Záróvizsga tételsor mérnökinformatikus hallgatóknak*

Palkovics Dénes,

2019

A záróvizsga tematikája és tartalma
A záróvizsgán kettő kérdésre kell válaszolni, egyre az általános kérdések közül, egyre pedig a specializációnak megfelelő kérdések közül

Tartalomjegyzék

*A Debreceni Egyetem mérnökinformatikus alapszakhoz

I. rész

Általános kérdések

1. Az informatika logikai alapjai

1.1. Az elsőrendű matematikai logikai nyelv.

Definíció (Elsőrendű nyelv). *Klasszikus elsőrendű nyelven az*

$$L^{(1)} = \langle LC, Var, Con, Term, Form \rangle$$

rendezett ötöst értjük, ahol

1. $LC = \{\neg, \supset, \wedge, \vee, \equiv, =, \forall, \exists, (,)\}$ *a nyelv logikai konstansainak halmaza*[1]
2. $Var = \{x_n | n = 0, 1, 2, \dots\}$ *a nyelv változóinak megszámlálhatóan végtelen halmaza*[2]
3. $Con = \bigcup_{n=0}^{\infty}(\mathcal{F}(n) \cup \mathcal{P}(n))$ *a nyelv nemlogikai konstansainak legfeljebb megszámlálhatóan végtelen halmaza*[3]
   a) $\mathcal{F}(0)$ *a névparaméterek (névkonstansok),*
   b) $\mathcal{F}(n)$ *az n argumentumú függvényjelek (műveleti jelek),*
   c) $\mathcal{P}(0)$ *a állításparaméterek (állításkonstansok),*
   d) $\mathcal{P}(n)$ *az n argumentumú predikátumparaméterek (predikátumkonstansok) halmaza.*
4. *Az* $LC, Var, \mathcal{F}(n), \mathcal{P}(n)$ *halmazok* $(n = 0, 1, 2, \dots)$ *páronként diszjunktak.*
5. *A nyelv terminusainak a halmazát, azaz a* $Term$ *halmazt az alábbi induktív definíció adja:*
   a) $Var \cup \mathcal{F}(0) \subseteq Term$
   b) *Ha* $f \in \mathcal{F}(n), (n = 1, 2, \dots)$,*és* $t_1, t_2, \dots, t_n \in Term$, *akkor* $f(t_1, t_2, \dots, t_n) \in Term$
6. *A nyelv formuláinak halmazát, azaz a* $Form$ *halmazt az alábbi induktív definíció adja meg:*
   a) $\mathcal{P} \subseteq Form$
   b) *Ha* $t_1, t_2 \in Term$, *akkor* $(t_1 = t_2) \in Form$
   c) *Ha* $P \in \mathcal{P}, (n = 1, 2, \dots)$, *és* $t_1, t_2, \dots, t_n \in Term$, *akkor* $P(t_1, t_2, \dots, t_n) \in Form$
   d) *Ha* $A \in Form$, *akkor* $\neg A \in Form$
   e) *Ha* $A, b \in Form$, *akkor* $(A \supset B), (A \wedge B), (A \vee B), (A \equiv B) \in Form$
   f) *Ha* $x \in Var, A \in Form$, *akkor* $\forall xA, \exists xA \in Form$

Megjegyzés. *Azokat a formulákat, amelyek a 6. a), b), c) szabályok által jönnek létre, atomi formuláknak vagy prímformuláknak nevezzük.*

[1] A logikai konstansok olyan nyelvi eszközök, amelyek jelentését a szemantikai szabályok (logikai kalkulusok esetén az axiómák) rögzítik. Egy adott logikai rendszer esetén a logikai konstansok rögzített jelentéssel (rögzített szemantikai értékkel)rendelkeznek, jelentésük (szemantikai értékük) minden interpretációban megegyezik. Egy adott logikai rendszer esetén a logikai konstansokat általában az adott logikai rendszer nyelvének LC halmaza tartalmazza.

[2] A köznyelvi mondatokban nevek helyett néha névmásokkal utalunk egyes individuumokra (objektumokra). A tudományos nyelvben gyakran kívánatos analóg kifejezési formák megadása. A szabatosság, az egyértelműség és a tömörség érdekében ilyenkor mesterséges névmásokat vezetnek be, amelyeket változóknak neveznek.

[3] A nemlogikai konstansok, más néven paraméterek olyan nyelvi eszközök, amelyek jelentését az interpretáció rögzíti. Egy adott logikai rendszer esetén a nemlogikai konstansok (a paraméterek) nem rendelkeznek rögzített jelentéssel (rögzített szemantikai értékkel), jelentésük (szemantikai értékük) interpretációról interpretációra változhat. Egy adott logikai rendszer esetén a nemlogikai konstansokat általában az adott logikai rendszer nyelvének Con halmaza tartalmazza.

1.2. A nyelv interpretációja, formulák igazságértéke az interpretációban adott változókiértékelés mellett.

Definíció (interpretáció (elsőrendű)). *Az $\langle U, \rho\rangle$ párt az $L^{(1)}$ nyelv egy interpretációjának nevezzük, ha*

1. $U \neq \emptyset$ azaz U nemüres halmaz

2. $Dom(\rho) = Con$ azaz a ρ a Con halmazon értelmezett függvény, amelyre teljesülnek a következők:

a) Ha $a \in F(0)$, akkor $\rho(a) \in U$

b) Ha $f \in \mathcal{F}(n)$ ahol $n \neq 0$, akkor $\rho(f)$ az $U^{(n)}$ halmazon értelmezett az U halmazba képező függvény ($\rho(f) : U^{(n)} \to U$)

c) Ha $p \in \mathcal{P}(0)$, akkor $\rho(p) \in 0, 1$

d) Ha $P \in \mathcal{P}(n)$ ahol $n \neq 0$, akkor $\rho(P) \subseteq U^{(n)}$

Definíció (értékelés (elsőrendű)). *Legyen $L^{(1)} = \langle LC, Var, Con, Term, Form\rangle$ egy elsőrendű nyelv, $\langle U, \rho\rangle$ pedig a nyelv egy interpretációja. Az $\langle U, \rho\rangle$ interpretációra támaszkodó ν értékelésen egy olyan függvényt értünk, amely teljesíti a következőket:*

- $Dom(\nu) = Var$
- $Hax \in Var$, *akkor* $\nu(x) \in U$

Definíció (értékelés (elsőrendű)). *Legyen $L^(1) = (LC, Var, Con, Term, Form)$ egy elsőrendű nyelv, $\langle U, \rho\rangle$ pedig a nyelv egy interpretációja, ν pedig az $\langle U, \rho\rangle$ interpretációra támaszkodó értékelés.*

1. Ha $a \in F(0)$, akkor $|a|_\nu^{\langle U,\rho\rangle} = \rho(a)$

2. Ha $x \in Var$, akkor $|x|_\nu^{\langle U,\rho\rangle} = \nu(x)$

3. Ha $f \in F(n)$,$(n = 1, 2, \dots)$ és $t_1, t_2, \dots, t_n \in Term$, akkor

$$|f(t_1, t_2, \dots, t_n)|_\nu^{\langle U,\rho\rangle} = \rho(f)(|t_1|_\nu^{\langle U,\rho\rangle}, |t_2|_\nu^{\langle U,\rho\rangle}, \dots, |t_n|_\nu^{\langle U,\rho\rangle})$$

4. Ha $p \in P(0)$, akkor $|p|_\nu^{\langle U,\rho\rangle} = \rho(p)$

5. Ha $t_1, t_2 \in Term$, akkor

$$|(t_1 = t_2)|_\nu^{\langle U,\rho\rangle} = \begin{cases} 1, & \text{ha } |t_1|_\nu^{\langle U,\rho\rangle} = |t_2|_\nu^{\langle U,\rho\rangle} \\ 0, & \text{egyébként.} \end{cases} \tag{1}$$

6. Ha $P \in P(n) aholn = 0, t1, \dots, tn \in Term$, akkor

$$|P(t_1, t_2, \dots, t_n)|_\nu^{\langle U,\rho\rangle} = \begin{cases} 1, & \text{ha } \left(|t_1|_\nu^{\langle U,\rho\rangle}, |t_2|_\nu^{\langle U,\rho\rangle}, \dots, |t_n|_\nu^{\langle U,\rho\rangle}\right) \in \rho(P) \\ 0, & \text{egyébként.} \end{cases} \tag{2}$$

7. Ha $A \in Form$, akkor $|\neg A|_\nu^{\langle U,\rho\rangle} = 1 - |A|_\nu^{\langle U,\rho\rangle}$.

8. Ha $A, B \in Form$, akkor

$$|(A \supset B)|_\nu^{\langle U,\rho\rangle} \begin{cases} 0, & \text{ha } |A|_\nu^{\langle U,\rho\rangle} = 1, \text{ és } |B|_\nu^{\langle U,\rho\rangle} = 0 \\ 1, & \text{egyébként.} \end{cases} \tag{3}$$

$$|(A \wedge B)|_\nu^{\langle U,\rho\rangle} \begin{cases} 1, & \text{ha } |A|_\nu^{\langle U,\rho\rangle} = 1, \text{ és } |B|_\nu^{\langle U,\rho\rangle} = 1 \\ 0, & \text{egyébként.} \end{cases} \tag{4}$$

$$|(A \vee B)|_\nu^{\langle U,\rho\rangle} \begin{cases} 0, & \text{ha } |A|_\nu^{\langle U,\rho\rangle} = 0, \text{ és } |B|_\nu^{\langle U,\rho\rangle} = 0 \\ 1, & \text{egyébként.} \end{cases} \tag{5}$$

$$|(A \equiv B)|_\nu^{\langle U,\rho\rangle} \begin{cases} 1, & \text{ha } |A|_\nu^{\langle U,\rho\rangle} = |B|_\nu^{\langle U,\rho\rangle} \\ 0, & \text{egyébként.} \end{cases} \tag{6}$$

9. Ha $A \in Form, x \in Var$, *akkor*

$$|(\forall_x A)|_\nu^{\langle U,\rho\rangle} = \begin{cases} 0, & \text{ha van olyan } u \in U, \text{ hogy} |A|_{\nu[x:u]}^{\langle U,\rho\rangle} = 0 \\ 1, & \text{egyébként.} \end{cases} \tag{7}$$

$$|(\exists_x A)|_\nu^{\langle U,\rho\rangle} = \begin{cases} 1, & \text{ha van olyan } u \in U, \text{ hogy} |A|_{\nu[x:u]}^{\langle U,\rho\rangle} = 1 \\ 0, & \text{egyébként.} \end{cases} \tag{8}$$

1.3. Logikai törvény, logikai következmény.

Definíció (modell). *Legyen* $L^{(1)} = (LC, Var, Con, Term, Form)$ *egy elsőrendű nyelv és* $\Gamma \subseteq Form$ *egy tetszőleges formulahalmaz. Az* (U, ρ, ν) *rendezett hármas elsőrendű modellje a* Γ *formulahalmaznak, ha*

- (U, ρ) *egy interpretációja az* $L^{(1)}$ *nyelvnek;*
- ν *egy* (U, ρ) *interpretációra támaszkodó értékelés;*
- *minden* $A \in \Gamma$ *esetén* $|A|_\nu^{\langle U,\rho\rangle} = 1$.

Definíció. *Legyen* $L^{(1)} = (LC, Var, Con, Term, Form)$ *egy elsőrendű nyelv és* $\Gamma \subseteq Form$ *egy tetszőleges formulahalmaz,* $A, B \in Form$ *egy tetszőleges formulák.*

- *Egy* Γ *formulahalmaz* kielégíthető, *ha van (elsőrendű) modellje;*
- *Egy* Γ *formulahalmaz* kielégíthetetlen, *ha nem kielégíthető, azaz nincs modellje;*
- *Az* A *formula* modellje *az* $\{A\}$ *egyelemű formulahalmaz modelljét értjük;*
- *Az* A *formula* kielégíthető, *ha* $\{A\}$ *formulahalmaz kielégíthető;*
- *Az* A *formula* kielégíthetetlen, *ha* $\{A\}$ *formulahalmaz kielégíthetetlen;*
- A Γ *formulahalmaznak* logikai következménye *az* A *formula, ha a* $\Gamma \cup \{\neq A\}$ *formulahalmaz kielégíthetetlen. Jelölés:* $\Gamma \models A$
- *Az* A *formulának* logikai következménye *a* B *formula, ha a* $\{A\} \models B$. *Jelölés:* $A \models B$
- *Az* A *formula* érvényes *(*logikai törvény*), ha* $\emptyset \models A$, *azaz ha az* A *formula* logikai következménye *az üres halmaznak. Másképpen, ha minden* $\langle U, \rho\rangle$ *interpretációjában, minden* ν *értékelés szempontjából* $|A|_\nu^{\langle U,\rho\rangle} = 1$ *Jelölés:* $\models A$
- *Az* A *és a* B *formula* logikailag ekvivalens, *ha* $A \models B$ *és* $B \models A$. *Jelölés:* $A \Leftrightarrow B$

1.4. Logikai ekvivalencia, normálformák.

Definíció (Logikai ekvivalencia). *lásd:a 1.3 fejezet definíciója.*

Definíció (elemi konjunkció). *Legyen* $L^{(0)} = (LC, Con, Form)$ *egy nulladrendű nyelv. Ha az* $A \in Form$ *formula literál vagy különböző alapú literálok konjunkciója, akkor A-t elemi konjunkciónak nevezzük.*

Definíció (elemi diszjunkció). *Legyen* $L^{(0)} = (LC, Con, Form)$ *egy nulladrendű nyelv. Ha az* $A \in Form$ *formula literál vagy különböző alapú literálok diszjunkciója, akkor A-t elemi diszjunkciónak nevezzük.*

Definíció (diszjunktív normálforma). *Egy elemi konjunkciót vagy elemi konjunkciók diszjunkcióját diszjunktív normálformának nevezzük.*

Definíció (konjunktív normálforma). *Egy elemi diszjunkciót vagy elemi diszjunkciók konjunkcióját konjunktív normálformának nevezzük.*

Definíció. *Legyen* $L^{(0)} = (LC, Con, Form)$ *egy nulladrendű nyelv és* $A \in Form$ *egy formula. Ekkor létezik olyan* $B \in Form$, *hogy*

- $A \Leftrightarrow B$
- B *diszjunktív vagy konjunktív normálformájú.*

1.5. Kalkulusok (Gentzen-kalkulus).

Logikai kalkulus Logikai kalkuluson olyan adott nyelv formuláihoz tartozó formális rendszert, szabályrendszert értünk, amely pusztán szintaktikailag, szemantika nélkül ad meg egy következményrelációt. A logikai kalkulus tehát egy axiómarendszer, amely magában a logikai tautológiákat állítja elő, adott formulákat ideiglenesen hozzávéve (premissza) pedig más formulákra (konklúzió) lehet jutni (következtetni) vele.

Gentzen-féle szekvenciakalkulus Ebben a kalkulusban nem formulákra vonatkoznak a szabályok és nem is formulák alkotják az axiómákat, hanem a formulák eddigi szerepét az ún. szekvencia töltik be. Szekvenciának nevezzük a

$$\Gamma \vdash \Delta$$

alakú jelsorozatokat, ahol Γ és Δ olyan rendezett jelsorozatok, amelyeknek minden tagja egy formula.

Definíció (axiómasémák). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv (a klasszikus állításlogika nyelve). A nulladrendű kalkulus (klasszikus állításkalkulus) axiómasémái (alapsémái):*

1. $A \supset (B \supset A)$

2. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$

3. $(\neg A \supset \neg B) \supset (B \supset A)$

Az axiómaséma szabályos behelyettesítésén olyan formulát értünk, amely az axiómasémából a benne szereplő betűk tetszőleges formulával való helyettesítése útján jön létre. A nulladrendű kalkulus (klasszikus állításkalkulus) axiómái az axiómasémák szabályos behelyettesítései.

Definíció (szintaktikai következmény). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv, $\Gamma \subseteq Form$ egy tetszőleges formulahalmaz. A Γ formulahalmaz szintaktikai következményeinek induktív definíciója:*

Bázis:

- *Ha $A \in \Gamma$, akkor $\Gamma \vdash A$*
- *Ha A axióma, akkor $\Gamma \vdash A$.*

Szabály (leválasztási szabály):

- *Ha $\Gamma \vdash B$, és $\Gamma \vdash (B \subset A)$, akkor $\Gamma \vdash A$.*

Definíció (szintaktikai következmény). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv és $A, B \in Form$ két tetszőleges formula. Az A formulának szintaktikai következménye a B formula, ha $\{A\} \vdash B$. Jelölés: $A \vdash B$*

Definíció (szekvencia). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv, $\Gamma \subseteq Form$ egy formulahalmaz és $A \in Form$ egy formula. Ha az A formula szintaktikai következménye a Γ formulahalmaznak, akkor a $\Gamma \vdash A$ jelsorozatot szekvenciának nevezzük.*

Definíció (levezethetőség). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv és $A \in Form$ egy tetszőleges formula. Az A formula levezethető, ha $\emptyset \vdash A$, azaz ha az A formula szintaktikai következménye az üres halmaznak. Jelölés: $\vdash A$*

Definíció (természetes levezetés szabályai). *Legyen $L^{(0)} = (LC, Con, Form)$ egy nulladrendű nyelv $\Gamma, \Delta \subseteq Form$ és $A, B, C \in Form$. A természetes levezetés által az $L^{(0)}$ nyelvben bizonyítható következményrelációk alábbiak:*

Bázis:

$$\frac{\omega}{\Gamma, A \vdash A} \tag{9}$$

Szabályok:

- *Struktúrális szabályok:*
  - *Bővítés* $\frac{\Gamma \vdash A}{\Gamma, B \vdash A}$
  - *Felcserélés* $\frac{\Gamma, B, C, \Delta \vdash A}{\Gamma, C, B, \Delta \vdash A}$
  - *Szűkítés* $\frac{\Gamma, B, B, \Delta \vdash A}{\Gamma, B, \Delta \vdash A}$

  - *Metszet* $\frac{\Gamma \vdash A \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$

- *Logikai szabályok:*
  - *Implikáció szabályai:*
    - *bevezető:* $\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}$
    - *alkalmazó:* $\frac{\Gamma \vdash A \Gamma \vdash A \supset B}{\Gamma \vdash B}$
  - *Negáció szabályai:*
    - *bevezető:*$\frac{\Gamma, A \vdash B \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}$
    - *alkalmazó:*$\frac{\Gamma \vdash \neg \neg A}{\Gamma \vdash A}$
  - *Konjunkció szabályai:*
    - *bevezető:*$\frac{\Gamma \vdash A \Gamma \vdash B}{\Gamma \vdash A \wedge B}$
    - *alkalmazó:*$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B, \vdash C}$
  - *Diszjunkció szabályai:*
    - *bevezető:*$\frac{\Gamma \vdash A}{\frac{\Gamma \vdash A \vee B \Gamma \vdash B}{\Gamma \vdash A \vee B}}$
    - *alkalmazó:*$\frac{\Gamma, A \vdash C \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}$
  - *(Materiális) ekvivalencia szabályai:*
    - *bevezető:*$\frac{\Gamma, A \vdash B \Gamma, B \vdash A}{\Gamma \vdash A \equiv B}$
    - *alkalmazó:*$\frac{\Gamma \vdash A \Gamma \vdash A \equiv B}{\frac{\Gamma \vdash B \Gamma \vdash B \Gamma \vdash A \equiv B}{\Gamma \vdash A}}$

2. Operációs rendszerek

2.1. Operációs rendszerek fogalma, felépítése, osztályozásuk.

Operációs rendszerek fogalma Egy program, amely közvetítő szerepet játszik a számítógép felhasználója és a számítógéphardver között. Az operációs rendszer feladata, hogy a felhasználónak egy olyan egyenértékű kiterjesztett vagy virtuális gépet nyújtson, amelyiket egyszerűbb programozni, mint a mögöttes hardvert

Operációs rendszerek felépítése Az operációs rendszerek alapvetően három részre bonthatók:

- a felhasználói felület (a shell, amely lehet egy grafikus felület, vagy egy szöveges)
- alacsony szintű segédprogramok
- kernel (mag), amely közvetlenül a hardverrel áll kapcsolatban.

Operációs rendszerek osztályozása

1. Az operációs rendszer alatti hardver "mérete" szerint:
   - mikroszámítógépek operációs rendszerei
   - kisszámítógépek, esetleg munkaállomások operációs rendszerei
   - nagygépek (Main Frame Computers, Super Computers) operációs rendszerei
2. A kapcsolattartás típusa szerint:
   - kötegelt feldolgozású operációs rendszerek vezérlőkártyás kapcsolattartással
   - interaktív operációs rendszerek.
3. cél szerint: általános felhasználású vagy céloperációs rendszer
4. a processzkezelés: single-tasking, multi-tasking
5. a felhasználók száma szerint: single, multi
6. CPU-idő kiosztása szerint: szekvenciális, megszakítás vezérelt, event-polling, time-sharing
7. a memóriakezelés megoldása szerint: valós és virtuális címzésű

2.2. Az operációs rendszerek jellemzése (komponensei és funkciói).

Operációs rendszerek komponensei:

Eszközkezelők (Device Driver) Felhasználók elől el fedik a perifériák különbségeit, egységes kezelői felületet kell biztosítani.

Megszakítás kezelés (Interrupt Handling) Alkalmas perifériák felől érkező kiszolgálási igények fogadására, megfelelő ellátására.

Rendszerhívás, válasz (System Call, Reply) az operációs rendszer magjának ki kell szolgálnia a felhasználói alkalmazások (programok) erőforrások iránti igényeit úgy, hogy azok lehetőleg észre se vegyék azt, hogy nem közvetlenül használják a perifériákat← programok által kiadott rendszerhívások, melyekre rendszermag válaszokat küldhet.

Erőforrás kezelés (Resource Management) Az egyes eszközök közös használatából származó konfliktusokat meg kell előznie, vagy bekövetkezésük esetén fel kell oldania.

Processzor ütemezés (CPU Scheduling) Az operációs rendszerek ütemező funkciójának a várakozó munkák között valamilyen stratégia alapján el kell osztani a processzor idejét, illetve vezérelnie kell a munkák közötti átkapcsolási folyamatot.

Memóriakezelés (Memory Management) Gazdálkodnia kell a memóriával, fel kell osztania azt a munkák között úgy, hogy azok egymást se zavarhassák, és az operációs renszerben se tegyenek kárt.

Állomány- és lemezkezelés (File and Disk Management) Rendet kell tartania a hosszabb távra megőrzendő állományok között.

Felhasználói felület (User Interface) A parancsnyelveket feldolgozó monito utódja, fejlettebb változata, melynek segítségével a felhasználó közölni tudja a rendszermaggal kívánságait, illetve annk állapotáról információt szerezhet.

Operációs rendszerek funkciói:

Folyamatkezelés A folyamat egy végrehajtás alatt álló program. Hogy feladatát ellássa erőforrásokra van szüksége (processzor idő, memória, állományok I/O berendezések). Az operációs rendszer feladata:

- Folyamatok létrehozása és törlése
- Folyamatok felfüggesztése és újraindítása
- Eszközök biztosítása a folyamatok kommunikációjához és szinkronizációjához.

Memória (főtár) kezelés Bájtokból álló tömbnek tekinthető, amelyet a CPU és az I/O közösen használ. Tartalma törlődik rendszerkikapcsoláskor és rendszerhibáknál. Az operációs rendszer feladata:

- Nyilvántartani, hogy az operatív memória melyik részét ki (mi) használja.
- Eldönteni melyik folyamatot kell betölteni, ha memória felszabadul.
- Szükség szerint allokálni és felszabadítani a memória területeket a szükségleteknek megfelelően.

Másodlagos tárkezelés Nem törlődik, és elég nagy hogy minden programot tároljon. A merevlemez a legelterjedtebb formája. Az operációs rendszer feladata:

- Szabadhely kezelés.
- Tárhozzárendelés.
- Lemez elosztás.

I/O rendszerkezelés

- Puffer rendszer.
- Általános készülék meghajtó (device driver) interface.
- Speciális készülék meghajtó programok.

Fájlkezelés Egy fájl kapcsolódó információk együttese, amelyet a létrehozója definiál. Általában program és adatfájlokról beszélünk. Az operációs rendszer feladata:

- Fájlok és könyvtárak létrehozás és törlése.
- Fájlokkal és könyvtárakkal történő alapmanipuláció.
- Fájlok leképezése a másodlagos tárra, valamilyen nem törlődő, stabil adathordozóra.

Védelmi rendszer Olyan mechanizmus, mely az erőforrásokhoz való hozzá férést felügyeli. Az operációs rendszer feladata:

- Különbséget tenni jogos (authorizált) és jogtalan használat között.
- Specifikálni az alkalmazandó kontrolt.
- Korlátozó eszközöket szolgáltatni.

Hálózat elérés támogatása Az elosztott rendszer processzorok adat és vezérlő vonallal összekapcsolt együttese, ahol a memória és az óra nem közös. Adat- és vezérlővonal segítségével történik a kommunikáció. Az elosztott rendszer a felhasználóknak különböző osztott erőforrások elérését teszi lehetővé, mely lehetővé teszi:

- a számítások felgyorsítását,
- a jobb adatelérhetőséget,
- a nagyobb megbízhatóságot.

Parancs interpreter alrendszer Az operációs rendszernek sok parancsot vezérlő utasítás formájában lehet megadni. Vezérlő utasítások minden területhez tartoznak (folyamatok, I/O kezelés...). Az operációs rendszernek azt a programját, amelyik a vezérlő utasítást beolvassa és interpretálja a rendszertől függően más és más módon nevezhetik:

- Vezérlő kártya interpreter.
- Parancs sor interpreter (command line).
- Héj (burok, shell)

2.3. A rendszeradminisztráció, fejlesztői és alkalmazói támogatás eszközei.

Rendszeradminisztráció Magának az operációs rendszernek a működtetésével kapcsolatos funkciók. Ezek közvetlenül semmire sem használhatók, csak a hardverlehetőségek kibővítését célozzák, illetve a hardver kezelését teszik kényelmesebbé. A rendszeradminisztráción belül a következő *összetett funkciókat* jelölhetjük ki:

1. **processzorütemezés:** a CPU-idő szétosztása a rendszer- és a felhasználói feladatok (taszkok, folyamatok) között;
2. **megszakításkezelés:** a hardver-szoftver megszakításkérések elemzése, állapotmentés, a kezelőprogram hívása;
3. **szinkronizálás:** az események és az erőforrásigények várakozási sorokba állítása;
4. **folyamatvezérlés:** a programok indítása és a programok közötti kapcsolatok szervezése;
5. **tárkezelés:** a főtár, – mint kiemelten kezelt erőforrás, – elosztása;
6. **perifériakezelés:** a bemeneti/kimeneti (B/K ill. I/O) igények sorba állítása és kielégítése;
7. **adatkezelés:** az adatállományokon végzett műveletek segítése (létrehozás, nyitás, zárás, írás, olvasás stb.);
8. **működés-nyilvántartás:** a hardver hibastatisztika vezetése és a számlaadatok feljegyzése;
9. **operátori interfész:** a kapcsolattartás az üzemeltetővel.

A konkrét operációs rendszerek a funkciókat másképpen oszthatják fel. Így például az IBM OS operációs rendszerek változataiban négy fő funkciót szoktak megkülönböztetni:

1. a munkakezelést,
2. a taszkkezelést,
3. az adatkezelést és
4. a rendszerstatisztikát.

A rendszeradminisztrációs funkciókat a **rendszermag** valósítja meg, amelynek a szolgáltatásait a már említett rendszerhívásokkal érhetjük el.

Programfejlesztési támogatás fő funkciói:

1. **rendszerhívások:** a programokból alacsony szintű operációsrendszeri funkciók aktivizálására,
2. **szövegszerkesztők:** a programok és dokumentációk írására,
3. **programnyelvi eszközök:** fordítóprogramok és interpreterek (értelmezők) a nyelvek fordítására vagy értelmezésére,
4. **szerkesztő- és betöltő-programok:** a programmodulok összefűzésére illetve tárba töltésére (végcímzés),
5. **programkönyvtári funkciók:** a különböző programkönyvtárak használatára,
6. **nyomkövetési rendszer:** a programok belövésére.

Alkalmazói támogatás Az alkalmazói támogatás funkciói a számítógépes rendszer több szintjén valósulnak meg, és az alábbi fő funkciókra bonthatók:

1. **operátori parancsnyelvi rendszer:** a számítógép géptermi üzemvitelének támogatására;
2. **munkavezérlő parancsnyelvi rendszer:** a számítógép alkalmazói szintű igénybevételének megfogalmazására;
3. **rendszerszolgáltatások:** az operációs rendszer magjával közvetlenül meg nem oldható rendszerfeladatokra;
4. **segéd-programkészlet:** rutinfeladatok megoldására;
5. **alkalmazói programkészlet:** az alkalmazásfüggő feladatok megoldására

3. Magas szintű Programozási nyelvek

3.1. Adattípus, konstans, változó, kifejezés.

Az adatabsztrakció első megjelenési formája az adattípus a programozási nyelvekben. Az adattípus maga egy absztrakt programozási eszköz, amely mindig más, konkrét programozási eszköz egy komponenseként jelenik meg. Az adattípusnak neve van, ami egy azonosító. 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
2. műveletek
3. reprezentáció

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. Az adattípushoz hozzátartoznak azok a műveletek, amelyeket a tartomány elemein végre tudunk hajtani. 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. A saját típus definiálási lehetőség az adatabsztrakciónak egy magasabb szintjét jelenti, segítségével a valós világ egyedeinek tulajdonságait jobban tudjuk modellezni. A saját típus definiálása általában szorosan kötődik az absztrakt adatszerkezetekhez. Saját típust úgy tudunk létrehozni, hogy megadjuk a tartományát, a műveleteit és a reprezentációját. Szokásos, hogy saját típust a beépített és a már korábban definiált saját típusok segítségével adjuk meg. Általános, hogy a reprezentáció megadásánál így járunk el. Csak nagyon kevés nyelvben lehet saját reprezentációt megadni (ilyen az Ada). Kérdés, hogy egy nyelvben lehet-e a saját típushoz saját műveleteket és saját operátorokat megadni. Van, ahol igen, de az is lehetséges, hogy a műveleteket alprogramok (l. 5.1. alfejezet) realizálják. A tartomány megadásánál is alkalmazható a visszavezetés technikája, de van olyan lehetőség is, hogy explicit módon adjuk meg az elemeket. 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 leszőkítem annak tartományát, változatlanul hagyva műveleteit és reprezentációját. Az alaptípus és az altípus tehát nem különböző típusok. Az *adattípusoknak* két nagy csoportjuk van:

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.

strukturált vagy összetett adattípus tartományának elemei maguk is valamilyen típussal rendelkeznek. Az elemek egy-egy értékcsoportot képviselnek, nem atomiak, az értékcsoport elemeihez külön-külön is hozzáférhetünk. Általában valamilyen absztrakt adatszerkezet programnyelvi megfelelői. Literálként általában nem jelenhetnek meg, egy konkrét értékcsoportot explicit módon kell megadni.

3.1.1. Egyszerű típusok

Minden nyelvben létezik az egész típus, sőt általában egész típusok. Ezek belső ábrázolása fixpontos. Az egyes egész típusok az ábrázoláshoz szükséges bájtok számában térnek el és nyilván ez határozza meg a tartományukat is. 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 hajthatók végre. A karakteres típus tartományának elemei karakterek, a karakterlánc vagy sztring típuséi pedig karaktersorozatok. Ábrázolásuk karakteres (karakterenként egy vagy két bájt, az alkalmazott kódtáblától függően), műveleteik a szöveges és hasonlító műveletek.

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 felsorolásos típus. A felsorolásos típust saját típusként kell létrehozni. A típus definiálása úgy történik, hogy megadjuk a tartomány elemeit. Ezek azonosítók lehetnek. Az elemekre alkalmazhatók a hasonlító műveletek.

Egyes nyelvek értelmezik az egyszerű típusok egy speciális csoportját, a sorszámozott típust. Ebbe a csoportba tartoznak általában az egész, karakteres, logikai és felsorolásos típusok. A sorszámozott típus tartományának elemei

listát (mint absztrakt adatszerkezetet) alkotnak, azaz van első és utolsó elem, minden elemnek van megelőzője (kivéve az elsőt) és minden elemnek van rákövetkezője (kivéve az utolsót). Tehát az elemek között egyértelmű sorrend értelmezett. A tartomány elemeihez kölcsönösen egyértelműen hozzá vannak rendelve a 0, 1, 2, ... sorszámok. Ez alól kivételt képeznek az egész típusok, ahol a tartomány minden eleméhez önmaga, mint sorszám van hozzárendelve. Egy sorszámozott típus esetén mindig értelmezhetők a következő műveletek:

- ha adott egy érték, meg kell tudni mondani a sorszámát, és viszont
- bármely értékhez meg kell tudni mondani a megelőzőét és a rákövetkezőjét

A sorszámozott típus az egész típus egyfajta általánosításának tekinthető. Egy sorszámozott típus altípusaként lehet származtatni az intervallum típust.

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 típusú programozási eszköz értéke tehát egy tárbeli cím, így azt mondhatjuk, hogy az adott eszköz a tár adott területét címzi, az adott tárterületre „mutat". A mutató típus egyik legfontosabb művelete a megcímzett tárterületen elhelyezkedő érték elérése. A mutató típus tartományának van egy speciális eleme, amely nem valódi tárcím. Tehát ezzel az értékkel rendelkező mutató típusú programozási eszköz „nem mutat sehova". A nyelvek ezt az értéket általában beépített nevesített konstanssal kezelik. A mutató típus alapvető szerepet játszik az absztrakt adatszerkezetek szétszórt reprezentációját kezelő implementációknál.

3.1.2. Összetett típusok

Az eljárás orientált nyelvek két legfontosabb összetett típusa a tömb (melyet minden nyelv ismer) és a rekord (egyes nyelvek, pl. a FORTRAN nem ismerik).

A tömb típus 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:

- dimenzióinak száma
- indexkészletének típusa és tartománya
- elemeinek a típusa

Egyes nyelvek (pl. a C) nem ismerik a többdimenziós tömböket. Ezek a nyelvek a többdimenziós tömböket úgy képzelik el, mint olyan egydimenziós tömbök, amelyek elemei egydimenziós tömbök. Többdimenziós tömbök reprezentációja lehet sor- vagy oszlop-folytonos. Ez általában implementációfüggő, a sorfolytonos a gyakoribb. Ha van egy tömb típusú programozási eszközünk, akkor a nevével az összes elemre együtt, mint egy értékcsoportra tudunk hivatkozni (az elemek sorrendjét a reprezentáció határozza meg). Az értékcsoport egyes elemeire a programozási eszköz neve után megadott indexek segítségével hivatkozunk. Az indexek a nyelvek egy részében szögletes, másik részében kerek zárójelek között állnak. Egyes nyelvek (pl. COBOL, PL/Ő) megengedik azt is, hogy a tömb egy adott dimenziójának összes elemét (pl. egy kétdimenziós tömb egy sorát) együtt hivatkozhassuk.

A nyelveknek a tömb típussal kapcsolatban a következő kérdéseket kell megválaszolniuk:

- Milyen típusúak lehetnek az elemek?
- Milyen típusú lehet az index?
- Amikor egy tömb típust definiálunk, hogyan kell megadni az indextartományt?
- Hogyan lehet megadni az alsó és a felső határt, illetve a darabszámot?

A tömb típus alapvető szerepet játszik az absztrakt adatszerkezetek folytonos ábrázolását megvalósító implementációknál.

A rekord típus absztrakt adatszerkezet megjelenése típus szinten. A rekord típus minden esetben heterogén, a tartományának 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. A különböző rekord típusok mezőinek neve megegyezhet.

A nyelvek egy részében (pl. C) a rekord típus statikus, tehát a mezők száma minden értékcsoportban azonos. Más nyelvek esetén (pl. Ada) van egy olyan mezőegyüttes, amely minden értékcsoportban szerepel (a rekord fix része), és van egy olyan mezőegyüttes, amelynek mezői közül az értékcsoportokban csak bizonyosak szerepelnek (a rekord változó része). Egy külön nyelvi eszköz (a diszkriminátor) szolgál annak megadására, hogy az adott konkrét esetben a változó rész mezői közül melyik jelenjen meg.Az ősnyelvek (pl. PL/Ő, COBOL) többszintű rekord típussal dogoznak. Ez azt jelenti, hogy egy mező felosztható újabb mezőkre, tetszőleges mélységig, és típus csak a legalsó szintű mezőkhöz rendelhető, de az csak egyszerű típus lehet. A későbbi nyelvek (pl. Pascal, C, Ada) rekord típusa egyszintű, azaz nincsenek almezők, viszont a mezők típusa összetett is lehet.

Egy rekord típusú programozási eszköz esetén az eszköz nevével az értékcsoport összes mezőjére hivatkozunk egyszerre (a megadás sorrendjében). Az egyes mezőkre külön minősített névvel tudunk hivatkozni, ennek alakja:

`eszköznév.mezőnév`

Az eszköz nevével történő minősítésre azért van szükség, mert a mezők nevei nem szükségszerűen egyediek. A rekord típus alapvető szerepet játszik az input-outputnál.

3.1.3. Literálok vagy konstansok

A literál olyan programozási eszköz, amelynek segítségével fix, explicit értékek építhetők be a program szövegébe. A literáloknak két komponensük van: típus és érték. A literál mindig önmagát definiálja. A literál felírási módja (mint speciális karaktersorozat) meghatározza mind a típust, mind az értéket. A nyelveknek saját literál rendszerük van.

Nevesített konstans Ez már konkrét eszköz, melynek három komponense van: név, típus, érték. Jelentősége: programozás technikai eszköz. A programozás szövegében a nevével jelenik meg, de az értéket jelenti. Azon programozási eszközökhöz, melyeknek van neve (név komponense) kötődik a deklaráció (a nyelvekben speciális utasítások állnak rendelkezésre). Mindhárom komponense a deklarációnál dől el (ott kell megadni, csak ott lehet megadni). Vannak olyan literálok, melyeknek nincs szemantikai értékük, ezeket, ha nevesítjük, beszélő névvel láthatjuk el. Technikai problémákat egyszerűsít, ha a program szövegében meg akarjuk változtatni ezt a névvel ellátott értéket, akkor nem kell annak valamennyi előfordulását megkeresni és átírni, hanem elegendő egy helyen, a deklarációs utasításban végrehajtani a módosítást.

3.1.4. Változó

A változó olyan programozási eszköz, amelynek négy komponense van:

1. név
2. attribútumok
3. cím
4. érték

A *név* egy azonosító. A program szövegében a változó mindig a nevével jelenik meg, az viszont bármely komponenst jelentheti. Szemlélhetjük úgy a dolgokat, hogy a másik három komponenst a névhez rendeljük hozzá.
Az *attribútumok* olyan jellemzők, amelyek a változó futás közbeni viselkedését határozzák meg. Az eljárás-orientált nyelvekben (általában a típusos nyelvekben) a legfőbb attribútum a típus, amely a változó által felvehető értékek körét határolja be. Változóhoz attribútumok deklaráció segítségével rendelődnek. A deklarációnak különböző fajtáit ismerjük.

Explicit deklaráció A programozó végzi explicit deklarációs utasítás segítségével. A változó teljes nevéhez kell az attribútumokat megadni. A nyelvek általában megengedik, hogy egyszerre több változónévhez ugyanazokat az attribútumokat rendeljük hozzá.

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.

Automatikus deklaráció A fordítóprogram rendel attribútumot azokhoz a változókhoz, amelyek nincsenek explicit módon deklarálva, és kezdőbetűjükhöz nincs attribútum rendelve egy implicit deklarációs utasításban. Az attribútum hozzárendelése a név valamelyik karaktere (gyakran az első) alapján történik:

Az eljárás-orientált nyelvek mindegyike ismeri az explicit deklarációt, és egyesek csak azt ismerik. Az utóbbiak általánosságban azt mondják, hogy minden névvel rendelkező programozói eszközt explicit módon deklarálni kell. 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. Egy változóhoz cím rendelhető az alábbi módokon:

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.

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 működését. A címkomponens a futás során változhat, sőt vannak olyan időintervallumok, amikor a változónak nincs is címkomponense.

A programozó által vezérelt tárkiosztás A változóhoz a programozó rendel címkomponenst futási időben. A címkomponens változhat, és az is elképzelhető, hogy bizonyos időintervallumokban nincs is címkomponens. Három alapesete van:

1. A programozó abszolút címet rendel a változóhoz, konkrétan megadja, hogy hol helyezkedjen el.
2. Egy már korábban a tárban elhelyezett programozási eszköz címéhez képest mondja meg, hogy hol legyen a változó elhelyezve, vagyis relatív címet ad meg. Lehet, hogy a programozó az abszolút címet nem is ismeri.
3. A programozó csak azt adja meg, hogy mely időpillanattól kezdve legyen az adott változónak címkomponense, az elhelyezést a futtató rendszer végzi. A programozó nem ismeri az abszolút címet.

Mindhárom esetben lennie kell olyan eszköznek, amivel a programozó megszüntetheti a címkomponenst.

A programozási nyelvek általában többféle címhozzárendelést ismernek, az eljárás-orientált nyelveknél általános a dinamikus tárkiosztás. A változók címkomponensével kapcsolatos a többszörös tárhivatkozás esete. Erről akkor beszélünk, amikor két különböző névvel, esetleg különböző attribútumokkal rendelkező változónak a futási idő egy adott pillanatában azonos a címkomponense ésígy értelemszerűen az értékkomponense is. Így ha az egyik változó értékét módosítjuk, akkor a másiké is megváltozik. A korai nyelvekben (pl. FORTRAN, PL/Ő) erre explicit nyelvi eszközök álltak rendelkezésre, mert bizonyos problémák megoldása csak így volt lehetséges. A szituáció viszont előidézhető (akár véletlenül is) más nyelvekben is, és ez nem biztonságos kódhoz vezethet.

A változó értékkomponense mindig a címen elhelyezett bitkombinációként jelenik meg. A bitkombináció felépítését a típus által meghatározott reprezentáció dönti el.

Egy változó értékkomponensének meghatározására a következő lehetőségek állnak rendelkezésünkre:

Értékadó utasítás Az eljárás-orientált nyelvek leggyakoribb utasítása, az algoritmusok kódolásánál alapvető.

3.1.5. Kifejezés

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 következő összetevőkből 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. Minden nyelv megengedi a redundáns zárójelek alkalmazását.

Attól függően, hogy egy operátor hány operandussal végzi a műveletet, beszélünk *egyoperandusú* (unáris), *kétoperandusú* (bináris), vagy *háromoperandusú* (ternáris) operátorokról. A kifejezésnek 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 *)

Az egyoperandusú operátorok általában az operandus előtt, ritkán mögötte állnak. A háromoperandusú operátorok általában infixek.

Kifejezés kiértékelése 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 a következő lehet:

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

Az infix alak nem egyértelmű. Az ilyen alakot használó nyelvekben az operátorok nem azonos erősségűek. Az ilyen nyelvek operátoraikat egy precedencia táblázatban adják meg. A precedencia táblázat sorokból áll, az egy sorban megadott operátorok azonos erősségűek (prioritásúak,precedenciájúak), az előrébb szereplők erősebbek. Minden sorban meg van adva még a kötési irány, amely megmondja, hogy az adott sorban szereplő operátorokat milyen sorrendben kell kiértékelni, ha azok egymás mellett állnak egy kifejezésben. A kötési irány lehet balról jobbra, vagy jobbról balra.

A kifejezés típusának meghatározásánál kétféle elvet követnek a nyelvek. Vannak a típus-egyenértékűséget és vannak a típuskényszerítést vallók. A típus-egyenértékűséget valló nyelvek azt mondják, hogy egy kifejezésben egy kétoperandusú vagy háromoperandusú operátornak csak azonos típusú operandusai lehetnek. Ilyenkor nincs konverzió, az eredmény típusa vagy az operandusok közös típusa, vagy azt az operátor dönti el (például hasonlító műveletek esetén az eredmény logikai típusú lesz). A különböző nyelvek szerint két programozási eszköz típusa azonos, ha azoknál fönnáll a:

deklaráció egyenértékűség az adott eszközöket azonos deklarációs utasításban, együtt, azonos típusnévvel deklaráltuk.

név egyenértékűség az adott eszközöket azonos típusnévvel deklaráltuk

struktúra egyenértékűség a két eszköz összetett típusú és a két típus szerkezete megegyezik.

A típuskényszerítés elvét valló nyelvek esetén különböző típusú operandusai lehetnek az operátornak. A műveletek viszont csak az azonos belső ábrázolású operandusok között végezhetők el, tehát különböző típusú operandusok esetén konverzió van. Ilyen esetben a nyelv definiálja, hogy egy adott operátor esetén egyrészt milyen típuskombinációk megengedettek, másrészt, hogy mi lesz a művelet eredményének a típusa. A kifejezés kiértékelésénél minden művelet elvégzése után előől az adott részkifejezés típusa és az utoljára végrehajtott műveletnél pedig a kifejezés típusa. Egyes nyelvek (pl. Pascal, C) a numerikus típusoknál megengedik a típuskényszerítés egy speciális fajtáját még akkor is, ha egyébként a típus-egyenértékűséget vallják. Ezeknél a nyelveknél beszélünk a bővítés és szűkítés esetéről. A bővítés olyan típuskényszerítés, amikor a konvertálandó típus tartományának minden eleme egyben eleme a céltípus tartományának is (pl. egész $\rightarrow$ valós). Ekkor a konverzió minden további nélkül, értékvesztés nélkül végrehajtható. A szűkítés ennek a fordítottja (pl. valós $\rightarrow$ egész), ekkor a konverziónál értékcsonkítás, esetleg kerekítés történik. A nyelvek közül az ADA-ban semmiféle típuskeveredés nem lehet, a PL/I viszont a teljes konverzió híve. A *konstans kifejezés* olyan kifejezés, melynek értéke fordítási időben előől, kiértékelését a fordító végzi. Operandusai literálok és nevesített konstansok lehetnek.

3.2. Paraméterkiértékelés, paraméterátadás.

3.2.1. 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. A 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, ezeket rendeljük a formális paraméterlistához.

A formális és aktuális paraméterek egymáshoz rendelése történhet *sorrendi kötés* vagy *név szerinti kötés* szerint. **Sorrendi kötés** esetén a formális paraméterekhez a felsorolás sorrendjében rendelődnek hozzá az aktuális paraméterek. Ezt minden nyelv ismeri, általában ez az alapértelmezés. **Név szerinti kötés** esetén az aktuális paraméter listában határozhatjuk meg az egymáshoz rendelést, a formális paraméter nevét és mellette valamilyen szintaktikával az aktuális paramétert megadva. Ilyenkor lényegtelen a formális paraméterek sorrendje. Néhány nyelv ismeri. Alkalmazható a sorrendi és név szerinti kötés kombinációja együtt is, az aktuális paraméter lista elején sorrendi kötés, utána név szerinti kötés van.

Ha a formális paraméterek száma fix, a formális paraméter lista adott számú paramétert tartalmaz. Ekkor az aktuális paraméterek számának meg kell egyeznie a formális paraméterek számával, vagy lehet kevesebb, mint a formális

paraméterek száma. Ez csak érték szerinti paraméterátadási mód esetén lehetséges. Azon formális paraméterekhez, amelyekhez nem tartozik aktuális paraméter, a formális paraméter listában alapértelmezett módon rendelődik érték. Ha a formális paraméterek száma tetszőleges, az aktuális paraméterek száma is tetszőleges. Létezik olyan megoldás is, hogy a paraméterek számára van alsó korlát.

A nyelvek egyik része a *típusegyenértékűséget* vallja, ekkor az aktuális paraméter típusának azonosnak kell lennie a formális paraméter típusával. A nyelvek másik része a *típuskényszerítés* alapján azt mondja, hogy az aktuális paraméter típusának konvertálhatónak kell lennie a formális paraméter típusára.

3.2.2. 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. Kérdés, hogy melyik irányban és milyen információ mozog. A nyelvek érték szerinti, cím szerinti, eredmény szerinti, érték-eredmény szerinti, név szerinti és szöveg szerinti paraméterátadási módokat ismernek.

Érték szerinti 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 paraméterátadás esetén 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, átnyúl a hívó területre. Időben gyors, mert nincs értékmásolás, de veszélyes lehet, mert a hívott hozzáfér a hívó területén lévő információkhoz. Az aktuális paraméter változó lehet.

Eredmény szerinti paraméterátadás 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 csak működésének befejeztekor másolja át a formális paraméter értékét erre a címre. A kommunikáció egyirányú, a hívottól a hívó felé irányul. Van értékmásolás. Az aktuális paraméter változó lehet.

Érték-eredmény szerinti paraméterátadás esetén 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 paraméterátadás esetén az aktuális paraméter egy az adott szövegkörnyezetben értelmezhető tetszőleges szimbólumsorozat lehet. A paraméterkiértékelésnél 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 paraméterátadás a név szerintinek egy változata, annyiban különbözik tőle, hogy a hívás után az alprogram elkezd működni, az aktuális paraméter értelmező szövegkörnyezetének rögzítése, a formális paraméter csak akkor íródik felül, 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, Java), a formális paraméter listában

explicit módon meg kell adni a paraméterátadási módot (pl. Ada), az aktuális és formális paraméter típusa együttesen dönti el, a formális paraméter típusa dönti el. Az alprogramok formális paramétereit három csoportra oszthatjuk:

1. **Input paraméterek** ezekkel az alprogram kap információt a hívótól (pl. érték szerinti).
2. **Output paraméterek** a hívott alprogram ad információt a hívónak (pl. eredmény szerinti).
3. **Input-output paraméterek** az információ mindkét irányba mozog (pl. érték-eredmény).

3.3. Hatáskör, névterek, élettartam.

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 név a programegység lokális neve. A nem a programegységben deklarált, de ott hivatkozott név a szabad név. Azt a tevékenységet, mikor egy név hatáskörét megállapítjuk, hatáskörkezelésnek hívjuk. Kétféle hatáskörkezelést ismerünk, a statikus és a dinamikus hatáskörkezelést.

3.3.1. Statikus hatáskörkezelés

A statikus hatáskörkezelés fordítási időben történik, a fordítóprogram végzi. Alapja a programszöveg programegység szerkezete. Ha a fordító egy programegységben 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, amíg meg nem találja lokális névként, vagy el nem jut a legkülső szintre. Ha kiért a legkülső szintre, akkor vagy a mivel a programozónak kellett volna deklarálnia a nevet, ez fordítási hiba, vagy mivel ismeri az automatikus deklarációt a nyelv, a fordító hozzárendeli a névhez az attribútumokat. A név ilyenkor 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. Egy programegység a lokális neveit bezárja a külvilág elől. Azt a nevet, amely egy adott programegységben nem lokális név, de onnan látható, globális névnek hívjuk. A globális név, lokálisnév relatív fogalmak. Ugyanaz a név az egyik programegység szempontjából lokális, egy másikban globális, egy harmadikban pedig nem is látszik.

3.3.2. Dinamikus hatáskörkezelés

A dinamikus hatáskörkezelés 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. 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 eszköz látszik, nincs „lyuk a hatáskörben” szituáció.

Statikus hatáskörkezelés esetén a programban szereplő összes név hatásköre a forrásszöveg alapján egyértelműen megállapítható. Dinamikus hatáskörkezelésnél viszont a hatáskör futási időben változhat és más-más futásnál más-más lehet.

Az eljárásorientált nyelvek statikus hatáskörkezelést valósítanak 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. A kulcsszavak, mint nevek a program bármely pontjáról láthatók. A standard azonosítók, mint nevek azon programegységekből láthatók, ahol nem deklaráltuk újra őket. A globális változók az eljárásorientált nyelvekben a programegységek közötti kommunikációt szolgálják.

Névtér tulajdonképpen egy csoport azon azonosítóknak (változók, konstansok (Math.PI, Math.E), függvény nevek (Math.Abs) stb) amik létezhetnek már egy másik fájlban, dokumentumban. Namespacek segítségével egyértelműen be tudjuk azonosítani, hogy melyik dokumentumban definiált azonosítót szeretnénk használni.

3.4. Fordítási egységek, kivételkezelés.

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. A C# fordítási egysége a névtér, ami a C forrásállományának felel meg, és hatásköri egység is.

3.4.1. 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, ennek mintájára 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.

A kivételeknek általában van neve (egy kapcsolódó sztring, amely gyakran az eseményhez kapcsolódó üzenet szerepét játssza) és kódja (ami általában egy egész szám). A kivételkezelés a PL/I-ben jelenik meg és az Ada is rendelkezik vele. A két nyelv kétfajta kivételkezelési filozófiát vall. A PL/I azt mondja, hogy ha egy program futása folyamán bekövetkezik egy kivétel, akkor az azért van, mert a program által realizált algoritmust nem készítettük föl az adott esemény kezelésére, olyan szituáció következett be, amelyre speciális módon kell reagálni. Ekkor keressük meg az esemény bekövetkeztének az okát, szüntessük meg a speciális szituációt és térjünk vissza a program normál működéséhez, folytassuk a programot ott, ahol a kivétel kiváltódott. Az Ada szerint viszont, ha bekövetkezik a speciális szituáció, akkor hagyjuk ott az eredeti tevékenységet, végezzünk olyan tevékenységet, ami adekvát a bekövetkezett eseménnyel és ne térjünk vissza oda, ahol a kivétel kiváltódott. A kivételkezelési eszközrendszerrel kapcsolatban felmerülnek kérdések:

- Milyen beépített kivételek vannak?
- Definiálhatunk-e saját kivételt?
- Mik a kivételkezelés hatásköri szabályai?
- Hogyan folytatódik a futás a kivételkezelés után?
- Mi történik a kivételkezelőben történt kivétel esetén?
- Van-e beépített kivételkezelő, illetve általános és parametrizált kivételkezelő?

Sem a PL/I-ben, sem az Adában nincs parametrizált és beépített kivételkezelő. Az Ada beépített kivételei általában eseménycsoportot neveznek meg. Alaphelyzetben minden kivétel figyelése engedélyezett, de egyes események figyelése (bizonyos ellenőrzések) letiltható. Saját kivétel az EXCEPTION attribútummal deklarálható. Kivételkezelő minden programegység törzsének végén, közvetlenül a záró END előtt helyezhető el, ebben WHEN-ágból tetszőleges számú megadható, de legalább egy kötelező. WHEN OTHERS ág viszont legfeljebb egyszer szerepelhet, és utolsóként kell megadni. Ez a nem nevesített kivételek kezelésére való (általános kivételkezelés). A kivételkezelő a teljes programegységben, továbbá az abból meghívott programegységekben látszik, ha azokban nem szerepel saját kivételkezelő. Tehát a kivételkezelő hatásköre az Adában dinamikus, hívási láncon öröklődik. Bármely kivételt explicit módon kiváltani a RAISE kivételnév; utasítással lehet. Programozói kivétel kiváltása csak így lehetséges.

Ha egy programegységben kiváltódik egy kivétel, akkor a futtató rendszer megvizsgálja, hogy az adott kivétel figyelése le van-e tiltva. Ha igen, akkor a program fut tovább, különben a programegység befejezi működését. Ezek után a futtató rendszer megnézi, hogy az adott programegységen belül van-e kivételkezelő. Ha van, akkor megnézi, hogy annak van-e olyan WHEN-ága, amelyben szerepel az adott kivétel neve. Ha van ilyen ág, akkor végrehajtja az ott megadott utasításokat. Ha ezen utasítások között szerepel a GOTO-utasítás, akkor a megadott címkén folytatódik a program. Ha nincs GOTO, akkor úgy folytatódik a program futása, mintha a programegység szabályosan fejeződött volna be. Ha a kivétel nincs nevesítve, megnézi, hogy van-e WHEN OTHERS ág. Ha van, akkor az ott megadott utasítások hajtódnak végre és a program ugyanúgy folytatódik mint az előbb. Ha nincs nevesítve a kivétel egyetlen ágban sem és nincs WHEN OTHERS ág, vagy egyáltalán nincs kivételkezelő, akkor az adott programegység továbbadja a kivételt. Ez azt jelenti, hogy a kivétel kiváltódik a hívás helyén, és a fenti folyamat ott kezdődik elölről. Tehát a hívási láncon visszafelé lépkedve keres megfelelő kivételkezelőt. Ha a hívási lánc elejére ér és ott sem talál kivételkezelőt, akkor a program a kivételt nem kezelte és a vezérlés átadódik az operációs rendszernek. Kivételkezelőben kiváltott kivétel azonnal továbbadódik. Csak a kivételkezelőben alkalmazható a RAISE; utasítás, amely újra kiváltja azt a kivételt, amely aktivizálta a kivételkezelőt. Ez viszont az adott kivétel azonnali továbbadását eredményezi. Deklarációs

utasításban kiváltódott kivétel azonnal továbbadódik. Csomagban bárhol bekövetkezett és ott nem kezelt kivétel beágyazott csomag esetén továbbadódik a tartalmazó programegységnek, fordítási egység szintű csomagnál viszont a főprogram félbeszakad.

Az Ada fordító nem tudja ellenőrizni a kivételkezelők működését. Az Adában a saját kivételeknek alapvető szerepük van a programírásban, egyfajta kommunikációt tesznek lehetővé a programegységek között az eseményvezérlés révén.

4. Magas Sintű programozási nyelvek 2

4.1. Speciális programnyelvi eszközök.

Programnyelven vagy nyelvi csoporton belüli eszközök összessége, ami más nyelvekre vagy nyelvcsoportokra általánosságban nem jellemző. Ilyen például a C nyelvben a mutató, ami olyan változó, amely egy memóriacímet tárol, és ami akár egy újabb memóriacímet tartalmazó változóra mutathat. Az assembly nyelvekben ilyen eszköznek tekinthető a regiszterek címzésére szolgáló utasítás, hiszen magasabb szintű nyelvekben konkrétan regiszterre nem hivatkozhatunk assembly kód beékelése nélkül, maximum jelezhetjük, hogy regiszterben szeretnénk tárolni a változót. Az objektumorientált (OO) programnyelveknél az objektumorientált paradigma által meghatározott eszközök mindegyike tekinthető speciálisnak a többi nyelv felől tekintve rájuk, és a nyelveken belül is előfordul speciális, más OO nyelvekre nem jellemző eszköz.

4.2. Az objektumorientált programozás eszközei és jelentősége.

Az objektumorientált (OO) paradigma középpontjában a programozási nyelvek absztrakciós szintjének növelése áll. Ezáltal egyszerűbbé, könnyebbé válik a modellezés, a valós világ jobban leírható, a valós problémák hatékonyabban oldhatók meg. Az OO szemlélet szerint az adatmodell és a funkcionális modell egymástól elválaszthatatlan, külön nem kezelhető. A valós világot egyetlen modellel kell leírni és ebben kell kezelni a statikus (adat) és a dinamikus (viselkedési) jellemzőket. Ez az egységbezárás elve.

4.2.1. Osztály

Az OO paradigma az absztrakt adattípus fogalmán épül fel. Az OO nyelvek legfontosabb alapeszköze az absztrakt adattípust megvalósító osztály. Az osztály maga egy absztrakt nyelvi eszköz, ezen nyelvek implementációi gyakran egy osztály együttesként jönnek létre. Az osztály rendelkezik attribútumokkal és módszerekkel. Az attribútumok tetszőleges bonyolultságú adatstruktúrát írhatnak le. A módszerek szolgálnak a viselkedés megadására. Ezek fogalmilag (és általában ténylegesen is) megfelelnek az eljárás orientált nyelvek alprogramjainak.

4.2.2. Objektum

A másik alapeszköz az objektum, ami konkrét nyelvi eszköz. Egy objektum mindig egy osztály példányaként jön létre a példányosítás során. Egy adott osztály minden példánya azonos adatstruktúrával és azonos viselkedésmóddal rendelkezik. Minden objektumnak van címe, az a memóriaterület, ahol az adatstruktúra elemei elhelyezkednek. Az adott címen elhelyezkedő érték együttest az objektum állapotának hívjuk. A példányosítás folyamán az objektum kezdőállapotba kerül. Az OO szemléletben az objektumok egymással párhuzamosan, egymással kölcsönhatásban léteznek. Az objektumok kommunikációja üzenetküldés formájában történik. Minden objektum példányosító osztálya meghatározza azt az interfészt, amely definiálja, hogy más objektumok számára az ő példányainak mely attribútumai és módszer-specifikációi látszanak (erre szolgál a bezárási eszközrendszer – l. később).

Tehát egy objektum küld egy üzenetet egy másik objektumnak (ez általában egy számára látható módszer meghívásával és az üzenetet fogadó objektum megnevezésével történik), ez pedig megválaszolja azt (a módszer visszatérési értéke, vagy változó paraméter segítségével). Az üzenet hatására lehet, hogy az objektum megváltoztatja az állapotát. Az objektumnak van öntudata, minden objektum csak önmagával azonos és az összes többi objektumtól különbözik. Minden objektum rendelkezik egyedi objektumazonosítóval (Object Identifier – OID), amelyet valamilyen nyelvi mechanizmus valósít meg.

4.2.3. Attribútumok és metódusok

Egy osztály attribútumai és metódusai vagy módszerei lehetnek osztály és példány szintűek. A példány szintű attribútumok minden példányosításnál elhelyezésre kerülnek a memóriában, ezek értékei adják meg a példány állapotát. Az osztály szintű attribútumok az osztályhoz kötődnek, nem „többszöröződnek”. Például ilyen attribútum lehet az osztály kiterjedése, ami azt adja meg, hogy az adott osztálynak az adott pillanatban hány példánya van. A példány szintű módszerek a példányok viselkedését határozzák meg. Ezen módszerek meghívásánál mindig meg kell adni egy konkrét objektumot. Azt az objektumot, amelyen egy módszer éppen operál, aktuális példánynak hívjuk. Az osztály szintű módszereknél általában nincs aktuális példány, általában az osztály szintű attribútumok manipulálására használjuk őket.

A példány szintű módszerek lehetnek beállító és lekérdező módszerek. A beállító módszerek hatására az aktuális példány állapotot vált, valamelyik (esetleg mindegyik) attribútumának értéke megváltozik. Ezek eljárás jellegűek, az új attribútum-értékeket paraméterek segítségével határozhatjuk meg. A lekérdező módszerek függvény jellegűek. Az aktuális példány aktuális állapotával térnek vissza. Példányosításkor lefoglalódik a memóriaterület az objektum számára, ott elhelyezésre kerülnek a példány szintű attribútumok, és beállítódik a kezdőállapot. Az objektum ettől kezdve él és tudja, hogy mely osztály példányaként jött létre.

4.2.4. Konstruktor

A kezdőállapot meghatározására az OO nyelvek általában egy speciális módszert, a konstruktort használják. Az aktuális példány kezelését az OO nyelvek a példány szintű módszerekbe implicit paraméterként beépülő speciális hivatkozással oldják meg. Az OO nyelvek az osztályok között egy aszimmetrikus kapcsolatot értelmeznek, melynek neve öröklődés. Az öröklődés az újrafelhasználhatóság eszköze. Az öröklődési viszonynál egy már létező osztályhoz kapcsolódóan – melyet szuperosztálynak (szülő osztálynak, alaposztálynak) hívunk – hozunk létre egy új osztályt, melynek elnevezése alosztály (gyermek osztály, származtatott osztály). Az öröklődés lényege, hogy az alosztály átveszi (örökli) szuperosztályának minden (a bezárás által megengedett) attribútumát és módszerét, és ezeket azonnal fel is tudja használni. Ezen túlmenően új attribútumokat és módszereket definiálhat, az átvett eszközöket átnevezheti, az átvett neveket újradeklarálhatja, megváltoztathatja a láthatósági viszonyokat, a módszereket újraimplementálhatja.

4.2.5. Öröklődés

Az öröklődés lehet egyszeres és többszörös. Egyszeres öröklődés esetén egy osztálynak pontosan egy, többszörös öröklődés esetén egynél több szuperosztálya lehet. Mindkét esetben igaz, hogy egy osztálynak akárhány alosztálya létrehozható. Természetesen egy alosztály lehet egy másik osztály szuperosztálya. Így egy osztályhierarchia jön létre. Az osztályhierarchia egyszeres öröklődés esetén fa, többszörös öröklődés esetén aciklikus gráf. A többszörös öröklődést valló nyelvek osztályhierarchiájában is van azonban általában egy kitüntetett „gyökér” osztály, amelynek nincs szuperosztálya, és léteznek olyan „levél” osztályok, amelyeknek nincsenek alosztályai. A többszörös öröklődésnél gondot okozhat a különböző szuperosztályokban használt azonos nevek ütközése. Az öröklődési hierarchiában az egy úton elhelyezkedő osztályok közvetlen vagy közvetett öröklődési viszonyban vannak. Az alosztályok irányába haladva az osztályok leszármazott osztályai helyezkednek el, a másik irányban viszont az elől osztályok találhatók. Az egymással elől-leszármazott viszonyban nem levő osztályokat kliens osztályoknak hívjuk. Az osztályok eszközeinek láthatóságát szabályozza a bezárás.

4.2.6. Bezárási szintek

Az OO nyelvekben általában a következő bezárási szintek léteznek. Publikus szint esetén az eszközt látja az összes kliens osztály. Védett szintnél az eszközökhöz csak a leszármazott osztályok férhetnek hozzá. A privát szintű eszközök viszont csak az adott osztályban használhatók (pontosabban: egy alosztály természetesen örökli a privát attribútumokat és módszereket, de ezekre közvetlenül, explicit módon nem hivatkozhat – láthatatlan öröklés). Több OO nyelv értelmez még egy negyedik szintet is, amely a nyelv programegység szerkezetén alapul.

4.2.7. Helyettesíthetőség

Az öröklődésen alapul és az újrafelhasználhatóságnak egy igen jellegzetes megnyilvánulása a helyettesíthetőség. Az OO paradigma azt mondja, hogy egy leszármazott osztály példánya a program szövegében minden olyan helyen megjelenhet, ahol az elől osztály egy példánya. Egy alosztály az örökölt módszereket újraimplementálhatja. Ez azt jelenti, hogy különböző osztályokban azonos módszer-specifikációkhoz különböző implementáció tartozik. Ezek után a kérdés az, hogy ha meghívunk egy objektumra egy ilyen módszert, akkor melyik implementáció fog lefutni. A választ egy nyelvi mechanizmus, a kötés adja meg.

Az OO nyelvek két fajta kötést ismernek. Statikus kötés esetén már fordításkor eldől a kérdés. Ekkor a helyettesíthetőség nem játszik szerepet. A forrásszövegben megadott objektum deklaráló osztályának módszere fog lefutni minden esetben. Dinamikus kötés esetén a kérdés csak futási időben dől el, a megoldás a helyettesíthetőségen alapul. Annak az objektumnak a példányosító osztályában megadott (vagy ha nem írta fölül, akkor az öröklött) implementáció fog lefutni, amelyik ténylegesen kezelésre kerül. Az OO nyelvek egy része a dinamikus kötést vallja. Másik részükben mindkettő jelen van, az egyik alapértelmezett, a másikat a programozónak explicit módon kell beállítania.

Az OO nyelvek általában megengedik a módszernevek túlterhelését. Ez annyit jelent, hogy egy osztályon belül azonos nevű és természetesen eltérő implementációjú módszereket tudunk létrehozni. Ekkor természetesen a hivatkozások feloldásához a specifikációknak különbözniük kell a paraméterek számában, vagy azok típusában (ez nem mindig elég).

4.2.8. Speciális osztályok

Az OO nyelvek általában ismerik az absztrakt osztály fogalmát. Az absztrakt osztály egy olyan eszköz, amellyel viselkedésmintákat adhatunk meg, amelyeket aztán valamely leszármazott osztály majd konkretizál. Egy absztrakt osztályban általában vannak absztrakt módszerek, ezeknek csak a specifikációja létezik, implementációjuk nem. Egy absztrakt osztályból származtatható absztrakt és konkrét osztály. A konkrét osztály minden módszeréhez kötelező az implementáció, egy osztály viszont mindaddig absztrakt marad, amíg legalább egy módszere absztrakt. Az absztrakt osztályok nem példányosíthatók, csak örököltethetők.

Egyes OO nyelvekben létrehozhatók olyan osztályok, amelyekből nem lehet alosztályokat származtatni (ezek az öröklődési hierarchia „levelei" lesznek). Ezek természetesen nem lehetnek absztrakt osztályok, hiszen akkor soha nem lehetne őket konkretizálni.

Egyes OO nyelvek ismerik a paraméterezett osztály fogalmát. Ezek lényegében az OO világ generikusai.

4.2.9. Objektumok élettartama

Az OO nyelvekben az objektumok memóriában kezelt konstrukciók. Egy objektum mindig tranziens, tehát nem éli túl az őt létrehozó programot. I/O segítségével természetesen bármely objektum állományba menthető és azután bármikor létrehozható egy másik objektum, amelynek állapota ugyanaz lesz.

A nem nyelvi OO rendszerek (pl. adatbázis-kezelők) ismerik a perzisztens objektum fogalmát. Ekkor az objektum túléli az őt létrehozó alkalmazást, bármikor újra betölthető a memóriába, és ugyanaz az objektum marad. Természetesen egy program működése közben is fel kell szabadítani a már szükségtelen objektumokhoz rendelt tárterületet. Erre az OO nyelvek kétféle megvalósítást tartalmaznak. Egy részük azt mondja, hogy a programozónak kell explicit módon megszüntetnie az objektumot. Más részük automatikus törlési mechanizmust biztosít (garbage collection). Ezeknél a háttérben, aszinkron módon, automatikusan működik a „szemétgyűjtögető" a szokásos algoritmusok valamelyike (pl. hivatkozásfigyelés) alapján.

4.2.10. Objetumorientált nyelvek fajtái

Az OO nyelveknek két nagy csoportja van. A tiszta OO nyelvek teljes mértékben az OO paradigma mentén épülnek fel, ezekben nem lehet más paradigma eszközeinek segítségével programozni. Ezen nyelvekben egyetlen osztályhierarchia létezik. Ez adja a nyelvi rendszert és a fejlesztői környezetet is egyben. Ezen nyelvekben a programozás azt jelenti, hogy definiáljuk a saját osztályainkat, azokat elhelyezzük az osztályhierarchiában, majd példányosítunk. A hibrid OO nyelvek valamilyen más paradigma (eljárásorientált, funkcionális, logikai, stb.) mentén épülnek fel, és az alap eszközrendszerük egészül ki OO eszközökkel. Ezen nyelvekben mindkét paradigma mentén lehet programozni. Általában nincs beépített osztályhierarchia (hanem pl. osztálykönyvtárak vannak), és a programozó saját osztályhierarchiákat hozhat létre. Egyes tiszta OO nyelvek az egységesség elvét vallják. Ezen nyelvekben egyetlen programozási eszköz van, az objektum. Ezekben a nyelvekben tehát minden objektum, a módszerek, osztályok is. Az OO paradigma imperatív paradigmaként jött létre. Tehát ezek a nyelvek algoritmikusak, és így eredendően fordítóprogramosak. A SIMULA 67 az első olyan nyelv, amely tartalmazza az OO eszközrendszert, de a paradigma fogalmait később a Smalltalk fejlesztői csapata tette teljessé. Aztán folyamatosan kialakultak a hibrid OO nyelvek, és megjelent a deklaratív OO paradigma (CLOS, Prolog++) is.

4.3. Logikai programozás.

A paradigma az 1970-es évek elején születik meg az első logikai programozási nyelv, a Prolog megkonstruálásával. A logikai paradigma a matematikai logika fogalom- és eszközrendszerén épül fel. A Prolog alapjait az elsőrendű predikátumkalkulus és a rezolúciós algoritmus képezi. Egy logikai program nem más, mint egy absztrakt modellre vonatkozó állítások egy halmaza. Az állítások a modell elemeinek tulajdonságait és a közöttük levő kapcsolatokat formalizálják. Az állítások egy konkrét kapcsolatot leíró részhalmazát predikátumnak nevezzük. Általánosságban egy logikai program lefuttatása egy, az állításokból következő tétel konstruktív bizonyítását jelenti. Ekkor a program állításai egy megoldási környezetet definiálnak, és ebben a környezetben tesszük fel a programnak a kérdést (vagy fogalmazzuk meg a feladatot), amire a választ egy következtető gép keresi meg.

A logikai programozási nyelvekben az állítás tény vagy szabály lehet. Az állításokat és a kérdéseket közös néven mondatoknak nevezzük. Egyes logikai nyelvekben a szigorúan vett logikai eszközökön túlmutató mondatok is lehetnek. A deklarációk a predikátumok alkalmazását pontosítják, a direktívák a program futtatási környezetét határozzák meg.

A Prolog egy általános célú magas szintű programozási nyelv. A teljes Prolog a logikai eszközrendszeren kívül tartalmaz metalogikai és logikán kívüli nyelvi elemeket is, továbbá be van ágyazva egy interaktív fejlesztői környezetbe.

Egy tiszta Prolog program felhasználói predikátumok együttese, amelyekben sehol sincs hivatkozás beépített predikátumra.

A Prolog egy nem típusos, interpreteres nyelv. A Prologban a mondatokat . zárja.

A Prolog nyelv alapeleme a term, amely lehet egyszerű és összetett. Egy egyszerű term az vagy állandó vagy változó. Az állandó az név vagy szám. A név egy kisbetűvel kezdődő azonosító, vagy a +, –, *, /,
, ^, <, >, =, ∼, :, ., ?, @, #, &, $ karakterekből álló karaktersorozat. Van négy foglalt név: ;, !, [], .

A szám egy olyan karaktersorozat, amely formálisan megfelel az eljárásorientált nyelvek egész és valós numerikus literáljának.

A változó speciális változó. Típusa nincs, címe nem hozzáférhető. Neve aláhúzásjellel vagy nagybetűvel kezdődő azonosító. Értékkomponensének kezelése speciális. A változó a matematikai egyenletek ismeretlenjének felel meg. Rá az egyszeres értékadás szabálya vonatkozik. Egy változónak tehát vagy nincs értéke és ekkor a neve önmagát, mint karaktersorozatot képviseli hatáskörén belül mindenütt, vagy van értéke és ekkor a név mindenütt ezt az értékkomponenst jelenti. Az értékkomponens nem írható felül. A tiszta Prologban egy változónak értéket a Prolog következtető gép adhat. A változó hatásköre az a mondat, amelyikben szerepel a neve. Kivétel ez alól az a változó, amelynek neve _ (ún. névtelen változó), amelynek minden előfordulása más- más változót jelöl.

Egy tiszta Prolog program futtatásának célja általában a kérdésekben szereplő változók lehetséges értékeinek meghatározása. Általános Prolog konvenció, hogy az eredmény szempontjából érdektelen változók nevét aláhúzásjellel kezdjük.

Az összetett term alakja: név(argumentum [, argumentum]...) ahol az argumentum egy tetszőleges term, vagy egy aposztrófok közé zárt tetszőleges karaktersorozat lehet.

A tény egy összetett term és mint olyan, egy igaz állítás.

Egy szabály áll fejből és törzsből és közöttük valamilyen elhatároló áll (nálunk ez a :- lesz). A fej egy összetett term, a törzs egy vesszőkkel elválasztott összetett term sorozat (ezek predikátumok).

A szabály egy következtetési szabály: a fej akkor igaz, ha a törzs igaz. A vessző tehát itt egy rövidzár és műveletnek felel meg. A kérdésnek csak törzse van.

A Prologban a deklarációk és direktívák alakja: :- törzs

A Prolog állításaiban szereplő változók univerzálisan, a kérdésben szereplők viszont egzisztenciálisan kvantáltak.

Tehát a tények törzs nélküli szabályok, vagyis a törzs mindig igaznak tekinthető. A kérdés viszont fej nélküli szabály, azaz vagy azt kérdezzük, hogy a megoldási környezet mely elemei teszik igazzá a predikátumokat, vagy pedig csak egy „igen-nem” típusú kérdést teszünk föl.

A szabályok lehetnek rekurzívak.

Akárhány olyan szabály lehet, ahol a fej azonos, ilyenkor az argumentumok közötti kapcsolatot az egyes állítások által definiált kapcsolatok uniója határozza meg.

A Prolog következtető gép a program futtatása során a memóriában egy keresési fát épít föl és jár be preorder módon. A fát teljes mértékben soha nem építi föl, mindig csak az aktuálisan kezelt út áll rendelkezésre, a bejárt csúcsot törli a feldolgozás után. A fa csúcsaiban a kérdés aktuális alakja áll, az éleket viszont az adott lépésben végrehajtott változóhelyettesítések címkézik. A kérdés megválaszolásánál a tényeket és szabályokat a felírásuk sorrendjében használja fel, a megoldásnál alkalmazott technika pedig az illesztés és a visszalépés.

A megoldás lépései a következők:

1. A keresési fa gyökerében az eredeti kérdés áll. Induláskor ez az aktuális csúcs.

2. Ha az aktuális csúcsban a kérdés törzse üres, akkor megvan egy megoldás. Ezt kiírja a rendszer, és rákérdez, hogy a felhasználó akar-e további megoldásokat. Ha nem, akkor a programnak vége, ha igen, akkor folytatás 4-től.

3. Ha az aktuális csúcsban a kérdés törzse nem üres, akkor veszi a törzs első predikátumát, majd az első tényre végrehajt egy illesztést. Ha nem sikerül az illesztés, akkor veszi sorra a további tényeket és próbál azokra illeszteni. Ha sikerül valamelyik tényre illeszteni, akkor a fában létrehoz egy új csúcsot és abban a kérdés aktuális alakja úgy áll elő, hogy elhagyja az első predikátumot. Az élet címkézi az illesztéshez esetleg szükséges változóhelyettesítésekkel. Ha egyetlen tényre sem sikerült illeszteni, akkor megpróbál illesztést találni a szabályok fejére. Ha van illeszkedés, akkor létrehoz egy új csúcsot a fában, az éleket ugyanúgy címkézi, és a kérdés új alakja úgy keletkezik, hogy a predikátumot felülírja az illeszkedő fejű szabály törzsével. Ha nem illeszkedik egyetlen tény és egyetlen szabályfej sem, akkor a Prolog azt mondja, hogy zsákutcába jutott és a végrehajtás folytatódik 4-től, különben az új csúcs lesz az aktuális és folytatás 2-től.

4. Ez a visszalépés. Ha az aktuális csúcs a fa gyökere, akkor a program véget ér, nincs több megoldás (az eddigiekkel együtt esetleg egy sem). Különben törli a fában az aktuális csúcsot, és a megelőző csúcs lesz az aktuális. Egyben törli a két csúcsot összekötő él változóhelyettesítéseit. Ezután 3-tól folytatva megpróbál illesztést keresni az eddig felhasznált állításokat követő állítások segítségével.

Az illesztés algoritmusa a következő:

1. Ha az illesztendő termsorozatok üresek, akkor vége (az illesztés sikeres), különben illeszti a két sorozat első elemeit 2 szerint, majd ha, azok illeszkednek, folytatódik az illesztés a sorozatok maradék elemeire 1 szerint.
2. Ha mindkét term állandó, akkor attól függően, hogy mint karaktersorozatok azonosak-e, az illesztés sikeres lesz, vagy meghiúsul.
3. Állandó és összetett term esetén az illesztés sikertelen lesz.
4. Két összetett term esetén az illesztés sikertelen, ha különbözik a nevük, vagy az argumentumaik száma. Különben az argumentumok sorozatai kerülnek illesztésre 1 szerint.
5. Ha mindkét term változó, bármelyik helyettesíthető a másikkal. Általában az állítás változói kapnak értéket.
6. Ha az egyik term változó, akkor az helyettesítődik a másik (állandó vagy összetett) termmel.

4.4. Funkcionális programozás

A funkcionális paradigma középpontjában a függvények állnak. Egy funkcionális (vagy applikatív) nyelvben egy program típus-, osztály- és függvénydeklarációk, illetve függvénydefiníciók sorozatából, valamint egy kezdeti kifejezésből áll. A kezdeti kifejezésben tetszőleges hosszúságú (esetleg egymásba ágyazott) függvényhívás- sorozat jelenhet meg. A program végrehajtását a kezdeti kifejezés kiértékelése jelenti. Ezt úgy képzelhetjük el, hogy a kezdeti kifejezésben szereplő függvények meghívása úgy zajlik le, hogy a hívást szövegszerűen (a paraméterek figyelembevételével) helyettesítjük a definíció törzsével. A helyettesítés pontos szemantikáját az egyes nyelvek kiértékelési (átírási) modellje határozza meg.

A funkcionális nyelvek esetén nem választható szét a nyelvi rendszer a környezettől. Ezek a nyelvi rendszerek eredendően interpreter alapúak, interaktívak, de tartalmaznak fordítóprogramokat is. Középpontjukban mindig egy redukciós (átíró) rendszer áll. Ha a redukciós rendszer olyan, hogy az egyes részkifejezések átírásának sorrendje nincs hatással a végeredményre, akkor azt konfluensnek nevezzük.

Egy funkcionális nyelvű program legfontosabb építőkövei a saját függvények. Ezek fogalmilag semmiben sem különböznek az eljárásorientált nyelvek függvényeitől. A függvény törzse meghatározza adott aktuális paraméterek mellett a visszatérési érték kiszámításának módját. A függvény törzse a funkcionális nyelvekben kifejezésekből áll.

Egy funkcionális nyelvi rendszer beépített függvények sokaságából áll. Saját függvényt beépített, vagy általunk már korábban definiált függvények segítségével tudunk definiálni (függvényösszetétel). Egy funkcionális nyelvben a függvények alapértelmezett módon rekurzívak lehetnek, sőt létrehozhatók kölcsönösen rekurzív függvények is.

A kezdeti kifejezés redukálása (a nyelv által megvalósított kiértékelési stratégia alapján) mindig egy redukálható részkifejezés (egy redex) átírásával kezdődik. Ha a kifejezés már nem redukálható tovább, akkor normál formájú kifejezésről beszélünk.

A kiértékelés lehet lusta kiértékelés, ekkor a kifejezésben a legbaloldalibb legkülső redex kerül átírásra. Ez azt jelenti, hogy ha a kifejezés egy függvényhívás, akkor az aktuális paraméterek kiértékelését csak akkor végzi el a rendszer, ha szükség van rájuk. A lusta kiértékelés mindig eljut a normál formáig, ha az létezik.

A mohó kiértékelés a legbaloldalibb legbelső redexet írja át először. Ekkor tehát az aktuális paraméterek kiértékelése történik meg először. A mohó kiértékelés gyakran hatékonyabb, de nem biztos, hogy véget ér, még akkor sem, ha létezik a normál forma.

Egy funkcionális nyelvet tisztán funkcionálisnak (tisztán applikatívnak) nevezünk, ha nyelvi elemeinek nincs mellékhatása, és nincs lehetőség értékadásra vagy más eljárásorientált nyelvi elem használatára.

A nem tisztán funkcionális nyelvekben viszont van mellékhatás, vannak eljárásorientált (néha objektumorientált) vagy azokhoz hasonló eszközök.

A tisztán funkcionális nyelvekben teljesül a hivatkozási átláthatóság. Ez azt jelenti, hogy egy kifejezés értéke nem függ attól, hogy a program mely részén fordul elő. Tehát ugyanazon kifejezés értéke a szöveg bármely pontján ugyanaz. A függvények nem változtatják meg a környezetüket, azaz a tartalmazó kifejezés értékét nem befolyásolják. Az ilyen

nyelvnek nincsenek változói, csak konstansai és nevesített konstansai. A tisztán funkcionális nyelvek általában szigorúan típusosak, a fordítóprogram ellenőrzi a típuskompatibilitást. A funkcionális nyelvek eszközként tartalmaznak olyan függvényeket, melyek paramétere, vagy visszatérési értéke függvény (funkcionálok, vagy magasabb rendű függvények). Ez a procedurális absztrakciót szolgálja.

A funkcionális nyelvek egy részének kivételkezelése gyenge vagy nem létezik, másoknál hatékony eszközrendszer áll rendelkezésre.

A függvényösszetétel asszociatív, így a funkcionális nyelven megírt programok kiértékelése jól párhuzamosítható. Az elterjedt funkcionális nyelveknek általában van párhuzamos változata.

A funkcionális nyelvek közül a Haskell egy erősen típusos, tisztán funkcionális, lusta kiértékelést megvalósító, a LISP egy imperatív eszközöket is tartalmazó, objektumorientált változattal (CLOS) is rendelkező, mohó kiértékelést valló nyelv.

5. Adatszerkezetek és algoritmusok

5.1. Adatszerkezetek reprezentációja.

A reprezentáció az absztrakt adatszerkezet tárolásának, ábrázolásának és leképezésének a módja. Az ábrázolás kétféleképpen történhet: folytonosan vagy szétszórtan.

Folytonos Egy tárhelyen csak az adatelem értéke található, **az adatszerkezethez** tartozó adatelemek **folytonosan egymás után következnek a memóriában**, az adatelemek mérete általában azonos. A kezdőcím és az elemek száma ismert. Az adatelemek tárolási jellemzői (típus, ábrázolás, hossz) azonosak. Közvetlen elérést biztosít, a keresés, rendezés és csere műveletek gyorsabbak, de a bővítés és a fizikai törlés nehezebb.

Szétszórt Egy tárhelyen az adatelemen kívül legalább egy cím is van, ami az adatszerkezetben szomszédos adatelem címe, ennek segítségével érhetjük el az összes adatelemet. A **memóriában nem egymást követően helyezkednek el az adatelemek**. Az adatelemek tárolási jellemzői eltérhetnek. Könnyebb a bővítés és a fizikai törlés, illetve nagyobb adatmennyiséget is könnyebb tárolni, de nehezebb a keresés, rendezés és csere, mert nem érjük el közvetlenül az adatelemeket.

5.2. Műveletek adatszerkezetekkel.

Az adatszerkezetek kezeléséhez műveletek állnak rendelkezésre, melyek mindegyik adatszerkezetnél más megvalósítást követelnek. Vannak olyan adatszerkezetek, melyeknél néhány művelet nem lehetséges, vagy nincs értelmezve.

1. **Létrehozás** Az adatszerkezet szerkezeti vázának leíróit adjuk meg. Létrehozzuk a fejmutatót, ami az első elemre tud hivatkozni. Néhány adatszerkezetnél kezdőérték is definiálható.
2. **Bővítés** A meglévő szerkezet bővítése egy vagy több adatelemmel. *Csak dinamikus adatszerkezeteknél lehetséges.*
3. **Törlés** Egy vagy több adatelem törlése a szerkezetből. Létezik logikai és fizikai törlés.
   a. **Logikai törlés** Felülírjuk az adatelem értékét egy olyan értékre, amely nem fordulhat elő, ezzel jelezve, hogy ott nincs értelmezhető adat.
   b. **Fizikai törlés** A tárhelyet is eltávolítjuk, így a teljes adatelem megszűnik. *Csak dinamikus adatszerkezeteknél lehetséges.*
4. **Csere** Két adatelem felcserélése. Általában az értékek felülírásával történik, nem a tárhelyek mozgatásával.
5. **Rendezés** Valamilyen szabály alapján növekvő vagy csökkenő sorrendet adunk meg az adatelemek között. Ehhez véges számú csere műveletet kell elvégezni. *Fajtái:* szélsőérték kiválasztásos, beszúrásos, buborék-, shell-, gyorsrendezés, stb.
6. **Keresés** Az adatszerkezet egy adott értékkel rendelkező adatelemének megtalálása, mellyel annak indexét és címét is megtudjuk. *Fajtái:* teljes, lineáris, bináris keresés.
7. **Elérés** Egy adatelemhez való hozzáférés annak címe segítségével, hogy valamilyen műveletet hajthassunk végre rajta.
8. **Bejárás** Egy adatszerkezet minden elemének egymás utáni elérése. *Fajtái:* soros, szekvenciális, közvetlen.
9. **Feldolgozás** Egy adatelemen végrehajtott módosítás.
10. **Felszabadítás** Memóriában tárolt adatszerkezetek memóriából való eltávolítása.

5.3. Adatszerkezetek osztályozása és jellemzésük.

1. **Adatelemek száma szerint**
   a. **Statikus** Az adatelemek száma állandó, az adatszerkezet létrehozásakor rögzül. Az adatelemek száma csak közvetetten módosítható: egy különböző elemszámmal rendelkező adatszerkezetet kell létrehozni és a megtartandó értékeket átmásolni a megfelelő helyekre, majd a régit felszabadítani. (pl. tömb)
   b. **Dinamikus** Az adatelemek száma időben változhat. (pl. lista)
2. **Adatelemek típusa szerint**

a. Homogén Az adatszerkezet minden adatelemének típusa azonos. Ez a típus lehet egyszerű vagy összetett, így további részekre bontható. (pl. halmaz)
b. Heterogén Az adatelemek típusa nem egyezik meg. (pl. rekord)

3. Adatelemek közötti kapcsolat szerint

a. Struktúra nélküli Nincs rögzített sorrendi kapcsolat az adatelemek között, csak az azonos típus köti össze őket. (pl. halmaz)
b. Asszociatív Nincs lényegi kapcsolat az adatelemek között, csak szabály nélküli sorrendiség. (pl. tömb, mátrix)
c. Szekvenciális Az adatelemek egymás után helyezkednek el úgy, hogy egy elem csak a szomszédjain keresztül érhető el, közvetlenül nem. (pl. lista)
d. Hierarchikus Minden adatelem csak egy elemből érhető el, de egy elemből több elem is elérhető. Az elemek között egy-sok kapcsolat áll fenn. (pl. fa)
e. Hálós Minden adatelem több elemből is elérhető, és egy elemből több elem is elérhető. Az elemek között sok-sok kapcsolat áll fenn. (pl. gráf)

4. Tárolás szerint

a. Folytonos Az adatelemek egymást követő címen helyezkednek el (1. oldal).
b. Szétszórt Az adatelemek véletlenszerűen helyezkednek el (1. oldal).

5.4. Szekvenciális adatszerkezetek: sor, verem, lista, sztring.

Lista Olyan dinamikus adatszerkezet, melynek adatelemeiben vagy a következő elem címe, vagy az előző és következő elemek címe is megtalálható a tárolt érték mellett. Az első elem (lista feje vagy fejmutató) speciális, mert csak az első tényleges adatelem címét tartalmazza, értéket nem. A lista méretét az elemek száma határozza meg, amit külön tárolhatunk, vagy függvénnyel határozhatjuk meg. Minden műveletet lehet rajta használni. Listafajták:

1. Egyirányban láncolt Az adatelemek a következő elem címét tárolják.
2. Kétirányban láncolt Az adatelemek az előző és következő elem címét is tárolják.
3. Cirkuláris Az utolsó elem következője az első, az elsőt megelőző az utolsó elem.
4. Multilista Az adatelemek valamilyen másik lista fejmutatói.

Sor Olyan speciális lista, melynek csak az egyik elemét érjük el, a bővítés és törlés műveletek speciálisan vannak megvalósítva. Folytonos és szétszórt ábrázolással is megvalósítható. Létrehozásához két értékre van szükség, melyek jelzik az első és utolsó elem címét. A sor FIFO (First In First Out) adatszerkezet, azt az elemet érjük el először, amelyik előbb került bele. Speciális műveletei:

1. ACCESS HEAD az első elem elérése
2. PUT sor bővítése a végén
3. GET sor első elemének elérése és törlése

Verem A verem olyan, mint egy fordított elérésű sor. Csak egy elemet érünk el, a bővítés és törlés műveletek speciálisan vannak megvalósítva. Folytonos (általában) és szétszórt ábrázolással is megvalósítható. Szerkezetét leírni két értékkel lehet: az egyik a verem alját (elejét), a másik a verem tetejét (végét) jelzi. A verem LIFO (Last In First Out) adatszerkezet, azt az elemet érjük el először, amelyik utoljára került bele. Speciális műveletei:

1. ACCESS HEAD az utolsó elem elérése
2. PUSH verem bővítése a végén
3. POP verem utolsó elemének elérése és törlése

Sztring Olyan speciális adatszerkezet, melynek adatelemei karaktereket kódolnak. A karakterek kódolása (ASCII, UTF-8, UNICODE, stb.) és a tárolás implementációja határozza meg, hogyan lehet kezelni őket. Lehetséges asszociatív adatszerkezettel is tárolni, ekkor minden karakterét közvetlenül el lehet érni. Gyakran úgy van megvalósítva, hogy a végén lehet bővíteni, így karakterenkénti olvasással sztringet lehet összefűzni. Speciális műveletei:

1. karakterképzés
2. részsztringképzés
3. konkatenáció (összefűzés)

5.5. Egyszerű és összetett állományszerkezetek.

Egyszerű állományszerkezet esetén a fizikai állomány csak a logikai állomány adatelemeit tartalmazza. Ez azt jelenti, hogy a fizikai állomány a logikai állomány adataiból kialakítható, nem szükséges hozzá technikai szerkezethordozó (strukturáló) adatokat is tárolni, illetve meglétük nem meghatározó a szerkezet szempontjából. Az egyszerű állományszerkezeteknek négy típusát különböztetjük meg:

1. **Szeriális** Szerkezet nélküli állomány. Logikai és fizikai szinten sincs megkötés a rekordok közötti kapcsolatra. Kezelése egyszerű, de lassú benne egy adott rekord keresése, és nem lehet rendezni. Szabadon szegmentálható, így a tárolása nem okoz problémát. Általában ideiglenes tárolásra használt.
2. **Szekvenciális** Logikai szinten sorrend van az egyes rekordok között. A háttértáron való elhelyezésre nincsenek megkötések. A keresés a sorrendiség miatt gyorsabb, jó kapacitáskihasználás jellemzi, bármilyen (soros és közvetlen) háttértárolón megvalósítható. Hátránya, hogy nem támogatja a közvetlen elérést, létrehozásához először rendezni kell az adatokat.
3. **Direkt** A logikai rekordok fizikai elhelyezését egy kölcsönösen egyértelmű hash függvény határozza meg a logikai rekordok azonosítója alapján, így a rekordok és a blokkok között szoros kapcsolat van. Emiatt minden rekord közvetlenül elérhető, de csak címezhető tárolón valósítható meg (mágnesesen például nem). A logikai rekordok között is jól meghatározható kapcsolat van. Csak fix formátumú rekordokat tartalmazhat, és az állomány nem szegmentálható. Nagyon gyors elérést biztosít, de nem minden rendszer kezeli.
4. **Random** A direkt állományokhoz hasonlóan hash függvény helyezi el a rekordokat a blokkokban, de ez a függvény csak egyértelmű (nem kölcsönösen egyértelmű). A logikai rekordok között nincs jól meghatározott kapcsolat. A hash függvény különböző rekordazonosítókhoz ugyanazt a blokkot is kijelölheti. Ennek kezelése:

   a. Nyílt címzés A foglalt helyre került elemet a következő szabad helyen helyezi el
   b. Láncolás Listába fűzi az egy helyre került elemeket

Összetett állományszerkezet azt jelenti, hogy a logikai rekordokon túl szerkezethordozó információkat is tárolunk az egyszerűbb és gyorsabb feldolgozás érdekében. Alapja egy egyszerű szerkezetű állomány, az alapállomány, amely legtöbbször szeriális vagy szekvenciális. Az alapállományra épülnek rá a plusz információhordozó adatok. Strukturáló adatok megadása:

1. **Láncolás** Az információhordozó adatok állományon belül jelennek meg mutatómezők formájában, ekkor a rekordokat láncolt listába fűzzük fel
2. **Indexelés** A plusz információk az állományon kívül egy (általában) vagy több indextábla formájában jelennek meg

Mivel mindkét technika lemezcímeket kezel, ezért az összetett állományszerkezetek csak közvetlen elérésű háttértárolón alakíthatók ki. A következő összetett állományszerkezeteket különböztetjük meg:

1. Láncolt szeriális állomány
2. Indexelt szeriális állomány
3. Indexelt szekvenciális állomány
4. Multilista állomány
5. Invertált állomány

6. Adatbázisrendszerek

6.1. Relációs, ER és objektumorientált modellek jellemzése.

6.2. Adatbázisrendszer.

6.3. Funkcionális függés.

6.4. Relációalgebra és relációkalkulus.

6.5. Az SQL.

7. Hálózati architektúrák

7.1. Az ISO OSI hivatkozási modell.

7.2. Ethernet szabványok.

7.3. A hálózati réteg forgalomirányító mechanizmusai.

7.4. Az internet hálózati protokollok, legfontosabb szabványok és szolgáltatások.

8. Fizika 1

8.1. Fizikai fogalmak, mennyiségek.

8.2. Impulzus, impulzusmomentum.

8.3. Newton törvényei.

8.4. Munkatétel.

8.5. Az I. és II. főtétel.

8.6. A kinetikus gázmodell.

9. Fizika 2

9.1. Elektromos alapfogalmak és alapjelenségek.

9.2. Ohm-törvény.

9.3. A mágneses tér tulajdonságai.

9.4. Elektromágneses hullámok.

9.5. A Bohr-féle atommodell.

9.6. A radioaktív sugárzás alapvető tulajdonságai.

10. Elektronika 1, 2

10.1. Passzív áramköri elemek tulajdonságai, RC és RLC hálózatok.

10.2. Diszkrét félvezető eszközök, aktív áramköri elemek, alapkapcsolások.

10.3. Integrált műveleti erősítők.

10.4. Tápegységek.

10.5. Mérőműszerek.

11. Digitális Technika

11.1. Logikai függvények kapcsolástechnikai megvalósítása.

11.2. Digitális áramköri családok jellemzői(TTL, CMOS, NMOS).

11.3. Különböző áramköri családok csatlakoztatása.

11.4. Kombinációs és szekvenciális hálózatok. A/D és D/A átalakítók.

II. rész

Infokommunikációs hálózatok specializáció

12. Távközlő hálózatok

12.1. Fizikai jelátviteli közegek.

12.2. Forráskódolás, csatornakódolás és moduláció.

12.3. Csatornafelosztás és multiplexelési technikák.

12.4. Vezetékes és a mobil távközlő hálózatok.

12.5. Műholdas kommunikáció és helymeghatározás.

13. Hálózatok hatékonyságanalízise

13.1. Markov-láncok, születési-kihalási folyamatok.

13.2. A legalapvetőbb sorbanállási rendszerek vizsgálata.

13.3. A rendszerjellemzők meghatározásának módszerei, meghatározásuk számítógépes támogatása.

14. Adatbiztonság

14.1. Fizikai, ügyviteli és algoritmusos adatvédelem, az informatikai biztonság szabályozása.

14.2. Kriptográfiai alapfogalmak.

14.3. Klasszikus titkosító módszerek.

14.4. Digitális aláírás, a DSA protokoll.

15. A RIP protokoll működése és paramétereinek beállítása (konfigurációja).

16. Bevezetés a Cisco eszközök programozásába 1

16.1. A forgalomszűrés, forgalomszabályozás (Trafficfiltering, ACL) céljai és beállítása (konfigurációja) egy választott példa alapján.

17. Bevezetés a Cisco eszközök programozásába 2

17.1. A forgalomirányítási táblázatok felépítése, statikus és dinamikus routing összehasonlítása.

Tárgymutató