Magas szintű programozási nyelvek 1

1.1. Modellezés

Objektumok, egyedek – **tulajdonság**aik és **viselkedésmód**juk van. Az egyedek osztályozhatóak, kategorizálhatóak.
A valós világ túl bonyolult, ezért az emberi gondolkodás az **absztrakció**n alapul, ezért **modell**ekben gondolkodunk. Az absztrakció lényege a közös és lényeges dolgok kiemelése.

A modell három követelménye:

- Leképezés követelménye: Léteznie kell olyan egyednek, amelyet modellezzük (lehet megtervezett is).
- Leszűkítés követelménye: Az egyednek nem minden tulajdonsága jelenik meg a modellben (a modell szegényebb).
- Alkalmazhatóság követelménye: A modellnek használhatónak kell lennie oda-vissza (ezért modellezünk).

Számítógépen az egyedek tulajdonságait az **adat**, viselkedésmódjaikat a **program**ok szemléltetik. Ezt **adatmodell**nek nevezzük.

1.2. Alapfogalmak

A programozási nyelvek három szintjét különböztetjük meg:
- Gépi nyelv
- Assembly szintű nyelv
- Magas szintű nyelv

A magas szintű nyelven megírt programot **forrásszöveg**nek nevezzük, a rá vonatkozó formai követelményeket **a szintaktikai szabályok**, a tartalmi szabályokat a **szemantikai szabályok** definiálják.

Minden processzor saját gépi nyelvvel rendelkezik, így csak az adott gépi nyelven írt programokat tudja végrehajtani. Tehát a magas szintű nyelven írt forrásszövegeket gépi nyelvű programmá kell alakítani. Ennek két módja van: **fordítóprogram**os és i**nterpreter**es.
A fordítóprogram alkalmas gépi kódú tárgyprogramot előállítani magas szintű nyelven írt programból. Az előállításkor a teljes forrásprogramot egyként kezeli és működés közben az alábbi lépéseket hajtja végre:
- Lexikális elemzés (lexikai egységekre bontás)
- Szintaktikai elemzés
- Szemantikai elemzés (ha szintaktikailag helyes)
- Kódgenerálás

Tárgyprogramot csak a szintaktikailag helyes forrásprogramból lehet előállítani. Az előállított program gépi nyelvű, de még nem futtatható. Belőle futtatható programot a **kapcsolatszerkesztő** állít elő. Majd a futtatható programot a **betöltő** helyezi a tárba és adja át neki a vezérlést. A futó program vezérlését a **futtató rendszer** felügyeli.

A magas szintű programozási nyelvek között létezik olyan, amelyben olyan forrásprogramot lehet, amely tartalmaz nem nyelvi elemeket is. Ilyenkor egy **előfordító** segítségével először a forrásprogramból egy adott nyelvű forrásprogramot állítunk elő, majd ezt feldolgozzuk a nyelv fordítójával. Az **interpreter**es technikai esetén nem készül tárgyprogram, utasításonként sorra veszi a forrásprogramot, értelmezi, s végrehajtja.

Minden nyelvnek megvan a saját szabványa, amit **hivatkozási nyelv**nek nevezünk, ebben vannak definiálva a szintaktikai és szemantikai szabályok. A hivatkozási nyelvek mellett léteznek még **implementáció**k. Ezek egy adott platformon realizált fordítóprogramot vagy interpreterek.

Az implementációk nagy problémája mai napig is a **hordozhatóság**ban rejlenek. Gyakorlatilag lehetetlen megoldani, hogy az egyik implementációban működő program a másik implementációban is ugyan azt az eredményt adja.

Napjainkban a programok írásához grafikus **integrált fejlesztői környezet**ek állnak rendelkezésünkre. Ezek tartalmaznak szövegszerkesztőt, fordítót, kapcsolatszerkesztőt, betöltőt, futtató rendszert és belövőt is.

1.3. A programnyelvek osztályozása

Imperatív nyelvek (eljárásorientált nyelvek, objektumorientált nyelvek)
- Algoritmikusak, a programozó algoritmus kódol és az működteti a processzort.
- A program utasítások sorozata.
- Legfőbb programozói eszköz: változó, amely a tár közvetlen elérését biztosítja.
- Szoros kötődés a Neumann-architektúrához.

Deklaratív nyelvek (funkcionális nyelvek, logikai nyelvek)
- Nem algoritmikusak, a programozó csak a problémát adja meg, a nyelvbe be van építve a megoldás módszere.
- A memóriaműveleti lehetőségek nagyon korlátozottak.
- Nem kötődnek szorosan a Neumann-architektúrához.

Máselvű nyelvek
- Olyan nyelvek, amelyek máshová nem sorolhatóak.

2.1. Karakterkészlet

A forrásszöveg legkisebb része a **karakter**. Egy nyelvben megjelenhető karaktereket **karakterkészlet** tartalmazza. A karakterekből állíthatóak elő a **bonyolultabb nyelvi elemek**, amelyek a következőek lehetnek:
- Lexikális egységek
- Szintaktikai egységek
- Utasítások
- Programegységek
- Fordítási egységek
- Program

Minden nyelvnek saját karakterkészlete van, általában a következő kategóriák alapján csoportosítva a karaktereket: betűk, számok, egyéb karakterek. Minden nyelvben **betű** az angol ABC 26 nagybetűje. Egyes nyelvekben (FORTRAN) a kisbetűk nem betűk. A Pascal szerint a kis- és nagybetű azonos, a C szerint nem. A legtöbb nyelv a nemzeti betűket nem tartja betűnek. A **szám**okat minden nyelv egységesen kezeli, a decimális számok mindenhol számok. **Egyéb karakter**ek a műveleti jelek, elhatároló jelek, írásjelek és a speciális karakterek (pl.: ~). A **szóköz** kitűntetett szerepű.

2.2. Lexikális egységek

- Többkarakteres szimbólum (általában operátorok, pl.: ++).
- Szimbolikus név (**azonosító**: betűvel kezdődik, számmal vagy betűvel folytatódhat, **kulcsszó**: a nyelv tulajdonít neki jelentést – IF, FOR, CASE, **standard azonosító**: a nyelv tulajdonít neki jelentést, de ez a jelentés megváltoztatható - NULL).

- Címke (a végrehajtható utasítások megjelölésére szolgál, hogy a program egy másik pontjából hivatkozni tudjunk rá, COBOL: nincs, Pascal: max. 4 jegyű e.n. e.sz., FORTRAN: max. 5 jegyű e.n. e.sz., PL/I,C,Ada: azonosító).
- Megjegyzés (olyan rész helyezhető el segítségével, amelyet a fordító program a lexikális elemzés során amúgy is töröl, így nem befolyásolja a programot. Három féle megjegyzés létezik: **egész sor**os megjegyzés, **sor végi** megjegyzés, **szóközök közötti** megjegyzés).
- Literál (fix értékkel rendelkező programozási eszköz, 2 komponense van: **típus**, **érték**).

Fortran: Egész, valós, tizedestört valós, exponenciális valós, komplex, hexadecimális, logikai, szöveges, sztring literál.
Cobol: Numerikus, alfanumerikus literál.
PL/I: Decimális fixpontos, decimális lebegőpontos, bináris fixpontos, bináris lebegőpontos, imaginárius, karakterlánc, bitlánc literál.
Pascal: Egész, tizedestört, exponenciális, literál.
C: Rövid egész, oktális egész, hexadecimális egész, előjel nélküli egész, hosszú egész, hosszú valós, rövid valós, karakter, sztring literál.
Ada: Egész, valós, bázisolt, karakter, szting literál.

2.3. A forrásszöveg összehasonlításának általános szabályai

Kötött formátumú nyelvekben alapvető szerepet játszik a sor, ugyanis azt vallják, hogy egy sor egy utasításnak felel meg. Ha egy utasítás nem fér el egy sorban, azt külön jelölni kell. Több utasítás egy sorban soha nem állhat.
Szabad formátumú nyelvekben a sor és az utasítás között nincs kapcsolat, egy sorban akárhány utasítás lehet, és egy utasítás is állhat több sorból. Mivel nem a sor vége jelöl az utasítás végét, így egy speciális karakter jelöli az utasítás végjelét, ez általában egy pontosvessző.

Az eljárásorientált nyelvekben a lexikális egységeket valamilyen **elhatároló jel**lel (vessző), vagy **szóköz**zel kell elválasztani egymástól. Az elhatároló jelek tetszőlegesen ismételhetőek.

2.4. Adattípusok

Az adattípus egy absztrakt programozási eszköz, amelynek neve van: ami egy azonosító. Az adattípus ismeretének függvényében léteznek **típusos nyelv**ek és **nem típusos nyelv**eknek. Az eljárásorientált nyelvek mind típusosak. Az adattípust három dolog határra meg:

- Tartomány (az adattípus által felvehető elemeket tartalmazza).
- Műveletek (az adattípus tartományának elemein végrehajtható műveletek listáját tartalmazza).
- Reprezentáció (azt határozza meg, hogy az adattípus elemei hány bájtra és milyen bitkombinációra képződnek le).

Minden nyelv rendelkezik beépített típusokkal, sőt egyes nyelvekben lehetőség van saját típus definiálására is.

Az egyes adattípusok egymástól különbözőek, van azonban olyan speciális eset, amikor egy típusból (**alaptípus**) úgy származtatunk egy másik típust (**altípus**), hogy leszűkítem annak tartományát, változatlanul hagyva műveleteit és reprezentációját. Az alaptípus és az altípus nem különböző típus.

Az adattípusoknak két nagy csoportja van:

- Skalár, egyszerű adattípus (tartománya atomi értékeket tartalmaz, nyelvi eszközökkel tovább nem bonthatóak).
- Strukturált, összetett adattípus (tartományának elemei maguk is valamilyen típussal rendelkeznek, az elemek egy-egy értékcsoportot képviselnek).

Egyszerű típusok (és reprezentációik)
- Egész (fixpontos).
- Előjel nélküli egész (direkt).
- Valós (implementáció függő).
- Karakteres (karakteres).
- Logikai (logikai).
- Felsorolásos - saját típusként kell létrehozni.
- Sorszámozott: a tartomány elemeihez kölcsönösen egyértelműen hozzá vannak rendelve a 0, 1, 2, … sorszámok, kivéve egész típusoknál.
- Intervallum: a sorszámozott típus egy altípusa.

Összetett típusok
Az eljárásorientált nyelvek két legfontosabb összetett típusa a **tömb** (minden nyelv ismeri) és a **rekord** (csak a FORTRAN nem ismeri).

A tömb az absztrakt adatszerkezet megjelenése típus szinten. A egy tömb statikus és homogén típus.

Egy tömböt meghatároz:

- Dimenziói.
- Indexkészletének típusa és tartománya.
- Elemeinek a típusa.

Többdimenziós tömbök reprezentációja lehet **sorfolytonos** vagy **oszlopfolytonos**, általában az implementáció dönti el. A nyelveknek a tömbbel kapcsolatban 4 dolgot kell tisztázniuk: elemek típusa, index típusa, indextartomány megadása, alsó és felső határ, illetve darabszám megadás módja.

A rekord heterogén típus, amelynek tartományának elemei olyan értékcsoportok, amelyeknek elemei különböző típusúak lehetnek. Az értékcsoporton belül az egyes elemeket **mező**nek nevezzük. Minden mezőnek saját neve és típusa van. A régebbi nyelvek többszintű rekord típussal dolgoztak, azaz egy mezőt fel lehetett osztani újabb mezőkre, tetszőleges mélységig. A mai nyelvekben már nincsenek ilyen, úgynevezett **almező**k. Az egyes mezőkre külön **minősített név**vel tudunk hivatkozni, ennek alakja: eszköznév.mezőnév.

Mutató típusok
Specialitását az adja, hogy tartományának elemei tárcímek, megvalósítható vele az **indirekt címzés**. Tartományának speciális eleme – amely nem valódi tárcím – egy nevesített konstans, amely nem mutat sehova, általában NULL –ként jelenik meg.

2.5. Nevesített konstans

Három komponense van:

- Név
- Típus
- Érték

Olyan programozási eszköz, amelyet mindig deklarálni kell. A program szövegében mindig a nevével jelenik meg és az mindig az értékkomponensét jelenti, amely a deklarációnál dől el, futás közben nem módosítható. A nyelveknek a nevesített konstanssal kapcsolatban 4 dolgot kell tisztázniuk:

- Létezik –e a nyelvben nevesített konstans? (FORTRAN-ban nincs)
- Definiálhatunk –e saját nevesített konstanst? (COBOL-ban csak beépített)
- Ha igen, milyen típusút?
- Hogyan adjuk meg az értékét? (Pl.: C-ben: #define konstans ertek)

2.6. Változó

Négy komponense van:
- Név (a név egy azonosító, a program szövegében mindig ezzel jelenik meg, a változó neve bármely komponensét jelölheti).
- Attribútumok (a futás közbeni viselkedést határozzák meg, a deklaráció során rendelődnek a változóhoz, a legfőbb attribútum a típus).
- Cím
- Érték

Deklarációs típusok:
- Explicit deklaráció (a programozó explicit módon utasítással deklarálja a változót, hozzárendelve nevéhez az attribútumait).
- Implicit deklaráció (a programozó betűkhöz rendel deklarációkat egy külön deklarációs utasításban, ha a változó nincs explicit módon deklarálva, akkor a nevének kezdőbetűjéhez rendelt attribútumokkal fog rendelkezni).
- Automatikus deklaráció (a fordítóprogram rendel attribútumot azokhoz a változókhoz, melyek nincsenek se explicit, se implicit módon deklarálva).

A futási idő azon részét, amikor egy változó rendelkezik címkomponenssel, a **változó élettartam**ának hívjuk.

Egy változóhoz cím rendelhető az alábbi módokon:
- Statikus tárkiosztás (a futás előtt eldől a változó címe, a futás alatt nem változik, a változó értéke a futás ideje alatt fix memóriacímen tárolódik).
- Dinamikus tárkiosztás (a cím hozzárendelését a futtató rendszer végzi, a változó akkor kap címkomponenst, amikor aktivizálódik a programegység, amelynek ő lokális változója, ha befejeződik: a címkomponens megszűnik, a címkomponens a futás során változhat).
- Programozó által vezérelt tárkiosztás (a változóhoz a programozó rendel címkomponenst, a címkomponens változhat, három típusa van: **abszolút** vagy **relatív** cím hozzárendelése, illetve egy **időpillanat** megadása, ahonnan a változóhoz címkomponens adódik, ezt a futtatórendszer végzi, mindhárom esetben van lehetőség a címkomponens megszűntetésére is).

Egy változó értékkomponensének meghatározására három módja van:
- Értékadó utasítás (bal oldalt a változó neve, jobb oldalt a leendő értékkomponens áll, típusegyenértékűséget valló nyelvekben a két típus azonos).
- Explicit kezdőértékadás (a programozó az explicit deklarációs utasításban a változó értékkomponensét is megadja).
- Automatikus kezdőértékadás (ha a programozó nem adott meg kezdőértéket, akkor a hivatkozási nyelv által meghatározott érték lesz felvéve).

A hivatkozási nyelvek egy része azt mondja, hogy mind addig, amíg a programozó nem határozza meg az értékkomponenst, az **határozatlan**.

3. Kifejezések

Olyan szintaktikai eszközök, amelyek segítségével már ismert értékekből új értékeket határozhatunk meg. Két komponensük van: érték és típus.

Egy kifejezés formálisan két összetevőből áll:
- Operandusok (lehet literál, nevesített konstans, változó vagy függvényhívás – ezek az értékek).
- Operátorok (műveleti jelek).
- Kerek zárójelek (a műveletek végrehajtási sorrendjét befolyásolhatják, redundánsan is alkalmazhatóak).

Egy kifejezés lehet **egyoperandusú** (unáris), **kétoperandusú** (bináris) vagy **háromoperandusú** (ternális).

Kétoperandusú kifejezések esetén három alak lehetséges attól függően, hogy hol helyezkedik el az operátor:
- Prefix (az operátor az operandusok előtt áll).
- Infix (az operátor az operandusok között áll).
- Postfix (az operátor az operandusok mögött áll).

Azt a folyamatot, amikor a kifejezés értéke és típusa meghatározódik, a **kifejezés kiértékelés**ének nevezzük. A kiértékelés során adott sorrendben végezzük el a műveleteket, melyekből előáll az érték és hozzárendelődik a típus. A műveletek végrehajtási sorrendje lehet: **balról-jobbra**, **jobbról-balra**, illetve ugyancsak **balról-jobbra** a **precedencia táblázat** szerint.

A precedencia táblázat sorokból áll, ahol a sorok fentről lefelé haladva az egyre kisebb prioritású operátorokat tartalmazzák. Minden azonos prioritású operátor azonos sorban szerepel. Minden sorban meg van adva a **kötési irány**, amely megmondja, hogy az adott sorban szereplő operátorokat milyen sorrendben kell kiértékelni, ha azok egymás mellett állnak egy kifejezésben.

Infix kifejezések esetében **zárójel**ek segítségével tudjuk felülbírálni a precedencia táblázat szabályait.

A kiértékelés szempontjából azonban speciálisak azok a kifejezések, amelyekben **logikai operátorok** szerepelnek. Ezeknél ugyanis létezik olyan eset, amikor a kifejezés értéke a teljes kiértékelés előtt eldől. Például ÉS műveletnél ha az első operandus értéke hamis, az eredmény biztosan hamis lesz, függetlenül a második operandus értékétől. Erre az eseményre a nyelvek a következőket reagálják:

- Teljes kiértékelés (FORTRAN).
- Rövidzár kiértékelés (PL/I,C).
- Mindkettő értelmezett, a programozó dönti el a kiértékelés módját (ADA: nem rövidzár: and, or | rövidzár: and then, or else).
- A kifejezés kiértékelését futási üzemmódként lehet beállítani (Turbo Pascal).

A kifejezés típusának meghatározáshoz kétféle elv létezik: (ahogy értékadó utasításnál és paraméterkiértékelésnél is)
- Típusegyenértékűség (az operandusok típusának meg kell egyeznie, feltétele a deklaráció-, név- vagy struktúra **egyenértékűség**, nincs konverzió).
- Típuskényszerítés (konverzió van, **bővítés** – értékvesztés nélküli, **szűkítés** – értékcsonkítás lehetséges, a szűkítést külön jelezni kell).

Azt a kifejezést, amelynek értéke fordítási időben dől el és kiértékelését a fordító végzi, **konstans kifejezé**snek nevezzük, operandusai literálok.

3.1. Kifejezés a C-ben

A C alapvetően kifejezésorientált nyelv, a típuskényszerítés elvét vallja. A tömb típusú eszköz neve mutató típusú, azaz a[i] = *(a+i).

Operátorai:

()	Függvényoperátor, a precedencia felülírására szolgál.
[]	Tömboperátor.
.	Minősítő operátor, struktúra és union esetén használjuk, ha névvel minősítünk.
->	A mutatóval történő minősítés operátora.
*	Indirekciós operátor. A mutató típusú operandusa által hivatkozott tárterületen elhelyezett értéket adja.
&	Az operandusának címét adja meg.
+ -	Előjelek.
!	Logikai tagadás.
~	Bitenkénti tagadás.
++ --	Operandusok értékének növelése, illetve csökkentése (n++, n-- esetén értékátadás, majd értékváltoztatás, ++n, --n esetén fordítva).
sizeof()	Ha kifejezést adunk meg: ábrázolási hosszát adja meg bájtban, ha típust adunk meg: ábrázolásához szükséges bájtok számát adja meg.

(típus) Explicit konverziós operátor.
* / % + - Aritmetikai műveletek operátorai.
>> << Bitenkénti léptetés operátorai.
< > <= >= = != Hasonlító operátorok, eredményük int típusú, igaz esetén 1, hamis esetben 0.
& ^ | Nem rövidzár logikai operátorok (bitenkénti műveltek).
&& || Rövidzár logikai operátorok.
? : Feltételes operátor (ha az 1. operandus értéke nem 0, akkor az eredményt a 2. operandus értéke határozza meg, egyébként a 3. operandus)
= += -= *= /= %= >>== <<== &= ^= |= Értékadó operátorok.
, A balról jobbra kiértékelést kényszeríti ki.

4. Utasítások

Az utasítások alkotják az eljárásorientált nyelveken megírt programok olyan egységeit, amelyekkel egyrészt az algoritmusok egyes lépéseit megadjuk, másrészt a fordítóprogram ezek segítségével generálja a tárgyprogramot. Két nagy csoportjuk van:

A **deklarációs utasítások** mögött nem áll tárgykód. Ezen utasítások teljes mértékben a fordítóprogramnak 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 **végrehajtható utasítások**ból generálja a fordítóprogram a tárgykódot. A végrehajtható utasításokat az alábbiak szerint csoportosíthatjuk:
- Értékadó utasítás (feladata beállítani vagy módosítani egy változó értékkomponensét).
- Üres utasítás (egyes nyelvekben külön alapszó jelöli, pl.: CONTINUE (Fortran), NULL (Ada), vagy nem jelöli semmi, pl.: kód; ; kód; … (C)).
- Ugró utasítás (feltétel nélküli vezérlésátadó, a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra ugorhatunk, GOTO).
- Elágaztató utasítások
- Ciklusszervező utasítások
- Hívó utasítás
- Vezérlésátadó utasítások
- I/O utasítások
- Egyéb utasítások

4.1. Elágaztató utasítások

Kétirányú elágaztató utasítás (feltételes utasítás)
Arra szolgál, hogy a program egy adott pontján két tevékenység közül válasszunk, illetve egy adott tevékenységet végrehajtsunk vagy sem. A nyelvekben meglehetősen általános a feltételes utasítás következő szerkezete: IF feltétel THEN tevékenység [ELSE tevékenység].

A **feltétel** egy logikai (vagy annak megfelelő típusú) kifejezés. A **tevékenység** megadásának módjáról azonban a nyelvek különböző nézeteket vallanak. A Pascal szerint ha a tevékenység nem csak egy utasításból áll, akkor **utasítás zárójelek**et kell alkalmazni, amely ez esetben a BEGIN és az END, az ilyen módon bezárójelezett utasássorozatot nevezzük **utasítás csoport**nak. Az Ada megengedi, hogy a tevékenység tetszőleges végrehajtható utasássorozatot tartalmazzon, más jelölések nélkül. A nyelvek egy harmadik csoportja - mint ahogy a C nyelv is – azt vallja, hogy a tevékenység helyén vagy **egyetlen** végrehajtható utasítás, vagy egy **blokk** állhat.

A kétirányú elágaztató utasításnál beszélünk **rövid** (nem szerepel ELSE ág) és **hosszú** (szerepel ELSE ág) alakról. Az IF utasítások tetszőlegesen egymásba ágyazhatóak, akár egymás ELSE ágába is. Ekkor felmerülhet a **„csellengő ELSE”** problémája. Erre három megoldás létezik: mindig hosszú IF utasítást írunk, a nyelv szintaktikája egyértelmű skatulyázást ír elő, implementációfüggő dolog: általában bentről kifelé való értelmezés.

A kétirányú elágazás szemantikája:
Kiértékelődik a feltétel. Ha **igaz**: végrehajtódik a THEN utáni tevékenység, majd a program az IF utasítást követő utasításon folytatódik. Ha **hamis**: ha **van ELSE ág**, akkor az ott megadott tevékenység hajtódik végre ugyanúgy, mint az igaz érték esetén, azonban ha **nincs ELSE ág**, akkor a program tovább folytatódik az IF utasítást követő utasítással, az IF utasításunk ekkor **üres utasítás**nak tekinthető.

Többirányú elágaztató utasítás
Arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró akárhány tevékenység közül egyet végrehajtsunk. A megfelelő tevékenységet egy kifejezés értéke alapján határozzuk meg. Az egyes nyelvekben eltérő szintaktikája, valamint szemantikája van:
- **Turbo Pascal**: konstanslista (literálok vagy intervallumok vesszővel elválasztott sorozata), a tevékenység lehet utasításcsoport is, ELSE ág.
- **Ada**: WHEN ágak, WHEN OTHERS ág, a kifejezés minden lehetséges értékére elő kell írni valamilyen tevékenységet, különben futtatási hiba.
- **C**: DEFAULT ág (bárhol állhat – az előzőekben csak a legvégén), a tevékenység lehet blokk is, BREAK parancs.
- **Fortran** és **Cobol**: nem tartalmaz többirányú elágaztató utasítást.
- **PL/I**: A kifejezések típusa tetszőleges, a tevékenység lehet utasítás csoport és blokk is, WHEN ágak, ha a kifejezés értéke nem szerepel egyik ágban sem, akkor az egyes ágak kifejezéseinek értékét bitlánccá konvertálja és az első olyan ágat választja ki, amelynek a bitjei nem csupa 0.

4.2. Ciklusszervező utasítások

Lehetővé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételhessen. Egy ciklus általános felépítése:
- Fej (az ismétlésre vonatkozó információkat tartalmazhatja).
- Mag (az ismétlendő végrehajtható utasításokat tartalmazhatja).
- Vég (az ismétlésre vonatkozó információkat tartalmazhatja).

A ciklusok működésénél megkülönböztethetünk két szélsőséges esetet: az **üres ciklus**t (egyszer sem fut le) és a **végtelen ciklus**t. A végtelen ciklus szemantikai hibát jelent, hiszen az soha sem fejeződik be.

A programozási nyelvekben a következő ciklusfajtákat különböztetjük meg:
- Feltételes ciklus (Az ismétlődést egy **logikai feltétel** szabályozza, ami a fejben vagy a végben van, két típusa van. **Kezdőfeltételes**: a feltétel a fejben van, addig ismétlődik, amíg a feltétel hamissá nem válik, lehet üres ciklus, ha a feltétel a legelső esetben már hamis. **Végfeltételes**: a feltétel általában a végben van, addig ismétlődik, amíg a feltétel igazzá nem válik (valahol fordítva), egyszer mindenképpen lefut, így nem lehet üres ciklus, végtelen azonban igen, ha a feltétel értéke a második ismétlés után nem változik.)
- Előírt lépésszámú ciklus (Az ismétlődésre vonatkozó adatok a **ciklusparaméterek**, amelyek a fejben vannak, hozzájuk tartozik a **ciklusváltozó**, értékeit egy megadott tartományból veheti fel, amelynek **kezdő-** és **végérték**ét mi biztosítjuk. A ciklusváltozó a hozzá tartozó tartományban felvehető értékeit a **lépésköz** határozza meg, amelynek **iránya** van.)

Az előírt lépésszámú ciklussal kapcsolatban a nyelveknek 6 dolgot kell tisztázniuk:
- Ciklusváltozó típusa (lehet egész, valahol sorszámozott is, máshol valós is, vagy: kezdőérték, végérték és lépésköz típusa konvertálható legyen).
- Kezdőérték, végérték és lépésköz megadásának formája (mindenhol megengedett a literál, változó és nevesített konstans, később a kifejezés is).
- Irány meghatározása (lépésköz előjele, külön alapszó).
- Ciklusparaméterek kiértékelődésének száma (általában egyszer, minden ciklusmag-végrehajtás előtt).
- Ciklus befejeződése (szabályos lefutás, GOTO utasítással – általában ez nem szabályos).
- Ciklusváltozó értéke a ciklus lefutása után (GOTO után az utolsó felvett érték, szabályos befejezés esetén ált. határozatlan, implementációfüggő).

Működését tekintve az előírt lépésszámú ciklus lehet **előltesztelő** vagy **hátultesztelő**. Nem minden hivatkozási nyelv definiálja ezt, ezért általában implementáció függő, így általában az előltesztelő változat valósul meg. Előltesztelő ciklus esetében először a ciklusváltozó kap értéket, illetve léptetődik, utána fut le a mag. Hátultesztelő ciklus esetében először lefut a mag, majd utána a ciklusváltozó megkapja az értékét, illetve léptetődik, amennyiben a leendő felvett érték még szerepel a tartományban. Fontos megjegyezni, hogy a hátultesztelő ciklus soha nem üres ciklus.

- Felsorolásos ciklus (Az előírt lépésszámú ciklus egyfajta általánosítása, van ciklusváltozója, amely a fejben van és értékét explicit módon kapja – ez egy kifejezést, nem lehet sem üres, sem végtelen ciklus.)
- Végtelen ciklus (sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan, a magban kell kiléptető utasítást alkalmazni).
- Összetett ciklus (az előző négy ciklusfajta kombinációiból áll elő).

A nyelvek egy részében vannak olyan **vezérlésátadó utasítások**, amelyek bármilyen típusú ciklus **szabályos befejeződés**ét eredményezik. A C nyelvben ez a három utasítás: **continue;** (a ciklus magjának hátralevő utasításait nem hajtja végre, hanem új ciklusba kezd), **break;** (a ciklust szabályosan befejezi), **return [kifejezes];** (szabályosan befejezteti a függvényt és visszaadja a vezérlést a hívónak). Az Ada nyelvben az **EXIT** utasítással tudunk szabályosan kilépni a ciklus magjából.

5. Programok szerkezete

Az eljárásorientált programnyelvekben a program szövege egymástól független **programegység**ekre bontható. Az egyes nyelveknek 5 dolgot kell tisztázniuk ezzel kapcsolatban:

- A teljes szöveget egyben kell lefordítani, vagy az feltördelhető önálló fordítható részekre? (fizikailag önálló, mélységében strukturálható, vegyes)
- Ha a részek külön fordíthatóak, mi alkothat egy önálló fordítási egységet?
- Milyen programegységek léteznek? (alprogram, blokk, csomag, taszk)
- Milyen a programegységek viszonya?
- A programegységek hogyan kommunikálnak egymással?

5.1. Alprogramok

Mint programozási eszköz az újrafelhasználás eszköze. Akkor alkalmazható, ha a program különböző pontjain ugyanaz a programrész megismétlődik. 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 **aktivizál**ható. Az alprogram attól lesz absztrakciós eszköz, hogy a kiemelt programrészt **formális paraméter**ekkel látjuk el.

Formálisan az alprogram a következőképpen épül fel:
- Fej (vagy specifikáció)
- Törzs (vagy implementáció)
- Vég

Az alprogram, mint programozási eszköz négy komponensből áll:
- Név (egy azonosító, mindig a fejben szerepel).
- Formális paraméter lista (a fej része, a listában azonosítók szerepelnek, ezeknek a törzsben az egyes programozási eszközök nevei, az alprogram meghívásakor ezeket konkretizálni kell – erre jók az **aktuális paraméterek**, ha üres, akkor paraméter nélküli alprogram).
- Törzs (itt szerepelnek a deklarációs és a végrehajtható utasítások, egyes nyelvekben ezt a két részt el kell egymástól különíteni, másokban keverik).
- Környezet

Az alprogramban deklarált programozási eszközöket az alprogram **lokális eszköze**inek nevezzük, ezek nevei az alprogram **lokális nev**ei. Ezek a nevek az alprogramon kívül nem láthatóak. Ugyanakkor léteznek **globális nev**ek, amelyeket nem az adott alprogramban deklaráltunk, hanem valamelyik hívóprogramjában. Az alprogram **környezet**e alatt a globális változók együttesét értjük.

Az alprogramoknak két fajtája van:
- Eljárás (tevékenységet hajt végre).
- Függvény (egyetlen értéket határoz meg, visszatérési értéke van, amelynek típusa meg van adva).

Azt a szituációt, amikor a függvény megváltoztatja paramétereit vagy a környezetét, a **függvény mellékhatás**ának nevezzük, ez általában káros.

Egy eljárást **aktivizál**ni utasításszerűen lehet, azaz elhelyezhető a programszövegben bárhol, ahol végrehajtható utasítás állhat. Egy eljárás meghívására egyes nyelvekben külön alapszó szolgál (általában CALL), más nyelvekben nincs alapszó. Híváskor a vezérlés átadódik az eljárásra. Egy eljárás szabályosan fejeződik be, ha elérjük a végét, vagy ha külön utasítással befejeztetjük. Nem szabályos azonban a befejezés, ha GOTO utasítással kiugrunk az alprogramból.

Függvényt **meghív**ni csak kifejezésben lehet, befejeződése után a vezérlés a kifejezésbe tér vissza, és tovább folytatódik annak a kiértékelése. Egy függvény a következő módokon határozhatja meg a visszatérési értékét:
- A függvény törzsében változóként használható a függvény neve (visszatérési értéke az utolsó felvett értéke lesz).
- A függvény törzsében a függvény nevéhez értéket kell hozzárendelni (a függvény neve hordozza az értéket).
- Külön utasítás szolgál a visszatérési érték meghatározására (C nyelvben: return parancs).

Egy függvény szabályosan ér véget, ha elérjük a végét (és van visszatérési értéke), befejező utasítást alkalmazunk (és van visszatérési értéke) vagy ha olyan befejező utasítást alkalmazunk, amely egyben meghatározza a visszatérési értéket. Azonban nem szabályos a befejezés, ha nincs visszatérési értéke, illetve ha GOTO utasítással lépünk ki.

A **főprogram** egy speciális programegység, amelynek a program indulásakor a vezérlő adja át a vezérlést és az összes többi programegység működését ő koordinálja. Szabályos program a főprogram szabályos befejeződésével ér véget, ekkor a vezérlés visszakerül az operációs rendszerhez.

5.2. Hívási lánc, rekurzió

Bármely programegység meghívhat egy másik programegységet, így kialakulhat egy **hívási lánc**, melynek első tagja mindig a főprogram. A lánc minden tagja **aktív**, de csak a legutoljára meghívott programegység **működik**. Azt a szituációt, amikor egy már aktív alprogramot hívunk meg, **rekurzió**nak nevezzük. Két típusa van: **közvetlen rekurzió**: egy alprogram önmagát hívja meg, **közvetett rekurzió**: a hívási lánc egy korábbi alprogramját hívjuk meg. Minden rekurziós algoritmus átírható iteratívvá (amely gyorsabb). A Fortran nem ismeri a rekurziót, más nyelvek szerint az alprogram alapértelmezetten rekurzív, egyes nyelvekben pedig a programozó döntheti el, hogy egy adott alprogram rekurzív legyen-e, vagy sem.

5.3. Másodlagos belépési pontok

Egyes nyelvek megengedik, hogy egy alprogramot meghívni ne csak a fejen keresztül lehessen, hanem a törzsben ki lehessen alakítani úgynevezett másodlagos belépési pontokat. A másodlagos belépési pont képzése formálisan meg kell feleljen a specifikációnak. Ha függvényről van szó, akkor a típusnak meg kell egyeznie. Ha az adott alprogramba a fejen keresztül lépünk be, akkor az alprogram teljes törzse végrehajtódik, másodlagos belépési pont használata esetén a törzsnek csupán egy része hajtódik végre.

5.4. Paraméterkiértékelés

Azt a folyamatot értjük **paraméterkiértékelés** alatt, amikor egy alprogram hívásánál egymáshoz rendelődnek a **formális** és **aktuális paraméter**ek. Paraméterkiértékelésnél mindig a formális paraméter lista az elsődleges, ezt az alprogram specifikációja tartalmazza, egy darab van belőle. Aktuális paraméter lista viszont annyi lehet, ahányszor meghívjuk az alprogramot. Mindig az aktuális paramétereket rendeljük a formálisakhoz. Az egyes nyelveknek a paraméterkiértékeléssel kapcsolatban 3 kérdésre kell választ adniuk:

- Melyik formális paraméterhez melyik aktuális paraméter rendelődik hozzá? (sorrendi kötés – felsorolás sorrendjében, név szerinti kötés - ritka)
- Hány darab aktuális paramétert kell megadni? (**Fix formális paraméter**: vagy egyeznek, aktuális paraméterek kevesebbek - ekkor a pártalan paraméterek alapértelmezett értéket vesznek fel. **Nem fix formális paraméter**: az aktuális paraméterek száma tetszőleges, létezik a paraméterek alsó korlátjának esete is.)
- Mi a viszony a formális és az aktuális paraméterek típusai között? (a legtöbb nyelv típusegyenértékűséget vall, mások a típuskényszerítését)

5.5. Paraméterátadás

A paraméterátadás az alprogramok és más programegységek közötti kommunikáció egy formája. A paraméterátadásnál mindig van egy hívó, és egy hívott, amelyik mindig alprogram. De mégis melyik irányba és milyen információ mozog? A nyelvek az alábbi paraméterátadási módokat ismerik:

– érték szerinti (formális paraméter: címkomponens, aktuális paraméter: értékkomponens, ezen érték kerül a címkomponensre a paraméterkiértékelés után, és az alprogram ezzel az értékkel dolgozik, a hívott alprogram saját területén dolgozik, független a hívótól és az is tőle).

– cím szerinti (formális paraméter: nincs címkomponens, aktuális paraméter: címkomponens, paraméterkiértékelés után az aktuális paraméter címkomponense átadódik a formális paraméternek, a meghívott alprogram a hívott területén dolgozik, kétirányú információátadás).

– eredmény szerinti (formális paraméter: címkomponens, aktuális paraméter: címkomponens, paraméterkiértékelés után az aktuális paraméter címe átadódik a formális paraméterhez, az alprogram azonban a saját területén dolgozik – nem használja a kapott címet, végül az eredményt visszamásolja az aktuális paraméter címére, a kommunikáció egyirányú).

– érték-eredmény szerinti (formális paraméter: címkomponens, aktuális paraméter: érték- és címkomponens, paraméterkiértékelés után az aktuális paraméter értéke és címe is átadódik a formális paraméterhez, a végeredmény értéke ezután visszamásolódik az aktuális paraméter címére, kétirányú kommunikáció, kétszeres értékmásolás).

– név szerinti (aktuális paraméter: tetszőleges szimbólumsorozat, paraméterkiértékeléskor rögzítődik az alprogram szövegkörnyezete, értelmezésre kerül az aktuális paraméter, majd a szimbólumsorozat felülírja a formális paraméter nevének minden előfordulását az alprogram szövegében, és ezután fut le).

– szöveg szerinti (annyiban különbözik a név szerinti paraméterátadástól, hogy a hívás után az alprogram elkezd működni, a felülírás azonban csak akkor következik be, amikor előfordul az alprogram szövegében a formális paraméter neve.)

Az első négy paraméterátadási módszerben az aktuális paraméter változó is lehet. Alprogramok esetén típust paraméterként átadni nem lehet.

Az egyes nyelvek különböző módon döntik el a paraméterátadás kérdését: csak egyetlen paraméterátadási típust értelmez (C), explicit módon megadjuk a paraméterátadás típusát (Ada), az aktuális és formális paraméter típusa dönti el (PL/I), a formális paraméter típusa dönti el (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).

5.6. A blokk

Olyan programegység, amely csak másik programegység belsejében helyezkedhet el. Formálisan van **kezdet**e, **törzs**e és **vég**e. A kezdetet és a véget egy-egy speciális karaktersorozat vagy alapszó jelzi. A törzsben lehetnek deklarációs és végrehajtható utasítások. Nincs paramétere, azonban egyes nyelvekben lehet neve. Ez általában a kezdet előtt álló címke. Bárhol elhelyezhető, ahol végrehajtható utasítás állhat. Aktivizálni vagy úgy lehet, hogy szekvenciálisan rákerül a vezérlés, vagy úgy, hogy GOTO utasítással ráugrunk a kezdetére. Egy blokk befejeződhet, ha elértük a végét, vagy GOTO utasítással kilépünk belőle. Az eljárásorientált nyelveknek csak egy része ismeri. Szerepe a nevek **hatáskör**ének elhatárolásában van.

5.7. Hatáskör

Egy név hatásköre alatt értjük a program szövegének azon részét, ahol az adott név ugyanazt a programozási eszközt hivatkozza. Szinonimája a láthatóság. Egy programegységben deklarált nevet a programegység **lokális nev**ének nevezzük. Azt a nevet, amelyet nem a programegységben deklaráltunk, de ott hivatkozunk rá, **szabad név**nek hívjuk.

Azt a tevékenységet, mikor egy név hatáskörét megállapítjuk, **hatáskörkezelés**nek hívjuk. Kétféle hatáskörkezelést ismerünk:

A **statikus hatáskörkezelés** fordítási időben történik, a fordítóprogram végzi. Alapja a programszöveg **programegység szerkezete**. Ha a fordító talál egy szabad nevet, akkor kilép a tartalmazó programegységbe, és megnézi, hogy a név ott lokális-e. Ha igen vége a folyamatnak, ha nem, akkor tovább lépked kifelé, egészen addig, amíg meg nem találja lokális névként, vagy el nem jut a legkülső szintre.

Ha kiért a legkülső szintre, akkor két eset lehetséges:

- A nyelvek egy része azt mondja, hogy a programozónak minden nevet deklarálni kell,. ha egy név nem volt deklarálva, az fordítási hiba.
- A nyelvek másik része ismeri az automatikus deklarációt, ilyenkor tehát a legkülső szint lokális neveként értelmeződik.

Statikus hatáskörkezelés esetén egy **lokális név hatásköre** az a programegység, amelyben deklaráltuk és minden olyan programegység, amelyet ez az adott programegység tartalmaz, hacsak a tartalmazott programegységekben a nevet nem deklaráltuk újra. A hatáskör befelé terjed, kifelé soha. Azt a nevet, amely egy adott programegységben nem lokális név, de ott látható, **globális név**nek hívjuk. A globális név és lokális név relatív fogalmak.

A **dinamikus hatáskörkezelés** futási idejű tevékenység, a futtató rendszer végzi. Alapja a **hívási lánc**. Ha a futtató rendszer 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. Ez utóbbi esetben vagy futási hiba keletkezik, 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. Újradeklarálás esetén a hívási lánc további elemeiben az újradeklarált név látszik, nincs „lyuk a hatáskörben” szituáció.

Az eljárásorientált nyelvek a statikus hatáskörkezelést valósítják meg. Általánosságban elmondható, hogy az alprogramok formális paraméterei az alprogram lokális eszközei, így neveik az alprogram lokális nevei. Viszont a programegységek neve a programegység számára globális.

5.8. Fordítási egység

A program fordítási egységekből épül föl, ezek olyan forrásszöveg részek, amelyek a program többi részétől különválasztva fordíthatóak le.

5.9. Az egyes nyelvek eszközei

FORTRAN:

- Fizikailag önálló programegységek, amelyek külön fordíthatóak (csak az alprogramot ismeri).
- Nincsenek globális változók, speciális tárterület alkalmazása a kommunikációhoz: közös adatmező.
- Automatikus deklaráció.
- A főprogramnak nincs külön kezdő utasítása.
- Eljárás hívása a CALL alapszóval történik, szabályos befejeződésének parancsa RETURN [n].
- Ismeri a másodlagos belépési pontot.
- Paraméterkiértékelésnél a sorrendi kötést és a típusegyenértékűség (legalább egy formális paramétert meg kell adni).
- Paraméterátadás: tömb esetén cím szerinti, skalár esetén érték szerinti, alprogram név esetén név szerinti, azonosító és pointer esetén cím szerinti.
- Nem ismeri a rekurziót.

PL/I:

- Az alprogramok lehetnek egymástól függetlenek és egymásba ágyazhatóak is (ismeri az alprogramot, blokkot és a taszkot is).
- A deklarációs és a végrehajtható utasítások tetszőlegesen keverhetőek.
- A formális paraméter lista csak a paraméterek neveit tartalmazza, deklarálni őket a törzsben kell.
- A programozó jelöli hogy egy alprogram rekurzív legyen-e vagy sem (RECURSIVE).
- RETURNS parancs csak függvények esetén használandó.
- Eljárást CALL paranccsal lehet hívni.
- Ismeri a másodlagos belépési pontot.
- Paraméterkiértékelésnél sorrendi kötés számbeli egyeztetés van, valamint típuskényszerítés.
- Paraméterátadás: címke, változó esetén: cím szerinti, állomány neve esetén név szerinti, minden más esetben érték szerinti.
- Automatikus deklaráció, a változó élettartamát attribútumok határozzák meg.
- Statikus hatáskörkezelés.

Pascal:

- A fordítási egység a főprogram (csak az alprogramot ismeri).
- Az alprogramok a főprogram deklarációs részébe skatulyázandóak.
- A formális paraméterek száma nem fix, paramétercsoportokból áll.
- VAR kulcsszóval a paraméterátadás cím szerinti, egyébként érték szerinti.
- Paraméterkiértékelésnél sorrendi kötés, számbeli- és típusegyeztetés van.
- Egy változó élettartam-kezelés és a tárkiosztás a programozó által vezérelt.
- Eljáráshívásra nincs külön alapszó.
- A rekurzió alapértelmezett.

Ada:

- Fordítási egység a forrásállomány (amely külső deklarációkat tartalmaz).
- A külön fordított egységek fizikailag önállóak, a kapcsolatszerkesztő dolga, hogy kapcsolatot létesítsen köztük (minden programegységet ismer).
- Nincs főprogram, implementációfüggő, hogy melyik program tekintendő annak.
- Eljáráshívásra nincs külön alapszó, RETURN vagy END parancs fejezi be.
- A formális paraméterlista paramétercsoportokból áll.
- A paraméterátadás módját egy alapszó dönti el: IN (érték szerinti - alapértelmezett), OUT (eredmény szerinti), IN OUT (érték-eredmény szerinti).
- Paraméterkiértékelésnél szigorú típusegyeztetés van, sorrendi vagy esetleg név szerinti kötés.
- Hatáskörkezelése statikus, kezeli a „lyuk a hatáskörben” problémát.
- A változók élettartama dinamikus.
- A rekurzió alapértelmezett.

C:

- A függvényt és a blokkot ismeri (a függvények nem ágyazhatóak egymásba, a főprogram is egy függvény, a blokkok egymásba skatulyázhatóak).
- Eljárás jelölése: void típus, a függvények alapértelmezetten int típusúak.
- Egy függvény befejeződhet RETURN kifejezes; paranccsal, vagy ha elérjük a záró } jelet.
- Lehetőség van nem fix paraméterszámú függvény deklarálására.
- A paraméterkiértékelés sorrendi kötést és típuskényszerítést vall, fix paraméterszám esetén számbeli egyeztetés.
- A paraméterátadás kizárólag érték szerinti.
- Az élettartam szabályozására tárolási osztály attribútumokat használ (extern, auto, register, static).
- A rekurzió alapértelmezett.

6. Absztrakt adattípus

Olyan adattípus, amely megvalósítja a **bezárás**t vagy információ rejtést. Ez azt jelenti, hogy ezen adattípusnál nem ismerjük a reprezentációt és a műveletek implementációját. 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.

7. A csomag

A csomag az a programegység, amely egyaránt szolgálja a procedurális és az adatabsztrakciót. A procedurális absztrakció oldaláról tekintve a csomag programozási eszközök **újrafelhasználható gyűjtemény**e. 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. A csomag az Adában jelenik meg. Az Ada csomagnak két része van: **specifikáció** és **törzs**.

Adában a csomag két részből áll: **látható** és **privát** rész. A látható részben deklarált programozási eszközök hivatkozhatóak a csomagon kívülről, azonban a privát részben deklarált eszközöket bezárja, elrejti a külvilág elől. A csomag törzse opcionális, ha viszont a specifikációban szerepel alprogramspecifikáció, akkor kötelező a törzs. A törzs a külvilág számára nem elérhető.

A látható részben szerepelhet a privát típusmegjelölés, azonban csak egyenlőségvizsgálat és az értékadás alkalmazható. Létezik a privát típusnak egy olyan változata, melynek neve **korlátozott privát típus**. Erre még az egyenlőségvizsgálat és az értékadás beépített művelete sem alkalmazható, ekkor ezekhez is a programozónak kell implementálnia.

Ha egy csomag látható részében változókat deklarálunk, azok **OWN típus**úak lesznek, ami azt jelenti, hogy ezek két alprogramhívás közt megtartják értéküket. Az Ada nyelvben a csomag fordítható önállóan, azonban ekkor a program más részei számára explicit módon láthatóvá kell tenni.

8.1. Az ADA fordításáról – Pragmák

Olyan utasítások, amelyek a fordító működését befolyásolják. A fordítóprogramnak szólnak, szolgáltatást kérnek tőle, valamilyen üzemmódot állítanak be, nem áll mögöttük közvetlen kód, de befolyásolhatják a kódot. A pragmák egy része a program szövegének bármely pontján elhelyezhető, másik része csak kötött helyen használható. Az Ada rendszerekben általában mintegy 50 féle pragma van. Például:

- INTERFACE (Az adott alprogram specifikációja után kell megadni, és azt jelzi, hogy az adott alprogram törzse az adott nyelven van megírva.)
- LIST (Fordítás közben a programszövegről lista készül a szabvány kimeneten (ON), vagy letiltjuk a listázást (OFF). Bárhol elhelyezhető.)

Ha a fordító nem ismeri föl a pragma nevét, akkor ignorálja azt.

8.2. Fordítási egységek

Az Adában fordítási egység lehet:

- Alprogram specifikáció
- Alprogram törzs
- Csomag specifikáció
- Csomag törzs
- Fordítási alegység

Azokat a fordítási egységeket, amelyek nem függetlenek más fordítási egységektől, **könyvtári egység**eknek nevezzük. Ilyeneket tetszőleges számban hozhatunk létre. Ezeknek külön, egyedi nevük van. Ha használni akarok egy olyan eszközt, amelye egy másik fordítási egységben van benne, akkor alapfeltétel, hogy annak már lefordítva kell lennie, mielőtt még hivatkoznánk rá. Azaz a főprogramot kell utoljára megírni.

Ha a specifikáció módosul, újra kell fordítani azt a fordítási egységet, ha csak az implementáció változik, csak az azt tartalmazó fordítási egységet kell újrafordítanunk. Így a képesek vagyunk a programok párhuzamos fejlesztésére, a program módosítása egyszerűbbé válik és biztonságossá.

Az Adában minden fordítási egység előtt meg kell adni az úgynevezett **környezeti előírás**t, amely a WITH paranccsal történik. Ebben soroljuk fel azon könyvtári egységeket, amelyekre az adott fordítási egységben hivatkozunk, ezek a könyvtári egységek alkotják a **fordítási egység környezetét**. Az Ada nyelvben kilenc szabvány könyvtári egység van, ezeket is szerepeltetni kell a környezeti előírásban. Erre **a STANDARD** nevű könyvtári egység nyújt hatékony megoldást, amely tartalmazza a szükséges könyvtári egységeket (ez tartalmazza az alapvető nyelvi eszközöket is).

Alegységnek hívjuk az önállóan nem létező fordítási egységeket. Ezek környezetét a **csonk** határozza meg, amely lehetővé teszi egy alprogram, csomag vagy taszk törzsének a beágyazását a fordítási egységbe úgy, hogy azt fordítási alegységnek tekintsük.

9. Kivételkezelés

Azt teszi lehetővé, hogy az operációs rendszertől átvegyük a megszakítások kezelését, felhozzuk azt a program szintjére. A **kivétel**ek 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 lehetőség**ét teszi lehetővé a programozásban.

Az 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, azonban nem tudjuk hogy ennek milyen hatása lesz a program további működésére. A kivételeknek általában van **nev**e (amely gyakran az eseményhez kapcsolódó üzenet szerepét játssza) és **kód**ja (ami egy egész szám).

A kivételkezelés a PL/I-ben és az Ada nyelvben van jelen. A két nyelv eltérő kivételkezelési filozófiát vall. A PL/I-ben ha a program futása közben kivétel következik be, akkor keressük meg a hiba okát, **szűntessük meg** és térjünk vissza oda, ahol a kivétel kiváltódott. Az Ada nyelv ellenkezőjét vallja: ha a program futása közben kivétel következik be, **hagyjuk ott** és végezzük a következő tevékenységet.

A kivételkezeléssel kapcsolatos kérdések:

- Milyen beépített kivételek vannak?
  - **PL/I**: Alapértelmezetten engedélyezett/tiltott és engedélyezhető/letiltható, illetve le nem tiltható.
  - **ADA**: Deklarációs, aritmetikai, tár, taszk és SELECT-utasítás hibát kivételkezelése (alapértelmezetten mindegyik engedélyezett).
- Lehet-e saját kivételt írni?
  - **PL/I**: CONDITON parancs segítségével.
  - **ADA**: EXCEPTION parancs segítségével.
- Milyen hatásköri szabályok vannak?
  - **PL/I**: Dinamikus, hatáskör ott kezdődik, hogy a vezérlés áthalad rajta, egészen REVERT parancsig, vagy programegység befejezéséig.
  - **ADA**: Dinamikus, a hívási láncon át öröklődik.
- Köthető-e programelemhez a kivételkezelés?
  - **PL/I**: ON függvények: segítenek behatárolni a kivételt kiváltó pontos eseményt, annak helyét, esetleg okát (ONCODE, ONKEY).
- Hogy folytatódik a program kivételkezelés után?
  - **PL/I**: Ha letiltott: folytatódik tovább, ha nem: lefut a kivételkezelés majd ha nincs hiba, folytatódik tovább a programegység végrehajtása.
  - **ADA**: Ha letiltott: folytatódik tovább, ha nem: befejeződik a programegység, majd a futtató rendszer megnézni, hogy az adott programegységben van-e kivételkezelő, ha van: WHEN ágak kivételei közül megkeresni a megfelelőt: azt lefuttatni, GOTO esetén szabályos folytatódás. Ha egyik ág se egyezik: WHEN OTHERS ág, ha ott sincs egyezés: visszalépünk a hívási láncon és ha a lánc elején sincs lekezelve, akkor a vezérlés átadódik az operációs rendszernek. A kivételeket átugorjuk, ha nem tudjuk kezelni.
- Mi van, ha a kivételben van kivétel?
  - **ADA**: Fordítója nem tudja ellenőrizni a kivételkezelők működését.
- Létezik-e általános kivételkezelő?
  - **PL/I**: ERROR (ADA nyelven nincs).
- Lehet-e paraméterezni a kivételkezelőt? (Sem PL/I-ben, sem Adában nincs)
- Van-e beépített kivételkezelő? (Sem PL/I-ben, sem Adában nincs)

10. 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 eszközrendszer. 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. Általános alakja a **törzs** és a **formális paraméter lista**.

A törzs egy teljes alprogram vagy csomag deklarációja, amiben a formális paraméterek szerepelnek, amelyek lehetnek változók, típusok vagy alprogramspecifikációk is. A formális paraméterek száma mindig fix, paraméterkiértékeléskor a sorrendi kötés alapértelmezett, de használható a név szerinti kötés is. A paraméterátadás változónál érték, típusnévnél név szerint történik.

11. Párhuzamos programozás

A Neumann-architektúrán felépülő gépek szekvenciálisak: a processzor a programnak megfelelő sorrendben hajtja végre az utasításokat. Egy processzor által éppen végrehajtott gépi kódú programot **folyamat**nak vagy **szál**nak hívunk. Ha ezek a működő kódok az erőforrásokat egymagukban birtokolják, akkor folyamatról, ha bizonyos erőforrásokat közösen birtokolhatnak, akkor szálakról beszélünk. A folyamatok kezelése operációs rendszer szinten történik. A kérdés az, hogy nyelvi szinten milyen eszközök állnak rendelkezésünkre ezek leprogramozásához.

A párhuzamos programozás nyelvi alapfogalmai:

Kommunikáció: A folyamatok kommunikálnak egymással, adatcserét folytatnak.

Szinkronizáció: A párhuzamosan futó folyamatoknak bizonyos időpillanatokban találkozniuk kell. Előfordul, hogy a szinkronizációs ponton keresztül történik adatcsere, a szinkronizációs ponton keresztül zajlik a kommunikáció. Például olyan információt vár az egyik a másiktól, ami nélkül nem tud továbbhaladni.

Konkurencia: A folyamatok vetélkednek a programbeli erőforrásokért.

Kölcsönös kizárás: Mivel a folyamatok kizárólagosan birtokolják az erőforrásokat, biztosítani kell, hogy amíg az egyik folyamat módosítja az adatot, addig a másik folyamat ne használhassa fel azt.

A párhuzamos programozási eszközrendszer először a PL/I-ben jelent meg. Létezik a Pascalnak és a C-nek is olyan változata, amely ebben az irányban bővíti tovább a nyelvet. Azok az algoritmusok, amelyekkel eddig találkoztunk, mind szekvenciális algoritmusok voltak.

A programozási nyelveknek a párhuzamos programozás megvalósításához rendelkezniük kell különböző eszközökkel: a folyamatok kódjának megadására, a folyamatok elindítására és befejeztetésére, a kölcsönös kizárás kérésére, a szinkronizációra, a kommunikáció megvalósítására, a folyamatok működésének felfüggesztésére, a folyamatok prioritásának meghatározására és a folyamatok ütemezésére.

12. A taszk

Az Adában a taszk mint programegység szolgál a **párhuzamos programozás** megvalósítására. Mint programegység önállóan nem létezik, csak egy másik programegységbe beágyazva jelenhet meg a program szövegében. A taszkot tartalmazó programegységet **szülőegység**nek hívjuk. Egy szülőegységen belül akárhány **testvértaszk** elhelyezhető. Ezek azonos szinten deklarált taszkok. A taszkok tetszőleges mélységben egymásba ágyazhatók. A szülőegység és a testvértaszkok törzse mögötti folyamatok működnek egymással párhuzamosan.

Többprocesszoros rendszerek esetén elképzelhető, hogy minden taszk más-más processzoron fut. Ez a valódi párhuzamosság. Egyprocesszoros rendszerek is programozhatók párhuzamos módon, ekkor az operációs rendszer szimulálja a párhuzamosságot. Ez a **virtuális párhuzamosság**.

Egy taszk akkor kezdi el a működését, amikor elindul a szülőegysége. Ez egy kezdeti szinkronizáció. Egy taszk befejezi a működését, ha elfogynak az utasításai, ha a szülőegysége, vagy egy testvértaszkja **ABORT név;** parancsot kap, illetve ha explicit módon befejezteti saját magát egy utasítással. A szülőegység akkor fejeződik be, ha ő, mint programegység befejeződött, és ha az összes általa tartalmazott testvértaszk befejeződött. Ez egyfajta **végszinkronizációs pont** a szülőegység számára.

A taszknak két része van: **specifikáció** és **törzs**. A specifikációs részben az úgynevezett **entry-specifikációk** deklarálhatóak, ezek segítségével belépési pontokat adhatunk meg. A belépési pontok jelentik az ADA nyelvben a **szinkronizáció** eszközeit. A taszkok felhasználási módjait tekintve megkülönböztetünk két típust:

- Passzív taszkok (valamilyen szolgáltatást nyújtanak).
- Aktív taszkok (igénybe veszik a passzív taszkok szolgáltatásait).

A szinkronizációt az ADA **randevú**nak hívja. Az aktív taszk egy entry-hívással képez egy randevúpontot. Ez formálisan megegyezik az eljáráshívással. A passzív taszkon belül minden egyes entry-specifikációhoz meg kell adni legalább egy elfogadó utasítást. A passzív taszk egy ilyen elfogadó utasítással képez egy randevúpontot.

Alaphelyzetben a randevú a következő módon megy végbe:

A randevúhoz kell egy aktív taszk (amely meghív egy entryt) és egy passzív (amelyben a megfelelő elfogadó utasítás van). Elindul a két taszk, szekvenciálisan hajtja végre az utasításokat, amíg egy randevúponthoz nem ér valamelyikük. Amelyik hamarább eléri, megvárja a másikat (működését addig felfüggeszti). A randevú elsősorban a szinkronizáció eszköze, de lehetőség van benne adatcserére is (erre szolgálnak a belépési pont formális paraméterei).

Ha mindkét taszk odaér a randevúponthoz, akkor az IN és IN OUT paraméterek esetén információ adódik át az aktív taszktól a passzív felé. Ezután, ha van DO-END rész, akkor az végrehajtódik, a randevú végén pedig az OUT és IN OUT praméterek segítségével a passzív taszk felől mozog információ az aktív taszk felé. Randevúban szinkronizáció mindig, a kommunikáció pedig opcionális lehetőség.

A taszkok természetesen kommunikálhatnak **globális változó**k segítségével is, ezeket egy időben használja az összes testvértaszk. A kölcsönös kizárást a **SHARED pragma** biztosítja, amelyet a szülőegység deklarációs részében kell megadni. A **PRIORITY pragma** segítségével prioritás rendelhető a taszkokhoz. Az alacsonyabb prioritású taszk soha nem akadályozhatja a magasabb munkáját.

A **DELAY** parancs segítségével egy megadott értékű másodpercekben értelmezett késleltetést indíthatunk az adott taszkban.

A randevú bekövetkezte a taszkokban befolyásolható a SELECT utasítással. Szerepe különböző: az aktív taszk mindig maga hívja meg a belépési pontot, a passzív taszk viszont nem tudja, hogy egy szolgáltatást később igénybe akar-e venni valamelyik aktív taszk.

Aktív taszkban két SELECT utasítás alkalmazható:

- Feltételes randevúra szolgáló SELECT (ha létrejöhet: végbemegy, ha nem: az aktív taszk nem vár, ELSE ágra lép, majd kilép a SELECT-ből).
- Időzített randevúra szolgáló SELECT (ha létrejöhet: végbemegy, ha nem: késlelteti magát, majd újra próbálkozik, mindezt időnként ismétli).

Passzív taszkban elhelyezhető SELECT utasítás:

Legfőbb jellemzői az alternatíva: **elfogadó** (legalább egy kell), **késleltető** és **befejező alternatíva**. A késleltető és a befejező alternatíva kizárják egymást. Egy **alternatívá**t **nyílt**nak nevezünk, ha nem szerepel előtte WHEN feltétel, vagy ha mégis de az igaz, egyébként az alternatíva **zárt**.

Aktív taszk csak olyan passzív taszkkal tud randevúzni, amelyik még működik, ha egy olyan taszkkal akar randevúzni, amelyikkel nem lehetséges, akkor kiváltódik a TASKING_ERROR kivétel.

13. Input / Output

Az I/O az a területe a programnyelveknek, ahol azok leginkább eltérnek egymástól. Legtöbb esetben implementációfüggő, a legtöbb nyelv erre bízza a megoldást. 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. Egy programban a **logikai állomány** egy olyan programozási eszköz, amelynek neve van, és amelynél az absztrakt állományjellemzők (pl.: rekordazonosító) 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, kérdéses, hogy az adatmozgatás közben történik-e konverzió.
Ennek megfelelően kétféle adatátviteli mód létezik:
- Folyamatos (van konverzió, 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. 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 melyik helyen és hány karaktert alkotva jelenjen meg az egyedi adat.
  A nyelvekben ezek megadására három alapvető eszközrendszer alakult ki:
  - Formátumos módú adatátvitel (minden egyes egyedi adathoz explicit módon meg kell adni a kezelendő karakterek darabszámát és a típust).
  - Szerkesztett módú adatátvitel (minden egyes egyedi adathoz meg kell adni egy maszkot, a maszk elemeinek száma határozza meg a kezelendő karakterek darabszámát).
  - Listázott módú adatátvitel (a folytonos karaktersorozatban magában vannak a tördelést végző speciális karakterek).
- Bináris (nincs konverzió).
  A bináris adatátvitel esetén az adatok a tárban és a periférián ugyanúgy jelennek meg. Ez csak háttértárakkal való kommunikációnál jöhet szóba. Az átvitel alapja itt a **rekord**. Ha egy programban állományokkal akarunk dolgozni, akkor a következőket kell végrehajtanunk:
  - Deklaráció (a logikai állományt mindig deklarálni kell az adott nyelv szabályainak megfelelően).
  - Összerendelés (a logikai állománynak megfeleltetünk egy fizikai állományt).
  - Állomány megnyitása (egy állománnyal csak akkor tudunk dolgozni, ha megnyitottuk).
  - Feldolgozás (ha az állományt megnyitottuk, akkor abba írhatunk, vagy olvashatunk belőle).
  - Lezárás.

A programozási nyelvek a programozó számára megengedik azt, hogy input-output esetén ne állományokban gondolkozzon, hanem az írás-olvasást úgy képzelje el, hogy az közvetlenül valamelyik perifériával történik. Ezt hívjuk **implicit állomány**nak. az implicit állományt nem kell deklarálni, összerendelni, megnyitni és lezárni.

Az **implicit input állomány** a szabvány rendszerbemeneti periféria (általában a billentyűzet), az **implicit output állomány** a szabvány rendszerkimeneti periféria (általában a képernyő). A programozó bármely állományokkal kapcsolatos tevékenységet elvégezhet explicit módon. Ha az író és olvasó eszközben nem adjuk meg a logikai állomány nevét, akkor a művelet az implicit állománnyal történik.

13.1 Az egyes nyelvek I/O eszközei

- Fortran: Szeriális, szekvenciális és direkt állományt tud kezelni.
- Cobol: Erős I/O eszközrendszere van, mindig konvertál.
- PL/I: Kiemelkedően a legjobb állománykezelési eszközrendszerrel dolgozik.
- Pascal: Állománykezelési rendszere szegényes.
- C: Az I/O eszközrendszer nem része a nyelvnek, STANDARD könyvtári függvények állnak rendelkezésre.
- Ada: Minden perifériát tud nyelvi eszközökkel kezelni. Az Adának sem része az I/O, csomagok segítségével valósítja meg azt.

14. Implementációs kérdések

Az eljárásorientált 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 velem (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. A **kódszegmens** a program gépi nyelvű utasításait, rendszerinformációkat és a literálok táblázatait tartalmazza.

Az eljárásorientált programozási nyelvek a programegységek futásidejű kezeléséhez, a hívási lánc implementálásához az úgynevezett **aktiváló rekord**ot használják. Ennek felépítése az alábbi:
- Dinamikus kapcsoló (mutató típusú mező, a hívó programegység aktiváló rekordját címzi).
- Statikus kapcsoló (mutató típusú mező, a tartalmazó programegység aktiváló rekordját címzi).
- Visszatérési cím (itt kell a programot folytatni a programegység szabályos befejezése esetén).
- Lokális változók.
- Formális paraméterek (alprogram esetén).
- Visszatérési érték (függvény esetén).

Egyszerű típusú lokális változók esetén egyszerű a tárterület lefoglalása. Összetett típusúaknál viszont nem: tömbök esetén helyet kell foglalni a tömbleíró információknak (dimenzió, indexek korlátjai, elemtípus, elemméret) is.

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.

Az **aktiváló rekordok** a veremben tárolódnak. A verem alján mindig a főprogram aktiváló rekordja van. Szabályos program befejezéskor a verem kiürül. Amikor meghívunk egy alprogramot vagy blokkot, akkor felépül hozzá az aktiváló rekord, és az a verem tetejére kerül. Szabályos befejeződéskor az aktiváló rekord törlődik

Taszkok esetén (egy processzoron) egy „kaktusz” verem épül föl. A szülőegység aktiváló rekordja elhelyeződik a verem tejére, és az általa meghívott nem taszk programegységek aktiváló rekordjai pedig fölé kerülnek. A testvértaszkok mindegyikéhez felépül egy-egy olyan verem, melynek az alján a szülőegység aktiváló rekordja van. Ezek a vermek egyidejűleg léteznek, és tartalmazzák az adott taszk által létrehozott hívási lánc aktiváló rekordjait. A szülőegység aktiváló rekordja csak akkor törölhető, ha minden testvértaszkjának verme kiürült.