

### 3. gyakorlat

Az eheti gyakorlaton a matematikai függvényekről, az elágazásokról (if és switch), a typedefről, és a „bitvarázsolásról” esett szó.

#### Elméleti alapok az eheti gyakorlathoz

##### Elágazások

A programvégrehajtás az utasításokon felülről lefelé halad végig, egymás után elvégzi az összes utasítást. Gyakran azonban egy bizonyos feltétel teljesülése esetén akarjuk végrehajtani az utasításokat. A programvégrehajtás az elágazásnál megvizsgálja a feltételt, és annak megfelelő úton halad tovább.

##### Matematikai függvények

A *standard library* matematikai függvényeit a `<math.h>` könyvtárból érjük el.

```
#include <math.h>
```

Elérhető függvények például

- trigonometrikus és hiperbolikus függvények
- `pow()` hatványozás
- `floor()`, `ceil()`, `round()` kerekítésre
- `abs()` egész számokra
- `fabs()` float vagy double számokra
- `log10()`
- `exp()`
- `sqrt()`

**Vigyázat!** Minden szögfüggvény radiánban számol.

##### Példa

```
double terfogat = pow(oldalhossz, 2);  
int mernok_pi = round(3.14159265);
```

##### Kétirányú elágazás - if

A program az elágazáshoz érve egy **feltétel** alapján választ **két út közül**. A feltétel lehet **igaz** vagy **hamis**, és eszerint végrehajtja az utasítást, vagy átugorja.

A feltétel teljesülése esetén végrehajtja a következő utasítást. Ha a feltétel nem teljesül, átugorja az utasítás, és az utána következő kóddal folytatja a végrehajtást.

```
if (feltétel)  
    printf("igaz út");
```

Amennyiben több utasítást is feltételesen szeretnénk végrehajtani, egy utasításblokkba fűzhetjük őket. Az utasításblokk jele a kapcsos zárójel. Az összekapcsolt utasításokra együttesen vonatkozik a feltétel.

```

if (feltétel)
{
    printf("igaz út");
    printf("további utasítás");
    ...
}

```

**Megjegyzés** A C nyelvben az utasítások helyén bárhol használhatunk utasításblokkot - erre leginkább az elágazások, ciklusok és függvények esetén van szükség.

Megszabhatjuk, hogy a feltétel nem teljesülése esetén egy másik blokk kerüljön végrehajtásra. Az `else` kulcsszóval definiálhatjuk a hamis utat.

```

if (feltétel)
{
    printf("igaz út");
    ...
}
else
{
    printf("hamis út");
    ...
}

```

A feltétel egy logikai kifejezés, melynek eredménye 0 (hamis) vagy 1 (igaz). Leggyakrabban összehasonlító, logikai operátorokkal állítjuk elő.

Az összehasonlító operátorok:

- < kisebb, > nagyobb
- <= kisebb vagy egyenlő, >= nagyobb vagy egyenlő
- == egyenlő
- != nem egyenlő

A logikai operátorok:

- ! negálás
- && logikai és
- || logikai vagy

## Példa

```

if (gyujtoszamla_egyenleg < 0)
{
    printf("A tárgyfelvétel az ön számára nem engedélyezett.")
}
else
{
    printf("A tárgy felvétele sikeres :)");
}

```

**Megjegyzés** Az utasítást ne írjátok ugyanabba a sorba, mint a feltételt, mert úgy ronda lesz a kód. Az `else` ágat meg végképp ne. Bár a fordító megengedi az egész egy sorba

írását, ez általában rossz ötlet.

## Egymásba ágyazás

Az if struktúrákat egymásba lehet ágyazni, az egyiken belül kezdhetsz egy következőt.

```
if (a_neved == "Lancelot de Genere Camelot.")
    if (amit_keresel == "Én biz' a Szent Kelyhet.")
        if (kedvenc_szined == "A kék.")
            printf("Rendben, átmehetsz.");
```

Ennek egy fajtája az `else if`, amikor a hamis ágban vizsgálunk egy újabb feltételt.

```
if (fecske_fajta == "afrikai")
    sebesseg = 80;
else if (fecske_fajta == "európai")
    sebesseg = 76;
```

*Megjegyzés* Vizsgálhatunk bármilyen feltételt az `else if`-ben, de leggyakrabban ugyanazt a változót vizsgálják, mint a fenti `if`-ben. Túl sok eset ( $>3$ ) vizsgálata esetén már érdemesebb `switch`-et használni.

## Ternary operátor

Az ún. ternary operátor egy **feltétel** alapján választ **két érték** közül. Az érték helyére egy kifejezést is írhatunk, pl. matematikai formulát, függvényt.

```
feltétel ? érték_ha_igaz : érték_ha_hamis;
```

*Megjegyzés* A ternary operator kényelmi funkció. Ugyanezt egy `if` struktúrával is el lehet érni.

`printf`-et ne írjunk bele, nem arra való.

## Példa

```
int fizetendo = (vizsgan_atment == 0) ? 5500 : 0;
int fizetendo = (vizsgan_atment == 0) ? 5500 + 2500 : 0;
```

Ha a `vizsgan_atment` változó értéke hamis (0), akkor a fizetendő összeg 5500 Ft. Ha értéke nem hamis, akkor 0 Ft-ot kell fizetni. Az értékek helyére kifejezést is írhatunk.

## Többirányú elágazás - switch

Switch struktúrát akkor használunk, ha **ugyanannak a kifejezésnek** különböző eredményei esetén akarunk különböző dolgokat csinálni. A sok `else if` ággal rendelkező `if` struktúra kiváltható `switch`-csel.

```
switch (kifejezés)
{
    case érték1:
        utasítás1;
        break;
    case érték2:
```

```

        utasítás2;
    break;
default:
    utasítás0;
}

```

A switch struktúra **default** ágában megadhatjuk, mi történjen, ha egyik eset (**case**) sem áll fenn.

A switch a teljesült case alatti összes többi case-t is lefuttatja. Ezt általában nem szeretnénk, ezért minden case-be teszünk egy **break** utasítást. Ez azonnal kilép a switch struktúrából.

## Példa

Ez a program bizonyos billentyűk bevitelekor válaszol nekünk.

```

char betu;
printf("Nyomj meg egy betűt! ");
scanf("%c", &betu);

switch (betu)
{
    case 'h':
        printf("Hello!\n");
        break;
    case 'g':
        printf("Goodbye!\n");
        break;
    case 'j':
        printf("JOE!\n");
        break;
    default:
        printf("no command for %c", betu);
}

```

## Bitműveletek

A bináris számokon végezhetünk olyan aritmetikai műveleteket, amiket a decimális világból nem ismerünk. A négy alap bináris művelet (AND, OR, NOT, XOR) természetes számokra elvégezhető, ekkor minden bitjükre párosával végrehajtódik a művelet.

### AND

ÉS művelet, más néven *bit reset*. Bármelyik számban szereplő nullák helyén nullázza az eredményt.

```

char a = 0b00010001; // így adunk meg binárisan számot
char b = 0b00010010;
char c = a & b;

```

A c változó értéke 0b00010000 lesz, mert csak az 5. biten van mindkét számban 1-es számjegy.

## OR

VAGY művelet, más néven *bit set*. Bármelyik számban lévő egyesek helyén 1-re változtatja az eredményt.

```
char a = 0b00010001;  
char b = 0b00010010;  
char c = a | b;
```

A c változó értéke 0b00010011 lesz, mert ezeken a biteken van legalább az egyik számban 1-es számjegy.

## NOT

NEGÁLÁS művelet. Megfordítja a biteket.

```
char a = 0b00010001;  
char c = ~a;
```

A c változó értéke 0b11101110 lesz, minden biten megfordul a számjegy értéke.

## XOR

KIZÁRÓ VAGY művelet. Az eredmény ott lesz 1, ahol a két számban különböző számjegy van.

```
char a = 0b00010001;  
char b = 0b00010010;  
char c = a ^ b;
```

A c változó értéke 0b00000011 lesz, mert az 1. és 2. biten csak az egyik számban van 1 számjegy, az 5. biten viszont mindkettőben.

**Megjegyzés** Emlékezzünk rá, hogy a `char` adattípus az egybájtos pozitív egész számmal egyezik.

## Bit shift

Van két operátor, amelyek a szám bitjeit jobbra vagy balra tolják. Az összes bit adott lépéssel jobbra ill. balra kerül, a megüresedett helyet pedig 0-k töltik fel.

```
char a = 0b00010001;  
char bal1 = a << 1;  
char bal2 = a << 2;  
char jobb1 = a >> 1;
```

A `bal1` változó értéke 0b00100010 lesz, a `bal2` változó értéke pedig 0b01000100. A `jobb1` értéke 0b00001000, az utolsó 1 számjegye eltűnt.

Mikrovezérlőknél igen gyakran előkerülnek a bitműveletek, érdemes őket megjegyezni.

## Typedef

A typedef kulcsszóval új típusokat hozhatunk létre. Az `enum` és a `struct` összetett adattípusoknál van értelme használni, de itt egy egyszerű példán megnézhetjük a használatát.

```
typedef char byte;
```

```
byte a = 0xFF; // hexában is megadhatjuk az értékét
```

A typedef egy alternatív nevet ad az adott típusnak. Ezután az alternatív típusnévvel is használhatjuk. Természetesen a char típusra alternatívát adni értelmetlen, de egy struct esetén már van haszna. Álljon itt egy példa illusztrációként, érteni még nem kell.

```
struct Book {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
}  
typedef struct Book Book;
```

A struct Book típust átnevezzük egyszerűen Book típusra.

## Feladatok

**1. Készíts programot, ami bekér egy szöveget, és kiírja annak szinuszát, koszinuszát, tangensét.**

A szöveget radiánban kelljen megadni.

*Segítség* Ezekre kell emlékezni:

- include-old a <math.h> könyvtárat
- sin(), cos(), tan()
- scanf("%i", &valtozo)
- double típusú változóba olvasd be
- double formátumban olvasd be ("%lf")

**2. Készíts egy olyan kő-papír-olló programot, ami mindig nyer.**

Bemenet: egy karaktert beírva lehet választani: kő k, papír p, olló o.

A bemenetet alapján a számítógép is választ követ, papírt vagy ollót.

**3. Bitvarázsoljunk!**

Adott a következő bájt:

```
char adat = 0b00101110; // 46d
```

Kiírni hexában lehet, binárisan nem. Lehet gyakorolni a hexa -> bináris átváltást (vagy megnyitni a számológépet).

```
printf("adat = %#x", adat);
```

**3.1. Állítsd be az 5. bitet 1-re!**

A biteket jobbról balra számoljuk, 1-től 8-ig.

**3.2. Változtasd meg a 8. bitet az ellenkezőjére!**

**3.3. Töröld ki az alsó négy bitet (állítsd 0-ra)!**

**3.4. Változtasd át az összes bitet az ellenkezőjére, a legegyszerűbb módon!**