5. gyakorlat

Ezen a gyakorlaton a konstansokról, a ciklusokról és a lokalizációról esett szó.

Ciklusok

A ciklusok olyan kódrészletek, amelyek többször végrehajtásra kerülnek. A végrehajtások számát egy feltétel szabja meg. A C nyelv háromféle ciklust tartalmaz: `while`, `do while` és `for` ciklusok. Ezek működése igen hasonló.

While ciklus

A legegyszerűbb ciklus a while. Ez a nevéhez hűen **addig ismétli a törzsében elhelyezett kódrészletet, amíg a feltétele teljesül**. Ha a feltétel nem teljesül elsőre sem, akkor a törzsét egyszer sem hajtja végre. *Elöl tesztelő* ciklusnak is hívják.

Példa

Számoljunk felfelé 10-ig!

```
int szam = 1;

while (szam <= 10)
{
    printf("%i, ", szam);
    szam++;
}
```

A kimenet ilyen lesz:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Amikor a végrehajtás a while ciklusra kerül, az megvizsgálja a feltételét. Ekkor `szam`=1. A `szam <= 10` feltétel igaz, ezért a törzs lefut. A törzsben kiírásra kerül a szám, majd egy *inkrementálás* (eggyel növelés) folytán `szam` értéke 2 lesz.

A feltétel újra kiértékelődik, `szam <= 10` még mindig igaz, ezért a törzset ismét végrehajtja.

Az utolsó végrehajtás `szam`=10 esetén lesz. Ekkor a számot kiírja, majd a `szam` értékét 11-re növeli. A következő feltételvizsgálat hamis eredményre vezet, ezért a végrehajtás kilép a ciklusból.

Példa

Írjunk ki tetszőleges számú csillagot!

```
int darab;
printf("Hány darab csillag legyen? ");
scanf("%i", &darab);

while (darab > 0)
{
    printf("*");
```

```
    darab--;
}
```

Do while ciklus

A do while ciklus a while ciklus *hátul tesztelő* párja. A törzsét legalább egyszer lefuttatja, a feltételt ezután vizsgálja meg. Ez a legritkábban alkalmazott ciklus.

Példa

```
int szam;
do
{
    printf("Írj be egy számot 10 és 15 között! ");
    scanf("%i", &szam);
} while (szam < 10 || szam > 15);
```

Ez a program csak a helyes bemenetet fogadja el, hibás esetén újra kéri.

For ciklus

A for ciklust akkor kell használni, ha ismert az ismétlések száma. Ezt a ciklust használjuk a leggyakrabban.

A `for` kulcsszó után zárójelbe teszünk három utasítást:

1. **inicializálás**: általában egy változó definíciója (`int i=0`)
2. **feltétel**: ekvivalens a while ciklus feltételével
3. **léptetés**: növelés, csökkentés, egyéb léptetés, stb. A törzs végén kerül végrehajtásra.

Példa

Számoljunk el egytől tízig!

```
for (int i=1; i <= 10; i++)
{
    printf("%i ", i);
}
```

A fenti for ciklus **teljesen ekvivalens** az alábbi while ciklussal:

```
int i = 1;
while (i <= 10)
{
    printf("%i ", i);
    i++;
}
```

Egymásba ágyazás

A ciklusokat gond nélkül egymásba lehet ágyazni, akármilyen mélységben. Egy cikluson belül kezdhetünk egy újabbat, azon belül is egy újabbat, és így tovább.

Példa

```
for (int sor = 1; sor <= 10; sor++)
{
    for (int karakter = 1; karakter <= sor; karakter++)
    {
        printf("*");
    }
    printf("\n");
}
```

Mit rajzol ki a fenti program?

Break

A ciklusból a „természetes” úton kívül máshogy is kiléphetünk: a `break` utasítással. A `break` azonnal kilép a ciklusból.

```
char betu = 0;
while (betu != 'q')
{
    scanf(" %c", &betu);

    if (betu == 'Q')
        break;

    printf("A betű: %c\n", betu);
}
```

A q betű beírására a program kiírja a szöveget, ezután kilép a ciklusból. A Q betű beírására print nélkül, azonnal kilép.

> Megjegyzés: *A `" %c"` format stringben a szóköz fontos, a bufferben maradt entert távolítja el. Enélkül minden bevitelre kétszer futna le a ciklus: egyszer a betűre, egyszer meg üresen. Ilyen trükközés csak %c és %s beolvasásnál kell.*

Konstansok

A konstansok olyan változók, melyeknek értéke állandó és kötelezően definiált. Konstansokat használunk a feladat számára adott értékek tárolására, például fizikai állandók, megadott konstrukciós jellemzők, IC lábkiosztások.

A konstansok használatának két előnye van:

1. Biztosítjuk, hogy véletlenül nem fogjuk megváltoztatni az értékét. Egyértelműen jelezzük a szándékunkat.
2. Ha a programmemória és az adatmemória külön van (mikrovezérlők), a konstansok az „olcsóbb" programmemóriába kerülnek, ellentétben a változókkal. Spórolhatunk a memóriával.

Konstansokat a `const` kulcsszóval tudunk létrehozni.

```
const double PI = 3.14159265;
```

A konstansok értékét definiálnunk **kell**.

```
const double PI;  // érvénytelen!
```

A konstansoknak nem lehet utólag értéket adni.

```
const double PI = 3.14159265;
PI = 3;  // érvénytelen!
```

#define

A define egy *preprocesszor direktíva*. Konstans értékek definiálására használjuk. A fordítás előtt a define-okban létrehozott neveket a preprocesszor kicseréli az értékükre. Úgy működik, mint a *keresés és csere* egy szövegszerkesztőben.

Előnye a konstans változókkal szemben: Nem kell változókat létrehozni a konstansok tárolására (mivel fordítás előtt behelyettesíti a preprocesszor), így nem használunk memóriát vele. PC-n ez nem szempont, de egy 1000 bájt memóriájú mikrokontrolleren fontos.

```
#define MIN_VALUE 415
#define MAX_VALUE 1018
```

Számokon kívül szövegeket, tömböket, makrókat (függvényeket) is elnevezhetünk a define segítségével.

```
#define USERNAME "Joe"
#define SQUARE(x) (x*x)
```

> Vigyázat! *A # karakterrel kezdődő preprocesszor direktívák után nincs pontosvessző!*
>
> Megjegyzés: *A konstansok nevét csupa nagybetűkkel szokás írni.*

Localizáció

A világ különböző részein eltérő nyelveken beszélnek az emberek, valamint más és más szabályok vonatkoznak számok, dátumok, idő kijelzésére. Ehhez a programjainkban sokszor szükséges alkalmazkodni.

A lokalizáció igen bonyolult mutatvány (ahogy az az ajánlott videókból is kiderül), ezért csak akkor próbálkozzunk vele, ha tényleg szükséges. Az egyetemi tananyag keretében csak az ékezetes kiírás vonatkozásában van rá szükség.

Ajánlott videók:

- Internationalis(z)ing Code - Computerphile
- The Problem with Time & Timezones - Computerphile

A C a következő dolgokat lokalizálja nekünk:

- **LC_ALL** minden az alábbiak közül
- **LC_COLLATE** szöveg sorba rendezés
- **LC_CTYPE** karaktertípusok (szám, betű, írásjel), kisbetű-nagybetű kapcsolatok
- **LC_MONETARY** pénz formázása
- **LC_NUMERIC** számok formázása (tizedespont / -vessző)
- **LC_TIME** dátum és idő
- **LC_MESSAGES** rendszerüzenetek nyelve

Lokalizáció beállítása

```
setlocale(LC_ALL, "en_US.UTF-8");
setlocale(LC_ALL, "hu_HU.UTF-8");
```

A `setlocale` parancs beállít bizonyos dolgokat a háttérben, innentől kezdve az új nyelv szerint működnek a dolgok.

A lokálok neve rendszerfüggő!

- Windows: "en-US", "hu-HU"
- Linux/Mac: "en_US.UTF-8", "hu_HU.UTF-8"

Példa

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

int main()
{
    time_t ido = time(NULL);
    char buffer[80];

    strftime(buffer, 80, "%c", localtime(&ido));
    printf("en_US: %s\n", buffer);

    setlocale(LC_ALL, "hu_HU.UTF-8");

    strftime(buffer, 80, "%c", localtime(&ido));
    printf("hu_HU: %s\n", buffer);

    return 0;
}
```

***Magyarázat (nem kell tudni)**: Az `strftime()` a dátum, idő formázott kiírására való, a "%c" formátumkód a teljes dátumot és időt jelenti az alapértelmezett formátumban. A `localtime()` a `time_t` időbélyeget alakítja át `struct tm` struktúrává. A char buffer[] egy tömböt hoz létre - ezekről később lesz szó.*

Kimenet:

```
en_US: Sat Oct  5 22:19:10 2019
hu_HU: 2019. okt. 5., szombat, 22:15:16 CEST
```

A lokál átállítása után a dátum magyar formátumban jelenik meg.

Ékezetes kiírás (Windows)

A Unix rendszerekkel ellentétben Windowson az ékezetes karakterek kiírása nincs alapértelmezetten megoldva.

1. Két header fájlt kell használnunk hozzá.

```
#include <locale.h>   // lokalizáció
#include <tchar.h>    // _tcprintf
```

2. Be kell állítanunk a lokált

```
setlocale(LC_ALL, "hu-HU");
```

3. A kiíráshoz a `_tcprintf()` függvényt kell használnunk.

```
_tcprintf("Árvíztűrő tükörfúrógép");
```

Feladatok

1. Szorzótábla

1.1. Írd ki a hetes szorzótáblát egy sorba!

Várt kimenet

```
7 14 21 28 35 42 49 56 63 70
```

1.2. Írd ki a teljes táblázatos szorzótáblát!

Várt kimenet

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

Tipp: *A printf-ben használd a "%3i" formátumot!*

2. Rajzolj ki egy fenyőfát ciklusok használatával!

Várt kimenet

```
     *
    ***
   *****
  *******
 *********
***********
    |||
```

A törzs opcionális.