1-2. gyakorlat

Bevezető a C nyelvről

A C általános célú "magas szintű" programozási nyelv. A C minden idők legszélesebb körben használt programozási nyelve. Ken Thompson B nyelvére alapozva fejlesztette ki Dennis Ritchie 1969-1973 között az AT&T Bell Labs-nál. A UNIX rendszerre célozták (amelyet Ken készített). Hamarosan minden processzorarchitektúrához készült fordítóprogram, így a UNIX világhódító útra indulhatott.

Miért használunk C-t?

- Szinte minden processzoron futtatható, legyen az mikrokontroller vagy szuperszámítógép.
- Megismerhető mennyiségű eszközt tartalmaz, "szabványos" programkód írható vele. Open-source projekteknél előszeretettel használják.
- Rendkívül hatékony. A C kód 10..100-szor gyorsabb lehet a magasabb szintű nyelveknél, mint a Java, C# vagy Python. A nagy számításigényű algoritmusokat célszerű C-ben készíteni.
- *Kis helyen elfér*, mikrokontrollerhez is készíthető fordító. A C++ könyvtárai már nem férnének bele a memóriába.

Elméleti alapok az eheti gyakorlathoz

Változók

A változók memóriában eltárolt értékek. Van nekik

- adattípusuk
- nevük, amivel eléred őket
- értékük
- memóriacímük, ahol megtalálhatók a memóriában.

Adattípusok

- `int` - egész szám, 32 bites. -2147483648 .. 2147483648
- `unsigned int` - pozitív egész szám, 32 bites. 0 .. 4294967296
- `float` - "lebegőpontos" valós szám, 32 bites. Általában inkább double-t használunk.
- `double` - dupla pontosságú valós szám, 64 bites, kb. 15 értékes tizedesjegy
- `char` (unsigned 8-bit int) - ASCII karakter. 0 .. 255
- pointer: `int*`, `char*`, ... - memóriacímet tárol, később sok szó lesz róla.
- `void` - nem igazi adattípus, függvények argumentumaként és visszatérési típusaként jelzi hogy nem lesz argumentum illetve visszatérési érték.

Továbbiak a teljesség igénye nélkül

- `short int` - PC-n 16 bites int. -32768 .. 32768
- `long int` - PC-n ugyanaz mint az `int`. 16 bites rendszeren ez a 32 bites int.
- `long long int` - 64 bites. -2 trillió .. +2 trillió
- `bool` - logikai érték. C-ben nincs! C++-ban van. Helyette int-et lehet használni.
- `size_t` - memóriaterület méretet jelentő típus, valamilyen hosszúságú unsigned int.
- `int8_t`, `int16_t`,`uint8_t`, `uint16_t` - definiált méretű egész számok, mikrokontrollereknél használjuk.

Típuskonverzió

A változók adattípusát át lehet alakítani, ha erre szükségünk van. Például egy törtszámból egészet akarunk csinálni. Az átalakításhoz a szám / változónév elé zárójelek közé írjuk az új típust.

```
int mernok_pi = (int)3.14159265;
double kozelito_pi = (double)355 / (double)113;
```

A fordító automatikusan elvégzi azokat a típuskonverziókat, aminek értelme van (például char -> int vagy int -> double). Pontosságvesztés esetén a fordító egy *warninggal* figyelmeztet.

> ***Vigyázat!*** Két egész szám elosztása egész számot ad eredményül, 355/113 eredménye 3. Ha törtként akarjuk elosztani, legalább az egyik operandust át kell alaktanunk törtszámmá.

Fiók analógia

Ezt a hasonlatot sok helyen használják és a későbbiekben sokat segít a pointerek, tömbök megértésében.

Képzeljünk el a memóriát egy szekrényként, amiben sok fiók van. A fiókokban adatokat tárolhatunk, minden fiókban egy darab adatot. A fiókokra rá van írva a nevük, ez felel meg a változónévnek. Készíthetünk egy `elet_ertelme` nevű, **egész szám** tárolására alkalmas fiókot, melybe beletehetjük a **42** számot. A változó neve tehát `elet_ertelme`, a típusa `int`, és a tartalma természetesen a 42. A fiók a szekrény valahányadik helyén van (ezt a fordító dönti el), például a 189. fiókot kaptuk. Ez a változó memóriacíme.

A változók létrehozása programkódban így néz ki:

```
int elet_ertelme;   // kérünk egy fiókot -- változó deklaráció
elet_ertelme = 42;  // belerakjuk az adatot -- definíció, értékadás
```

Az előző két sort írhatjuk egybe is:

```
int elet_ertelme = 42;
```

Függvények

A függvények többször felhasználható, elnevezett kódrészletek.

A függvény neve után mindig zárójel következik. Itt adhatjuk meg a függvény bemenő adatait, vagyis az *argumentumokat*. A függvény a bemenő adatokkal (ha vannak) csinál valamit, és esetleg *visszaad* eredményül egy kimenő adatot.

Példaként vizsgáljunk egy egyszerű függvényt, a `sin()` szinuszfüggvényt a `<math.h>` könyvtárból.

```
double szog = 1.251623;
double szinusz = sin(szog);
```

A függvény argumentuma a szög **radiánban**, visszatérési értéke (eredménye) pedig a szög szinusza.

Argumentum nélküli függvények is vannak. Az általuk végrehajtott feladathoz nincs szükségük bemenetre. Például a `<stdlib.h>` könyvtárban lakó `rand()` függvény egy véletlen egész számot ad vissza, argumentumot nem vár.

```
int veletlen = rand();
```

A visszatérési érték nélküli függvények `void` típusúak. Ezek elvégeznek egy feladatot - például kiírnak egy szöveget a képernyőre -, de eredményt nem adnak.

A C nyelv biztosít sok beépített függvényt, melyek az ún. *standard library*-ben vannak. A standard library könyvtárai a különböző `.h` header fájlok, mint a `<time.h>`, `<stdio.h>`, `<math.h>`. Az eheti gyakorlaton a `<stdio.h>` (standard input/output) könyvtárban található `printf()`, `scanf()` és `getchar()` függvényeket ismerkedünk meg. A könyvtárakat így tudjuk importálni a kódunkba:

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

> ***Megjegyzés*** Az include-okat mindig a fájl tetejére írjuk.

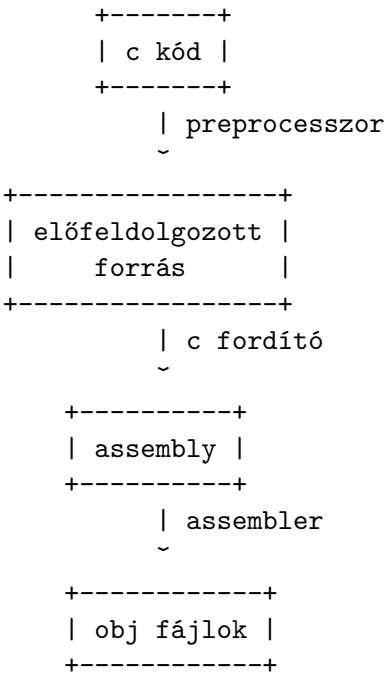
Van továbbá egy speciális függvény, a `main()`, ami minden programban benne van - ez tulajdonképpen egy saját függvény, de a neve kötelezően main. A program megnyitásakor a számítógép ezt a függvényt kezdi el végrehajtani.
Tehát a programot mindig így kezdjük:

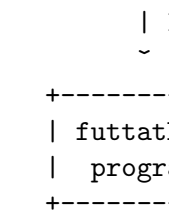
```
int main()
{
  ...
  return 0;
}
```

> ***Megjegyzés*** A main függvény visszatérési értéke int típusú. A `return` a függvény visszatérési értékét (eredményét) adja meg, jelen esetben a `0` hibakódot (nincs hiba). Argumentumként egyszerűbb esetben nem vár semmit - ezt jelzi az üres argumentumlista `()` vagy a `(void)` *üres* argumentum.

A C fordítási folyamata

A C ún. lefordított (compiled) nyelv. A forrásfájlt egy speciális programmal, a fordítóval (compiler) átalakítjuk futtatható programmá, majd a számítógépen ezt a programot elindíthatjuk.





- **preprocesszor**: `#include`, `#define` direktívák feloldása
- **előfeldolgozott forrás**: még mindig C forráskód
- **assembly**: nyers processzorutasítások, de még olvasható formában
- **obj fájlok**: object code, végrehajtható, bináris programrészek
- **linker**: összefűzi a saját és a külső obj fájlokat egy futtatható programmá

Szintaxis

Nevek (identifiers): A változók és függvények nevei angol betűket, számokat és alulvonást tartalmazhatnak, és nem kezdődhetnek számmal. A C megkülönbözteti a kis- és nagybetűket.

Pontosvessző: A C nyelvben minden utasítás után pontosvesszőt teszünk, ez jelenti az utasítás végét.

Blokkok: A kódblokkokat, vagyis az együtt végrehajtandó utasításokat kapcsos zárójelek közé {} tesszük. (Ezek után nem kell pontosvessző, hiszen nem utasítások).

Szövegek, karatkerek: A szövegeket mindig "macskakörmök" közé tesszük, a karaktereket pedig ' ' aposztrofok közé.

Behúzás (indentation): A behúzás nem számít a C nyelvben, de mindenképpen érdemes használni, hogy a kód olvasható legyen. Az egy blokkba tartozó utasítások ugyanannyira legyenek behúzva. Új blokkba lépéskor a behúzás mértéke nő.

Kommentek: A kód közé írhatunk megjegyzéseket, amiket a fordító figyelmen kívül hagy.

```
/* Így írhatunk hosszú, több
   soros kommenteket. */

// Az ilyen kommentek a sor végéig működnek.
```

Ékezetek:

A Windows Command Prompt meghülyül az ékezetektől, ezért a kiírandó szövegekben kerüljük az ékezetek használatát. Csak az ASCII kódtábla karaktereit tudjuk használni.

A Unix-szerű operációs rendszerek (Linux, MacOS) ki tudják írni a Unicode karaktereket. Ezekben a jegyzetekben használok ékezeteket az olvashatóság kedvéért.

> ***Megjegyzés*** Programozásra a Linux és a MacOS nagyon jó eszközöket nyújt. Jóval kevesebb problémával szembesül az ember, mint Windows alatt. Windowsra is lehet telepíteni egy hivatalos linux alrendszert (WSL), ezt érdemes kipróbálni.

A programkód olvasása

A programvégrehajtás a **main** függvényben kezdődik, ott kell kezdeni az olvasást is. Olvassuk felülről lefelé a kódot.

`printf()`

Az első függvény, amivel megismerkedünk, a `printf()`. A printf jelentése: print formatted. A konzolra tudunk vele szöveget és adatokat írni.

A printf függvénynek legegyszerűbb esetben 1 argumentuma van, mégpedig hogy mit szeretnénk kiírni a konzolra. A kiírandó szöveget *macskakörmök közé* rakjuk, így jelezzük, hogy egy szövegkonstanst írtunk be. A szöveget mindig macskakörmök közé kell rakni.

Formázott szöveg kiírásánál az első argumentum a szöveg, melyben formázási szimbólumokat (%) hagyunk. A további argumentumok ezek helyére kerülnek, a definiált formátumban.

Formátumok:

- `%i` egész szám
- `%f` törtszám, tizedestört alak
- `%e` törtszám, normálalak
- `%g` törtszám, automatikus alak (egész / tizedes / normál)
- `%lf`, `%le`, `%lg` duplapontos törtszám (double)
- `%x` hexa
- `%c` karakter
- `%s` karakterlánc (szöveg)

Az összes formátum a "printf format strings" kifejezésre keresve található meg.

Megadhatjuk még a mező szélességét, pl. `"%8i"`. A tizedestörtek felbontása is definiálható, pl. `"%.2f"`. Szélesség és felbontás egyszerre `"%8.2f"`. Új sort a `\n` *new line* vezérlőkarakterrel kezdhetünk.

Példa:

```
printf("Üdvözöllek %s! %i új üzeneted van. A hőmérséklet %.1f°C\n",
    "Joe", 12, 24.33);
```

```
Üdvözöllek Joe! 12 új üzeneted van. A hőmérséklet 24.3 °C
```

`scanf()`

A második függvény a `scanf()`, jelentése scan formatted. A konzolról beolvas egy sort, és abban a megadott formátum szerint értelmezett adatot beírja a megadott változóba.

Az első argumentuma a format string.

- `%i` egész szám
- `%lf` duplapontos törtszám (double)
- `%c` karakter
- `%s` szó (szóköznél befejezi az olvasást)

Vigyázat! A `%d` NEM a double jele, hanem a decimális egész számé.

A második argumentum adja meg hogy *hova* olvassa be az adatot. Egy változó neve kerül ide, megelőzve egy `&` jellel.

Miért kell az & jel?

A C nyelvben a függvény az átadott argumentumok a **másolatát** kapja meg. A másolatot megváltoztatva az eredeti változó nem módosul.
Ha az eredetit akarjuk megváltoztatni - mint ahogy a scanf akarja -, meg kell szereznünk az eredeti változó memóriacímét. Ezt csinálja a & operátor. A függvény az átadott memóriacímen megtalálja a módosítandó adatot.

Példa:

```
int sebesseg;
printf("Mennyi egy töketlen fecske maximális repülési sebessége? (km/h)");
scanf("%i", &sebesseg);
```

> ***Megjegyzés*** A printf-hez hasonlóan több adatot is be lehet olvasni egy scanf utasítással, a format string több szimbólumot tartalmazhat.
> `scanf("%s %i", &nev, &eletkor)`

`_getch()` vagy `getchar()`

Ezek a függvények egy karaktert vesznek be. Megállíthatjuk velük a programfutást egy billentyű lenyomásáig. Így nem tűnik el a konzolablak a programunk lefutása után, mert még megvárja a billentyűlenyomást. A `Nyomj egy gombot a bezáráshoz` funkció valósítható meg velük.

- A `_getch()` a `<conio.h>` könyvtárban van, csak Windowson elérhető. Bármelyik karakterre továbbmegy.
- A `getchar()` a `<stdio.h>` könyvtár része. Ez egy karaktert és utána egy entert vár, tehát a továbblépéshez nekünk egy entert kell nyomni.

`sizeof()`

Ez a „függvény" az argumentumként megadott típus méretét adja vissza, bájtban.

```
size_t meret = sizeof(int);
```

Két célra használjuk:

1. Megadja, hogy mekkora az adattípus az adott rendszeren - 16 bites mikrovezérlőn az int 16 bites, számítógépen 32 bites.
2. Az adatstruktúrák (`struct`) méretét lehet vele megtudni. Erről később lesz szó.

> ***Megjegyzés*** A `sizeof()` valójában nem is függvény, hanem egy operátor. Szintaktikailag nem is lenne helyes függvény.

Feladatok

Ezeket a gyakorlófeladatokat próbáld meg minél kevesebb segítséggel, minél több fejtöréssel megoldani. Csak akkor olvasd el a segítséget, ha már gondolkodtál rajtuk egy percet.

1. Írd meg kézzel, emlékezetből a Hello World programot! Köszöntsön a neveden, a nevet behelyettesítéssel írd ki!

Várt kimenet:

```
Szia Joe!
```

Segítség Ezekre kell emlékezni:

- `#include <stdio.h>`
- `int main() {...}`
- `printf("%s", "szoveg");`
- pontosvesszők

2. Írasd ki a neved kezdőbetűjének ASCII kódját! A karaktert egész számként kell kiírni.

Várt kimenet:

```
A J karakter ascii kódja 74.
```

Segítség Ezekre kell emlékezni:

- karakter létrehozása `char betu = 'L';`
- kiírás karakterként a `"%c"` format stringgel, egész számként a `"%i"` format stringgel.

3. Olvass be egy karaktert a billentyűzetről (scanf segítségével), majd írd ki karakterként és egész számként!

Várt kimenet:

```
Nyomj egy billentyűt! J
A J karakter ascii kódja 74.
```

Segítség Ezekre kell emlékezni:

- `scanf("%i", &valtozo)`
- karakter típusú változóba olvasd be
- karakter formátumban olvasd be (`"%c"`)

SZORGALMI Oldd meg a feladatot scanf helyett `getchar()` segítségével is! A `getchar()` visszatérési értéke a beolvasott karakter.