A struktúrált programozás

A *struktúrált programozás* olyan programozási elvek összessége, amelyek segítenek megteremteni, hogy a program szövegszerkezete tükrözze a program végrehajtása során követett vezérlési folyamatot. A struktúrált programozás során a műveletekre összpontosítunk, az adatok szerepe, szerkezete másodlagos, de egyáltalán nem lebecsülendő.

Alkalmazásával könnyebben és hatékonyabban lehet programot felépíteni és könnyebb a program megértése. A struktúrált programozási elv megjelenését Edsger W. Dijkstra nevéhez kapcsoljuk az 1968-as cikke révén.

A jó program ismérvei:

1. Kielégíti a követelményeket, a felhasználói igényeket.
2. Helyesen, hibamentesen működik.
3. Amennyire szükséges, hatékony.
4. Teljesítése idő- és költséghatáron belül történik.
5. A jövőben módosítható. (Upgrade.) Ez megfelelő dokumentáltságot is jelent.

A struktúrált programozás általában együtt jár a *felülről lefelé programtervezési koncepció*val. (Lásd később.) A cél a bonyolultság kézbentartása.

Az emberi agy általában nehezen tud egyazon időpontban több célra koncentrálni. A struktúrált programozási elvek elősegítik ezen egybeesések szétválasztását. Egy iskolapélda erre az alább mellékelt program, amelyben kezdő programozók által elkövetett tipikus hiba található. A program a *New Haven-i esőátlagot kiszámító program* néven vált ismertté.

A program feladata: Napi csapadékmennyiségeket kell egy program számára megadni, amely számadatokból a program átlagot számol. Az adatok sorát egy 99999-es szám zárja, amelyet már nem kell figyelembe venni a csapadékátlag kiszámításánál. Negatív számot nem szabad elfogadni csapadékmennyiségnek, hanem helyette az adatot újra be kell kérni. A hibás verzió következik alább.

```
    // Soloway, CACM 9-86 p. 853 cikkbeli program pszeudokódosítva
1.
2.  VÁLTOZÓK Összeg, Eső, Átlag ∈ R
3.               Számláló ∈ N
4.
5.  Összeg ← 0
6.  Számláló ← 0
7.  OUTPUT ('Kérem az eső mennyiségét!')
8.  INPUT (Eső)
9.  WHILE  Eső ≠ 99999 DO
10.            WHILE  Eső < 0 DO
11.                   OUTPUT ('Eső mennyisége nem lehet negatív. Adja meg újra!')
12.                   INPUT (Eső)
13.            Összeg ← Összeg + Eső
14.            INC (Számláló)
15.            OUTPUT ('Kérem az eső mennyiségét!')
16.            INPUT (Eső)
17. IF         Számláló > 0
18.            THEN   OUTPUT ('Átlagos esőmennnyiség = ', Összeg / Számláló)
19.            ELSE   OUTPUT ('Nincs értékelhető adat, átlagszámítás nem történt!')
20. STOP
21. // ---------- New Haven-i esőátlag program vége ----------
```

A fenti hibás verzió kipróbálását egy kapkodó felhasználó a -3-as szám begépelésével kezdte, majd észrevéve a hibát megijedt, és a 99999-cel ki akart lépni a programból, de az számolni kezdett. Nyílvánvalóan a program nem felelt meg az előírt elvárásoknak. Tanulság: *Nehéz egyidejűleg több célt is követni!*

Alább mellékelünk egy lehetséges helyes változatot.

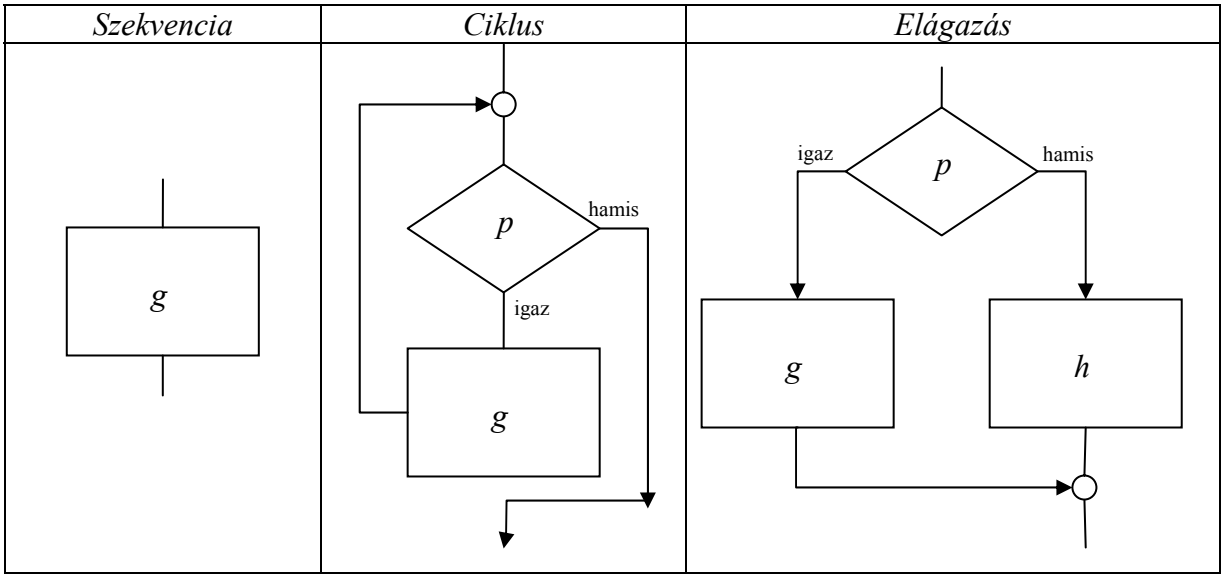
	New Haven-i esőátlag, javított változat
1.	
2.	// ---------- Egy procedúra a helyes inputra ----------
3.	Helyes_Input (*@x*) // ---------- A procedúra fejléce
4.	Output paraméter : $x \in \boldsymbol{R}$
5.	**OUTPUT** (’Kérem az eső mennyiségét!’)
6.	**INPUT** (*x*)
7.	**WHILE** $x < 0$ **DO**
8.	**OUTPUT**(’Eső mennyisége nem lehet negatív. Adja meg újra!’)
9.	**INPUT** (*x*)
10.	**RETURN** (*x*) // ----------Procedúra vége
11.	
12.	// ========== Főprogram
13.	**KONSTANS** *Szentinel* = 99999 (Szentinel jelentése őrszem)
14.	**VÁLTOZÓK** *Összeg*, *Eső* $\in \boldsymbol{R}$
15.	*Számláló* $\in \boldsymbol{N}$
16.	
17.	*Összeg* $\leftarrow$ 0
18.	*Számláló* $\leftarrow$ 0
19.	**CALL** Helyes_Input (*@Eső*)
20.	**WHILE** *Eső* $\neq$ *Szentinel* **DO**
21.	*Összeg* $\leftarrow$ *Összeg* + *Eső*
22.	**INC** (*Számláló*)
23.	**CALL** Helyes_Input (*@Eső*)
24.	**IF** *Számláló* > 0
25.	**THEN** **OUTPUT** (’Átlagos esőmennnyiség = ’, *Összeg* / *Számláló*)
26.	**ELSE** **OUTPUT** (’Nincs értékelhető adat, átlagszámítás nem történt!’)
27.	**STOP** // ========== Főprogram vége

A Helyes_Input procedúra függvény formában is megírható lett volna. Ebben az esetben az alábbi alakban állhatna.

	Helyes_Input $\in \boldsymbol{R}$ // ---------- A procedúra fejléce
1.	// Nincs input paramétere, a visszaadott függvényérték valós
2.	**OUTPUT** (’Kérem az eső mennyiségét!’)
3.	**INPUT** (*x*)
4.	**WHILE** $x < 0$ **DO**
5.	**OUTPUT**(’Eső mennyisége nem lehet negatív. Adja meg újra!’)
6.	**INPUT** (*x*)
7.	*Helyes_Input* $\leftarrow$ *x*
8.	**RETURN** // ----------Procedúra vége

A függvény változatot használva a programban a **CALL** Helyes_Input (*@Eső*) utasítás helyett az *Eső* $\leftarrow$ Helyes_Input utasítást írhatnánk.

A struktúrált programozás a programokat (és folyamatábráikat) három alapelemből építi föl:



Jellemzőjük:

1. Minden építőelemnek csak egy belépési és egy kilépési pontja van.
2. A belépési pont a struktúra elején, a kilépés a végén van.

Az egyes alapelemek formulával is leírhatók.
A szekvencia formulája: ***S***(*g*)
Ha több szekvencia követi egymást, akkor azokat írhatjuk egy közös zárójelben felsorolva és vesszővel elválasztva egymástól. A pszeudokódban általában az értékadás, procedúrahívás, input-output, adatmozgatás, függvényérték kiszámítás, adattranszformáció a megfelelője.

A ciklus formulája: ***C***(*p*; *g*)
A *p* itt predikátum, a *g* ciklusmag a predikátum „igaz” értéke esetén kerül végrehajtásra.

Pszeudokódban: **WHILE** *p* **DO**
g

Az elágazás formulája: ***E***(*p*; *g*, *h*)
A *p* itt predikátum, melynek igaz értéke esetén az elsőnek feltüntetett *g*, hamis értéke esetén a másodiknak feltüntetett *h* kerül végrehajtásra. Egyikük lehet üres is. Ebben az esetben helyükre a mínusz (-) jelet, vagy az ∅ üres halmaz jelet írhatjuk.

Pszeudokódban: **IF** *p*
THEN *g*
ELSE *h*

Figyeljük meg, hogy mind a ciklus, mind az elágazás formulában a predikátum a formula elején foglal helyet és a formula többi részétől pontosvessző választja el, míg máshol az elválasztás a vessző jellel történik

Definíció: **<u>Két program ekvivalenciája</u>**
Két programot ekvivalensnek nevezünk, ha minden lehetséges ugyanazon input esetén ugyanazon outputot eredményezik. (Ugyanazon kezdőállapotból ugyanazon végállapotba jutnak el.)

Definíció: **Programgráf**

A programgráf, vagy vezérlési gráf olyan összefüggő irányított gráf, amely vonalakból (folyamatvonalakból), szekvenciákból (függvényekből), elágazásokból (*predikátum*okból, egy befutó, két kifutó él) és gyűjtő szimbólumokból (két befutó, egy kifutó él) épül fel. A vonalak a gráf éleit adják, a többiek pedig a gráf csomópontjait. A predikátumok az adatmezőt osztják föl két diszjunkt részre.

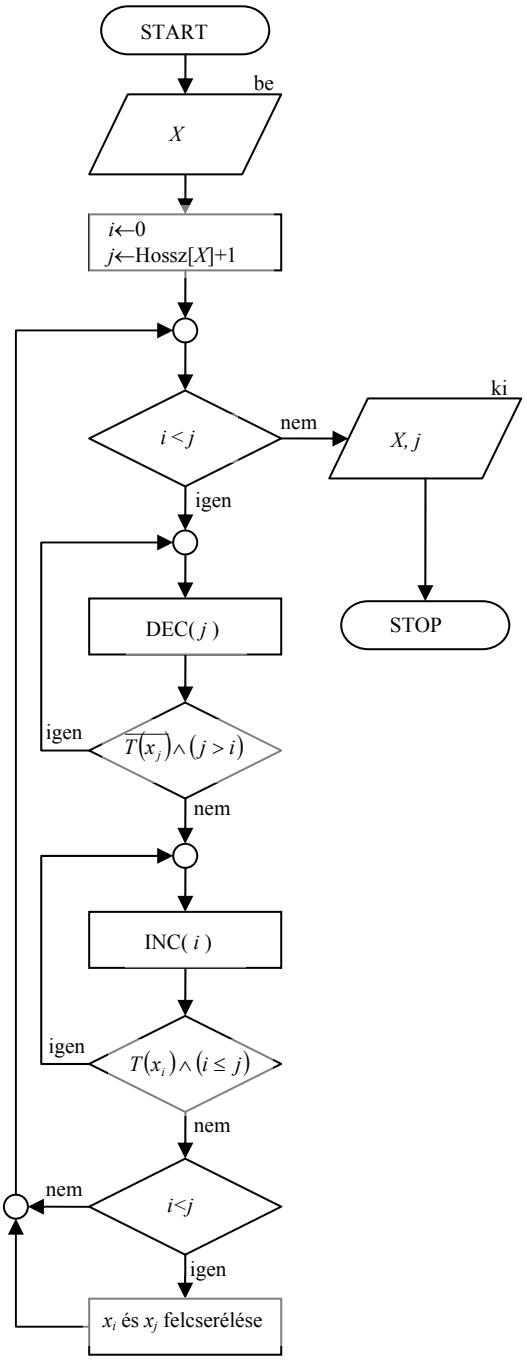
$$P_i(d) = \begin{cases} igaz, & ha\ d \in D_k \\ hamis & ha\ d \in D_l \end{cases} \qquad D = D_k \cup D_l \text{ és } D_k \cap D_l = \varnothing$$

A programgráf struktúrájának vizsgálatakor egy bemenő (START) és egy kimenő (STOP) szimbólumra korlátozódhatunk, ami az általánosságot nem csorbítja, a sok STOP-ot egyetlen közös STOP-ba torkoltatjuk. Az ábrázolások során a bemenő és a kimenő szimbólumot el is fogjuk hagyni, mivel ez a minket érdeklő következtetéseinket nem fogja befolyásolni. Az elágazásnak (predikátumnak) egy bemenő és két kimenő éle van, a gyűjtő szimbólumnak két bemenő és egy kimenő éle van, pontosabban az általánosság csorbítása nélkül ezt feltételezhetjük, megkövetelhetjük a programgráf ilyen alakban történő felrajzolását. A leírtakból következik, hogy a sokirányú elágazást több kétirányúval kell helyettesíteni.

A programgráf megadása, leírása *élhalmaz* segítségével a következő módon történik. A gráf minden csomópontja kap egy megnevezést, azonosítót. Kisebb méret esetén ez lehet egy betű például. Ezt követően felsoroljuk kapcsos zárójelek között a gráfot alkotó éleket, mint ahogy egy véges halmaz elemeit szokás felsorolni. Minden él egy csomópont párosból áll. A páros első tagja az él kezőpontja, a második tag az él végpontja. A párost zárójelbe tesszük és a páros két tagját vesszővel választjuk el egymástól. Ugyancsak vesszőt teszünk az egyes élek közé a felsorolásban. Elágazásból kiinduló élek végpontjánál egy / jellel elválasztva *i* vagy *h* betűvel jelezzük, hogy az igaz vagy a hamis ágról van-e szó. Az élek felsorolási sorrendje általában kötetlen, tetszőleges lehet. Ezt a precíz megadást, amely egy kicsit terjengős, tömörebbé, ugyanakkor talán egy kissé nehezebben követhetővé, de egyértelművé tudjuk tenni a következő módon. Ha csökkenteni akarjuk a zárójelek számát, akkor az élt megadó párost közbezáró zárójelpárt elhagyhatjuk, de ebben az esetben egy $\rightarrow$ nyilat teszünk a páros tagjait elválasztó vessző helyére, szimbolizálva az él irányítottságát. Tömör írásmódot is alkalmazhatunk, ha a gyűjtő csomópontokat kihagyjuk a leírásból a belőle kivezető élt hozzáfűzve a bemenő élhez. Tömörebben írható az elágazás is úgy, hogy a $\rightarrow$ jelet követően először az igaz ág élének végpontját, majd a hamis ág élének végpontját adjuk meg vesszővel elválasztva egymástól. Ebben az esetben az igaz és a hamis ág élvégeit zárójellel fogjuk egybe.

Példa: Legyen a feladat egy tömbben lévő elemek között a *T* tulajdonságúaknak a tömb elejére gyűjtése és a darabszámuk megállapítása, a *T* tulajdonsággal nem rendelkezőknek pedig a további, a tömb végén lévő helyekre gyűjtése. Az algoritmus (most egy teljes programot készítünk) lényege, hogy két indexet mozgatunk a tömbön. Az egyik (*i*) az első eleme elé mutat, a másik (*j*) az utolsó elem mögé kezdetben. Ezután felváltva mozgatjuk az indexeket mindaddig, amíg a *j* nem *T* tulajdonságúra mutat, az *i* pedig *T* tulajdonságúra. Amikor mindkettőnél megváltozik az elem minősége (rendelkezés a *T* tulajdonsággal, vagy nem), vagy pedig a két mutató eléri egymást, elmegy egymás mellett, akkor megállunk. Ha még nem érték volna el egymást, akkor a mutatott két elemet felcseréljük, ezáltal a *T* tulajdonságú felülre, a nem *T* tulajdonságú alulra kerül. Ha a mutatók már elérték, vagy elhagyták egymást, akkor befejeztük az áthelyezéseket és következik a végeredmények kiíratása.

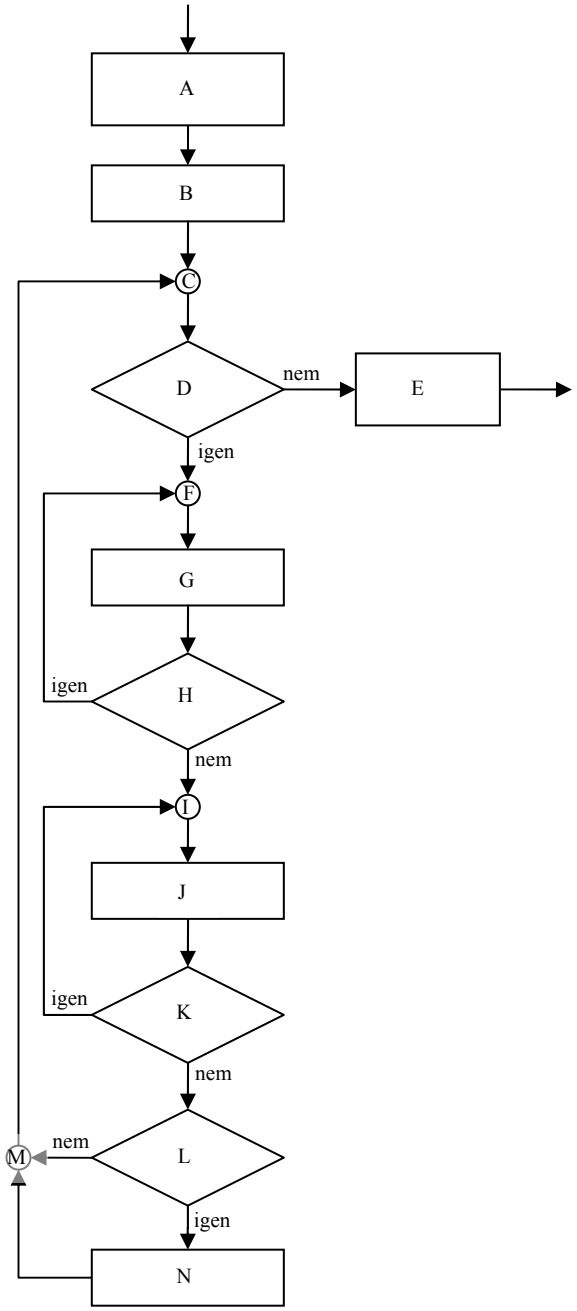
Alább megadjuk az algoritmus folyamatábráját és mellette egy, a folyamatábrának megfelelő pszeudokódot. A program neve legyen *P*.



PROGRAM *P*
VÁLTOZÓK $X \in \boldsymbol{R}^{20}$ $i,j \in \boldsymbol{N}$
INPUT (*X*)
$i \leftarrow 0$
$j \leftarrow$ Hossz[*X*]+1
WHILE $i < j$ **DO** **DO** **DEC**(*j*) **WHILE** $\overline{T(x_j)} \wedge (j > i)$ **DO** **INC**(*i*) **WHILE** $T(x_i) \wedge (i \le j)$ **IF** $i < j$ **THEN** x_i és x_j felcserélés
OUTPUT(*X*, *j*)
STOP

A folyamatábra és a pszeudokód után most elkészítjük a programgráfot, amely a folyamtábrára épül. A folyamatábráról leválasztjuk a START és a STOP szimbólumokat meghagyva a hozzájuk kapcsolódó folyamatvonalakat. (Ha több STOP lenne az ábrán, akkor azokat egyetlen közös STOP-ba futtatnánk össze előbb gyűjtő szimbólumok beiktatásával.) A folyamatábrában az input-output szimbólumokat téglalappal helyettesítjük és a szimbólumokban lévő szöveget is jelekre, azonosítókra (a példánkban az ábécé nagybetűire)

változtatjuk, mert most nem az a lényeges, hogy azok mit csinálnak, bár meg is hagyhatnánk őket. Ez azonban megnehezítené a szerkezet áttekintését. A programgráf mellett megadjuk a programgráf élhalmazos leírását teljes és rövidített, tömör formában is.



Teljes élhalmazos leírás	
P =	{ (START, A), (A, B), (B, C), (C, D), (D, E/h), (E, STOP), (D, F/i), (F, G), (G, H), (H, I/h), (H, F/i), (I, J), (J, K), (K, L/h), (K, I/i), (L, N/i), (L, M/h), (M, C) }

Tömör élhalmazos leírás	
P =	{ START → A, A → B, B → D, D → (G, E), E → STOP, G → H, H → (G, J), J → K, K → (J, L), L → (N, D)}

Láthatjuk, hogy a programgráf megrajzolása a teljes élhalmazos leírásból egyszerűen megvalósítható, de lehet, hogy az egyes csomópontok geometriailag másként helyezkednek majd el a papíron, mint eredetileg voltak, viszont a közöttük lévő kapcsolatok (élek) ugyanolyanok maradnak. A tömör élhalmazos leírásból már nehezebb a rajz elkészítése, mivel láthatóan a leírásból kimaradtak a gyűjtő csomópontok, tehát azokat rajzolás közben kell helyreállítani, amit lehet, hogy csak időben eltolva tudunk megtenni

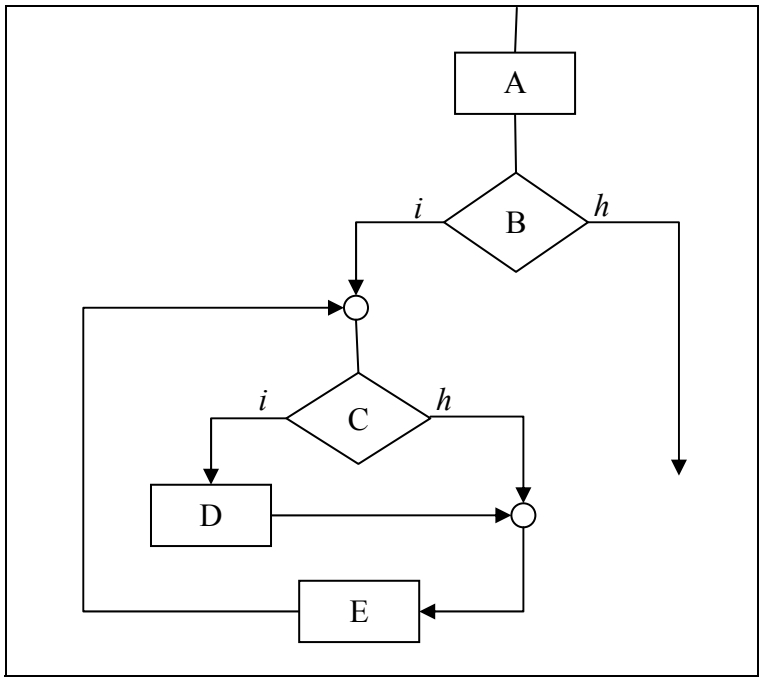
Definíció: **Valódi program**

Egy programot valódi programnak nevezünk, ha:

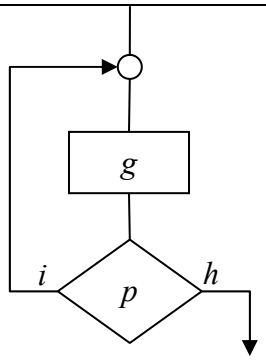
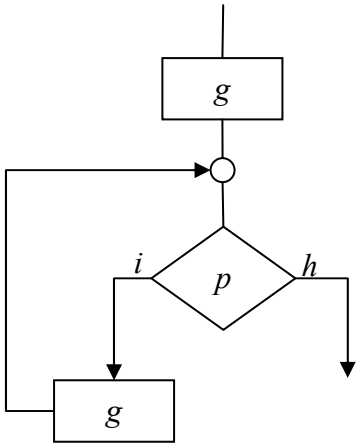
1. programgráfja véges számú nem zérus bemenő éllel és kimenő éllel rendelkezik,
2. programgráfjának csomópontjai predikátumok, függvények és gyüjtők,
3. programgráfjának minden csomópontján keresztül vezet legalább egy útvonal, amely egy bemenő éllel és egy kimenő éllel rendelkezik (a programgráf kezdő élétől a kilépési élig vezet).

A valódi programok halmaza megszámolható. Az egy élből álló programból kiindulva csomópontok egyenkénti hozzáadásával az összes valódi program előáll.

Példa nem valódi programra:

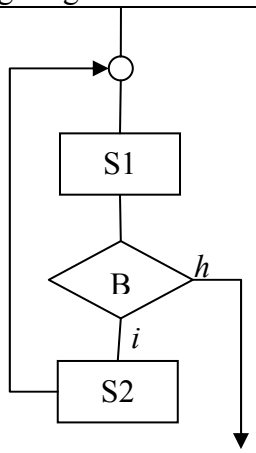
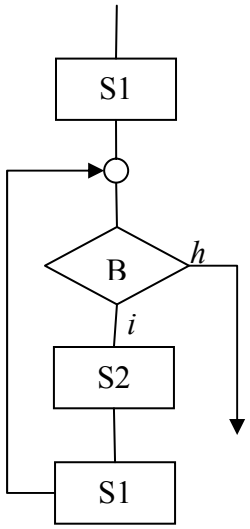


Kvázi-struktúrált szerkezet, az ismétlés: Formulája ***I***(*p*; *g*)

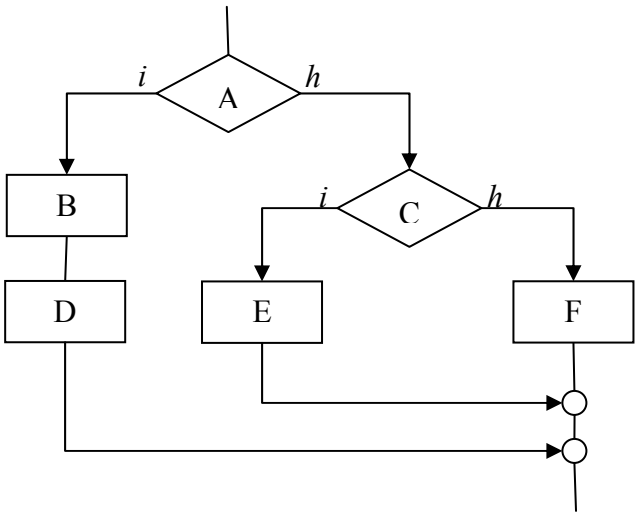
Kvázi-struktúrált ismétlés	Ekvivalens struktúrált programgráf
	
	Programgráf formulával: ***I***(*p*; *g*)=***S***(*g*, ***C***(*p*; *g*))

A későbbiekben az ismétlést is alapelemként fogjuk kezelni az egyszerűbb következtetések végett.

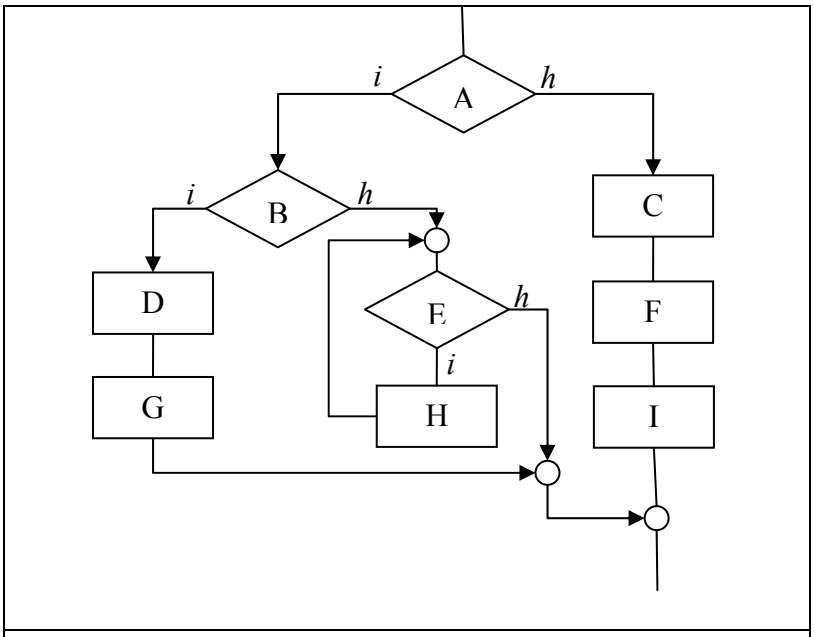
Példa nem struktúrált programrész átalakítására ekvivalens struktúrálttá:

Nem struktúrált programrész	Ekvivalens struktúrált programrész
K: S1 **IF** B **THEN** S2 **GOTO** K	S1 **WHILE** B **DO** S2 S1
Programgráffal:	
	
Programgráf élhalmazzal: {START→S1, S1→B, B→(S2, STOP), S2→S1 }	Programgráf formulával: ***S***(S1, ***C***(B; ***S***(S2, S1)))

Példák összetett struktúrára (több szintű struktúráltság).



E(A; ***S***(B, D), ***E***(C; E, F))



E(A; ***E***(B; ***S***(D, G), ***C***(E; H)), ***S***(C, F, I))

Kvázi struktúrált folyamatábrák: a **REPEAT**… **UNTIL** átírása struktúráltra.

Kvázi struktúrált programrész	Ekvivalens struktúrált programrész
REPEAT S **UNTIL** F	S **WHILE** $\overline{F}$ **DO** S

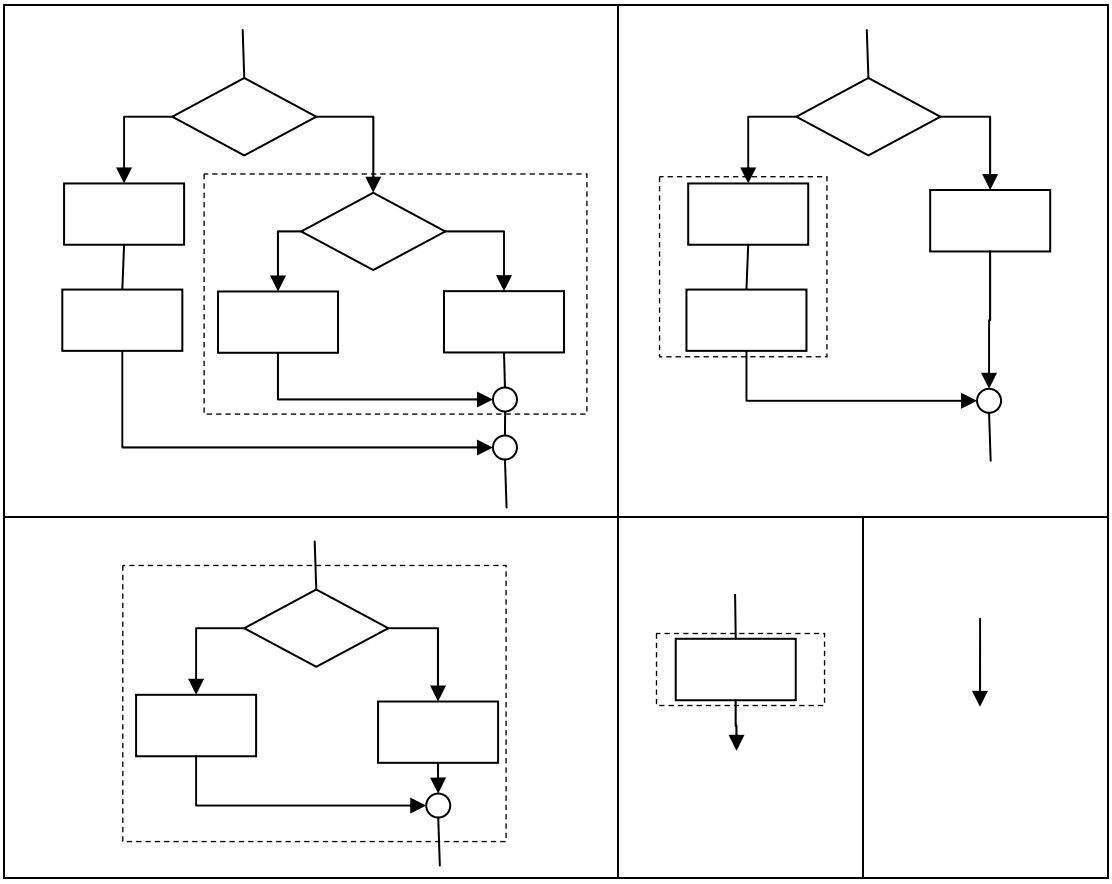
A **DO** … **WHILE** átírása struktúráltra.

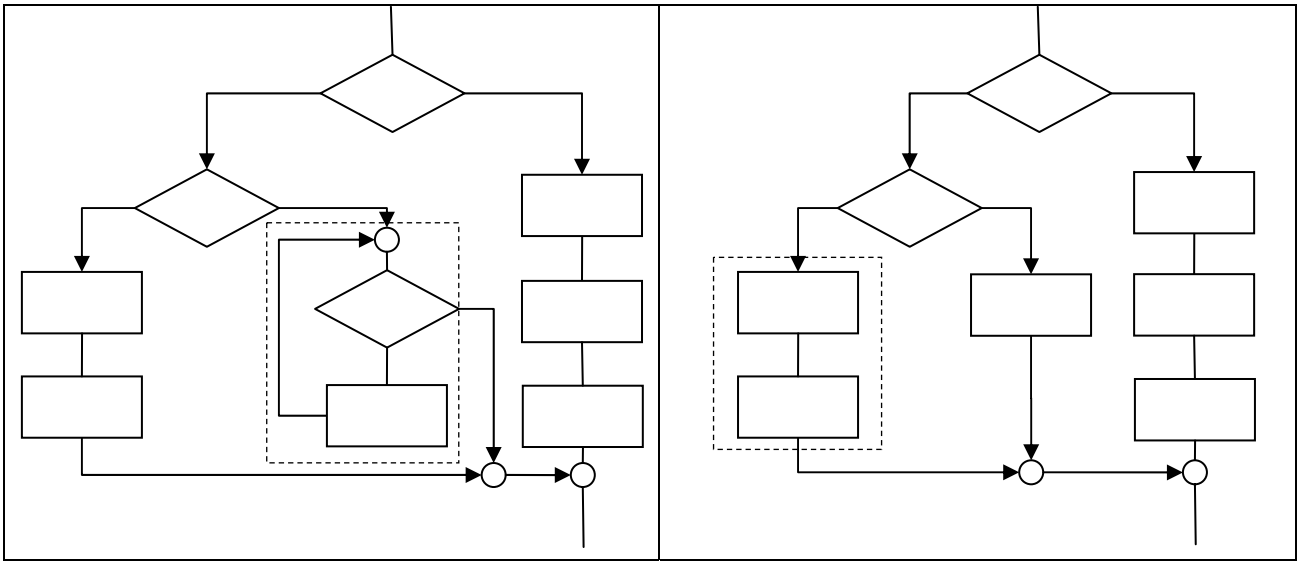
Kvázi struktúrált programrész	Ekvivalens struktúrált programrész
DO S **WHILE** F	S **WHILE** F **DO** S

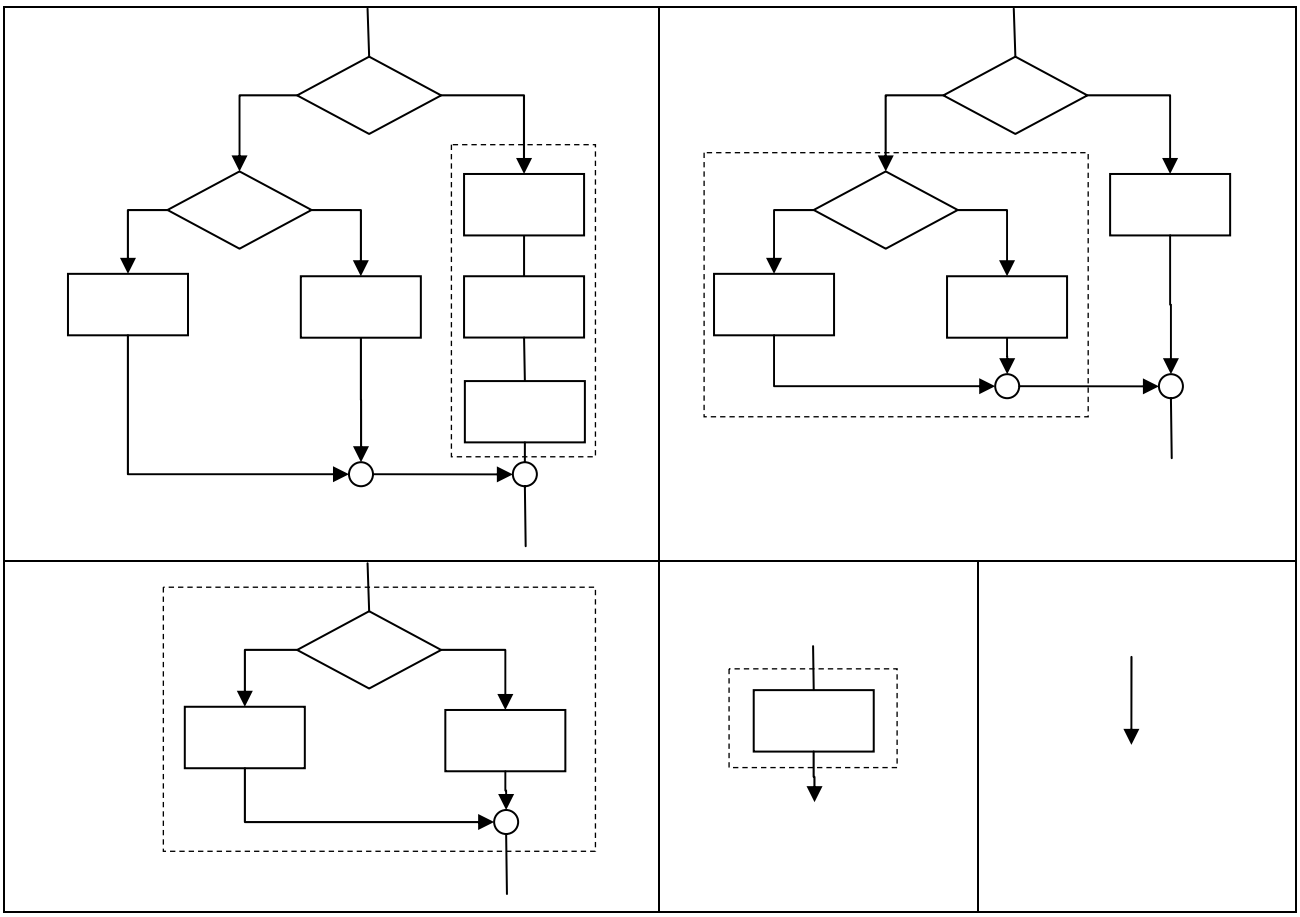
Definíció: **<u>A vezérlőgráf lebontása</u>**

A vezérlőgráf lebontásának nevezzük azt az eljárást, melynek során a struktúrált alapszerkezetek valamelyikét egy függvény csomóponttal helyettesítjük, és ezt mindaddig folytatjuk, amíg ez lehetséges. Az egyetlen csomópontból álló gráfot egyetlen éllel helyettesítjük. (Az egyetlen irányított él neve *üres program*.)

Példák lebontásra.



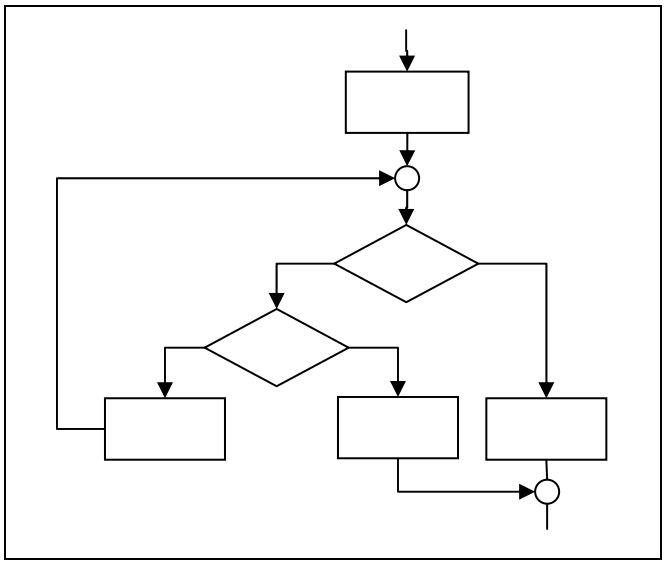




Definíció: **A struktúrált program**
Struktúrált programnak nevezzük azt a programot, amelynek vezérlőgráfja lebontható az önmagában álló irányított élre.

Mint látható, az előző két programgráf lebomlott az önmagában álló irányított élre, ezért a nekik megfelelő program struktúrált.

Példa valódi, nem struktúrált programra és lebontására.



A jelen példában egyetlen lebontási lépést sem tudunk végezni. A programgráf nem bontható le az egyetlen önálló irányított élre, tehát a program nem struktúrált. Nem minden eset ilyen

feszes, van, amikor néhány lebontási lépés megtétele után nem tudunk már tovább haladni még mielőtt elértük volna az egyetlen önálló irányított élt.

Tétel: **A struktúrált program formulája**
Egy *f* program akkor és csak akkor struktúrált program, ha formulája felírható az ***S***(*g*) vagy a ***C***(*p*;*g*) vagy az ***E***(*p*;*g*,*h*) alakban, ahol *p* predikátum, *g* és *h* pedig struktúrált programok (beleértve az *üres program*ot is, azaz melynek programgráfja az önmagában álló irányított él).

Példa struktúrált program formulájára. Megadjuk a tömbelemeket a *T* tulajdonság szerint a saját tömbjükben szétválogató programnak a formuláját, mivel az (kvázi)struktúrált program. (Csak a csomópontok neveit használjuk, a tartalmat nem írjuk ki, hogy rövidebb legyen a formula.)
f = ***S***(A, B, ***C***(D; ***S***(***I***(H; G), ***I***(K, J), ***E***(L, ∅))), E)

Tétel: **Bőhm – Jacopini tétel (a struktúrált programozás alaptétele)**
Minden valódi program megadható ekvivalens, struktúrált program formájában is.

A tétel csak az átalakíthatóságot állítja, a hogyanról nem nyilatkozik. A tételnek azonban van konstruktív bizonyítása, azaz olyan, amely algoritmust ad egy lehetséges ekvivalens struktúrált program konstruálására és bebizonyítja ezen konstrukció helyességét. Nemcsak egyféleképpen valósítható meg az ekvivalens átírás, mi egy konkrét algoritmust fogunk megadni a későbbiekben.

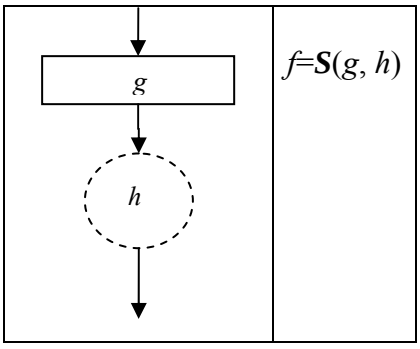
A struktúrált szerkezetek használata nem garancia egy program jóságára. Ezekkel is lehet hibás programot készíteni. (Lásd a New Haven-i esőátlag program esete.)

Módszer valódi program programgráfjának átalakítására struktúrált programgráffá

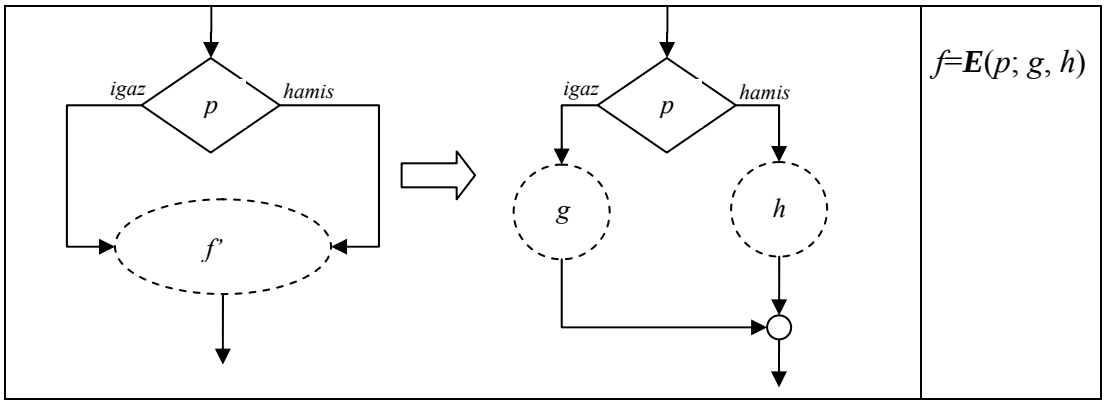
A programot függvénynek tekinthetjük, hiszen az *x* bemenetből az *y* kimenetet állítja elő a *D* megengedett inputok halmazán, ezért egy *y*=*f*(*x*) formulával szimbolizálhatjuk a program működését. A formulát a programgráf teszi szemléletessé, és ez a formula struktúrált esetben a korábban tárgyalt ***S***, ***C***, és ***E*** (vagy ***I***) formulákon keresztül fel is írható. Nem struktúrált esetben ez nem tehető meg ilyen egyszerű módon. Ezért fontos, hogy a struktúrált programgráf a rendelkezésünkre álljon.

A programgráf kezdőélén vizsgáljuk az első csomópontot és a típusának megfelelően redukáljuk azt és alakítjuk ki párhuzamosan a formuláját.

Függvény (szekvencia) csomópont esete:



Predikátum (elágazás) csomópont esete:

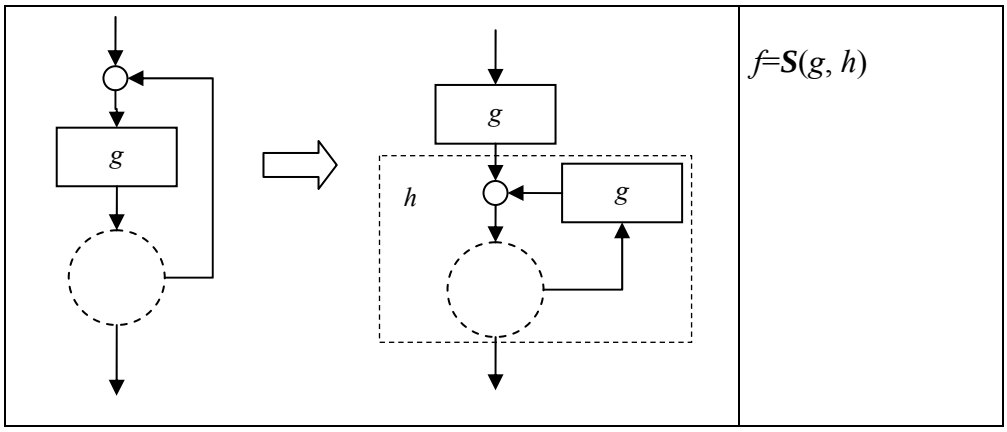


Az *f'*-ben a *p* predikátum *igaz* ágán haladó utakat *g*-be, a *hamis* ágán haladókat *h*-ba gyűjtjük. Ha szükséges, akkor a csomópontokat megkettőzzük. Azok a csomópontok lesznek megkettőzve, amelyeken mindkét út átmegy. Előfordulhat, hogy gyűjtő csomópontok a kettőzés után kiegyszerűsödnek.

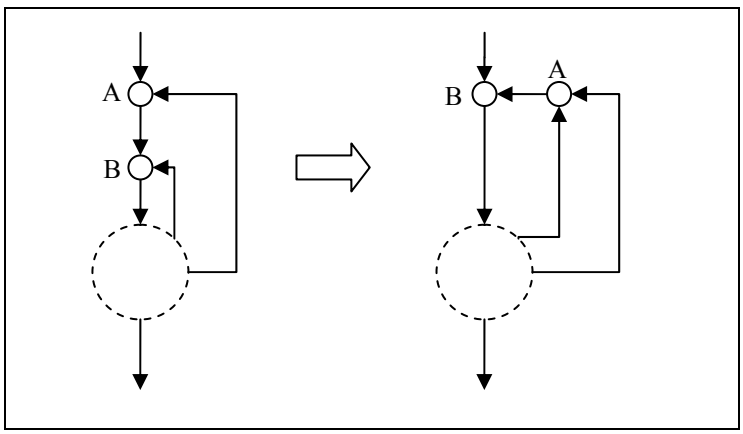
Gyűjtőpont csomópont esete:

Ez az eset három alesetre bomlik szét aszerint, hogy a vizsgált gyűjtő csomópontra rákövetkező csomópont milyen típusú. Ezek a következők.

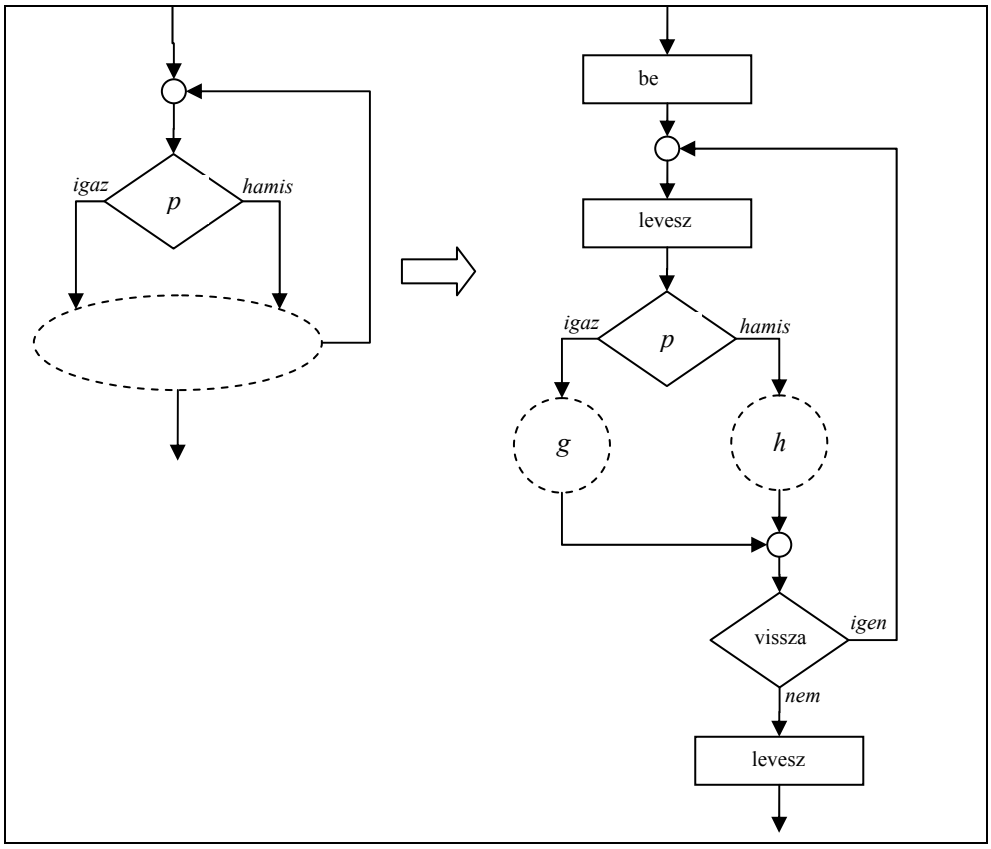
Gyűjtőpontot függvény követ aleset:



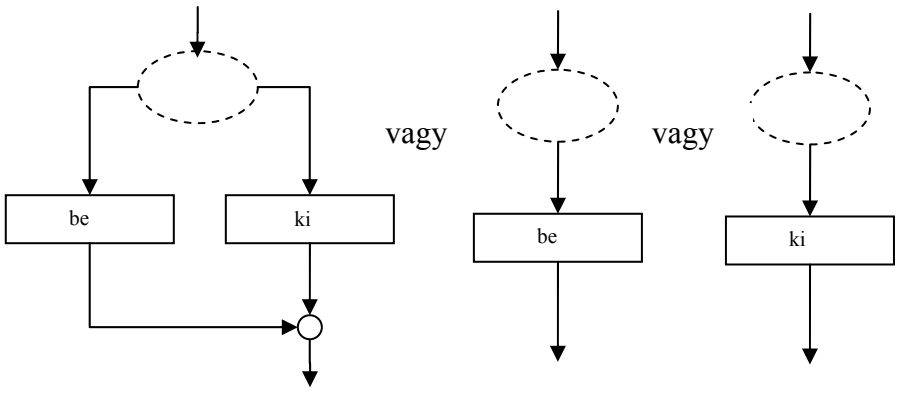
Gyűjtőpontot gyűjtőpont követ aleset:



Gyűjtőpontot predikátum követ aleset:

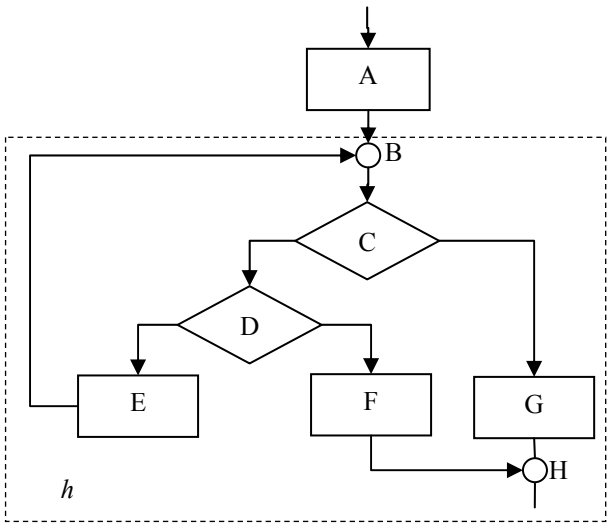


Ennél az esetnél a formula f=***S***(be, ***I***(vissza; ***S***(levesz, ***E***(p; g, h))), levesz).
A „be” függvény egy extra bitet definiál és beállít 1-re.
A „ki” függvény egy extra bitet definiál és beállít 0-ra.
A „levesz” függvény eltünteti, megszünteti az extra bitet.
A „vissza” predikátum az extra bitet vizsgálja. Ha az extra bit 1, akkor az *igen* ágon, ha 0, akkor a *nem* ágon lépünk ki belőle.
A g és h ágak szerkezete az alábbiak egyike.

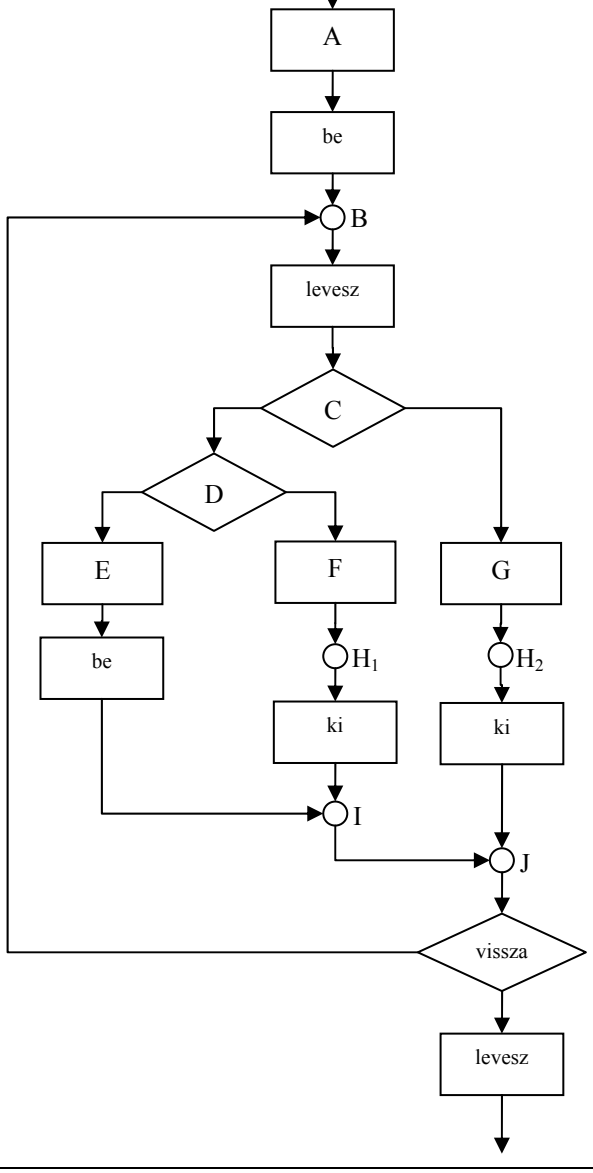


A g ágban a p igaz ágán a kimenő élhez (STOP) vezető (ki), vagy vissza a gyűjtőbe (be) vezető utak vannak.
A h ágban a p hamis ágán a kimenő élhez (STOP) vezető (ki), vagy vissza a gyűjtőbe (be) vezető utak vannak.

Példa nem struktúrált programgráf átalakítására struktúrálttá.



*Szekvencia esete f = **S**(A, h), ezután gyűjtőpontot predikátum követ.*

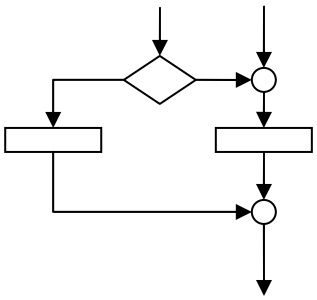
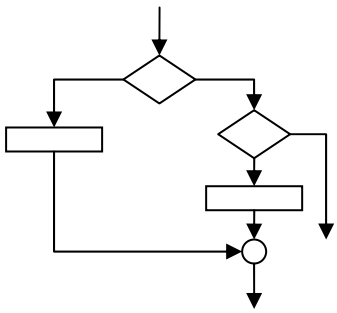


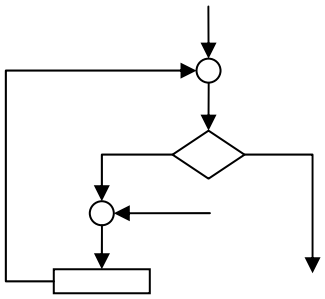
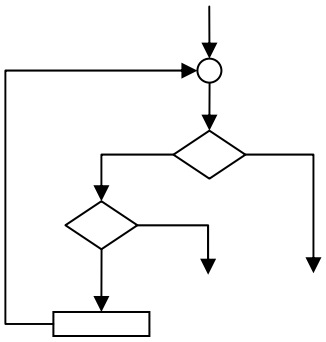
Az útvonalak bejárása során a H csomópontot meg kellett kettőzni, mivel több úton is áthaladtunk rajta. Ezek a megkettőzött gyűjtők azután feleslegesnek bizonyultak, mivel csak

egy bemenetük maradt jelen esetben. Ezért a programgráfból el is hagyhatók. A végső formula: f = ***S***(A, ***S***(be, ***I***(vissza; ***S***(levesz, ***E***(C; ***E***(D; ***S***(E,be), ***S***(F, ki)), ***S***(G, ki)))), levesz)).

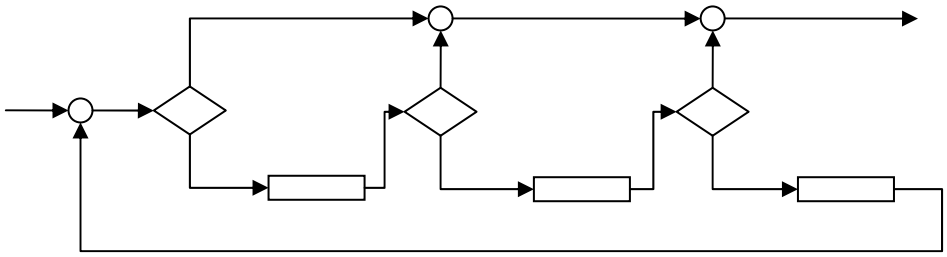
A nem struktúráltság négy alapformája

Egy program nem struktúráltságát megállapíthatjuk a programgráfja vizsgálatával azáltal, hogy jellegzetes, a struktúráltság elvét megsértő részgráfokat keresünk benne. Az alább vázolt négy eset ilyen részgráfokat ad meg.

Elágazás két bemenettel	Elágazás két kimenettel
	

Ciklus két bemenettel	Ciklus két kimenettel
	

Bőhm – Jacopini példája nem struktúrált programra (programgráf).



Melyik alapformákat találjuk meg ebben a példában és hányszor? A fenti négy alapforma a vázoltnál fontosabb, ugyanis kimondható a következő tétel.

Tétel: A nem struktúráltság karakterizálása
Egy program akkor és csak akkor nem struktúrált, ha annak programgráfjában részgráfként előfordul több ki-, illetve belépő éllel rendelkező ciklus, vagy elágazás.

Tétel: Alapformák a nem struktúrált programban

A nem struktúrált program a négy nem struktúráltsági alapforma közül legalább kettőt mindig tartalmaz.

Definíció: <u>A program modul</u>

Modulnak nevezzük az egy file-ban lévő, egyszerre fordítandó programrészt. Ez általában egy vagy több procedúra vagy a főprogram.

További elvek a struktúrált programok írásához:

a. használjunk „beszélő" változóneveket!
b. Névvel ellátott konstansokat használjunk, melyek definíciója a program meghatározott helyén (általában az elején) legyen!
c. Kerüljük a **GOTO** használatát és sohase írjunk „spagetti" kódot! (A **GOTO** egy négybetűs szó! = The **GOTO** is a four-letter word!)
d. Moduláris programokat írjunk, így a részek kölcsönhatása tiszta és egyszerű.
e. A modul részei felfoghatók legyenek és sorban olvashatók.
f. A modulok egyike se tudjon túl sokat az egész program hierarchiájából. Csak néhány szomszédjával tartsa a kapcsolatot. Ez csökkenti a veszélyt, hogy egy változás sok részre kihasson.
g. A modulok amennyire lehet, legyenek egymástól függetlenek.
h. Rövid procedúrákat írjunk, amelyek csak egy dolgot valósítanak meg, de azt jól!
i. A procedúra lehetőleg ne legyen egy oldalnál hosszabb!
j. Minden procedúra feladata egy-két sorban leírható legyen.
k. A procedúra úgy készüljön, hogy a specifikációját (a feladat leírását) kielégítse, azaz először a specifikáció jön, majd a procedúra írása következik és nem fordítva!
l. Használjuk ki a bekezdések eltolhatóságának lehetőségét a programszövegben!
m. Minél több értelmes kommentárt készítsünk, minél teljesebb dokumentációt írjunk!
n. Túl sok IF-et ne skatulyázzunk egymásba!
o. A tervezési döntéseket és az adatszerkezeteket fedjük el. Amelyik procedúra nem használja valamelyiket, az ne is ismerje, ne is lássa.
p. A program legyen minél egyszerűbb. Vegyük figyelembe a gazdaságossági és szabványosítási szempontokat. A fejlesztés több lépésben történjen: először a központi részeket fejlesszük, a részletekkel ne foglalkozzunk.
q. Hagyjuk, hogy a feladat nehezét más oldja meg. Ez azt jelenti, hogy építsünk a saját magunk, vagy más által elkészített munkákra, procedúrakönyvtárakra.

A **GOTO** polémia

1. Túl sok célra használható (mert túl egyszerű) és a környezetében nem derül ki, hogy mi a célunk vele.
2. Ha egy címkét látunk egy programszövegben, nem tudjuk, hogy honnan lehet odaugrani.
3. A programszöveg olvasása során gyakran kell összevissza ugrálnunk a szövegben, ami nehézkessé teszi a szöveg olvasását, nehezen követhetővé a vezérlést. Struktúrált esetben lefelé és balról jobbra olvasunk.
4. Struktúrált programnak könnyebb a programhelyesség vizsgálata.

Kvázi struktúrált folyamatábrák: ciklusból idő előtti kiugrás
Be lehet vezetni egy **EXIT** utasítást (a **GOTO** helyett!), amelynek a szerepe: kilépés a struktúrából a struktúra végén, az egyetlen kimeneten.

Az *a* elem keresése az *X* tömbben.

Gyorsabb, de zavarosabb „gépi” megoldás

1.	**FOR**	*i*←1	**TO** Hossz[*X*] **DO**
2.		IF	x_i=*a*
3.			**THEN GOTO** 6
4.	// nincs meg		
5.	**GOTO** 7		
6.	// megvan		
7.	…		

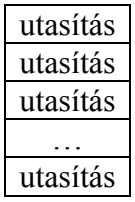
Lassabb, de áttekinthető „emberi” megoldás

1.	*i*←1	
2.	**WHILE**	*i*≤Hossz[*X*] és x_i≠*a* **DO**
3.		**INC**(*i*)
4.	**IF**	*i*> Hossz[*X*]
5.		**THEN** … // nincs meg
6.		**ELSE** … // megvan

Nassi – Shneidermann ábrák, a struktogram

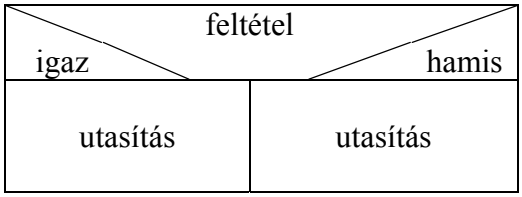
A struktúrált programok vezérlési szerkezetének szemléltetésére jól használhatók az úgynevezett *Nassi – Shneidermann ábrák*, vagy népszerű (németből átvett) nevén a *struktogram*ok. Isaak Nassi és Ben Shneidermann fejlesztette ki 1972-ben. A struktogramok a folyamatábra szabadosságát korlátozzák és csak (kvázi)struktúrált szimbólumokat tartalmaznak. Ebből következik, hogy csak (kvázi)struktúrált programok szerkezete ábrázolható a segítségükkel. Alapelem a téglalap, amit különböző módokon osztunk fel részekre. Maga a program egyetlen partícionált (felosztott) téglalap. A következő lehetőségek adottak.

Szekvencia



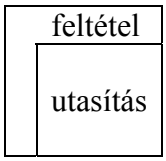
Elágazás (kétfelé)

Ha a feltétel igaz, akkor az igaz alatti, ha hamis, akkor a hamis alatti utasításnál folytatjuk lefelé. Ha valamely ág üres, akkor az utasítás helyét kiikszeljük.



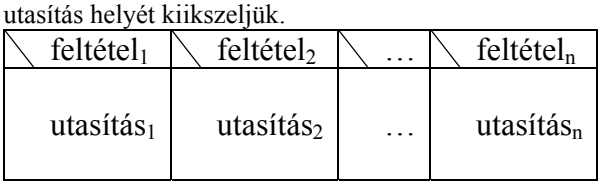
Ciklus (elöltesztelős)

Ismétlés, amíg a feltétel igaz.



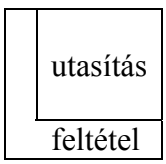
Elágazás (sokfelé)

Egy feltételnek mindig teljesülni, kell és egyidejűleg csak egy teljesülhet. Ha valamely ág üres, akkor az utasítás helyét kiikszeljük.



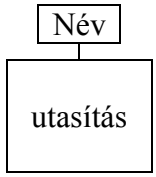
Ciklus (hátultesztelős)

Ismétlés, amíg a feltétel igaz.



Eljárásdefiníció

A lokális változókat és a paramétereket a név mellett soroljuk fel a névhez hozzákapcsolva.



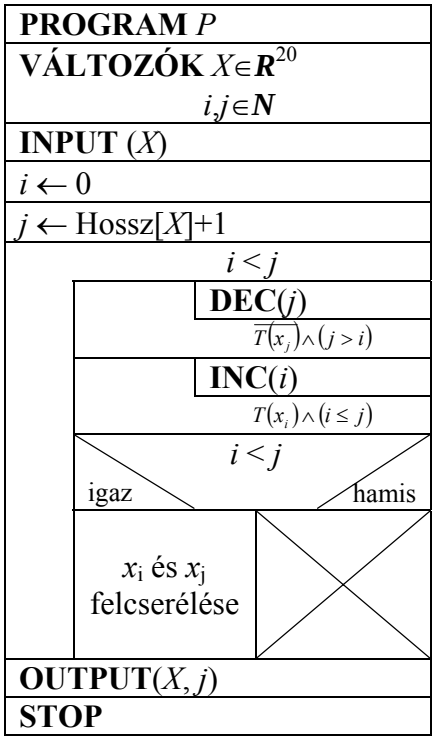
A struktogram előnyei.

- Az struktogram tömörebb, mint a folyamatábra, főként a folyamatvonalak kiküszöbölése miatt.
- Garantált a program struktúráltsága, mivel más szimbólumai nincsenek.
- Használható *a lépésenkénti finomítás módszerével* történő programfejlesztésre.

A struktogram hátrányai.

- Nehezebben követhető, mint a folyamatábra.
- A rajz miatt terjedelmes, viszont a korlátozott szélesség miatt nehezen javítható. Gyakorlatilag újra kell rajzolni.
- Nem ad lehetőséget nem struktúrált utasításokra.

Struktogramra példaként a fentebb tárgyalt programot adjuk meg, amelyben egy tömb elemeit rendezzük át úgy, hogy a T tulajdonságúak a tömb elején, a többiek a végén helyezkedjenek el. Fentebb megadtuk a folyamatábrát és a pszeudokódot, most alább megadjuk a struktogramot.



FELADATOK

1. Tömör élhalmazzal adottak alább programgráfok. Rajzoljuk meg a programgráfot! Lássuk be, hogy nem struktúrált, vagy pedig, hogy az! Írjuk át struktúrálttá, ha nem az! Írjuk fel ezután a formuláját! Rajzoljuk fel a struktogramot!

   a. {START→A, A→(D, B), B→(C, E), C→A, D→A, E→STOP}
   b. {START→A, A→(B, C), B→C, C→(B, STOP)}
   c. {START→A, A→(B, C), B→(D, E), C→STOP, D→A, E→STOP}
   d. {START→A, A→(B, C), B→F, C→(D, E), D→F, E→(G, F), F→E, G→(H, STOP), H→A}
   e. {START→A, A→(C, B), B→(E, D), C→(F, E), D→A, E→G, F→G, G→(H, STOP), H→F}
   f. {START→A, A→(B, C), B→STOP, C→(STOP, D), D→A}
   g. {START→A, A→(B, C), B→C, C→(D, E), E→STOP, D→A}
   h. {START→A, A→(B, STOP), B→C, C→(A, STOP)}
   i. {START→A, A→(C, B), B→STOP, C→(D, STOP), D→A}
   j. Legyen a programgráf a fent említett Bőhm – Jacopini példa. (Adjuk meg az élhalmazt!)
   k. Legyen a programgráf a nem struktúráltság négy alapformája. (Adjuk meg az élhalmazt! Két bemenet esetén azokat a START-ból egy predikátum két kimenete adja, két kimenet esetén azokat futtassuk egybe egy STOP-ba!)
   l.

2. a. Készítsünk programot egy 1 és 1000 közötti gondolt szám kitalálására. (A program próbálkozzon kitalálni az általunk gondolt számot, lehetőleg minél kevesebb lépéssel.) A program tegyen fel kérdéseket, amelyekre mi csak az igen vagy nem választ adhatjuk. Igyekezzünk struktúrált programot készíteni. Ha nem az sikeredne, alakítsuk át a tanult módszerrel struktúrálttá! Írjuk fel ezután a formuláját! Rajzoljuk fel a struktogramot!

   b. Módosítsuk az előző programot úgy, hogy feltételezzük, hogy a válaszadó esetleg csalhat. Ekkor a végén rákérdezünk a gondolt számra, és ha a válasz az, hogy a gondolt szám nem a megtalált, akkor bekérjük a válaszoló szerinti helyes számot és rámutatunk, hogy hol csalt először.

3. Adott az alábbi formális pszeudokód. Rajzoljuk meg a programgráfot! Írjuk át struktúráltra, ha nem az! Írjuk fel ezután a formuláját! Rajzoljuk fel a struktogramot!

a.
```
1.  IF   A
2.       THEN   B
3.              GO TO 8
4.       ELSE   IF   C
5.                   THEN   GO TO 8
6.                   ELSE   D
7.                          GOTO 1
8.  STOP
```

b.
```
1.  A
2.  IF   B
3.       THEN   C
4.              GO TO 1
5.  STOP
```

c.
```
1.  WHILE   B   DO
2.              A
3.              IF   C
4.                   THEN   D
5.                   ELSE   E
6.  STOP
```

d.
```
1.  A
2.  IF   B
3.       THEN
4.              IF   D
5.                   THEN   F
6.                          GOTO 4
7.       ELSE   D
8.              E
9.  G
10. STOP
```