilva(1.paraméter értéke, 2. paraméter értéke)
```

A szubrutin definíciójában szereplő paramétereket **formális paramétereknek** szokás nevezni, a szubrutin meghívásában szereplő értékeket pedig **aktuális paramétereknek**.

A legtöbb programozási nyelvben, akkor is ki kell írni a zárójeleket, amikor nincs paramétere a szubrutinnak.

```
Szubrutin Szollo()
Utasításblokk kezdete
……..
Utasításblokk vége
```

Illetve a hívás:
```
Szollo()
```

Paraméterek hatásköre és élettartama: A szubrutinok formális paramétereinek hatásköre a szubrutin utasításblokkja, élettartama pedig dinamikus, hiszen a szubrutin hívásakor jön létre és a szubrutin elhagyásakor elpusztul, azaz felszabadul a lefoglalt terület.

Paraméterátadás fajtái

Legtöbb programozási nyelvben (C, C++, Java, stb.) csak az úgynevezett értékszerinti paraméterátadás létezik, míg más nyelvekben (pl. pascal, visual basic) létezik címszerinti paraméterátadás is.
Értékszerinti paraméterátadás:
Ennél a paraméterátadás fajtánál az aktuális paraméterek tetszőleges kifejezések lehetnek és értékeiket rendre (első az elsőnek, második a másodiknak) átadják a formális paramétereknek. A hívott függvény nem képes megváltoztatni az aktuális paraméterek értékeit, hiszen számára azok nem láthatóak. Megoldható viszont, hogy a hívott megváltoztasson mégis egy hívóbeli értéket (azaz eredményt adjon vissza) a mutatók vagy a referenciák használatával.
(A mutató egy olyan változó, amely egy memóriacímet képes tárolni, valamint képes az adott címen lévő adatot manipulálni.)
Ha a formális paraméter egy mutató és a hívó ennek a mutatónak, egy változójának a címét adja át, akkor a hívott képes megváltoztatni a hívó egy változójának értékét.
Ha a formális paraméter egy referencia változó, akkor képes felvenni egy változó referenciáját, azaz minden műveletben, amelyben szerepel, maga helyett a hivatkozott (akinek a referenciáját felvette) változó fog szerepelni, azaz úgy fog viselkedni, mintha a hivatkozott változónak egy másik neve lenne.
Mutatókat vagy referenciákat célszerű használni akkor is, ha nagyobb adatot akarunk átadni (pl tömböt vagy struktúrát), hiszen gyorsabb átadni egy címet (vagy referenciát).

Címszerinti paraméterátadás:
Egyes nyelvek (pl. pascal, visual basic) ismerik a címszerinti paraméterátadást is.
Ennél a paraméterátadásnál a formális paraméter számára ugyanaz a memória terület lesz kijelölve, mint amelyet az párjaként szereplő aktuális paraméter használ. Így bármilyen műveletet végzünk a formális paraméterrel az kihatással van az aktuális paraméterre, azaz ha az előállított eredményt egy címszerint átadott paraméterbe pakoljuk, akkor azt a hívó is megkapja az aktuális paraméteren keresztül. A szubrutin lefutásakor a címszerint átadott formális paraméter területe nem szabadul fel, csak az azonosító szűnik meg.

Ellenőrző kérdések, feladatok

1. Mi a szubrutin? Használatuk előnyei?
2. Írja le a szubrutinok definiálásának és hívásának szintaktikáját.
3. Mi a szubrutinok hatásköre egyszintű szubrutin definíciós rész esetén, és mi a hatáskör, ha a szubrutinnak lehet saját szubrutin definíciós része?

4. Hogyan deklarálhatunk szubrutinokat?
5. Mi a különbség a függvény és az eljárás között?
    a. Hogyan kell definiálni egy függvényt és hogyan kell egy eljárást?
    b. Hogyan hívható meg egy függvény és hogyan hívható meg egy eljárás?
    c. Hol folytatódik a végrehajtás függvényhívás illetve eljáráshívás után?
    d. Mikor célszerű függvény és mikor célszerű eljárás használata?
    e. Hogyan képes eredmény(eke)t visszaadni egy függvény és hogyan egy eljárás?
6. Mire valók a paraméterek?
7. Mit nevezünk formális és aktuális paraméternek?
8. Hogyan zajlik a paraméterátadás értékszerinti paraméterátadásnál?
9. Hogyan zajlik a paraméterátadás címszerinti paraméterátadásnál?
10. Hogyan lehet eredményt visszaadni paraméterek segítségével, ha csak értékszerinti paraméterátadás létezik.

Feladatok:
A feladatok megoldásához használjon egy olyan fiktív nyelvet, amelyben
típusok: egész, valós, logikai
függvény fejléc alakja: `függvény típus név(paraméterek)`
eljárás fejléc alakja: `eljárás név(paraméterek)`
értékszerint átadott paraméter: `típus név`
címszerint átadott paraméter: `C:típus név`

1. Írja le egy olyan szubrutin fejrészét, amely egy háromszög oldalaiból kiszámolja és visszaadja a kerületet, területet!

   Megoldás: Nem egy eredmény állít elő, így célszerűbb az eljárás használata. Az eljárás címszerinti átadott paraméterekkel képes eredményt visszaadni a hívónak. Tehát:
   ```
   eljárás kerter(egész a, egész b, egész c, C:egész ker, C:valós ter)
   ```

2. Írja le egy olyan szubrutin fejrészét, amely három egész számról megállapítja, hogy lehetnek-e egy háromszög oldalhossz értékei!
3. Írja le egy olyan szubrutin fejrészét, amely egy paraméterként kapott valós szám kétszeresét adja vissza!
4. Írja le egy olyan szubrutin fejrészét, amely egy paraméterként kapott változó értékét megduplázza!

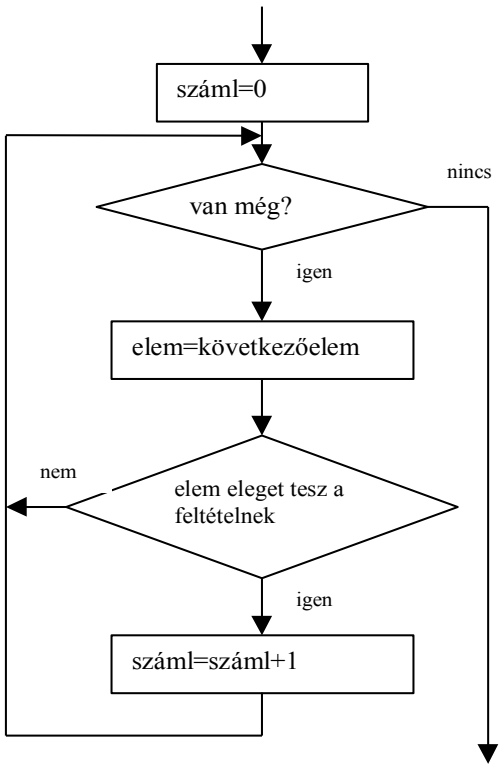
Tipikus algoritmusok (alap algoritmusok)

Létezik néhány igen gyakran használt algoritmus. Mégpedig a
- számlálás,
- összegzés, átlagképzés
- minimum vagy maximum keresés
- rendezés: (ebben a félévben nem tananyag)

A számlálás

Feladat: meg kell számolni, hogy egy adathalmazból hány elem tesz eleget egy feltételnek. (Az adatok legtöbbször egy tömbben szoktak lenni.)
Megoldás: szükség van egy számláló változóra, amely miközben végigvizsgálom az összes elemet, mindig azt mutatja, hogy eddig hány elem tett eleget a feltételnek.
Algoritmus leírása szavakban:
1. A számláló változó értékét nullára állítom
2. Megnézem van-e még megvizsgálatlan elem
3. ha igen, akkor a veszem a következő elemet és megnézem eleget tesz-e a feltételnek
   ha igen, akkor növelem a számláló változó értékét egyel
   vissza a 2. pontra
4. ha nem, akkor vége a ciklusnak, a számláló most már a végeredményt tartalmazza

Algoritmus leírása folyamatábrával:

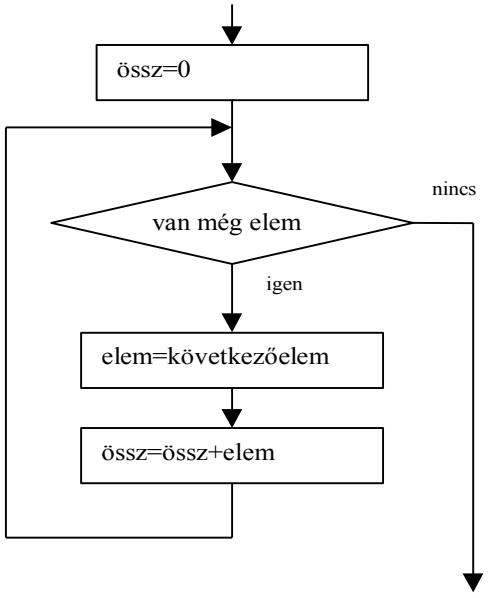


Összegzés, átlagszámítás

Feladat: Össze kell adni egy adathalmaz elemeit. Az átlagszámítás csak annyi különbséget jelent, hogy a végén az összeget még el kell osztani az elemek számával.
Megoldás: szükség van egy változóra (nevezzük összegző változónak), amelyhez sorban az összes elemet hozzáadom.
Algoritmus leírása szavakban:

1. Az összegző változót kinullázzuk.
2. megnézem van-e még hozzáadatlan elem
3. ha igen, akkor veszem a következő elemet és az összegző változó értékét megnövelem ezzel az elemmel, majd vissza a 2. pontra.
4. ha nem, akkor vége a ciklusnak és az összegző változóm már tartalmazza az összes elem összegét.

Algoritmus leírása folyamatábrával:

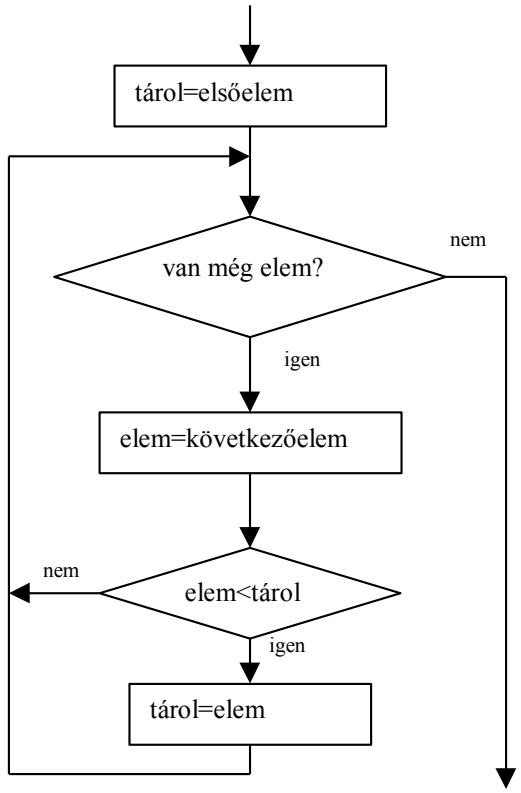


Minimum (maximum) keresés

Feladat: egy adathalmaz elemei közül keressük meg a legkisebbet (legnagyobbat).
Megoldás: szükség van egy változóra, amely miközben végigvizsgáljuk az elemeket, tartalmazza az eddig megtalált legkisebb elemet (vagy az azonosítóját, pl indexét, vagy az adathalmazbeli pozícióját).
Algoritmus leírása szavakban:

1. A tároló változóban eltároljuk az első elemet
2. megnézem van-e még megvizsgálatlan elem
3. ha igen, akkor veszem a következő elemet és megnézem, hogy kisebb-e (nagyobb-e, maximumkeresésnél) a tároló változóban eltárolt elemnél
    4. ha igen, akkor ezt az elem tárolom el a tároló változóba
    5. vissza a 2. pontra
6. ha nem, akkor vége a ciklusnak, és a tároló változóm a legkisebb (legnagyobb) elemet fogja tartalmazni.

Algoritmus leírása folyamatábrával:



Ellenőrző kérdések, feladatok

1. Milyen alapalgoritmusokat ismer?
2. Ismertesse a számlálás algoritmust!
3. Ismertesse az összegzés algoritmust!
4. Ismertesse az átlagszámítás algoritmust!
5. Ismertesse a minimum (maximum) keresési algoritmust!