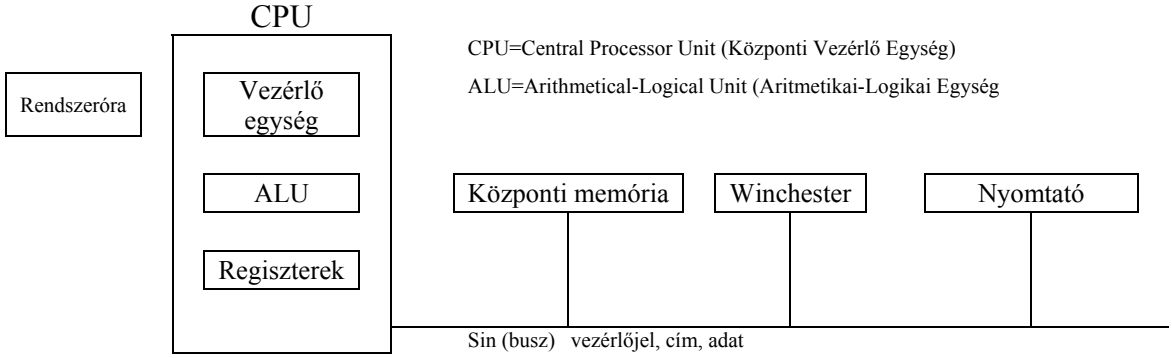
A számítógép programozásáról

A számítógép felépítésének elvi szerkezeti vázlata

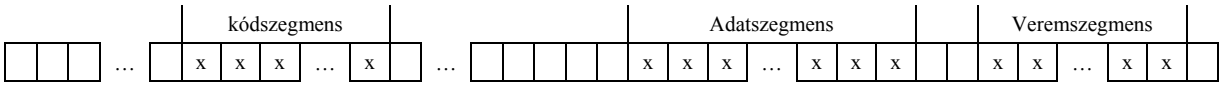


A memória

A számítógép központi memóriája tulajdonképpen byte-ok sorozata. A byte-ok elhelyezkedése logikailag a meóriában lineáris, azaz egymást követik. A byte-ok azonosítása a címükkel történik. Minden byte-nak van egy rá és csak rá vonatkozó fizikai címe, ami nem más, mint a byte sorszáma. A memória kezdőbyte-jának a sorszáma (címe) zérus és a továbbiaké egyesével növekszik, ahogy haladunk tovább a memóriában. Egy *b* byte címének jelölésére a *@b*, vagy *@(b)* jelölést fogjuk alkalmazni. Nem témánk annak tárgyalása, hogy ezt a címet fizikailag hogyan valósítják meg, milyen összetevőkből alakul ki. A memória méretét, nagyságát, *memória-kapacitás*át a byte mellett kényelmesebb nagyobb egységben megadni. Ilyenek a kilobyte (Kb), 1Kb=1024byte, a megabyte (Mb), 1Mb=1024Kb, a gigabyte (Gb), 1 Gb=1024Mb, a terrabyte (Tb), 1Tb=1024Gb. A mai PC-ken 32-bites címmel számolva (32 eres címvezeték) 4 gigabyte méretű memórák is lehetnek, amely kb. 4 milliárd byte-ot jelent. A byte mellett fontos fogalom a *rekesz*, amely fogalom azonban már absztrakció következménye, mivel a memória fizikailag nem különböztet meg rekeszeket. A rekesz egymást követő byte-ok egy véges sorozata, amelynek méretét, bitjeinek, byte-jainak belső szerkezetét mi írjuk elő akkor, amikor a benne tárolt adat típusát megadjuk. Mivel a rekesz a memória egy logikai egysége, ezért van címe, ezzel hivatkozunk rá, és ez a rekeszt alkotó byte-ok közül a legelső (a legalacsonyabb című), a kezdőbyte címével egyezik meg. Például egy négybyte-os egész számot tartalmazó rekesz címe a négy byte közül a kezdőbyte címe lesz. A magasszintű programozási nyelvekben nekünk nem kell a címekkel törődnünk, mivel a rekeszeknek nevet adunk és majd a program fordítása során a név címmel helyettesítődik.

A memória egy passzív tároló. Csak arra képes, hogy a beleírt információt, biteket, byte-okat tárolja, és igény esetén engedje kiolvasni, vagy átírni, megváltoztatni. A kiolvasás nem változtatja meg a memóriát, az ott tárolt információ megmarad. A beíráskor az előző tartalom elvész és az új helyettesíti. A memória minden byte-jában a gép üzemelése során mindig van valami, legfeljebb számunkra érdektelen. Még akkor is van a byte-oknak tartalma, ha mi oda nem tettünk semmit. Ekkor az előző programok, vagy az operációs rendszer által otthagyott információt találjuk ott, de hogy az mi, nem tudhatjuk. A memória azért fontos, mert egy feladat megoldása közben a megoldás lebonyolítására készített program is, és a műveletek végzéséhez szükséges adatok is legalább a feldolgozás egy szakasza alatt a memóriában vannak. Egy program logikailag három fő részből áll: a kódterület, az adatterület és a veremterület. (Fizikailag a programterület is és az adatterület is állhat több területdarabból.)

Ezen területeket *szegmens*eknek nevezzük. A *kódszegmens*ben a feladat megoldásához alkotott parancsok, utasítások állnak, amelyek a processzornak szólnak. A processzor ezeket olvassa egymás után és hajtja őket végre, tehát a processzor az utasításokat a memóriából veszi. Az *adatszegmens*ben helyezkednek el az adatok, amelyeken az utasítások végrehajtják az előírt műveleteket. A *veremszegmens*, amely szintén a memóriában helyezkedik el, hasznosan használható fel a hatékony programvégrehajtáshoz. A veremben átmenetileg tárolunk fontos információkat, vagy átmeneti időre segédtárolóként használjuk. Szerepét később világítjuk meg. A program szegmensei a memóriában nem feltétlen egymás mellett helyezkednek el. Sematikusan ábrázolva:



A számítógép lelke a *Rendszeróra* (System clock), amely megadott ütemben (frekvenciával) impulzusokat bocsát ki. Ezek az impulzusok vezérlik a kapuáramköröket és ezáltal az processzor munkáját az utasítások végrehajtására. Az utasítás végrehajtása mindig óraütésre indul, de lehet, hogy több óraütés idejéig tart. A CPU fontos egysége az *Aritmetikai-Logikai Egység*, amely azokat az aritmetikai, logikai műveleteket hajtja végre, amelyekkel a *Vezérlő egység* megbízza. A műveletek elvégzéséhez szükség van a *Regiszterblokkra*, amely a memóriához hasonló szervezésű rekeszeket tartalmaz, de a memóriától elkülönülten, más (jóval drágább) fizikai felépítéssel, és ami lényeges, lényegesen gyorsabb elérhetőséggel, mint a memória. Mérete kicsi, néhány (tucat) tárolóregiszterből áll. Ezek egy részét a programutasítások (azaz a program készítője) elérheti, míg más részüket nem, hanem a hardver kezeli az utasításoknak megfelelően. A legfőbb regiszterek az alábbiak szoktak lenni.

- IP regiszter (Instruction Pointer, utasításmutató). A processzor számára mutatja a következő végrehajtandó utasítás memóriabeli címét.
- IR regiszter (Instruction Register, utasítás regiszter). A memóriából kiolvasott utasítást a processzor itt tárolja és elemzi.
- Akkumulátor regiszter. Az aritmetikai-logikai műveletek eredményét tárolja (összeadás, kivonás, szorzás, osztás, VAGY, ÉS, NEM, KIZÁRÓ VAGY, eltolások, bitműveletek stb.)
- Bázisregiszter. Valamely memóriabyte címét tárolja. Rajta keresztül is elérhetjük a memóriabyte-ot.
- Indexregiszter. Egy bytecímhez képesti eltolódást ad meg.
- Szegmensregiszterek. Az egyes szegmensek (program, adat, verem) kezdőcímét adják meg.
- Bázis Pointer regiszter. A veremben történő adatelérést könnyíti meg.
- SP regiszter (Stack Pointer). A verem tetejének a címét tartalmazza.
- FR Flag regiszter. Jelzőbitek regisztere. Minden bitjének van valamilyen szerepe, jelez valamit. Az utasítások és a műveletek beállíthatják ezen bitek értékét és ezáltal mód nyílik a program utasításainak a végrehajtásában eltérni a sorrendtől (elágaztatni) a jelzőbít (flag) értékétől függően (ugró utasítások). Egyes bitek a számítógép üzemmódját határozzák meg.

A processzor a rá jellemző, az áramköreibe beépített utasításokat tudja csak végrehajtani, ezeket gépi kódú utasításoknak nevezzük. A gépi kódú utasítások típusai: általában regiszter-regiszter, regiszter-memória, vagy memória-memória. Ez a besorolás azt jellemzi, hogy a műveletek milyen kiinduló és célrekeszek között zajlanak. A memória-memória típus nagyon ritka, mivel lassú (kétszer is ki kell nyúlni a memóriába az utasítás végrehajtása közben).

Az utasításokat a műveletek szerint is csoportosíthatjuk. Így beszélhetünk
- adatmozgató utasításokról,
- egyszerű aritmetikai utasításokról,
- logikai utasításokról,
- shiftelő és rotáló (eltoló és forgató) utasításokról,
- összehasonlító utasításokról,
- feltétel nélküli és feltételes ugró utasításokról,
- sztringkezelő utasításokról,
- veremkezelő utasításokról,
- procedúrakezelő (hívó és visszatérő) utasításokról,
- megszakításkezelő utasításokról,
- koprocesszor utasításokról, (társprocesszor, amely főként a lebegőpontos műveletek elvégzésére készült, ma már a processzorral egy tokban helyezkedik el),
- Fizikai szintű I/O (Input/Output, Bemeneti/Kimeneti) utasításokról, stb.

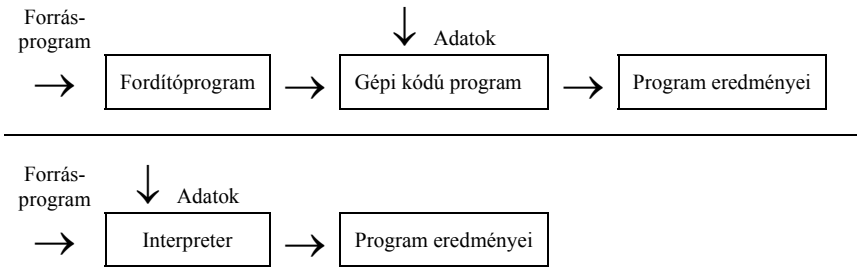
A processzor működési sémája

A processzor megadott lépések sorát végzi ciklikusan, míg csak ki nem kapcsolják a gépet. Ezek a lépések sematikusan:

1. Utasítás kiolvasása a memóriából arról a címről, amit az utasításmutató regiszter (IP) mutat.
2. Az utasításmutató regiszter (IP) módosítása az utasítás hossza alapján (növelés).
3. Az utasítás dekódolása, értelmezése, mit kell csinálnia a processzornak, és honnan vegye hozzá az adatokat (adatok címei).
4. Az utasítás végrehajtása.

Egy ilyen ciklus után az egész kezdődik elölről.

A számítógépes programozás területéről több fogalomra lesz szükségünk annak ellenére, hogy igazán egyetlen programozási nyelv mellett sem kötelezzük el magunkat. A számításaink, adatokon végzett tevékenységeink elvégzéséhez gépi utasítások, parancsok rögzített sorozatára lesz szükségünk. Ezeket összefogva *program*nak fogjuk nevezni. A programot valamilyen magas szintű programozási nyelven (az ember gondolkodásmódjához közel álló nyelven) írjuk meg, majd azt a gép nyelvére egy fordítóprogram (*compiler*) segítségével fordítjuk le (remélhetően jól). Ezt a lefordított, úgynevezett gépi kódban lévő programot fogja a processzor végrehajtani. Ha van *interpreter* programunk, akkor azzal is megoldható a feladatvégzésnek a gépre történő átvitele. Ebben az esetben az interpreter program (ami maga gépi kódban van) fogja olvasni a mi programunk szövegét, értelmezni és végre is hajtani az ott leírtakat. Nyilvánvaló, hogy ez az utóbbi lelassítja a programunk végrehajtását, hiszen újra és újra értelmezni kell az utasításokat. A kétféle munkamenetet az alábbi ábra szemlélteti.

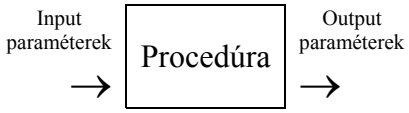


Egy program felépítése

- Főprogram,
- eljárások, procedúrák
- - saját
- - könyvtári (beépített, vagy külső)
- - operációs rendszer szolgáltatásai

Egy program indításakor a főprogram indul el a logikailag kezdő utasításával, működés közben pedig procedúrák is meghívódhatnak általa, vagy akár ezek egymást is hívogathatják.

Definíció: **Procedúra**
A procedúra logikailag önálló, zárt programegység, amely egy specializált részfeladatot old meg. A procedúra egy aktivizáló (hívó) utasítás hatására kezd el működni. Addig működik, amíg egy befejező utasításhoz nem ér, amikor a hívás helyét követő helyre (a hívó utasítást követő utasításra) visszaadja a vezérlést. A procedúra a többi eljárással, a főprogrammal a *softvare interface*-en (*programozói csatoló felület*en) keresztül tartja a kapcsolatot. Ez a gyakorlatban általában paraméterlistát és/vagy globális változó(ka)t jelent.



Definíció: **Globális változó**
Olyan adatrekesz, amely a program futása, munkája alatt mindvégig létezik, és a programból mindenhonnan (főprogram, procedúrák) látható, elérhető.

Definíció: **Lokális változó**
Olyan adatrekesz, amely csak az adott programegységben (procedúra) létezik, míg az aktív. Máshonnan nem látható, nem érhető el. Megszűnik, mihelyst a procedúra befejezi a minkáját. Általában a veremben található.

Definíció: **Dinamikus változó**
Olyan adatrekesz, amely a program futása közben jön létre (a programozó általi) extra memória igénylésével és futás közben meg is szüntethető az általa lefoglalt memória felszabadításával. A heap (kupac) területen helyezkedik el, az adat- és veremszegmensen kívül.

Procedúrák paraméterlistája szerepel a procedúra leírásakor (deklarációjakor), és a procedúra hívási utasításában is. A deklarációban szereplő neve *formális paraméterlista*, a hívásban szereplőé *aktuális paraméterlista*. A paraméterlistát jellemzi a benne felsorolt elemek *darabszáma*, *típusa* és *sorrendje*. A listaelemek lehetnek értékük szerint megadva – *érték szerinti hívás* – és lehetnek cím szerint megadva – *cím szerinti hívás*. A jelentésük benne van a neveikben.

Az *érték szerinti paraméter átadás*kor a procedúra értéket kap, amivel ugyan számolni tud, de nem tudja, hogy az érték honnan jött, ezért ha meg is változtatja, az eredeti helyén az adat változatlan marad, mivel rejtve van, a változás csak a másolatként átadott értéken történik meg, ami a visszatérés során elvész.

A *cím szerint paraméter átadás*kor az adatot tartalmazó memóriarekesz címét adjuk át, ami alapján a procedúra megtalálja az adatot a memóriában, számol vele és akár az erdeti helyén a cím birtokában meg is változtathatja azt.

A procedúrahívás (eljáráshívás) mechanizmusa

Hívó programegység végzi	1. Paraméterek lerakása a verembe. 2. Visszatérési cím lerakása a verembe és elugrás a procedúra kezdő utasítására.
Hívott programegység (procedúra) végzi	3. Ha kell, a lokális változók számára helyfoglalás a veremben. 4. Számolás, az eljárásra bízott feladat elvégzése. 5. Lokális változók törlése a veremből. 6. Visszatérési cím ürítése a veremből és visszatérés a visszatérési cím szerinti helyen lévő utasításhoz.
Hívó programegység végzi	7. Paraméterek ürítése a veremből.

A programok általában *procedúrák* (eljárások) sokaságát tartalmazzák. Ezek a zárt programegységek egy-egy kisebb feladat elvégzésére specializáltak. A program többi részével csak a paramétereik révén tartják a kapcsolatot. *Fekete doboz*nak kell őket tekintenünk, amin azt értjük, hogy nem ismerjük a belső szerkezetüket. A dobozra „rá van írva”, hogy miből mit csinál. Vannak (lehetnek) bemenő (*input*) és vannak (lehetnek) kimenő (*output*) paraméterei. A bemenetet alakítják át a kimenetté. Ha ismerjük a procedúra belső szerkezetét, — mert mondjuk mi készítettük —, akkor *fehér doboz* a neve, ha nem ismerjük, — mert nem vagyunk kíváncsiak rá, vagy másoktól kaptuk, — akkor fekete doboz szerkezet a neve. Például készíthetünk olyan procedúrát, amely bekéri (input) az *a, b, c* három valós számot, melyeket egy ax^2+bx+c kifejezés (itt x valós szám, változó) konstans együtthatóinak tekint, majd eredményül (output) meghatározza a kifejezés valós gyökeinek a számát és ha van(nak) gyök(ök) , akkor az(oka)t is megadja. Példa egy lehetséges másik procedúrára: egy file nevének ismeretében a procedúra a file rekordjait valamilyen szempont szerint megfelelő sorrendbe rakja (rendezi), például neveket ábécé szerinti sorrendbe rak. A procedúrák által használt memóriarekeszek — a paramétereket és a globálisakat kivéve — a zártságnak köszönhetően *lokálisak* a procedúrára nézve. Csak addig foglaltak, míg a procedúra dolgozik, aktív. A procedúrát munkára fogni az aktivizáló utasítással lehet. Ezt *eljáráshívás*nak is nevezik. Az aktivizált procedúra lehet saját maga az aktivizáló is, ekkor *rekurzív hívás*ról beszélünk, a procedúrát pedig *rekurzív procedúrá*nak nevezzük. A procedúra munkája végén a vezérlés visszaadódik az aktivizáló utasítást követő utasításra. Ezt a mechanizmust a *verem* (stack) révén valósítjuk meg. A verem a memóriának a programfutás idejére egy erre a célra kiválasztott része. A procedúra aktivizálásakor ide kerülnek beírásra a procedúra paraméterei és a *visszatérési cím* (az aktivizáló utasítást követő utasítás címe). A procedúrából való visszatéréskor ezen cím és információk alapján tudjuk folytatni a munkát, a programot. A visszatéréskor a veremből az aktivizálási információk törlődnek. Ha a procedúra aktivizál egy másik procedúrát, akkor a verembe a korábbiakat követően az új aktivizálási információk is bekerülnek, azt mondjuk, hogy a verem mélyül, a veremmélység szintszáma eggyel nő. Kezdetben a verem üres, a szintszám zérus, procedúrahíváskor a szintszám nő eggyel, visszatéréskor csökken eggyel. Az egyes szintek nem feltétlenül jelentenek ugyanolyan mennyiségű területfoglalást a veremből. A dolog pikantériájához tartozik továbbá, hogy a procedúra a lokális változóit is a verembe szokta helyezni, csak ezt közvetlenül nem érzékeljük, mivel a visszatéréskor ezek onnan törlődnek, a helyük felszabadul. Időnként azonban a hatás sajnálatosan látványos, amikor *verem túlcsordulás* (stack overflow) miatt hibajelzést kapunk és a program futása, a feladat megoldásának menete megszakad. Adódhat

azonban úgy is, hogy mindenféle hibajelzés nélkül „lefagy a gép”. A veremnek a valóságban van egy felső méretathatára, amelyet nagyon nem tanácsos túllépni. Ugyanilyen veszélyes a verem alulcsordulás (*stack underflow*) hiba is, amikor az üres veremből akarunk valamit eltávolítani, kivenni.

Nézzünk egy példát a veremhasználatra. Tegyük fel, hogy össze kell adni 5 egész számot és erre programot készítünk. Ugyanakkor óvatosak vagyunk, mert sejtjük, hogy más alkalommal esetleg nem öt, hanem hat, hét, stb. számot kell majd összeadni, ezért úgy írjuk meg, hogy a programot egy bizonyos határig, mondjuk, 20 számig lehessen használni. A program kérje be az összeadandó számok számát, majd a számokat, végezze el az összegzést, és írja ki a végösszeget! Az összegzés gyakran előforduló részfeladat, ezért rá külön procedúrát készítünk, hogy esetleg más programokban is felhasználhassuk. Először az összegző procedúrát készítjük el például az alábbi módon. A procedúra neve legyen Summa és legyen paramétere egy ***A*** tömb (vektor) címe. A tömb részére 20 memóriarekeszt foglalunk le. Legyen továbbá paramétere az *n* szám, amely azt mutatja meg, hogy a tömb egyes indexű elemével kezdve, az *n*-essel bezárólag kell összegezni a tömbelemeket. Paraméterként (output paraméter) kell megadnunk azt is, hogy hova kerüljön az összeg. Ennek a rekesznek a neve legyen *s*. Valójában nekünk a címe *@s* kell. A név formális, mert csak szimbolizálja, hogy a procedúra hívásakor a konkrét rekeszcímet kell megadnunk. Kirészletezzük egy kissé a procedúra teendőit. Szükség lesz továbbá egy lokális számlálóra, legyen a neve *k*, amellyel egytől egyesével elszámolunk *n*-ig és minden egyes *k*-ra az *s*-hez a számlálás közben hozzáadjuk az a_k számot. Az *s*-et a munka kezdetén természetesen ki kell nullázni, hiszen nem tudjuk, hogy mi van benne az induláskor. Ezek után egy kissé rendezettebb alakban is írjuk le a tudnivalókat, teendőket.

A Summa procedúra leírása

Összefoglaló adatok a procedúráról:

A procedúra neve:	Summa.
Bemenő paraméter:	*@**A**,* ahol $\boldsymbol{A} \in \boldsymbol{Z}^{20}$, tömb, amely tartalmazza az összeadandó számokat.
Bemenő paraméter:	$n \in \{1,2, \ldots, 20\}$, megadja, hogy az egyes indexű elemmel kezdve meddig kell az összeadást elvégezni. Tulajdonképpen lehet *n*=Hossz[***A***].
Kijövő paraméter:	*@s,* ahol $s \in \boldsymbol{Z}$, tartalmazza az összeget a végén.
Lokális változó:	$k \in \{1,2, \ldots, 20\}$, számláló, amellyel egytől elszámolunk *n*-ig egyesével.

A procedúra tevékenysége:

0. lépés:	lokálisan helyet készíteni a *k* rekesznek.
1. lépés:	*s* kinullázása
2. lépés:	*k* beállítása 1-re, innen indul a számlálás
3. lépés:	*s* megnövelése a_k-val (*s*-hez hozzáadjuk a_k-t és az eredmény *s*-ben marad.
4. lépés:	Eggyel megnöveljük a *k* számláló értékét
5. lépés:	Ellenőrizzük, hogy a *k* számláló nem lépett-e túl az *n*-nen, a végértéken. Ha még nem, akkor folytatjuk a munkát a 3. lépésnél. Ha igen, akkor pedig a 6. lépéshez megyünk.
6. lépés:	Készen vagyunk, az eredmény az *s*-ben található.

A főprogram, amely használja ezt a procedúrát, legyen a következő.

Az összeadandó számokat egy ***X*** nevű globális tömbben fogja tárolni. (A főprogram írója nem biztos, hogy látja a procedúra szövegét, lehet, hogy csak annyit tud, hogy hogyan kell meghívni azt. Általában ez szokott lenni a helyzet.) Az összeadandó számok darabszámát egy *darab* nevű globális rekeszben tárolja. Az eredményt pedig egy *végösszeg* nevű globális rekeszben szeretné látni, és majd onnan kiíratni.

Összefoglaló adatok a főprogramról:

A program neve:	Szummázó.
Globális változó:	$\boldsymbol{X} \in \boldsymbol{Z}^{20}$, tömb, amely tartalmazza az összeadandó számokat.
Globális változó:	$darab \in \{1,2, \ldots, 20\}$, megadja, hogy az egyes indexű elemmel kezdve meddig kell az összeadást elvégezni.
Globális változó:	$végösszeg \in \boldsymbol{Z}$, tartalmazza az összeget a végén.

A főprogram tevékenysége:

0. lépés:	Globálisan helyet készíteni az ***X***, *darab*, és *végösszeg* rekeszeknek. (Változók deklarációja)
1. lépés:	A *darab* és az $x_1, x_2, \ldots, x_{darab}$ bekérése.
2. lépés:	A Summa procedúra meghívása a *@**X***, *darab* és *@végösszeg* paraméterekkel
3. lépés:	Az eredmény kiíratása a *végösszeg* rekeszből.
4. lépés:	STOP

Egy kis szimbolikát használva még tömöríthetünk a leírtakon.

Summa procedúra		
1.	Input paraméterek:	*@**A***, ahol $\boldsymbol{A} \in \boldsymbol{Z}^{20}$,
2.		$n \in \{1,2, \ldots, 20\}$
3.	Output paraméter:	$@s$, ahol $s \in \boldsymbol{Z}$.
4.	Lokális változó:	$k \in \{1,2, \ldots, 20\}$,
5.	$k \leftarrow 1$	
6.	$s \leftarrow s + a_k$	
7.	$k \leftarrow k+1$	
8.	Ha $k \le n$ akkor vissza 6-os sorhoz, egyébként folytatás a 9-es sornál	
9.	Kész. Visszatérés.	

Szummázó főprogram		
1.	Globális változók:	$\boldsymbol{X} \in \boldsymbol{Z}^{20}$,
2.		$darab \in \{1,2, \ldots, 20\}$
3.		$végösszeg \in \boldsymbol{Z}$
4.	Bekérni *darab*-ot és az $x_1, x_2, \ldots, x_{darab}$-ot. (Input utasítások.)	
5.	Meghívni a Summa procedúrát: CALL Summa(*@**X***, *darab*, *@végösszeg*)	
6.	Kiírni *végösszeg*-et. (Output *végösszeg*).	
7.	STOP	

Ezután, ha szükségünk van, mondjuk, az $x_1=3$, $x_2=-4$,, $x_3=2$, $x_4=8$, $x_5=-1$ számok összegére, a főprogram elindítása után *darab*=5 lesz, az *x*-ek beállnak a fentebbi értékekre, a CALL révén a verembe bekerül az

X tömb címe	5	*végösszeg* címe	főprogram 6. sor visszatérési cím

információ. Kezdetben üres volt a verem, most egy szint került bele bejegyzésre. Amikor a procedúra munkája véget ér, akkor ez a bejegyzés a veremből törlődik, így az újra üres lesz. (Tulajdonképpen a *k* számláló számára lefoglalt helyet is fel kellett volna tüntetni a bejegyzésben, de ez a számunkra most nem fontos.) Megjegyezzük, hogy kissé pazarolnánk a vermet, ha a *@**X*** helyett csak ***X***-et írtunk volna, mivel ekkor a teljes tömb is bekerült volna oda és az sok helyet foglalt volna el. (Persze ekkor a procedúrában a *@**A*** helyett ***A*** állna.) A tömb elemei a címen keresztül is elérhetők és a cím csak pár byte-ot foglal el függetlenül a tömb méretétől. A fenti procedúrát *iteratív procedúrá*nak nevezzük, mert egy ismétlés (*iteráció*) található benne, ugyanis a 6-os, 7-es sorok ismétlődnek mindaddig, míg az összeg el nem készül. Az elkészültét pedig egy vizsgálat figyeli a 8-as sorban.

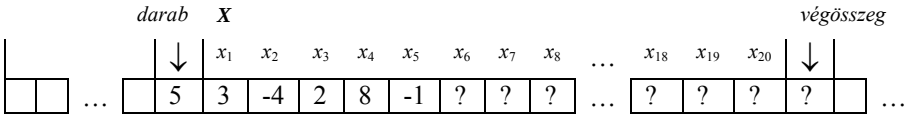
Más módon is megoldható azonban az összegzés. A feladatot vissza is vezethetjük kisebb méretű feladatokra. Az *n* szám összege képezhető úgy is, hogy az *n* szám első felének az összegéhez hozzáadjuk az *n* szám második felének az összegét. Viszont ez a két kisebb méretű feladat is tovább felezhető. Meddig? Egészen addig, míg már csak egyetlen elem marad a felezés után. Ennek az egyetlen számnak pedig már tudjuk az összegét – saját maga. Ezután készíthetünk a Summa procedúra helyett egy másikat, mondjuk *Rekurzív_Summa* néven. (A *rekurzió* szó visszavezetést jelent). A tömör formájú változat valahogy így nézhet ki.

Rekurzív_Summa procedúra		
1.	Input paraméterek:	$@\boldsymbol{A}$, ahol $\boldsymbol{A} \in \boldsymbol{Z}^{20}$,
2.		$m,n \in \{1,2, \ldots, 20\}$ és $m \le n$,
3.	Output paraméter:	$@s$, ahol $s \in \boldsymbol{Z}$
4.	Lokális változók:	$u,v \in \boldsymbol{Z}$
5.	Ha $m=n$ akkor	$s \leftarrow a_m$ és folytatás a 9. sornál.
6.	egyébként	CALL *Rekurzív_Summa*$(@\boldsymbol{A}, m, \left\lfloor \frac{m+n}{2} \right\rfloor, @u)$
7.		CALL *Rekurzív_Summa*$(@\boldsymbol{A}, \left\lfloor \frac{m+n}{2} \right\rfloor + 1, n, @v)$
8.		$s \leftarrow u+v$ és folytatás a 9. sornál.
9.	Kész. Visszatérés.	

Szummázó főprogram		
1.	Globális változók:	$\boldsymbol{X} \in \boldsymbol{Z}^{20}$,
2.		$darab \in \{1,2, \ldots, 20\}$
3.		$végösszeg \in \boldsymbol{Z}$
4.	Bekérni *darab*-ot és az $x_1, x_2, \ldots, x_{darab}$-ot. (Input utasítások.)	
5.	Meghívni a Summa procedúrát: CALL Summa(*@**X***, 1, *darab*, *@végösszeg*)	
6.	Kiírni *végösszeg*-et. (Output *végösszeg,* kiíró, output utasítás).	
7.	STOP	

Az 1-3. sorok úgynevezett deklarációs sorok, amelyben megadjuk a használt globális változóink nevét, típusát, méretét. A deklarációban leírtakat a fordítóprogram használja fel

információként a fordításhoz, a memória kiosztáshoz. A 4. sorral kezdődnek a végrehajtható sorok, amelyek már ténylegesen csinálnak is valamit (belőlük processzor utasítás, gépi kódú utasítás keletkezik). A 4. sor végrehajtása után a memória a fentebb megadott input adatok esetén az alábbi módon épül fel, legalább ís elvileg. A ? jelzi, hogy az adott rekeszeknek a tartalma ismeretlen.



Hogyan alakul a verem sorsa ezután ebben az esetben? A főprogram 5. sora aktivizálja a procedúrát először, melyre a hívás hatására a verem:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@**X**	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?

Miután $m \neq n$, ezért ezután következik a procedúra 6. sorában a CALL hívás. A procedúra meghívja saját magát. A hatása:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@**X**	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@**X**	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?

Miután a veremben a procedúra minden meghívása során létrehozza az *u* és *v* lokális változókat az újabb szinten új példányként, ezért az „Output paraméter” oszlopban az *u* és *v* neve után zárójelben feltüntettük, hogy melyik szinten lévő példányról van szó. Az új hívás után újra a procedúra 5. sora jön. A hatás:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@**X**	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@**X**	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?
2	@**X**	*m*=1	*n*=2	@*u*(1)	procedúra 7. sorra	*u*=?	*v*=?

Ez megint nem számolható közvetlenül, újra az 5. sor jön, mire a verem új képe:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@**X**	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@**X**	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?
2	@**X**	*m*=1	*n*=2	@*u*(1)	procedúra 7. sorra	*u*=?	*v*=?
3	@**X**	*m*=1	*n*=1	@*u*(2)	procedúra 7. sorra	*u*=?	*v*=?

Itt már van eredmény, mert $m=n$ és így átmenetileg nincs több rekurzív hívás. Az eredmény $u(3)=a_1=3$.

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@**X**	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@**X**	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?
2	@**X**	*m*=1	*n*=2	@*u*(1)	procedúra 7. sorra	*u*=3	*v*=?
3	@**X**	*m*=1	*n*=1	@*u*(2)	procedúra 7. sorra	*u*=?	*v*=?

A hívás befejezte után a veremből kiürül a legutolsó bejegyzés, visszatérünk a procedúra 7. sorához, amely azonban egy újabb rekurzív hívás. Hatására a verem képe:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?
2	@***X***	*m*=1	*n*=2	@*u*(1)	procedúra 7. sorra	*u*=3	*v*=?
3	@***X***	*m*=2	*n*=2	@*v*(2)	procedúra 8. sorra	*u*=?	*v*=?

Az innen történő visszatérés után a verem képe:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=?	*v*=?
2	@***X***	*m*=1	*n*=2	@*u*(1)	procedúra 7. sorra	*u*=3	*v*=-4

A 8. sorban elvégezve az összeadást az eredmény u(1)=3+(-4)=-1.

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=-1	*v*=?

Visszatérés után újra a 7. sor jön:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=-1	*v*=?
2	@***X***	*m*=3	*n*=3	@*v*(1)	procedúra 8. sorra	*u*	*v*

Visszatérés után az eredmény $v(1)=x_3=2$ és a verem:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=?	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=-1	*v*=2

A 8. sor szerinti összeg u(0)=-1+2=1 és jön.

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=1	*n*=3	@*u*(0)	procedúra 7. sorra	*u*=-1	*v*=2

Visszatérés a 7.sorra, ahol újabb hívás van:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=4	*n*=5	@*u*(0)	procedúra 8. sorra	*u*=?	*v*=?

Mivel $m \neq n$, ezért újabb hívás, mélyül a verem:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	@*végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=4	*n*=5	@*u*(0)	procedúra 8. sorra	*u*=?	*v*=?
2	@***X***	*m*=4	*n*=4	@*u*(1)	procedúra 7. sorra	*u*=?	*v*=?

Itt van eredmény: $u(0)=x_4=8$.

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	*@végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=4	*n*=5	*@u*(0)	procedúra 8. sorra	*u*=8	*v*=?
2	@***X***	*m*=4	*n*=4	*@u*(1)	procedúra 7. sorra	*u*=?	*v*=?

Visszatérés után a 7. sorban újabb hívás:

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	*@végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=4	*n*=5	*@u*(0)	procedúra 8. sorra	*u*=8	*v*=?
2	@***X***	*m*=5	*n*=5	*@v*(1)	procedúra 8. sorra	*u*=?	*v*=?

Van eredmény: $v(1)=x_5=-1$.

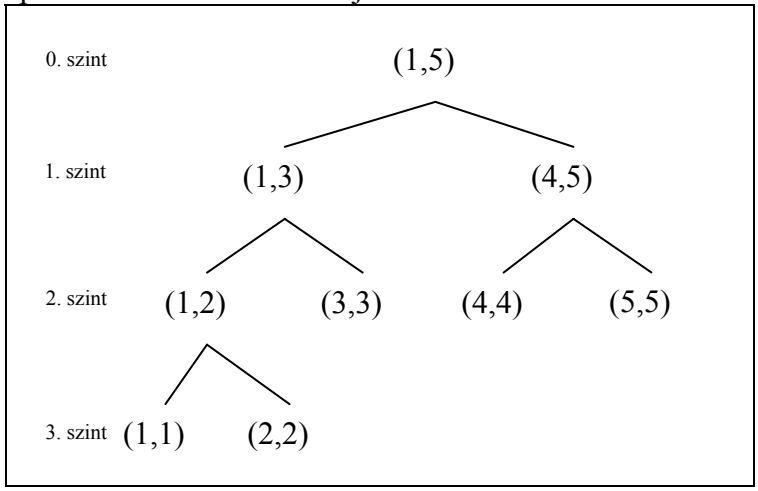
szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	*@végösszeg*	főprogram 6. sorra	*u*=1	*v*=?
1	@***X***	*m*=4	*n*=5	*@u*(0)	procedúra 8. sorra	*u*=8	*v*=-1
2	@***X***	*m*=5	*n*=5	*@v*(1)	procedúra 8. sorra	*u*=?	*v*=?

Visszatérés után a először v(0)=8+(-1)=7.

szint	Input paraméterek			Output paraméter	Visszatérési cím (IP)	Lokális változók	
0	@***X***	*m*=1	*n*=5	*@végösszeg*	főprogram 6. sorra	*u*=1	*v*=7
1	@***X***	*m*=4	*n*=5	*@u*(0)	procedúra 8. sorra	*u*=8	*v*=-1

Újabb visszatérés után *végösszeg=u+v*=1+7=8. majd a verem kiürül. Visszatérünk a főprogram 6. sorára, ahol kiíratjuk az eredményt a végösszeg rekeszből és ezután a STOP miatt a program befejezi a munkáját, a vezérlést visszaadja az operációs rendszernek.

Láthattuk, hogy négy veremszint alakult ki a program futása során. Mivel a szintek számozása zérusról indul és *a maximális veremmélység definíció szerint a legmagasabb számú szint száma*, ezért a feladat megoldásához szükséges maximális veremmélység most három volt. A rekurzív hívások száma eggyel kevesebb, mint az összes hívásszám, mivel az első aktivizáló hívás még nem számít rekurzív hívásnak. Konkrétan a rekurzív hívások száma nyolc volt, ha utána számolunk. Ebben a rekurzióban minden hívás, kivéve a legalsóbb szinten levőket két újabbat eredményezett, de ezek a veremnek ugyanazon szintjét használták. A hívások szerkezetét egy úgynevezett *hívási fa séma*val tudjuk ábrázolni, melyben most csak az *m*, *n* paraméter értékeket tüntetjük fel. Íme:



Az ábrán jól látszik a verem négy szintje. A legfelső szint kivételével a többi szinten lévő hívások rekurzívak. Az azonos szinten lévő hívások a verem azonos szintjét használják csak eltérő időben.

A fenti *Rekurzív_Summa* procedúra matematikai formula (egy *RSum* nevű függvény) formájában is megfogalmazható az alábbi módon például:

$$RSum(m,n)=\begin{cases} a_m, & ha\ m=n \\ RSum\left(m,\left\lfloor\frac{m+n}{2}\right\rfloor\right)+RSum\left(\left\lfloor\frac{m+n}{2}\right\rfloor+1,n\right), & ha\ m<n. \end{cases}$$

Nézzük meg, hogy hogyan is számol ez a ravasz függvény a mi speciális $s=RSum(1;5)$ esetünkben, ahol $m=1$, $n=5$ és $a_1=3$, $a_2=-4$, $a_3=2$, $a_4=8$, $a_5=-1$.

$s=RSum(1;5)$ $=RSum(1;3) + RSum(4;5)$
$=(RSum(1;2) + RSum(3;3)) + (RSum(4;4) + RSum(5;5))$
$=((RSum(1;1) + RSum(2;2)) + 2) + (8 + (-1)))$
$=((3 + (-4)) + 2) + (8 + (-1)))$
$=((-1) + 2) + (8 + (-1))$
$=(1 + 7)$
$=8$

FELADATOK

1. Az *n*! (*n faktoriális)* fogalmát pozitív egész számokra az alábbi rekurzív formulával definiáljuk:

$$n! = \begin{cases} 1, & ha\ \ n = 1 \\ n \cdot (n-1)!, & ha\ n > 1 \end{cases}$$

Procedúrahívással kiszámítottuk a 7! értékét. Hogyan alakult a verem felépítése, mélysége? Rajzoljuk meg a hívási fát! Mekkora a minimális méretű verem, amely a feladat elvégzéséhez szükséges? Hány rekurzív hívás volt a számolás során?

2. Definiáljunk egy $P(m,n)$ függvényt, amely pozitív egész *m* és *n* argumentumokra van definiálva és $m \le n$ fenn kell, hogy álljon.

$$P(m,n) = \begin{cases} m, & ha\ m = n \\ P\left(m, \left\lfloor \frac{m+n}{2} \right\rfloor\right) \cdot P\left(\left\lfloor \frac{m+n}{2} \right\rfloor + 1, n\right), & ha\ m \neq n \end{cases}$$

Világos (de azért lássuk is be), hogy *n*! ezzel a függvénnyel kiszámítható a $P(1,n)$ paraméterezéssel. Procedúrahívással számíttassuk ki a 7! számot! Hogyan alakul a verem felépítése, mélysége? Rajzoljuk meg a hívási fát! Mekkora a minimális méretű verem, amely a feladat elvégzéséhez szükséges? Hány rekurzív hívás lesz a számolás során?

3. A binomiális együttható szimbólum definíciója: $\binom{n}{k} \overset{def}{=} \frac{n!}{k! \cdot (n-k)!}$. A szimbólum kiolvasása „*n* alatt a *k*”. A szimbólumot nemnegativ egész *n* és *k* értékekre definiáljuk, ahol $0 \le k \le n$ fönn kell, hogy álljon, továbbá megegyezés szerint $0! \overset{def}{=} 1$. Bizonyítsuk be, hogy $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. Ezt felhasználva adjunk rekurzív módszert $\binom{n}{k}$ kiszámítására. Konkrétan határozzuk meg $\binom{7}{3}$ értékét rekurzívan! Hogyan alakul a verem felépítése, mélysége? Rajzoljuk meg a hívási fát! Mekkora a minimális méretű verem, amely a feladat elvégzéséhez szükséges? Hány rekurzív hívás lesz a számolás során? Tudnánk-e ugyanezen kérdésekre általános választ is adni?

4. A nemnegatív egész számokon értelmezzük a következő függvényt rekurzív módon:

$$F(n) = \begin{cases} 0, & ha\ \ n = 0 \\ 1, & ha\ \ n = 1 \\ F(n-1) + F(n-2), & ha\ \ n > 1 \end{cases}$$

Procedúrahívással számíttassuk ki az F(7) számot! Rajzoljuk meg a hívási fát! Hogyan alakul a verem felépítése, mélysége? Mekkora a minimális méretű verem, amely a feladat elvégzéséhez szükséges? Hány rekurzív hívás lesz a számolás során?

5. Legyen az alábbi kétváltozós függvényünk, amelyet nemnegatív egész argumentumokra értelmezünk rekurzívan:

$$P(a,b) = \begin{cases} 0, & ha\ \ b = 0 \\ P(2a, b\ div\ 2), & ha\ \ b > 0\ \ és\ \ b\ \ páros \\ a + P(a, b-1), & egyébként \end{cases}$$

Procedúrahívással számíttassuk ki a P(2,5) számot! Hogyan alakul a verem felépítése, mélysége? Rajzoljuk meg a hívási fát! Mekkora a minimális méretű verem, amely a feladat elvégzéséhez szükséges? Hány rekurzív hívás lesz a számolás során?