1. Bevezetés

Modellezés

A valós világban mindenféle objektumok (személyek, tárgyak, intézmények, számítógépes programok) vannak – **nevezzük ezeket egyedeknek**. Az egyedeknek egyrészt rájuk jellemző tulajdonságaik vannak, másrészt közöttük bonyolultabb kapcsolatrendszer áll fenn.

A valós világ egyedei, közös tulajdonságaik és viselkedési módjuk alapján **kategorizálhatóak, osztályozhatóak**.

Az absztrakció lényege, hogy kiemeljük a közös, lényeges tulajdonságokat és viselkedésmódokat, az eltérőeket, lényegteleneket pedig elhanyagoljuk. Ezáltal létrejön **a valós világ modellje**, amely már nem az egyes egyedekkel, hanem az egyedek egy csoportjával, osztályával foglalkozik.

Az ember modelleket használ, amikor egy megoldandó problémán gondolkodik, amikor beszélget valakivel stb. Egy modellel szemben **három követelményt** szoktak támasztani:

1. **Leképzési követelmény**: Léteznie kell olyan egyednek, amelynek a modellezését végezzük. Ez az „eredeti egyed".
2. **Leszűkítési követelmény**: Az eredeti egyed nem minden tulajdonsága jelenik meg a modellben, csak bizonyosak.
3. **Alkalmazhatóság követelménye**: A modellnek használhatónak kell lennie, azaz a benne levont következtetéseknek igaznak kell lenniük, ha azokat visszavetítjük az eredeti egyedre.

Az egyedek **tulajdonságait** számítógépen **tárolt adatokkal**, a **viselkedésmódot pedig programokkal** tudjuk kezelni – ezzel természetesen szintén egyfajta modellt megadva. **Így beszélhetünk adatmodellről és funkcionális modellről (eljárásmodellről).**

Alapfogalmak

A számítógépek programozására kialakult nyelveknek három szintjét különböztetjük meg:

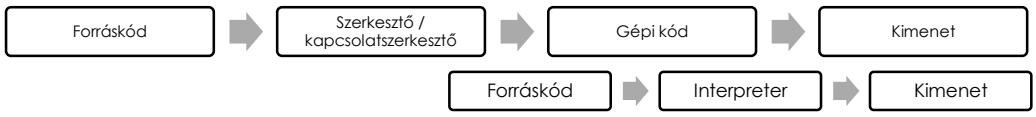
1. Gépi nyelv
2. Assembly szintű nyelv
3. Magas szintű nyelv

A magas szintű nyelven megírt programot **forrásprogramnak, vagy forrásszövegnek** nevezzük. A forrásszöveg összeállítására vonatkozó formai „nyelvtani" szabályok összességét **szintaktikai szabályoknak** hívjuk. A tartalmi, értelmezési, jelentésbeli szabályok alkotják a **szemantikai szabályokat**. **Egy magas szintű programozási nyelvet szintaktikai és szemantikai szabályainak együttese határoz meg.**

Minden processzor rendelkezik saját gépi nyelvvel, és csak az adott gép nyelven írt programokat tudja végrehajtani. A magas szintű nyelven megírt forrásszövegből tehát valamilyen módon gépi nyelvű programokhoz kell eljutni. Erre kétféle technika létezik, a **fordítóprogramos** és az **interpreteres.**

A **fordítóprogram** egy speciális program, amely a magas szintű nyelven megírt forrásprogramból gép kódú **távprogramot** állít elő. A fordítóprogramot egyetlen egységként kezeli, és működése közben a következő **lépéseket** hatja végre:

- Lexikális elemzés: A forrásszöveg feldarabolása lexikális egységekre
- Szintaktikai elemzés: Ellenőrzi, hogy teljesülnek-e az adott nyelv szintaktikai szabályai
- Szemantikai elemzés:
- Kódgenerálás



Az **interpreteres** technika esetén is meg van az első három lépés, de az interpreter nem készít tárgyprogramot. Utasításonként (vagy egyéb nyelvi egységenként) sorra veszi a forrásprogramot, értelmezi azt, és végrehajtja. Rögtön kapjuk az eredményt, úgy, hogy lefut valamilyen gépi kódú rutin.

Az egyes programnyelvek vagy fordítóprogramosak, vagy interpreteresek, vagy együttesen alkalmazzák mindkét technikát. Minden programnyelvenk megvan a saját szabványa, amit hivatkozási nyelvnek hívunk. A hivatkozási nyelv mellett (néha vele szemben) léteznek az **implementációk**. Ezek egy adott platformon (processzor, operációs rendszer) realizált fordítóprogramok, interpreterek.

Napjainkban a programok írásához **grafikus integrált fejlesztői környezetek** (IDE – Integrated Development Enviroment) állnak rendelkezésünkre. Ezek tartalmaznak szövegszerkesztőt, fordítót (esetleg interpretert), kapcsolatszerkesztőt, betöltőt, futtató rendszert és belövőt.

Programnyelvek osztályozása

Imperatív nyelvek

- **Algoritmikus nyelvek:** A programozó mikor egy programszöveg leír, algoritmust kódol, és ez az algoritmus működteti a processzort.
- A programok **utasítások** sorozata.
- A legfőbb programozási eszköz a **változó**, amely a tár közvetlen elérését biztosítja, lehetőséget ad az abban lévő értékek közvetlen manipulálására. Az algoritmus a változók értékeit alakítja, tehát a program a hatását a tár egyes területein lévő értékeken fejti ki.
- Szorosan kötődnek a Neumann-architektúrához.
- **Alcsoportjai**:
    - Eljárásorientált nyelvek
    - Objektumorientált nyelvek

Deklaratív nyelvek

- Nem algoritmikus nyelvek.
- Nem kötődnek szorosan a Neumann-architektúrához, mint az imperatív nyelvek.
- A programozó csak a problémáit adja meg, a nyelvi implementációkba van beépítve a megoldás megkeresésének módja.
- A programozónak nincs lehetősége memóriaműveletekre, vagy csak korlátozott módon.
- **Alcsoportjai**:
    - Funkcionális (applikatív) nyelvek
    - Logikai nyelvek

2. Alapelemek

Karakterkészlet

A forrásszöveg összeállításánál alapvető a **karakterkészlet**, ennek elemei jelenhetnek meg az adott nyelv programjaiban, ezekből állíthatók össze a bonyolultabb nyelvi elemek. Az eljárásorientált nyelvek esetén ezek a következők:

- **Lexikális egységek**: A lexikális egységek a program szövegének azon elemei, melyeket a fordító a lexikális elemzés során felismer és tokenizál (közbenső formára hoz). Fajtái a következők:
    - **Többkarakteres szimbólum** (//, ++, --, &&, /*, */)
    - **Szimbolikus név**
        - Azonosító(x)
        - Kulcsszó (alapszavak: if, for, case, break)
        - Standard azonosító (beépített függvények nevei)
    - **Címke**: Az eljárorientált nyelvekben a végrehajtható utasítások megjelölésére szolgál
    - **Megjegyzés**: Segítségével a programban olyan karaktersorozat helyezhető el, amely nem a fordítónak szól, hanem a program szövegét olvasó embernek
    - **Literál** ('c'): Mindig önmagát definiálja.
- **Szintaktikai egységek**
- **Utasítások**
- **Programegységek**
- **Fordítási egységek**
- **Program**

A forrásszöveg összeállításának általános szabályai

A **forrásszöveg**, mint minden szöveg, sorokból áll. Kérdés, hogy milyen szerepet játszanak a sorok a programnyelvek szempontjából.

Kötött formátumú nyelvek

A korai nyelveknél alapvető szerepet játszott a sor. Egy sorban egy utasítás helyezkedett el, tehát a sorvége jelezte az utasítás végét.

Szabadformátumú nyelvek

Ezeknél a nyelveknél a sornak és az utasításnak semmi kapcsolata nincs egymással. A sorvége nem jelenti az utasítás végét. Éppen ezért ezek a nyelvek bevezetik az utasítás végjelet, ez elég általánosan a pontosvessző. **Tehát a forrásszövegben két pontosvessző között áll egy utasítás**.

Adattípusok

Az absztrakció első megjelenési formája az **adattípus** a programozási nyelvekben. Az adattípus maga egy **absztrakt programozási eszköz**, amely mindig más, konkrét programozási eszköz komponenseként jelenik meg.

A programozási nyelvek egy része ismeri ezt az eszközt, más része nem. Ennek megfelelően beszélünk **típusos és nem típusos nyelvekről**. Az eljárásorientált nyelvek típusosak. Egy adattípust három dolog határoz meg, ezek:

1. **Tartomány:** Az **adattípusok tartománya** azokat az elemeket tartalmazza, amelyeket az adott típusú konkrét programozási eszköz fölvehet értékként. Bizonyos típusok esetén a tartomány elemei jelenhetnek meg a programban literálként.
1. **Műveletek:** Az adattípushoz hozzátartoznak azok a **műveletek**, amelyeket a tartomány elemein végre tudunk hajtani.
2. **Reprezentáció:** Minden adattípus mögött van egy megfelelő belső ábrázolási mód. A reprezentáció az egyes típusok tartományába tartozó értékek tárban való megjelenését határozza meg, tehát azt, hogy az egyes elemek hány bájtra és milyen bitkombinációra képződnek le.

Minden típusos nyelv rendelkezik **beépített (standard) típusokkal**. Egyes nyelvek lehetővé teszik azt, hogy a **programozó is definiálhasson típusokat**. **Saját típust úgy tudunk létrehozni, hogy** megadjuk a tartományát, a műveleteit és a reprezentációját. De lehet létrehozni már beépített adattípus segítségével.

1. Adattípusok

bool 1, int 4, long int 8, float 4 double 8, char 1, short int 2

Ha létrehozunk egy saját adattípust pl. extra short int az intből, akkor a műveletei, a reprezentációi is jönnek vele, a tartománya pedig a szűkített / bővített részre változik

Az egyes adattípusok, mint programozási eszközök önállóak, **egymástól különböznek**. Van azonban egy speciális eset, **amikor egy típusból (ez az alaptípus) úgy tudok származtatni egy másik típust (ez lesz az altípus**), hogy leszűkítem annak tartományát, változatlanul hagyva műveleteit és reprezentációját. **Az alaptípus és az altípus tehát, nem különböző típusok.**

Az adattípusoknak két nagy csoportjuk van:

Egyszerű típusok

A **skalár vagy egyszerű** adattípus tartománya atomi értékeket tartalmaz, minden érték egyedi, közvetlenül nyelvi eszközökkel tovább nem bontható. A skalár típusok tartományaiból vett értékek jelenhetnek meg literálként a program szövegében.

Minden nyelvben létezik az **egész típus**, sőt általában egész típusok**. Ezek belső ábrázolása fixpontos**. **Néhány nyelv ismeri az előjel nélküli egész típust**, ennek belső ábrázolása előjel nélküli (direkt).

Alapvetőek a **valós típusok, belső ábrázolásuk lebegőpontos**. A tartomány itt is az alkalmazott ábrázolás függvénye, ez viszont általában implementációfüggő.

Az egész és valós típusokra közös néven, mint **numerikus** típusokra hivatkozunk. A numerikus típusok értékein a numerikus és hasonlító műveletek hajthatóak végre.

A **karakteres** típus tartományának elemei karakterek, a karakterlánc vagy sztring típuséi pedig karaktersorozatok.

Egyes nyelvek ismerik a **logikai** típust. Ennek tartománya a hamis és igaz értékekből áll, műveletei a logikai és hasonlító műveletek, belső ábrázolása logikai.

Speciális egyszerű típus a **felsorolás** típus. A típus definiálása úgy történik, hogy megadjuk a tartomány elemeit. Ezek azonosítók lehetnek. Az elemekre alkalmazhatók a hasonlító műveletek.

A **sorszámozott** típusba tartoznak általában az egész, karakteres, logikai és felsorolás típusok. A tartományainak elemei listát alkotnak, azaz van első és utolsó elem. Az elemek között egyértelmű sorrend értelmezett. Egy sorszámozott típus altípusaként lehet származtatni az **intervallum** típust.

Összetett típusok

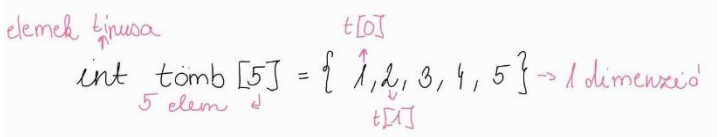
A **strukturált vagy összetett** adattípusok tartományainak elemei maguk is valamilyen típussal rendelkeznek. Az elemek egy-egy értékcsoportot képviselnek, nem atomiak, az értékcsoport elemeihez különkülön is hozzáférhetünk. Általában valamilyen absztrakt adatszerkezet programnyelvi megfelelői.

Az eljárásorientált nyelvek két legfontosabb összetett típusa a **tömb** és a **rekord.**

Tömb

A tömb típus a tömb absztrakt adatszerkezet megjelenése típus szinten. A tömb statikus és homogén összetett típus, vagyis tartományának elemei olyan értékcsoportok, amelyekben az elemek száma azonos, és az elemek azonos típusúak. A tömböt, mint típust meghatározza:

- Dimenziók száma
- Indexkészletének típusa és tartománya
- Elemeinek típusa



A többdimenziós tömbök reprezentációja lehet sor- vagy oszlopfolytonos.

Rekord

A rekord típus a rekord absztrakt adatszerkezet megjelenése típus szinten. A rekord típus minden esetben heterogén, a tartomány elemei olyan értékcsoportok, amelyeknek elemei különböző típusúak lehetnek. Az értékcsoporton belül az egyes elemeket **mezőnek** nevezzük. Minden mezőnek saját, önálló neve (ami egy azonosító) és saját típusa van.

Mutató típus

A mutató típus lényegében egyszerű típus, specialitását az adja, hogy tartományának elemei **tárcímek**. A mutató típus segítségével valósítható meg a programnyelvekben az indirekt címzés. A mutató típus **tartományának** van egy speciális eleme, amelyen nem valódi tárcím (NULL). **A mutató típus alapvető szerepet játszik az absztrakt adatszerkezetek szétszórt reprezentációját kezelő implementációknál.**

Nevesített konstans

A nevesített konstans olyan programozási eszköz, amelynek **három komponense van**:

1. **Név**
2. **Típus**
3. **Érték**

A nevesített konstanst mindig deklarálni kell. A program szövegében a nevesített konstans a nevével jelenik meg, és az mindig az értékkomponenst jelenti. A nevesített konstans értékkomponense a deklarációnál eldől, és **nem változtatható meg** a futás folyamán.

A változó

A változó olyan programozási eszköz, amelynek **négy komponense van:**

1. **Név:** A név egy azonosító. A program szövegében a változó mindig a nevével jelenik meg.

2. **Attribútumok:** Olyan jellemzők, amelyek a változó futás közbeni viselkedést határozzák meg. Változóhoz attribútumok deklaráció segítségével rendelődnek. **Ennek 3 fajtája van**:
    a. **Explicit deklaráció:** A programozó végzi explicit deklarációs utasítás segítségével. A változó teljes nevéhez kell az attribútumokat megadni.
    b. **Implicit deklaráció:** A programozó végzi, betűkhöz rendel attribútumokat egy külön deklarációs utasításban. Ha egy változó neve nem szerepel explicit deklarációs utasításban, akkor a változó a nevének kezdőbetűjéhez rendelt attribútumokkal fog rendelkezni, tehát az azonos kezdőbetűjű változók ugyanolyan attribútumúak lesznek
    c. **Automatikus deklaráció:** A fordítóprogram rendel attribútumot azokhoz a változókhoz, amelyek nincsenek explicit módon deklarálva, és a kezdőbetűjükhöz nincs attribútum rendelve egy implicit deklarációs utasításban. Az attribútum hozzárendelése a név valamelyik karaktere alapján történik.
3. **Cím:** A változó címkomponense a tárnak azt a részét határozza meg, ahol a változó értéke elhelyezkedik. A futási idő azon részét, amikor egy változó rendelkezik címkomponenssel, a változó **élettartamának** hívjuk. A változóhoz cím rendelhető az alábbi módokon:
    a. **Statikus tárkiosztás:** A futás előtt eldől a változó címe, és a futás alatt az nem változik. Amikor a program betöltődik a tárba, a statikus tárkiosztású változók fix tárhelyre kerülnek.
    b. **Dinamikus tárkiosztás:** A cím hozzárendelését a futtató rendszer végzi. A változó akkor kap címkomponenst, amikor aktivizálódik az a programegység, amelynek ő lokális változója, és a címkomponens megszűnik, ha az adott programegység befejezi a működést. A címkomponens a futás során változhat, sőt vannak olyan időintervallumok, amikor a változónak nincs címkomponense.
    c. **Programozó által vezérelt tárkiosztás:** A változóhoz a programozó rendel címkomponenst futási időben. Három alapesete van:
        i. A programozó **abszolút** címet rendel a változóhoz, konkrétan megadja, hogy hol helyezkednek el. (pl. cím x = 0XF711)
        ii. Egy már korábban a tárban elhelyezett programozási eszköz címéhez képest mondja meg, hogy hol legyen a változó elhelyezkedve, vagyis **relatív** címet ad. (pl. int x = 1; -> 0X01, cím y = x +1; -> 0X02, delete cím y; -> 0)
        iii. A programozó csak azt adja meg, hogy mely időpillanattól kezdve legyen az adott változónak címkomponense, az elhelyezést a futtató rendszer végzi.

    Mindhárom eseteben lennie kell olyan eszköznek, amivel a programozó megszüntetheti a címkomponenst.
4. **Érték:** Az eljárásorientált nyelvek leggyakoribb utasítása, az algoritmusok kódolásánál alapvető. Alakja: változó = kifejezés
    a. **Input:** A változó értékkomponensét egy perifériáról kapott adat határozza meg.
    b. **Kezdőértékadás:** Két fajtája van
        i. **Explicit kezdőérték:** A programozó explicit deklarációs utasításban a változó értékkomponensét is megadja. Amikor a változó címkomponenst kap, akkor egyben az értéket reprezentáló bitsorozat is beállítódik.
        ii. Van olyan hivatkozási nyelv, amely az **automatikus kezdőértékadás** elvét vallja.

Alapelemek

- Aritmetikai típusok
    - Integrális típusok
        - Egész
        - Karakter
        - Felsorolásos
    - Valós
- Származtatott típusok
    - Tömb
    - Függvény
    - Mutató
    - Struktúra
    - Union
- Void típus

3. Kifejezések

A kifejezések szintaktikai eszközök. Arra valók, hogy a program egy adott pontján ott **már ismert értékekből új értéket határozzunk meg**. **Két komponensük van, érték és típus**. Egy kifejezés formálisan a következőkből áll:

- **Operandusok**: Az operandus literál, nevesített konstans, változó vagy függvényhívás lehet. Az értéket képviseli.
- **Operátorok**: Műveleti jelek. Az értékekkel végrehajtandó műveleteket határozzák meg.
- **Kerek zárójelek:** A műveletek végrehajtási sorrendjét befolyásolják

Attól függően, hogy egy operátor hány operandussal végzi a műveletet, beszélünk

- **egyoperandusú (unáris):** ++, --, a++, a--
- **kétoperandusú (bináris):** a + b
- **háromoperandusú (ternáris):** a + b – c

A kifejezések három alakja lehet attól függően, hogy kétoperandusú operátorok esetén az operandusok és az operátor sorrendje milyen. A lehetséges esetek:

- **Prefix**: az operátor az operandusok előtt áll (* 3 5)
- **Infix**: az operátor az operandusok között áll (3 * 5)
- **Postfix**: az operátor az operandusok mögött áll (3 5 *)

Azt a folyamatot, amikor a kifejezés értéke és típusa meghatározódik, a **kifejezés kiértékelésének** nevezzük. A kiértékelés során adott sorrendben elvégezzük a műveleteket, előáll az érték, és hozzárendelődik a típus. A **műveletek végrehajtási sorrendje lehet**:

- A műveletek felírási sorrendje, azaz balról-jobbra
- A felírási sorrenddel ellentétesen, azaz jobbról-balra.
- Balról-jobbra a precedencia táblázat figyelembevételével.

A műveletek elvégzése előtt meg kell határozni az operandusok értékét.

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right →
2	a++ a--	Suffix/postfix increment and decrement	
	type() type{}	Functional cast	
	a()	Function call	
	a[]	Subscript	
	. ->	Member access	
3	++a --a	Prefix increment and decrement	Right-to-left ←
	+a -a	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	C-style cast	
	*a	Indirection (dereference)	
	&a	Address of	
	sizeof	Size-of[note 1]	
	co_await	await-expression (C++20)	
	new new[]	Dynamic memory allocation	
	delete delete[]	Dynamic memory deallocation	
4	.* ->*	Pointer-to-member	Left-to-right →
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	<=>	Three-way comparison operator (since C++20)	
9	< <= > >=	For relational operators < and ≤ and > and ≥ respectively	
10	== !=	For equality operators = and ≠ respectively	
11	a&b	Bitwise AND	
12	^	Bitwise XOR (exclusive or)	
13	|	Bitwise OR (inclusive or)	
14	&&	Logical AND	
15	||	Logical OR	
16	a?b:c	Ternary conditional[note 2]	Right-to-left ←
	throw	throw operator	
	co_yield	yield-expression (C++20)	
	=	Direct assignment (provided by default for C++ classes)	
	+= -=	Compound assignment by sum and difference	
	*= /= %=	Compound assignment by product, quotient, and remainder	
	<<= >>=	Compound assignment by bitwise left shift and right shift	
	&= ^= |=	Compound assignment by bitwise AND, XOR, and OR	
17	,	Comma	Left-to-right →

Kifejezés típusának meghatározása

A típusegyenértékűséget valló nyelvek azt mondják, hogy egy kifejezésben egy kétoperandusú operátornak csak azonos típusú operandusai lehetnek. Ilyenkor nincs konverzió, az eredmény típusa vagy a két operandus közös típusa, vagy azt az operátor dönti el (például hasonlító műveletek esetén az eredmény logikai típusú lesz).

A típuskényszerítés elvét valló nyelvek esetén különböző típusú operandusai lehetnek egy kétoperandusú operátornak. A műveletek viszont csak az azonos belső ábrázolású operandusok között végezhetők el, tehát különböző típusú operandusok estén konverzió van. Ilyen esetben a nyelv definiálja, hogy egy adott operátor esetén egyrész milyen típuskombinációk megengedettek, másrészt, hogy mi lesz a művelet eredményének típusa.

Egyes nyelvek a numerikus típusoknál megengedik a típuskényszerítés egy speciális fajtáját még akkor is, ha egyébként a típusegyenértékűséget vallják. Ezeknél a nyelveknél beszélünk bővítés és szűkítésről.

- **Bővítés**: Olyan típuskényszerítés, amikor a konvertálandó típus tartományának minden eleme egyben eleme a céltípus tartományának is. (pl. egész -> valós). Ekkor a konverzió minden további nélkül, értékvesztés nélkül végrehajtandó. A bővítés automatikusan történik.

- **Szűkítés**: A programozónak kell explicit módon előírnia, különben fordítási hiba lép fel.

4. Utasítások

A **deklarációs utasítások** mögött nem áll tárgykód. Ezen utasítások teljes mértékben a fordítóprogramoknak szólnak, attól kérnek valamilyen szolgáltatást, üzemmódot állítanak be, illetve olyan információkat szolgáltatnak, melyeket a fordítóprogram felhasznál a tárgykód generálásánál. Alapvetően befolyásolják a tárgykódot, de maguk nem kerülnek lefordításra. A programozó a névvel rendelkező saját programozási eszközeit tudja deklarálni.

A **végrehajtandó utasításokból** generálja a fordítóprogram a tárgykódot. Ezeket az alábbiak szerint csoportosítjuk:

Értékadó utasítás

Feladata beállítani vagy módosítani egy (esetleg több) változó értékkomponensét a program futásának bármely pillanatában.

Üres utasítás

Jelentősége általánosságban abban áll, hogy segítségével egyértelmű programszerkezet alakítható ki. Az üres utasítás hatására a processzor egy üres gépi utasítást hajt végre.

Ugró utasítás

Az ugró utasítás segítségével a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra adhatjuk át a vezérlést. GOTO címke

Elágaztató utasítások

Kétirányú elágaztató utasítás

A kétirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján két tevékenység közül választunk, illetve egy adott tevékenységet végrehajtsunk vagy sem.

```
IF feltétel
THEN tevékenység
 [ELSE tevékenység]
```

A feltétel egy logikai kifejezés

Többirányú elágaztató utasítás

A többirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró akárhány tevékenység közül egyet végrehajtsunk.

```
SWITCH (kifejezés) {
        CASE egész_konstans_kifejezés : [ tevékenység ]
        [ CASE egész_konstans_kifejezés : [tevékenység ]]…
        [ DEFAULT: tevékenység ]};
```

Ciklusszervező utasítások

A ciklusszervező utasítások lehetővé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételjünk. Egy ciklus általános felépítése a következő: fej – mag – vég. Az ismétlésre vonatkozó információk vagy a fejben vagy a végben szerepelnek. A mag az ismétlendő végrehajtandó utasításokat tartalmazza.

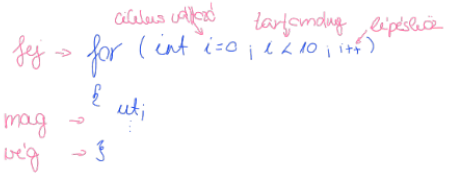
A ciklusok működésénél megkülönböztetünk két szélsőséges esetet. Az egyik az, amikor a mag egyszer sem fut le, ezt hívjuk **üres** ciklusnak. A másik az, amikor az ismétlődés soha nem áll le, ez a **végtelen** ciklus. A működés szerinti végtelen ciklus a programban nyilván szemantikai hibát jelent, hiszen az sohasem fejeződik be.

Feltételes ciklus

Ennél a ciklusnál az ismétlődést egy feltétel igaz vagy hamis értéke szabályozza. A feltétel maga vagy a fejben vagy a végben szerepel. Szemantikájuk alapján beszélünk **kezdőfeltételes** és **végfeltételes** ciklusról.

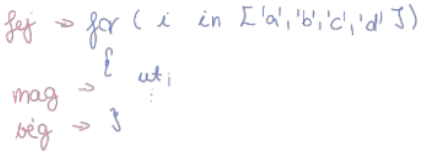
Kezdőfeltételes ciklus		**Végfeltételes ciklus**	
fej ->	while (feltétel)	fej ->	do
	{		{
mag ->	utasítás;	mag ->	utasítás;
vég ->	}	vég ->	} while (feltétel);
Futási szám: 0 … végtelen		Futási szám: 1 … végtelen	

Előírt lépésszámú ciklus



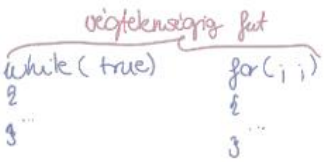
Ennél a ciklusfajtánál az ismétlődésre vonatkozó információk (az ún. ciklusparaméterek) a fejben vannak. Minden esetben tartozik hozzá egy változó, a **ciklusváltozó**. A változó által felvett értékekre fut le a ciklusmag. A változó az értékeit egy **tartományból** veheti föl. Ezt a tartományt a fejben adjuk meg kezdő- és végértékével. A ciklusváltozó a tartománynak vagy minden elemét fölveheti (beleértve a kezdő- és végértéket is), vagy csak a tartományban szabályosan (ekvidisztánsan) elhelyezkedő bizonyos értékeket. Ekkor meg kell adni a tartományban a felvehető elemek távolságát meghatározó **lépésközt**. A változó az adott tartományt befuthatja növekvőleg, illetve csökkenőleg, ezt határozza meg az irány.

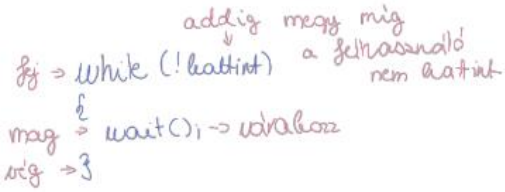
Felsorolásos ciklus



A felsorolásos ciklus az előírt lépésszámú ciklus egyfajta általánosításának tekinthető. Van ciklusváltozója, amely explicit módon megadott értékeket vesz fel és minden felvett érték mellett lefut a mag. A ciklusváltozót és az értékeket a fejben adjuk meg, ez utóbbiakat kifejezéssel. A ciklusváltozó típusa általában tetszőleges. Nem lehet sem üres, sem végtelen ciklus.

Végtelen ciklus





A végtelen ciklus az a ciklusfajta, ahol sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan. Működését tekintve definíció szerint végtelen ciklus, üres ciklus nem lehet. Használatánál a magban kell olyan utasítást alkalmazni, amelyik befejezteti a ciklust. Nagyon hatékony lehet **eseményvezérelt** alkalmazások implementálásánál.

Összetett ciklus

Az előző négy ciklusfajta kombinációiból áll össze. A ciklusfejben tetszőlegesen sok ismétlődésre vonatkozó információ sorolható föl, szemantikájuk pedig szuperponálódik. Nagyon bonyolult működésű ciklusok építhetők fel a segítségével.

Hívó utasítás

Vezérlésátadó utasítások

A C-ben három vezérlő utasítás van még az eddigiekben tárgyalt végrehajtható utasításokon kívül:

Continue

Ciklus magjában alkalmazható. A ciklus magjának hátralévő utasításait nem hajtja végre, hanem az ismétlődés feltételeit vizsgálja meg és vagy újabb ciklislépésbe kezd, vagy befejezi a ciklust.

Break

Ciklus magjában, vagy többszörös elágaztató utasítás CASE-ágában helyezhető el. A ciklust szabályosan befejezteti, illetőleg kilép a többszörös elágaztató utasításból.

Return [kifejezés]

Szabályosan befejezteti a függvényt és visszaadja a vezérlést a hívónak.

I/O utasítások

Egyéb utasítások

5. Programok szerkezete

Az eljárás orientált nyelvekben az alábbi programegységek léteznek: alprogram, blokk, csomag, taszk

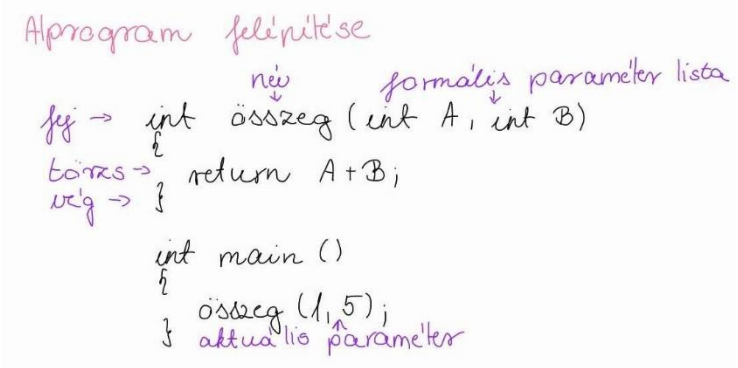
Alprogramok

Az alprogram az eljárásorientált nyelvekben a **procedurális absztrakció** első megjelenési formája, alapvető szerepet játszik ebben a paradigmában, sőt meghatározója annak. Az alprogram mint

absztrakciós eszköz egy bemeneti adatcsoportot képez le egy kimeneti adatcsoportra úgy, hogy egy specifikáció megadja az adatok leírását, de **semmit sem tudunk a tényleges leképzésről**. **Ismerjük a specifikációt, de nem ismerjük az implementációt.**

Az alprogram, mint programozási eszköz az **újrafelhasználás eszköze.** Akkor alkalmazható, ha a program különböző pontjain ugyanaz a programrész **ismétlődik.** Ez az ismétlődő programrész kiemelhető, egyszer kell megírni, és a program azon pontjain, ahol ez a programrész szerepelt volna, csak **hivatkozni kell rá** – az alprogram az adott helyeken **meghívható, aktivizálható.**

Az alprogram attól lesz abszrakciós eszköz, hogy a kiemelt programrészt **formális paraméterekkel** látjuk el, vagyis **általánosabban** írjuk meg, mint ahogyan az adott helyeken szerepelt volna.



Alprogram felépítése

- Formálisan: fej (specifikáció) – törzs (implementáció) – vég
- Mint, programozási eszköz:
  - **Név**: Egy azonosító, amely mindig a fejben szerepel
  - **Formális paraméterlista**:
    - **Azonosítók szerepelnek**, ezek a törzsben saját programozási eszközök nevei lehetnek, és egy általános szerepkört írnak le, amelyet a hívás helyén konkretizálni kell az **aktuális paraméterek** segítségével.
    - **Lehet üres**, ekkor **paraméternélküli alprogramról** beszélünk.
  - **Törzs:** Deklarációs és végrehajtható utasítások szerepelnek benne.
    - A **deklarált** programozási eszközök az alprogram **lokális** eszközei, ezek nevei az alprogram **lokális** nevei.
    - Léteznek **globális** nevek, melyeket nem az alprogramban definiálunk hanem rajta kívül.
  - **Környezet:** A globális változók együttese.

Alprogram fajtái

Eljárás

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

Függvény

A függvény olyan alprogram, amelynek az a **feladata**, hogy **egyetlen értéket határozzon meg**. Ez az érték **általában tetszőleges típusú** lehet. A függvény **visszatérési értékének** a típusa egy további olyan információ, amely hozzátartozik a függvény specifikációjához. **A függvény visszatérési értékét mindig a neve hordozza**, formálisan az közvetíti vissza a hívás helyére. A függvény törzsének végrehajtható utasításai a visszatérési érték meghatározását szolgálják.

Azt a szituációt, amikor a függvény megváltoztatja paramétereit, vagy környezetét, a függvény mellékhatásának nevezzük. A mellékhatást általában károsnak tartják.

Hívási lánc, rekurzió

Egy programegység bármikor meghívhat egy másik programegységet, az egy újabb programegységet, és így tovább. Így kialakul egy hívási lánc. A hívási lánc **első tagja mindig a főprogram**. A hívási lánc minden tagja **aktív**, de csak a legutoljára meghívott programegység működik. Szabályos esetben mindig az utoljára meghívott programegység fejezi be legelőször a működését, és a vezérlés visszatér az őt meghívó programegységbe. A hívási lánc futás közben dinamikusan épül föl és bomlik le.

Azt a szituációt, amikor egy aktív alprogramot hívunk meg, rekurziónak nevezzük.

- **közvetlen**: egy alprogram önmagát hívja meg, vagyis a törzsben van egy hivatkozás saját magára.
- **közvetett**: a hívási láncban már korábban szereplő alprogramot hívunk meg.

A rekurzióval megvalósított algoritmus mindig átírható iteratív algoritmussá.

Paraméterkiértékelés

Paraméterkiértékelés alatt értjük azt a folyamatot, **amikor egy alprogram hívásánál egymáshoz rendelődnek a formális- és aktuális paraméterek, és meghatározódnak azok az információk, amelyek a paraméterátadásnál a kommunikációt szolgáltatják.**

Paraméterátadás

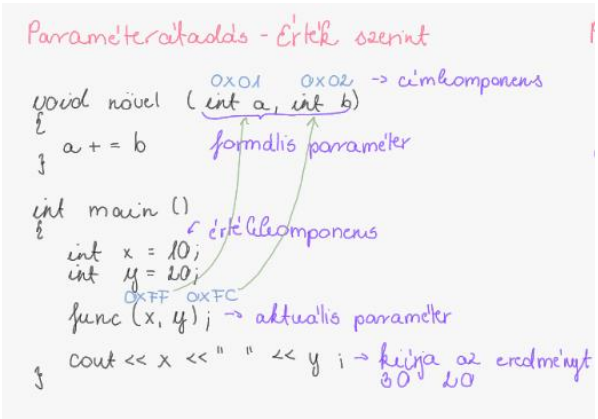
A paraméterátadás az alprogramok és más programegységek közötti kommunikáció egy formája. A paraméterátadásnál mindig van egy **hívó**, ez tetszőleges programegység, és egy **hívott**, amelyik mindig alprogram.

Paraméterátadási módok

	Érték	Cím	Eredmény	Érték-eredmény
Formális paraméternek a hívott alprogram területén címkoponense	Van	Nincs	Van	Van
Az aktuális paratméternek rendelekeznie kell címkomponenssel	Igen	Igen	Igen	Van
A formális paraméternek van kezdőértéke	Igen	Igen	Nincs	Igen
Információáramlás	Egyirányú	Kétirányú	Egyirányú	Kétirányú
Értékmásolás	Mindig van	Nincs	Van	Kétszer van
Sebesség	Lassú	Gyors	-	-
Aktuális paraméter	Kifejezés	Változó	Változó	Változó

Érték szerinti

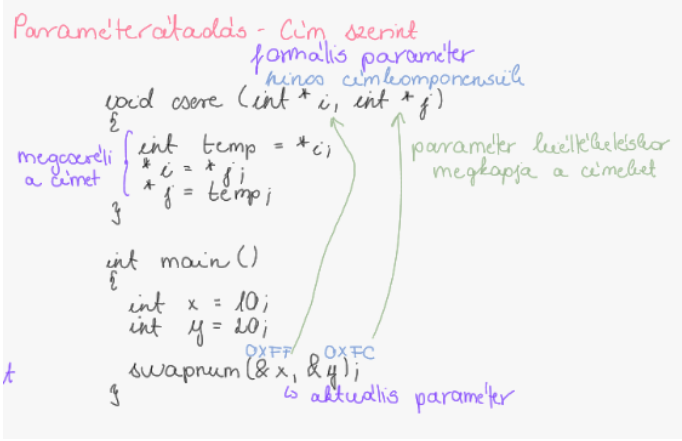
Az értékszerinti paraméterátadás esetén a **formális paramétereknek van címkomponensük** a hívott alprogram területén. Az aktuális paraméternek **rendelkeznie kell értékkomponenssel a hívó oldalon**. Ez **az érték meghatározódik a paraméterkiértékelés folyamán**, majd átkerül a hívott alprogram területén lefoglalt címkomponensre. A **formális paraméter kap egy kezdőértéket, és az alprogram ezzel az értékkel dolgozik a saját területén.** Az **információáramlás egyirányú**, a hívótól a hívott felé irányul. A hívott alprogram semmit sem tud a hívóról, a saját területén dolgozik. **Mindig van egy értékmásolás**, és ez az érték tetszőleges bonyolultságú lehet. **Ha egy egész adatcsoportot kell átmásolni, az hosszadalmas. Lényeges, hogy a két programegység egymástól függetlenül működik**, és egymás működését az érték meghatározáson túl nem befolyásolják.



Az aktuális paraméter kifejezés lehet.

Cím szerinti

A cím szerinti paraméterátadásnál a **formális paramétereknek nincs címkomponensük** a hívott alprogram területén. Az aktuális paraméternek viszont **rendelkeznie kell címkomponenssel a hívó területén**. **Paraméterkiértékeléskor** meghatározódik az aktuális paraméter címe és átadódik a hívott alprogramnak, ez lesz a **formális paraméter címkomponense**. Tehát a meghívott alprogram a hívó területén dolgozik. **Az**

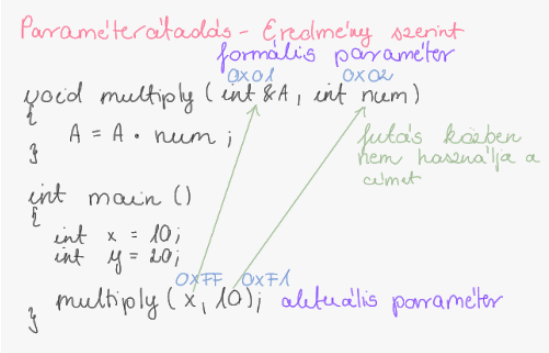


információátadás kétirányú, az alprogram a hívó területéről átvehet értéket, és írhat is oda. Az alprogram átnyúl a hívó területre. **Időben gyors, mert nincs értékmásolás**, de veszélyes lehet, mivel a hívott alprogram hozzáfér a hívó területén lévő információkhoz, és szabálytalanul használhatja föl azokat.

Az aktuális paraméter változó lehet.

Eredmény szerinti

Az eredmény szerinti paraméterátadásnál a **formális paraméternek van címkomponense** a hívott alprogram területén, **az aktuális paraméternek pedig lennie kell címkomponensének**. **Paraméterkiértékeléskor meghatározódik az aktuális paraméter címe** és átadódik a hívott alprogramnak, azonban az alprogram a saját területén dolgozik, és **a futás közben nem használja ezt a címet**. A működésének befejeztekor viszont átmásolja a formális paraméter értékét erre a címkomponensre. **A kommunikáció egyirányú**, a hívottól a hívó felé irányul. **Van értékmásolás**.



Az aktuális paraméter változó lehet.

Érték-eredmény szerinti

Az érték-eredmény szerinti paraméterátadásnál a **formális paraméternek van címkomponense** a hívott területén és az **aktuális paraméternek rendelkeznie kell érték- és címkomponenssel**. A paraméterkiértékelésnél meghatározódik az aktuális paraméter értéke és címe és mindkettő átkerül a hívotthoz. Az **alprogram a kapott értékkel, mint kezdőértékkel kezd el dolgozni a saját területén** és a címet nem használja. Miután viszont befejeződik, a formális paraméter értéke **átmásolódik** az aktuális paraméter címére. A **kommunikáció kétirányú, kétszer van értékmásolás.**

Az aktuális paraméter változó lehet.

Név szerinti

Név szerinti paraméterátadásnál **az aktuális paraméter egy, az adott szövegkörnyezetben értelmezhető tetszőleges szimbólumsorozat** lehet. A param**éterkiértékelésnél rögzítődik az alprogram szövegkörnyezete**, itt értelmezésre kerül az aktuális paraméter, majd a szimbólumsorozat a formális paraméter nevének minden előfordulását felülírja az alprogram szövegében és ezután fut le az. **Az információáramlás iránya az aktuális paraméter adott szövegkörnyezetbeli értelmezésétől függ.**

Szöveg szerinti

A szöveg szerinti paraméterátadás **a név szerintinek egy változata**, annyiban különbözik tőle, hogy a hívás után az alprogram elkezd működni, az aktuális paraméter értelmező szövegkörnyezetének rögzítése és a formális paraméter felülírása csak akkor következik be, amikor a formális paraméter neve először fordul elő az alprogram szövegében a végrehajtás folyamán.

Alprogramok esetén típust paraméterként átadni nem lehet!!! Egy adott esetben a paraméterátadás módját az alábbiak döntik el:

- a nyelv csak egyetlen paraméterátadási módot ismer (pl. C)
- a formális paraméter listában explicit módon meg kell adni a paraméterátadási módot (pl. Ada)
- az aktuális és formális paraméter típusa együttesen dönti el (pl. PL/I)
- a formális paraméter típusa dönti el (pl. FORTRAN)

Az alprogramok formális paramétereit három csoportra oszthatjuk:

- **Input** paraméterek: ezek segítségével az alprogram kap információt a hívótól (pl. érték szerinti paraméterátadás).
- **Output** paraméterek: a hívott alprogram ad át információt a hívónak (pl. eredmény szerinti paraméterátadás).
- **Input-output** paraméterek: az információ mindkét irányba mozog (pl. érték-eredmény szerinti paraméterátadás).

A blokk

A blokk olyan programegység, **amely csak másik programegység belsejében helyezkedhet el**, külső szinten nem állhat.

Formálisan a blokknak van kezdete, törzse és vége. A kezdetet és a véget egy-egy speciális karaktersorozat vagy alapszó jelzi. A törzsben lehetnek deklarációs és végrehajtható utasítások. Ugyanúgy mint az alprogramoknál, ezek az utasítások vagy tetszőlegesen keverhetők, vagy van külön deklarációs rész és végrehajtható rész. **A blokknak nincs paramétere**. A blokknak egyes nyelvekben lehet neve. Ez általában a kezdet előtt álló címke.

A blokk bárhol elhelyezhető, ahol végrehajtható utasítás állhat.

Hatáskör

A hatáskör a nevekhez kapcsolódó fogalom. Egy név hatásköre alatt értjük a program szövegének azon részét, **ahol az adott név ugyanazt a programozási eszközt hivatkozza**, tehát jelentése, felhasználási módja, jellemzői **azonosak**. A hatáskör szinonimája a láthatóság.

A név hatásköre az eljárásorientált programnyelvekben a programegységekhez illetve a fordítási egységekhez kapcsolódik.

Egy programegységben deklarált nevet a programegység lokális nevének nevezzük. Azt a nevet, **amelyet nem a programegységben deklaráltunk, de ott hivatkozunk rá, szabad névnek hívjuk.**

Azt a tevékenységet, mikor egy név hatáskörét megállapítjuk, **hatáskörkezelésnek** hívjuk. Kétféle hatáskörkezelést ismerünk, a **statikus és a dinamikus hatáskörkezelést**.

Statikus hatáskörkezelés	Dinamikus hatáskörkezelés
Fordítási időben történik, a fordítóprogram végzi. **Alapja a programszöveg programegység szerkezete.** **Egy lokális név hatásköre** az a programegység, amelyben deklaráltuk és minden olyan programegység tartalmaz, hacsak a tartalmazott programegységekben a nevet nem deklaráltuk. **A hatáskör befelé terjed, de kifelé soha.** Azt a nevet, amely egy adott programegységben nem lokális név, de onnan látható, **globális névnek nevezzük.** A programban szereplő összes név hatásköre a **forrásszöveg alapján egyértelműen megállapítható.**	**Futási idejű tevékenység**, a futtató rendszer végzi. **Alapja a hívási lánc.** Ha a futtató rendszer egy programegységben talál egy szabad nevet, akkor a hívási láncon keresztül kezd el visszalépkedni mindaddig, amíg meg nem találja lokális névként, vagy a hívási lánc elejére nem ér. (Ha odaér futási hiba vagy automatikus deklaráció következik be.) Dinamikus hatáskörkezelésnél egy név hatásköre az a programegység, amelyben deklaráltuk, és minden olyan programegység, amely ezen programegységből induló hívási láncban helyezkedik el, hacsak ott nem deklaráltuk újra a nevet. **A hatáskör futási időben változhat és más-más futásnál más-más lehet.**

Hatáskör és élettartam

A C a hatáskör és élettartam szabályozására bevezeti a tárolási osztály attribútumokat, melyek a következők:

- **extern:** A fordítási egység szintjén deklarált nevek alapértelmezett tárolási osztálya, lokális neveknél explicit módon meg kell adni. Az ilyen nevek hatásköre a teljes program, élettartamuk a program futási ideje. Van automatikus kezdőértékük.
- **auto:** A lokális nevek alapértelmezett tárolási osztálya. Hatáskörkezelésük statikus, de csak a deklarációtól kezdve láthatók. Élettartamuk dinamikus. Nincs automatikus kezdőértékük.
- **register:** Speciális *auto*, amelynek értéke regiszterben tárolódik, ha van szabad regiszter, egyébként nincs különbség.
- **static:** Bármely névnél explicit módon meg kell adni. Hatáskörük a fordítási egység, élettartamuk a program futási ideje. Van automatikus kezdőértékük.

Fordítási egység

Az eljárásorientált nyelvekben **a program közvetlenül fordítási egységekből épül föl**. Ezek olyan forrásszöveg-részek, melyek önállóan, a program többi részétől fizikailag különválasztva fordíthatók le. Az

egyes nyelvekben a **fordítási egységek felépítése igen eltérő lehet**. **A fordítási egységek általában hatásköri és gyakran élettartam definiáló egységek is.**

6. Absztrakt adattípus

Az absztrakt adattípus olyan adattípus, amely **megvalósítja a bezárást vagy információ rejtést**. Ez azt jelenti, hogy **ezen adattípusnál nem ismerjük a reprezentációt és a műveletek implementációját**. Az adattípus ezeket nem mutatja meg a külvilág számára. Az ilyen típusú programozási eszközök értékeihez csak **szabályozott módon, a műveleteinek specifikációi által meghatározott interfészen keresztül férhetünk hozzá.** Tehát az értékeket véletlenül vagy szándékosan nem ronthatjuk el. Ez nagyon lényeges a biztonságos programozás szempontjából. **Az absztrakt adattípus** (angol rövidítéssel: ADT – Abstract Data Type) az elmúlt évtizedekben a programnyelvek **egyik legfontosabb fogalmává vált** és alapvetően befolyásolta a nyelvek fejlődését.

7. Csomag

A csomag az a programegység, amely egyaránt szolgálja a procedurális és az absztrakciót. A procedurális absztrakció oldaláról tekintve a csomag programozási eszközök újrafelhasználható gyűjteménye. Ezek az eszközök:

- Típus
- Változó
- Nevesített konstans
- Saját kivétel
- Alprogram
- Csomag

Ezek az eszközök a csomag hatáskörén belül mindenhonnan tetszőlegesen hivatkozhatók. A csomag mint programegység megvalósítja a bezárást, ezért alkalmas absztrakt adattípus implementálására.

8. Kivételkezelés

A kivételkezelési eszközrendszer azt teszi lehetővé, hogy az operációs rendszertől **átvegyük a megszakítások kezelését, felhozzuk azt a program szintjére. A kivételek olyan események, amelyek megszakítást okoznak**. A kivételkezelés az a tevékenység, amelyet a program végez, ha egy kivétel következik be. Kivételkezelő alatt egy olyan programrészt fogunk érteni, amely működésbe lép egy adott kivétel bekövetkezte után, reagálva az eseményre.

A kivételkezelés az eseményvezérlés lehetőségét teszi lehetővé a programozásban.

Operációs rendszer szinten lehetőség van bizonyos **megszakítások maszkolására**. Ez a lehetőség megvan nyelvi szinten is. **Egyes kivételek figyelése letiltható vagy engedélyezhető**. Egy kivétel figyelésének letiltása a legegyszerűbb kivételkezelés. Ekkor az esemény hatására a megszakítás bekövetkezik, feljön programszintre, kiváltódik a kivétel, de a program nem vesz róla tudomást, fut tovább. Természetesen nem tudjuk, hogy ennek milyen hatása lesz a program további működésére, lehet, hogy az rosszul, vagy sehogy sem tudja folytatni munkáját.

A kivételeknek általában van neve (vagy egy kapcsolódó sztring, amely gyakran az eseményhez kapcsolódó üzenet szerepét játssza) és kódja (ami általában egy egész szám).

9. Generikus programozás

A generikus programozási paradigma az **újrafelhasználhatóság és így a procedurális absztrakció eszköze**. Ez a paradigma **ortogonális** az összes többi paradigmára, tehát **bármely programozási nyelvbe beépíthető ilyen eszközrendsze**r. A generikus programozás lényege, hogy egy **paraméterezhető forrásszöveg-mintát adunk meg**. Ezt a mintaszöveget **a fordító kezeli**. A mintaszövegből aktuális paraméterek segítségével előállítható egy konkrét szöveg, ami aztán lefordítható. Az újrafelhasználás ott érhető tetten, hogy egy mintaszövegből tetszőleges számú konkrét szöveg generálható. **És ami talán a leglényegesebb, hogy a mintaszöveg típussal is paraméterezhető.**

10. INPUT / OUTPUT

Az I/O az a területe a programnyelveknek, ahol azok leginkább eltérnek egymástól. Az I/O az az eszközrendszer a programnyelvekben, **amely a perifériákkal történő kommunikációért felelős**, amely az **operatív tárból oda küld adatokat, vagy onnan vár adatokat**. **Az I/O középpontjában az állomány áll**. A programnyelvi állományfogalom megfelel az absztrakt állományfogalomnak.

Egy programban a logikai állomány egy olyan programozási eszköz, **amelynek neve van** és amelynél **az absztrakt állományjellemzők (rekordfelépítés, rekordformátum, elérés, szerkezet, blokkolás, rekordazonosító, stb.) attribútumként jelennek meg**. A fizikai állomány pedig a szokásos operációs rendszer szintű, konkrét, a perifériákon megjelenő, az adatokat tartalmazó állomány.

Egy állomány funkció szerint lehet:

- **input állomány**: a feldolgozás előtt már léteznie kell, és a feldolgozás során változatlan marad, csak olvasni lehet belőle,
- **output állomány**: a feldolgozás előtt nem létezik, a feldolgozás hozza létre, csak írni lehet bele,
- **input-output állomány**: általában létezik a feldolgozás előtt és létezik a feldolgozás után is, de a tartalma megváltozik, olvasni és írni is lehet.

Az I/O során adatok mozognak a tár és a periféria között. A tárban is, és a periférián is van valamilyen ábrázolási mód. Kérdés, hogy az adatmozgatás közben történik-e konverzió. Ennek megfelelően létezik kétféle adatátviteli mód: **a folyamatos (van konverzió) és a bináris vagy rekord módú (nincs konverzió).**

A folyamatos módú adatátvitelnél **a tárban és a periférián eltér a reprezentáció**. Ebben az esetben a nyelvek a periférián **az adatokat egy folytonos karaktersorozatnak tekintik**, a tárban pedig a típusnak megfelelő belső ábrázolás által definiált bitsorozatok vannak. Az adatátvitel ekkor egyedi adatok átvitelét jelenti **konverzióval**. **Olvasáskor meg kell mondania,** hogy a folytonos karaktersorozatot hogyan **tördeljük fel olyan karaktercsoportokra**, amelyek az egyedi adatokat jelentik, és hogy az adott karaktercsoport milyen típusú adatot jelent. **Íráskor** pedig rendelkezni kell arról, hogy a tárbeli, adott típusú adatot reprezentáló bitsorozatból a folytonos karaktersorozatban **melyik helyen és hány karaktert alkotva jelenjen meg az egyedi adat.**

A nyelvekben ezek megadására három alapvető eszközrendszer alakult ki:

1. **Formátumos módú adatátvitel**: minden egyes egyedi adathoz a formátumok segítségével explicit módon meg kell adni a kezelendő karakterek darabszámát és a típust.
2. **Szerkesztett módú adatátvitel**: minden egyes egyedi adathoz **meg kell adni egy maszkot**, amely szerkesztő és átviendő karakterekből áll. **A maszk elemeinek száma határozza meg a kezelendő karakterek darabszámát, a szerkesztő karakterek megadják, hogy az adott pozíción milyen kategóriájú karakternek kell megjelennie, a többi karakter változtatás nélkül átvitelre kerül.**
3. **Listázott módú adatátvitel:** itt a folytonos karaktersorozatban magában vannak a tördelést végző speciális karakterek, amelyek az egyedi adatokat elhatárolják egymástól, a típusra nézve pedig **nincs explicit módon megadott információ**.

Ha egy programban állományokkal akarunk dolgozni, akkor a következőket kell végrehajtani:

1. **Deklaráció**: A logikai állományt mindig deklarálni kell az adott nyelv szabályainak megfelelően. El kell látni a megfelelő névvel és attribútumokkal.
2. **Összerendelés**: A logikai állománynak megfeleltetünk egy fizikai állományt. Innentől kezdve csak a logikai állománynévvel dolgozunk.
3. **Állomány megnyitása**: Csak akkor tudunk egy állománnyal dolgozni ha meg van nyitva. Egy program futása folyamán ugyanazt az állományt más-más funkcióra is megnyithatjuk.
4. **Feldolgozás**: Írhatunk, olvashatunk a fájlba.
5. **Lezárás**: A lezárás megszünteti a kapcsolatot a logikai állomány és a fizikai állomány között.

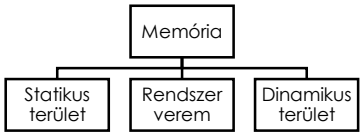
Implicit állomány: Az írás-olvasás közvetlenül valamelyik perifériával történik. Tehát nem kell deklarálni, összerendelni, megnyitni és lezárni.

C-ben

Az I/O eszközrendszer nem része a nyelvnek. Standard könyvtári függvények állnak rendelkezésre**. Létezik bináris és a folyamatos módú átvitel**, ez utóbbinál egy formátumos és egy szerkesztett átvitel keverékeként. **Szeriális szerkezetet kezel fix és változó rekordformátummal**. Az I/O függvények minimálisan egy karakter vagy karaktercsoport, illetve egy bájt vagy bájtcsoport írását és olvasását teszik lehetővé.

11. Implementációs kérdések

Az eljárás orientált programozási nyelvek a rendelkezésükre álló memóriát általában a következő területekre osztják fel futás közben:



- **Statikus terület**: ez tartalmazza a kódszegmenst és a futtató rendszer rutinjait.
- **Rendszer verem**: tárolja az aktiváló rekordokat
- **Dinamikus terület**: A mutató típusú eszközökkel kezelt dinamikus konstrukciók helyezkednek el benne.

Sok nyelvi implementáció úgy kezeli a memóriát, hogy a szabad tárterület a verem és a dinamikus terület között van, tehát ezek egymás rovására növekszenek.

Az eljárásorientált programozási nyelvek a programegységek futásidejű kezeléséhez, a hívási lánc implementálásához az ún. aktiváló rekordot használják. Ennek a felépítése a következő:

- **Dinamikus kapcsoló**: Ez egy mutató típusú mező, amely a hívó programegység aktiváló rekordját címzi. A hívási környezet érhető el vele és a programegység szabályos befejeződésekor az aktiváló rekord törlésénél van alapvető szerepe.
- **Statikus kapcsoló**: Ez egy mutató típusú mező, amely a tartalmazó programegység aktiváló rekordját címzi. Statikus hatáskörkezelésnél ennek segítségével érhető el a tartalmazó környezet.
- **Visszatérési cím:** A kódszegmens azon címe, ahol a programegység szabályos befejezése esetén a programot folytatni kell.
- **Lokális változók**
- **Formális paraméterek** (csak alprogram esetén)
- **Visszatérési érték** (csak függvény esetén)

Az egyszerű típusú lokális változók számára a típushoz megfelelően foglalódik le a tárterület. Az összetett típusúaknál már bonyolultabb a helyzet.

- **Tömb:** S lefoglalt tárterület elején egy ún. tömbleíró helyezkedik el, amely tartamazza dimenziónként az indexek alsó és felső határát, az elemek típusát és az egy elem tárolásához szükséges bájtok számát. Ezután pedig jönnek az elemek sor- vagy oszlopfolytonosan.
- **Rekord:** A mezők típusa dönt, ezek egymásután helyezkednek el. Változó hosszúságú rekordtípusnál a helyfoglalás a maximális méretre történik.
- **Sztring:** Egyaránt szóba jöhet a fix és a változó hosszon való tárolás, az utóbbinál hosszmegadással, illetve végjellel.
- **Halmaz:** Kevés elemszámnál karakterisztikus függvény, nagy elemszám esetén kulcstranszformációs táblázat alkalmazása a szokásos.

A formális paraméterek számára lefoglalt tárterület a paraméterátadástól függ.

- **Érték szerinti** esetben a formális paraméter típusának megfelelő tárterület szükséges.
- **Cím és eredmény szerinti** esetben egy cím tárolásához szükséges bájtmennyiség foglalódik le.
- **Érték-eredmény** szerintinél pedig az előző kettő.
- **A név és szöveg szerinti paraméterátadás esetén** ide egy paraméter nélküli rendszer rutin hívása kerül. Ez mindig lefut, amikor a formális paraméterre hivatkozás történik. Feladata a szövegkörnyezet meghatározása és abban az aktuális paraméter kiértékelése, aztán a formális paraméterek nevének felülírása.

Aktiváló rekord: Veremben tárolódnak. A verem alján mindig a főprogram aktiváló rekordja van. Befejezéskor az aktiváló rekord törlődik.

12. Objektumorientáltság

Az objektum-orientált programozási módszertan filozófiáját – **amelynek alapgondolata, hogy az adat és a funkcionális modell egymástól elválaszthatatlan, lényegében egyetlen modell** – követik az objektum-orientált programozási nyelvek. Az OO programozási nyelvek **imperatív jellegűek**: algoritmikus szemléletet tükröznek. Az OO paradigma bevonul minden más nyelvi osztályba is. **Jellemzői**:

- Az adatmodell és az eljárásmodell elválaszthatatlan (így szemléli a világot).
- Absztrakt eszköz és fogalomrendszer: Az újrafelhasználhatóságot olyan magas szintre elviszi, ameddig lehetséges, a valós világot nagyon megközelíti.
- Szemlélete: imperatív (algoritmus – kódolni kell) eszközrendszer

Fogalomrendszer

Objektum

Az eljárásorientált nyelvek változó fogalmának kiterjesztése (általánosítása), olyan konkrét programozási eszköz melynek vannak:

- **Attribútumai (attribute)**: ez az , a struktúra, tetszőleges bonyolultságú adatszerkezet. Szokás ezt az objektum statikus részének is nevezni. Minden objektum mögött van egy jól definiált tárterület, ezen vannak az attribútumok értékeit reprezentáló bitsorozatok.
- **Terminológia**: az obektumok állapotairól (state) beszélünk, ahol minden egyes állapotot egy-egy bitkombináció ír le, ami egy jóldefiniált címen van.
- **Módszerei (method)**: a viselkedés leírására szolgál (eljárásmodell leírására) az eljárásorientált nyelvek eljárásai és függvényei. A módszerek adják meg nyelvi szinten az objektum viselkedésmódját (behavior).
- **Azonossággal rendelkezik (van azonosság tudata):** bármely objektum csak és kizárólag önmagával azonos, minden mástól megkülönböztetett. Minden objektumnak van azonosítója (OID: object identifier). Nyelvi szinten ezzel nem foglakozunk.
- **Analógia**: változó – név objektum – OID (nem egy név!) A változó neve igazából soha nem azonosító csak hatáskörön belül egyértelmű a névhivatkozás. Az OID viszont tényleg egyedi, még programok között is!

Objektum viselkedése: Az objektum állapota időben módosul(hat).

Módszerek csoportjai:

- le tudja kérdezni az objektum állapotát
- meg tudja változtatni az objektum állapotát

Objektumok élettartama: Az objektumot létre kell hozni, és addig él, amíg meg nem szűnik. A megszüntetés lehet a nyelvi rendszer feladata, vagy a programozóé. Az objektumazonosító minden szinten él, mindig léteznie kell.

Absztrakt osztályok

Absztrakt osztályoknak hívjuk azokat az osztályokat, amelyeknek nincsenek példányaik, amelyek nem példányosíthatók. Csak örököltetésre való. Vannak továbbá absztrakt módszerek. Ezek azok a módszerek, amelyeknek csak a specifikációjuk van megadva implementáció nélkül. Az abszrakt osztályokból konkrét, példányosítható osztályok származtathatók. Az egész eszközrendszer az absztrakciót szolgálja. A rendszerfejlesztési ciklusban és a programfejlesztésnél lesz érdekes.

Konstruktor

- Az osztály nevével megegyező nevű függvény, amely a példányosítás során hívódik meg. Célja az objektum inicializálása, adatokkal feltöltése.
- Default construktor – Üres konstruktor – Paraméter nélküli
- Parametrized construktor – Paraméterezett
- Nincs visszatérési érték!
- Másoló konstruktor: Animal a; Animal b(a);

Destruktor

- Az osztály nevével megegyező nevű függvény, amely előtt ~ karakter áll.
- Az objektum törlésekor fog lefutni.

This pointer

- A this kulcsszó az osztály aktuális példányára fog hivatkozni.
- Használati esetei:
    - Az aktuális objektumot paraméterül adjuk másik metódusnak.
    - Hivatkozhatunk az aktuális példány változóra.

Static kulcsszó

- Azt jelzi, hogy a függvény/adattag az osztályhoz tartozik, nem a példányhoz. Nem szükséges az osztályból példányosítani, hogy elérjük.
- C++-ban static lehet mező, metódus, konstruktor, osztály, tulajdonságok, operátorok és események.