

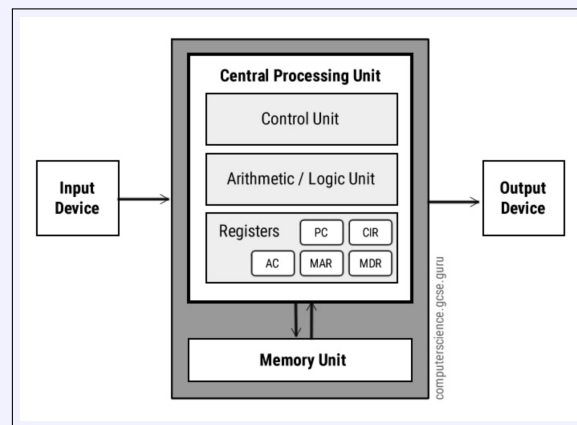
Informatika és Programozás Alapjai szóbeli vizsgatételek

Kun László Ákos

2022/23/2

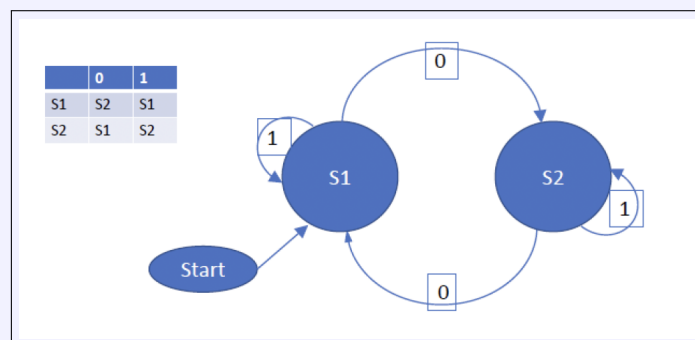
1. Ismertesse a Neumann-elveket!

- Teljesen elektronikus működés
- Bináris számrendszer használata (bit: 0/1, qbit 10/01)
- Szekvenciális művelet végrehajtás
- Adatok és programok a belső memóriában
- Univerzális felhasználás
- Öt funkcionális egység: aritmetikai egység, központi vezérlőegység, memóriák, bemeneti és kimeneti egységek.



2. Ismertesse a determinisztikus véges automata megadását és működését!

- M automata, input: 0,1-ből álló tetszőleges hosszúságú string. Output: eddig páros számú 0 volt az inputban?
- $M = (S, \Sigma, T, s, A)$, ahol
- $\Sigma = 0, 1$ input
- $S = S1, S2$, állapot
- $s = S1$, start
- $A = S1$, kimenet (S1-nél igaz)
- T állapotátmenet-táblázat



3. Ismertesse a Turing-gép felépítését és működését!

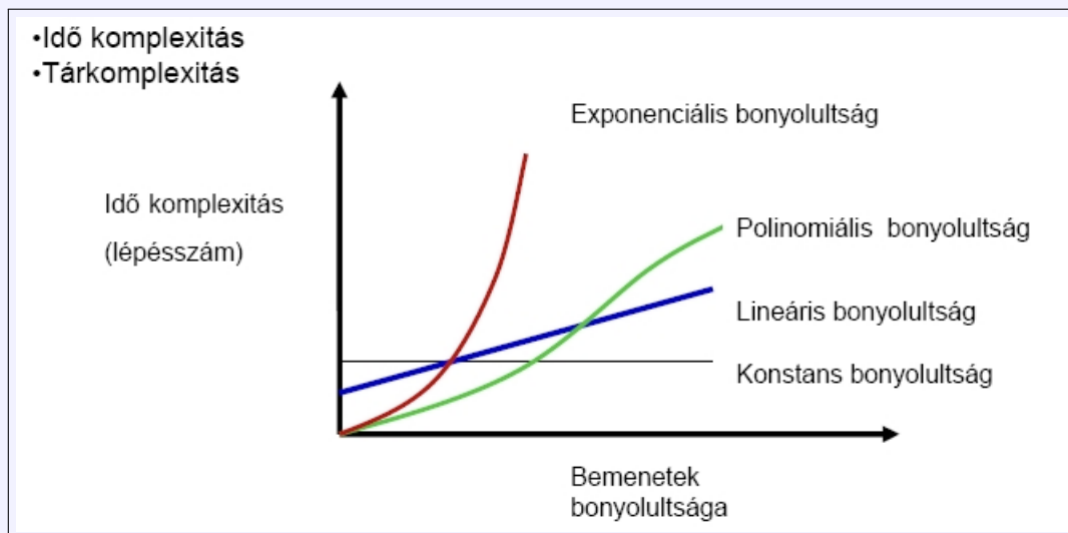
- Végtelen, cellákra osztott szalag. A cellában lehet szimbólum, vagy üres. Az adatok, a műveletek és eredmények cellái véges számúak, ezentúl a szalag üres.
- **Író/olvasó fej:** egyszerre egy cellával foglalkozik. A cellát írhatja, olvashatja, és törölheti. A szalagon jobbra/balra is lépkedhet, tartalom változás nélkül.
- **Vezérlőegység:** állapotai be vannak számozva, és végesek (véges állapotú automata). A működést helyettesítési táblázat adja meg (állapot+művelet+adat-> új állapot, eredmény, fejmozgás).
- **Matematikailag:** 5-10 elemből álló szabály halmaz
- **Informatikailag:** szalag = memória, vezérlőegység = CPU, fej = busz
- CT1-nek megfelelően rekurzióra is alkalmasnak kell lennie: veremtár(stack).

4. Ismertesse az algoritmus köznyelvi és matematikai és Turing-géppel megfogalmazott definícióját!

- Emberi nyelven megfogalmazott feladat, cselekvéssorozat. (köznyelv)
- Az algoritmusra nem létezik formális matematikai definíció.
- Algoritmus: a megoldási eljárást akkor tekintjük algoritmusnak, ha bármilyen bemenet esetén véges számú lépés után eredményt kapunk (a Turing gép megáll).

5. Ismertesse az algoritmusok komplexitásának meghatározását! Mondjon példát gyakori komplexitás típusokra!

- **Exponenciális/faktoriális komplexitás:** Az algoritmus futási ideje exponenciálisan növekszik az input méretével. Pl: faktoriális számítás, ahol az algoritmus futási ideje N faktoriálisával arányos, tehát $O(N!)$
- **Polinomiális komplexitás:** Az algoritmus futási ideje polinomiálisan növekszik az input méretével. Példa erre a buborékrendezés algoritmus, ahol az algoritmus futási ideje N négyzetével arányos, tehát $O(N^2)$
- **Lineáris komplexitás:** Az algoritmus futási ideje lineárisan növekszik az input méretével. Példa erre a lineáris keresés algoritmus, ahol az algoritmus futási ideje lineárisan arányos az input méretével, tehát $O(N)$
- **Logaritmikus komplexitás:** Az algoritmus futási ideje logaritmikusan növekszik az input méretével. Példa erre a bináris keresés algoritmus, ahol az algoritmus futási ideje logaritmikusan arányos az input méretével, tehát $O(\log N)$
- **Konstans komplexitás:** Az algoritmus futási ideje állandó marad az input méretétől függetlenül. komplexitása: $O(1)$



6. Ismertesse a 2 bemenetű logikai függvényeket! Ezek közül néhányat nevezzen is el!

- 2 (független) változó esetén:

A	B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

$f_1 : AND$; $f_7 : OR$; $f_6 : XOR$; $f_8 : NOR$; $f_{14} : NAND$; $f_0 : 0$; $f_{15} : 1$;

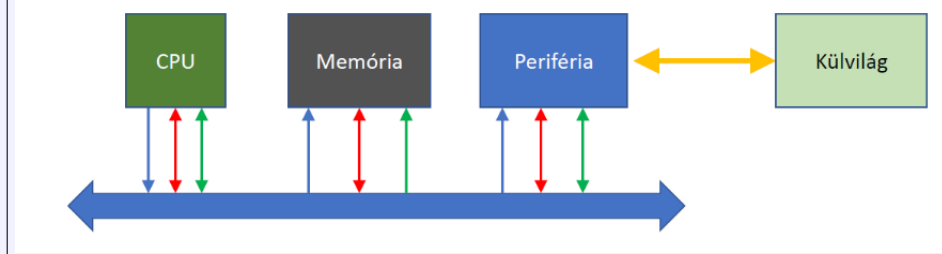
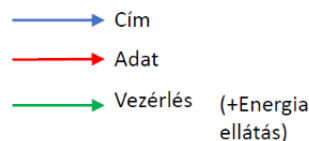
•

Ismertesse (indoklással) az n bemenetű logikai függvények számát!

- n bemenet esetén: n darab bemenő változó, mindegyiknek 2 értéke van, vagyis $2n$ bemeneti kombinációhoz 2 elemet rendelünk, azaz: 2^{2^n} darab függvény lesz (2 változó esetén 16 db függvény lesz).

7. Ismertesse a számítógépes rendszer logikai felépítését!

- Busz: adatok, címek, vezérlő jelek továbbítása
- CPU: műveletvégzés
- Memóriák: adattárolás
- Perifériák: kapcsolat a külvilággal



- **Busz:** párhuzamos (manapság soros) jel köteg.
- **Busz szélességek:** egyszerre átvihető adat mérete.
- **Adatbusz:** 8 bit (C64, ZX spectrum), 16 bit (IBM PC, pic24), 32 bit (IBM 386-486-Pentium), 64 bit PC manapság. Szinte mindig az akkumulátor regisztermérete.
- **Címbusz:** memória maxméretét adja meg. 8 bites CPU-nál gyakran 16 bites (65536 cella), IBM PC-nél 20 bites, i9: 128 GB címezhető meg.
- Ha a memória-vezérlő is a CPU része, a címbusz csak belül jelenik meg.

8. Ismertesse az 1 bites teljes összeadó példáján keresztül a kombinációs logikai hálózatok elvét!

A	B	C_{in}	C_{out}	Y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Bemenetek: A , B , C_{in} ; Kimenetek: Y , C_{out}
- $Y = A \text{ xor } B \text{ xor } C$
- $C_{out} = (A \text{ and } B) \text{ or } (A \text{ and } C_{in}) \text{ or } (B \text{ and } C_{in})$
- A kombinációs logikai hálózatok kimenetei csak a bementektől függnnek. Minden kimenetet egy függvény ír le. ($F_1(X_1, X_2, \dots, X_n)$)

9. Ismertesse számpéldán keresztül a negatív egész számok kettes komplementes ábrázolását!

- To be continued...

10. Ismertesse az S-R tároló példáján keresztül a szekvenciális logikai hálózatok elvét!

- To be continued...

11. Ismertesse a D tároló felépítését és működését!

- To be continued...

12. Melyik tároló tartalmaz a számítógépben D tárolót?

- RAM, SSD, HDD

13. Ismertesse a tárolók hierarchia szintjeit!

- **Regiszterek:** A regiszterek a CPU belső tárolói, amelyek nagyon gyors hozzáférést biztosítanak az adatokhoz. A regiszterek közvetlenül a CPU-ban találhatóak, és az utasítások végrehajtásához és az adatok köztes tárolásához használhatóak.
- **Cache:** A cache olyan kis méretű, gyors és közel a processzorhoz elhelyezkedő tároló, amely az aktuálisan leggyakrabban használt adatokat és utasításokat tárolja. Célja, hogy csökkentse a memória-hozzáférési időt és javítsa a rendszer teljesítményét.
- **Operatív tár (RAM):** Az operatív tár az adatok és utasítások átmeneti tárolására szolgál a számítógépben. Ez a tár rendelkezik nagyobb kapacitással, mint a regiszterek vagy a cache, de lassabb hozzáférést biztosít. Az adatok ideiglenesen tárolódnak itt a futó programok számára.
- **Mágneses/SSD direkt elérésű háttértár:** Ez a tároló típus olyan háttértár, amely mágneses lemezeket (merevlemezeket) vagy szilárdtest meghajtókat (SSD) használ. Ezek a tárolók nagyobb kapacitással rendelkeznek, mint az operatív tár, és lehetővé teszik az adatok hosszú távú tárolását.
- **Szekvenciális elérésű háttértár:** Ez a tároló típus olyan adathordozókat használ, mint például a szalagok. A szekvenciális tárolóknál az adatok egymás után találhatók, és az adatokhoz való hozzáférés sorrendben történik. A szekvenciális elérés lassabb, mivel az adatokat folyamatosan kell olvasni vagy írni a szalag vagy más hordozó mentén.

14. Ismertesse a CPU részeit és működését egy utasítás végrehajtási folyamatában!

- To be continued...

15. Ismertesse a periféria csatolási módszereket, módszerenként kitérve az adott módszer előnyére, és hátrányára! Part 1.

- **Polling**

- A periféria állapotát le kell kezelni: pl. észre kell venni, ha leütöttek egy billentyűt.
- Ennek legegyszerűbb módja az ún. pollozás/polling: a program bizonyos időközönként kiolvassa a perifériát, ha nincs változás, foglalkozik a többi programmal, ha változás van, lekezeli a változást, majd foglalkozik a többi programmal.
- A periféria változása a főprogramhoz képest aszinkron bekövetkezésű (vagyis nem tudjuk, hogy mikor kell beavatkozni).
- A módszer nem hatékony: túl sok utasítás megy el felesleges lekérdezéssel, a főprogramból gyakran kell lekérdezést kérdezni,

- **IRQ**

1. Egészítsük ki a perifériát egy plusz kimeneti lábbal: ezen küldjön aktív értéket, ha állapotváltozás miatt „figyelemre” van szüksége.
2. Ezt a lábat nevezzük el „interruptrequest”/megszakítás kérő lábnak.
3. A lábat kössük be a CPU-ba! (a gyakorlati megvalósítás bonyolultabb: pl. azonosítani kell, hogy melyik periféria volt a megszakítás kérő).
4. A megszakítási kérelemre a CPU futtasson le egy IRQ kezelő függvényt. A függvény címe a memóriában fix helyen, IVT-ben van tárolva. A függvényt úgy kell megírni, hogy ne rontsa el a regiszterek értékét, mert az IRQ bármikor jöhet.

Felhasználás:

- Hardver hibák (memória hiba, busz hiba) lekezelése
- Szoftver hibák (0-val osztás, védett memóriaterületre hivatkozás, verem alul/felülcsordulás) lekezelése
- Idő kezelése (1s-onként irqaktualizálja a rendszeridőt)
- Szoftveresen is hívható, intel-nél utasítás van rá: operációs rendszer belépési pontjai.
- Maszkolhatómegszakítás: utasítással letiltható, ekkor nem fut le az irqsubrutin
- Nem maszkolhatómegszakítás (NMI) szoftverből nem tiltható le.
- A megszakítások prioritási sorba állíthatók: a nagyobb prioritású irqmegszakíthatja a kisebb prioritású irqkiszolgálását, fordítva nem.

15. Ismertesse a periféria csatolási módszereket, módszerenként kitérve az adott módszer előnyére, és hátrányára! Part 2.

- **DMA nélkül**

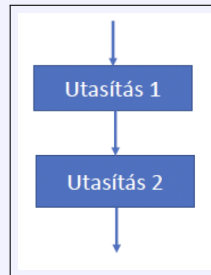
- Nagymennyiségű adat átvitele a háttértárról (periféria)
- CPU megcímzi a perifériát (megadja, hogy hol található az adat) a buszon keresztül
- Periféria szolgáltatja az adatot (idő!) Az adat a CPU regiszterébe kerül (buszon keresztül)
- A CPU a memóriába teszi az adatot a regiszterből.
- Következő bájtt a memória következő címére a háttértárról
- CPU mással nem tud közben foglalkozni.

- **DMA**

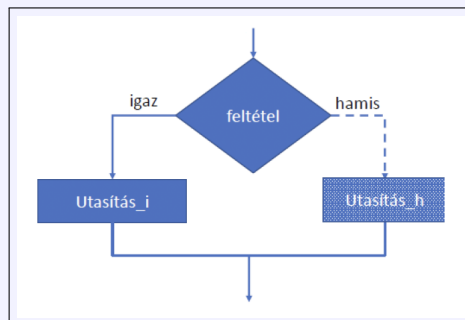
- A művelet közben nincs nagy számítás igény: buszt kell vezérelni, címet növelni 1-gyel.
 - Erre a műveletre készülhet egy speciális áramkör: DMA vezérlő.
1. A CPU megmondja a DMA vezérlőnek, hogy honnan, hova, hány bájtot.
 2. A DMA vezérlő elkéri a buszt a CPU-tól, ha annak éppen nincs rá szüksége (plbelső művelet zajlik). A CPU lekapcsolja a buszmeghajtókat.
 3. A DMA vezérlő átvisz 1 bájtot a perifériából, a saját adatregiszterébe(olvasás esetén).
 4. A DMA vezérlő által elkért buszon átvisz 1 bájtot a memóriába. Írás esetén a két lépés fordított sorrendben.
 5. A DMA vezérlő visszaadja a buszt
 6. DMA vezérlő újra elkéri a buszt.
 7. Az utolsó bájtt átvitele után nem kéri el a buszt, egy IRQ-n keresztül jelzi, hogy az átvitel kész.
 8. A busz elkérés neve: busz arbitráció.
 9. Dinamikus memória frissítésére is kiváló!

16. Ismertesse az alapvető algoritmus-elemeket a Böhm-Jacopini tétel alapján!

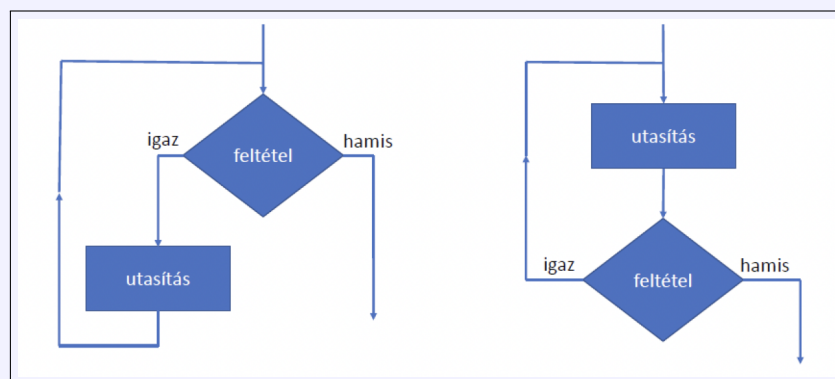
- Böhm-Jacopini tétele: Minden algoritmus leírható 3 logikai struktúrával:
- Rákövetkezés (konkatenáció):
 - Az utasítások egymás után hajtódnak végre
 - Külön kulcsszó nem tartozik hozzá (alapesetben is egymás után hajtjuk végre az utasításokat, a címek szerint növekvő sorrendben.)
 - C-ben az egymás alá írt utasítások egymás utáni címekre kerülnek.



- Választás (alternáció):
 - Egy logikai feltétel függvényében kerül végrehajtásra az utasítás
 - Feltétel hamis értéke esetén is megadható másik utasítás
 - C-ben: `if(condition) commandtrue; commandfalse;`



- Ciklus (iteráció):
 - Egy feltételtől függően ismételjük az utasításokat
 - A feltétel és az utasítás egymáshoz képesti elhelyezkedése miatt előltesztelt és hátúltesztelt



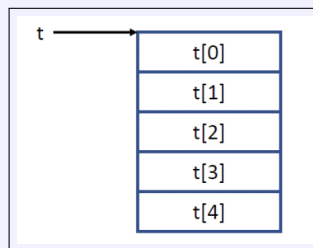
17. Ismertesse az alapvető adatszerkezet típusokat és ezek műveleteit!

- **Adatszerkezet:** egyszerű vagy összetett alapadatok rendszerének matematikai, logikai modellje
- **Típusok:**
 - **Tömb:** (lineáris, egy vagy több dimenziós)
 - **Rekord:** összetartozó adatok egy példányhoz
 - **Kapcsolt (láncolt) lista:** a kapcsolati információt is tároljuk
 - **Gráf** két kapcsolódó adathalmaz: csomópontok és élek
 - **Fa:** Hurok nélküli gráf. Általános és bináris fa
 - **Verem:** LIFO (Last In First Out)
 - **Sor:** FIFO (First In First Out)
- **Műveletek**
 - **Bejárás:** az elemek elérése.
 - **Keresés:** adott feltételnek megfelelő elemek kiválasztása.
 - **Beszúrás:** új adat beillesztése az adatszerkezetbe
 - **Törlés:** adat eltávolítása az adatszerkezetből
 - **Rendezés:** adatok logikai sorrendbe állítása
 - **Összeválogatás:** különböző rendezett elemhalmazotból új elemhalmaz kialakítása.

18. Ismertesse a lineáris és a többdimenziós tömbök memóriamodelljét!

Lineáris tömbök:

- Azonos típusú (méretű) adatelemek
- Az elemekre indexsegítségével hivatkozunk
- Egymást követő memóriacímeken tároljuk, folytonos területen
- Egy elemhez bejárás nélkül férünk hozzá, konstans idővel
- Beszúrás/törlés nem hatékony!
- i -edik elem memória címe = tömb első elemének címe + i * adatmérete. A pointer az adat méretét tudja! Ezért $*(t+i)$

**Többsdimenziós tömbök:**

- A memória lineáris. Több dimenziós tömböt kétféleképpen tárolhatunk:
- A tömb elemei pointerok, amelyek tömbök (C és C++). Duplaíndirekció: `int **t`; hivatkozás: `t[i][j]`
- A programozó vagy a fordító leképezi az indexeket a lineáris memóriába. Pl. `C# t[i, j]`
- $[i, j]$ -edik elem címe egy $m \times n$ -estömbben: tömb kezdőcíme + $(n \cdot i + j) \cdot \text{adatmérete}$

	Column 1	Column 2	Column 3	Column 4
Row 1	<code>x[0][0]</code>	<code>x[0][1]</code>	<code>x[0][2]</code>	<code>x[0][3]</code>
Row 2	<code>x[1][0]</code>	<code>x[1][1]</code>	<code>x[1][2]</code>	<code>x[1][3]</code>
Row 3	<code>x[2][0]</code>	<code>x[2][1]</code>	<code>x[2][2]</code>	<code>x[2][3]</code>

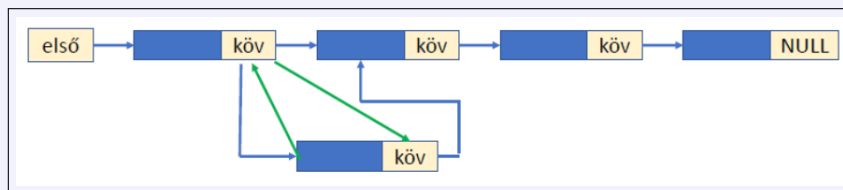
19. Ismertesse a kapcsolt lista adatszerkezet felépítését és gyakori műveleteit!

A kapcsolt lista adatszerkezet felépítése:

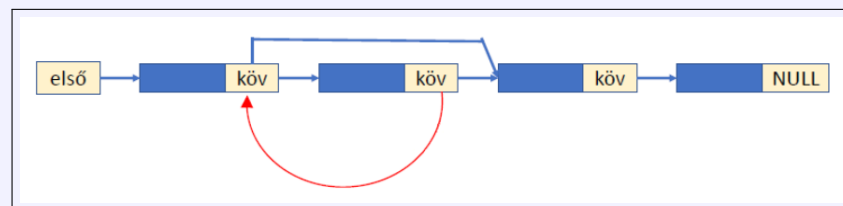
- Csomópontokból áll, amelyek tartalmazzák az adatot és egy mutatót a következő csomópontra.
- Az utolsó csomópont mutatója NULL, jelezve a lista végét.

Gyakori műveletek a kapcsolt listán:

- Elem hozzáadása a lista elejéhez (prepend).
- Elem hozzáadása a lista végéhez (append).
- Elem beszúrása adott helyre.



- Elem törlése a listából.



- Adott elem keresése a listában.
- Lista méretének lekérdezése.
- Lista kiíratása.

Előnyök:

- Dinamikus méret, könnyű bővítés és csökkentés.
- Hatékony beszúrás és törlés az adott helyen.
- Memóriatakarékos, mert csak az aktuális elemekhez szükséges memóriát foglalja le.

Hátrányok:

- Random hozzáférés lassú, mert a kereséshez végig kell menni a listán.
- Több memóriaterületet igényel, mert minden csomóponthoz szükséges egy mutató.
- Összekapcsoltság elvesztése, ha valamelyik mutató hibás lesz.

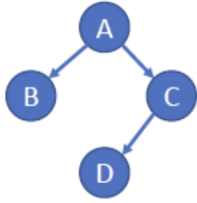
20. Ismertesse a gráf definícióját valamint az alábbi gráfelméleti fogalmakat: út, kör, összefüggő, teljes, címkézett, súlyozott és irányított!

- **Gráf definíciója:** A gráf egy matematikai struktúra, amely csomópontokból (vagy csúcsokból) és azokat összekötő élekből áll. A gráfot grafikusán ábrázoljuk, ahol a csomópontokat pontokkal, az éleket pedig vonalakkal jelöljük.
- **Út:** Az út olyan sorozat vagy láncolat, amelyben egymás után következnek a gráf csomópontjai úgy, hogy az élek az egymást követő csomópontokat összekötik.
- **Kör:** A kör egy olyan út, amely a kezdőpontjába visszatér, vagyis a kör kezdő- és végpontja azonos csomópont.
- **Összefüggő:** Egy gráf akkor összefüggő, ha bármely két csomópontja között van legalább egy út.
- **Teljes:** Egy gráf teljes, ha minden csomópontja között van él, azaz bármely két csomópont között van egy-egy él.
- **Címkézett:** Egy címkézett gráfban a csomópontokhoz vagy élekhez hozzárendelhetünk címkéket vagy jellemzőket, amelyek információt hordoznak a gráf elemeiről.
- **Súlyozott:** Egy súlyozott gráfban az élekhez hozzárendelünk súlyokat vagy értékeket, amelyek jelzik az élek közötti kapcsolat erősségét vagy távolságát.
- **Irányított:** Egy irányított gráfban az élek egyirányúak, tehát a csomópontok közötti kapcsolatok egyirányúak lehetnek.

21. Ismertesse a gráfok szomszédsági mátrixát!

- **Szomszédsági mátrix:** $a_{ij} = 1$, ha i -ből j -be halad él, egyébként $a_{ij} = 0$

M	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	1
D	0	0	0	0

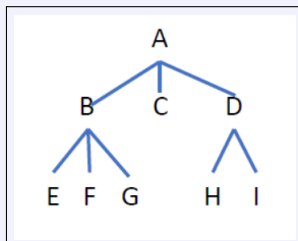


- Ha M a G gráf szomszédsági mátrixa, akkor M_{ij}^k -edik eleme az i -ből a j -be vezető k hosszú utak számát adja

M^2	A	B	C	D
A	0	0	0	1
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

22. Ismertesse az általános fa felépítését és tárolását számítógépen!

- Elemek véges halmaza (T), amely
 - Tartalmaz egy kitüntetett R gyökérelemet
 - A többi elem nem nulla diszjunkt részfája T-nek

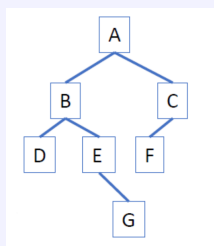


- Tárolás
 - Info(k) - az elem adatai
 - Gyermek(ek) - az első gyerek index
 - Testvér - az első testvér indexe

index	1	2	3	4	5	6	7	8	9
info	A	B	C	D	E	F	G	H	I
gyermek	2	5	0	8	0	0	0	0	0
testvér	0	3	4	0	6	7	0	9	0

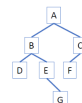
23. Ismertesse a bináris fák szerkezetét és tárolási lehetőségeit számítógépen!

- Elemek véges halmaza, amely, vagy üres, vagy egyetlen T elemhez (gyökér) kapcsolt két diszjunkt T1 és T2 részfa alkotja



- **Gyökérellem:** A
- **Baloldali részfa:** $L(A)=(B,D,E,G)$
- **Jobboldalirészfa:** $R(A)=(C,F)$
- **Szukcesszor:** leszármazott. Egy elemnek 0,1,2 lehet.
- Bal és jobb oldali szukcesszor
- B bal oldali szukcesszora A-nak, H jobb oldali szukcesszora E-nek.
- **0 szukcesszor:** zárócsomópont, levél
- **Összekötő vonalak:** élek
- **Utolsó él:** ág
- **Szintszám:** gyökér 0, leszármazott=szülő+1
- **Generáció:** azonos szintszámú elemek
- **Teljes:** az utolsó szintet kivéve minden elemnek 2 leszármazottja van
- **Kiterjesztett:** minden csomópontnak 0 vagy 2 leszármazottja van

• 3 tömböt használunk, egyet az adatnak, egyet a bal oldali, egyet a jobb oldali leszármazott indexének



index	1	2	3	4	5	6	7
adat	A	B	C	D	E	F	G
indexL	2	4	6	0	0	0	0
indexR	3	5	0	0	7	0	0

• Gyökér indexe 1

• K indexű elem bal oldali leszármazottjának indexe $2 \cdot k$

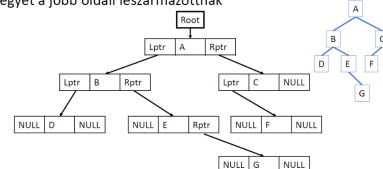
• Jobb oldali leszármazott indexe $2 \cdot k + 1$

• Ha nincs leszármazott, az ottani adat NULL.



1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	NULL	NULL	NULL	NULL	G	NULL	NULL	NULL

A tároló struktúra tartalmaz 2 pintert is, egyet a bal oldali, egyet a jobb oldali leszármazottjának



24. Ismertesse a veremtár működését!

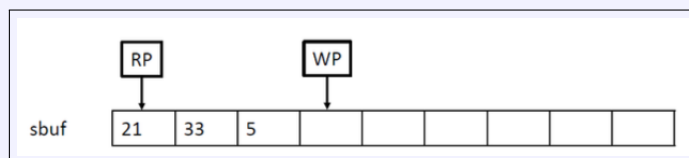
- **LIFO:** az utoljára berakott elem jön ki először (Last In First Out)
- **Push:** elemet a verembe rak
- **Pop:** elemet leemel a verem tetejéről
- **SP:** stackpointer, a verem tetejét mutatja

C nyelvű programok futása közben hol van szerepe a veremtárnak?

- **Alkalmazások:**
 - függvényhívás
 - rekurzió
 - böngésző „vissza”
 - szövegszerkesztő „undo” gombja.

25. Ismertesse a sor adatszerkezet és a ciklikus buffer működését!

- **FIFO:** First In, First Out
- Sor tárolásához 2 mutató szükséges: beírási mutató (WP), kiolvasási mutató (RP). A kiolvasási nem előzheti meg a beírást. A mutató index is lehet a tömbelemre.
- Ha $WP == RP$, nincs új adat a sorban. Beírás és kiolvasás előtt vagy után mutatót növelni. Az ábra az „után” szituációt mutatja.
- Billentyűzet puffer, windowsesemények queue-ja.



Ciklikus Buffer:

- Egy speciális **queue**: a tárolásra használt tömb elfogyása után újra kezdődik az elejétől.
- Tömb neve legyen sbuf, a két kétindex WP és RP
- Új adat érkezése: $sbuf[WP++] = adatbe$; beírás
- Ha $WP == tömbméret$, legyen $WP = 0$;
- $WP \neq RP$ esetben kiolvasás következik
- Kiolvasás: $adatki = sbuf[RP++]$;
- Ha $RP == tömbméret$, legyen $RP = 0$;
- Ha nem olvassuk ki, akkor a régi értékek törlődnek.

26. Ismertesse számpéldával a gyorsrendezés (quick sort) algoritmust! Adja meg az algoritmus átlagos és legrosszabb komplexitását!

- Legyen S halmaz a rendezendő elemek halmaza
- Válasszunk egy t támpont elemet.
- S elemeit t kivételével két diszjunkthalmazba (S_1 és S_2) soroljuk:
- $S_1 = \{x \in S - t \mid x \leq t\}$
- $S_2 = \{x \in S - t \mid x > t\}$
- S_1 -re és S_2 -re rekurzívan meghívjuk a gyorsrendezést.
- Eredmény: $quicksort(S_1) + t + quicksort(S_2)$
- Amikor az összes halmaz 1 elemet tartalmaz, megállunk.
- Átlagos komplexitás: $O(n * \log(n))$. Legrosszabb eset $O(n^2)$.

Példa a gyorsrendező algoritmsra a C-nyelvben:

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;}
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}
```

27. Ismertesse az egyed-kapcsolat modellt és alábbi fogalmait: egyedtípus, kapcsolat- típus, tulajdonság!

- **Egyedtípus (Entity type):** Az egyedek osztályát vagy kategóriáját jelöli. Az egyedtípus meghatározza, hogy az egyedek milyen tulajdonságokkal rendelkeznek és milyen kapcsolatokat tarthatnak fenn más egyedekkel. Például lehet egy "Diák" egyedtípus, amelynek tulajdonságai lehetnek a neve, életkora, osztálya stb.
- **Kapcsolat-típus (Relationship type):** Az egyedek közötti kapcsolatot jelöli. A kapcsolat-típus meghatározza, hogy az egyedek milyen módon kapcsolódnak egymáshoz. Például lehet egy "Tanul" kapcsolat-típus, amely egy diák és egy tantárgy közötti kapcsolatot reprezentál.
- **Tulajdonság (Attribute):** Az egyedek jellemzőit vagy attribútumait jelöli. A tulajdonságok konkrét információkat tárolnak az egyedekről. Például a "Diák" egyedtípusnál lehetnek olyan tulajdonságok, mint a név, életkor, osztály stb. A tulajdonságok meghatározzák az egyedek jellemzőit és értékeit.

28. Ismertesse a relációs adatbázisok felépítését és alábbi fogalmait: foksám, kardinalitás, attribútum, vetítés, kiválasztás, kulcs, szuperkulcs, elsődleges kulcs, külső kulcs.

- A matematikában reláción n darab halmaz **direkt szorzatának részhalmazát** értjük.
- A relációt **névvel** azonosítjuk.
- A relációban minden **sor különböző**.
- Létezik a szorzatot alkotó halmazok olyan halmaza, ami a reláció bármely elemét egyértelműen azonosítja (**kulcs**)
- A szorzatot alkotó halmazok (értelmezési tartományok) száma a **reláció fokszáma**
- A reláció **elemeinek száma** a reláció **kardinalitása**
- Az egyes elemekben a tényezők konkrét értéke az **attribútum**
- **Csonkító műveletek:**
 - **Vetítés (projection):** tényezők (értelmezési tartományok) kiemelése
 - **Kiválasztás (select):** elemek kiválasztása
- **Szuperkulcs:** a sorokat megkülönböztető oszlophalmaz
- **Kulcs:** minimális elemszámú szuperkulcs
- **Elsődleges kulcs (Primary Key):** a megkülönböztetésre választott kulcs
- **Külső kulcs (Foreign Key):** 1:N és M:N kapcsolat leírása esetén a másik tábla elsődleges kulcsa

29. Tervezzon egy relációs adatbázis-szerkezetet, amely M:N típusú kapcsolatot tud tárolni!

Példa, mely tárolja egy bolt termékeinek és vásárlóinak kapcsolatát.

- 1. Tábla: Products
 - *product_{id}* (egyedi azonosító)
 - *product_{name}*
 - *product_{description}*
 - egyéb tulajdonságok
- 2. Tábla: Buyers
 - *buyer_{id}* (egyedi azonosító)
 - *buyer_{name}*
 - *buyer_{address}*
 - egyéb adatok
- 3. Kapcsolótábla: Purchases
 - *purchase_{id}* (egyedi azonosító)
 - *buyer_{id}* (külső kulcs a Vásárlók táblában)
 - *product_{id}* (külső kulcs a Termékek táblában)
 - egyéb adatok (pl. vásárlás dátuma, mennyiség, ár stb.)

A Purchases tábla a kapcsolatot tárolja a vásárlók és a termékek között. Minden sor egy adott vásárlást reprezentál, ahol a *buyer_{id}* mező a kapcsolódó vásárlót azonosítja, a *product_{id}* mező pedig a kapcsolódó terméket azonosítja. Így a Purchases tábla segítségével nyomon követhetjük, hogy melyik vásárló melyik terméket vásárolta.

30. Ismertesse a kernel feladatait az operációs rendszerben!

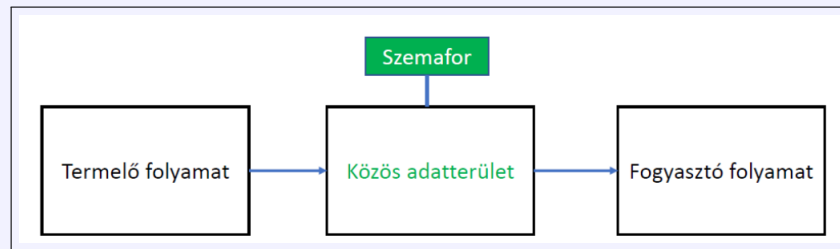
- Folyamat kezelés (Process management)
- Memória kezelés (allokáció, felszabadítás, relokáció)
- Háttértár kezelés
- I/O rendszer kezelés
- Fájl kezelés
- Védelmi rendszer
- Hálózat elérés támogatása

31. Ismertesse a folyamatok típusait a többi folyamathoz viszonyulás alapján!

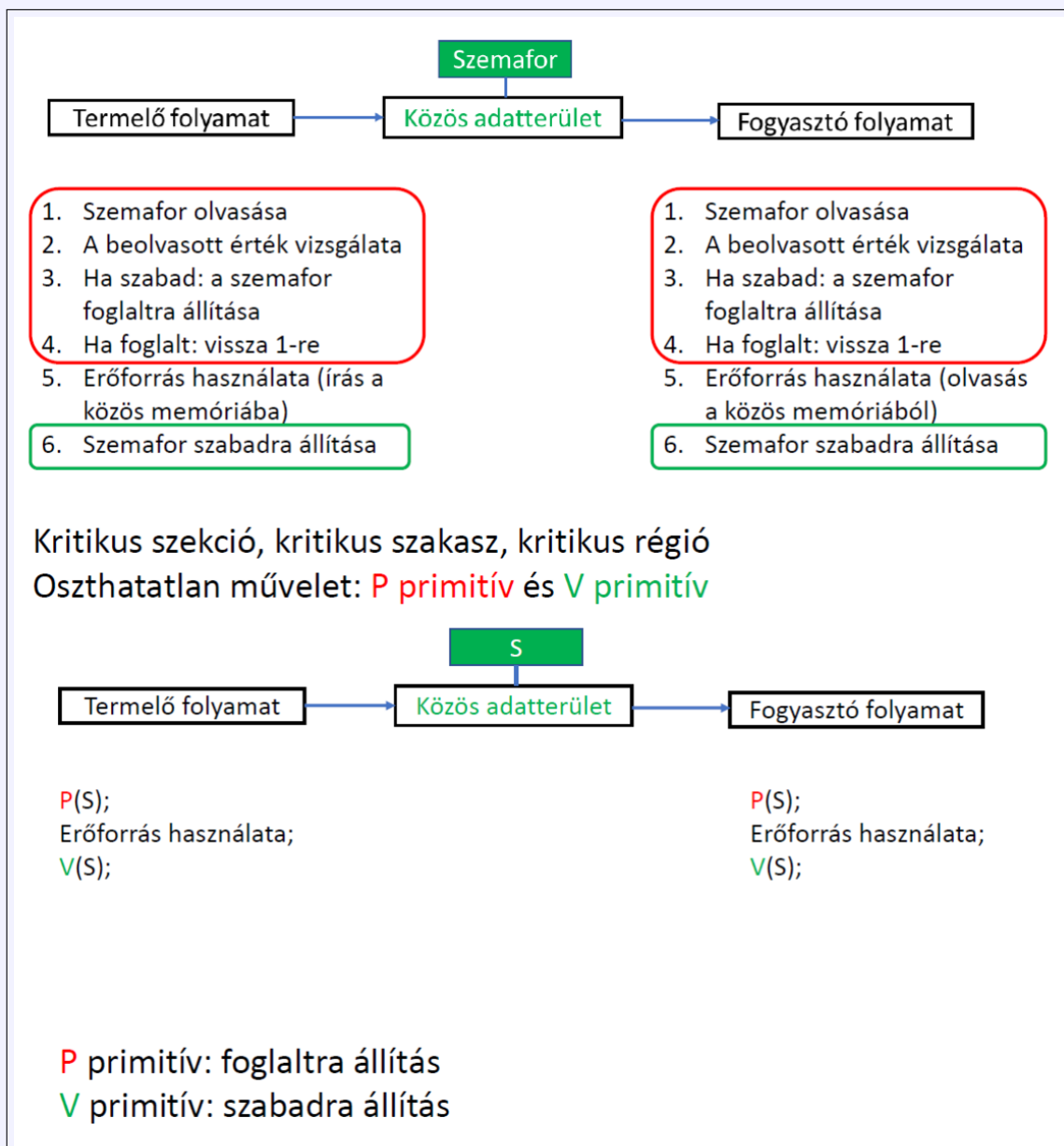
- **Független folyamatok:** egymás működését semmilyen módon nem befolyásolják.
- **Versengő folyamatok:** nem ismerik egymást, de közös erőforrásokon kell osztozniuk.
- **Együttműködő folyamatok:** ismerik egymást, együtt dolgoznak egy feladat megoldásán, információt cserélnek.

32. Ismertesse a folyamatok kommunikációját bináris szemafor segítségével!

- **Vezérlés:** szemafor segítségével.

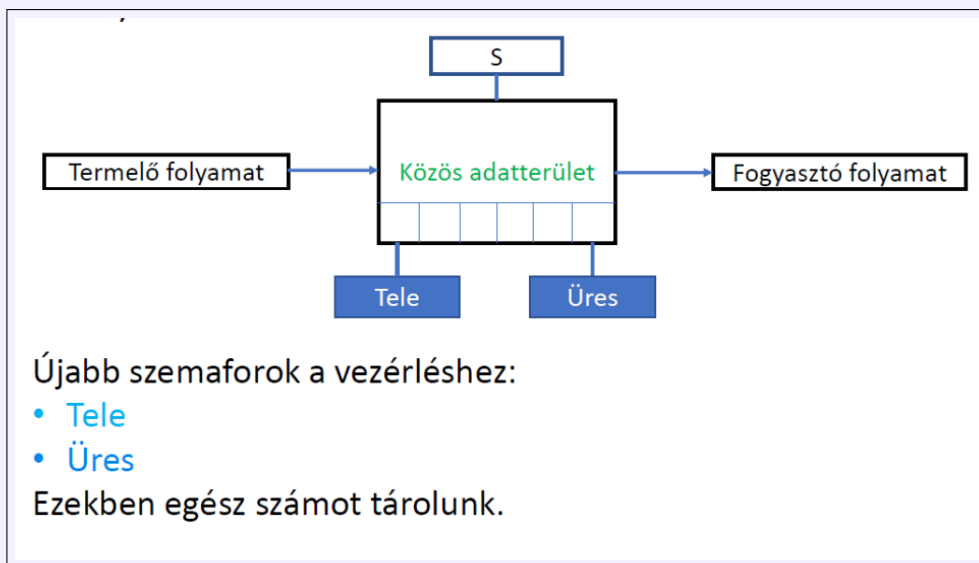


- Mielőtt a folyamat használni kezdené a közös erőforrást, ellenőriznie kell, hogy az szabad-e. (ezt az adott közös erőforráshoz rendelt bináris szemafor jelzi.)
- Csak akkor kezdheti el használni, ha a szemafor szabadot jelzett, ellenkező esetben várakoznia kell.

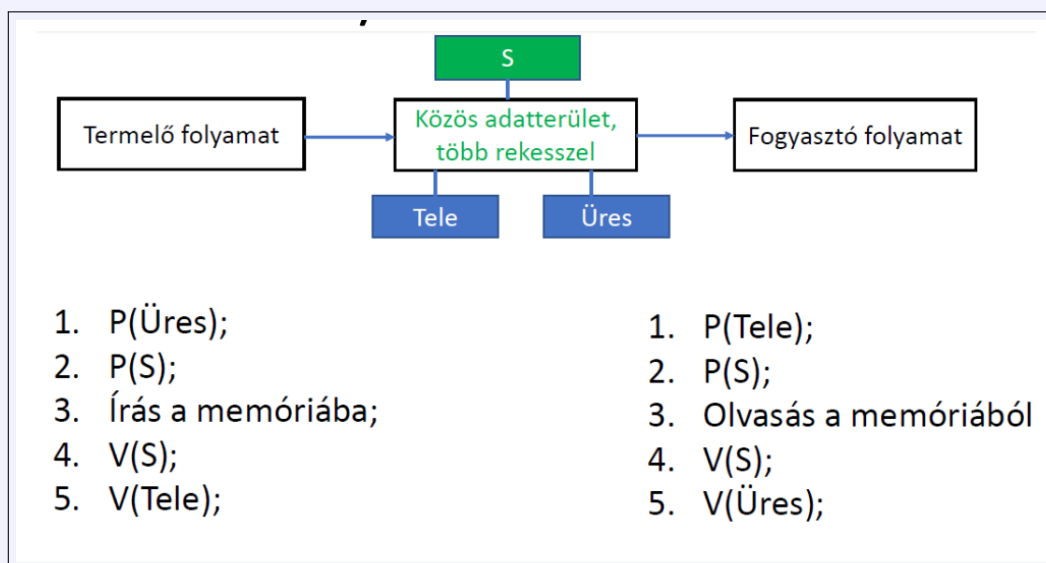


33. Ismertesse a folyamatok kommunikációját postaláda-kezelés esetén

- Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db) üzenet írható



- 3 db. szemafor a vezérléséhez
- **S**: a kölcsönös kizárást megvalósító szemafor (bináris; 0=**foglalt**; 1=**szabad**; kezdeti értéke: szabad)
- **Tele**: a tele helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:0)
- **Üres**: az üres helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:N)
- **P primitív**: a paraméterül kapott szemafor értékének EGGYEL CSÖKKENTÉSE (bináris szemafor esetén ez a **FOGLALTTÁ ÁLLÍTÁS**)
- **V primitív**: a paraméterül kapott szemafor értékének EGGYEL NÖVELESE (bináris szemafor esetén ez a **SZABADDÁ ÁLLÍTÁS**)



34. Ismertesse példával a bankár algoritmust!

- Biztonságosan tervezett az a folyamatok és erőforrásokat tartalmazó rendszer, amelyben létezik a folyamatoknak (legalább egy) olyan sorrendje, amely szerint végrehajtva őket, azok maximális erőforrás igénye is kielégíthető.
- A biztonságos rendszerben nem lehetséges holtpont kialakulása.
- Az ellenőrzést a bankár algoritmussal végezzük folyamat indítás és erőforrás foglalás előtt.

Bankár algoritmus példa

• Egy rendszerben 3 erőforrás van: E1 10 darab, E2 5 darab, E3 7 darab.

• Ebben a rendszerben 5 folyamat fut: P1,P2,P3,P4,P5.

• Max igény (1. lépés)

	E1	E2	E3
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

Aktuálisan **foglalt**(2.lépés)

	E1	E2	E3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

• Biztonságos-e ez az állapot?

• 3. lépés: igény = max igény - foglal.

Max	E1	E2	E3
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

Foglalt	E1	E2	E3
P1	0	1	0
P2	3	0	2
P3	3	0	2
P4	2	1	1
P5	0	0	2

igény	E1	E2	E3
P1	7	4	3
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

• 4. lépés: a szabad erőforrások számának (készlet) meghatározása: E1: $10-8=2$, E2: $5-2=3$, E3: $7-7=0$. A készletet vektorban írva: (2,3,0).

• 5. lépés: megvizsgáljuk, hogy a készletből (2,3,0) valamelyik folyamat igénye kielégíthető-e?

• igény ≤ készlet

• P2 folyamat ilyen.

• 6. lépés: P2-t lefuttatjuk.

• P2 lefuttatása után az általa lefoglalt erőforrások (3,0,2) felszabadulnak, készlethez adódnak, P2-t mátrixból töröljük.

• 7. lépés: új készlet: (5,3,2).

• 8. lépés: ha még van folyamat, vissza 5-ös lépésre.

igény	E1	E2	E3
P1	7	4	3
P2	0	2	0
P3	6	0	0
P4	0	1	1
P5	4	3	1

• 5. lépés: készlet (5,3,2) igény kielégíthető-e?

• P5 (4,3,1) folyamat ilyen.

• 6. lépés: P5-t (0,0,2) lefuttatjuk.

• 7. lépés: új készlet: (5,3,4).

• 8. lépés: még van folyamat, vissza 5-ös lépésre.

igény	E1	E2	E3
P1	7	4	3
P3	6	0	0
P4	0	1	1
P5	4	3	1

• 5. lépés: készletből (5,3,4) igény kielégíthető-e?

• P4 (0,1,1) folyamat ilyen.

• 6. lépés: P4-t (2,1,1) lefuttatjuk.

• 7. lépés: új készlet: (7,4,5).

• 5. lépés: P1(0,1,0) lefuttat.

• 7. lépés: (7,5,5),

• 5. lépés: P3(6,0,0) igény lefuttat. (3,0,2) új készlet:

• 7. lépés (10,5,7)

• 8. lépés: találtunk 1 sorrendet, amely elkerüli a holtpontot, biztonságos állapotban vagyunk.

igény	E1	E2	E3
P1	7	4	3
P3	6	0	0
P4	0	1	1

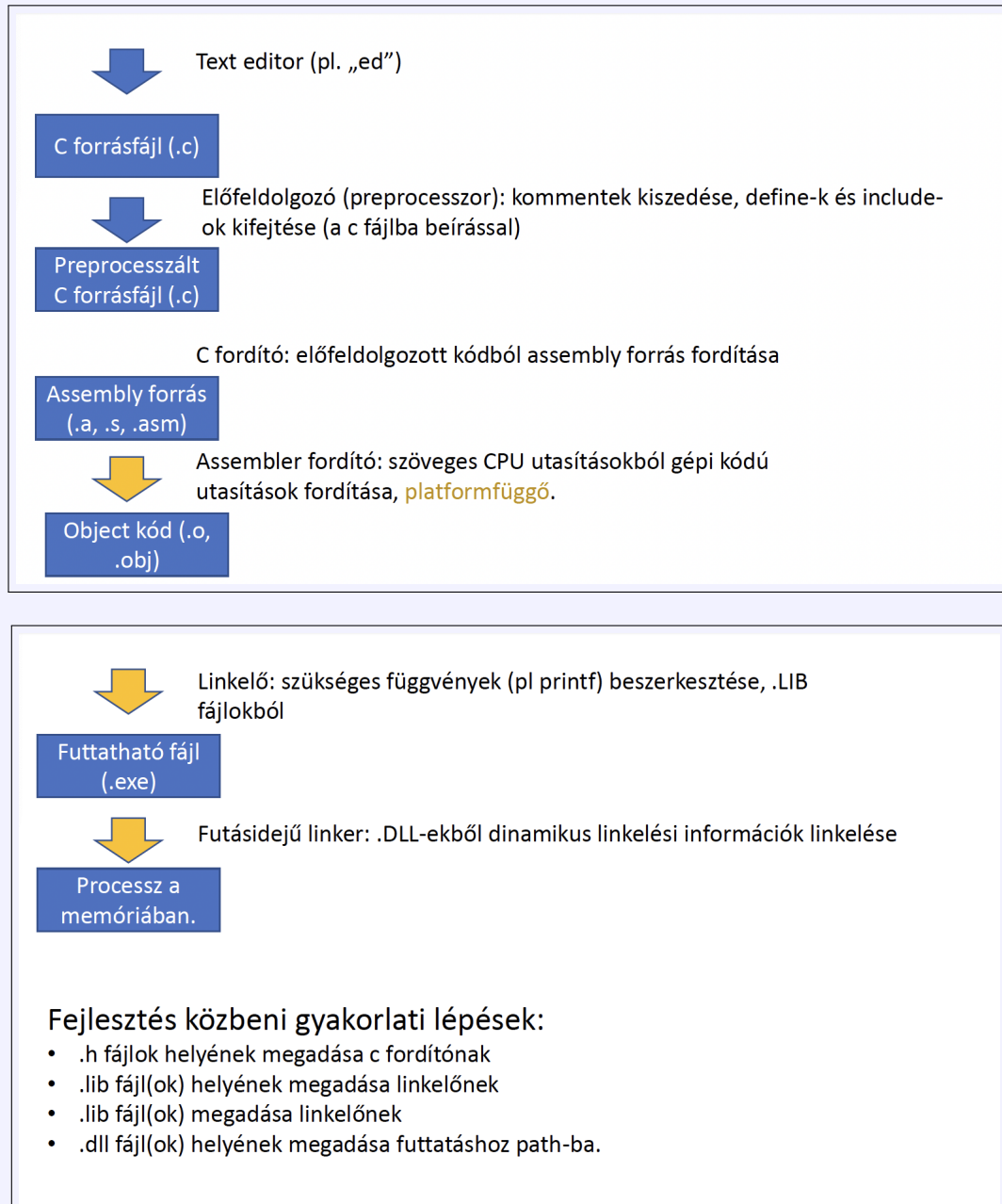
35. Ismertesse az operációs rendszerek feladat-ütemezési algoritmusait!

Többfeladatos operációs rendszerben a folyamatok közti átkapcsolást végzik.

Alap algoritmusok:

- FCFS (First Come First Served) előbb jött, előbb fut. Egyszerű, érkezési időtől függ, egy hosszú folyamat a többieket megfogja (kamion hatás)
- SJF (Shortest Job First) előbb a legrövidebb fut. Legrövidebb várakozás, előre tudni kell a lefutási időt, hosszú folyamat később kerül sorra (kiéheztetés)
- RR (Round Robin) időosztásos.
 - A folyamatokat körbe szervezzük, minden folyamat csak egy időszeletet kap, lejártá után elveszi tőle az OS a vezérlést.
 - Előbb is visszaadhatja, ha éppen nincs dolga.
 - Prioritással kombinálható, ekkor egy prioritási szintnek saját köre van.
 - Egyszerű algoritmus, nincs kiéheztetés.
 - Időszelet lejártakor állapot mentés, kör újra odaértekor visszaállítás (idő).
 - Idle folyamat (unix): egy kör végén már nincs folyamat, amelyet végre kellene hajtani.

36. Ismertesse a C nyelvű forráskód fordítási és futtatási folyamatát!



37. Ismertesse a UNIX file-elérési jogok oktális kódolását!

A UNIX file-elérési jogok oktális kódolása három számjeggyel történik, ahol minden számjegy a jogosultságok egy csoportját reprezentálja. Az oktális számjegyeket 0-tól 7-ig lehet használni, és a következő jelentéseik vannak:

1. Az első számjegy a tulajdonos jogait reprezentálja

- 0: Nincs jogosultság
- 1: Csak végrehajtási jog
- 2: Csak írási jog
- 3: Írási és végrehajtási jog
- 4: Csak olvasási jog
- 5: Olvasási és végrehajtási jog 6: Olvasási és írási jog
- 6: Olvasási és írási jog
- 7: Olvasási, írási és végrehajtási jog

2. A második számjegy a csoport jogait reprezentálja.

- Az értékek megegyeznek az első számjegy jelentéseivel.

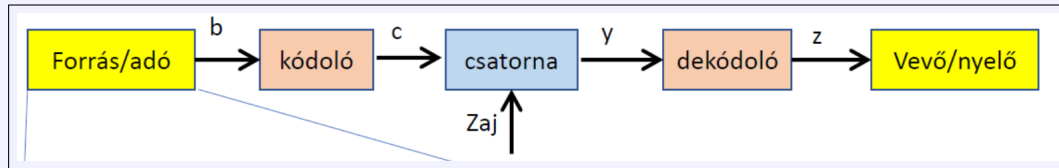
3. A harmadik számjegy a többi felhasználó jogait reprezentálja.

- Az értékek megegyeznek az első számjegy jelentéseivel.

Példa: jogosultságokat össze lehet adni a megfelelő számjegyekkel. Például, ha a tulajdonosnak olvasási, írási és végrehajtási jogai vannak (7), a csoportnak csak olvasási és végrehajtási jogai vannak (5), és a többi felhasználónak csak végrehajtási joga van (1), akkor a jogokat az oktális kódolásban a következőképpen reprezentálhatjuk: 751

38. Ismertesse a hírközlés Shannon-féle modelljét a modellben található egységek funkciójának leírásával!

A hírközlés során egy üzenetet juttatunk el térben és/vagy időben másik pontra.



1. **Adatkészlet:** Az eredeti üzenet vagy adat, amit továbbítani szeretnénk.
2. **Forráskódolás:** Az adatkészlet hatékonyabb formába való átalakítása a tárolás és továbbítás céljából.
3. **Kódolás:** Az adatok kódolása a zajállóság és hibajavítás érdekében.
4. **Csatorna:** A közeg, amelyen keresztül az adatok továbbításra kerülnek.
5. **Zaj:** A zavaró hatás, amely befolyásolja az adatok továbbítását és értelmezését.
6. **Dekódolás:** A csatornakódolt adatok visszaalakítása az eredeti formába.
7. **Végpont(Vevő/Nyelő):** A kommunikáció résztvevőinek végpontjai, ahol az adatok fogadása és továbbítása történik.

39. Ismertesse az információ mennyiség Hartley-féle és Shannon-féle meghatározását!

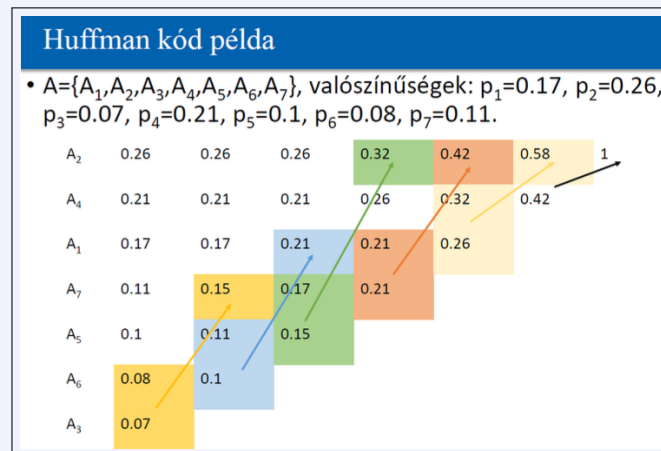
Az információvalamely véges számú, előre ismert esemény közül annak megnevezése, hogy melyik következett be.

- **Hartley:** m számú, azonos valószínűségű esemény közül egy megnevezésével nyert információ: $I = \log_2 m$
- **Shannon:** minél váratlanabb egy esemény, bekövetkezése annál több információt jelent.

40. Ismertesse számpéldával a Huffman-kódolást!

A legrövidebb átlagos szóhosszúságú prefix kód.

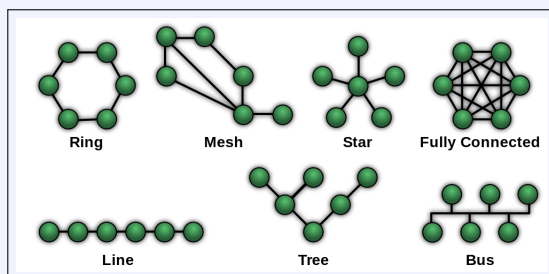
1. Valószínűségek szerint sorba rendezia forrásszimbólumokat.
2. A két legkisebb valószínűségű szimbólumot összevonja. Az összevont „szimbólum” valószínűsége a két másik összege.
3. Az 1-2 lépést addig ismétli, amíg egy darab, 1 valószínűségű szimbólum marad.
4. A kapott gráf minden csomópontja előtti két élt megcímkézi 0-val és 1-gyel: ez lesz a kódfa. Bináris fa.
5. A kódfa gyökerétől indulva megkeresi az adott szimbólumhoz tartozó útvonalat, kiolvassa az éleknek megfelelő biteket. A kapott bitsorozatot rendeli a szimbólumhoz kódszóként.



41. Ismertessen fizikai és logikai hálózati topológiákat!

Fizikai hálózati topológiák:

1. **Busz topológia:** Az eszközök egy közös adatbuszon vannak összekapcsolva. Az adatokat az adatbuszon továbbítják, és az összes eszköz figyeli a buszt. Az adatok mindkét irányban közösek a buszon.
2. **Gyűrű topológia:** Az eszközök egy körforgásban vannak elrendezve, és az adatok körforgásszerűen továbbítódnak az eszközök között. Minden eszköz fogadja és továbbítja az adatokat.
3. **Csillag topológia:** Az eszközök központi csomópontra (switch vagy hub) vannak csatlakoztatva. Az adatok a csomóponton keresztül továbbítódnak az eszközök között.
4. **Fa topológia:** Az eszközök fa szerkezetben vannak elrendezve, ahol a központi csomópontok csatlakoznak az alsó szintű eszközökhöz. Az adatok a fa struktúrán keresztül továbbítódnak.



Logikai hálózati topológiák:

1. **Csillag topológia:** Az eszközök központi csomópontra vannak csatlakoztatva, de a kommunikáció közvetlenül a csomóponton keresztül történik.
2. **P2P (peer-to-peer) topológia:** Az eszközök közvetlenül egymással vannak összekapcsolva, és közvetlenül kommunikálnak egymással anélkül, hogy központi csomópontot használnának.
3. **Hierarchikus topológia:** Az eszközök hierarchiában vannak szervezve, ahol van egy központi csomópont, amely összekapcsolja az alsóbb szintű csomópontokat.
4. **Mesh topológia:** Az eszközök közvetlenül egymással vannak összekapcsolva, és több út áll rendelkezésre az adatok továbbításához. Ez a topológia magas redundanciát és megbízhatóságot biztosít.

Melyik hálózati elsőbbségi elvhatékony egy kis terhelésű és egy nagy terhelésű hálózaton?

- **Kis terhelésű hálózaton:** Az arányos elosztás vagy fair share elve hatékony. Ez az elv az erőforrásokat egyenlően osztja szét az eszközök vagy felhasználók között, így mindenki azonos feltételekkel érheti el a rendelkezésre álló erőforrásokat.
- **Nagy terhelésű hálózaton:** A prioritás alapú elv hatékony. Ebben az esetben bizonyos forgalom vagy szolgáltatások kapnak előnyt a többi forgalommal szemben. Ez lehetővé teszi a kritikus feladatok gyorsabb és megbízhatóbb végrehajtását a nagy terhelésű környezetben.

42. Ismertesse az ethernet hálózaton használt eszközöket!**EZT ÁT KELL NÉZNI**

- **Számítógép:** A számítógépek az ethernet hálózatok alapvető részei, amelyek csatlakoznak a hálózati infrastruktúrához és kommunikálnak más eszközökkel.
- **Switch:** A switch (kapcsoló) az ethernet hálózat központi eszköze, amely összekapcsolja a számítógépeket és más eszközöket a hálózaton belül, és csomagokat továbbít a megfelelő célállomásokhoz.
- **Router:** A router (útválasztó) a hálózatok közötti adatforgalmat irányítja és továbbítja. Az ethernet hálózaton a router a hálózatok közötti kapcsolatot biztosítja.
- **Modem:** A modem lehetővé teszi az internet-hozzáférést az ethernet hálózaton keresztül. Átalakítja az analóg jeleket digitális formátumba, és fordítva.
- **Hub:** A hub (központi elosztó) az adatokat egy portról minden másik portra továbbítja. Azonban a hub nem tudja figyelni és különbséget tenni a címzett eszközök között, ami korlátozza a hálózati teljesítményt.
- **Media converter:** A media converter (közvetítő) lehetővé teszi a különböző hálózati médiumok (például réz és optikai kábel) közötti átalakítást és összekapcsolást.

Hogy védi ki a zavarokat a csavartérpár?

- **Crossover elrendezés:** A csavart érpárookban a vezetékek páronként keresztezik egymást, így minimalizálva a zavarok befolyását a jelekre.
- **Párhuzamos vezetékek:** A csavart érpárok párhuzamos elhelyezése a kábelben segít minimalizálni a zavarokat, mivel a különböző párokból áramló jelek egymástól távolabb vannak.
- **Árnyékolás:** A csavart érpárok kábeleit általában árnyékolják, hogy csökkentsék a külső elektromágneses interferenciát.
- **A földelés szerepe:** A csavart érpárok földelése további védelmet nyújt a zavarok ellen, mivel a földelés lehetővé teszi a nem kívánt elektromos energiák levezetését.

43. Ismertesse az internetes címek maszkolási algoritmusát!

- **IP-cím:** Az IPv4 protokoll használ 32 bites IP-címeket, amelyeket négy oktetre osztanak fel (pl. 192.168.0.1).
- **Hálózati maszk:** A hálózati maszk meghatározza, hogy az IP-cím hálózati része hogyan van elkülönítve a hoszt résztől. Pl. 255.255.255.0 jelenti, hogy az első három oktett a hálózat, az utolsó oktett pedig a hoszt része.
- **AND művelet:** Az IP-cím és a hálózati maszk bitjeit az AND művelettel kombinálják, hogy meghatározzák az adott IP-cím hálózati részét. Pl. 192.168.0.1 AND 255.255.255.0 = 192.168.0.0, a hálózat címe.
- **Címek osztályozása:** Az IP-cím első néhány bitje meghatározza az IP-cím osztályát (A, B, C stb.), ami alapján a hálózati maszkot is meghatározzák.
- Az IP-címek maszkolása lehetővé teszi a hálózati és hoszt részek elkülönítését

Mire használható a maszkolás?

- **IP-címek osztályozása:** Az IP-címeket maszkolással osztályokba vagy alcsoportokba lehet sorolni, ami segíti az internetes forgalom irányítását és hálózati felépítést.
- **Alhálózatok létrehozása:** A maszkolás lehetővé teszi, hogy egy nagyobb hálózatot kisebb alhálózatokra osszunk fel, ami hatékonyabb hálózati erőforrásfelhasználást és forgalomkezelést eredményez.
- **Hálózati szegmentálás:** A maszkolás segít a hálózatok fizikai vagy logikai szegmentálásában, amely javítja a hálózati teljesítményt, biztonságot és skálázhatóságot.
- **Hozzáférés-szabályozás:** A maszkolás lehetőséget nyújt hálózati hozzáférés-szabályozásra, hogy csak bizonyos IP-címrészek vagy alcsoportok kommunikálhassanak egymással.

A maszkolásnak fontos az IP-hálózatok hatékony tervezésében, kezelésében és biztonságában.

44. Ismertesse a NAT (Network Address Translator) eszközt használatát az internethez való kapcsolódásban!

- **Privát IP-címek:** Az otthoni vagy vállalati hálózatokon belül az eszközök privát IP-címet használnak, amelyek nem egyediek a globális interneten.
- **Közös IP-cím:** A NAT-eszköz az internetre kapcsolódó privát hálózatokat egy közös IP-cím alatt reprezentálja.
- **IP-cím átirás:** Amikor a privát hálózatról egy eszköz csatlakozik az internetre, a NAT-eszköz átírja az IP- címet a közös IP-címre, és megjegyzi a kapcsolódó portszámot.
- **Válaszok visszirányítása:** Amikor az internetről érkezik válasz a kapcsolat kérelmére, a NAT-eszköz a portszám alapján visszairányítja a választ a megfelelő privát IP-címre és portra.
- **Többes kapcsolatkezelés:** A NAT-eszköz lehetővé teszi több eszköz számára, hogy ugyanazon a közös IP- címen keresztül kommunikáljon az interneten, ami csökkenti az IP-címek szükségességét.
- A NAT-eszköz használata lehetővé teszi, hogy a privát hálózatok több eszköze is kapcsolódhasson az internethez egy korlátozott mennyiségű közös IP-cím alatt. Ez növeli az internetes kapcsolat kihasználtságát és biztosítja az eszközök hálózati kommunikációjának hatékonyságát és biztonságát.

45. Ismertesse az internet DNS rendszerét! (Domain Name System)

- **DNS rendszer:** Elérési útvonal a tartomány nevek (pl. example.com) és az IP-címek között.
- **DNS kérések:** Kliensek (pl. böngészők) DNS-kérést indítanak a webhelyek IP-címének megszerzéséhez.
- **DNS kiszolgálók:** Az interneten elhelyezett szerverek, amelyek végzik a DNS-kérések kezelését és a válaszok adását.
- **Domain regisztráció:** A webhelyek tulajdonosai domainekeket regisztrálnak és hozzárendelik az IP- címüket.
- **DNS zónák:** A DNS-rendszer hierarchikus struktúrában van szervezve, zónákra oszlik, melyek tartalmazzák az adott tartományhoz (pl. example.com) tartozó rekordokat.
- **DNS feloldás:** A DNS-kiszolgálók megkeresik a kért tartomány IP-címét, majd visszaküldik a kérőnek.
- **Gyorsítótár (caching):** A DNS-kiszolgálók gyakran tárolják a korábban kérések során megszerzett adatokat, hogy csökkentsék a válaszidőt és a hálózati forgalmat.

46. Rendszerezze a tömörítő eljárásokat az eredeti adat visszaállíthatósága szerint!

- **Lossless:** Fájlokhoz veszteség mentes (zip)
- **Lossy:** Kép és hangadatokhoz veszteséges (jpg, mp3). Codec: tömörített és nyers (raw) adat közti szoftver vagy hardver.

Mondjon példát a ma mobiltelefonokban is használt képtömörítési eljárás lépéseire!

- Szétválasztja a fényesség (Y) és szín (C) komponenst. Szín felbontást felére redukál.
- 8x8-as blokkokra osztja a képet, majd DCT-t futtat rajtuk (Fourier-hez hasonló)
- Frekvencia amplitúdókat sorba rak, magasakat elhagy
- Veszteségmentesen tömörít Huffman eljárással.

47. Ismertesse a C nyelv alapvető típusait, tárolási méretükkel! Hol vannak ezek a méretek definiálva (típusonként eltérő)?

A C nyelvben számos alapvető adattípus található, amelyek különböző tárolási mérettel rendelkeznek. Ezek az alapvető típusok és a tárolási méretek általában platformfüggőek, tehát a méretek eltérhetnek a különböző rendszerek és implementációk között. Azonban a C nyelv szabványa definiál néhány minimális tárolási méretet az alábbi alapvető típusok számára:

- **char:** Legalább 1 bájt méretű, ami általában 8 bitet jelent.
- **short:** Legalább 2 bájt méretű.
- **int:** Legalább 2 bájt méretű.
- **long:** Legalább 4 bájt méretű.
- **longlong:** Legalább 8 bájt méretű.

Az alapvető típusok tárolási méretei platformonként változhatnak, és a "sizeof" operátorral lehet lekérdezni a konkrét méreteket a programban. A "sizeof" operátor visszaadja a kifejezés által foglalt tárolóegységek számát bájtban.

Példa:

```
#include <stdio.h>
int main() {
    printf("char: %zu byte\n", sizeof(char));
    printf("short: %zu bytes\n", sizeof(short));
    printf("int: %zu bytes\n", sizeof(int));
    printf("long: %zu bytes\n", sizeof(long));
    printf("long long: %zu bytes\n", sizeof(long long));
    return 0; }
```

48. Ismertesse példával a pre-és posztinkrementálás közti különbséget!

- A preinkrementálás esetén az érték előbb növekszik, majd az új érték adódik át a kifejezésben vagy értékadásban részt vevő változónak. A posztinkrementálás esetén viszont az értékadás vagy a kifejezés kiértékelése előtt az eredeti érték kerül átadásra, majd csak utána növekszik az érték.

Preinkrementálás ($++x$):

```
int x = 5;
int y = ++x; // x értéke előbb növekszik, majd értékadás történik
// x = 6, y = 6
```

Posztinkrementálás ($x++$):

```
int x = 5;
int y = x++; // értékadás történik, majd x értéke növekszik
// x = 6, y = 5
```

49. Ismertesse a C nyelv háromoperandusú operátorát!

Logikai_kifejezés ? érték_ha_igaz : érték_ha_hamis

- Megspórolunk egy elsét, és egy változónevet.
- `if (a > b) c = a; else c = b; // c értéke legyen a nagyobbik a és b közül`
- Helyette írható: `c = a > b ? a : b;`

50. Ismertesse a „break” utasítás szerepét a „switch” utasításban!

A **case** esetek csak belépési pontok. Ha a **switch**-ben lévő változó értéke egyenlő a **case**-ben található értékkel, a vezérlés oda kerül. Ha nem írtunk **break**-et, a végrehajtás után a következő **case**-ben található utasítást is végrehajtja, a végéig utána a **switch** végétől folytatja. Ha a **break** az utolsó eset, a záró **break** elhagyható.

51. Ismertesse példával a C nyelvben alkalmazható bitműveleteket!

- **Egyoperandusú negálás:** minden bitet külön negál ($0 \rightarrow 1$, $1 \rightarrow 0$)
- **Kétooperandusú műveletek:**
 - "És" `&` az eredményben ott lesz 1-es bit, ahol mindkét operandusban 1-es volt. Máshol 0 lesz.

A	0	0	1	1	0	0	1	0
B	0	1	0	1	1	1	1	0
A & B	0	0	0	1	0	0	1	0

- "Vagy" `||` az eredményben ott lesz 1-es bit, ahol valamelyik operandusban 1-es volt. Mindkét 0-nál 0 lesz.

A	0	0	1	1	0	0	1	0
B	0	1	0	1	1	1	1	0
A B	0	1	1	1	1	1	1	0

- "Kizáró vagy" az eredményben ott lesz 1-es bit, ahol csak egyik operandusban volt 1-es. Mindkét 0-nál és 1-nél is 0 lesz. (Elnevezése még: xor, exor, antivalencia.)

A	0	0	1	1	0	0	1	0
B	0	1	0	1	1	1	1	0
A ^ B	0	1	1	0	1	1	0	0

Miért volt szükség logikaiműveletekre a bitműveletek mellett?

A logikai műveletek (ÉS, VAGY, XOR stb.) azért voltak szükségesek a bitműveletek mellett, mert ezekkel a műveletekkel egyszerűen és hatékonyan manipulálhatók az egyes bitek egy adott adatban. Például bitenkénti ÉS használatával egyes bitek maszkolhatók, logikai VAGY használatával bitjei beállíthatók, XOR használatával bitjei inverz állapotba hozhatók stb. Ez lehetővé teszi a hatékonyabb és precízebb adatmanipulációt.

52. Ismertesse a C nyelv „pointer” fogalmát és az indirekció fogalmát!

- Annak a változónak amely nem adatot, hanem címet tárol, Pointernek nevezzük.
- Az adatot pointer használatakor nem direktben érjük el (mint a sima változónál), hanem indirektben (először el kell menni a címéért). Az indirekció a jele a „*”.

53. Ismertesse a pointer aritmetikát!

- A fordító értelmez néhány műveletet a pointerekkel is.
- A pointer és egy egész szám összege/különbsége: a pointer után/előtt álló adat a memóriában
- Ha p egy int-re mutató pointer, $p + 1$ a következőint-re mutató pointer.
- Ha q egy double-re mutató pointer, a $q - 1$ az előző double-re mutató pointer lesz.
- Hasonlóan $p + n$ az n . int-re mutató pointer lesz
- Két (azonos típusú adatra mutató) pointer különbsége a köztük lévő adatok száma.
- Két pointer összehasonlítás értelmezve, fordítási hibát okoz.
- A **pre-és posztinkrementálás** és dekrementálás is értelmezett: $(++p)$ $(p++)$; utasítástól a pointer a következő/előző adatra mutat.
- ha **malloc**-kal foglaltunk helyet a memóriában, nem szabad a pointert máshova állítani („elrontani”), mert a free már hibát fog okozni, mert a terület kezdetére kell mutatni a pointernek a felszabadításhoz. A hiba a programból való kilépéskor keletkezik.

54. Ismertesse a tömbök használatánál előforduló komplexitás-típusokat! Mondjon példát arra a műveletre, amelyik adott komplexitású!

- **Konstans:** egy elem helyének (címének) kiszámítása, elem elérése
- **Lineáris:** összes elem elérése (bejárás)
- **Logaritmikus:** bináris keresés.
- **Polinomiális:** rendezés.

55. Ismertesse a tömbből való kicímzés veszélyeit!**• Túlindexelés:**

- Ha nem ellenőrizzük a tömb határait, túlindexelhetünk, vagyis hozzáférhetünk olyan tömbelemhez, amely a tömb határain kívül esik.
- Ez memóriaszivárgáshoz, adatvesztéshez vagy hibás működéshez vezethet.

• Alulindexelés:

- Az alulindexelés esetén olyan tömbelemhez próbálunk hozzáférni, amelynek indexe negatív vagy az első elem előtti.
- Ez nem várt értékekhez vagy hibás működéshez vezethet.

• Nem megfelelő méretű tömb használata:

- Ha nem megfelelő méretű tömböt használunk a tárolandó adatokhoz, túl sok vagy túl kevés memóriát foglalhatunk le.
- Ez memóriakimaradáshoz, memóriakorrupcióhoz vagy futásidejű hibákhoz vezethet.

• Tömb határainak átlépése:

- Ha egy tömbben tárolt adatot megváltoztatunk anélkül, hogy tisztában lennénk a határaival, a változtatás más adatokat is érinthet.
- Ez nem várt eredményekhez, hibás működéshez vagy adatvesztéshez vezethet.

• Tömbkezelési hibák:

- Tömbök használatakor figyelni kell az indexelési szabályokra, a tömbméretre, az adat-típusokra és az adatrendezésre.

Nem megfelelő tömbkezelés hibás működéshez, memóriaproblémákhoz vagy adatvesztéshez vezethet

56. Ismertesse a tömbök [] operátorának jelentését! Hogy írható fel a [] operátor pointeraritmetikát felhasználva?

- A [] operátor segítségével hivatkozhatunk és manipulálhatunk egy tömb elemire. Az operátor index paramétert vár, amely megadja, hogy melyik elemre szeretnénk hivatkozni a tömbben. Az index 0-tól kezdődik, és a tömb méretével egyenlő vagy annál kisebb egész értéket vehet fel.
- A [] operátor pointer aritmetikával is felírható. Ha XYZ egy tömbre mutató pointer, és index az elem indexe, akkor az alábbi módon használható fel a pointer aritmetika: $XYZ[index]$ ekvivalens a $*(XYZ + index)$ kifejezéssel.
- Ez azt jelenti, hogy a [] operátor a $*(XYZ + index)$ kifejezés rövidítése, ahol XYZ a tömbre mutató pointer, és index az elem indexe. Az $(XYZ + index)$ kifejezés az XYZ címéhez hozzáadja az index értékét, majd a "*" operátorral megfelelően dereferálja azt, így elérve az adott tömb elemét.

Fontos megjegyezni, hogy a [] operátor használata a tömb méretét nem ellenőrzi, így fontos gondoskodni arról, hogy az index ne lépje túl a tömb határait.

57. Ismertesse a szomszédcserés rendezés működését és komplexitását!

- A legegyszerűbb(en megírható), kis elemszáma hatékony rendezés a szomszédcserés (buborék) rendezés. Elve:
 - Összehasonlítunk két szomszédos elemet (pl. $t[i]$ -t és $t[i+1]$ -et).
 - Ha nekünk rossz sorrendben vannak, felcseréljük őket, hogy jó sorrendben legyenek.
 - Ha jó sorrendben vannak, nem rontjuk el őket (vagyis semmit sem csinálunk velük).

```
procedure BubbleSort(A: array of Comparable)
n = length(A)
for i from 0 to n - 1
    swapped = false
    for j from 0 to n - i - 2
        if A[j] > A[j + 1]
            swap A[j] and A[j + 1]
            swapped = true
    end for
    if not swapped
        break
    end if
end for
end procedure
```

58. Ismertesse a függvények írásának és paraméterezésének nyelvi szabályait a C nyelvben!

- Minden nyelvi elemet használat előtt deklarálni kell, a változókat is, a struktúrákat is, és a függvényeket is.
- Függvényen belül függvény nem definiálható (csak 1 szint létezik a memóriában).
- Készíthető paraméterezett, paraméter nélküli, visszatérő érték kel rendelkező és visszatérő érték nélküli függvény. A használatkor ugyanannyi és ugyanolyan típusú változó szükséges, mint a deklarációnál. Ez alól két speciális kivétel van: a változó számú és típusú paraméterrel használható *printf()* és *scanf()*. A *va_arg*-ot használó függvény nem része a tematikának, ilyet nem fogunk írni.
- A függvények neveire ugyanazok a megkötések, mint a változók neveire: egyedi nek kell lenni, ami nem lehet egy, már deklarált változó/függvény neve. Nem lehet foglalt kulcsszó, csak az engedélyezett karaktereket
- A programunk alapértelmezetten elinduló függvénye a *main()* nevű, amelyet a program indításakor az operációs rendszer lefuttat. A *main()* átdefiniálható, mert ez csak egy cím a futtatható fájlban (exe). A return a hívóhoz tér vissza: windows-ból indítva windows-ba, visual studióból indítva oda.

A használat meghívással valósul meg. A veremtárba elmentjük, hogy mi következik, majd a függvény memóriacímére ugrunk a memóriában a cpu végrehajtó utasítás pointerével (pl. pc-nél: IP), majd a függvény végén a veremtárból felolvasott címen (a hívás utáni címen) folytatjuk.

59. Ismertesse a formális és aktuális paraméter fogalmát a C nyelv függvényeinél!

- **Formális Paraméternek:** A függvény neve utáni zárójelben a paramétereit adhatjuk meg. Minden paraméter típusát egyenként meg kell adni.
- **Aktuális paraméterek:** Ezek a paraméterek az adott függvényhívás során megadott értékek vagy kifejezések. Az aktuális paraméterek a függvényhívás során átadódnak a függvénynek, és azok a formális paraméterekkel kerülnek párosításra.

60. Milyen paraméter-átadást használ a C nyelv?

A C nyelv alapértelmezetten érték szerinti paraméter-átadást használ. Ez azt jelenti, hogy amikor egy függvénynek átadunk egy paramétert, akkor a függvény a paraméter értékének másolatát kapja meg, és a függvényen belül végzett módosítások nem változtatják meg az eredeti változó értékét a hívó folyamatban.

Hogy tudja megváltoztatni a függvény aparaméterei értékét úgy, hogy a hívó folyamatban is a változtatott érték szerepeljen? A C nyelvben a paraméterek értékének megváltoztatásához pointer-eket használhatunk. Azaz, ha egy függvénynek átadunk egy paramétert cím szerint ('&' operátorral), akkor a függvényen belül közvetlenül a paraméter címére mutató pointerrel dolgozhatunk, és az értékeket közvetlenül módosíthatjuk, ami a hívó folyamatban is érvényesül.

61. Ismertesse a lokális és globális változó fogalmakat!

- Minden változó, amelyet egy program blokkon {} belül deklaráltunk, a blokkon belül lokálisváltozóként működnek, a blokk végén } megszűnnek. Ha még egyszer meghívjuk a függvényt, a megszűnt változót újra deklaráljuk, az előző megszűnésekor lévő érték nem lesz benne.
- Azok a változók, amelyek a file szintjén (vagyis a függvényeken kívül) lettek deklarálva, globálisváltozók. Minden függvényben ugyanazt jelentik. Ha adott nevű változóból van lokális és globális is, a változó neve a lokális változót jelenti, elérakva a :: operátort a globális változót. Globális változó megadható .h-fájlban is extern-ként, valamint valamelyik .c-ben (csak 1-ben) deklarálva.

62. Ismertesse a string tárolás C nyelvű megvalósítást, valamint a stringek kezelésére leggyakrabban használt függvényeket!

Deklaráció és helyfoglalás:

- `char * stringnv = (char*)malloc(stringmrete);`
- A mallocba nem kell `sizeof`, mert esetünkben definíció szerint 1. Kis rendszereknél, vagy nagy memórafogyasztású programoknál ellenőrizzük a visszatérő értéket! NULL pointer esetén elfogyott a memória.
- A végjel 0-nak is szüksége van helyre. Vagyis n karakter tárolásához $n + 1$ helyet kell allokalni.

Stringek kezelésére leggyakrabban használt függvények:

- `strlen()`: String hosszának meghatározása.
- `strcpy()`: String másolása.
- `strcat()`: Stringek összefűzése.
- `strcmp()`: Stringek összehasonlítása.
- `strchr()`: Karakter keresése a stringben.
- `strstr()`: String keresése a másik stringben.

Ezek a függvények a `< string.h >` könyvtárban találhatók

63. Ismertesse a szöveges fájlok módosításának lépéseit!

- **Fájl megnyitása:** `fopen()` függvény (fájl megnyitása olvasásra/írásra).
- **Fájl tartalmának módosítása:** `fread()`, `fwrite()`, `fscanf()`, `fprintf()` függvények (adatok olvasása és írása a fájlba).
- **Fájl bezárása:** `fclose()` függvény (fájl bezárása és erőforrások felszabadítása).