Olvasólecke az **Informatika a fizikában 2.** kurzushoz	Szerző: **Majorosi Szilárd** (TTIK Elméleti Fizikai Tanszék)	Olvasási idő: **35-40 perc**.

FÜGGVÉNYEK ÉS SZELEKCIÓK

1. Függvények és standard vezérlés

Ebben az olvasóleckében a C nyelv vezérlésének két alapvető eleméről lesz szó, amelyek lehetővé teszik a kód egyik pontjából egy másik kódrészlet végrehajtását (függvények), illetve a vezérlésben elágazások tehetők segítségükkel (szelekciók).

1.1. Kódblokk

C-ben több utasítást az ún. **kódblokk** (scope) foglal egységbe, amely gyakorlatban azt jelenti, hogy {} kapcsos zárójelek közé standard C utasítások sorozatát írjuk:

```
{ utasitas1; utasitas2; ...; utasitasn; }
```

Amennyiben nincs más külön kijelentve (`kulcsszó`, #direktiva), a pontosvesszővel elválasztott utasítások egyesével hajtódnak végre, sorban (balról jobbra, fenntről lefele). Kódblokkhoz külön tartozik memóriaterület, az úgynevezett **verem** (stack). Itt a memóriafoglalásokat a C automatikusan intézi. *A kódblokkok egymásba ágyazhatók* (egy utasítás helyett), azaz több szintet képezhetnek, hozzájuk vezérlési szerkezet társulhat.

A kódblokkok határt is képeznek, ezért csak olyan változókhoz (és függvényekhez) **férünk hozzá** amit már:

- adott utasítás előtt *deklaráltunk* ugyanabban kódblokkban
- adott utasítás előtt *deklaráltunk* egy vagy több szinttel feljebb lévő kódblokkban ($\rightarrow$ függvények, globális változók)

C nyelvben minden kódblokkon kívül definiált változókat *globális változóknak* nevezzük, azonban ezek alapból csak az adott fordítási egységből nézve globálisak. Ez azt jelenti, hogy a forráskódban minden alattuk lévő kódblokkban (függvényekben) hozzáférhetők. [1]

[1] Amennyiben a programunk több fordítási egységből áll, meg kell tanulnunk az `extern` és `static` kulcsszók helyes használatát is.

1. Forráskód. Példa kódblokkokra, vezérlési szerkezet nélkül.

```c
#include <stdio.h>   //Kiíratáshoz kell
double a = 1.0; //ez egy globális változó
int main(){ //függvényhez tartozó kódblokk
        double x = a; //hozzáférünk efeletti globális változókhoz
        { //beágyazott kódblokk
           double y = 6.1; //beágyazott utasítás
           x = y; //külső kódblokkbeli változókkal tudunk műveleteket végezni
        }
        printf("%lg", x); //itt csak a-t és x-et tudjuk kiírni
}
```

Kimenet: 6.1

1.2. Függvénydefiníció

Egy **függvény** önálló programrészletet definiál (ún. eljárást) *szabadon választott névvel*, amely függvényhívással máshonnan is végrehajtható. A nevéből sejthető, hogy a működésük hasonló a szokásos matematikai függvényekéhez. Egy standard C függvény definíciója általános esetben a következőképp nézhet ki:

```c
típus fuggvenynev(típus1 arg1, ..., típusn argn){
     kód;
     return visszateresi_ertek;
}
```

A használatához a következőket érdemes még észben tartani:

- A függvényhívás eredménye a visszateresi_ertek, aminek típusa **típus**.
- Rendelkezhet **bemeneti argumetumokkal**: ezeket felsorolva a ()-ben kell megadni vesszővel elválasztva. Az argumentumok olyan változók, amelyeket a kódblokk számára deklarálunk.
- A függvények kódblokkját **függvénytörzsnek** (function body) is nevezzük.
- A függvénydefiníciók minden kódblokkon kívül vannak, azaz globálisak. Egy programban egy függvénydefiníció **ugyanazon a névvel** csak **egyszer** szerepelhet (több fordítási egység esetén is)!

Ha a függvénynek nem akarunk adni visszatérési értéket (és **return** utasítást), akkor függvénydefiníciójába a **típus** helyére azt írjuk, hogy **void**. Ez azt jelenti, hogy a **void** egy olyan speciális kulcsszó, amely nem meghatározott típust jelöl, önállóan változóval nem állhat.

1.3. Függvénydeklaráció

A függvények deklarációja csupán annak kijelentése, hogy az adott függvény definíciója létezik, a fordítási egységeink valamelyikében. Kijelentése a következő alakú:

```
típus fuggvenynev(típus1 arg1, ..., típusn argn);
```

Általában fejlécekben (header) használatos. Természetesen az adott forráskódban lévő függvénydefiníció ott deklarációnak is számít.

1.4. Függvényhívás

A függvényhívás utasítása szükséges egy adott függvény törzsének tényleges végrehajtásához.

```
fuggvenynev(valtozo1, ..., valtozon);
```

A fenti függvényhíváskor a következők történnek:

- A vezérlés a függvény kódblokkjára ugrik, de előtte a `valtozo1,...` értéke átmásolódik a függvénydefiníció `arg1,...` változóiba (érték szerinti átadás).
- Végrehajtódnak a függvény kódblokkjában lévő utasítások.
- A vezérlés visszakerül a függvényhívás eredeti pontjára, odamásolva a `visszateresi_ertek` értékét a megfelelő típusban (érték szerinti visszatérés). Ezután ezzel az értékkel szokásos módon hajthatók végre műveletek.

Természetesen, ha ilyen függvényhívást végeznénk, akkor minimum a megfelelő függvénydeklarációnak szerepelnie kell a forráskódban a hívás felett. Ezenfelül magának a függvény definiciójának is léteznie kell, különben a linkelési fázisban fogunk hibaüzenetet kapni (”undefined reference to”).

2. Forráskód. Ebben a példában írtunk egy olyan függvényt, amely *két szám számtani közepét számolja ki*. Ezután ennek a függvénynek a meghívásával számoltunk.

```c
#include <stdio.h>   //Kiíratáshoz kell
//Ez a függvény a,b  valós szám számtani közepét számolja ki
double szamkozep(double a, double b){
    double kozep = 0.5*(a+b); //kódblokkban szokásosan szereplek az utasítások
    return kozep; //itt ér véget a függvény, kozep változó értékével
}
int main(){
        double a = 10., b = 0.; //ezek adják meg a konkrét értékeket
        double c = szamkozep(a, b); // függvényhívás és visszatérés
        printf("%lg", c);
}
```

Kimenet: 5

1.5. Beépített függvények

Nem csak mi írhatunk függvényeket, hanem már jó pár használatra készen áll a C **standard könyvtárában** (standard library). Az itt felsoroltak a programunk alapvető feladatainak ellátásához szükségesek. Amennyiben használni akarjuk őket a forráskódunkban, akkor annak elejere kell írni az `#include` direktívát, és utána a kiszemelt fejléc fájlt. Például:

- **<stdio.h>**: Standard input és output függvények (→ link)
  - képernyőre íratás – `int` **printf( );** – nem standard C függvény
  - billentyűzetről beolvasás – `int` **scanf( );** – nem standard C függvény
  - stb.
- **<math.h>**: Matematikai függvények (→ link)
  - hatványozás – `double` **pow(**`double ,double`**);**
  - gyökvonás – `double` **sqrt(**`double`**)**;
  - abszolútérték – `int` **abs(**`int`**);** `double` **fabs(**`double`**);**
  - exponenciális függvény – `double` **exp(**`double`**);**
  - *e*, 10 alapú logaritmus: `double` **log(**`double`**);** `double` **log10(**`double`**);**
  - trigonometrikus függvények
    `double` **sin(**`double`**);** `double` **cos(**`double`**);** `double` **tan(**`double`**)**
  - és inverzeik
    `double` **asin(**`double`**);** `double` **acos(**`double`**);** `double` **atan(**`double`**)**
  - hiperbolikus függvények ...

  Amennyiben terminálból fordítunk, akkor a matematikai könyvtár használatához a **fordításnál linkelni** is kell: **gcc** *fájlnév*.c **-o** *out_név* **-lm**

Ezen kettő standard könyvtár használata gyakorlatilag alapvető, különösen számolási feladatok elvégzéséhez - a forráskódjainkba gyakorlatilag mindig becsatolhatjuk őket.

3. Forráskód. Ebben a példában írtunk egy olyan függvényt, amely *két szám mértani közepét számolja ki*. A kód nincs tekintettel a negatív előjelre.

```c
#include <math.h> //Matematikai függvények használatához kell
double geomkozep(double a, double b){
    double kozep = sqrt(a*b); //sqrt() függvény hívása
    return kozep; //itt ér véget ez a függvény, kozep változó értékével
}
```

2. Szelekciók

A C nyelvben a **szelekciók** a vezérlési szerkezetek egy csoportját képezik, ezek lehetővé teszik, hogy a programunk vezérlésébe döntéseken alapuló **elágazásokat** tegyünk, bizonyos feltételeknek megfelelően. A következő alfejezetben azonban egy operátorról lesz szó, amely bár nem a szó szoros értelmében szelekció, viszont hasonló feladatok oldhatóak meg vele.

2.1. ? : feltételes operátor

Az úgynevezett **? :** feltételes operátor egyszerűbb számolások esetén, kompakt módon képes megvalósítani elágazásokat. Szintaxisa a következő:

```
feltetel ? ertek1 : ertek2;
```

Ez az operátor három argumentummal rendelkezik, viszont *egy* határozott eredményt ad:

- Ha a `feltetel` igaz (= nem 0), akkor az eredmény `ertek1`;
- Ha a `feltetel` hamis (= 0), akkor az eredmény `ertek2`;

Alapból ez az aritmetikai operátorok után van a műveleti sorrendben. Általában változók értékeinek kiválasztására használatos.

4. Forráskód. Példa *abszolútérték számolására* a feltételes operátorral.

```c
#include <stdio.h>   //Kiíratáshoz kell
int main(){
        double x = -3.; printf("%lg", (x < 0 ? -x : x) );
}
```

Kimenet: 3

2.2. if . . . else szerkezetek

Az **if** és **else** kulcsszók által alkotott vezérlési szerkezetek egy, kettő vagy akár többirányú elágazások megvalósítására valók. Minden elágazás saját kódblokkal rendelkezik, amelybe újabb utasítások írhatók.[2]

if – *ha igaz*, akkor ez történjen

Az `if` kulcsszó által alkotott önálló vezérlési szerkezet a következő:

```
if (feltetel) {
       utasitas1; utasitas2; ...;
}
```

Jelentése: ha `feltetel` igaz (= nem 0), akkor az `if` utáni kódblokk végrehajtódik. Ezután a vezérlés a szerkezetből kilép.

else – *ha nem volt igaz*, akkor ez történjen

Az `else` kulcsszó által alkotott egyszerű vezérlési szerkezet:

```
else{
       utasitas1; utasitas2; ...;
}
```

Közvetlenül egy `if`-es kifejezés után kell írni, azt egészíti ki a következő módon: ha az előtte lévő `if`-es kifejezésben egy feltétel sem teljesült, akkor az `else` utáni kódblokk végrehajtódik. Ezután a vezérlés a szerkezetből kilép.

else if – többirányú elágazások

Egy szimpla **if**-es szerkezet tovább kiegészíthető **else if**-es szerkezet hozzáadásával. Általánosan a következővé:

[2]Ezen vezérlési szerkezetek szintaxisát még egyszerűsíteni lehet, amennyiben bármelyik hozzájuk tartozó kódblokk csak *egy* utasításból áll: ekkor az adott kódblokkokból elhagyható a határóló kapcsos zárójel.

```
if (feltetel1)
       {  utasitas1;   utasitas2; ...;      }
else if (feltetel2)
       {  utasitas1;   utasitas2; ...;      }
       :
else if (felteteln)
       {  utasitas1;   utasitas2; ...;      }
else
       {  utasitas1;   utasitas2; ...;      }
```

Az **else if**-es szerkezet logikája a következő: ha az adott `else if` előtt lévő `if`/`else if`-es kifejezésekben egy feltétel sem teljesült, és az utána lévő argumentumban lévő `feltetel` igaz (= nem 0), akkor az adott `else if` utáni kódblokk végrehajtódik. Ezután a vezérlés a szerkezetből kilép. A bővített szerkezetben szereplő csak az `if` és `else` kulcsszavakkal jelzett ágak jelentése nem változik.

5. Forráskód. Ebben a példában írtunk egy olyan függvényt, amely *két szám mértani közepét számolja ki*. Ha negatív számból vonunk gyököt hibaüzenetet kapunk, ha mindkét szám negatív volt, akkor a számtani közép is negatív lesz.

```c
#include <stdio.h>   //Kiíratáshoz kell
#include <math.h>    //Matematikai függvények használatához kell
double geomkozepneg(double a, double b){ //elágazásos geometriai közép
   double kozep = 0.;     //beírjuk külön változóba
   if(a >= 0 && b >= 0){//esetek: a és b is nemnegatív - rendben
        kozep = sqrt(a*b);
   }
   else if(a < 0 && b < 0){//a és b is negatív -  az érték legyen negatív
        kozep = -sqrt(a*b);
   }
   else{//minden más esetben hiba
        printf("Hiba! ");
        return 0.; //ide is rakhatunk return utasítást
   }
   return kozep;     //eredmény visszaadása
}
int main(){
       printf("%lg ", geomkozepneg(-2,-8)); printf("%lg ", geomkozepneg(2,-8));
}
```

```
Kimenet:  -4 Hiba!  0
```

2.3. switch . . . case szerkezet

Ez a vezérlési szerkezet csak egy kódblokkból áll, és abban *az esetben használható, ha egy változó érékét több konstanssal kell összehasonlítani, viszont ekkor többirányú elágazást is meg lehet vele valósítani*. Ehhez a vezérlési szerkezethez négy új kulcsszó járul, és általánosan a következőképp írható:

```
switch (valtozo){
     case ertek1:
          utasitas1;   utasitas2; ...; break;
     case ertek2:
          utasitas1;   utasitas2; ...; break;
       ⋮
     case ertekn:
          utasitas1;   utasitas2; ...; break;
     default:
          utasitas1;   utasitas2; ...;
}
```

Ez a szerkezet a C programozás egy olyan vezérlésére épül, amelyről eddig nem esett szó: névszerint egy kódblokkban a program futása adott helyre is ugorhat, bizonyos nyelvi elemek segítségével. A **switch** szerkezet szintaxisa ezt a megközelítés tükrözi, és a következőképp működik:

1. Amikor belépünk a **switch** szerkezetbe, akkor az argumentumába írt változó értéke kimásolódik. Ezen *érték alapján esetkiválasztás történik*, a további pontok szerint.
2. Amennyiben `valtozo` értéke megegyezik az egyik előre beírt értékkel (`ertek1`, ..., vagy `ertekn`), akkor a program vezérlése a megfelelő értékű **case** . . . : *helyére ugrik*.
3. Amennyiben `ertek1`, ..., vagy `ertekn` értékkel sincs egyezés, akkor a vezérlés **default:** *helyére ugrik*. Ebben az előbbi esetben, ha nincs **default**, akkor a vezérlés a szerkezetből azonnal kilép.
4. Az ugrás után, az adott ponttól elkezdve minden egyes a szerkezett kódblokkjába írt **utasítás** *sorban végrehajtódik*.
5. A **break** kulcsszó *egy speciális utasítást jelez*: amikor a vezérlés ráfut az egyikre, akkor a vezérlés a **switch** szerkezetből azonnal kilép.

Miután az értékkiválasztás megtörtént (és a vezérlés ugrása) a **case** . . . : és a **default:** szintaktikai elemeket a vezérlés ignorálja. Ennek megfelelően ezek nem utasítások, hanem csak azoknak a pontoknak a helyét jelzi, hogy hova ugorhat a vezérlés. Ha nem írunk **break** utasításokat a fenti szerkezetbe, akkor a vezérlés akár az összes beleírt utasítást végrehajthatja. Természetesen, a fenti **switch** szerkezet működése **if else** szerkezettel is megvalósítható. [3]

[3] Ez nem azt jelenti, hogy a kétféle megoldás a fordítás után a futtatható állományban is teljesen azonos lesz. Mivel a **switch** szerkezet csak egy kódblokkból áll, és egy adott változó értékét már a fodítási időben ismert értékekkel kell összehasonlítani, ezért az némileg hatékonyabb. A gyakorlatban néhány esetet kivéve – ennek nincs jelentősége.

6. Forráskód. Ebben a példában írtunk egy olyan függvényt, amely a C nyelv logikája szerint *eldönti egy egész számról, hogy az igaz vagy hamis, és kiírja ezt a konzolra.*

```c
#include <stdio.h>     //Kiíratáshoz kell
//Kiírja egy egész számról, hogy az igaz vagy hamis a Boole algebra szerint
void printbool(int num){
     switch(num){
          case 0: //0 érték jelenti azt, hogy hamis
               printf("false"); break;
          case 1:  //1 érték jelenti azt, hogy igaz
          default: //minden más értékre is érvényes
               printf("true");
     }
}
//Innen kezdődik a program futása
int main(){
    printbool(0); printf(", "); printbool(1); printf(", "); printbool(-23);
}
```

```
Kimenet:  false, true, true
```

Ellenőrző kérdések

1. Egy C programban az utasítások milyen sorrendben hajtódnak végre?
2. Miért hasznos a C nyelvben függvényt írni?
3. Mit nevezünk érték szerinti visszatérésnek?
4. Két valós változó közül a nagyobb meghatározására a `?:` operátor vagy az `if-else` szerkezet az egyszerűbb?
5. Milyen vezérlési feladatot látnak el a szelekciós szerkezetek?
6. Szükséges-e az `if-else`-es szerkezetekbe beépítenünk mindig az `else if` kulcsszót?
7. Mi történik akkor ha két `if`-es kifejezést egymás után írunk?
8. Mi a jelentése `case`, `default` és `break` kulcsszavaknak egy `switch` szerkezetben?

Gyakorló feladatok

1. Írjunk egy programot, amely három a, b, c valós számról eldönti, hogy azok egy háromszög oldalai-e. Ha igen, kiírja a háromszög területét Héron-formulával. Megoldáshoz használjuk a további feladatokban megírt függvényeket.
2. Írjunk egy függvényt, amely eldönti, hogy a, b, c szakaszból szerkeszthető-e háromszög. Egy háromszög bármely oldalának hossza kisebb a másik két oldal hosszának összegénél.
3. Írjunk egy függvényt, amely kiszámítja egy tetszőleges háromszög területét az oldalhosszak ismeretében! Erre szolgál a Héron-formula:

$$T = \sqrt{s\,(s-a)\,(s-b)\,(s-c)},\ \text{ahol}\ s = \frac{1}{2}(a+b+c)$$

   - Teszteljük a függvényt derékszögű háromszöggel.

Kiegészítések a kiválósághoz

1. A C nyelvben a vezérlésben ugorni `goto` kulcsszó alkalmazásával is lehet, azonban annak használatát általában rossz programozási megoldásnak tartják. Nézzünk utána, hogy mit is csinál a `goto`, és miért nem tanácsos használni!
2. A standard könyvtárban több helyen találhatók igen hasznos függvények és eszközök. Ezek például <time.h>, <stdlib.h>, <string.h>. Nézzünk ezeknek utána! Mellőzhetjük az újabb C99, vagy annál későbbi fejléceket.

JELEN TANANYAG
A SZEGEDI TUDOMÁNYEGYETEMEN KÉSZÜLT,
AZ EURÓPAI ÚNIÓ TÁMOGATÁSÁVAL.
PROJEKTAZONOSÍTÓ: EFOP-3.4.3-16-2016-00014.