Elmélet

2-es szint:

- Algoritmus és leírási módszerei.
- Adat, adatszerkezet, adattípus, változó, konstans fogalmai.
- Láthatóság és élettartam fogalmai.
- Standard I/O könyvtári függvények, standard header állományok.
- Kifejezések és operátorok. Operátorok precedenciája, kifejezések kiértékelése.
- Implicit, explicit típuskonverzió.
- A C nyelv utasításai. Vezérlési szerkezetek C nyelvi megvalósítása.
- Egydimenziós numerikus tömbök tárolása, kezelése.
- Függvény definiálása, deklarálása. Függvényhívás mechanizmusa, paraméterátadás.
- Alapalgoritmusok: számlálás, összegzés, eldöntés, kiválasztás, keresés, rendezés minimum/ maximum kiválasztással**.(Ez a rész ebben a jegyzetben nincs kidolgozva!)**

Sikeres felkészülést és vizsgát bárkinek, aki erre a jegyzetre vetemedik!

Készítette: Sirankó Boldizsár vulgo Az univerzális hanggenerátor avagy csak a mémeket kajálom

Algoritmus és leírási módszerei.

Algoritmus (eljárás):

Olyan megengedett lépésekből álló módszer, utasítás(sorozat), részletes útmutatás, amely valamely felmerült probléma megoldására alkalmas.

Megengedett lépések:

Elemi utasítások, amelyeket a feladat végrehajtója képes értelmezni és végrehajtani.

Algoritmus:

Egy probléma megoldásának végesz számú részlépésben történő egyértelmű és teljes leírása.

Megjegyzés: A számítógépes program az algoritmus konkrét, az adott gép lehetőségeihez igazított alakja.

A helyes algoritmus jellemzői:

- egy értékhez vagy ezek halmazához (input) hozzárendel egy értéket vagy ezek egy halmazát(output)
- véges számú, elvégezhető lépésből áll
- redundáns lépéseket nem tartalmaz
- egyértelmű(minden lépés pontosan definiált)
- determinisztikus(ugyanarra a bemenetre ugyanazt az eredményt szolgáltatja)
- hatékony és teljes(minden azonos jellegű feladatra alkalmazható

Algoritmusok leírási módszerei:

- szöveges leírás(nem exakt)
- Pszeudokód(metanyelv)
- Folyamatábra vagy blokkdiagram (szabványos)
- Struktogram: Egyetlen téglalap tagolása, amely a teljes feladat részekre bontását jelenti. Top down tervezéshez hasznos.
- Jackson diagram

Adat, adatszerkezet, adattípus, változó, konstans fogalmai

Adat:

Adatnak (angolul data) nevezzük a számokkal leírható dolgokat, melyek számítástechnikai eszközökkel rögzíthetők, feldolgozhatók és megjeleníthetők.

Adatszerkezet:

Adatszerkezetnek nevezzük az adattárolási célokat szolgáló strukturális, formai elrendezést.

Adattárolási egységek:

- Bit
- Bájt(8 bit)
- Szó(32 bit)
- Dupla szó(64 bit)

A kettes számrendszer béli működés miatt a szokásos mértékegységeknek megvan párja:

(10^3)Kilobyte	->	Kibibyte(2^{10})
(10^6)Megabyte	->	Mebibyte(2^{20})
(10^9)Gigabyte	->	Gibibyte(2^{30})
(10^{12})Terabyte	->	Tebibyte(2^{40})

Adattípus:

A változó tárolásához szükséges memóriaterület méretét és a tárolt érték értelmezését határozza meg.

Típuselőírások:

<table>
<tr><td rowspan="3">Egyszerű típusok</td><td>char
int
enum</td><td rowspan="2">egész jellegű</td><td rowspan="2">aritmetikai</td></tr>
<tr><td>float
double</td><td>lebegőpontos</td></tr>
<tr><td>pointer</td><td colspan="2"></td></tr>
<tr><td rowspan="2">Összetett típusok</td><td>tömb
struktúra</td><td rowspan="2">összeállított</td><td rowspan="2"></td></tr>
<tr><td>unió</td></tr>
<tr><td>Üres típus</td><td>void</td><td colspan="2"></td></tr>
</table>

Típusmódosítók:

- Hossz:
  - long, short
- Előjel:
  - signed, unsigned

Típusminősítők

- const: értéke nem változik
- volatile: programtól független kód is megváltoztathatja

Változó:

Olyan memóriaterület, amely egy vagy több értéket tartalmaz.

Jellemzői:

- Azonosító név
- Típus
- Tárolási osztály
- Memóriacím
- Aktuális érték

Értéket rendelhetünk a változóhoz, illetve lekérdezhetjük a változóban tárolt értéket.

A változók használata:

- Deklarálás:
  - int a;
- Incializálás:
  - a = 5;
- Hivatkozás:
  - a=a+5;

Konstans:

Olyan változó amelynek értéke futási időben nem változtatható meg. (Azaz állandó.)

Konstans létrehozása:

- const-tal (pointerekkel változtatható az értéke így is.)
- konstans makrokkal (csak forrásfáljban)
- enum adattípussal.

Láthatóság és élettartam fogalmai

Hatáskör/Láthatóság:

A program azon része, melynek határain belül az adott azonosító látható.

Élettartam:

A program végrehajtásának az az időszaka, amíg adott változó a vagy függvény a memóriában létezik.

- Statikus/Globális
  - A programban végig látható, és végig foglal memóriát.
- Automatikus/Lokális
  - Adott blokkban látható, és csak a blokkon belül foglal memóriát.

Megjegyzés: Minden függvény vagy változó saját blokkjában lokálisnak tekinthető, a blokk alblokkjaiban globálisnak.

- Dinamikus: A programozó által van lefoglalva.

Standard I/O könyvtári függvények, standard header állományok

Standard I/O függvények deklarációja:

#include <stdio.h>

Formázott output:

printf(”formátum”, argumentumlista);

Formátumok	
%d	int
%c	char
%f	float
%lf	double
%s	string

Formázott input:

scanf(”formátum”, argumentumlista);

- A szabványos bemenetről olvas karaktereket, formátumsztring szerint konvertál. A konvertálás eredményét a soron következő argumentum által kijelölt memóriacímre tölti.
- Spaceig(” ”) vagy Új sor karakterig(’\n’) olvas.
- Enter->Puffer->Feldolgozás

Standard header állományok

(Megjegyzés: A tanárnő nem igazán mondta el, hogy mi is az a standard az I/O-n kívül, szóval egy két hasznos header állomány van ide kigyűjtve.)

- stdio.h
  - I/O függvények
- stdlib.h
  - Általános segédfunkciók
- math.h
  - Matematikai függvények
- string.h
  - sztringfüggvények
- time.h
  - Dátum, idő függvények

Kifejezések és operátorok, operátorok precedenciája, kifejezések kiértékelése

Operandus:

A C nyelv azon eleme, amelyen az operátor kifejti hatását

Kifejezés:

Vagy egyetlen operandusból, vagy operandusok és műveleti jelek (operátorok) kombinációjából épül fel.

Fajtái:

- Elsődleges kifejezés:
  - Konstans érték, azonosító, sztring literál, függvényhívás, tömbindex kifejezés, struktúra tagkiválasztó kifejezés
- Összetett kifejezés:
  - Ami zárójelezett vagy zárójel nélküli, operátorokkal összekapcsolt operandusokból áll.

Kifejezés kiértékelése:
Egy kifejezés kiértékelésének eredménye lehet:

- egy érték kiszámítása(főhatás)
- függvényhívás
- mellékhatás

Mellékhatás:

Ha a kifejezés kiértékelése a program változóiban olyan változást idéz elő, amely nem explicit.

Például:
a = 2 + 3;
Főhatás: 2+3 kiszámítása
Mellékhatás: a értéke beállítódik 5-re.

Megjegyzés: Ha nincs mellékhatás, akkor a kifejezés transzparens.

Operátorok

Csoportosításuk az operandusok száma szerint:

- Egyoperandusú operátor esetén a kifejezés alakja:
  - Prefix/előrevetett alak: operátor operandus
  - Postfix/hátravetett alak: operandus operátor
- Kétoperandusú operátor esetén a kifejezés alakja:
  - operandus1 operátor operandus2
- Háromoperandusú (feltételes) operátor
  - kifejezés1 ? kifejezés2 : kifejezés3
  - Először kiértékelődik a kifejezés1, ami ha nem 0(azaz igaz), akkor a kifejezés2 végrehajtódik, különben a kifejezés3 hajtódik végre.

Elsődleges operátorok

Azok az egyoperandusú (unary) operátorok, amelyekkel elsődleges kifejezések hozhatók létre. ((), [], ., ->)

- Függvény(argumentumok): függvényhívás operátor
- tömb[index]: tömbindexelés operátor
- str.tag vagy pstr->tag
  struktúra ill. struktúrára mutató pointer adattagjára hivatkozás operátorai
- Zárójel () operátor: kifejezések csoportosítása, műveletek kiértékelési sorrendjének előírása

Aritmetikai operátorok

+	Összeadás
-	Kivonás
*	Szorzás
/	Osztás
%	Modulo osztás(a maradékot adja eredményül)

Megjegyzés: Polimorfizmusra figyelni kell:
26/4=6
26/4.0=6.5

Megjegyzés:
Ha a – jelet egyoperandusú operátorként használjuk, az előjelváltásként működik.

Összehasonlító és logikai operátorok

C-ben nincs logikai adattípus!!!

Összehasonlító operátorok:

<	kisebb
<=	kisebb egyenlő
>	nagyobb
>=	nagyobb egyenlő
==	egyenlő
!=	nem egyenlő

Logikai operátorok:

&&	ÉS
||	VAGY
!	NEM

Léptető operátorok

Unáris operátorok, csak balérték operandussal

++	Inkrementáció
--	Dekrementáció

Értékadó operátor

=	értékbeállítás
operátor=	műveletes értékbeállítás (pl += vagy *=)

Pointer operátorok

& :

A címe operátor az operandusként megadott memóriaváltozó címét adja meg. Arra használjuk, hogy a mutatót már meglévő változóra irányítsuk.

*** :**

Az indirekt hivatkozás operátora (dereferencia operátor) a pointer által mutatott memóriaváltozót jelöli ki.

sizeof operátor

A változó méretét adja meg.

Használata:

sizeof változó **VAGY** sizeof(típusnév)

Vessző operátor

Egy kifejezésben több, akár egymástól független kifejezés is szerepelhet a vessző operátor alkalmazásával. Az ilyen kifejezés balról jobbra kerül kiértékelésre; és a kifejezés értéke és típusa a jobboldali operandus értékével és típusával egyezik meg.
x = (y = 3, y + 2); → x = 5

Gyakran használjuk változók kezdőértékeinek egyetlen utasításban történő beállítására.

int x, y, z;
x = 2, y = 3, z = 5;

Megjegyzés: Deklarációkban a változónevek (azonosítók) elkülönítésére, valamint a függvényargumentumok elkülönítésére használt vessző – ami írásjel, elválasztó karakter – NEM azonos a ***VESSZŐ OPERÁTOR****ral.*

Feltételes operátor

(?:)
A háromoperandusú operátornál használjuk.

- Háromoperandusú (feltételes) operátor
  - kifejezés1 ? kifejezés2 : kifejezés3
  - Először kiértékelődik a kifejezés1, ami ha nem 0(azaz igaz), akkor a kifejezés2 végrehajtódik, különben a kifejezés3 hajtódik végre..

Operátorok precedenciája

A precedencia szabály:

> A kifejezésekben szereplő operátorok/műveletek kiértékelésének sorrendjét írja elő. (Zárójel felülírhatja)

Asszociativitási szabály:

> Azonos precedenciájú műveletek sorrendjét írja elő.

Megjegyzés: A következő táblázat pontosan így van a diasorban.

Operátorok	Asszociativitás
Unáris: ( ) [ ] . ->	Balról jobbra
Unáris: ! ~ - ++ -- & * (típus) sizeof	Jobbról balra
Multiplikatív: * / %	Balról jobbra
Additív: + -	Balról jobbra
Biteltoló: << >>	Balról jobbra
Relációs: < <= > >=	Balról jobbra
Egyenlőségvizsgáló: == !=	Balról jobbra
&	Balról jobbra
^	Balról jobbra
|	Balról jobbra
&&	Balról jobbra
||	Balról jobbra
?:	Jobbról balra
Értékadó: = += -= *= /= %= <<= >>= &= |= ^=	Jobbról balra
Vessző: ,	Balról jobbra

Implicit, explicit típuskonverzió

Típuskonverzió

Mikor van rá szükség?

- Kifejezések kiértékelésekor, ha a kétoperandusú operátor különböző típusú operandusokkal rendelkezik.
- Függvények megfelelő argumentummal történő meghívásához.

Típusai:

- Implicit(automatikus)
  - A fordító végzi, rögzített szabályok alapján.
- Explicit(kikényszerített)
  - A programozó írja elő típuskonverziós operátor segítségével.

Példa:

```
int a =5, b =2;

double c;

//Implicit:

    c=a/b;

    //így c=2.0

//Explicit:

    c=(double)a/b;

    //így c=2.5
```

Általános szabály:

"Szűkebb" adattípus információvesztés nélkül konvertálódik "szélesebb" adattípusra.

- Az egész típusok int-re konvertálódnak.
- Az egész típus mindig átalakítható lebegőpontos típussá.
- Lebegőpontos típus egész típusra alakításakor a törtrész elvész.
- A kétoperandusú műveletek többsége az operandusait és az eredményt a legnagyobb közös típusúra konvertálja

A C nyelv utasításai, vezérlési szerkezetek C nyelvi megvalósítása

A program végrehajtható része elvégzendő tevékenységekből, azaz utasításokból épül fel.

Csoportosításuk:

1. Kifejezés utasítás
2. Üres utasítás
3. Összetett utasítás
4. Szelekciós utasítások
5. Címkézett utasítások
6. Vezérlésátadó utasítások
7. Iterációs (ciklus) utasítások

Kifejezés utasítás:

Tetszőleges kifejezés utasítás lesz, ha utána ;-t teszünk. Egy kifejezés utasítás végrehajtása a kifejezés kiértékelését jelenti.

Példa:

```
kerulet = a + b + c;   // értékadás
            i++;   // i növelése eggyel
        a = b = 0;   // többszörös értékadás
      kisebb(a,b);   // függvényhívás
z = sin(alfa) + 1.25;   // függvényt hívó kifejezés
```

Üres utasítás

Üres utasítás: **;**

Akkor van rá szükség, amikor logikailag nem kívánunk semmilyen tevékenységet végrehajtani, de a szintaktikai szabályok szerint a program adott pontján utasításnak kell szerepelnie.

Példa:

```
double a, b, c;
if (b != 0)
c = a / b;
else      ; //üres utasítás
```

Összetett utasítás

A logikailag összetartozó deklarációk és utasítások egyetlen összetett utasításba (blokkba) csoportosítására szolgál. Mindenhol használható, ahol szintaktikailag utasítás megadása szükséges.

Általános formája:

```
{
lokális definíciók és deklarációk
utasítások
}
```

Megjegyzés: Nincs utána ;

Mikor használjuk?

- Amikor több logikailag összefüggő utasítást egyetlen utasításként kell kezelni (ilyenkor csak utasításokat tartalmaz a blokk).
- Függvények törzseként.
- Definíciók és deklarációk érvényességének lokalizálására.

Szelekciós utasítások

A program kódjának feltételhez kötött végrehajtását teszik lehetővé.

If utasítás

Valamely utasítás végrehajtását egy feltétel kifejezésértékétől teszi függővé.

- A feltétel kifejezés igaz, ha értéke nem nulla. A kifejezés akkor hamis, ha értéke nulla.
- A feltétel kifejezés egészre konvertálódó logikai kifejezés.

Szintaktikája:

```
if(kif)
{
utasitas1
} else
{
utasitas2
}
```

Megjegyzés: Ifek egymásba ágyazhatók, és else után is használhatunk ifet.

Switch utasítás

Többirányú programelágazást tesz lehetővé olyan esetekben, amikor egy egész kifejezés értékét több konstans értékkel kell összehasonlítani.

Szintaktikája:

```
switch (kifejezés) {
case konstans kif. :
utasítások
case konstans kif. :
utasítások
default :
utasítások
}
```

- A megadható esetek száma implementációfüggő.
- Minden konstans kifejezésnek egyedi értékkel kell rendelkeznie.
- A default megadása nem kötelező; egyébként bárhol elhelyezkedhet a switch utasításon belül.
- A case és a default a címkézett utasítások közé tartozik.

Megjegyzés: Több esethez hozzárendelhető ugyanaz a program rész:

case konstans1:
case konstans2:
utasítások

A switch utasítás – kiértékelése

1. Kiértékelődik a switch utáni kifejezés.
2. A vezérlés átadódik arra a case címkére, amelyben a konstans kifejezés értéke megegyezik a kiértékelt kifejezés értékével. A program futása ettől a ponttól folytatódik, azaz ettől a ponttól kezdve a megadott utasítások sorban végrehajtásra kerülnek!
3. Ha egyik case konstans sem egyezik meg a kifejezés értékével, a program futása a default címkével megjelölt utasítással folytatódik.
4. Ha nincs default címke, a vezérlés a switch blokkját záró } utáni utasításra adódik.

Általában egy adott esethez tartozó programrészlet végrehajtása után ki szeretnénk lépni a switch utasításból (nem folytatni ettől a ponttól kezdve az utasítások végrehajtását).

Alkalmazható vezérlésátadó utasítások:

- break: hatására a vezérlés a switch { } után következő utasításra kerül.
- return: a vezérlés a hívó függvényhez kerül; ha a main függvényben használjuk, a program befejezi futását (a vezérlés az operációs rendszernek adódik vissza).

Címkézett utasítások

A **case** és a **default** utasítások. Használatukat lásd a switch utasításnál.

Vezérlésátadó utasítások

- break: utasításblokkból való kilépésre szolgál; a vezérlés az utasításblokk utáni utasításra adódik
- continue: iterációs utasítások törzsében használva a program működése a soron következő iterációval folytatódik
- return: a vezérlés a hívó függvényhez kerül; a main függvényben használva a program befejezi a futását

Iterációs (ciklus) utasítások

Utasítások automatikus ismétlését biztosító programszerkezet. Az ismétlés addig tart, amíg az ismétlési feltétel igaz (nem nulla).

- A ciklus befejezi működését, amikor a vezérlő feltétel hamissá (nulla) válik. Az a ciklus, amelynek vezérlő feltétele soha nem lesz hamis a végtelen ciklus.
- Ciklusból kilépni a vezérlő feltétel hamissá válása előtt a break vagy a return utasítással lehet.
- A ciklus utasításblokkjának (törzsének) vége előtt a következő iterációra lépni a continue utasítással lehet.
- A vezérlőfeltétel-kifejezés egészre konvertálódó, logikai kifejezés kell legyen.
- A vezérlőfeltétel-kifejezés kiértékelésének helye alapján lehet a ciklus:
  - Elöltesztelő ciklus: a vezérlő feltétel az utasítás végrehajtása előtt kiértékelődik.
  - Hátultesztelő ciklus: a vezérlő feltétel az utasítás végrehajtása után értékelődik ki (vagyis az utasítás legalább egyszer mindig végrehajtódik).

While ciklus

- Addig ismétli az utasításokat (a ciklus törzsét) amíg a vezérlő feltétel (kifejezés) értéke igaz (nem nulla).
- Elöltesztelő ciklus.

Szintaktikája:

```
while(kifejezés)
{
utasitas
}
```

For ciklus

- Akkor használjuk, ha a ciklusmagban megadott utasítást adott számszor kívánjuk végrehajtani (az ismétlések száma adott).
- Elöltesztelő ciklus.
- A kifejezések opcionálisak ; -vel elválasztva.
- A léptető kifejezésben a léptető operátor prefix és postfix alakja is használható.

Szintaktikája:

```
for(init_kif; feltétel_kif; léptető_kif)
{
utasitas
}
```

Megjegyzés: A while ciklus speciális esete, ezért a for ciklus mindig átírható while ciklussá.

Do-while ciklus

Az utasítás végrehajtását követi a kifejezés kiértékelése, ezért legalább egyszer mindig végrehajtódik a ciklusmag (hátultesztelő ciklus). Ha a kifejezés igaz (értéke nem nulla), új iteráció kezdődik; ha hamis (értéke 0) a ciklus befejeződik.

Szintaktikája:

```
do
{
utasitas
} while(kifejezés);
```

A végtelen ciklus

- for (;;) utasítás;
- while (1) utasítás;
- do utasítás while (1);

Vezérlő feltétele mindig igaz.
Általában akkor használjuk, amikor nem tudunk (vagy túl bonyolult lenne) kilépési feltételt megfogalmazni.
A végtelen ciklus megszakításáról (a ciklusból való kilépésről) a programozónak kell gondoskodnia vezérlésátadó utasítások használatával.

Egydimenziós numerikus tömbök tárolása, kezelése

Tömb:

Olyan változók halmaza amelyek azonos típusúak és a memóriában sorfolytonosan helyezkednek el.

- A tömbelemek típusa a void és a függvény típus kivételével tetszőleges típus lehet.
- A tömb elemeire hivatkozni a tömb nevét követő indexelés operátorban megadott elemsorszám (index) segítségével lehet. Például a tömb 5. indexű eleme: tomb[5]

Egydimenziós tömb (vektor) létrehozása (deklarálása):

típus tömbnév[méret];

Egydimenziós tömb egy elemére hivatkozás:

tömbnév[index];

- típus : a tömbelemek típusa
- tömbnév : a tömb neve
- [ ] : indexelés operátor
- méret : a fordító által kiszámítható konstans kifejezés, a tömbben tárolható elemek max. száma (egész)
- index : a hivatkozott tömbelem sorszáma (egész)

Tömb inicializálása (kezdőérték adás) (nem kötelező):

típus tömbnév[méret] = {vesszővel tagolt konstansok};

- Ha az inicializációs lista több elemet tartalmaz, mint atömbméret, akkor fordítási hibát kapunk.
- Ha az inicializációs lista kevesebb elemet tartalmaz, mint a tömbméret, akkor az inicializálatlan elemek értéke 0 vagy határozatlan.

Megjegyzés: A tömb mérete elhagyható, ha az megegyezik az inicializációs lista elemszámával (a fordító a lista alapján automatikusan meghatározza).

Tömbök kezelése

A tömb azonos típusú változókból álló, fix méretű tároló(container). Ezért int tomb[ ]; hibás deklaráció! A tömb méretét ANSI C-ben a program írásakor meg kell adni.

A tömböt csak elemenként lehet kezelni!
C-ben a tömbindexek tartománya 0-tól (méret−1)-ig terjed.

Ökölszabály:
Akkor kell tömböt használni, ha emlékezni kell az összes elemre, vagy azokat nem sorrendben kell feldolgozni.

Mikor kell tömb?

- Elemek fordított sorrendű kiírása
- Elemek rendezett (növekvő/csökkenő) kiírása
- Átlagnál kisebb/nagyobb számok kiírása

Függvény definiálása, deklarálása, függvényhívás mechanizmusa, paraméterátadás

Függvény:

A program olyan névvel ellátott egysége, amely a program más részeiben annyiszor meghívható ahányszor szükség van a függvényben definiált tevékenység-sorozatra.

Függvény definíciója (kód):

<visszatérési típus> fv. neve (<param.dekl. lista>)	Fejléc

{
<lokális definíciók és deklarációk>
<utasítások>
}
	Függvénytörzs

- Saját függvényeinket mindig definiálni kell.
- Egy fv. definíció csak egyszer szerepelhet a pr-ban.
- A programon belül bárhol elhelyezkedhet, kivéve egy másik függvény törzsében.

Lokális változók:

- függvényen belül vannak definiálva, a függvénybe belépéskor jönnek létre
- csak a fv-en belül láthatók → a láthatóságuk (scope) csak a fv-en belülre terjed ki
- mivel minden fv csak a saját lokális változóit látja, nincs névütközés
- a fv. futásának végén a lokális változók megszűnnek (nem foglalnak memóriát) → a változó élettartama (lifetime, storage duration) csak a fv. végrehajtási idejére terjed ki

Vegyük észre!

A függvénynek neve van, amivel a programban több helyen is hivatkozhatunk rá (hívhatjuk), azaz többször elvégezhetjük ugyanazt a részfeladatot (különböző argumentumokon).

A függvény olyan, mint egy program: van bemenete és kimenete is. A fv. bemenete az argumentumlista (a paraméterek aktuális értékei), kimenete a visszatérési érték.

A fv. visszatérési értékét (fv. érték) a return utasítással adjuk meg. Ezzel egyúttal vissza is térünk a fv. hívás helyére! Tehát az utána következő utasítások nem hajtódnak végre.

Függvény visszatérési típusa

- A fv visszatérési típusa a fv által előállított konstans érték (fv. érték) típusát határozza meg. Ez tetszőleges típus lehet, kivéve tömb.
- Ahhoz, h. egy fv. visszaadjon egy értéket, a fv. törzsében ki kell adni a return utasítást. **A fv. által visszaadott érték a return utasításban szereplő kifejezés értéke.** Az utasítás feldolgozásakor (a kif. kiértékelésekor) a fordító - ha szükséges - **implicit típuskonverziót végez**: a kiértékelt kif. típusát a visszatérési típusra konvertálja.
- Ha a fv. definícióban nem adjuk meg a visszatérési típust, **alapértelmezés szerint int** lesz (ANSI C-ben) a fv. érték.

Megjegyzés: A fv. törzsében több return utasítás is szerepelhet!

```
int absz(int a)
{
if (a<0)
return a*(-1);
else
return a;
}
```

Függvény paraméterei

- Cél: a fv. bizonyos belső változóinak a függvény hívás során akarunk értéket adni. Ezeket a változókat a fv. paraméterlistájában adjuk meg.
- Az ANSI C szabvány szerinti fv. definícióban a (formális) paraméterlista a paraméterek típusát és nevét tartalmazza (ezeket space választja el). A listában szereplő paramétereket vessző választja el.
- A paraméterek tetszőleges típusúak lehetnek, és a fv-en belül mint a fv. lokális változói használhatók, de a fv-en kívülről nem érhetők el.

Fv paraméter: kívülről inicializált lokális változó

- A fv. hivatkozásában (hívásakor) megadott aktuális paraméterlistát argumentumlistának nevezzük.
- A fv. paraméterlistája és argumentumlistája között típusegyeztetés történik, ezért a paraméterek megadási sorrendje rögzített. A fordító ellenőrzi a paraméterek számának, típusának egyezését (de nem kell h. a nevük is egyezzen).

Függvény deklarációja, prototípusa

A fv. deklarációja a fv. nevét, visszatérési értékének típusát és a paraméterek típusát tartalmazza.
Ha a fv deklarációja a paraméterek nevét is tartalmazza, akkor ezt a fv prototípusának nevezzük. Ezt mindig a fv. hívás előtt kell elhelyezni a programban (nem hiba, ha többször szerepel).

Megjegyzés: Ha egy fv. definíciója megelőzi a hívás helyét, akkor nincs szükség a prototípus külön megadására.

Függvény deklaráció:

visszatérési_típus fvnév(tipus1, tipus2, ...);

Függvény prototípus:

visszatérési_típus fvnév(paraméterlista);

Függvény hívása

fvnev(argumentumlista);

A fv. hívás olyan kifejezés, amely átadja a vezérlést és az argumentumokat a hívott fv-nek.

Ha a fv. prototípusában a paraméterlista helyén a void kulcsszó áll (vagy üres), akkor a fv. argumentumlistája üres, de híváskor a zárójeleket így is ki kell tenni.
Return utasítás után a függvény visszatér a hívó helyére, és a visszatérési érték az a fv. hívás kifejezés értéke.