Tartalomjegyzék

Ismétlés

- Fordítás **gcc**-vel:

```
gcc -W -Wall -o hello hello.c
```

- A program futása a main függvénnyel kezdõdik, ezáltal mindig kell, hogy létezzen.
- Pontosvesszõk a sor végén.
- Blokkok kapcsoszárójelekkel.
- A változókat deklarálni kell (meg kell adni a típusukat).
- stdio.h-ban hasznos függvények, pl: printf
- Egy példa printf-re:

```
printf("A %d nagyobb mint a %d\n.", 3, 2);
```

- A 3 és a 2 sorban be lesz helyettesítve a string-be, a \n a sorvége karakter.

- Az if a szokásos módon mûködik, a szintaxis:

```
if(feltetel){
    utasítások
} else if(feltetel){
    utasítások
```

```
}
...
} else{
    utasítások
}
```

A gyakorlat anyaga

Beolvasás

- A programunk felhasználójától beolvashatunk adatokat **elõre deklarált változókba**. Ezt egyszerûen a **scanf** függvénnyel tehetjük meg (stdio része).
- A **scanf** elsõ paramétere egy string, mellyel megadjuk, hogy milyen minta szerint érkezik a beolvasandó adat. Legegyszerûbb esetben, ez lehet egy "http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, azaz egy egész számot olvasunk be.
- A további paraméterek a már deklarált változók pointerei, errõl késõbb részletesen lesz szó, jelenleg legyen elég annyi, hogy a változók elé egy & jelet kell tenni.

- Egy példa a **scanf** használatára:

```
int z;
printf("Add meg z értékét: ");
scanf("%d", &z);
```

- Amint a **scanf**-hez ér a program a terminálban megjelenik egy kurzor és bementet vár a felhasználótól. A **z** változónk felveszi ezt a megadott értéket.

- Egy scanf-el több adatot is bekérhetünk egyszerre, de az átláthatóság kedvéért, ezt csak olyan esetekben tegyük, ahol logikus, hogy egyszerre több adat érkezik (pl fix méretû mátrix egy sora).

Ciklusok

For ciklus

- A for ciklus nem a sage-ben megszokottak szerint mûködik, hisz itt nincsenek listáink amiket bejárhatnánk.
- Ehelyett a for a while ciklushoz nagyon hasonlóan mûködik, a szintaxis:

```
for(inicializálás; feltétel; inkrementálás){
    utasítások
}
```

- Az inicializálás részben adhatjuk meg azokat az utasításokat amiket csak egyszer a ciklus kezdetekor szeretnénk végrehajtani, ha úgy gondolunk rá mint egy szummára, akkor ez lehet az i = 0 például.
- A ciklus akkor áll le, amikor a feltétele hamis lesz, tehát amíg igaz, addig fut.
- Az inkrementálás lépés a ciklus belsejében levõ utasíások (a ciklus magja) után hajtódik végre. Itt tipikusan növelünk egy ciklusváltozót, de akármi mást is lehetne csinálni.

- Példa egy for ciklusra, ami kiírja a számokat 0-tól 9-ig:

```
int i;
```

```
for (i=0; i<10; i++) {
    printf("A ciklusváltozo erteke: %d\n", i);
}
```

- Az i++ egyenértékû az i = i + 1 vagy az i += 1 utasítással.
- Példa egy sokkal kevésbé hagyományos for ciklusra:

```
int i = 1;
int j = 1;
for (; i + j != j * 2; i = j + 1) {
    j = i * 2;
}
```

- Ez a példa azt is mutatja, hogy a ciklus fejének különbözõ részei egymástól függetlenül elhagyhatók, valamint hogy létezik végtelen ciklus.

While ciklusok

- A while ciklus a már sage-ben megszokottak szerint mûködik, addig fut, amíg igaz a fejében található feltétel.
- Talán annyi plusz van, hogy bevezetjük a hátul tesztelõs ciklust, a do while ciklust. Ez a nevébõl is kikövetkeztethetõen utólag tesztel, tehát a magja legalább egyszer lefut.
- A szintaxis:

```
while(feltétel){
    utasítások
}

do{
    utasítások
} while(feltétel);
```

- Példa egy bonyolultabb while ciklusra, a feladatokban lesz egy hozzá nagyon hasonló:
- A felhasználótól egy ciklusban egész számokat kér be addig, amíg 0 értéket nem kap. Ekkor pedig kiírja a képerny?re a kapott nemnulla számok átlagát.

```
#include<stdio.h>

int main(void) {
    int i = 0;
    float sum = 0;
    int szam = 0;

    do{
        scanf("%d", &szam);
        sum += szam;
        i++;
    } while(szam != 0);

    i--;
    printf("%f", sum / i);

    return 0;
}
```

- Bejött egy új típus, a float, õ a lebegõpontos szám, mondhatjuk, hogy a tizedestört. A printf és

scanf-ben használandó mintája a "http://wiki.math.bme.hu%f"http://wiki.math.bme.hu.

Feladatok

1. For ciklus

Egészítsd ki a következ? kódot a megjegyzések helyén! Adjuk össze a számokat 1-tõl kezdve, egyesével, pl: 1 + 2 + 3 = 6. A programunk azt a számot adja ki, hogy az elsõ hány darab számot kell összeadni, hogy legalább 4212-t kapjunk.

```
#include<stdio.h>

int main(void) {
    /* változók deklarálása, értékadás */
    /* i-t is deklarálni kell ! */

    for (/* inicializálás */ ; /* feltétel */; /* minden ciklusmag végén */) {
        /* számolás */
    }
    /* kiírás */
    return 0;
}
```

Ha ezt sikerült megoldani, akkor egészítsük ki úgy a kódot, hogy a 4212-nek megfelelõ számot a for ciklus elõbb scanf-el kérjük be a felhasználótól.

2. Do / while ciklusok

Írjunk egy korábbi példa mintájára egy programot, ami a felhasználótól egész számokat kér be, amíg egymás után két azonos számot nem kap. Ha ez megtörtént, akkor írja ki, hogy hány számot adtunk be.

Most talán segít, ha már a cikluson kívül is kérünk be elõre számokat, és nem hátultesztelõs ciklust használunk, de azzal is kényelmesen meg lehet oldani, sõt for ciklussal is.

3. Min / Max

Írj programot, ami pontosan 5 számot kér be a felhasználótól, majd kiírja közülük a legnagyobbat. (Érdemes valami értelmes szöveggel pl: "http://wiki.math.bme.huA legnagyobb: %d"http://wiki.math.bme.hu)

Ha ez megvan, egészítsd ki a programot, hogy ne csak a maximálisat, hanem a minimálisat is írja ki.

4. Sakktábla

Rajzolj ki egy *NxN*-es sakktábla mintát, ahol X-szel jelöljük a fekete mez?ket, és üresen hagyjuk (egy szóköz) a fehéreket. Nem kell keretet adni a táblának. A sakktábla méretét (*N*) a felhasználótól kérd be!

Tipp: a ciklusokat egymásba is ágyazhatjuk, de ilyenkor nagyon kell figyelni a ciklusváltozókra!

Egymásba ágyazott ciklus példa:

```
#include<stdio.h>
```

```
int main(void) {
    int i;
    int j;

    for(i = 0; i < 10; i++) {
        printf("i: %d \n", i);
        for(j = 0; j < i; j++) {
            printf(" (%d, %d) ", i, j);
        }
        printf("\n");
    }
    printf("\n");
}
```

5. Prímtényezõ keresés

Írj programot, ami megkeresi egy a felhasználó által adott szám prímtényezõit és sorban kiírja azokat. (A hiba elkerülése végett elõször vizsgáljuk meg, hogy nem 0-t vagy 1-et kaptunk-e.)

Nem kell bonyolultra gondolni azonnal, meg lehet oldani úgy is, hogy egyesével megpróbáljuk elosztani az adott számunkat 2-tõl kezdve egyesével haladva egész számokkal, amíg 1-hez nem jutunk.

Emlékezzünk, hogy a maradék képzés (modulo) jele C-ben is a %

6. Pi közelítés

Közelítsd a Pi-t, a négyzetszámok reciprokösszege segítségével (ez ugye Pi négyzet per 6). Ha úgy mint az stdio.h-t betöltöd a math.h-t akkor mûködni fog az sqrt függvény, mellyel a gyökvonást megoldhatod.

Valamint, ha már sikerült közelíteni, akkor a 4 * atan(1) kifejezéssel ellenõrizheted magad (ennek elég jól kell becsülnie a Pi-t).