Programozás I
gyakorlat

2. Változók, I/O

Kiíratás a képernyőre, hogy lássuk, mit csinál a programunk

```
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

A printf-el íratunk ki a képernyőre*

* Ezt majd később pontosítjuk

Kiíratás a képernyőre, hogy lássuk, mit csinál a programunk

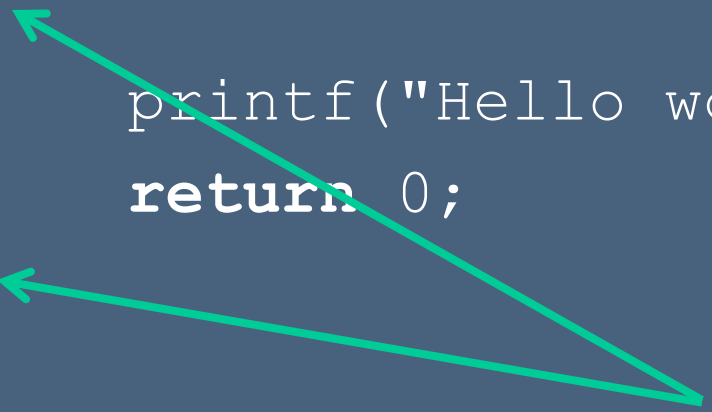
```
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Ez szükséges ahhoz, hogy a printf-et használhassuk*

* Majd a függvényeknél megértjük, hogy hogyan működik

Kiíratás a képernyőre, hogy lássuk, mit csinál a programunk

```
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

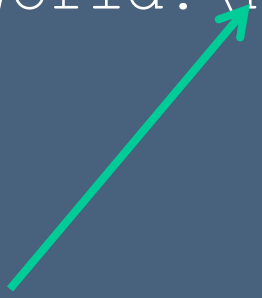


Egy ideig csak e két zárójel közé írjuk a kódot*

* A függvényeknél majd megértjük, mi is az a **main** és a **return**

Kiíratás a képernyőre, hogy lássuk, mit csinál a programunk

```
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```



Ezt nem írja ki a képernyőre, hanem a kurzor új sorba ugrik

Escape szekvenciák

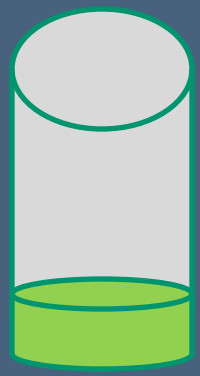
- \a csengő
- \n új sor
- \b visszalépés
- \f lapdobás
- \r kocsi vissza
- \t vízszintes tabulátor
- \v függőleges tabulátor
- \\ \
- \” ”
- \’ ’

Változók, hogy számolni is tudjunk

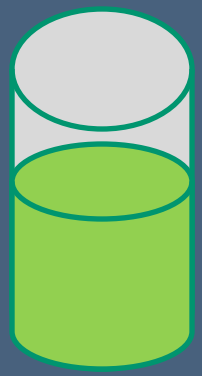
- Változó: A számítógép írható és olvasható memóriájában található, valamilyen meghatározott memória területen
- Fizikailag 1 bites tárolók tartalmazzák a változó értékét
- Ezen memóriaterületekre a C-ben névvel hivatkozhatunk

Változók: Kezdjük az elején

- 1 bit: két állapotot tárol, 0 vagy 1
- Például: van áram a tárolóban, vagy nincs áram?



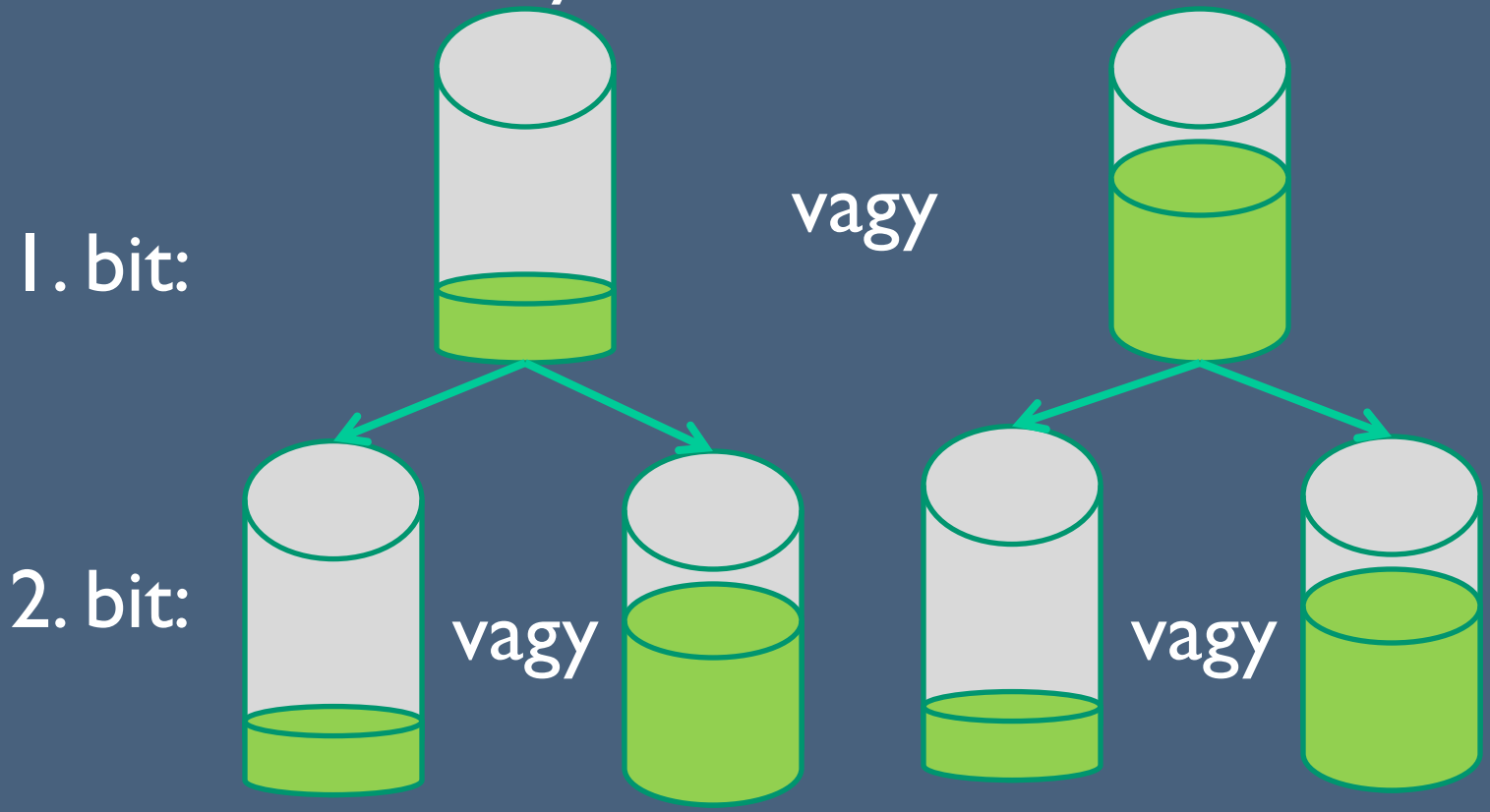
Kevés áram: 0



Sok áram: 1

Változók: Kezdjük az elején

- Ha 1 bittel 2 értéket ábrázolhatunk, akkor 2 bittel mennyit?



Változók: Kezdjük az elején

- Minden újabb bittel megkettőzzük az ábrázolható értékek számát

- N darab bittel 2^N különböző érték ábrázolható

- 8 bit: 256
- 16 bit: 65 536
- 32 bit: 4 294 967 296

Változók: Kettes számrendszer

- A kettes számrendszer alkalmas arra, hogy az előbbi módszerrel számokat ábrázoljunk
- Példa: $123_{(10)}$, 10-es számrendszerben:

  $\underline{1} * 10^2 + \underline{2} * 10^1 + \underline{3} * 10^0$
- Kettes számrendszer: A helyi értékek a 2 hatványai, például $1101_{(2)}$:

  $\underline{1} * 2^3 + \underline{1} * 2^2 + \underline{0} * 2^1 + \underline{1} * 2^0$

Változók: Kettes számrendszer

- A $13_{(10)}$ tárolásához hány bitre van szükség?
- Mivel $13_{(10)} = 1101_{(2)} =$

  $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$
- Legalább 4 bitre van szükség
- Hány bitet szánjunk egy változóra?

Változó típusok

- Legtöbbször 8, 16, 32 és 64 bites változókat használunk, de előfordul 80 bites is
- Milyen számokat ábrázolhatunk?
- Egy bitsorozatot több módon is értelmezhetünk

Változó típusok

- Előjel nélküli egészek:
- Minden bit 1-1 helyi értéket jelöl, például az $1101_{(2)}$ a $13_{(10)}$-at ábrázolja
- A legkisebb ábrázolható szám a 0
- A legnagyobb ábrázolható szám N biten:

$$2^N-1$$

- Példa: 8 biten 0-tól 255-ig ábrázolhatjuk a számokat

Változó típusok

- Előjeles egészek:
- Negatív számokat is ábrázolhatunk
- Pozitív számok: Ugyanúgy, mint az előjel nélküli számoknál
- Negatív számok: Kettes komplemens kód
- $-5_{(10)}$ ábrázolása 8 biten:

  $5_{(10)} = 00000101_{(2)} \longrightarrow 11111010_{(2)}$

  $-5_{(10)} = 11111011_{(2)}$

Változó típusok

- $-5_{(10)} = 11111011_{(2)}$

- Legmagasabb helyi értékű bit: Előjel
- A kettes komplemens ábrázolás megkönnyíti a kivonás műveletet
- A legnagyobb ábrázolható érték feleződik
- Példa: 8 biten -128-tól 127-ig ábrázolhatjuk számokat

Változó típusok

- A változó típusa megadja:
  - Hány biten ábrázoljuk az értéket
  - Hogyan értelmezzük a tárolt bitsorozatot
- Változó típus választáskor döntsünk az alapján, hogy mit szeretnénk ábrázolni
  - [signed] char: 8 bit, előjeles
  - [signed] short: 16 bit, előjeles
  - [signed] int : 32 bit, előjeles
  - [signed] long long int: 64 bit, előjeles

Változó típusok

- Előjel nélküli típusoknál kötelező az unsigned kulcsszó használata:
  - unsigned char: 8 bit, előjel nélküli
  - unsigned short: 16 bit, előjel nélküli
  - unsigned int : 32 bit, előjel nélküli
  - unsigned long long int: 64 bit, előjel nélküli
- A long (vagy long int) típus egyes architektúrákon 32, máshol 64 bites

Változó típusok

- Tört számok: Lebegőpontos szám ábrázolás
- float: 32 bit
  - 1 bit az előjelnek
  - 23 bit a mantisszának
  - 8 bit a kitevőnek
- double: 64 bit
  - 1 bit az előjelnek
  - 52 bit a mantisszának
  - 11 bit a kitevőnek

Változó típusok

- A lebegőpontos számábrázolás részleteit nem tárgyaljuk, de a lényeg:

- Teljesen másképp ábrázoljuk a lebegőpontos számokat, mint az egészeket

Változók definiálása


```
#include <stdio.h>
int main() {
    int a = 100;
    int b;
    printf("Az a erteke: %d\n", a);
    printf("A b erteke: %d\n", b);
    a = a + 20;
    printf("Az a erteke: %d\n", a);
    return 0;
}
```

Változók definiálása

```
#include <stdio.h>
int main() {
    int a = 100;
    int b;
    printf("Az a erteke: %d\n", a);
    printf("A b erteke: %d\n", b);
    a = a + 20;
    printf("Az a erteke: %d\n", a);
    return 0;
}
```

Az inicializálás nem kötelező

Változók kiírása

```
printf("Az a erteke: %d\n", a);
```

Ezt így kiírjuk a képernyőre

A kimenet:

```
Az a erteke:
```

Változók kiírása

```
printf("Az a erteke: %d\n", a);
```

Ezt nem írjuk ki, a % jelzi, hogy most egy változót kell kiíratni

A kimenet:

```
Az a erteke:
```

Változók kiírása

```
printf("Az a erteke: %d\n", a);
```

A % utáni d pedig megadja, hogy a változó által tárolt bitsorozatot előjeles decimális egész számként kell értelmezni

A kimenet:

```
Az a erteke: 100
```

Változók kiírása

```
printf("Az a erteke: %x\n", a);
```

A % utáni x megadja, hogy a változó által tárolt bitsorozatot előjel nélküli hexadecimális egész számként kell értelmezni

A kimenet:

```
Az a erteke: 64
```

Változók kiírása

```
printf("Az a erteke: %c\n", a);
```

A % utáni c pedig megadja, hogy a változó által tárolt bitsorozatot egy karakter kódjaként kell értelmezni.
A 100 a ‘d’ karakter kódja
A kimenet:

```
Az a erteke: d
```

Változók kiírása

- A % jel után kötelező egy konverziós karaktert elhelyezni:
- d, i: decimális előjeles egész
- u: decimális előjel nélküli egész
- X, x: előjel nélküli hexadecimális egész
- o: előjel nélküli oktális
- c: karakter
- f: lebegőpontos
- g: lebegőpontos esztétikusabb kivitelben

Változók kiírása

```
printf("Az a erteke: %g\n", a);
```

A % utáni g pedig megadja, hogy a változó által tárolt bitsorozatot lebegőpontos számként kell értelmezni

A kimenet:

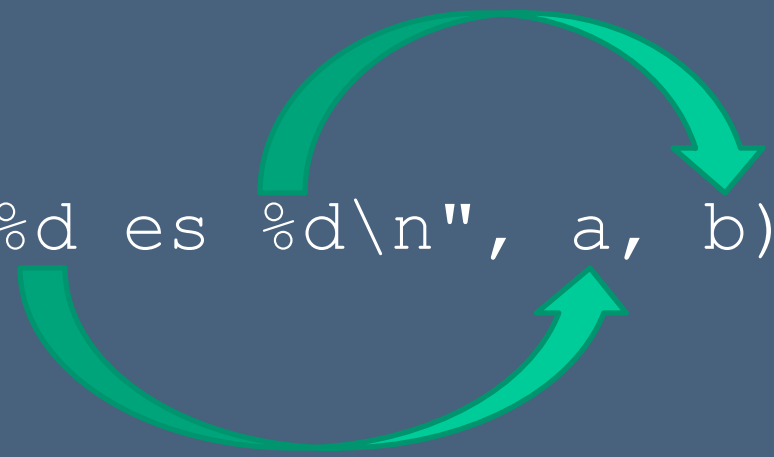
```
Az a erteke: 5.56197e-308
```

Ez hiba, a g most nem megfelelő konverziós karakter!

Változók kiírása

Egy printf-el több változót is kiírathatunk!

```
printf("A szamok: %d es %d\n", a, b);
```



Kimenet:

A szamok: 100 es 45

Változók beolvasása

```
#include <stdio.h>
int main() {
    int a;
    scanf("%d", &a);
    printf("Az a erteke: %d\n", a);
    return 0;
}
```

Kimenet:

43 ↵

Az a erteke: 43

Változók beolvasása

```
int a;
scanf("%d", &a);
```

A scanf függvénnyel olvassuk be a változókat

Változók beolvasása

```
int a;
scanf("%d", &a);
```

Formátum string: Egyetlen előjeles egész számot olvasunk be decimális alakban…

Változók beolvasása

```
int a;
scanf("%d", &a);
```

… és az a nevű változóban tároljuk a beolvasott értéket

Változók beolvasása

```
int a;
scanf("%d", &a);
```

Az & karaktert ne felejtsük el beírni a változó elé, ha scanf-et használunk*

* Egyébként címképző operátornak hívják, a változó címét adja vissza, majd a mutatóknál megértjük, hogy mit jelent és miért kell ide

Változók beolvasása

```
int a, b, c;
scanf("%d.%d.%d", &a, &b, &c);
```

Először be kell írni egy decimális egész számot, majd egy pontot, majd újabb számot, megint pontot és végül egy harmadik számot

Gyakran ismételt kérdések

- Fura értéket ír ki az alábbi program, mi a baj?

```
int a;
printf("Az a erteke: %d\n", a);
```

- Nem adtál értéket (azaz nem inicializáltad) a változónak, ezért azt a számot íratod ki, amely korábbról ott maradt abban a memória részben, ahol az a változó létrejött: Ezt hívjuk memóriaszemétnek.

Gyakran ismételt kérdések

- Mi történik, ha van %d a printf formátumstringében, de lefelejtem a változót?

```
printf("Az a erteke: %d\n");
```

- Olyan memóriaterület tartalmát íratod ki a képernyőre, amely nem változóhoz tartozik.

Gyakran ismételt kérdések

- És ha ugyanezt csinálom a scanf-el?

```
scanf("%d");
```

- A program egy véletlenszerű memóriacímre próbálja elmenteni a beolvasott értéket, nagy valószínűséggel elszáll.

Gyakran ismételt kérdések

- Az alábbi programban miért kell kétszer Enter-t ütnöm?

```
scanf("%d\n", &a);
```

- Először is a scanf bevitelének a végét az Enter-el kell jelezni. Másrészt ez a formátumstring a következőt írja elő: először egy decimális számot, majd egy Enter-t kell ütni. Végül Enter-el kell zárni a bevitelt.

Gyakran ismételt kérdések

- Az alábbi program nem írja ki a képernyőre, hogy „Kerem a szamot: ”, mi a baj?

```
scanf("Kerem a szamot: %d", &a);
```

- A scanf nem ír ki semmit. Ez a formátumstring azt jelenti, hogy a felhasználónak először be kell gépelnie, hogy „Kerem a szamot: ”, majd meg kell adnia egy decimális egész számot.

Gyakran ismételt kérdések

- Hová tűnt Daemon Hill?!
- Nem tudom, Lepsénynél még megvolt.

Gyakran ismételt kérdések

- Eclipse-ben nem működnek a \b és \r escape-szekvenciák, mit tegyek?
- Eclipse konzolban ez valóban nem működik, tudom ajánlani a rendes terminált. Viszont zh-n ezekre a szekvenciákra nem lesz szükség.

Gyakran ismételt kérdések

- Nem működik a \v, \f, elrontottam valamit?
- Ezeket nem minden terminál programban implementálják, de a félév során nem is lesz rájuk szükség.

Gyakran ismételt kérdések

- Az alábbi program -10 helyett 4294967286-ot ír ki, miért?

```
#include <stdio.h>
int main() {
    int a = -10;
    printf("Az a erteke: %u\n", a);
    return 0;
}
```

- A -10-et ábrázoljuk kettes komplemens kódban, majd az eredményt értelmezzük előjel nélküli egészként.