Záróvizsga tételsor mérnökinformatikus hallgatóknak

A Debreceni Egyetem mérnökinformatikus alapszakához

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

*Az egyes tételek kidolgozásai nem tőlem származnak. Az esetenként előforduló hibákért és pongyola fogalmazásért felelősséget nem vállalok.

I. rész
Általános kérdések

1. Az informatika logikai alapjai

Az elsőrendű matematikai logikai nyelv. A nyelv interpretációja, formulák igazságértéke az interpretációban adott változókiértékelés mellett. Logikai törvény, logikai következmény. Logikai ekvivalencia, normálformák. Kalkulusok (Gentzen-kalkulus).

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.*

1.2. 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*(ró) 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*(nű) értékelésen egy olyan függvényt értünk, amely teljesíti a következőket:*

[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.

- $Dom(\nu) = Var$
- $Hax \in Var$, *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 $\Gamma \cup \{\neg A\}$ 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[4], 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[5], 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.

[4] konjunkció: Logikai ÉS($\wedge$) művelet
[5] diszjunkció: Logikai VAGY($\vee$) művelet

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

2. Operációs rendszerek

Operációs rendszerek fogalma, felépítése, osztályozásuk. Az operációs rendszerek jellemzése (komponensei és funkciói). A rendszeradminisztráció, fejlesztői és alkalmazói támogatás eszközei.

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

Adattípus, konstans, változó, kifejezés. Paraméterkiértékelés, paraméterátadás. Hatáskör, névterek, élettartam. Fordítási egységek, kivételkezelés.

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 típusúi 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 feloszthatóújabb 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 eldől 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 eldől, 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 forrásszövegrészek, 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

Speciális programnyelvi eszközök. Az objektumorientált programozás eszközei és jelentősége. Funkcionális és logikai programozás.

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 előd osztályok találhatók. Az egymással előd-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 előd 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

Adatszerkezetek reprezentációja. Műveletek adatszerkezetekkel. Adatszerkezetek osztályozása és jellemzésük. Szekvenciális adatszerkezetek: sor, verem, lista, sztring. Egyszerű és összetett állományszerkezetek.

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

Relációs, ER és objektumorientált modellek jellemzése. Adatbázisrendszer. Funkcionális függés. Relációalgebra és relációkalkulus. Az SQL.

6.1. Relációs, ER és objektumorientált modellek jellemzése.

6.1.1. Bachmann-féle fogalomrendszer

	Absztrakt	Konkrét
Egyed	Egyedtípus	Egyed-előfordulás
Tulajdonság	Tulajdonságtípus	Tulajdonság-előfordulás
Kapcsolat	Kapcsolattípus	Kapcsolat-előfordulás

Egyed A valós világnak az az eleme (tárgy, jelenség, elképzelés, személy, fogalom stb.), amely a modellezés tárgyát képezi.
Egyedtípus Az azonos tulajdonságtípusokkal rendelkező egyedek absztrakciója.
Tulajdonság Az egyednek a modellezés szempontjából lényeges jellemzője.
Tulajdonságtípus Az azonos szerepű tulajdonságok absztrakciója. A tulajdonságtípusok (attribútumok) osztályozása:

1. a tulajdonság-előfordulás szerkezete (összetettsége) szerint
   - egyszerű (atom)
   - összetett
2. a tulajdonság-előfordulás hány értéket vehet föl egyszerre
   - egyértékű
   - halmazértékű (többértékű)
3. a tulajdonság-előfordulás minden esetben megjelenik-e a háttértárolón (a fizikai adatbázisban)
   - tárolt
   - származtatot

kapcsolat A két vagy több egyedtípus egyedei között fennálló viszony.
Kapcsolattípus Két vagy több egyedtípus közötti jól meghatározott viszony. Osztályozása:

1. A kapcsolat **foka** meghatározza, hogy hány egyedtípus vesz részt a kapcsolatban, eszerint beszélhetünk bináris (másodfokú), ternáris (harmadfokú),... kapcsolatról;
2. A (bináris) kapcsolat **számossága** megmondja, hogy legfeljebb hány kapcsolat-előfordulásban vehet részt egy egyed-előfordulás. 3 fajtája lehet: 1:1, 1:N, M:N;
3. A (bináris) kapcsolat **szorossága** meghatározza, hogy a kapcsolatban részt vevő egyedtípusok minden egyedének részt kell-e vennie legalább egy kapcsolat-előfordulásban. Eszerint a kapcsolat lehet kötelező, félig kötelező illetve opcionális.

6.1.2. Relációs modell

A relációs modell a legelterjedtebb modell. Ez köszönhető egyrészt annak, hogy a személyi számítógépek elterjedésekor a megjelenő adatbázis-kezelő rendszerek ezen a modellen alapultak, valamint jól formalizált matematikai háttérrel rendelkezik.
A **tartomány (domain)** atomi értékek halmaza. Rendelkezik névvel, típussal és formátummal Jele: **D**
Az **attribútum** kijelöli az adott tartomány szerepét valamely rendszer esetén. Jele: **A**
A **relációs séma** ekkor az R(A 1 , ..., A n) módon formalizálható, ahol R a séma neve, míg A i -k attribútumok.
A relációs séma **foka** az attribútumainak száma.

A reláció ekkor a következő formalizmussal értelmezett:

$$r = r(R), \quad \text{ahol } r \subset dom(A_1) \times \cdots \times dom(A_n)$$

A reláció az *egyedtípusnak* feleltethető meg Az érték n-esek az *egyed-előfordulások*. Szokták **rekordnak** is nevezni. A reláció kétdimenziós táblázattal szemléltethető. A táblázat oszlopait az attribútumok címkézik, a sorokban pedig a rekordok helyezkednek el. Megjegyzendő, hogy logikai szinten a rekordoknak nincs sorrendjük, a fizikai megvalósítás során mégis szükséges valamilyen sorrend felállítása.

A relációs modell megszorításai

Tartomány megszorítás a tartomány elemei atomiak.

Kulcs megszorítás *Szuper kulcsnak* nevezzük az olyan attribútum halmazt, amelynél nincs két olyan rekord, ahol az attribútumok rendre megegyeznek. Szuper kulcs mindig van. (Ha más nincs, akkor maga a rekord.) Az elsődleges kulcs olyan halmaz, ami szuper kulcsot alkot, de bármely attribútumot elhagyva belőle már nem kapunk szuper kulcsot. (minimális szuper kulcs) Legyen R_1 és R_2 két reláció. Az R_1 reláció F_k-val jelölt attribútum halmazát külső kulcsnak nevezzük, ha az F_k-hoz tartozó attribútumok tartományai megegyeznek a R_2 elsődleges kulcsának tartományaival, és ha teljesül, hogy valamely R_1-beli rekord esetén vagy megegyezik valamely R_2-beli rekord elsődleges kulcsával, vagy NULL. (A kapcsolatok realizálják a külső kulcsok.)

Integritási megszorítások

Egyedintegritási megszorítás Az elsődleges kulcs nem lehet NULL értékű.

Hivatkozási integritási megszorítás Egy R_1 relációséma FK-val jelölt attribútumhalmaza külső (idegen) kulcsa R_1-nek, amely hivatkozik az R_2 relációsémára, ha eleget tesz a következő feltételeknek:

- Az FK-beli attribútumoknak és az R_2 PK-val jelölt elsődleges kulcsattribútumainak páronként azonos a tartománya; ekkor azt mondjuk, hogy az FK attribútumok hivatkoznak az R_2 relációsémára.
- Bármely $r_1(R_1)$ aktuális állapotának egy t_1 rekordjában egy FK-beli érték vagy megjelenik egy $r_2(R_2)$ aktuális állapotának valamely t_2 rekordjában PK értékeként, vagy az értéke NULL. Az előbbi esetben $t_1[FK] = t_2[PK]$, ekkor azt mondjuk, hogy a t_1 rekord hivatkozik a t_2 rekordra.

6.1.3. ER modell

- egyed - kapcsolat modell
- a leggazdagabb modell
- sématervező eszköz, objektum alapszó, magasszintű eszköz, az 1. szemantikus modell
- koncepciós szinten tervezünk és grafikusan is megjelenik a modell

Megjegyzés. *Nincsen ER alapú adatbázis kezelő, ami azt jeleni, hogy le kell képezzük relációs sémává az ER modellt.*

Definíció (Egyedtípusok)**.** *Azokat az egyedtípusokat, amelyek nem rendelkeznek saját kulcsattribútumokkal,* ***gyenge egyedtípusoknak*** *nevezzük. Ezzel ellentétben azokat a (hagyományos) egyedtípusokat, amelyeknek van kulcsattribútumuk,* ***erős egyedtípusoknak*** *nevezzük.*

Definíció. *A gyenge egyedtípusoknak* ***részleges kulcsuk*** *(*diszkriminátoruk*) van, amely azon attribútumok halmaza, amelyek egyértelműen azonosítják azokat a gyenge egyedeket, amelyek ugyanazon tulajdonos egyed(ek)hez kapcsolódnak.*

ER séma leképezése relációs sémára

1. Erős egyedtípusok leképezése
   Az ER séma minden E erős egyedtípusához rendeljünk hozzá egy R relációsémát, amely tartalmazza E összes egyszerű attribútumát. Az összetett attribútumoknak csak az egyszerű komponenseit adjuk hozzá R attribútumaihoz. Válasszuk E kulcs attribútumainak egyikét az R relációséma elsődleges kulcsául. Ha az E-ből választott kulcs összetett, akkor annak egyszerű attribútumai együttesen fogják alkotni R elsődleges kulcsát.
2. Gyenge egyedtípusok leképezése
   Az ER séma minden W gyenge egyedtípusához rendeljünk hozzá egy R relációsémát, melynek attribútumai legyenek W összes egyszerű attribútuma és W összetett attribútumainak egyszerű komponensei. Továbbá adjuk hozzá R attribútumaihoz külső kulcs attribútumként azoknak a relációsémáknak az elsődleges kulcs attribútumait, amelyeket a domináns egyedtípusoknak feleltettünk meg; ezzel képezzük le a W -hez tartozó azonosító kapcsolattípust. R elsődleges kulcsa a tulajdonos egyedtípusok elsődleges kulcsainak és a W gyenge egyedtípus diszkriminátorának az együttese.
3. Bináris 1:1 számosságú kapcsolattípusok leképezése
   Minden bináris 1:1 számosságú R kapcsolattípus esetén meg kell határozni az R-ben részt vevő egyedtípusokból képzett S és T relációkat. Három lehetséges megközelítés létezik: (1) külső kulcs használata, (2) összevonás és (3) keresztreferencia vagy kapcsoló reláció használata. Az első megközelítés a leghasznosabb, és azt célszerű alkalmazni, hacsak bizonyos feltételek nem állnak fenn, ahogy azt mindjárt látni fogjuk:
   (a) külső kulcs használata
   Válasszuk ki az egyik relációt (mondjuk S-t) és vegyük fel S külső kulcsaként T elsődleges kulcsát. Célszerű S-nek azt

a relációt választani, amelyiket abból az egyedtípusból képeztünk le, amelyik totális résztvevője az R kapcsolatnak. Vegyük fel továbbá R egyszerű attribútumait, illetve R összetett attribútumainak egyszerű komponenseit S attribútumaiként. (Azt is megtehetnénk, hogy S (a totális résztvevő) elsődleges kulcsát vesszük fel T külső kulcsaként, de ebben az esetben T minden olyan rekordjánál, amelyik nem vesz részt a kapcsolatban, ehhez a külső kulcshoz null értéket kellene rendelni.)

(b) összevonás
Egy másik lehetőség az 1:1 kapcsolatok leképezésére, ha a két egyedtípust és a kapcsolatot egyetlen relációba vonjuk össze. Ezt akkor tehetjük meg, ha mindkét egyedtípus totális résztvevője a kapcsolatnak.

(c) kereszthivatkozás v. kapcsoló reláció használata
A harmadik lehetőség, hogy felveszünk egy harmadik R relációt abból a célból, hogy kereszthivatkozással lássuk el a két egyedtípusból képzett S és T relációk elsődleges kulcsait. Ahogy látni fogjuk, ezt a megközelítést alkalmazzuk a bináris M:N kapcsolatoknál is. Az R relációt kapcsoló relációnak nevezzük, mert R minden rekordja egy kapcsolat-előfordulást reprezentál, amely S egy rekordját T egy rekordjával kapcsolja össze.

4. Bináris 1:N számosságú kapcsolattípusok leképezése
Minden bináris 1:N számosságú R kapcsolattípus esetén meg kell határozni azt az S relációt, amelyiket a kapcsolattípus N-oldali egyedtípusából képeztünk. Vegyük fel S külső kulcsaként az R-ben részt vevő másik egyedtípusból képzett T reláció elsődleges kulcsát; mindezt azért tesszük, mert az N-oldali egyed-előfordulások a másik oldalról legfeljebb egy egyed-előforduláshoz tartoznak. Vegyük fel továbbá R egyszerű attribútumait, illetve R összetett attribútumainak egyszerű komponenseit S attribútumaiként.
5. Bináris M:N számosságú kapcsolattípusok leképezése
Minden bináris M : N számosságú R kapcsolattípus esetén hozzunk létre egy új S relációt, amely R-et reprezentálja. Vegyük fel S külső kulcsaként a kapcsolatban részt vevő egyedtípusokból képzett relációk elsődleges kulcsait; ezek együttese alkotja S elsődleges kulcsát. Vegyük fel továbbá R egyszerű attribútumait, illetve R összetett attribútumainak egyszerű komponenseit S attribútumaiként. Egy M:N kapcsolatot nem tudunk egyetlen külső kulccsal reprezentálni az egyik résztvevő relációban: létre kell hoznunk egy külön S kapcsoló relációt
6. Többértékű attribútumok leképezése
Minden egyes A többértékű attribútum esetén hozzunk létre egy új R relációt. Ez az R reláció tartalmazzon egy, az A-nak megfelelő attribútumot, valamint annak a relációnak a K elsődleges kulcsát – R külső kulcsaként –, amelyet az A-t tartalmazó egyedtípusból vagy kapcsolattípusból képeztünk. R elsődleges kulcsát A és K együttese alkotja. Ha a többértékű attribútum összetett, akkor az egyszerű komponenseit vegyük fel R attribútumaiként.
7. N-ed fokú kapcsolattípusok leképezése
Minden N-edfokú R kapcsolattípus esetén, ahol $N > 2$, hozzunk létre egy új S relációt, amely R-et reprezentálja. Vegyük fel S külső kulcsaként a kapcsolatban részt vevő egyedtípusokból képzett relációk elsődleges kulcsait. Vegyük fel továbbá R egyszerű attribútumait, illetve R összetett attribútumainak egyszerű komponenseit S attribútumaiként. S elsődleges kulcsa általában az összes külső kulcs együttese. Ha azonban az R-ben részt vevő valamely E egyedtípusból csak egy rekord vehet részt a kapcsolatban (számossági megszorítás), akkor S elsődleges kulcsának nem kell tartalmaznia az E-ből képzett E 0 relációra hivatkozó külső kulcsot.

6.1.4. OO modell

Az adatbázis tervezésben már elérhető az OO paradigma is. Az ODL egy példa az objektumorientált definíciós nyelvekre. A tervezés során a valós világ egyedeit objektumoknak tekintjük.

az objektum négy jellemzője

1. azonosító (OID)
2. név (adható neki egyedi név egy adatbázison belül, ezzel hivatkozhatunk rá)
3. élettartam (perzisztens vagy tranziens)
4. struktúra -> milyen típus konstruktorok segítségével hoztuk létre az objektumot.

Az azonos típusú attribútumokkal rendelkező fogalmak tartoznak egy osztályba. Az osztályok rendelkeznek attribútumokkal, metódusokkal és kapcsolatokkal. Az osztály definíciója:

```
interface osztálynév [:szuper típus]
( [key kulcsattribútumok] )
{
  attribútumok
  kapcsolatok
  metódusok
}
```

Az attribútumaik lehetnek egyszerűek (atomiak), illetve különféle kollekció típusúak.(halmaz, lista, struktúra, tömb) Megadásuk: attribute típus név;

A kapcsolatoknak nincs attribútumuk, és nem lehet sokágú kapcsolatokat létrehozni. Ezek megvalósításához egy új, a kapcsolatot reprezentáló osztályt kell létrehozni. Definiálható viszont egy, és kétirányú kapcsolat is. Megadásuk: relationship típus név [inverse inverzkapcsolat];

A metódusok az objektumra alkalmazható műveleteket írják le. A modell ezeknek csak a szignatúráját kezeli, a megvalósításukat nem. Megadásuk: [típus] függvénynév(paraméterek);

Az ODL-ben az elsődleges kulcs egy vagy több olyan attribútum, amelynél nincs két olyan objektum, melyre ezek mindegyike megegyezik. Egy attribútum: egyszerű; több attribútum: összetett kulcs.

Az OO paradigmának megfelelően az osztályok kiterjeszthetőek. Ekkor a szülő osztály minden attribútuma, kapcsolata és metódusa átkerül az alosztályba. Egy osztálynak több szülője is lehet.

ODMG szabvány /Object Data Managament Group/ objektumorientált adatbázisrendszerekkel foglalkozó cégek tömörülése 1993: az első ilyen szabvány megjelenése Több részből áll a szabvány és a modell is:

- objektummodull leírása
- objektum definíciós nyelv (ODL)
- objektum lekérdező nyelv (OQL)
- kapcsolódások az OO programnyelvekhez / C++, Java, Smalltalk /

6.2. Adatbázisrendszer.

Az adatbázisrendszer számítógépekből, adatokból, kezelő szoftverből és kezelőkből áll. Az adatbázis a **meta-adatbázisból** és a **fizikai adatbázisból** tevődik össze. A meta-adatbázis az adatbázis szerkezetére vonatkozó információk összessége, a fizikai adatbázis pedig az egyed-előfordulások összessége. A kezelő rendszer valamilyen szoftver (**DBMS**), mindig konkrét operációs rendszer alatt fut, gyakran sok funkciót átvesz tőle. Programozhatóság szerint létezik **saját nyelvű** (része valamilyen programozási eszközrendszer, jellemzően eljárásorientált, imperatív jellegű) és **befogadó nyelvű** (utasításokkal rendelkezik lekérdezésekhez, manipulációhoz, programozáshoz magas szintű nyelven írt programot kell hozzáilleszteni). A korábbi rendszerek jellemzően befogadóak voltak. A saját nyelv általában kevesebbet tud, mint a magas szintű nyelvek.

A felhasználó tényleges adatokhoz csak a DBMS-en keresztül férhet, az operációs rendszert használva. A fizikai adatbázist kezelő utasítások összessége az **adatmanipulációs nyelv** (DML), míg a szerkezet (séma) leírására alkalmas nyelvrész az **adatleíró nyelv** (DDL). Mellettük létezik még az **adatvezérlő nyelv** (DCL), mely a jogosultságokkal, tranzakciókkal kapcsolatos utasításokat tartalmazza. A teljes adatbázisrendszer felügyeletéért felelős személyek az **adminisztrátorok** (DBA).

Az egyes felhasználói csoportok aszerint különíthetőek el, hogy hozzáférésük során mi jellemző munkájukra. Az **eseti felhasználók** eseti kérdéseket tesznek fel, optimális eszközrendszerük interaktív, kérdései véletlenszerűek és nagyon különböző információkra vonatkoznak. A **parametrikus felhasználók** szűk, előre meghatározott kérdéskörben tesznek fel kérdéseket, eszközrendszerük jól programozható, paraméterezhető alkalmazást futtatnak. A szakemberek ismerik a DBMS lehetőségeit, alkalmazásokat írnak és fejlesztenek tovább, nagy mennyiségű adattal dolgoznak.

Egy adatbázisrendszeren belül a legfontosabb erőforrás az adat. **Cél:** a felhasználók optimális kiszolgálása az adatbázisrendszeren belül.

Az adatbázis architektúra *fizikai (belső)*, *logikai (külső)* és *koncepcionális* szintekből áll. Fizikai szinten az adatok fizikai elhelyezkedését és az elérési módokat kell leírni. Koncepcionális szinten az adatbázissal, mint logikai egységgel kell foglalkozni, a sémát kell leképezni fizikai szintre (rekordtípusok, tulajdonságtípusok, hozzáférések megadása). Logikai szinten jelenik meg a felhasználói csoportokra történő felosztás, az alsémák megadják, hogy a egyes csoportok mit látnak az adatbázisból.

Az adatbázis-építés során először elemzés, tervezés, modellezés útján kell eljutni sémához és alsémákhoz koncepcionális szinten, majd e sémát leírva meta adatbázisba kerül (üres adatbázis), fel kell tölteni a fizikai adatbázist, végül lekérdezések útján nyerhetők ki a szükséges információk.

6.3. Funkcionális függés.

Definíció (1). *Legyen $R(A_1, A_2 \dots A_n)$ relációséma és X, Y az $\{A_1, A_2 \dots A_n\}$ attr. halmaz részhalmazai. X-től funkcionálisan függ Y, ha bármely R feletti T tábla esetén valahányszor két sor megegyezik X-n, akkor megegyezik Y-n is. Jele: $X \to Y$*

Definíció (2). *Az R két attribútumhalmaza, X és Y között, $X \to Y$-nal jelölt funkcionális függés előír egy megszorítást azokra a lehetséges rekordokra, amelyek egy R fölötti r relációt alkothatnak. A megszorítás az, hogy bármely két, r-beli t_1 és t_2 rekord esetén, amelyekre $t_1[X] = t_2[X]$ teljesül, teljesülnie kell $t_1[Y] = t_2[Y]$-nak is.*

Megjegyzés. *Két attribútumhalmaz közötti összefüggést/kapcsolatot írja le. Legyen X és Y az R reláció két attribútumhalmaza! Y funkcionálisan függ X-től (vagy X funkcionálisan meghatározza Y-t), ha bármelyik két rekord esetén abból, hogy a két rekord X attribútum-értékei megegyeznek következik, hogy az Y attribútum-értékei is megegyeznek.*

Megjegyzés. *A funkcionális függés nem két irányú kapcsolat!*
Ha $X \to Y$ és $Y \to X$ egyszerre áll fenn, akkor azt mondjuk, hogy X és Y ***kölcsönös funkcionális függésben*** *áll egymással.*

Tétel (A funkcionális függés tulajdonságai)**.**

1. *reflexivitás: Ha$X \subseteq Y$, akkor$X \to Y$*
   Egy attribútumhalmaz mindig meghatározza önmagát, vagy saját maga bármilyen részhalmazát.
   $X = \{A_1, A_2, A_3, A_4\}$
   $Y = \{A_2, A_3\}$
   $t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$
2. *augmentivitás:* $\{X \to Y\} \implies XZ \to YZ$
   egy funkcionális függés mindkét oldalának ugyanazzal az attribútumhalmazzal történő bővítése újabb érvényes funkcionális függést eredményez.
3. *tranzitivitás:* $\{X \to Y, Y \to Z\} \implies X \to Z$
   a funkcionális függések tranzitívak.
4. *dekompozíció:* $\{X \to YZ\} \implies X \to Y$
   egy funkcionális függés jobb oldaláról eltávolíthatunk attribútumokat.
5. *additivitás szabályai:* $\{X \to Y, X \to Z\} \implies X \to YZ$
   funkcionális függések egy $\{X \to A_1, X \to A_2, \ldots, X \to A_n\}$ ***halmazát összevonhatjuk egyetlen*** $X \to \{A_1, A_2, \ldots, A_n\}$ ***funkcionális függéssé.***
6. *pszeudotranzitivitás:* $\{X \to Y, WY \to Z\} \implies WX \to Z$

Definíció (Triviális függés)**.** *Egy $X \to Y$ funkcionális függés* ***triviális****, ha $X \subseteq Y$ (Y részhalmaza X-nek), egyébként nemtriviális.*

Tétel (Triviális függés)**.** *Egy $X \to Y$ funkcionális függés* ***teljesen nem triviális****, ha $X \cap Y = \emptyset$ (X és Y metszete üres halmaz).*

6.4. Relációalgebra és relációkalkulus.

A relációalgebra egy absztrakt kezelő nyelv. Minden relációs algebrai művelet eredménye egy reláció lesz.

6.4.1. Unáris műveletek

Szelekció (σ) Egy adott reláció rekordjainak egy részhalmazát adja, amely részhalmazba egy feltételt teljesítő rekordok fognak beletartozni. Ez a feltétel a szelekciós feltétel, ami általában egy logikai kifejezés. Ebben a kifejezésben hasonlító és logikai műveleti jelek lehetnek. Formálisan a műveletben attribútumok és konstansok vannak.

$$\sigma_{\text{feltétel}}(\text{relációs\_séma\_név})$$

Az eredményül kapott reláció fokszáma megegyezik a kiindulási reláció fokszámával. A kapott rekordok száma kisebb vagy egyenlő mint az induló rekordok száma.
pl.: σ feltétel$_{=4\text{OR DNO}=5}(EMPLOYEE)$
A szelekció kommutatív művelet: $\sigma_{f1}(\sigma_{f2}(R)) = \sigma_{f2}(\sigma_{f1}(R))$
Lényeges az a szemlélet a konkrét megvalósításnál, hogy szelekció a rekordokon egyenként hajtódik végre.
Szelekciósorozat mindig értelmezhető egyetlen szelekcióként: $\sigma_{f1}(\sigma_{f2} \ldots (\sigma_{fn}(R)) \ldots) = \sigma_{f1\text{AND}f2\text{AND}\ldots fn}(R)$
A szelekció logikai szinten : a táblázatból kiemelek bizonyos sorokat.
SQL: `SELECT * FROM R WHERE szelekciós feltétel`

Projekció (π) A kiinduló relációból kiválaszt bizonyos attribútumokat, és az eredményrelációba csak ezen attribútumokhoz tartozó értékeket viszi át.

$$\pi_{\text{attribútum\_lista}}(\text{reláció\_séma\_név})$$

Logikai szint: a táblából oszlopok kiválasztása.
pl.: $\pi_{\text{FNAME,LNAME,BDATE}}$(EMPLOYEE) Az attribútumlistán az attribútumok sorrendje tetszőleges, de a keletkezett reláció attribútumainak sorrendjét meghatározza a kiindulási attribútumok sorrendje. Az

eredményreláció fokát a kiindulásban felsorolt attribútumok határozzák meg. Rekordszám:
Ha a kiválasztott attribútumok között van kulcs, akkor az induló rekordok száma egyenlő az eredményrekordok számával. Ha nem szerepel kulcs, akkor az eredményrekordok száma kevesebb lehet, mint az induló rekordok száma. (Mivel nincs duplikálás, - azonos rekordok esetén a projekció csak egyet választ ki.)
Logikai szinten megengedi a multihalmazt.
SQL: `SELECT attribútumlista FROM R`

Átnevezés (ρ) Megváltoztathatjuk a relációnk jelölését és átnevezhetjük az attribútumait is értékadás végrehajtásakor

$$\rho_{S(B_1,B_2,\ldots,B_n)}(R)\ \rho_S(R)\ \rho_{(B_1,B_2,\ldots,B_n)}(R)$$

Az S a reláció jelölésére használt új szimbólum,$B_1, B_2, \ldots, B_n$ az új attribútumnevek. Az átnevezés *unáris* művelet. Az új S reláció *foka*, *számossága* megegyezik R-ével. S sémája megegyezik R sémájával vagy a $B_1, B_2, \ldots, B_n$ attribútumok által meghatározott séma lesz.

Halmazműveletek A relációkat rekordok halmazának tekintjük, ezen hajtjuk végre a műveleteket $\rightarrow$ újból reláció.

Unió ($\cup$) az új relációban a két rekordban található elemek uiója lesz benne. Kommutatív, Asszociatív.

Metszet ($\cap$) a két relációnak unió-kompatibilisnek[6] kell lennie. A szokásos halmazelméleti metszet. Kommutatív, Asszociatív.

Különbség ($-$ vagy $\setminus$) A szokásos halmazelméleti kivonás. Nem kommutatív.

Descartes–szorzat ($\times$) a halmazelméleti értelemben vett Descartes szorzat. Ehhez nem szükséges az unió-kompatibilitás.

Összekapcsolás (join) Két reláció összekapcsolása

Általános összekapcsolás (theta join, $\bowtie$) Bináris művelet, operandusai $R(A_1, A_2, \ldots, A_n)$ és $S(B_1, B_2, \ldots, B_m)$

$$R \bowtie_{sszekapcsolsi\_felttel} S$$

Az eredmény $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ fokszáma $n + m$. Q-ban bennel lesz az R és az S relációk rekordjainak minden olyan kombinációja, amely kielégíti az összekapcsolási feltételt.
Összekapcsolási feltétel: $\langle$feltétel$\rangle$AND ... And$\langle$feltétel$\rangle$, ahol mindegyik feltétel $A_i \Theta B_j$ alakú ($\Theta \in \{=, \neq, <, >, \leq, \geq\}$), A_i és B_j tartománya megegyezik.
SQL: `SELECT * FROM R, S WHERE összekapcsolási_feltétel`

egyenlőség alapú összekapcsolás (equijoin) Olyan általános összekapcsolási műveletet, amelynek összekapcsolási feltételében csak az egyenlőségjel ($=$) szerepel összehasonlító műveleti jelként. Az egyenlőségen alapuló összekapcsolás eredményeként kapott reláció minden rekordjában van legalább egy pár azonos érték.
SQL: `SELECT * FROM R [INNER] JOIN S ON R.ID = S.ID`

természetes összekapcsolás (natural join, $*$) A természetes összekapcsolás műveletét az egyenlőségen alapuló összekapcsolás műveletéből származtatjuk oly módon, hogy az ott kapott relációból eltávolítjuk az összekapcsolás alapjául szolgáló, a hozzájuk tartozó értékek egyenlősége miatt felesleges attribútumok egyikét. Az összekapcsolandó két relációban az összekapcsolás alapjául szolgáló attribútumok nevének meg kell egyezniük.

$$R * S$$

SQL: `SELECT * FROM R NATURAL JOIN S`

bal oldali/jobb oldali/teljes külső összekapcsolás (left/right/full outer join,) Legyen R egy n-ed fokú S egy m-ed fokú reláció, R és S természetes összekapcsolását úgy képezzük, hogy az RxS-ből meghagyjuk azokat a sorokat ahol azonos attribútum azonos értékű.

Tétel (A relációalgebrai műveletek teljes halmaza)**.** *Bebizonyítható, hogy a relációalgebrai operátorok $\{\sigma, \pi, \cap, -, \times\}$ halmaza teljes halmaz, azaz bármelyik másik relációalgebrai művelet kifejezhető ezen halmazbeli operátorokkal végzett műveletek sorozataként.*

6.4.2. Relációkalkulus

A reláció kalkulus egy elsőrendű logikán alapuló lekérdező nyelv.

[6] Az uniókompatibilitás azt jelenit, hogy adott két relációnak ugyanannyi attribútuma van, és azok tartományai páronként megegyeznek egymással

rekord alapú (soralapú) relációkalkulus (Tuple Relation Calculus – TRC) formuláiban rekordváltozók és attribútumaik szerepelnek. A formulák elején az eredménybe kerülő attribútumok szerepelnek (amiket az ABC kisbetűivel jelölünk), a végén a feltételek, hogy milyen sorokból. Atomi formulái sor attribútumértéke hasonlító operátor másik sor attribútumértéke vagy (konstans) érték formában épülnek fel. Formulái atomi formulákból képezhetőek logikai összekötőjelekkel és kvantorokkal.

tartomány alapú relációkalkulus tartományváltozókkal (attribútumok) dolgozik. Formuláiban elöl az attribútumok, hátul a feltételek szerepelnek. Atomi formulái tartományváltozó, hasonlító operátor, érték vagy tartományváltozó. Formulái atomi formulákból képezhetőek logikai összekötőjelekkel és kvantorokkal.

6.4.3. Normálformák

Első normál forma (1NF) Egy reláció első normál formában van, ha minden attribútuma egyszerű, nem összetett adat.

Második normál forma (2NF) Az első normál forma nem elegendő feltétel a redundanciák megszüntetésére. Egy reláció második normál alakjában nem tartalmazhat tényeket a reláció kulcs egy részére vonatkozóan. A második normál forma definíciója két feltétellel írható le.

- A reláció első normál formában van
- A reláció minden nem elsődleges attribútuma funkcionálisan függ az elsődleges kulcstól.

Harmadik normál forma (3NF) A második normál formájú relációkban nem lehetnek olyan tények, amelyek a reláció kulcs részeihez kapcsolódnak. Azonban ennek ellenére is lehet bennük redundancia, ha olyan tényeket tartalmaznak, amelyek a nem elsődleges attribútumokkal állnak kapcsolatban. Ezt a lehetőséget szünteti meg a harmadik normál forma. Egy reláció harmadik normál formában van, ha

- A reláció második normál formában van.
- A reláció nem tartalmaz funkcionális függőséget a nem elsődleges attribútumok között.

Boyce/Codd normál forma (BCNF) A normál formák tárgyalása során eddig olyan relációkra mutattunk példákat, melyeknek csak egy reláció kulcsa van. A normál formák definíciója természetesen alkalmazható a több kulccsal rendelkező relációkra is. Ebben az esetben minden attribútum, mely valamely kulcsnak a része, elsődleges attribútum, de ez az attribútum függhet egy másik, ezt nem tartalmazó kulcs részétől. Ha ez a helyzet fennáll, redundanciát tartalmaz a reláció. Ennek a felismerése vezetett a harmadik normál forma egy szigorúbb definíciójához, a Boyce/Codd normál formához.

- Minden elsődleges attribútum teljes funkcionális függőségben van azokkal a kulcsokkal, melyeknek nem része

Negyedik normál forma (4NF) Egy reláció negyedik normál formában van, ha 3. normál formában van, és legfeljebb egy többértékű függés[7] van benne. Másként: Az összetett azonosító egyik része sem függ a másiktól, csak az összetett azonosító egészétől.

Ötödik normál forma (5NF) Az összetett azonosító nem okoz pszeudotranzitív funkcionális függést.

A harmadik normálformáig mindenféleképpen érdemes normalizálni a relációkat. Ez a redundanciák nagy részét kiszűri. Azok az esetek, melyekben a negyedik illetve az ötödik normál formák alkalmazására van szükség, ritkábban fordulnak elő. Az ötödik normál forma esetén a redundancia megszüntetése nagyobb tároló terület felhasználásával lehetséges csak. Így általában az adatbázis tervezője döntheti el, hogy az ötödik normál formát és a nagyobb adatbázist vagy a redundanciát és a komplikáltabb frissítési, módosítási algoritmusokat választja.

6.5. Az SQL.

Az SQL (Structured Query Language) szabványosított struktúrált lekérdezőnyelv, amely a relációs adatmodell alapján felépülő adatbázisok kezelésére képes. Az SQL olyan nyelv, mely platformfüggetlen. Részei:

DDL – adatdefiníciós nyelv Az adatdefiníciós nyelv segítségével hozhatjuk létre, illetve szüntethetjük meg az adatbázist és az azt felépítő relációkat, az indexeket, illetve a nézet táblákat. Azon utasítások, amelyekkel a sémát le tudjuk írni, azaz kezelni tudjuk a metaadatbázist. Parancsai: CREATE, ALTER, DROP

[7] Többértékű függőség: $X \rightarrow\rightarrow Y$ függőséget kielégíti egy R reláció, ha $\forall t_1, t_2 \in R \exists t_3 \in R$ úgy, hogy: $1) t_1[X] = t_2[X]$ $2) t_3[XY] = t_1[XY]$ $3) t_3[R-XY] = t_2[R-XY]$. Következménye: $\exists t_4 \in R$ úgy, hogy: $1) t_4[XY] = t_2[XY]$ $2) t_4[R-XY] = t_1[R-XY]$

DML – adatmanipulációs nyelv Az adatmanipulációs rész biztosítja a relációk feltöltését, az attribútumok módosítását és a sorok törlését. Azon utasítások, melyek a fizikai adatbázist kezelik. (Bővítés, törlés, módosítás, csere, lekérdezés). Utasításai: SELECT, INSERT, DELETE, UPDATE

DCL – adatvezérlő nyelv Tranzakciós és jogosultságokkal kapcsolatos utasítások. Az adatbázissal kapcsolatos alkalmazások vezérlését valósítja meg. Utasításai: GRANT, REVOKE, COMMIT (utasítások véglegesítése), ROLLBACK (visszagörgetés az utolsó savepoint-hoz)

6.5.1. parancsok szintaktikája

Adatbázis létrehozása:

```
CREATE DATABASE név;
```

Tábla létrehozása:

```
CREATE TABLE táblanév (
oszlopnév adattípus (méret) [,
oszlopnév adattípus (méret), ... ] );
```

Tábla törlése:

```
DROP TABLE név;
```

Tábla módosítása: -új mező hozzáadása

```
ALTER TABLE táblanév
ADD ( oszlopnév adattípus (méret) [,
oszlopnév adattípus (méret),...] );
```

- meglévő oszlop törlése:

```
ALTER TABLE táblanév
DROP oszlopnév;
```

- meglévő oszlop módosítása:

```
ALTER TABLE táblanév
MODIFY ( oszlopnév adattípus (méret) [,
oszlopnév adattípus (méret),...] );
```

Új sor beszúrása:

```
INSERT INTO táblanév
[ ( oszlopnév lista) ]
VALUES ( értéklista );
```

Sorok módosítása:

```
UPDATE táblanév
SET oszlopnév = kifejezés
[, oszlopnév = kifejezés ...]
[WHERE feltétel];
```

Sorok törlése:

```
DELETE FROM táblanév
[WHERE feltétel]
```

Lekérdezés:

```
SELECT [DISTINCT | ALL] oszlopnév, ...
[INTO vátozónév lista]
[FROM tábla lista, ...
[WHERE feltétel]
[GROUP BY oszlopnév, ...]
[HAVING feltétel]
[UNION | INTERSECT | MINUS SELECT ...]
[ORDER BY oszlopnév | oszlop-sorszám, ... [ASC|DESC]]
];
```

Jogosultság adása:

```
GRANT jogosultság_1,... ON objektum
TO felhasználó_1,...
WITH GRANT OPTION;
```

Jogosultság visszavonása:

```
REVOKE jogosultság_1,... ON objektum
FROM felhasználó_1,... ;
```

7. Hálózati architektúrák

Az ISO OSI hivatkozási modell. Ethernet szabványok. A hálózati réteg forgalomirányító mechanizmusai. Az internet hálózati protokollok, legfontosabb szabványok és szolgáltatások.

7.1. Az ISO OSI hivatkozási modell.

Az ISO Nemzetközi Szabványügyi Szervezet által kidolgozott referenciamodellt (ISO OSI hivatkozási modell, röviden: OSI modell) értik alatta, amely a különbözõ nyílt hálózatok, rendszerek összekapcsolására vonatkozik (olyan rendszerek, amelyek nyitottak más rendszerekkel való kommunikációra). A modell szerint a hálózatot legjobban úgy lehet megvalósítani, hogy azt a feladatok szerint egymástól független különbözõ rétegekre osztjuk, ahol aztán az egyes rétegek egymással kommunikálnak. A modell hét réteget különböztet meg. Ezek a rétegek aztán a rájuk vonatkozó protokollok szerint végzik a feladatukat.

Az OSI modell a hálózatot mintegy réteges tortát képzeli el. Ennek a tortának hét rétege van. Szakszerûen azt mondanánk, hogy protokoll-veremrõl van szó. Minden rétegnek megvan a maga protokoll-készlete, amelyek a réteg feladatát hivatottak szabályozni. Az egyes rétegek egymással csak logikai kapcsolatban állnak, fizikailag mindig az alattuk lévõ réteggel kommunikálnak. A két hálózat tényleges fizikai kapcsolatát csak a fizikai réteg adja. Az alsó négy réteg azzal foglalkozik, hogy hogyan kell egy üzenetet a hálózat két pontja között továbbítani.

1. Fizikai réteg (physical layer) Elektromos és mechanikai jellemzők procedurális és funkcionális specifikációja két (közvetlen fizikai összeköttetésű) eszköz közötti jeltovábbítás céljából. A bitátvitel megvalósítása a legfontosabb feladata.
Adategység: bit

2. Adatkapcsolati réteg (data-link layer) Megbízható adatátvitelt biztosít egy fizikai összeköttetésen keresztül. Ezen réteg legfontosabb funkcionalitása a közeghozzáférés (egy csatorna megosztása a rá kapcsolt állomások közt), problémaköréhez tartozik a fizikai címzés, hálózati topológia, közeghozzáférés, fizikai átvitel hibajelzése és a keretek sorrendhelyes kézbesítése. Az IEEE két alrétegre (MAC- Medium Access Controll, LLC – Logical Link Controll) bontotta az adatkapcsolati réteget.
Adategység: keret

3. Hálózati réteg (network layer) Összeköttetést és útvonalválasztást biztosít a nem közvetlen összeköttetésű hálózati csomópontok közt. Világméretben bárhonnan bárhová lehessen küldeni információt. Redundáns útvonalak is lehetnek. Ezek közül a 3. rétegnek kell választania.
Adategység: csomag

4. Szállítási réteg (transport layer) Megbízható hálózati összeköttetést létesít két csomópont között. Feladatkörébe tartozik pl.: a virtuális áramkörök kezelése, átviteli hibák felismerése/javítása és az áramlásszabályozás. Az egyik oldalon bemenő bitfolyamnak kell kijönnie a másik oldalon.
Adategység: Szegmens vagy TPDU

5. Viszonyréteg (session layer) Ez a réteg építi ki, kezeli és fejezi be az alkalmazások közötti dialógusokat (session, dialógus kontroll).
Közvetlenül a szállítási rétegre épül. Ez a réteg azt teszi lehetõvé, hogy különbözõ gépek felhasználói viszonyt létesíthessenek egymással. Lényegében közönséges adatátvitelrõl van szó, amihez néhány kényelmes szolgáltatást adtak hozzá. Ilyen például az úgynevezett kölcsönhatás-menedzselés, ami vezérli, hogy a két oldal egyszerre ne próbálkozzon ugyanazzal a mûvelettel. Ez például úgy oldható meg, hogy vezérlõjelet tartanak fent, és csak az az oldal végezheti az adott mûveletet, amelyiknél ez a vezérlõjel van. Egy másik fontos szolgáltatás a szinkronizáció. Képzeljük el például, hogy egy állománytovábbítás valamilyen hálózati hiba miatt megszakad. Jó lenne, ha ilyen esetben nem kellene elölrõl kezdeni az egészet. Ezért a viszonyréteg az adatokhoz úgynevezett szinkronizációs jeleket ragaszt, amelyek segítségével a hiba megszûnése után az adatok továbbítása az utolsó ellenõrzési jeltõl folytatódhat.
(**Adategység: Üzenet, PDU – Protocol Data Unit**)

6. Megjelenítési (ábrázolási) réteg (presentation layer) Feladata a különböző csomópontokon használt különböző adatstruktúrákból eredő információ-értelmezési problémák feloldása. Információábrázolási különbségeket fedi el a réteg. Megoldás: kell egy köztes ábrázolás, amit mindenki használ az átvitelkor.
(**Adategység: Üzenet, PDU – Protocol Data Unit**)

7. Alkalmazási réteg (application layer) Az alkalmazások (fájl átvitel, e-mail, stb.) működéséhez nélkülözhetetlen szolgáltatásokat biztosítja (pl. fájl átvitel esetén a különböző fájlnév konvenciók figyelembe vétele).

A fentiek fényében foglaljuk össze, hogy hogyan mûködik egy ilyen hálózat ! A küldõ számítógépen egy alkalmazói program adatot küld a protokoll-vermen. A megjelenítési réteg az adatokat tömöríti, esetleg titkosítja, majd tovább adja a szállítási rétegnek, amely a megfelelõ méretû csomagokra bontja az üzenetet. Minden csomag információt tartalmaz arra nézve, hogy hová kell küldeni. A csomagok lejjebb kerülnek a hálózati réteghez, amely meghatározza az útvonalat, majd az adatkapcsolati és a fizikai réteg segítségével kiadja azokat a hálózatra. A célállomáson a folyamat fordítottja történik: az adatokat a vevõ oldal fizikai és adatkapcsolati rétegén keresztül a hálózati réteg fogadja. A szállítási réteg a csomagokat megfelelõ sorrendben összeállítja, a megjelenítési réteg pedig dekódolja az alkalmazási réteghez forduló programok számára. Egy ilyen kommunikációt úgy lehet elképzelni, hogy az adatok az egyik oldalon a protokoll-verem aljára (fizikai réteg) kerülnek, ott kijutnak a hálózatra, majd a másik oldalon a fizikai rétegen keresztõl bejutnak a protokoll-verembe, és fel egészen az alkalmazói programokhoz. Mindebbõl a felhasználó nem vesz észre semmit, õ ezt úgy látja, hogy az adó oldal alkalmazási rétege kommunikál a vevõ oldal alkalmazási rétegével.
(**Adategység: Üzenet, PDU – Protocol Data Unit**)

7.2. Ethernet szabványok. (IEEE 802.3)

Az OSI hivatkozási modell két legalsó rétegét kielégítő protokoll gyűjtemény.

(Klasszikus) Ethernet – 10BASE A megnevezés első száma az átviteli sebességet jelöli, az ezt követő Base az alapsávú átvitelre utal. A következő szám, koaxiális kábel esetén a kábel hosszát adja meg 100 méteres egységekre kerekítve.

Megnevezés	Kábel	Max. szegmenshossz	Csomópont/szegmens	Megjegyzés
10BASE5	Vastag koaxiális	500 m	100	Eredeti kábeltípus, idejétmúlt
10BASE2	Vékony koax	185 m	30	Nincs szükség elosztóra (MAE)
10BASE-T	sodort érpár	100 m	1024	Legolcsóbb rendszer
10BASE-F	optikai	2000 m	1024	Épületek között

Fast Ethernet – 100BASE Különböző médiumokra tervezték: Category 5 árnyékolatlan (UTP) kábel, Category 5 árnyékolt (STP) kábel, Optikai szál. Mindegyik más fizikai médiumfüggő alréteggel rendelkezik.

Az FDDI hálózatra kifejlesztett 4B5B (4B/5B) bit kódolást adaptálták a 100 Base X-re. Az adat minden 4 bitjét (nibble) 5 biten kódolnak. Csak olyan 5 bites szimbólumokat használnak, amelyben legfeljebb két '0' bit van egymás mellett. A garantált 2 bitenkénti jel átmenet jó bit szinkronizálást biztosít. Az adat kódolásra nem használt 16 öt bites szimbólum közül 2-2 a keret elejét és végét határolja.

Megnevezés	Kábel	Max. szegmenshossz	Megjegyzés
100BASE-T4	sodort érpár	100 m	3-as kategóriájú UTP
100BASE-TX	sodort érpár	100 m	Duplex 100Mb/s (5-ös kat. UTP)
100BASE-FX	optikai	2000 m	Nagy távolságra, duplex 100Mb/s

Gigabit Ethernet – 1000BASE Adatkapcsolati rétegből tekintve a Gigabit és a 10Gigabit Ethernet ugyanolyan, mint a 10, ill. 100 megabites Ethernet. A GbE-nél a FastEthernet gyorsítását és a FiberChannel ANSI X3T11 szabványát ötvözték. 8B/10B kódolást alkalmaz.

- Használt közegek: SM (Single Mode): egymódusú optikai szál; MM (Multi Mode): többmódusú optikai szál; STP (Shielded Twisted Pair): árnyékolt sodrott érpár; UTP (Unshielded Twisted Pair): árnyékolatlan sodrott érpár.
- Duplexitás: HD (Half-Duplex) és FD (Full-Duplex) üzemmód.
- Áthidalható távolságok: Réz érpárral max. 150 méter, koax kábellel max. 25 méter. Multimódusú szál esetén 850 nm hullámhosszú jellel 220–550 méteres, 1310, ill. 1550 nm esetén 550 méteres távolságra működik, egymódusú szállal 1310 nm esetén 5 km a hatósugár.

Gigabit Ethernetnek a Short Range (SX), a Long Haul (LX), Copper Physical Interface (CX) típusai léteznek. Az LX interfész 10 km-es távolságot képes áthidalni.
A Gigabit Ethernet kompatibilis visszafelé, de a keret mérete minimálisan 512 B.

Megnevezés	Kábel	Max. szegmenshossz	Megjegyzés
1000BASE-SX	Optikai	550 m	Többmódosú optikai szál
1000BASE-LX	Optikai	5000 m	Egymódosú optikai
1000BASE-CX	2 pár STP	25 m	Árnyékolt, sodort érpár
1000BASE-T	4 pár UTP	100 m	CAT5 sodort érpár

10Gigabit Ethernet – 10GBASE

- Használt közegek: Optikai szálat használ 850, 1310, ill. 1550 nm hullámhosszú jelekkel. A 850 nm-es jel még látható, a másik kettő már nem.
- Duplexitás: csak a FD üzemmódot ismeri.
- Áthidalható távolságok: multimódusú optikai szállal 850 nm-es jel esetén 65, 1310 nm-es jel esetén 300 métert képes áthidalni, egymódusú szállal pedig interfésztől függően 10–40 km-t.
- 64B/66B kódolást alkalmaz
- Szabványok: 10GBASE-R, 10GBASE-W, 10GBASE-X
- 10GBASE-W WAN technológia, amelynek akár 40 km-es max. szegmenshossza is lehet.

7.3. A hálózati réteg forgalomirányító mechanizmusai.

Forgalomirányítás (routing) Csomagok (IP datagramok) továbbítási irányának meghatározásával kapcsolatos döntések meghozatala.

Forgalomirányítási táblázat (routing table) A forgalomirányításhoz szükséges információkat tartalmazó táblázat. Tipikus (legfontosabb) mezők: Célhálózat; Netmask; Kimenő interfész; Következő hop; Metrika. Feltöltése történhet statikus vagy dinamikus módon. Statikusnál direkt módon megadunk egy vagy több sort a táblázatba, míg dinamikus esetben routing protokollok végzik automatikusan, figyelve a hálózatot. Pl.: RIP, OSPF

Forgalomirányított protokoll (routed protocol) Olyan hálózati réteghez kötődő általános adatszállító protokoll, melyet a forgalomirányító (router) irányítani képes (pl. IP, IPX).

Metrika Egy adott forgalomirányítás eredményeként előálló útvonal minőségének mérési módja, alapvetően két (egymásba transzformálható) kategória: Távolság alapú (költség alapú) metrika és jóság alapú metrika.

Forgalomirányítók (alapvető) működése:

1. A router az input interfészen érkező csomagot fogadja.
2. A router a csomag célcímét illeszti a routing táblázat soraira. Ha a célcím több sorra illeszkedik, akkor a leghosszabb prefixű sort tekintjük illeszkedőnek.
3. Ha nem létezik illeszkedő sor, akkor a cél elérhetetlen, a csomag nem továbbítható. A csomagot a router eldobja és ICMP hibajelzést küld a feladónak.
4. Ha létezik illeszkedő sor, akkor a csomagot az ebben szereplő kimeneti interfészen továbbítjuk (adatkapcsolati rétegbeli beágyazással) a következő hop-ként megadott szomszédhoz, ill. a célállomáshoz, ha már nincs több hop.

Forgalomirányítás – IP cím illesztés:

1. A routing tábla sorait prefix hossz szerint csökkenő sorrendbe rendezzük. N=1.
2. Ha nem létezik a táblázatban az N. sor, akkor nincs illeszkedő sor és vége.
3. A csomag célcíme és az N. sor hálózati maszkja között bitenkénti AND műveletet hajtunk végre.
4. Ha a bitenkénti AND művelet eredménye megegyezik az N. sor célhálózat értékével, akkor a cím az N. sorra illeszkedik és vége.
5. N=N+1, és folytassuk a 2. pontnál.

Forgalomirányítási konfigurációk osztályozása:

- Minimális routing: Teljesen izolált (router nélküli) hálózati konfiguráció.
- Statikus routing: A forgalomirányítási táblázatot a rendszeradminisztrátor tartja karban.
- Dinamikus routing: A forgalomirányítási táblázat(ok) valamilyen routing protokoll segítségével kerülnek karbantartásra.
  - Belső forgalomirányítási protokollok (IGP - Pl. RIP, OSPF). Legfőbb alapelv a „legjobb útvonal” meghatározása ún. távolságvektor alapú vagy link állapot alapú módszerrel.
  - Külső forgalomirányítási protokollok (EGP - Pl. EGP, BGP). Nem feltétlenül a legjobb útvonal meghatározása a cél (politika alapú forgalomirányítás - BGP)

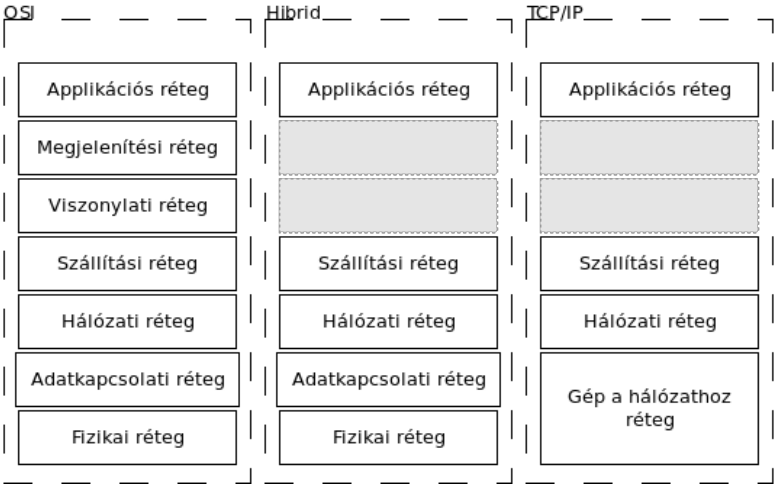
7.4. Az internet hálózati protokollok, legfontosabb szabványok és szolgáltatások.

A hétköznapi életben leginkább elterjedt hálózati technológia a TCP/IP *protokollrendszerre* épülő hálózat. A TCP/IP architektúra – korántsem egységes – modellszemlélete eltér az OSI modell szemlélet módjától: A gyártóspecifikus technológiákkal ellentétben a TCP/IP-t nyílt szabványként alakították ki. Ez azt jelenti, hogy a TCP/IP-t mindenki szabadon használhatja. Ez hozzájárult ahhoz, hogy a TCP/IP gyorsan szabvánnyá fejlődjön. A TCP/IP modell a következő négy rétegből áll:

1. Applikációs réteg – OSI L7
   - Az OSI L5 és L6 rétegekkel nincs megfeleltetés

2. Szállítási réteg – OSI L4
3. Hálózati réteg – OSI L3
4. Hálózat-elérési réteg – OSI L1 és L2 rétegével rokon

Hibrid referenciamodell A. S. Tanenbaum Számítógép-hálózatok c. művében javasolta, hogy a hálózati kommunikáció tanulmányozására egy ún. „hibrid modellt" használjunk: az alsó két rétegben – az OSI modellt követve – a fizikai és adatkapcsolati rétegek jelennek meg, a felsőbb rétegeket pedig – a TCP/IP modellt követve – a hálózati, szállítási és applikációs rétegek képviselik. Ez a modell jobban összevethető az OSI modellel:



1. ábra. A három modell összehasonlítása

A TCP/IP tervezői úgy gondolták, hogy az alkalmazási rétegnek magába kell foglalnia az OSI viszonyrétegének és megjelenítési rétegének részleteit is. Így az általuk létrehozott alkalmazási réteg a megjelenítés, a kódolás és a párbeszédvezérlés kérdéseit kezeli.

A szállítási réteg a szolgáltatás minőségi kérdéseivel foglalkozik, vagyis a megbízhatósággal, az adatfolyamvezérléssel és a hibajavítással. Az egyik ide tartozó protokoll, a Transmission Control Protocol (TCP) igen hatékony és rugalmas módon teszi lehetővé a megbízható, gyors, alacsony hibaarányú hálózati kommunikációt.

A TCP egy összeköttetés alapú (más néven kapcsolatorientált) protokoll. Ez a protokoll párbeszédszerű kommunikációt tart fenn a forrás- és a célállomás között, és szegmensekbe csomagolja az alkalmazási rétegből származó információkat. A kapcsolatorientáltság nem jelenti azt, hogy áramkör létezik a kommunikáló számítógépek között. Azt jelenti, hogy a két állomás között 4. rétegbeli szegmensek haladnak oda-vissza, nyugtázva, hogy bizonyos időtartamra fennáll a logikai összeköttetés.

Az internet réteg rendeltetése, hogy csomagokra bontsa a TCP-szegmenseket, és elküldje őket bármely hálózatról. A csomagok megérkeznek a célhálózatra, attól függetlenül, hogy milyen útvonalon jutottak el oda. Ennek a rétegnek a feladatát az Internetprotokoll (IP) látja el. A legjobb útvonal kiválasztása és a csomagkapcsolás ebben a rétegben történik.

A hálózat-elérési réteg nevének nagyon tág a jelentése, ezért némileg megtévesztő lehet. Másképpen állomás-hálózat közötti rétegnek is nevezzük. Ebbe a rétegbe tartozik minden olyan fizikai és logikai összetevő, amely a fizikai összeköttetés létrehozásához szükséges. Ide tartoznak a hálózati technológiák részletei, beleértve az OSI modell fizikai és adatkapcsolati rétegének minden részletét.

A leggyakrabban használt alkalmazási rétegbeli protokollok közé tartoznak például a következők:

- FTP – File Transfer Protocol (20-21/tcp)
- HTTP – HyperText Transfer Protocol (80/tcp)
- HTTPS – HyperText Transfer Protocol Secure (443/tcp)
- SMTP – Simple Mail Transfer Protocol (25/tcp)
- DHCP – Dynamic Host Configuration Protocol (67-28/udp)
- DNS – Domain Name System (53/udp,tcp)
- SFTP – Secure File Transfer Protocol (22/tcp)
- SSH – Secure SHell (22/tcp)

A szállítási réteg gyakoribb protokolljai:

- TCP – Transport Control Protocol
- UDP – User Datagram Protocol

Az internet réteg a TCP/IP protokollstruktúra fontosabb protokolljai:

- IP – Internet Protocol
- ICMP – Internet Control Message Protocol

8. Fizika 1

Fizikai fogalmak, mennyiségek. Impulzus, impulzusmomentum. Newton törvényei. Munkatétel. Az I. és II. főtétel. A kinetikus gázmodell.

8.1. Fizikai fogalmak, mennyiségek.

Alapfogalmak

Mérőszám Megmutatja, hogy a mértékegységet hányszor lehet a mérendő mennyiségbe belefoglalni.

Mértékegység Egy mérőszám típusát és nagyságrendjét meghatározó jelző.

Fizikai mennyiség Adott fizikai jellemzőt leíró mérőszám és annak mértékegysége. Valamely jelenség, folyamat minőségileg megkülönböztethető, és mennyiségileg meghatározható tulajdonsága.

1. **Skalármennyiség** Olyan mennyiség, melynek nincs iránya, tehát teljes mértékben leír a nagysága (pl. hőmérséklet, tömeg, térfogat).
2. **Vektormennyiség** Olyan mennyiség, melynek nagyságán kívül iránya is van (pl. erő, sebesség, térerősség).

Erő *Olyan hatás, ami egy tömeggel rendelkező testet gyorsulásra késztet.* Az *eredő erő* a testre ható *összes erő vektoriális összege.* $1N$ erő olyan hatás, amely egy $1kg$ tömegű testet $1\frac{m}{s^2}$ mértékű gyorsulásra késztet. Vektormennyiség. Iránya megegyezik a gyorsulás irányával. Néhány alapvető erő: elektromágneses erő, gravitációs erő, nukleáris erő.
Jele: F (force) $\quad \vec{F} = m \cdot \vec{a}$
Mértékegysége: N (Newton) $\quad N = {}^{kg \cdot m}/_{s^2}$

Tömeg *A fizikai testek tulajdonsága, amely a tehetetlenségüket méri.* A súlytól eltérően a tömeg mindig ugyanaz marad, akárhová kerül is a hordozója. Skalármennyiség.
Jele: m (mass)
Mértékegysége: kg (kilogramm)
Szigorúan véve három különböző dolgot neveznek tömegnek:

Tehetetlen tömeg a test tehetetlenségének mértéke: a rá ható erő mozgásállapot változtató hatásával szembeni ellenállás. A kis tehetetlen tömegű test sokkal gyorsabban változtatja mozgásállapotát, mint a nagy tehetetlen tömegű.

Passzív gravitáló tömeg a test és a gravitációs tér kölcsönhatásának mértéke. Azonos gravitációs térben a kisebb passzív gravitáló tömegű testre kisebb erő hat, mint a nagyobbra.

Aktív gravitáló tömeg a test által létrehozott gravitációs tér erősségének a mértéke. Például a Hold gyengébb gravitációs teret hoz létre, mint a Föld, mert a Holdnak kisebb az aktív gravitáló tömege.

Súly Az az erő, amellyel a test az **alátámasztást nyomja** vagy a **felfüggesztést húzza**, tehát *a test környezetére gyakorolt erőinek az eredője.* Azonos tömegű testek különböző erősségű gravitációs terekben különböző súlyúak, mert a testre ható gravitációs erőt a test közvetíti a környezetének.
Jele: $G \quad \vec{G} = m \cdot \vec{g}$
Mértékegysége: N (Newton)

Távolság Két pont közötti távolság az a legrövidebb út, melyen eljutunk egyik pontból a másikba, tehát a helyváltozás mértéke. A legrövidebb út általában egyenes mentén van. Skalármennyiség.
Jele: d (distance) vagy l (length)
Mértékegysége: m (méter)

Pálya Azt a vonalat, amin a test mozog, pályának nevezzük.

Út A mozgó test által befutott pályaszakasz hossza a megtett út.

Elmozdulás A kezdőpontból a végpontba mutató vektort elmozdulásnak nevezzük. Az út hossza nem lehet kisebb az elmozdulás nagyságánál, hiszen két pont között az egyenes szakasznak a legkisebb a hossza.

Idő A folyamatokban bekövetkező *események sorrendiségének kifejezésére való skalármennyiség.* Az idő SI-alapegysége az SI-másodperc(?). Az ebből származó nagyobb időegységek, mint perc, óra és nap, nem SI-egységek, mert nem tízes számrendszerűek, és szükség van időnként szökőmásodpercre.
Jele: t (time)
Mértékegysége: s (szekundum vagy másodperc)

Nyomás Az egységnyi felületre ható erőhatást adja meg.
Jele: p (pressure) $p = {}^{F}/_{A}$
Mértékegysége: Pa (Pascal) $Pa = {}^{N}/_{m^2}$

Munka Amikor egy *testre kifejtett erő hatására a test elmozdul, mechanikai munkavégzés történik*, ami arányos a kifejtett erő nagyságával és a megtett úttal. A munka az erő és az elmozdulás skaláris szorzata. Egy joule munkát végez az egy newton nagyságú erő a vele egyirányú egy méter hosszúságú elmozdulás közben. Ugyancsak egy joule az egy watt teljesítménnyel egy másodpercig végzett munka.
Jele: W (work) $W = F \cdot s \cdot \cos\alpha$
Mértékegysége: J (Joule) $J = N \cdot m = kg \cdot \frac{m^2}{s^2}$

Teljesítmény A munkavégzés vagy energiaátvitel sebessége, más szóval az *egységnyi idő alatt végzett munka.*
Jele: P (power) $P = {}^{W}/_{t}$
Mértékegysége: W (Watt) $W = {}^{J}/_{s}$

Sebesség Egy pontszerű test kitüntetett ponthoz viszonyított mozgásának jellemzésére szolgáló fizikai mennyiség. **Az út idő szerinti deriváltja**, tehát az *időegység alatt bekövetkezett helyváltozás mértéke.* Vektormennyiség.
Jele: v (velocity) $v = {}^{s}/_{t}$
Mértékegysége: ${}^{m}/_{s}$

Gyorsulás **A sebességvektor idő szerinti deriváltja**, tehát az *időegység alatt bekövetkezett sebességváltozás mértéke.* Vektormennyiség.
Jele: a (acceleration)

$$\overline{a} = \frac{\Delta v}{\Delta t} = \frac{v - u}{t} \quad \text{ahol v a végsebesség, u a kezdeti sebesség}$$

$$a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt}$$

Mértékegysége: ${}^{m}/_{s^2}$

8.2. Impulzus, impulzusmomentum.

Lendület (impulzus) Egy test mozgását leíró vektormennyiség. Nagysága arányos a tömeggel és a sebességgel. Lendületmegmaradás törvénye: zárt rendszer összes lendülete állandó. Nyugalomban lévő testnek nincs lendülete, a lendületet csakis külső erő változtathatja meg.
Jele: I (impulzus)

$$\vec{I} = m \cdot \vec{v}$$

Mértékegysége: $kg \cdot {}^{m}/_{s} = N \cdot s$

Perdület (impulzusmomentum) Forgómozgásban lévő test lendülete által létrehozott nyomatékot jellemző vektormennyiség. Az erőkar és a lendület szorzata. Zárt rendszerben a lendületmegmaradás következtében a perdület állandó. Rögzített tengely körüli forgásnál a perdületet a tehetetlenségi nyomaték és a szögsebesség szorzatából számíthatjuk ki. Jele: L

$$L = \Theta \cdot \omega$$

Mértékegysége: $kg \cdot {}^{m^2}/_{s} = N \cdot m \cdot s$

8.3. Newton törvényei.

I. Tehetetlenség törvénye Minden inerciarendszerben vizsgált test nyugalomban marad vagy egyenes vonalú egyenletes mozgást végez mindaddig, míg ezt az állapotot egy másik test vagy erő hatása meg nem változtatja egy kölcsönhatás során.

II. Dinamika alaptörvénye Egy pontszerű test gyorsulása azonos irányú a rá ható erővel, nagysága egyenesen arányos az erő nagyságával, és fordítottan arányos a test tömegével. $\vec{F} = m \cdot \vec{a}$

III. Hatás-ellenhatás törvénye Két test kölcsönhatása során mindkét testre azonos nagyságú, azonos hatásvonalú és egymással ellentétes irányú erő hat.

IV. Szuperpozíció elve Ha egy testre egy időpillanatban több erő hat, akkor ezek együttes hatása megegyezik a vektori eredőjük (vektoriális összegük) hatásának vonalával.

8.4. Munkatétel.

Tétel. *Egy test mozgási energiájának változásának mértéke megegyezik a testre ható összes erő munka előjeles összegével.*

$$\Delta E_m = \sum_{i=1}^{n} W_i$$

8.5. A termodinamika I. és II. főtétel.

Tétel (I. főtétel)**.** *Egy zárt rendszer belső energiájának változása egyenlő a rendszerrel közölt hő és a rendszeren végzett munka összegével.*

$$\Delta U = Q + W$$

A testek belső energiájának megváltozása egyenlő a testtel közölt hőmennyiség és a testen végzett munka előjeles összegével. Ez az energiamegmaradás törvénye, mert azt mondja ki, hogy külső beavatkozás nélkül nincs energiaváltozás.

Tétel (II. főtétel)**.** *Termikus kölcsönhatással járó természetes folyamatoknál csak a nagyobb hőmérsékletű test képes a hőátadásra. Tehát egy elszigetelt rendszer állapota időben termikus egyensúly felé halad.*

8.6. A kinetikus gázmodell.

A gázok tömeggel rendelkező részecskékből állnak, melyek energiaveszteség nélkül ütköznek egymással és a környezetükben lévő testekkel. Mozgási energiájuk csak a rendszer hőmérsékletétől függ. Állandóan mozgásban vannak, és az ütközések között egyenes vonalú egyenletes mozgást végeznek. Össztérfogatuk mindig jóval kisebb, mint a gázt tartalmazó tároló térfogata. Alapegyenlet:

$$p \cdot V = \frac{2}{3} N \cdot \frac{1}{2} m v^2$$

ahol p az ütközésekkel keltett nyomás, V a térfogat, N a molekulaszám, az $1/2mv^2$ pedig egy molekula átlagos mozgási energiája.

A kinetikus gázmodell segítségével értelmezhetők a gázok olyan makroszkopikus tulajdonságai, mint a nyomás, a hőmérséklet, vagy a térfogat. Newton úgy gondolta, hogy a gáz nyomását a molekulái között fellépő állandó taszítás okozza, a kinetikus gázelmélet szerint viszont a nyomás a különböző sebességgel mozgó gázmolekulák ütközéseiből származik. A kinetikus gázmodell a természetben nem létező ideális gázt feltételez. A kinetikus gázmodell szerint:

1. a gáz részecskékből áll, amelyeknek tömege és súlya van, viszont össztérfogatuk elhanyagolható a gázt tartalmazó edény térfogatához képest
2. a részecskék állandó mozgásban vannak
3. a részecskék tökéletes gömb alakúak és rugalmas természetűek
4. a részecskék mozgási energiája csak a rendszer hőmérsékletétől függ
5. a részecskék egymással és az edény falával energiaveszteség nélkül ütköznek
6. a részecskék közötti erőhatások elhanyagolhatóak, ezért két ütközés között egyenes vonalú egyenletes mozgást végeznek

A kinetikus gázelmélet alapegyenlete:

$$pV = \frac{2}{3} N \frac{m_0 v^2}{2}$$

p – nyomás V – térfogat N – molekulák száma m_0 – egy molekula tömege $\frac{m_0 v^2}{2}$ – egy molekula átlagos mozgási energiája

9. Fizika 2

Elektromos alapfogalmak és alapjelenségek. Ohm-törvény. A mágneses tér tulajdonságai. Elektromágneses hullámok. A Bohr-féle atommodell. A radioaktív sugárzás alapvető tulajdonságai.

9.1. Elektromos alapfogalmak és alapjelenségek.

9.1.1. Testek elektromos állapota

Minden atom magja pozitív töltésű protonokból és töltés nélküli neutronokból áll. A mag körüli elektronhéjon keringenek a protonokkal megegyező számú, negatív töltésű elektronok.
Alapesetben az atom kifelé elektromos hatást nem mutat. Ha elektronokat ad le vagy vesz fel, akkor viszont pozitív, illetve negatív töltésű ionná válik. Ennek megfelelően megkülönböztetünk pozitív, negatív és semleges elektromos állapotot.
Elektromos szempontból az anyagokat vezetőkre és szigetelőkre oszthatjuk. Az elektromos vezetők mozgásképes töltéshordozókat (elektronokat, ionokat) tartalmaznak. Megkülönböztetünk elsőrendű vezetőket (fémek és a szén), másodrendű vezetőket (elektrolitok és ritkított gázok), félvezetőket, (pl. germánium, szelén, szilícium) amelyek kristályos szerkezetű anyagok és részben vezetők, részben szigetelők. Speciálisak a szupravezetők, amelyek hőmérsékletét csökkentve, a kritikus hőmérséklet elérésekor hirtelen teljesen elvesztik ellenállásukat. Az elektromos szigetelők (pl. üveg, olaj, műanyag) nem tartalmaznak mozgásképes töltéshordozókat.

9.1.2. Elektromos töltés

Az anyag alapvető tulajdonsága, bizonyos elemi részecskék jellemzője. Megkülönböztetünk pozitív és negatív töltést. Azonos nemű töltések taszítják, különböző neműek vonzzák egymást.
Jele: Q
Mértékegysége: Coulomb (C)
1 Coulomb = 1As (amperszekundum)
Az elektromosan töltött anyag elektromágneses teret hoz létre. Az elektromos töltés kvantált, azaz minden test töltése egy legkisebb töltés többszöröse. Ez a legkisebb töltés az elemi töltés, ami a proton töltése: 1,6 · 10 -19 Coulomb. Az elektron töltése ennek a -1 -szerese. (A kvarkok felfedezése óta úgy gondolják, hogy azok töltése még kisebb lehet, az elemi töltés 1/2, vagy 1/3 része.) Az elektromos töltésre is igaz a megmaradási elv, azaz zárt rendszer töltése állandó. Az elektromos töltések egymásra gyakorolt erőhatásán keresztül a töltés mérhető. A pontszerű töltések között ható (F) erő egyenesen arányos a töltésekkel (Q 1 , Q 2) és fordítottan arányos a töltések közötti távolság (r) négyzetével. Ez Coulomb törvénye.

$$F = k \cdot \frac{Q_1 \cdot Q_2}{r^2}$$

ahol k az arányossági tényező, melynek értéke: $9 \cdot 10^9 \frac{Nm^2}{C^2}$

9.1.3. Elektromos Térerősség

Két pontszerű töltés egymásra gyakorolt hatását úgy is értelmezhetjük, hogy az egyik töltés maga körül elektromos teret hoz létre és ebben a térben a másik töltésre erő hat. Ez az F erő arányos a térben lévő Q töltés nagyságával:

$$F = Q \cdot E$$

Az E arányossági tényező jellemzi az elektromos teret, neve elektromos térerősség. Vektormennyiség, ha a Q töltés pozitív, akkor a töltésre ható erő iránya megegyezik E irányával. Mértékegysége: $^N/_C$ és $^V/_m$

9.1.4. Fluxus

Az elektromos mezőt erővonalakkal (Faraday vonalak) szemléltetjük. Az erővonalak bármely pontjába húzott érintő megadja az adott pontban mérhető térerősség irányát, a vonalak sűrűsége pedig a térerősség nagyságával arányos. A fluxus az erővonalak száma. Jele a Ψ (pszi), mértékegysége: $\frac{N}{C}m^2$.

$$\Psi = E \cdot A = 4k \cdot \pi \cdot Q$$

E : Q elektromos töltéstől r távolságban a térerősség ($k\frac{Q}{r^2}$)
A : egy r sugarú gömb felszíne ($4\pi r^2$)

Tétel (Gauss-tétel)**.** *Zárt felületen az elektromos tér fluxusa egyenlő a felületen belül elhelyezkedő töltés $4k\pi$–szeresével. Ha Q pozitív, akkor az erővonalak a felületről kifelé, ha negatív, akkor befelé irányulnak. Tehát minden Q pozitív töltésből $4k\pi Q$ erővonal indul és minden Q negatív töltésben $4k\pi Q$ erővonal végződik.*

9.1.5. Elektromos potenciál és feszültség

Az elektromos mezőbe helyezett töltés az elektromos erő hatására elmozdul, az erőtér munkát végez, a munka nagysága:

$$W = Q \cdot U$$

Ebből a potenciál:

$$U = k \cdot \frac{Q}{r}$$

Az elektromos teret egy kiválasztott pontjához képest fennálló munkavégző képessége szempontjából a potenciál jellemzi. A villamos töltések a villamos tér egyes pontjaiban potenciális energiával rendelkeznek. Az elektromos tér valamely pontjának potenciálja az a munkamennyiség, amellyel a tér az egységnyi pozitív töltést abból a pontból a nullapotenciálúnak választott pontba képes mozgatni. Az elektromos tér két pontja közötti feszültség az a munkamennyiség, amellyel a tér az egységnyi pozitív töltést a nagyobb potenciálú pontból a kisebb potenciálú pontba képes mozgatni. A feszültség tehát a két pont potenciáljának különbsége. Így az A és B pont közötti feszültség az A-ból a B-be történő mozgatáshoz szükséges munka és a mozgatott töltés hányadosa:

$$U_{AB} = U_A - U_B = \frac{W_{AB}}{Q}$$

Mértékegysége a Volt. Az elektromos mező két pontja között 1 Volt a feszültség, ha 1 Coulomb töltés átvitelekor az erők 1 Joule munkát végeznek.

9.1.6. Elektromos áram és áramerősség

Az elektromos áram a szabadon mozgó töltéshordozók rendezett áramlása. Az elektromos áram létrejöttének feltétele a potenciálkülönbség. Ha az áramlás folyamatosan egyirányú, akkor egyenáramról, ha az áramlás iránya szabályos időközönként megfordul, akkor váltakozóáramról beszélünk.
Az áramerősség a vezető keresztmetszetén 1 másodperc alatt átáramló töltések mennyisége:

$$I = \frac{Q}{t}$$

Mértékegysége az Amper. 1 Ampernyi áramerősség a vezetőréteg keresztmetszetén 1 sec alatt átáramló 1 Coulomb töltést jelent.

9.1.7. Kapacitás

Egy rendszer azon képessége, hogy mennyi elektromos töltést képes tárolni. Két vezetőből álló kondenzátor esetén a kapacitás egyenlő az egyik vezetőn felhalmozott töltés és a vezetők közötti feszültség hányadosával:

$$C = \frac{Q}{U}$$

Mértékegysége a Farad.

9.1.8. Ohm-törvény

Egy áramkörben az áramerősség állandó nagyságú ellenállás esetén egyenesen arányos a feszültséggel:

$$U = I \cdot R \quad R = \frac{U}{I}$$

R az ellenállás, ami a vezető anyagi minőségétől, méreteitől és hőmérsékletétől függő állandó.
Mértékegysége az Ohm (Ω).
1 Ohm ellenállású a vezető, ha abban 1 Volt feszültség 1 Amper erősségű áramot hoz létre.
Az ellenállás a vezető hosszával egyenesen, a keresztmetszetével fordítottan arányos:

$$R = \rho \cdot \frac{1}{A}$$

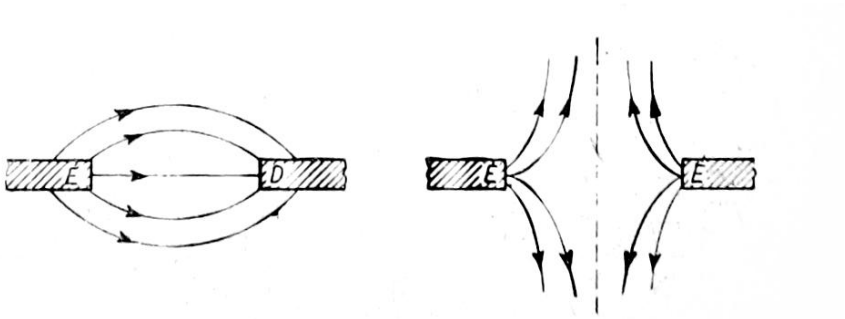
ρ (ró) a vezető fajlagos ellenállása, ami az anyagi minőségétől és hőmérsékletétől függő állandó.

9.1.9. Elektromos áram munkája és teljesítménye

A feszültség: $U = \frac{W}{Q} \rightarrow W = U \cdot Q$
Az áramerősség: $I = \frac{Q}{t} \rightarrow Q = I \cdot t$
Elektromos munkavégzés: $W = U \cdot I \cdot t$
Másképpen: $U = IR$, $I = \frac{U}{R} \rightarrow W = I^2 R \cdot t = \frac{U^2}{R} \cdot t$
Elektromos teljesítmény: $P = \frac{W}{t} = \frac{UI \cdot t}{t} \rightarrow P = U \cdot I = I^2 \cdot R = \frac{U^2}{R}$
Az elektromos teljesítmény általános mértékegysége a kilowatt (kW), a munkáé pedig a kilowattóra (kWh).

9.2. A mágneses tér tulajdonságai.

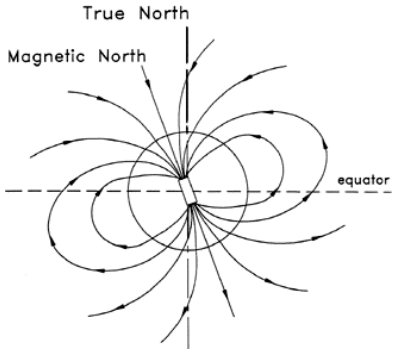
Két mágnesrúd egymásra vonzó vagy taszító hatást fejt ki. Az azonos mágneses pólusok taszítják, az ellentétesek vonzzák egymást. Ebből következik, hogy minden mágnes maga körül – az elektromos erőtérhez hasonlóan – mágneses erőteret létesít. Az erőteret a térerősséggel, a mágneses indukcióval jellemezhetjük és erővonalakkal szemléltethetjük (3 ábra).



2. ábra

Az erővonalak olyan görbék, amelyek a mágnesben záródnak és irányuk a térben az északi pólustól a déli pólus felé mutat. Ha az erővonalak párhuzamosak és mindenütt egyenlő sűrűségűek, akkor a mágneses tér homogén. A mágneses tér és az elektromos tér legfőbb különbsége, hogy míg az elektromos teret akár a pozitív, akár a negatív töltés egymagában is létrehozza, addig a mágneses teret az északi és déli pólus mindig együttesen létesíti.

Abból, hogy a vízszintes síkban szabadon forgó mágnes mindig észak-déli irányba áll be az következik, hogy a Föld is egy mágnes. A Föld északi sarkán déli, déli sarkán északi mágnesség van. Ezért az erővonalak iránya délről észak felé mutat (lásd:3 ábra).



3. ábra

Ha egy vezetőben áram folyik, akkor az mágneses erőteret hoz létre maga körül. A mágneses erővonalak a vezetőre merőleges síkban kialakuló koncentrikus körök, amelyek sűrűsége a vezetőtől távolodva csökken. Az erővonalak irányát jobb kezünk behajlított ujjainak iránya adja, ha hüvelykujjunk az áram irányába mutat. Ha tehát az áram iránya megváltozik, az erővonalak iránya is ellentétessé válik (Jobbkéz-szabály).

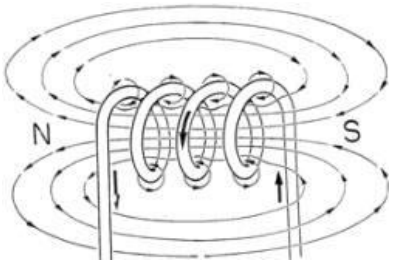
Tétel (Biot–Savart-törvény). *A mágneses térerősség jele H. Ha a vezetőben I az áramerősség és α az irányszög, akkor a vezető ΔL hosszúságú szakasza által, tőle r távolságban létrehozott térerősség nagysága:*

$$H = \frac{\mu_0}{4\pi} \cdot \frac{I \cdot \Delta L}{r^2} \cdot \sin\alpha$$

μ_0*: permeabilitás*

A térerősség mértékegysége az Amper/méter.

Egy többmenetes hengeres tekercs mágneses erővonalai az egyes menetek mágneses terének eredőjeként alakulnak ki. Mivel a tekercs belsejében a mágneses tér majdnem homogén, a tekercs egy olyan rúdmágnesnek tekinthető, amelynek hossztengelye egybeesik a tekercs tengelyével (lásd:4 ábra). Ha egy N menetszámú



4. ábra

tekercsben I az áramerősség és a tekercs hossza L, akkor a tekercsben a mágneses térerősség:

$$H = \frac{I \cdot N}{L}$$

Ha egy két végén hintaszerűen felfüggesztett vezetőt egy mágnes erőterébe helyezünk és a vezetőn az áramot bekapcsoljuk, akkor a vezető az áram irányára merőleges irányba mozdul ki. Az elmozdulást létesítő F erő nagysága:

$$F = B \cdot I \cdot L$$

I az áramerősség, L a vezetőnek a mágneses térbe eső hossza, B pedig a mágneses térben lévő anyagtól függő állandó, a mágneses indukció melynek mértékegysége a Tesla.

A mágneses indukciót indukcióvonalakkal jellemezhetjük. Egy adott felületen áthaladó indukcióvonalak száma a mágneses fluxus, ami az indukció és a felület szorzata: $f = B \cdot A$
A mágneses fluxus mértékegysége a Weber.

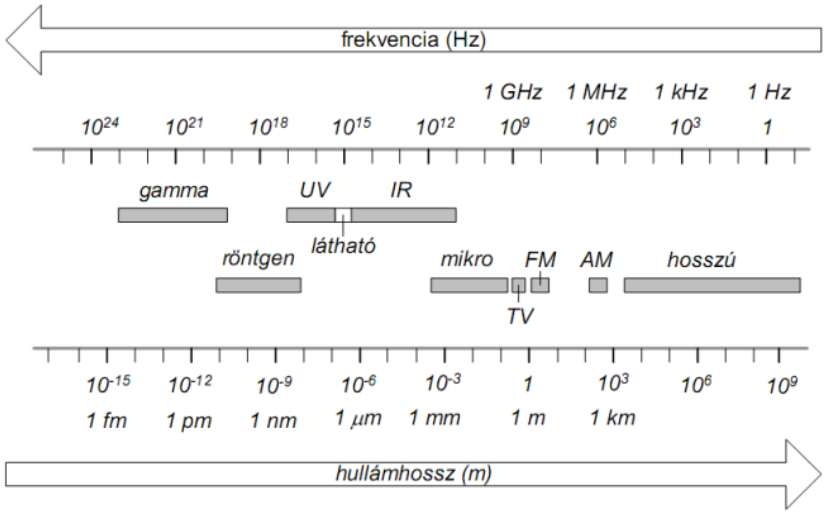
9.3. Elektromágneses hullámok.

Az elektromos töltéssel rendelkező testeknek a töltésük miatt fellépő kölcsönhatását az elektromos és mágneses tér segítségével írhatjuk le. A kölcsönhatás úgy működik, hogy egyrészt minden töltés maga körül elektromágneses teret hoz létre, másrészt az elektromágneses tér a töltésekre erőt fejt ki. Így azt mondhatjuk, hogy két töltött test kölcsönhatása az elektromágneses tér közvetítésével valósul meg. Az elektromágneses tér létrehozásához munkát kell végezni, amely munka révén a létrehozott elektromágneses térben energia halmozódik fel. Tudjuk, hogy az elektromágneses tér időbeli változása a térben meghatározott sebességgel (fénysebességgel, amely vákuumban 300 000 km/s) tovaterjed: elektromágneses hullám jön létre, ami energiát visz magával, az elektromágneses tér energiájának sajátos transzportja jön létre. Az elektromágneses hullám energiaszállító képességére utal az elektromágneses sugárzás elnevezés. Egy hozzánk képest nyugvó elektromos töltés elektromos teret, egyenletesen mozgó töltés elektromos és mágneses teret hoz létre maga körül. Kimutatható, hogy a fenti két esetben a tér és a benne felhalmozott energia a töltéstől nem szakítható el, mintegy hozzá van láncolva. Ha azonban a töltés gyorsul, akkor a körülötte kialakuló, időben változó elektromágneses tér elektromágneses hullámot kelt, amely a töltésről leszakadva a térben tovaterjed, és energiát visz magával: a gyorsuló töltés elektromágneses sugárzást bocsát ki magából. Elektromos töltéssel rendelkező testek azonban nemcsak sugározni képesek, hanem a rájuk eső elektromágneses sugárzást el is nyelhetik. Ha ugyanis az anyag egy töltött részecskéjét elektromágneses sugárzás éri, akkor a sugárzás elektromágneses tere a tér által a töltésre ható erő révén a részecskét felgyorsítja, miáltal a test a ráeső sugárzás egy részét elnyeli. A fenti két folyamat teszi lehetővé, hogy két test kölcsönhatásba léphet egymással úgy is, hogy az egyik a másiknak elektromágneses sugárzás formájában energiát ad át. Ennek a jelenségnek számos konkrét példáját ismerjük. Az elektromágneses sugárzás útján történő energiaátadás közismert példája az elektromágneses hullámokkal megvalósított távközlés (rádió, TV): egy rádióadóban pl. a továbbítandó elektromos jellel (váltakozó áram) rezgőmozgásba (gyorsuló mozgás) hozzák az adóantenna elektronjait, amelyek ennek megfelelő elektromágneses sugárzást bocsátanak ki. Ennek a

sugárzásnak egy része eléri a vevőkészülék antennáját, és a benne lévő elektronokat a sugárzás elektromos tere rezgésbe hozza. Az elektronoknak ez a rezgőmozgása (váltakozó áram) azután a vevőkészülékben létrehozza a leadott jelnek megfelelő elektromos jelet. A sugárzásos energiaátadás másik, közismert példája a hőmérsékleti sugárzás kibocsátása és elnyelése. Tapasztalati tény, hogy az anyagok a hőmérsékletüktől függően különböző hullámhosszú elektromágneses sugárzást bocsátanak ki magukból, s a rájuk eső sugárzás egy részét elnyelik.

A foton az elektromágneses hullámok elemi részecskéje. A fotonok nyugalmi tömege nulla, sebességük a fénysebesség. Az elektromágneses hullám keletkezése atomi szinten úgy történik, hogy a gerjesztés hatására az atommag körül keringő elektron egy nagyobb energiájú pályára ugrik, majd onnan eredeti pályájára visszazuhanva egy fotont bocsát ki. A keletkező foton hullámhossza az energiaszintek különbségétől függ.

A különböző körülmények között létrejött elektromágneses sugárzások lényegében a kibocsátott hullám hullámhosszában és ezzel együtt frekvenciájában térnek el egymástól, és ez eredményezi azt, hogy az anyaggal való kölcsönhatásaik, az anyagra gyakorolt hatásaik is eltérőek.Az elektromágneses sugárzás hullámhossz szerinti felosztása, az ún. elektromágneses spektrum (lásd:5 ábra)



5. ábra

A fény szűkebb értelemben az elektromágneses spektrumnak az a része, amelyet az emberi szem érzékelni képes (kb. 350-től 750 nanométerig). Tágabb értelemben ide sorolják a tartományhoz közvetlenül csatlakozó hosszabb hullámhosszú infravörös- és a rövidebb hullámhosszú ultraibolya sugárzást is. Az emberi szem érdekes sajátsága, hogy a különböző hullámhosszú fényt különböző színűnek észleli. A klasszikus felfogás szerint a fényhullámként terjed, tehát érvényesek rá mindazok az általános törvények, amelyek a hullámokra érvényesek. A fénynek azonban vannak, elektromágneses jellegével és a hullámhosszával összefüggő, speciális tulajdonságai is.

A fény egy anyagban terjedve, és egy határfelülethez érve részben behatol az új anyagba, részben pedig visszaverődik a határfelületről. Vannak anyagok, amelyekbe a fény gyakorlatilag nem tud behatolni, mert a határfelületeikről a fénysugárzás nagyobb része visszaverődik, vagy igen rövid távolságon belül elnyelődik. Az anyagok egy része a fényt többé-kevésbé átereszti, ezeket az anyagokat legtöbbször átlátszó anyagoknak nevezzük. A fény és anyag kölcsönhatása általában függ a fény hullámhosszától, így egy anyag bizonyos hullámhosszakra lehet áteresztő, más hullámhosszakat viszont elnyel.

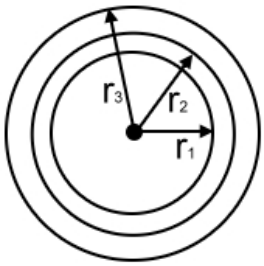
Az elektromágneses hullámok létezését elméletben Maxwell (1864), kísérletileg Hertz mutatta ki (1888).

9.4. A Bohr-féle atommodell.

Az 1900-as évek elején Ernest Rutherford megalkotta róla elnevezett atommodelljét, amely szerint az atom középpontjában van a pozitív töltésű atommag, ami körül keringenek az elektronok. Az atom kifelé semleges, tehát az elektronok száma egyenlő az atommag pozitív töltésének a számával. A modell legnagyobb hibája az volt, hogy mivel a keringő elektronok váltakozó elektromágneses teret hoznak létre maguk körül, ezzel együtt energiát kell kisugározniuk, viszont az energia kisugárzás csökkenti az elektron mozgási energiáját, így az elektron egy idő után az atommagba kellene, hogy essen.

1913-ban Niels Bohr dán fizikus a Rutherford-féle atommodell módosításával megalkotta saját atommodelljét, amely szerint:

1. Az atom normális (gerjesztés nélküli) állapotában az elektron csak meghatározott pályán keringhet és ebben az esetben nem sugároz energiát. Emellett azt is feltételezte, hogy az elektronpályák sugarai úgy aránylanak egymáshoz, mint a természetes egész számok négyzetei:



6. ábra. $r_1 : r_2 : r_3 = 1 : 4 : 9$

2. A megengedett pályákon keringő elektronoknak meghatározott energiájuk van, amelyet mozgási energiájuk és az elektromos erőtér potenciális energiája határoz meg. Egy elektron összenergiája a nagyobb sugarú pályán nagyobb.
3. Ha az atommal energiát közlünk, akkor az elektron átugrik egy másik, nagyobb sugarú pályára. Ez az ún. gerjesztés. A gerjesztéskor felvett energia megegyezik a két pályához tartozó energiaértékek különbségével. A gerjesztett atom azonban instabil, így az elektron a nagyobb energiájú pályáról a maghoz közelebbi kisebb energiájú belső pályára ugrik és az energiakülönbséget elektromágneses hullám formájában kisugározza.

9.5. A radioaktív sugárzás alapvető tulajdonságai.

Becquerel francia fizikus 1896-ban kutatásai során megállapította, hogy az uránszurokérc olyan sugarakat bocsát ki magából, amelyeknek nagy az áthatolóképességük, a levegőt ionizálják. Két évvel később a Curie házaspárnak sikerült az uránércből az uránnál kb. milliószor erősebben sugárzó anyagot a rádiumot kiválasztani. Később még ennél is erősebben sugárzó anyagot találtak, ez a polónium. A további kutatások során megállapították, hogy ilyen radioaktív sugarakat még sok más elem is kibocsát és ezekre az anyagokra jellemző, hogy külső behatás nélkül, állandóan sugároznak.

A radioaktív sugárzás a radioaktív bomlás során keletkezik. A bomlás izotópokban megy végbe. Majdnem valamennyi a természetben előforduló elem izotópok keverékéből áll. Egy elem izotópjai, az elemmel azonos rendszámú, de eltérő tömegszámú és atomsúlyú elemek. Egy adott elem mindegyik izotópja ugyanannyi protont és elektront, de eltérő számú neutront tartalmaz. Egy elem izotópjai kémiailag teljesen azonosak, de fizikai tulajdonságaik eltérnek. Azokat, amelyeknél nem figyeltek meg radioaktív bomlást, stabil izotópoknak nevezik, míg, amelyeknél megfigyeltek azok instabil izotópok. Egy elem csak meghatározott proton/neutron arány esetén stabil. Nagyon sok elemnek vannak természetes radioaktív izotópjai. Vannak olyan elemek, amelyeknek csak radioaktív izotópja létezik (pl. polónium, radon, rádium), viszont mesterségesen minden elemnek előállítható radioaktív izotópja.

A radioaktív bomlás során a radioaktív izotópok instabil atomjai hosszabb-rövidebb idő elteltével alacsonyabb energiaszintű állapotba mennek át, és eközben emberi érzékszervekkel nem észlelhető, de műszerekkel jól kimutatható radioaktív sugárzást bocsátanak ki.

A bomlás mértéke lemérhető. Az 1 másodperc alatt bekövetkező bomlások száma az aktivitás. Mértékegysége a Becquerel. 1 Bq = 1 bomlás/másodperc.

A bomlást a felezési idővel is jellemezhetjük, ez az az idő, amely alatt egy bizonyos mennyiségű anyag atomjainak a száma radioaktív bomlással a felére csökken. Másodpercektől éves nagyságrendekig változhat, akár sok millió vagy milliárd év is lehet.

A bomlás során keletkező sugárzásnak 3 fajtáját különböztetjük meg:

Alfa-sugárzás Pozitív töltésű héliumionokból áll. Töltésük az elemi töltés kétszerese, tömegük kb. a hidrogénatom tömegének négyszerese, sebességük függ a kibocsátó anyagtól. Az alfa-sugaraknak nagy az áthatolóképességük és erősen ionizálnak, viszont a levegőmolekulákkal való ütközés során hamar lefékeződnek, így hatótávolságuk kevesebb, mint egy centiméter.

Béta-sugárzás elektronok vagy pozitronok (béta-részecskék) alkotják. Sebességük széles skálán változik, de jóval nagyobb, mint az alfa-sugárzás esetében és bár tömegük jóval kisebb, mint az alfa-részecskék tömege, a nagy sebesség miatt hatótávolságuk levegőben néhányszor tíz centiméter.

Gamma-sugárzás agy energiájú elektromágneses sugarak, frekvenciájuk elérheti a 10^{21} Hz értéket. Hatótávolságuk végtelen, a nagy tömegszámú és sűrűségű elemek (pl. ólom) viszont hatékonyan gyengítik.

A bomlásokat az általuk létrehozott sugárzás alapján különböztetjük meg. Eszerint van alfa-, béta- és gamma-bomlás.

10. Elektronika 1, 2

Passzív áramköri elemek tulajdonságai, RC és RLC hálózatok. Diszkrét félvezető eszközök, aktív áramköri elemek, alapkapcsolások. Integrált műveleti erősítők. Tápegységek. Mérőműszerek.

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

Ellenállás Jele: R
Kiszámítása: $R = {}^{U}/_{I}$
Az ellenállás kapcsolata a teljesítménnyel: $P = {}^{U^2}/_{R} \quad P = U \cdot I$

- Állandó értékű ellenállások
  - Felépítés: szigetelő hordozó, vezető réteg, fém kivezetések
  - Főbb típusok: huzalellenállás, rétegellenállás, tömbellenállás
  - Beszerelés: furatba, felületre (SMD)
  - Értékét Ohm-ban [Ω] adják meg $\rightarrow R = \rho \cdot {}^{l}/_{A}$ alapján
  - Névleges érték, tűrés $\rightarrow$ nem tudják pontosan gyártani őket, ezért van egy tűréshatár %-ban
  - Terhelhetőség: Watt-ban, maximális teljesítménye; az ellenállás melegszik $\rightarrow$ hődisszipáció
  - Ellenálláskódok $\rightarrow$ ellenálláson színkódok és számkódok
- Változtatható ellenállások (potenciométerek)
  - Típusok: huzalpotenciométer, rétegpotenciométer
  - Szabályozási jellemző: lineáris, nem lineáris (logaritmikus, fordított logaritmikus, S alakú)
  - Terhelhetőség: a teljes névleges ellenállásra vonatkozik, az ebből számított áramot a csúszka egyik állásában sem haladhatja meg a potenciométer árama

$$I_{\max} = \sqrt{\frac{P}{R_{\text{névleges}}}}$$

- Speciális ellenállások (PTK, NTK, VDR)

Kondenzátor Jele: C
Kiszámítása: $Q = C \cdot U$
Mértékegysége: ${}^{As}/_{V}$ = Farad

- Állandó kapacitású kondenzátorok
  - Felépítés: fém fegyverzetek, fém kivezetések, dielektrikum
  - Főbb típusok: sík, hengeres, tekercselt, többrétegű
- Változtatható kondenzátorok
  - Felépítés: mozgatható fegyverzetek, légrés (a fegyverzetek alakja határozza meg a szabályozási jelleget)

Tekercs Jele: L
Kiszámítása: $B = {}^{\Phi}/_{A}$

Indukció a tekercsben feszültség jön létre, ha a tekercsen átmenő fluxus megváltozik.
Önindukció feszültség indukálódik a tekercsben akkor is, ha a fluxus változását áramának megváltoztatásával saját maga idézte elő.
A feszültség azért jön létre, mert megváltozik az áram folyásának iránya, mert ilyenkor a fluxus is megváltozik.

Transzformátor Magyar találmány: Bláthy-Zipernowsky-Déry. Zárt vasmag, két oldal: primer és szekunder tekercs.

$$\frac{U_1}{U_2} = \frac{N_1}{N_2}$$

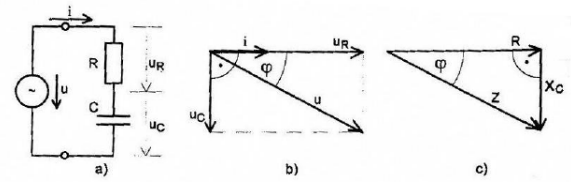
ahol N a tekercs menetszáma.
Felhasználás: igény szerinti feszültség előállítás a 230V-os hálózati feszültségből és a villamos energia gazdaságos szállítása.

10.1.1. RC és RLC hálózatok

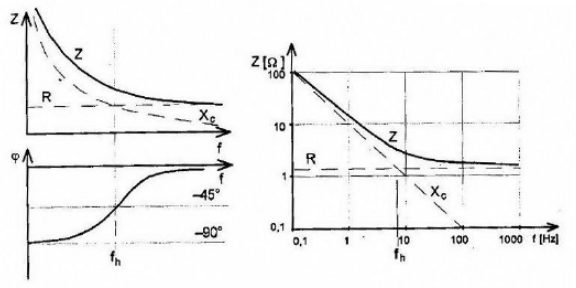
Soros RC hálózat esetén az *áramerősség* a közös mennyiség. A feszültségvektorok diagramjából impedancia háromszöget kapunk, melyből Z kiszámíthatjuk a Z impedanciát:

$$Z^2 = R^2 + X_c^2 \rightarrow Z = \sqrt{R^2 + X_c^2}$$

X_c-vel a kondenzátor reaktanciája. A kapcsolás kis frekvencián X_c miatt szakadásként, nagy frekvencián vezetőként, ohmos ellenállásként viselkedik. Határfrekvenciának (f_h) nevezzük az áramkörben folyó váltakozó áram azon frekvenciáját, melynél $R = X_c$.



(a) Soros R-C kapcsolás és vektor diagramja

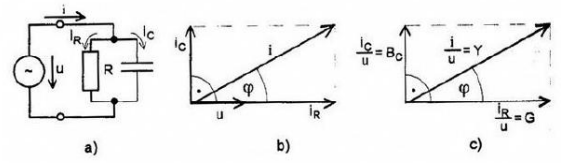


(b) Soros R-C áramkör impedanciájának és fázisszögének változása

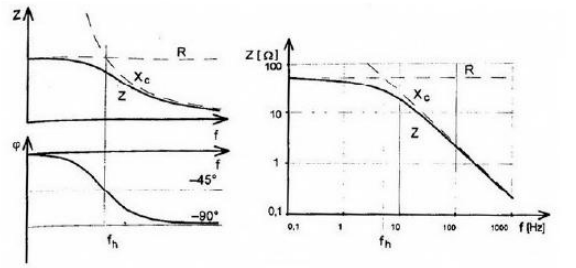
Párhuzamos RC hálózatnál a *feszültség* a közös mennyiség. Az áramok vektorainak diagramjából admittancia (Y) háromszöget kapunk:

$$Y^2 = G^2 + B_c^2 \rightarrow Z = \sqrt{G^2 + B_c^2}$$

ahol $B_c = {}^{I_c}/_{U}$ és $G = {}^{I_R}/_{U}$. A kapcsolás kis frekvencián X_c miatt ohmos ellenállásként, nagy frekvencián rövidzárként (0Ω os ellenállásként) viselkedik. A határfrekvenciát ugyan úgy határozhatjuk meg, mint a soros RC hálózatok esetében.



(a) Párhuzamos R-C kapcsolás és vektor diagramja

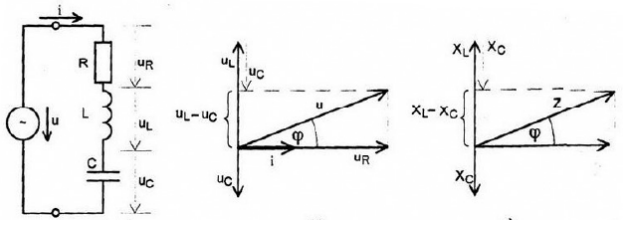


(b) Párhuzamos R-C áramkör impedanciájának és fázisszögének változása

Soros RLC Másnéven *sávzáró szűrő*. A soros kapcsolás miatt mindegyik elemen ugyanaz az I áram folyik át, tehát az *áramerősség a közös mennyiség*. Az impedancia háromszög alalpján:

$$Z = \sqrt{R^2 + (X_L - X_C)^2}$$

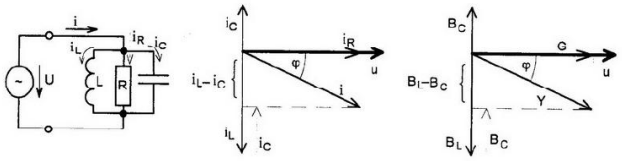
ahol X_L és X_C a tekrecs, ill. a kondenzátor reaktanciája.



9. ábra. Soros R-L-C kapcsolás és vektor diagramja

Párhuzamos RLC Másnéven *sávátereszt ő szűrő.* Itt a *feszültség* a közös mennyiség. Minden áramot (I_R, I_L, I_C) a közös feszültséggel osztva admittancia háromszöget kapunk, melynek alapján:

$$Y = \sqrt{G^2 + (B_L - B_C)^2}$$



10. ábra. Párhuzamos R-L-C kapcsolás és vektor diagramja

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

10.2.1. Dióda

Egy P és Egy N réteget tartalmaz. Jelölése kapcsolási rajzon: 

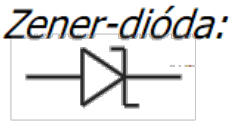
Működését egyszerűen jellemezhetjük: az egyik irányban engedi folyni az áramot, a másik irányban nem. A jelölés szerint a háromszög irányában folyhat az áram, visszafelé nem. Felhasználásának legáltalánosabb módja az *egyenirányítás.*

A dióda legfőbb jellemzői:

- rajta átfolyó maximális áramerősség
- rajta eső feszültség
- felhasználási terület

A diódák általában henger alakúak, két kivezetésük a henger két véglapján található, a rajta levő jelzés alapján megállapíthatjuk melyik vezeték a katód. A diódán az áram az anód irányából a katód irányába folyik (nyitó irányú kapcsolás). Záró irányú kapcsolás esetén elenyészően kicsi áram folyik keresztül a diódán.

A dióda működőképességét ellenállásmérő műszerrel ellenőrizhetjük a következőképpen: mindkét irányban megmérjük az ellenállását. Ha azt tapasztaljuk, hogy az egyik irányban mutat valamekkora ellenállást, a másikban pedig közel végtelen ellenállású akkor a dióda működőképes. A félvezetőkben a szabad töltéshordozók száma és anyag vezetőképessége a hőmérséklettel arányosan változik. Félvezető tulajdonsággal rendelkeznek az alábbi anyagok nagy tisztaságban: germánium (Ge), szilícium (Si), szelén (Se), valamint néhány vegyület: galliumarzenid (GaAs), indiumfoszfid (InP) stb. Nyitóirányú és záróirányú feszültség.(???)



Zener-dióda
Stabilizál. Adott nagyságú záróirányú feszültségnél hirtelen megnő a félvezető dióda árama. Konstrukciótól függően különböző letörési feszültségek vannak

Schottky dióda nagyon gyorsan nyitó/záró dióda típus. Tápegységekben gyakran találkozunk velük.

10.2.2. Alapkapcsolások

Tétel (Kirchhoff I. törvénye)**.** *Egy csomópontba befolyó áramok összege megegyezik az onnan elfolyó áramok összegével. (csomóponti törvény)*

$$\Sigma I = I_1 + I_2 + I_3 + \dots$$

Tétel (Kirchhoff II. törvénye)**.** *Bármely zárt hurokban az áramköri elemeken lévő feszültségek előjel helyesen vett összege nulla. (hurok törvény)*

$$\Sigma U = 0$$

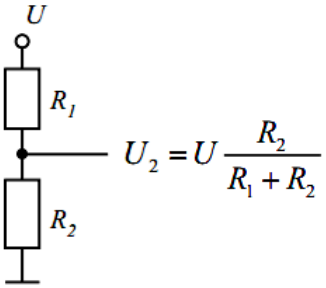
Soros kapcsolás Soros kapcsolásban ugyanaz az áram folyik át minden ellenálláson, a feszültségek összeadódnak. A sorosan kapcsolt ellenállások eredőjét az ellenállások összegzésével kapjuk, így az eredő nagyobb lesz bármely elem értékénél.

$$R_e = R_1 + R_2 + R_3 \quad L_e = L_1 + L_2 + L_3 \quad \frac{1}{C_e} = \frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}$$

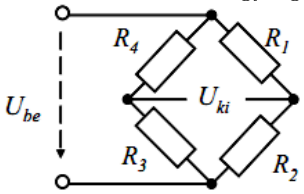
Párhuzamos kapcsolás Párhuzamos kapcsolásban azonos feszültség lép fel minden ellenálláson, az áramerősségek pedig összeadódnak. A párhuzamosan kapcsolt ellenállások eredőjét az ellenállások reciprokának összegével képezzük, ami még nem az eredőt,hanem annak a reciprokát adja ezért ennek is venni kell még a reciprokát.

$$\frac{1}{R_e} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} \quad \frac{1}{L_e} = \frac{1}{L_1} + \frac{1}{L_2} + \frac{1}{L_3} \quad C_e = C_1 + C_2 + C_3$$

Feszültségosztó Két ellenállás soros kapcsolása. A tápláló feszültség megoszlik az R_1 és R_2 ellenállás között. A feszültség az ellenállásokkal egyenes arányban oszlik meg.



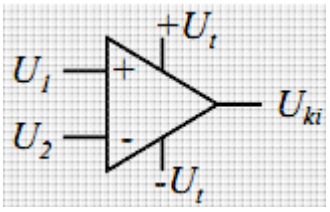
Wheatstone híd A híd olyan négypólus, amelyben az áramköri elemek értékét úgy választjuk meg, hogy a kimeneti feszültség nulla legyen. Ezt nevezzük a híd kiegyenlített állapotának. A Wheatstone híd felhasználható ellenállás mérésre, mivel a szemben elhelyezkedő ellenállások szorzata = a másik két szemben lévő ellenállás szorzatával. Ha a négy ág közül három ismert, a negyedik kiszámítható. Kis áramok mérésére nem alkalmas.



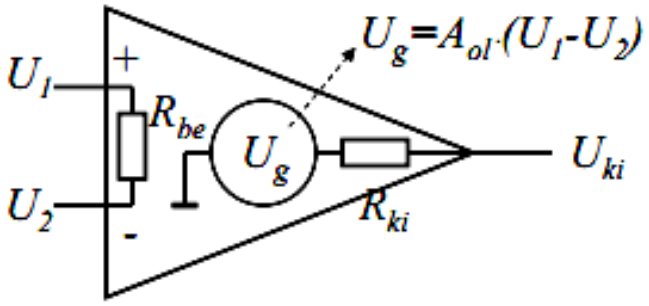
$U_{ki} = 0\,\text{V}$ ha $R_1R_3 = R_2R_4$

10.3. Integrált műveleti erősítők.

A műveleti erősítő kiváló minőségű differenciálerősítő integrált áramkör, amely egyenfeszültség erősítésére is alkalmas. Analóg számítás- és szabályzástechnikai alkalmazásokhoz fejlesztették ki, de igen sokoldalúan alkalmazzuk őket.



Helyettesítő áramkör:



Ideális műveleti erősítő jellemzői:

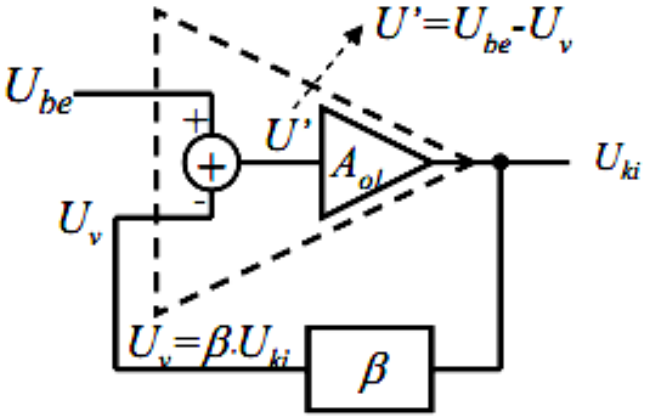
- $U_{\text{ki}} = A_{\text{ol}} \cdot (U_1 - U_2)$
- Végtelen nagy nyílt hurkú feszültségerősítés ($A_{\text{ol}} = \infty$)
- Végtelen nagy bemeneti ellenállás ($R_{\text{be}} = \infty$)
- Zéró kimeneti ellenállás ($R_{\text{ki}} = 0$)

- Végtelen nagy sávszélesség: minden frekvencián ugyanakkora az erősítése
- Zéró ofszet (tökéletesen szimmetrikus felépítés): ha a bemeneti feszültségek megegyeznek, akkor a kimeneti feszültség zéró.

Valódi műveleti erősítők jellemző értékei:

- szimmetrikus tápfeszültségre van szüksége (tipikusan $\pm 15V$)
- $A_{\mathrm{ol}} \approx 10^5$–$10^6$
- $R_{\mathrm{be}} \approx 1$–$200M\Omega$ bipoláris bemenet, $R_{\mathrm{be}} \approx 1000$–$2000M\Omega$ FET bemenet
- $R_{\mathrm{ki}} \approx 10\Omega$
- $f_{\mathrm{min}} \approx 0Hz, f_{\mathrm{max}} \approx$ MHz
- közös módusú elnyomási tényező: CMRR $\approx$ 90–100dB
- véges kimeneti feszültségtartomány: $-U_t < U_{\mathrm{ki}} < +U_t$
- véges maximális jelváltozási sebesség (slew rate): SR ≈ 0.5–$30^V/_{\mu s}$

Negatív visszacsatolás A kimeneti jel egy részét visszavezetik és kivonják a bemeneti jelből, így erősítésre ténylegesen a bejövő jel és az adott hányadban visszacsatolt kimeneti jel különbsége kerül. Így a kimenet megváltoztatása a negatív visszacsatolás révén ellene hat az U' különbség növekedésének. Állandó bemeneti jel esetén a kimeneti feszültség is stabil értékre áll be. a negatív visszacsatolást alkalmazó áramköröknél a műveleti erősítő két bemeneti feszültségének különbsége a nagy nyílthurkú erősítés miatt igen kicsi.



10.4. Tápegységek.

Kapcsoló üzemű tápegységek A korábban ismertetett tápegységek (hálózati transzformátor + egyenirányító + áteresztő tranzisztor) hatásfoka csak 25-50%. Az áteresztő tranzisztor vesztesége jelentősen csökkenthető, ha kapcsolóval helyettesítjük. A transzformátor vesztesége (és mérete) pedig úgy csökkenthető, ha nagyfrekvenciás (20kHz-200kHz) váltakozó feszültséget transzformálnak.

A kapcsoló üzemű tápegységekben (switch-mode power supply (SMPS)) a feszültségszabályozást egy teljesítménytranzisztor változtatható kitöltési tényezővel történő egymást követő bekapcsolásával (teljes telítési állapot) és kikapcsolásával (lezárt állapot) valósítják meg. A terhelésnek megfelelő kitöltési tényezőjű gyorsan ismétlődő (10kHz–100kHz) be- és kikapcsolás a szűrés után a kívánt nagyságú egyenfeszültséget biztosítja a kimeneten

A kapcsolóüzemű (feszültség csökkentő) tápegység működési elve: A K kapcsolót ciklikusan (T periódusidővel) kapcsolgatják: $\mathrm{T_{be}}$ ideig az 1-es, $\mathrm{T_{ki}}$ ideig a 2-es potícióba kapcsolják ($T = Tbe + T_{\mathrm{ki}}$).

10.5. Mérőműszerek.

Feszültségmérő igen nagy belső ellenállású mérőműszer. Párhuzamosan kapcsolandó a mérendő alkatrésszel.

Áramerősségmérő igen kis belső ellenállású mérőműszer, az áramkörbe sorosan kapcsolandó.

Digitális multiméter Elsősorban egyenfeszültség, egyenáram, ellenállás és kapacitás mérésére használatos. Váltóáram és váltófeszültség effektív értékének mérésére is használható, de csak kisebb frekvenciákon.

Analóg multiméterek elektronikus elven működő mérőműszerek,villamos mennyiségek mérésére alkalmasak, a mért mennyiség kijelzése analóg (pl. mutató) műszerrel történik.

Digitális multiméterek többfunkciós mérőműszerek, a mért mennyiség számjegyes kijelzővel történik, ehhez mindegyik tartalmaz egy beépített A/D átalakítót és számkijelzőt.

Oszcilloszkóp Mind feszültség, mind váltófeszültség mérésére alkalmas. A hagyományos analóg oszcilloszkóp képes megjeleníteni egy periodikus jel időbeli változását és így alkalmas a váltakozófeszültség olyan paramétereinek mérésére, mint pl. a periódusidő, felfutási idő, amplitúdó. Csak periodikus jeleket képesek stabil képpel megjeleníteni. A tárolófunkciós analóg oszcilloszkópok és a digitális oszcilloszkópok egyedi impulzusok megjelenítésére is alkalmasak.

Analóg oszcilloszkópok Villamos jelek vizuális vizsgálatát teszi lehetővé egy katódsugárcsöves kijelző segítségével.

Digitális oszcilloszkópok Villamos jelek vizuális vizsgálatát teszi lehetővé egy folyadékkristályos kijelző segítségével.

Szkópméterek egy kétsugaras digitális tárolóoszcilloszkóp és egy digitális multiméter kombinációja, a mérési funkciók automatikusan a legjobb üzemi állapotra állítódnak be.

11. Digitális Technika

Logikai függvények kapcsolástechnikai megvalósítása. Digitális áramköri családok jellemzői (TTL, CMOS, NMOS). Különböző áramköri családok csatlakoztatása. Kombinációs és szekvenciális hálózatok. A/D és D/A átalakítók.

11.1. Logikai függvények kapcsolástechnikai megvalósítása.

Egy-egy alapáramkör megvalósítására több áramkörtechnikai megoldás született, ezek a teljesítményfelvételben, tápfeszültségigényben, H (high) és L (low) szint feszültségben, sebességben és a kimeneti terhelhetőségben térnek el egymástól. Az áramkörcsaládok helyes megválasztásához ismernünk kell a belső felépítésüket és a bemeneti-kimeneti terhelhetőségüket is. Egy logikai kapu minden bemeneti állapotához meghatározott kimeneti állapot tartozik. A logikai kapu által megvalósított függvény nem egyértelmű, ha a szintállapot és a logikai állapot közötti kapcsolat nincs tisztázva. Ez az összerendelés lehet: pozitív logika (H=1,L=0) és lehet negatív logika (H=0,L=1) ,attól függően, hogy azt az áramkörcsaládot használjuk, amelyikkel egyszerűbb a kapcsolás. Ha negatív logikára térünk át a függvényeket a következőképpen kel megcserélnünk:
Nem-vagy $\leftrightarrow$ Nem-és , vagy $\leftrightarrow$ és , nem $\leftrightarrow$ nem
A fizikai megvalósításukhoz: kapcsolókat,komparátorokat,tranzisztorokat használunk. A gyakorlatban szinte minden logikai kaput Nem-és kapukkal realizálnak, olcsósága miatt.

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

11.2.1. TTL — Tranzisztor-tranzisztor logika

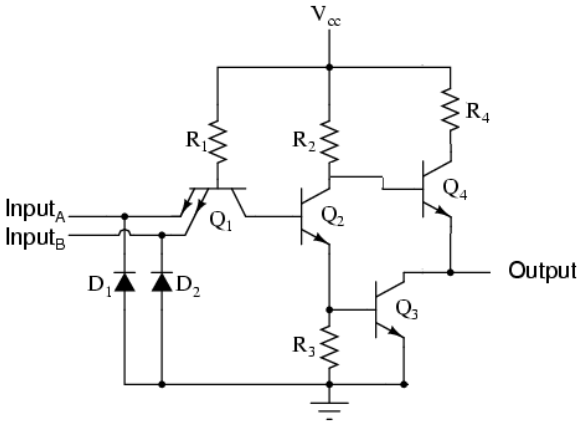
Legnagyobb típus-választékú, univerzális célra készülő bipoláris integrált áramkör rendszer. A bemenetet egy többemitteres tranzisztor szolgáltatja. Kis sebességű, így nagyobb a késleltetés az egyes kapuknál. A fogyasztása a többi családhoz képest elég nagy, de az órajel emelkedésével csak kis mértékben emelkedik és az elektromos kisülések ellen is kellően védett.

Kis és közepes bonyolultságú integrált áramkörökben a legelterjedtebb bipoláris áramkör. A logikai áramkörökben a kapcsolási és terjedési-késleltetési idők szabják meg a sebességet, a kapunkénti disszipált teljesítmény (fogyasztás) is fontos korlátot szab a rendszernek. Típusai:

- H (high speed)
- S (schottky-diódás)
- L (low power)
- LS (low power schottky)

TTL család	Egy kapura eső fogyasztás (mW)	Terjedési-késleltetési idő (ns)
normál	20	10
H TTL	30	6
S TTL	20	3
L TTL	2	35
LS TTL	2	15

TTL nem-és kapu felépítése és működése:
$U_t = +5V$, L szint: $0V - 0{,}8V$, H szint: $2{,}4V - 5V$



Ha a Q_1 tranzisztor bármelyík emitter kivezetésére 0 logikai szint kerül, a kimeneten 1-es logikai szint lesz a Q_4 tranzisztor és az R_2 ellenállás miatt. Ha a Q_1 tranzisztor mindkét emitter kivezetésére 1 logikai szint kerül, akkor a Q_2 tranzisztor kinyit, a a Q_4 tranzisztor lezár és a Q_3 tranzisztor nyitott állapota miatt alacsony logikai szintre kerül a kimenet.

11.2.2. MOS — Metal-Oxid-Semiconductor

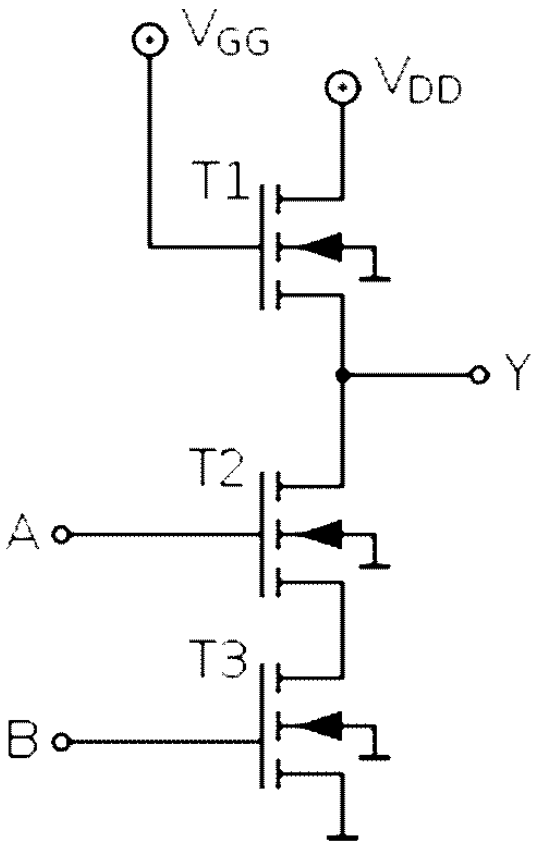
A MOS áramkörök bemenetei érzékenyek a túlfeszültséggel szemben, a megengedettnél nagyobb GATE feszültség miatt átüthet a GATE alatti vékony oxidréteg. Ennél a családnál is tranzisztorokat használnak, de a bipoláris tranzisztorok helyett a fémoxid-félvezetőből készült térvezérlésű FET, vagyis Field-effect tranzisztorokat. Ezáltal sokkal nagyobb műveleti sebességet tudnak elérni.

11.2.3. N-MOS — N csatornás MOS (térvezérlésű) tranzisztor

Csak N-csatornás MOSFET-eket használ, ezáltal csak a magas jelszintet tudja stabilan megjeleníteni. Kisebb a zavarvédettsége és nagyobb a fogyasztása, de a TTL családhoz képest hasonló a CMOS családhoz, csak egyszerűbb kialakítású, ezért olcsóbb is.

Felületigénye jóval kisebb, mint a TTL-nek, gyártási technológiája is gyorsabb (kevesebb művelet), jelentősen kisebb a teljesítményfelvétele. Az N-csatornás tranzisztorok működésében részt vevő Negatív töltéshordozók mozgékonysága majdnem 3-szor nagyobb,mint a (régi technikája miatt kiszorított) P-csatornás tranzisztorok Pozitív töltéshordozóinak mozgékonysága, ezért az N-MOS áramkörök kisebb Gate-kapacitása miatt nagyobb sebességre képesek, csökken a tranzisztor U TO küszöbfeszültsége is, amely alacsonyabb tápfeszültség alkalmazását teszi lehetővé,emiatt könnyebb az N-MOS tranzisztorokat könnyű illeszteni a TTL áramkörökhöz.

N-MOS nem-és kapu felépítése és működése:



A T_1 tranzisztort R_D (100KΩ) ellenállás helyett alkalmazzák a térkihasználás és a nagyobb drain ellenállás miatt. Az Y kimeneten csak akkor jelenik meg alacsony logikai szint, ha a T_2 és T_3 tranzisztor is vezet, azaz A-ra és B-re is logikai magas szintet rakunk.

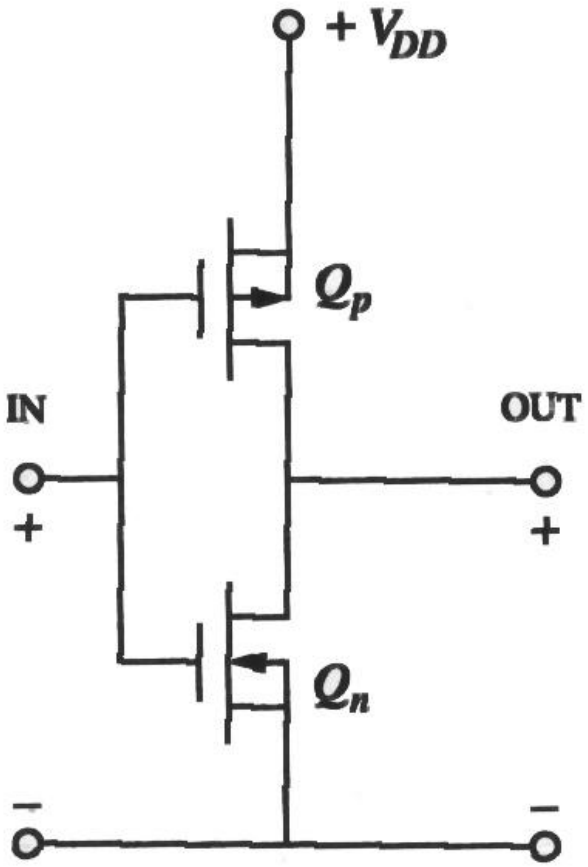
A MOS kapu egyenáramú bemeneti ellenállása nagyon nagy értékű. A bemeneti feszültség változása a gate-kapacitást töltő és kisütő áramot hoz létre. Ez a rövid idejű áramimpulzus nagyobb, mint a szivárgási áram. Ennek ellenére úgy lehet venni, hogy a MOS tranzisztor nem terheli le az előző kapu kimenetét. N-MOS-ok terjedési-késleltetési ideje 15 ns körüli, a teljesítményfelvételük pár szár μW-ig emelkedik.

11.2.4. C-MOS — Komplementer MOS tranzisztor

A működési elve az, hogy N- és P-csatornás MOSFET tranzisztorokat is alkalmaz a logika megvalósításához. A nagy sebesség mellett a fogyasztása is sokkal kisebb a TTL családhoz képest, és a zavarvédettsége is nagyon jó, továbbá széles a működési tápfeszültség tartománya. Egyetlen nagy hátránya, hogy a frekvencia emelkedésével nő a fogyasztása. Működése: egyik helyzetben a felső, P-csatornás tranzisztor nyitott és a kimenetet a pozitív tápfeszültséggel köti össze. A másik helyzetben az alsó, N-csatornás tranzisztor nyit ki és a kimenetet a 0V-tal köti össze. Tehát alacsony jelre a P-MOSFET, magas jelre az N-MOSFET nyit.

A C-MOS-t P-csatornás és N-csatornás növekményes MOS tranzisztorok alkotják. Jellegzetességei a rendkívülien kis áramfogyasztás, széles működési tápfeszültség-tartomány és a nagy zavarvédettség. A C-MOS áramkörök alapeleme az inverter.

C-MOS nem-és kapu felépítése és működése:



Alacsony logikai bemeneti fesztültségnél az N-csatornás Q_n tranzisztor lezárt állapotban van és a P-csatornás Q_p tranzisztor nyitott állapotban így a kimenet logikai magas szinten lesz (megközelítőleg V_{DD} értékű). Ha a bemenetre logikai magas szint kerül, akkor Q_n tranzisztor kinyit és Q_p tranzisztor lezár, ezáltal logikai alacsony szint kerül a kimenetre. Ha a működési frekvencia megnövekszik, vele együtt nő a teljesítményfelvétel is a tranzisztorok kapcsolási ideje miatt kialakuló tápáramfogyasztás miatt.

A C-MOS áramkörök tápfeszültsége +3 és +15 V közötti értéket vehet fel. Egy nem-és kapu terjedési késleltetési ideje átlagosan 25 ns, a nyugalmi teljesítmény: 50 nW

A C-MOS áramkörök nagy jelentőségű változata az SOS (Silicon on Sapphire). Szilícium helyett zafír hordozóra alakítják a a C-MOS-okat. Az áramköri elemek között a szigetelési ellenállás nagyon nagy értékű,ezáltal csökken a hordozók kapacitása és nagyságrendekkel nő a sebesség is (1-2 ns). Jelentősen csökken az áramkör nyugalmi teljesítménye is.

11.3. Különböző áramköri családok csatlakoztatása.

A logikai függvények fizikai megvalósításához a gyártó cégek bizonyos alapáramkör választékot (családokat) alakítanak ki. Az áramkörök összekapcsolását a katalógusokban található jellemző adataik közlése és egyeztetése teszi lehetővé. A digitális technikában a pozitív logika terjedt el, és az áramkörök pozitív feszültségrendszerben dolgoznak. Amikor csatlakoztatjuk a különböző áramkörcsaládokat, a következő jellemzőket kell megvizsgálni:

1. Tápfeszültség az áramkör működéséhez szükséges feszültség

2. Logikai szintek U_{Hmin} és U_{Hmax} valamint U_{Lmin} és U_{Lmax} jellemző értékek

3. Zajtartalék, zajérzékenység feszültségingadozás, ami még nem változtatja meg az állapotot

4. Bementeti terhelhetőség egységterhelés, mind alacsony mind magas logikai szinten

5. Kimenteti terhelhetőség károsodás nélkül a kimenetet képes hajtani

6. Teljesítményfelvétel teljesítményigény 50%-os terhelésnél

7. Jelkésleltetési idő a kimenet reagál a bemenet hatására, amihez idő kell

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

Kombinációs hálózatok *időfüggetlen* logikai függvényeket valósítanak meg, memória nélküli logikai áramkörök, a kimeneti logikai változókat az adott időpontban megjelenő bemeneti logikai változók határozzák meg, az áramköröket IC-kel (Integrated circuit) valósítják meg, a logikai alapfüggvényeket megvalósító áramköröket logikai kapuknak nevezzük, egy IC-n belül több logikai kapu is található.

Szekvenciális (sorrendi) hálózatok *időfüggő* logikai függvényeket valósítanak meg, a kimeneti események alakulását a pillanatnyi bemeneti feltételek mellett a korábbi időpillanatokban bekövetkezett kimeneti események is befolyásolják, attól függően, hogy az állapotváltozás hogyan következik be két csoportot különbözetünk meg:

Aszinkron a kimenet előző állapotától való függést visszacsatolással vagy tárolókkal valósítják meg, a kimenet a bemenetre azonnal reagál

Szinkron az állapotváltozás a kimeneten egy engedélyező jel (Clock) hatására, azzal azonos fázisban zajlik le, a kimenet előző állapotától való függést tárolókkal oldják meg.

Kombinációs és szekvenciális hálózatok típusai:
- R-S tároló
- Inverz R-S tároló
- J-K tároló
- T tároló
- D tároló

A/D átalakítók Feladatuk, hogy a bemenetre érkező „A" analóg jelnek megfelelő „D" digitális jelet (bináris számot) állítson elő a kimeneten, a működéshez szükséges egy R referencia (ált. egy U R referenciafeszültség), melyhez a konverter az A analóg mennyiségét viszonyítja és amely a kimeneti feszültség maximális értékét is meghatározza. Az A/D átalakító kvantál (viszonyítási tartományt használ) a digitális jelek előállításához. A digitalizáláshoz elemi lépcsőket kell használni, és minden lépcsőhöz egy digitális mennyiséget (bináris szám) kell rendelni. Az analóg jel annál pontosabban ábrázolható minél kisebb egy elemi lépcső, vagy kvantum nagysága. Az A/D átalakítóknak annál nagyobb a felbontóképessége minél több bit áll rendelkezésre az ábrázoláshoz. Az átalakítók felbontásának növelése az áramköri megvalósítást nehezíti, drágítja. A kis pontosság-igény esetén elég a 8 bit-es (256 elemi lépcső), nagyobb pontosságot biztosítanak a 10,12,14 bites átalakítók és nagy pontosságú rendszerekben 16,18,20 biteseket alkalmaznak.

D/A átalakítók Feladatuk, hogy a bemenetre érkező „D" digitális jelnek megfelelő „A" analóg jelet (általában feszültséget vagy áramot) állítson elő a kimeneten, működéséhez szükséges egy U_R referenciafeszültség (nagyon pontos feszültségforrás), amelyből a kimeneti feszültséget származtatjuk és ez határozza meg a kimeneti feszültség maximális értékét (végkitérését) is. A digitális technikában többnyire bináris alakban állnak rendelkezésre, valamilyen meghatározott kódban kifejezve, ezt a kódot a D/A átalakítónak ismerni kell, csak ezeket a meghatározott bináris kódokat tudják analóg jellé alakítani. Az ideális D/A átalakítók kimeneti jele egyenesen arányos a bemenetükre digitálisan adott szám értékével. A pontosságot itt is az elemi lépcsők száma határozza meg, minél kisebb egy ilyen lépcső, annál pontosabb az átalakítás. Vannak soros és párhuzamos működésű átalakítók és megkülönböztetünk közvetlen és közvetett elvű átalakítókat.

12. Távközlő hálózatok

Fizikai jelátviteli közegek. Forráskódolás, csatornakódolás és moduláció. Csatornafelosztás és multiplexelési technikák. Vezetékes és a mobil távközlő hálózatok. Műholdas kommunikáció és helymeghatározás.

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

Markov-láncok, születési-kihalási folyamatok. A legalapvetőbb sorbanállási rendszerek vizsgálata. A rendszerjellemzők meghatározásának módszerei, meghatározásuk számítógépes támogatása.

13.1. Markov-láncok, születési-kihalási folyamatok.

A Markov-lánc fogalmához legegyszerűbben a független kísérletek fogalmának általánosításával jutunk el. Tekintsük egymás után végrehajtott kísérletek sorozatát. Legyen $E_1, E_2, \ldots, E_i, \ldots$ egy teljes eseményrendszer. Vizsgáljuk az egyes kísérletek eredményét az E_i események bekövetkezése szempontjából. Definiáljuk a $\xi_n \quad (n = 0, 1, 2, \ldots)$ valószínűségi változókat úgy, hogy $\xi_n = i$, ha az n-edik kísérletnél az E_i esemény fordul elő.

Független kísérletek esetén érvényes, hogy

$$P(\xi_n = j | \xi_1 = i_1, \xi_2 = i_2, \ldots, \xi_{n-1} = i_{n-i}) = P(\xi_n = j)$$

minden n-re, és a szóban forgó valószínűségi változók valamennyi lehetséges értéke. A Markov-lánc fogalmához akkor jutunk, ha a fenti valószínűségeknél feltesszük, hogy a ξ_n változó eloszlása függ az előző $\xi_1, \ldots, \xi_{n-1}$ változóktól is.

Ha fennáll minden lehetséges n-re és a változók összes lehetséges értékeire, hogy

$$P(\xi_n = j | \xi_1 = i_1, \xi_2 = i_2, \ldots, \xi_{n-1} = i_{n-i}) = P(\xi_n = j | \xi_{n-1} = i_{n-i})$$

úgy azt mondjuk, hogy az egymást követő kísérletek, ill. a ξ_n valószínűségi változók *egyszerű Markov-láncot* alkotnak.

A Markov-láncok fontos speciális esetét képezik a *homogén Markov-láncok*. Ezeknél a $P(\xi_n = j | \xi_{n-1} = i)$ átmenetvalószínűségek függetlenek az n-től, azaz

$$P(\xi_n = j | \xi_{n-1} = i) = p_{ij}$$

írható.

Fizikai alkalmazásokat szem előtt tartva a Markov-láncokkal kapcsolatban rendszerint a következő terminológia szokásos: az E_i eseményeket a rendszer *állapotainak* nevezzük. a ξ_0 változó $P(\xi_0 = i) = P_i(0)$ eloszlást *kezdeti eloszlásnak* és a $P(\xi_n = j | \xi_{n-1} = i)$ feltételes valószínűséget *átmenetvalószínűségnek* nevezzük. Ha pedig $\xi_{n-1} = i$ és $\xi_n = j$ akkor azt mondjuk, hogy a rendszer az n-edik lépésben *átmenetet* tett.

Ha egy Markov-láncnál ismerjük a kezdeti eloszlást és az átmenetvalószínűségeket, úgy ezek segítségével az összes ξ_n változó eloszlása egyértelműen meghatározható.

13.1.1. Születési-kihalási folyamatok

A Markov-folyamatok egy igen fontos speciális osztálya születési-halálozási folyamatok néven ismert. A definiáló feltétel: minden állapotból csak "szomszédos" állapotba mehet végbe átmenet (± 1-el változhat). Állapottérnek ekkor a nemnegatív egész számok halmazát választjuk (ami nem megy az általánosság rovására), és $X_k = i$ esetében X_{k+1} vagy $i-1$ vagy i vagy $i+1$ lehet.

Ahhoz, hogy egy $X(t)$ Markov-lánc születési-halálozási folyamat legyen, ki kell elégítenie az alábbi feltételeket:

1. $$P(X(t+h) = k+1 | X(t) = k) = \lambda_k h + o(h)$$

2. $$P(X(t+h) = k-1 | X(t) = k) = \mu_k h + o(h)$$

3. $$P(X(t+h) = k | X(t) = k) = 1 - (\lambda_k + \mu_k)h + o(h)$$

4. $$P(X(t+h) = m | X(t) = k) = o(h) \quad |m-k| > 1$$

ahol h egy tetszőleges kis intervallumot jelent, $o(h)$ pedig olyan mennyiséget jelöl, amely gyorsabban tart 0-hoz, mint h, ahogy h nullához tart, vagyis

$$\frac{o(h)}{h} \to 0, \quad \text{ha} h \to 0$$

Vegyük észre, hogy λ_k, μ_k pozitív mennyiségek függetlenek az időtől. A λ_k-kat *születési intenzitásnak*, a μ_k-kat pedig *halálozási intenzitásnak* nevezzük.

Jelöljük $P_k(t)$-vel annak a valószínűségét, hogy a folyamat a t időpillanatban a k állapotban van, vagyis

$$P_k(t) = P(X(t) = k)$$

Ezt szokás *abszolút valószínűségnek* is nevezni. Ezen valószínűségek kiszámításához figyelembe kell venni a következőket: A $t + h$ időpillanatban az $X(t)$ k állapotban van akkor és csak akkor, ha az alábbi feltételek teljesülnek:

1. t időpillanatban a folyamat a k állapotban van, és a $(t, t + h)$ időintervallumban változás nem következik be
2. t időpillanatban a folyamat a $k - 1$ állapotban volt, és a k-ba történt átmenet
3. t időpillanatban a folyamat a $k + 1$ állapotban volt, és a k-ba történt átmenet
4. $(t, t + h)$ alatt 2 vagy több átmenet történt.

Látható, hogy az 1-3 feltételek kölcsönösen kizárják egymást és a 4. eset valószínűsége $o(h)$. Világos, hogy t minden értékére teljes eseményrendszerről van szó, így:

$$\sum_{k=0}^{\infty} P_k(t) = 1$$

Az előbbi feltételek teljesülése után mára felírhatjuk a $P_k(t + h)$ valószínűséget:

$$P_k(t+h) = P_k(t)\{1 - \lambda_k h - o(h)\} + P_{k-1}(t)\{\lambda_{k-1}h + o(h)\} + P_{k+1}(t)\{\mu_{k+1}h + o(h)\} + o(h), \quad k \geq 1$$

13.2. A legalapvetőbb sorbanállási rendszerek vizsgálata.

13.2.1. Rendszerjellemzők

Kendall-féle jelölés: N/A/B/m/K
N: igényforrások száma
A: beérkezési időközök eloszlása
B: kiszolgálási időközök eloszlása
m: a kiszolgálók száma
K: a várakozási sor kapacitása

13.2.2. M/M/1

Az M/M/1 rendszer a legegyszerűbb nemtriviális rendszer. A beérkezési folyamat λ **paraméterű Poisson–folyamat**, vagyis a beérkezési időközök λ *paraméterű exponenciális eloszlású* valószínűségi változók.

A kiszolgálási időközök μ paraméterű exponenciális eloszlású valószínűségi változók. feltesszük továbbá, hogy a beérkezési időközök és a kiszolgálási idők egymástól független valószínűségi változók.

Az ergodikusság szükséges és elégséges feltétele az M/M/1 sor eseten egyszerűen

$$\lambda < \mu, \quad P_0 = 1 - \frac{\lambda}{\mu}$$

A kihasználtsági tényező: $\rho = \frac{\lambda}{\mu}$
A k.-adik állapot valószínűsége:

$$P_k = (1 - \rho)\rho^k \quad k = 0, 1, 2, \ldots$$

A rendszer jellemzői

A rendszerben tartózkodó igények átlagos száma $N = \frac{\rho}{1-\rho}$

A rendszerben tartózkodó igények átlagos számának szórásnégyzete $\sigma_N^2 = \frac{\rho}{(1-\rho)^2}$

A várakozó igények átlagos száma (átlagos sorhossz) $\overline{Q} = \overline{N} - \rho = \frac{\rho^2}{1-\rho}$

A szerver kihasználtsága $U_s = 1 - P_0 = \frac{\lambda}{\mu} = \rho$

A kiszolgáló átlagos foglaltsági periódushossza: $E_\delta = \frac{1}{\lambda} \cdot \frac{\rho}{1-\rho} = \frac{1}{\lambda}\overline{N} = \frac{1}{\mu-\lambda}$

Little-formulák

$$\lambda T = \lambda \frac{1}{\mu(1-\rho)} = \frac{\rho}{1-\rho} = \overline{N}$$

$$\lambda \overline{W} = \lambda \frac{\rho}{\mu(1-\rho)} = \frac{\rho^2}{1-\rho} = \overline{Q}$$

13.2.3. M/M/1/K

Rögzített a várakozó igények számának maximuma. Felteszük, hogy a rendszerben legfeljebb K igény tartózkodhat (beleértve a kiszolgálás alatt álló igényt is). Az ezen felül érkező igények nem kerül be a rendszerbe, azonnal távozik, nem kerül kiszolgálásra. Továbbra is **Poisson–folyamat** szerint érkeznek az igények, azonban csak azok az igények léphetnek be a rendszerbe, amelyek érkezésekor a rendszer állapota kisebb, mint K.

$$\lambda_k = \begin{cases} \lambda, & \text{ha } k < K \\ 0, & \text{ha } k \geq K \end{cases}$$

$$\mu_k = \mu \quad k = 1, 2, \ldots, K$$

Látszik, hogy ez a rendszer mindig ergodikus, mert állapottere véges. Továbbá:

$$P_k = \begin{cases} P_0 \prod_{j=0}^{k-1} \frac{\lambda}{\mu} = P_0 (\frac{\lambda}{\mu})^k & \text{ha } k \leq K \\ 0 & \text{ha } k > K \end{cases}$$

$$P_0 = \frac{1 - \lambda/\mu}{1 - (\lambda/\mu)^{K+1}}$$

K=1 esetén a M/M/1/1 rendszer azt jelenti, hogy egyáltalán nincs várakozás.

13.2.4. M/M/N

A rendszer n db kiszolgálócsatornával (szerverrel) van ellátva. Az ergodikusság feltétele: $\frac{\lambda}{n\mu} < 1$
Ha $k < n$:

$$P_k = P_0 (\frac{\lambda}{\mu})^k \frac{1}{k!}$$

Ha viszont $k \geq n$:

$$P_K = P_0 (\frac{\lambda}{\mu})^k \frac{1}{n! n^{k-n}}$$

13.3. A rendszerjellemzők meghatározásának módszerei, meghatározásuk számítógépes támogatása.

13.3.1. M/M/n/n – Elang-féle veszteséges rendszer

Az n csatornás rendszerbe Poisson–folyamat szerint érkeznek az igények. Ha van üres csatorna (vagy szerver), az igény kiszolgálása exponenciális időtartalmú μ paraméterrel. Ha minden kiszolgálóegység foglalt, akkor az igény elvész, azaz nincs sorban állás. Ezen probléma a tömegkiszolgálás egyik legrégebbi problémája, mellyel a XX. század elején a telefonközpontok kihasználtságával kapcsolatban foglalkozott A. K. Erlang és C. Palm.

$$\lambda_k = \begin{cases} \lambda & \text{ha } k < n \\ 0 & \text{ha } k \geq n \end{cases}$$

$$\mu_k = k\mu \quad k = 1, 2, \ldots, K$$

Azt mondjuk a rendszer k állapotban van, ha k szerver foglalt, azaz ha k igény tartózkodik a rendszerben.

A folyamat stacionárius eloszlása:

$$P_k = \begin{cases} P_0 (\frac{\lambda}{\mu})^k \frac{1}{k!} & \text{ha } k \leq n \\ 0 & \text{ha } k > n \end{cases}$$

A normalizáló feltétel miatt:

$$P_0 = (\sum_{k=0}^{n} (\frac{\lambda}{\mu})^k \frac{1}{k!})^{-1} \implies P_k = \frac{\frac{\rho^k}{k!}}{\sum_{i=0}^{n} \frac{\rho^i}{i!}} \quad k \leq n$$

A rendszer egyik jellemzője a $P_n = \frac{\rho^n}{n!}/\sum_{k=0}^{n} \frac{\rho^k}{k!}$ valószínűség, melyet először Erlang vezetett be 1917-ben és *Erlang–féle veszteségformula* vagy *Erlang–féle B formula* néven ismert. Általában $B(n, \lambda/\mu$ szimbólummal jelölik. A P_n valószínűség annak a valószínűsége stacionárius esetben, hogy egy újonnan érkező igényt nem fogad a rendszer, azaz az igény elveszik.

Ez a rendszerjellemző meghatározható rekurzív módon, így a köv. formulával számítógép segítségével gyorsan számolható a $B(n, \rho)$:

$$B(0, \rho) = 1$$
$$B(1, \rho) = \frac{\rho}{1 + \rho}$$
$$B(n, \rho) = \frac{\rho B(n-1, \rho}{1 + \rho B(n-1, \rho)}$$

További számítógépes segítség a rendszerjellemzők meghatározására a szimuláció. A szimulációs folyamat után statisztikai eszközökkel határozzuk meg a rendszerjellemzőket.

14. Adatbiztonság

Fizikai, ügyviteli és algoritmusos adatvédelem, az informatikai biztonság szabályozása. Kriptográfiai alapfogalmak. Klasszikus titkosító módszerek. Digitális aláírás, a DSA protokoll.

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

Definíció (Adatvédelem). *azon fizikai, ügyviteli és algoritmikus eszközök együttes felhasználását értjük, amelyek segítségével a véletlen adatvesztések és szándékos adatrongálódások és információ kiszivárogtatások megelőzhetők, vagy jelentős mértékben megnehezíthetők*

Fizikai adatvédelem két lényegi dolgot takar: Egyrészt biztosítani kell az optimális, de legalább a még elfogadható **üzemi körülményeket** (hőmérséklet, páratartalom, por, tartalék alkatrészek stb.), másrészt pedig a szükséges **vagyonvédelmi intézkedésekről** sem szabad megfeledkezni. Például: Villamos hálózat helyes kialakítása; Szünetmentes tápegységek használata; Megfelelő szerverterem kialakítása(klimatizálás, füstérzékelés, árnyékolás,...); Megfelelő adattároló eszközfajták használata; Betörésvédelem.

Ügyviteli adatvédelem a folyamatok szabályozásának, a szabályzatoknak a kialakítása és védelme. A fizikai adatvédelem önmagában ugyanis nem elegendő. Példa: hiába zárjuk be a szerverszoba ajtaját, ha a portás beengedi azt, aki egy szerszámos táskával érkezvén arról tájékoztatja, hogy 'zsírozni kell a switcheket' (social hacking/engineering). Tehát szükséges **pontosan szabályozni**, hogy **ki**, mikor, **mit** és **hogyan** tehet meg, illetve nem tehet meg. Szükség van **informatikai biztonsági szabályzatra** is, amely mindezt egységes módon áttekinti. Megfelelő felhasználó menedzselési rendet kell kialakítani, hogy a felhasználók, hozzáférési jogosultságaik, munkájukból adódó szerepköreik kezelése összhangba hozható legyen. Példák: Feladat- és jogkörök szétválasztása; Hozzáférések és tevékenységek regisztrálása; Személyazonosítás; Hatáskörök és felelősségek szétválasztása vagy átlapolása.

Algoritmikus adatvédelem feladata olyan programok és eljárások alkalmazása, amelyek segítik az előző két terület feladatait és létrehozzák azokat a számítógépes védelmi funkciókat, amik ezen a területen meggátolják az adatokhoz való illetéktelen hozzáférést és módosítást. Példák: Hálózati azonosítás; **Titkosítás**; Behatolásvédelem; Automatikus adatmentés; Többforrásos adattárolás.

Az informatikai biztonsági szabályrendszer szükségessége: (1) az adatok egyre inkább elektronikus formában jelennek meg; (2) a Szervezetek informatika nélkül működésképtelenek; (3) az informatikai függőség egyre nagyobb; (4) ugyanakkor a fenyegetettség is egyre növekszik; (5) az üzletfolytonossághoz kritikus fontosságú; (6) a kárpotenciál és a kockázati tényezők szervezetenként eltérőek lehetnek!

14.1.1. Biztonsági célok

Alapkövetelmények, amelyek teljesülése az üzemszerű használhatóság előfeltétele:

1. rendelkezésre állás (elérhetőség az arra jogosultak számára)
2. sértetlenség (valódiság)
3. bizalmasság (jellegtől függően)
4. nyomon követhetőség, hitelesség
5. Biztosítékok (az információs rendszer teljességére nézve)

Ez alapján úgy lehet meghatározni az Informatikai Biztonság fogalmát, hogy az akkor áll fenn, ha az információs rendszer védelme az alapkövetelmények szempontjából

- **zárt:** minden fontos fenyegetést figyelembe vesz
- **teljes körű:** a rendszer összes elemére kiterjedő
- **folyamatos:** megszakítás nélküli, az időben változó körülmények ellenére is
- **kockázatarányos:** a feltehető kárérték és a kár valószínűségének szorzata nem haladhat meg egy előre rögzített küszöböt, amely egy üzleti döntés.

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).

Az irányító protokollok a forgalomirányítók között folytatnak adatcserét. Az irányító protokollok segítségével a forgalomirányítók megoszthatják az általuk ismert hálózatokról rendelkezésükre álló információkat a többi forgalomirányítóval, illetve közölhetik más forgalomirányítóktól való távolságukat. A forgalomirányítók a többi forgalomirányítótól kapott információk alapján saját irányítótáblát építenek fel és tartanak karban.

15.1. A távolságvektor alapú forgalomirányító protokollok szolgáltatásai

A távolságvektor alapú irányító algoritmusokat használó forgalomirányítók periodikusan elküldenek egymásnak egy-egy másolatot az irányítótáblájukról. Ezek a rendszeres frissítések viszik át a topológia változásaira vonatkozó információkat a forgalomirányítók között. A távolságvektor alapú algoritmusokat Bellman-Ford algoritmusoknak is nevezik.

Minden forgalomirányító a közvetlen szomszédjaitól kap irányítótáblákat. A B forgalomirányító az A forgalomirányítótól kap adatokat. A B forgalomirányító a kapott adatokhoz valamilyen távolságadatot ad hozzá, például az ugrások számát, így növeli a távolságvektor nagyságát. A B forgalomirányító ezt a módosított irányítótáblát adja tovább szomszédjának, a C forgalomirányítónak. A szomszédos forgalomirányítók között lépésről lépésre ugyanez a folyamat játszódik le.

Az összegyűlt távolságértékek alapján az algoritmus egy adatbázist tart karban a hálózat topológiájáról. A távolságvektor alapú algoritmusokkal azonban nem alkotható teljes kép az összekapcsolt hálózatról, mivel minden forgalomirányító csak a szomszédos forgalomirányítókat látja.

A távolságvektor alapú forgalomirányítást használó forgalomirányítók először szomszédjaikat azonosítják. A közvetlenül csatlakoztatott hálózatok felé vezető interfészek távolsága nulla. A távolságvektor alapú felismerő folyamat előrehaladtával a forgalomirányítók a különféle célhálózatok felé vezető legjobb útvonalakat a szomszédjaiktól kapott adatok alapján ismerik fel. Az A forgalomirányító a B forgalomirányítótól kapott adatok alapján ismeri meg a többi hálózatot. Az irányítótábla minden más bejegyzése akkumulált távolságvektort tartalmaz, amely az adott hálózat adott irányban mért távolságát tükrözi.

Az irányítótábla frissítései a topológia változásait követően történnek meg. A hálózatfelderítő folyamathoz hasonlóan a topológiaváltozásra vonatkozó frissítések is lépésenként jutnak el az egyik forgalomirányítótól a másikig. A távolságvektor alapú irányító algoritmusokat futtató forgalomirányítók a teljes irányítótáblájukat elküldik valamennyi szomszédjuknak. Az irányítótáblák többek között az útvonalaknak az adott mérték szerinti teljes költségét és az egyes hálózatokhoz vezető útvonalak első forgalomirányítójának logikai címét tartalmazzák.

A távolságvektorokat az autópályák kereszteződéseiben látható feliratokhoz hasonlíthatnánk. Minden csomópontban nyíl mutatja a célállomások irányát, távolságukat pedig szám jelzi. Ha továbbhaladunk, akkor újabb nyíl mutatja a helyes irányt, de már kisebb távolság lesz kiírva. Amíg a távolság csökken, addig tudhatjuk, hogy a legjobb úton haladunk.

A Routing Information Protocol (forgalomirányítási információs protokoll, RIP) egy távolságvektor alapú, a világ több ezer hálózatában alkalmazott protokoll. Mivel a RIP nyílt szabványokon alapul, üzembe helyezése pedig rendkívül egyszerű. A RIP korszerű, szabványos változatának, korábban két külön dokumentumban volt a leírása. Az első dokumentum az RFC (Request for Comments) 1058, a másik pedig az STD (Internet Standard) 56 dokumentum.

Legfőbb jellemzői a következők:

- Távolságvektor alapú irányító protokoll.
- Az útválasztás mértékeként az ugrásszámot veszi figyelembe.
- Ha az ugrásszám nagyobb mint 15, a csomagot eldobja.
- Alapbeállítás szerint 30 másodpercenként küld frissítéseket.

Az irányítótábla frissítései periodikusan vagy a távolságvektor alapú protokollt alkalmazó hálózat topológiájának megváltozásakor történnek meg. Az irányító protokollokkal szemben fontos elvárás az irányítótáblák hatékony frissítése. A hálózatfelderítő folyamathoz hasonlóan a topológiaváltozásokra vonatkozó frissítések is lépésenként jutnak el az egyik forgalomirányítótól a másikig. A távolságvektor alapú irányító algoritmusokat futtató forgalomirányítók a teljes irányítótáblájukat elküldik valamennyi szomszédjuknak. Az irányítótáblák többek között az útvonalaknak az adott mérték szerinti teljes költségét és az egyes hálózatokhoz vezető útvonalak első forgalomirányítójának logikai címét tartalmazzák.

A RIP az évek során osztály alapú irányító protokollból – RIP 1-es változat (RIP v1) – osztály nélküli irányító protokollá – RIP 2-es változat (RIP v2) – fejlődött. A RIP v2 újdonságai a következők voltak: Több csomagirányítási információ továbbítása. Hitelesítési eljárások a táblafrissítések biztonságossá tételéhez. A változó hosszúságú alhálózati maszkok (VLSM) támogatása.

A RIP maximálja a forrás és a cél közötti útvonalon megtehető ugrások számát, így megakadályozza a végtelen hurkok kialakulását. Egy-egy útvonalon legfeljebb 15 ugrás lehet. Ha egy forgalomirányító új vagy megváltozott bejegyzést tartalmazó frissítést kap, akkor a benne szereplő mértéket eggyel növeli, ezzel önmagát is egy ugrásnak számolja az útvonalon. Ha emiatt a mérték 15 fölé emelkedik, akkor a rendszer végtelennek veszi, és a hálózati cél elérhetetlen lesz. A RIP más protokollok szolgáltatásai közül is többet támogat. Például a RIP látóhatár-megosztásra és visszatartási időzítők fenntartására egyaránt képes annak érdekében, hogy megakadályozza a helytelen irányítási információk terjedését.

15.2. A maximális ugrásszám megadása

Egy hálózattal kapcsolatos helytelen információk addig járnak körbe, míg ezt valamilyen más folyamat meg nem szakítja. Ebben a végtelenig számolásnak nevezett szituációban a csomagok folyamatosan keringenek a hálózatban, annak ellenére, hogy a célhálózat, nem érhető el. Amíg a forgalomirányítók végtelenig számolnak, a hibás információk miatt irányítási hurok jön létre.

A végtelenig számolás folyamatának megállítását célzó óvintézkedések nélkül valahányszor a csomag egy forgalomirányítón áthalad, az ugrásszám alapú távolságvektor mindig nőni fog. A csomagok az irányítótáblákban szereplő hibás információk miatt keringenek a hálózatban.

A távolságvektor alapú forgalomirányító algoritmusok önjavítóak, de az irányítási hurkok problémája miatt végtelenig számolás következhet be. A probléma megoldására a távolságvektor alapú protokollok egy megadott maximális számot tekintenek végtelennek. Ez a RIP esetén maga az ugrásszám.

Ennél a megoldásnál az irányító protokoll addig engedi az irányítási hurok jelenlétét, ameddig a mérték el nem éri a megengedett maximális értéket. Az ábrán látható esetben a mérték értéke 16 ugrás. Ez nagyobb, mint a távolságvektor számára alapesetben előírt maximálisan 15 ugrás, ezért a csomagot a forgalomirányító eldobja. Ha a mérték meghaladja a maximumértéket, akkor az 1-es hálózat elérhetetlennek minősül, bármi legyen is a maximum túllépésének oka.

15.3. Az irányítási hurkok kialakulásának megelőzése látóhatár-megosztással

A látóhatár-megosztási szabály arra az elvre épít, hogy egy útvonalról nem érdemes információkat küldeni oda, ahonnan a vele kapcsolatos adatok eredetileg érkeztek. Egyes hálózatokban szükség lehet a látóhatár-megosztás letiltására. Irányítási hurok akkor is kialakulhat, ha egy forgalomirányítóhoz visszakerülő hibás információ ellentmond a forgalomirányító által eredetileg terjesztett helyes információnak.

15.4. Az irányítási hurkok kialakulásának megelőzése eseményvezérelt frissítésekkel

Az új irányítótáblák elküldése a szomszédoknak rendszeres jelleggel történik. Például a RIP 30 másodpercenként küld frissítéseket. Az eseményvezérelt frissítések továbbítására az irányítótábla változásainak hatására azonnal sor kerül. A topológia megváltozását észlelő forgalomirányító azonnal frissítési üzenetet küld szomszédjainak, amelyek szintén eseményvezérelt módon értesítik saját szomszédjaikat. Ha egy útvonal meghibásodik, akkor a frissítés elküldése azonnal megtörténik, függetlenül attól, hogy a frissítési időzítő lejárt-e. Az eseményvezérelt frissítéseket az útvonalak mérgezésével együtt alkalmazva biztosítható, hogy minden forgalomirányító tudomást szerezzen az útvonalak meghibásodásáról, függetlenül a visszatartási időzítők állásától. A frissítési hullám végighalad a hálózaton.

15.5. Az irányítási hurkok kialakulásának megelőzése visszatartási időzítőkkel

A végtelenig számolás problémája a visszatartási időzítők segítségével előzhető meg: Amikor egy forgalomirányító arról kap értesítést az egyik szomszédjától, hogy egy korábban elérhető hálózat elérhetetlenné vált, az adott útvonalat elérhetetlennek nyilvánítja, és elindít egy visszatartási időzítőt. Ha az időzítő lejárta előtt ugyanaz a szomszéd arról tájékoztatja a forgalomirányítót, hogy a hálózat újra elérhető, akkor a forgalomirányító elérhetőnek nyilvánítja a hálózatot, és törli az időzítőt. Ha egy másik szomszédtól olyan frissítés érkezik, amely jobb távolságvektor-mértékkel rendelkezik, mint a hálózathoz tartozó eredeti mérték, akkor a forgalomirányító szintén elérhetőnek nyilvánítja a hálózatot, és törli a visszatartási időzítőt. Ha a visszatartási időzítő lejárta előtt olyan frissítés érkezik egy másik szomszédtól, amelynek távolságmértéke rosszabb, akkor a forgalomirányító figyelmen kívül hagyja ezt a frissítést. Azzal, hogy a rosszabb távolságmértékű frissítéseket az időzítő lejártáig figyelmen kívül hagyjuk, több időt biztosítunk a hibát jelző információ egész hálózatban való elterjesztésére.

15.6. A RIP konfigurálása

A RIP irányító protokollként való használata a `router rip` paranccsal engedélyezhető. Ezt követően a `network` paranccsal írható elő a forgalomirányító számára, hogy mely interfészeken futtassa a RIP-et. Az irányító folyamat ezután az egyes interfészeket társítja amegfelelő hálózatcímekkel, majd ezeken az interfészeken megkezdi a frissítési üzenetek küldését és fogadását.

A RIP-et futtató forgalomirányítók minden cél felé csak a legjobb útvonalat tartják nyilván, de azonos költségű útvonalból egy-egy célhoz többet is kezelni tudnak. A legtöbb irányítóprotokoll az idővezérelt és az eseményvezérelt frissítések kombinációját használja. A RIP idővezérelt, de a RIP Cisco implementációja a változások észlelését követően is küld frissítéseket. A topológia megváltozása az IGRP forgalomirányítóknál is azonnali frissítési üzenetet vált ki, függetlenül a frissítési időzítéstől. A RIP-et előzetesen engedélyezni kell, a hálózatokat pedig meg kell adni.

- A többi művelet elhagyható:
- Eltolások megadása az irányítási mértékekhez
- Időzítők beállítása
- RIP-változat kiválasztása
- A RIP-hitelesítés engedélyezése
- Útvonal-összefogás engedélyezése valamely interfészen
- Az IP útvonal-összefogás ellenőrzése
- Az automatikus útvonal-összefogás letiltása
- Az IGRP és a RIP párhuzamos futtatása
- A forrás IP-címek ellenőrzésének letiltása
- A látóhatár-megosztás engedélyezése és letiltása
- A RIP csatlakoztatása WAN-hoz

A RIP engedélyezésének műveletét globális konfigurációs módban az alábbi parancsok kiadásával kell elkezdeni:
`Router(config)#router rip` – A RIP forgalomirányító folyamat engedélyezése `Router(config-router)#network hálózatszám` – Hálózat hozzárendelése a RIP forgalomirányító folyamathoz

15.6.1. A RIP konfigurálásával kapcsolatos általános kérdések

Az irányítási hurkok kialakulásának megelőzésére és a végtelenig számolás megakadályozására a RIP a következő megoldásokat alkalmazza:

- Végtelenig számolás
- Látóhatár-megosztás
- Visszirányú mérgezés
- Visszatartási időzítők
- Eseményvezérelt frissítések

A RIP használatakor a maximális ugrásszám 15. A 15 ugrásnál távolabb lévő célállomásokat a rendszer elérhetetlennek nyilvánítja. A RIP – éppen a maximális ugrásszám miatt – nagyméretű hálózatokban csak korlátozottan alkalmazható, ám mivel a végtelenig számolás problémája segítségével megelőzhető, használatakor a csomagok végtelen körbeutazásától sem kell tartani.

A látóhatár-megosztási szabály arra az elvre épít, hogy egy útvonalról nem érdemes információkat küldeni oda, ahonnan a vele kapcsolatos adatok eredetileg érkeztek. Egyes hálózatokban szükség lehet a látóhatár-megosztás letiltására.
Erre a következő parancs használható: `GAD(config-if)#no ip split-horizon`

A visszatartási időzítők segítségével megelőzhető a végtelenig számolás, viszont nő a konvergencia ideje is. A RIP esetében az alapértelmezett visszatartási idő 180 másodperc. A visszatartási időzítők értéke csökkenthető, ekkor ugyan gyorsul a konvergencia, ám más jellegű problémák jelentkezhetnek. Ideális esetben az időzítő értéke kevéssel nagyobb, mint az összekapcsolt hálózatban lehetséges legnagyobb frissítési idő. Az ábrán például egy négy forgalomirányítóból álló hurok látható. Ha minden forgalomirányító frissítési ideje 30 másodperc, akkor a leghosszabb hurok 120 másodperces. Vagyis a visszatartási időzítőt 120 másodpercnél kicsivel nagyobb értékre kell állítani.

A visszatartási időzítő valamint a frissítés, érvénytelenítés és a törlés időzítőinek beállítására a következő parancsot használjuk:
`Router(config-router)#timers basic update invalid holddown flush [alvásidő]`

Ha kiadjuk a network parancsot egy hálózatra vonatkozóan, akkor a RIP azonnal megkezdi a hirdetések küldését a megadott hálózati címtartományba tartozó interfészeken. Ha a hálózati rendszergazda meg szeretné határozni, hogy mely interfészek továbbítsanak útvonalfrissítéseket, akkor a frissítések küldését egy-egy interfészen a `passive-interface` paranccsal tilthatja le.

Mivel a RIP szórásos protokoll, a hálózati rendszergazda részéről külön konfigurálást igényelhet a RIP szórásra nem alkalmas, például Frame Relay hálózatokon való beüzemelése. Az ilyen hálózatokban a RIP-nek

külön meg kell adni a szomszédos RIP forgalomirányítókat.
Parancs: `neighbor ip address`

Alapbeállítás szerint a Cisco IOS szoftver a RIP 1-es és 2-es változata szerinti csomagokat egyaránt fogadja, de csak 1-es változat szerinti csomagokat küld. Kézi beállítás:

Parancs	Cél
`GAD(config-router)#version {1|2}`	Beállítja, hogy a szoftver 1-es és 2-es verziójú RIP csomagokat is tudjon fogadni és küldeni
`GAD(config-if)#ip rip send version 1`	Beállítja, hogy a szoftver 1 -es verziójú RIP csomagokat küldjön
`GAD(config-if)#ip rip send version 2`	Beállít egy interfészt, hogy csak 2-es verziójú RIP csomagokat küldjön
`GAD(config-if)#ip rip send version 1 2`	Beállítja, hogy a szoftver 1-es vagy 2-es verziójú RIP csomagokat küldjön

Ha szabályozni szeretnénk az adott interfészen keresztül beérkező csomagok feldolgozásának módját, akkor:

Parancs	Cél
`GAD(confiq-if)#ip rip receive version 1`	Beállít egy interfészt, hogy csak 1-es verziójú RIPcsomagokat fogadjon el
`GAD(config-if)#ip rip receive version 2`	Beállít egy interfészt, hogy csak 2-es verziójú RIPcsomagokat fogadjon el
`GAD(config-if)#ip rip receive version 1 2`	Beállít egy interfészt, hogy 1-es és 2-es verziójú RIPcsomagokat is elfogadjon

16. Bevezetés a Cisco eszközök programozásába 1

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

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

A rendszergazdáknak meg kell találniuk annak a módját, hogy megakadályozzák a hálózat jogosulatlan elérését, miközben a belső felhasználók számára lehetővé teszik a szükséges szolgáltatások használatát. Bár a biztonsági eszközök, mint például a jelszavak, a visszahívó berendezések és a fizikai biztonsági eszközök sok segítséget nyújtanak, gyakran nem rendelkeznek az alapvető forgalomszűréshez szükséges rugalmassággal és a rendszergazdák igényeinek megfelelő vezérlési lehetőségekkel. Előfordulhat például a hálózati rendszergazda lehetővé kívánja tenni a felhasználók számára az internet elérését, de nem akarja, hogy külső felhasználók telnettel hozzáférhessenek a LAN-hoz.

A forgalomirányítók alapvető forgalomszűrési lehetőségeket biztosítanak, például hozzáférési listákat (access control list, ACL) az internetes forgalom letiltásához. Az ACL engedélyező és tiltó utasítások sorozata, melyek címekre vagy felsőbb rétegbeli protokollokra alkalmazhatók. Ebben a modulban a normál és kiterjesztett ACL-ek használatáról, mint a hálózati forgalom vezérlési módszeréről fogunk tanulni, továbbá arról, hogyan használhatók az ACL-ek valamilyen biztonsági megoldás részeként.

Ezenkívül a fejezetben tippek, elgondolások, javaslatok és általános útmutatások is szerepelnek az ACL-ek használatával, valamint a létrehozásukhoz szükséges parancsokkal és beállításokkal kapcsolatban. Végül a fejezet példákat mutat a normál és a kiterjesztett ACL- ekre és a forgalomirányítók interfészein történő alkalmazásukra.

Az ACL-ek akár egyetlen sorból is állhatnak, ha a cél egy adott állomásról származó csomagok továbbításának engedélyezése, de tartalmazhatják szabályok és feltételek rendkívül összetett halmazát is, amelyek pontosan megadják az engedélyezett forgalom jellegét, illetve befolyásolják a forgalomirányító folyamatok teljesítményét.

16.1.1. A hozzáférési listák működésének alapelvei

Az ACL-ek feltétellisták, amelyek a forgalomirányító interfészén keresztülhaladó forgalomra vonatkozóan lépnek érvénybe. Ezek a listák írják elő a forgalomirányító számára, hogy a csomagokat fogadja el vagy utasítsa vissza. Az elfogadás és a visszautasítás meghatározott feltételek alapján is történhet. Az ACL-ek segítségével lehetővé válik a forgalom felügyelete, valamint a hálózatról kiinduló és az oda befutó elérések biztonságossá tétele.

ACL-ek minden irányított protokollhoz létrehozhatók, többek közt az internetprotokollhoz (IP) és a hálózatközi csomagcseréhez (IPX) is. Az ACL-ek a forgalomirányítón konfigurálva adott hálózat vagy alhálózat elérhetőségének vezérlésére használhatók.

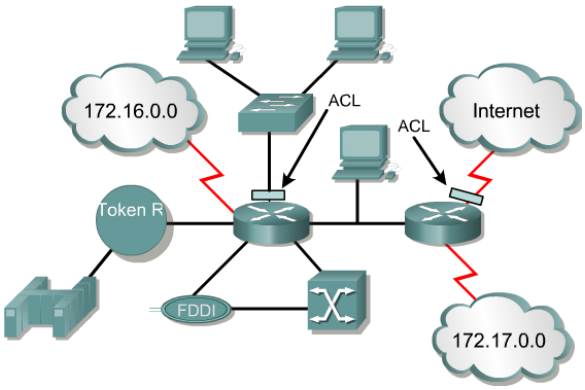
Az ACL-ek úgy szűrik a hálózati forgalmat, hogy az irányított csomagok továbbítását vagy eldobását írják elő a forgalomirányító interfészein. A forgalomirányító minden csomagot megvizsgál annak meghatározásához, hogy azt az ACL-ben meghatározott feltételek szerint továbbítani vagy eldobni kell. Az ACL-ek a forrás- és a célcímekre, a protokollokra és a felsőbb rétegbeli portszámokra nézve adnak meg feltételeket.

Az ACL-eket protokollonként, irányonként és portonként kell létrehozni. Ha szabályozni szeretnénk egy adott interfész forgalmát, akkor a rajta engedélyezett protokollok mindegyikéhez külön ACL-t kell készíteni. Egy ACL egy interfészen egyszerre csak egy irány forgalmát szabályozza. Minden irányhoz külön ACL-t kell létrehozni, egyet a kimenő, egyet pedig a bejövő forgalomhoz. Végül minden interfészen több protokoll és irány is definiálható. Ha például egy forgalomirányítónak két interfésze van, és ezeken IP, AppleTalk és IPX alapú forgalom zajlik, akkor 12 külön ACL-t kell rajta létrehozni: minden protokollhoz egy-egy ACL, szorozva kettővel a kimenő és a bejövő forgalom miatt, szorozva kettővel, vagyis a portok számával.

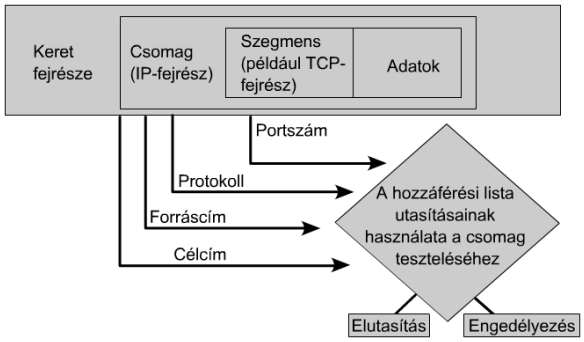
ACL-eket többek között a következő okok miatt szokás létrehozni:

1. Korlátozható a hálózat forgalma, és növelhető a teljesítménye.
2. Biztosítják a forgalom szabályozását.
3. Az útvonalfrissítések továbbítása is korlátozható. Ha a hálózati környezet miatt nincs szükség a frissítésekre, sávszélességet lehet megtakarítani.
4. Alapszintű hálózati hozzáférés-szabályozást biztosítanak. Az ACL-ek engedélyezhetik például a hálózat egy részének elérését egy állomás számára, és megtilthatják egy másiknak. Lehetséges például, hogy az A állomás hozzáférhet a személyzeti osztály hálózatához, míg a B állomásnak ezt megtiltjuk.
5. Eldönthetjük, hogy milyen típusú forgalom továbbítódjon és milyen törlődjön a forgalomirányító interfészein. Például engedélyezhetjük az elektronikus levelek továbbítását, de a telnetet tilthatjuk.
6. A rendszergazdák szabályozhatják, hogy az ügyfelek a hálózat mely részeihez férhetnek hozzá.
7. Ki lehet választani bizonyos állomásokat, és számukra engedélyezni vagy megtiltani a hálózat egy részének elérését.
8. A felhasználók számára engedélyezni vagy tiltani lehet bizonyos szolgáltatások, például az FTP vagy a HTTP elérését.

Ha egy forgalomirányítón nincsenek ACL-ek megadva, akkor a forgalomirányítóba befutó csomagok mindegyike továbbhaladhat a hálózat bármely része felé.



(a)



(b)

11. ábra

16.1.2. ACL-ek létrehozása

Az ACL-ek létrehozása globális konfigurációs módban történik. Számos különböző típusú ACL létezik, így normál, kiterjesztett, IPX, AppleTalk stb. ACL. Amikor egy forgalomirányítón ACL-eket konfigurálunk, akkor egy-egy szám hozzárendelésével mindegyiket egyértelműen azonosítanunk kell. A szám megadja az ACL típusát; ebből következően az adott típushoz tartozó értéktartományba kell esnie:

Protokoll	Tartomány
IP	1-99, 1300-1999, 2000-2699
Kiterj. IP	100-199, 2000-2699
AppleTalk	600-699
IPX	800-899
Kiterj. IPX	900-999
IPX szolgáltatáshirdetés (SAP)	1000-1099

Miután beléptünk a megfelelő parancsmódba és eldöntöttük, hogy milyen típusú listát szeretnénk létrehozni, az `access-list` paranccsal, illetve a szükséges paraméterek segítségével kell megadnunk a hozzáférési lista utasításait.
A hozzáférési listák létrehozása az első lépés.
A második végrehajtandó művelet a listák hozzárendelése a megfelelő interfészekhez.

Az ACL definiálása a következő paranccsal:

```
Router(config) #access-list access-list-number {permit / deny test-conditions}
```

1. lépés: Az ACL-t egy globális utasítás azonosítja. A normál IP-címekhez az 1-99 tartomány van fenntartva. Ez a szám mutatja az ACL típusát. A Cisco lOS 11 .2-es verziójától kezdődően az ACL-ekhez nem csak szám, de név is rendelhető, például oktatasi_csoport. A globális ACL utasítás permit illetve deny kifejezése adja meg, hogy a Cisco lOS szoftver hogyan kezelje a tesztfeltételeket kielégítő csomagokat. A permit általában azt jelenti, hogy a csomag használhat egy vagy több - később meghatározandó - interfészt. A záró kifejezés(ek) az ACL állítás által használandó tesztfeltételeket adják meg.

2. lépés: Ezután alkalmazni kell az ACL-eket egy interfészre az `access-group` paranccsal.
Példa: `Router (config-if) #{protocol) access-group access-list number`
A hozzáférési lista számával azonosított összes ACL utasítás hozzárendelődik egy vagy több interfészhez. Az ACL tesztfeltételeket kielégítő csomagok a hozzáférési csoport bármely interfészét használhatják.

TCP/IP használata esetén az ACL-eket egy vagy több interfészhez lehet hozzárendelni. Az `ip access-group` parancs segítségével a bejövő vagy a kimenő forgalom szűrésére állíthatjuk be.

```
Router(config)#access-list 2 deny 172.16.1.1
Router(config)#access-list 2 permit 172.16.1.0 0.0.0.255
Router(config)#access-list 2 deny 172.16.0.0 0.0.255.255
Router(config)#access-list 2 permit 172.0.0.0 0.255.255.255
```

```
Router(config) #interface ethernet 0
Router(config)#ip access-group 2 in
```

Az access-group parancsot interfészkonfigurációs módban kell kiadni. Amikor egy ACL-t hozzárendelünk egy interfészhez, akkor ki kell választanunk, hogy a bejövő vagy a kimenő forgalomra vonatkozzon. A szűrés tehát az adott interfészre beérkező és a róla távozó csomagokra vonatkozhat. Annak megállapításához, hogy az ACL a bejövő vagy a kimenő forgalmat szűrje, az egyes interfészeket a forgalomirányító belsejéből kell szemlélnünk. Ezt a szemléletet mindvégig meg kell őrizni. A valamilyen interfészen keresztül beérkező forgalmat bejövő ACL, a kimenő forgalmat pedig kimenő ACL alapján szűrjük. A számozott ACL-t létrehozása után hozzá kell rendelni egy interfészhez. Számozott ACL-utasításokat tartalmazó ACL nem módosítható. Előbb törölnünk kell a `no access-list lista-szám` paranccsal, majd újra be kell vinnünk a parancsokat. (`Router(config)#no access-list 2`)
Az ACL-ek létrehozásakor és életbe léptetésekor a következő alapvető szabályokat kell betartani:

1. Irányonként és protokollonként egy ACL-t kell létrehozni.
2. A normál hozzáférési listákat a célhoz a lehető legközelebb kell alkalmazni.
3. A kiterjesztett hozzáférési listákat a forráshoz a lehető legközelebb kell alkalmazni.
4. A kimenő és a bejövő jelzőket úgy kell használni, mintha a forgalomirányító belsejéből néznénk a portokat.
5. Az utasítások feldolgozása sorban, a lista tetejétől az alja felé haladva történik, amíg a forgalomirányító egyezést nem talál. Ha nincs egyezés, a forgalomirányító eldobja a csomagot.
6. Minden hozzáférési lista alján egy implicit deny any (mindent letilt) szabály található. Ez szabály nem jelenik meg az utasításlista alján.
7. A hozzáférési listák utasításait a specifikusabbaktól az általánosabbak felé haladva kell megadni. Az egyes állomásokra vonatkozó tiltásokat kell először megadni, a csoportokra vonatkozó vagy általános szűrőket utolsóként kell elhelyezni.
8. Elsőként az egyezési feltétel vizsgálata történik meg. Az engedélyező vagy tiltó részre kizárólag akkor kerül át a vezérlés, ha az egyezés igaz volt.
9. Soha ne dolgozzunk aktívan működő hozzáférési listával!
10. Először a logikai utasításokat felvázoló megjegyzéseket készítsük el szövegszerkesztővel, a tényleges végrehajtó műveleteket csak ezt követően írjuk meg.
11. Az új sorok mindig a hozzáférési lista végére kerülnek. A no access-list x parancs a teljes listát törli. Számozott ACL-ek sorainak egyenként való hozzáadására vagy eltávolítására nincs lehetőség.
12. Az IP alapú hozzáférési listák a célállomás elérhetetlenségét jelző ICMP-üzenetet küldenek az elutasított csomagok forrásainak, majd a bitszemetesbe dobják a csomagokat.
13. Hozzáférési lista eltávolítását mindig körültekintően kell végezni. Ha a hozzáférési lista aktív interfészre vonatkozik, és eltávolítjuk, akkor az IOS verziójától függően alapértelmezett tiltó szabály léphet érvénybe az interfészen, ami a forgalom teljes leállását okozza.
14. A kimenő szűrők nem vonatkoznak a helyi forgalomirányítóról kiinduló forgalomra.

16.1.3. Normál ACL-ek

A normál ACL-ek az irányítandó IP-csomagok forráscímét ellenőrzik. Az összehasonlítás a hálózati, alhálózati és állomáscím alapján egy egész protokollkészlet számára eredményez engedélyezést vagy tiltást.
Például a Fa0/0 interfészen keresztül beérkező csomagok forráscímét és protokollját egyaránt ellenőrizzük. Ha mindkettő engedélyezve van, a csomagok a forgalomirányítón keresztülvalamelyik kimenő interfészre kerülnek. Ha tiltva vannak, akkor eldobásuk a bejövő interfészen történik meg.

A globális konfigurációs mód `access-list` parancsának normál változatával normál, 1 és 99 közötti számú ACL definiálható. A Cisco IOS Software Release 12.0.1 és újabb változataiban a sorszám 1300 és 1999 között is lehet, így akár 798 normál ACL-t is készíthetünk. Ezt az újabb tartományt kibővített IP ACL-nek nevezzük.

```
access-list 2 deny 172.16.1.1
access-list 2 permit 172.16.1.0 0.0.0.255
access-list 2 deny 172.16.0.0 0.0.255.255
access-list 2 permit 172.0.0.0 0.255.255.255
```

- Hozzáférési lista tartományok: I - 99 és 1300 - 1999
- Szürés csak az IP-forráscim alapján
- Helyettesítö maszkok
- A célhoz legközelebbi portra kell alkalmazni

Vegyük észre, hogy az első ACL-utasításnál nincs megadva helyettesítő maszk. Ilyenkor a forgalomirányító az alapértelmezett 0.0.0.0 maszkot használja, vagyis vagy a teljes címnek egyeznie kell, vagy az ACL ezen sora nem fog illeszkedni, és a forgalomirányító az ACL következő sorára lép tovább.

A normál ACL-utasítások teljes szintaxisa a következő:
`Router(config)#access-list hozzáférési-lista-száma {deny | permit | remark} forrás [forrás-helyettesítő-maszk`

A `remark` (megjegyzés) kulcsszó az ACL-eket könnyebben érthetővé teszi. A megjegyzések hossza nem haladhatja meg a 100 karaktert. Példa:

```
access-list 1 remark Csak Jones állomását engedélyezzük
access-list 1 permit 171.69.2.88
```

Normál ACL-t törölni a parancs no változatával lehet. Ennek szintaxisa:
`Router(config)#no access-list hozzáférési-lista-száma`

Az ip access-group parancs hozzákapcsol egy normál ACL-t egy interfészhez: `Router(config)#ip access-group {`

A táblázat a szintaxisban használt paraméterek leírásait tartalmazza:

Paraméter	Leírás
access-list-nurnber	Az ACL azonosító száma. Decimális szám 1 -99 (normál IP ACL) és 1300 - 1999 (kibővitett IP ACL).
deny	Megtagadja a hozzáférést, ha a feltételek teljesülnek.
permit	Engedélyezi a hozzáférést, ha a feltételek teljesülnek.
remark	A remark (megjegyzés) parancs használata a listák könnyebb megértését és megkeresését segíti.
source	Annak a hálózatnak vagy állomásnak a címe, ahonnan a csomagot elküldik. A forrás kétféleképpen adható meg: 1. 32 bites cím megadása négy részből álló, pontokkal elválasztott decimális formátumban. 2. az any kulcsszó a forrás rövidítése: source-wildcard of 0.0.0.0 255.255.255.55.
source-wildcard	(Opcionális) A forrásra alkalmazandó helyettesítő bitek. A forráshelyettesítő maszkja kétféleképpen adható meg: 1. 32 bites cím megadása négy részből álló, pontokkal elválasztott decimális formátumban. A figyelmen kívül agyandó bitpozíciókba 1-et kell írni. 2. Az any kulcsszó a 0.0.0.0 255.255.255.255 értékű forrás és forráshelyettesítő maszk rövidítése.
log	(Opcionális) A bejegyzésnek megfelelő csomagról egy tájékoztató célú naplózási üzenet jut a konzolra, (A konzolra küldött naplózási üzenetek részletessége a logging console paranccsal vezérelhető.)Az üzenet tartalmazza az ACL számát, a forráscímet, a csomagok számát és azt, hogy a csomagot engedélyezték-e vagy letiltották. Az üzenet az első megfelelő csomag megtalálásakor generálódik, ezután ötpercenként, megmutatva azoknak a csomagoknak a számát is, amelyek a megelőző öt percben lettek engedélyezve vagy elutasítva.

16.1.4. Kiterjesztett ACL-ek

A kiterjesztett ACL-eket a normál ACL-eknél gyakrabban használjuk, mivel szélesebb körű ellenőrzést tesznek lehetővé.
A kiterjesztett ACL-ek a csomagok forrás- és célcímét egyaránt ellenőrzik, illetve a protokollok és a portszámok egyeztetésére is alkalmasak. Ezek a lehetőségek nagyobb szabadságot biztosítanak az ACL által vizsgált adatok körülhatárolására. A csomagok engedélyezése és tiltása forrás, cél, protokolltípus és portszám alapján egyaránt történhet. Egy kiterjesztett ACL például engedélyezheti az elektronikus levelezést a Fa0/0 interfészről megadott S0/0 célok felé, miközben tilthatja a fájlátviteleket és a webböngészést. A csomagok eldobásakor bizonyos protokollok egy visszhangcsomagot küldenek a forrásnak, jelezve, hogy a cél nem érhető el.

Egy-egy ACL-hez több utasítás is konfigurálható. Az utasítások mindegyikének ugyanazt a hozzáférési lista számot kell tartalmaznia, így tud hivatkozni az azonos ACL-en belüli utasításokra. Feltételutasításból tetszőleges számú adható meg, az ilyen utasítások mennyiségét csak a forgalomirányító memóriájának nagysága korlátozza. Természetesen minél több az utasítás, annál nehezebb az ACL megértése és kezelése.

A kiterjesztett ACL-utasítások szintaxisa meglehetősen hosszadalmas is lehet, akár a teljes terminálablakot is kitöltheti. A helyettesítéseknél ugyancsak mód nyílik a host vagy az any kulcsszó használatára a parancsokon belül.

A kiterjesztett ACL-utasítások végén az opcionálisan megadható TCP vagy UDP portszámokkal tovább pontosíthatók a szabályok. A TCP/IP protokollkészlet jól ismert portszámai az ábrán láthatók.

Decimális	Kulcsszó	Leírás	TCP vagy UDP
0		Foglalt	Foglalt
1-4		Nem használatos	
5	RJE	Távoli feladatbejegyzés	TCP, UDP
7	ECHO	Visszhang	TCP, UDP
9	DISCARD	Törlés	TCP, UDP
11	USERS	Aktív felhasználók	TCP, UDP
13	DAYTIME	Nappali időszak	TCP, UDP
15	NETSTAT	Ki van fenn vagy NETSTAT	TCP, UDP
17	QUOTE	A nap tippje	TCP, UDP
19	CHARGEN	Karaktergenerátor	TCP, UDP
20	FTP-DATA	FTP adatok	TCP, UDP
21	FTP	File Transfer Protocol (fájlátviteli protokoll)	TCP, UDP
23	TELNET	Terminálkapcsolat	TCP, UDP
25	SMTP	Simple Mail Transfer Protocol (egyszerű levéltovábbító protokoll)	TCP, UDP
37	TIME	Napszak	TCP, UDP
39	RLP	Resource Location Protocol (erőforrás-hely	TCP, UDP
42	NAMESERVER	Host Name Server (állomásnév-kiszolgáló)	TCP, UDP
43	NICNAME	Ki az?	TCP, UDP
53	DOMAIN	Domain Name Server (tartománynév-kezelő	TCP, UDP

Decimális	Kulcsszó	Leírás	TCP vagy UDP
67	BOOTPS	Bootstrap Protocol Server (rendszerindító protokoll kiszolgáló)	TCP, UDP
68	BOOTPC	Bootstrap Protocol Client (rendszerindító protokoll ügyfél)	TCP, UDP
69	TFTP	Trivial File Transfer Protocol (triviális fájlátviteli	TCP, UDP
75		Bármilyen privát behívási szolgáltatás	TCP, UDP
77		Bármilyen privát RJE szolgáltatás	TCP, UDP
79	FINGER	Finger	TCP, UDP
80	HTTP	HyperText Transfer Protocol (hiperszöveg-átviteli protokoll)	TCP
95	SUPDUP	SUPDUP protokoll	TCP
101	HOSTNAME	NIC állomásnév-kiszolgáló	TCP
102	ISO-TSAP	ISO-TSAP	TCP
110	POP3	Post Office Protocol (postahivatal protokoll)	TCP
113	AUTH	Hitelesítési szolgáltatás	TCP
117	UUCP-PATH	UUCP útvonal-szolgáltatás	TCP
123	NTP	Network Time Protocol (hálózati idő protokoll)	TCP, UDP
133-159		Nem használatos	Nem használatos
160-223		Foglalt	Foglalt
224-241		Nem használatos	Nem használatos
242-255		Nem használatos	Nem használatos

A kiterjesztett ACL-ek logikai műveleteket – egyenlő (equal, eq), nem egyenlő (not equal, neq), nagyobb mint (greater than, gt), kisebb mint (less than, lt) – is képesek végezni a megadott protokollokon. A kiterjesztett ACL-ek hozzáférési lista száma 100 és 199 között lehet. (Az újabb IOS-változatoknál a sorszám a 2000–2699 tartományba is tartozhat.)

Az `ip access-group` paranccsal egy meglévő kiterjesztett ACL köthető hozzá egy interfészhez. Ne feledjük, hogy interfészenként, irányonként és protokollonként csak egy ACL adható meg! A parancs formátuma: `Router(config-if)#ip access-group hozzáférési-lista-száma {in | out}`

17. Bevezetés a Cisco eszközök programozásába 2

A forgalomirányítási táblázatok felépítése, statikus és dinamikus routing összehasonlítása.

17.1. A forgalomirányítási táblázatok felépítése

A forgalomirányítás a csomagok (IP datagramok) továbbítási irányának meghatározásával kapcsolatos döntések meghozatala. A forgalomirányításhoz szükséges információkat ún. forgalomirányítási táblázatban tartjuk nyilván, melynek segítségével gyorsan meghatározható, hogy egy input interfészen érkező csomagot melyik output interfészen kell továbbítani.

Célhálózat	Netmask	Kimenő interfész	Köv. Hop	Metrika

1. A router az input interfészen érkező csomagot fogadja.
2. A router a csomag célcímét illeszti a routing táblázat soraira.
   a) Ha a célcím több sorra illeszkedik, akkor a leghosszabb prefix sort tekintjük illeszkedőnek.
3. Ha nem létezik illeszkedő sor, akkor a cél elérhetetlen, a csomag nem továbbítható??
   a) A csomagot a router eldobja és ICMP hibajelzést küld a feladónak VAGY alapértelmezett átjárón továbbküldi
4. Ha létezik illeszkedő sor, akkor a csomagot az ebben szereplő kimeneti interfészen továbbítjuk (adatkapcsolati rétegbeli beágyazással) a következő, hop-ként megadott szomszédhoz, ill. a célállomáshoz, ha már nincs több hop.

1. A routing tábla sorait prefix hossz szerint csökken, sorrendbe rendezzük. N=1.
   a) Ezzel biztosítjuk, hogy több illeszkedő sor esetén a leghosszabb prefixűt fogjuk eredményként kapni.
2. Ha nem létezik a táblázatban az N. sor, akkor nincs illeszkedő sor és vége.
3. A csomag célcíme és az N. sor hálózati maszkja között bitenkénti AND műveletet hajtunk végre.
4. Ha a bitenkénti AND művelet eredménye megegyezik az N. sor célhálózat értékével, akkor a cím az N. sorra illeszkedik és vége.
5. N=N+1, és folytassuk a 2. pontnál.

Forgalomirányítási táblázatok feltöltése történhet statikus vagy dinamikus módon. Statikusnál direkt módon megadunk egy vagy több sort a táblázatba, míg dinamikus esetben routing protokollok végzik automatikusan, figyelve a hálózatot. Pl.: RIP, OSPF

17.2. statikus és dinamikus routing összehasonlítása

nem adaptív statikus, nem támaszkodhat döntéseiben a mérésekre, a használandó útvonalakat előre definiálják a routerek indulásakor, bekapcsolásakor. Fix-egyutas esetben egyetlen irányba történik az adattovábbítás, központi forgalomirányító táblát alkalmaznak, vonal szakadása esetén 100%-os veszteség várható. A fix többutas esetben alternatív utakat kínál fel a csomópontoknak. Ezek közül egy kívülről meghatározott módon (például random) választ egyet.Ekkor egy esetleges vonalszakadás nem okozza minden csomag elveszését.

adaptív dinamikus, a forgalomirányítás döntésiben figyelembe veszik a topológiában és a forgalomban történt változásokat. A frissítendő adatokat a (helyi, szomszédos, összes többi) routertől kapja meg.

Statikus routing A forgalomirányítási táblázatot a rendszeradminisztrátor tartja karban. Amit egyszer beírunk, az marad a cím. A világ felé egy, vagy csak néhány kapcsolódási pont van.

Dinamikus routing Ha több kapcsolódási pont van. Hogyan tartják karban a táblát? A forgalomirányítási táblázat(ok) valamilyen routing protocol segítségével kerülnek karbantartásra.

A dinamikus forgalomirányítás osztályozása:

Belső forgalomirányíási protokollok legfőbb alapelv a „legjobb útvonal” meghatározása. (IGP, RIP, OSPF)

külső forgalomirányítási protokollok Nem feltétlenül a legjobb útvonal megállapítása a cél, politikai alapú forgalomirányítás. (EGP, BGP)

17.2.1. A dinamikus routing áttekintése

Az irányító protokollok a forgalomirányítók között folytatnak adatcserét. Az irányító protokollok segítségével a forgalomirányítók megoszthatják az általuk ismert hálózatokról rendelkezésükre álló információkat a többi forgalomirányítóval, illetve közölhetik más forgalomirányítóktól való távolságukat. A forgalomirányítók a többi forgalomirányítótól kapott információk alapján saját irányítótáblát építenek fel és tartanak karban.

Példák irányító protokollokra:

- RIP (Routing Information Protocol)
- IGRP (Interior Gateway Routing Protocol)

- EIGRP (Enhanced Interior Gateway Routing Protocol)
- OSPF (Open Shortest Path First)

Az irányított protokollok a felhasználók adatait továbbítják. Az irányított protokollok elegendő hálózati rétegbeli információt biztosítanak ahhoz, hogy a csomagok a címzési séma alapján eljussanak a célállomáshoz.

Példák irányított protokollokra:

- IP (Internet Protocol)
- IPX (Internetwork Packet Exchange)

17.2.2. Autonóm rendszerek

Az autonóm rendszer egyazon felügyelet alá tartozó, közös irányítási stratégiát alkalmazó hálózatok együttese. A külvilág számára az autonóm rendszer egyetlen entitásnak látszik. Az autonóm rendszer üzemeltetését egy vagy több operátor is végezheti, ám a külvilág számára mindig egységes forgalomirányítási képet tükröz.

Az autonóm rendszerek számára az ARIN (American Registry of Internet Numbers) nevű szervezet, valamilyen szolgáltató vagy egy rendszergazda osztja ki az azonosítókat. Az autonómrendszer-azonosító egy 16 bites szám. Az irányító protokollok – például a Cisco IGRP – egy egyedi autonómrendszer-azonosító hozzárendelését követelik meg.

17.2.3. Az irányító protokollok és az autonóm rendszerek feladata

Az irányító protokollok feladata az irányítótábla létrehozása és karbantartása. Ez a táblázat a megismert hálózatokat és a hozzájuk rendelt portokat tartalmazza. A forgalomirányítók az irányító protokollokat a más forgalomirányítóktól kapott és a saját interfészeik konfigurációjából felismert információk kezelésére használják, kiegészítve velük a kézzel konfigurált útvonalakat.

Az irányító protokoll minden rendelkezésre álló útvonalat felmér, a legjobbakat az irányítótáblába írja, az érvénytelenné váltakat pedig törli. A forgalomirányító az irányítótáblában lévő információk alapján továbbítja az irányított protokollok csomagjait.

Az irányító algoritmus a dinamikus forgalomirányítás legfontosabb eleme. Amikor a hálózat topológiája a hálózat növekedése, átkonfigurálása vagy meghibásodása folytán megváltozik, a hálózatot leíró tudásbázisnak is követnie kell a változást. A tudásbázisnak pontos, konzisztens képet kell adnia az új topológiáról.

Amikor már az összekapcsolt hálózat valamennyi forgalomirányítója ugyanazt a megváltozott információbázist használja, akkor azt mondjuk, hogy a hálózati konvergencia megtörtént. Kívánatos, hogy a hálózat gyorsan konvergáljon, mert így csökken az az idő, amíg a forgalomirányítók a régi információk alapján helytelen forgalomirányítási döntéseket hoznak.

Az autonóm rendszereknek köszönhetően a világméretű összekapcsolt hálózat kisebb, könnyebben kezelhető hálózatokra oszlik. Minden autonóm rendszer saját szabálykészlettel és házirenddel rendelkezik, valamint egyedi autonómrendszer-azonosítója van, amely a világ összes más autonóm rendszerétől megkülönböztethetővé teszi.

17.2.4. Az irányító protokollok osztályai

A legtöbb irányító algoritmus besorolható a következő két alapvető osztály valamelyikébe:

- távolságvektor alapú
- kapcsolatállapot alapú

A távolságvektor alapú eljárás az összekapcsolt hálózatban minden összeköttetéshez egy irányt (vektort) és egy távolságot határoz meg.

A kapcsolatállapot alapú, más néven legrövidebb utat kereső módszer a teljes összekapcsolt hálózatról térképet készít.

17.2.5. A távolságvektor alapú forgalomirányító protokollok szolgáltatásai

A távolságvektor alapú irányító algoritmusokat használó forgalomirányítók periodikusan elküldenek egymásnak egy-egy másolatot az irányítótáblájukról. Ezek a rendszeres frissítések viszik át a topológia változásaira vonatkozó információkat a forgalomirányítók között. A távolságvektor alapú algoritmusokat Bellman-Ford algoritmusoknak is nevezik.

Minden forgalomirányító a közvetlen szomszédjaitól kap irányítótáblákat. A B forgalomirányító az A forgalomirányítótól kap adatokat. A B forgalomirányító a kapott adatokhoz valamilyen távolságadatot ad hozzá, például az ugrások számát, így növeli a távolságvektor nagyságát. A B forgalomirányító ezt a módosított irányítótáblát adja tovább szomszédjának, a C forgalomirányítónak. A szomszédos forgalomirányítók között lépésről lépésre ugyanez a folyamat játszódik le.

Az összegyűlt távolságértékek alapján az algoritmus egy adatbázist tart karban a hálózat topológiájáról. A távolságvektor alapú algoritmusokkal azonban nem alkotható teljes kép az összekapcsolt hálózatról, mivel minden forgalomirányító csak a szomszédos forgalomirányítókat látja.

A távolságvektor alapú forgalomirányítást használó forgalomirányítók először szomszédjaikat azonosítják. A közvetlenül csatlakoztatott hálózatok felé vezető interfészek távolsága nulla. A távolságvektor alapú felismerő folyamat előrehaladtával a forgalomirányítók a különféle célhálózatok felé vezető legjobb útvonalakat a szomszédjaiktól kapott adatok alapján ismerik fel. Az A forgalomirányító a B forgalomirányítótól kapott adatok alapján ismeri meg a többi hálózatot. Az irányítótábla minden más bejegyzése akkumulált távolságvektort tartalmaz, amely az adott hálózat adott irányban mért távolságát tükrözi.

Az irányítótábla frissítései a topológia változásait követően történnek meg. A hálózatfelderítő folyamathoz hasonlóan a topológiaváltozásra vonatkozó frissítések is lépésenként jutnak el az egyik forgalomirányítótól a másikig. A távolságvektor alapú irányító algoritmusokat futtató forgalomirányítók a teljes irányítótáblájukat elküldik valamennyi szomszédjuknak. Az irányítótáblák többek között az útvonalaknak az adott mérték szerinti teljes költségét és az egyes hálózatokhoz vezető útvonalak első forgalomirányítójának logikai címét tartalmazzák.

A távolságvektorokat az autópályák kereszteződéseiben látható feliratokhoz hasonlíthatnánk. Minden csomópontban nyíl mutatja a célállomások irányát, távolságukat pedig szám jelzi. Ha továbbhaladunk, akkor újabb nyíl mutatja a helyes irányt, de már kisebb távolság lesz kiírva. Amíg a távolság csökken, addig tudhatjuk, hogy a legjobb úton haladunk.

17.2.6. A kapcsolatállapot alapú irányító protokollok szolgáltatásai

A kapcsolatállapot alapú algoritmusokat Dijkstra-algoritmusoknak vagy legrövidebb utat kereső algoritmusoknak is nevezzük.
A kapcsolatállapot alapú irányító algoritmusok a topológiáról szerzett információból összetett adatbázist készítenek. A távolságvektor alapú algoritmusok a távoli hálózatokról nem rendelkeznek specifikus információkkal, és a távoli forgalomirányítókat sem ismerik. A kapcsolatállapot alapú algoritmusok ezzel szemben minden adattal rendelkeznek a távoli forgalomirányítókról és kapcsolataikról.

A kapcsolatállapot alapú forgalomirányítás a következő adatokat, eszközöket használja:

Kapcsolatállapot-hirdetések (link-state advertisement, LSA) A kapcsolatállapot-hirdetés egy forgalomirányítási adatokat tartalmazó kisméretű csomag. A forgalomirányítók továbbítják egymás között.

Topológiai adatbázis A topológiai adatbázis az LSA-kból kinyert információkat foglalja össze.

Legrövidebb utat kereső algoritmus A legrövidebb utat kereső (shortest path first, SPF) algoritmus az adatbázis adatait dolgozza fel. Eredményként az SPF-fát adja.

Irányítótáblák Az ismert útvonalak és interfészek listája.

A kapcsolatállapot alapú forgalomirányítás által alkalmazott hálózatfelderítési módszerek Az LSA-k kölcsönös elküldése a forgalomirányítók között a közvetlenül csatlakozó hálózatok adataival kezdődik, ugyanis ezekről a forgalomirányítók közvetlen adatokkal rendelkeznek. A többiekkel párhuzamosan az LSA-k felhasználásával az összes forgalomirányító létrehoz egy topológiai adatbázist.

Az SPF algoritmus a hálózatok elérhetőségét határozza meg. A forgalomirányító a logikai topológiát egy fába szervezi, amelynek saját maga a gyökere. A fa a kapcsolatállapot alapú forgalomirányítást használó összekapcsolt hálózat összes hálózatához minden lehetséges útvonalat tartalmaz. Ezután az algoritmus az útvonalakat hosszuk szerint sorba állítja úgy, hogy a legrövidebb útvonal kerüljön előre (innen származik az SPF, „shortest path first" elnevezés). A forgalomirányító a legjobb útvonalakat és a hozzájuk tartozó interfészeket az irányítótáblájába írja. Ezenfelül a hálózati topológia elemeiről és azok állapotáról további adatbázisokat tart karban.

Az a forgalomirányító, amely elsőként szerez tudomást valamilyen kapcsolat állapotának megváltozásáról, ezt az információt továbbítja, így a többi forgalomirányító megfelelő frissítést tud végezni. A folyamatba az is beletartozik, hogy általános irányítási információkat küldenek az összekapcsolt hálózat valamennyi forgalomirányítójának. A konvergencia megvalósítása érdekében minden forgalomirányító nyilvántartja szomszédjait, nevüket, interfészeik állapotát, illetve a hozzájuk vezető összeköttetés költségét. Minden forgalomirányító egy LSA-csomagot készít, amely az új szomszédokkal, az összeköttetések költségének megváltozásával és az érvénytelenné vált összeköttetésekkel kapcsolatos adatokat tartalmazza. Ezután elküldi az LSA-csomagot, amelyet az összes forgalomirányító megkaphat.

Amikor a forgalomirányító LSA-csomagot kap, az adatbázist a legújabb adatokkal frissíti, ezek alapján újraszámítja az összekapcsolt hálózat térképét, valamint az SPF algoritmussal meghatározza az egyes hálózatokhoz vezető legrövidebb útvonalakat. Valahányszor egy beérkező LSA-csomag változást okoz a kapcsolatállapotokat tároló adatbázisban, az SPF algoritmus újraszámolja a legjobb útvonalakat, és ennek megfelelően módosítja az irányítótáblát.

A kapcsolatállapot alapú irányítás problémái:

- Nagy processzorterhelés
- Komoly memóriakövetelmények
- A sávszélesség erőteljes igénybe vétele

A kapcsolatállapot alapú protokollokat futtató forgalomirányítóknál több memóriára és nagyobb processzorteljesítményre van szükség, mint a távolságvektor alapú protokollt használóknál. A forgalomirányítóknak elegendő memóriával kell rendelkezniük ahhoz, hogy tárolják a különféle adatbázisokból származó adatokat, a topológiafát és az irányítótáblát. A kezdeti kapcsolatállapot-csomagok elárasztással történő szétküldése jelentős sávszélességet köt le. A kezdeti felderítés folyamán a kapcsolatállapot alapú protokollokat használó valamennyi forgalomirányító az összes többi forgalomirányítónak LSA-csomagot küld. Ezek a csomagok elárasztják az összekapcsolt hálózatot, és emiatt átmenetileg csökken az irányított forgalom által felhasználói adatok továbbítására használható sávszélesség. A kezdeti elárasztás után a kapcsolatállapot alapú irányító protokollok általában csak minimális sávszélességet igényelnek, mert a forgalomirányítók a topológiaváltozásokat mutató LSA- csomagokat ritkán vagy eseményvezérelten küldik.

17.2.7. Autonóm rendszerek, az IGP és az EGP összehasonlítása

A belső irányító protokollokat több részből álló, de egyetlen szervezet felügyelete alatt lévő hálózatokhoz tervezték. A belső irányító protokollok esetében a legfontosabb elvárás, hogy megtalálják a legjobb útvonalat a hálózaton keresztül. Másként fogalmazva mondhatjuk, hogy a mérték jellege és használatának módja a belső irányító protokollok legfontosabb jellemzője.

A külső irányító protokollokat különálló, eltérő szervezetek ellenőrzése alatt lévő hálózatok között használják. Általában internetszolgáltatók között, illetve internetszolgáltató–ügyfél kapcsolatok felett használják őket. Adott cég például saját és internetszolgáltatójának forgalomirányítója között dönthet a BGP – egy külső irányító protokoll – használata mellett. Az IP alapú külső átjáróprotokollok esetében a forgalomirányítás megkezdése előtt a következő három adathalmazra van szükség:

- A szomszédos forgalomirányítók listája, ezekkel folyik az irányítási információk cseréje
- A közvetlenül elérhető hálózatok listája, melyeket hirdetni kell
- A helyi forgalomirányító autonómrendszer-azonosítója

A külső irányító protokolloknak el kell különíteniük az autonóm rendszereket. Ne feledjük, az autonóm rendszerek más és más rendszergazdák felügyelete alatt állnak. A hálózatoknak biztosítaniuk kell valamilyen protokoll támogatását, amely felett ezek a különböző rendszerek kommunikálni tudnak egymással.

Az autonóm rendszerek rendelkeznek egy azonosítóval, amelyet az ARIN (American Registry of Internet Numbers) szervezet vagy egy szolgáltató oszt ki nekik. Az autonómrendszer- azonosító egy 16 bites szám. Az irányító protokollok – például a Cisco IGRP és EIGRP – egy egyedi autonómrendszer-azonosító hozzárendelését követelik meg.