Programozás I.

Horváth Ernő

Elérhetőségek

Hatwágner Ferenc Miklós
http://www.sze.hu/~hatwagnf/
hatwagnf@sze.hu

Horváth Ernő
http://www.sze.hu/~herno/

Tanszéki honlap
http://it.sze.hu

RS1.SZE.HU/~HERNO/

szelearning.sze.hu

szelearning.sze.hu/course/view.php?id=623

repl.it

Kabinet használat

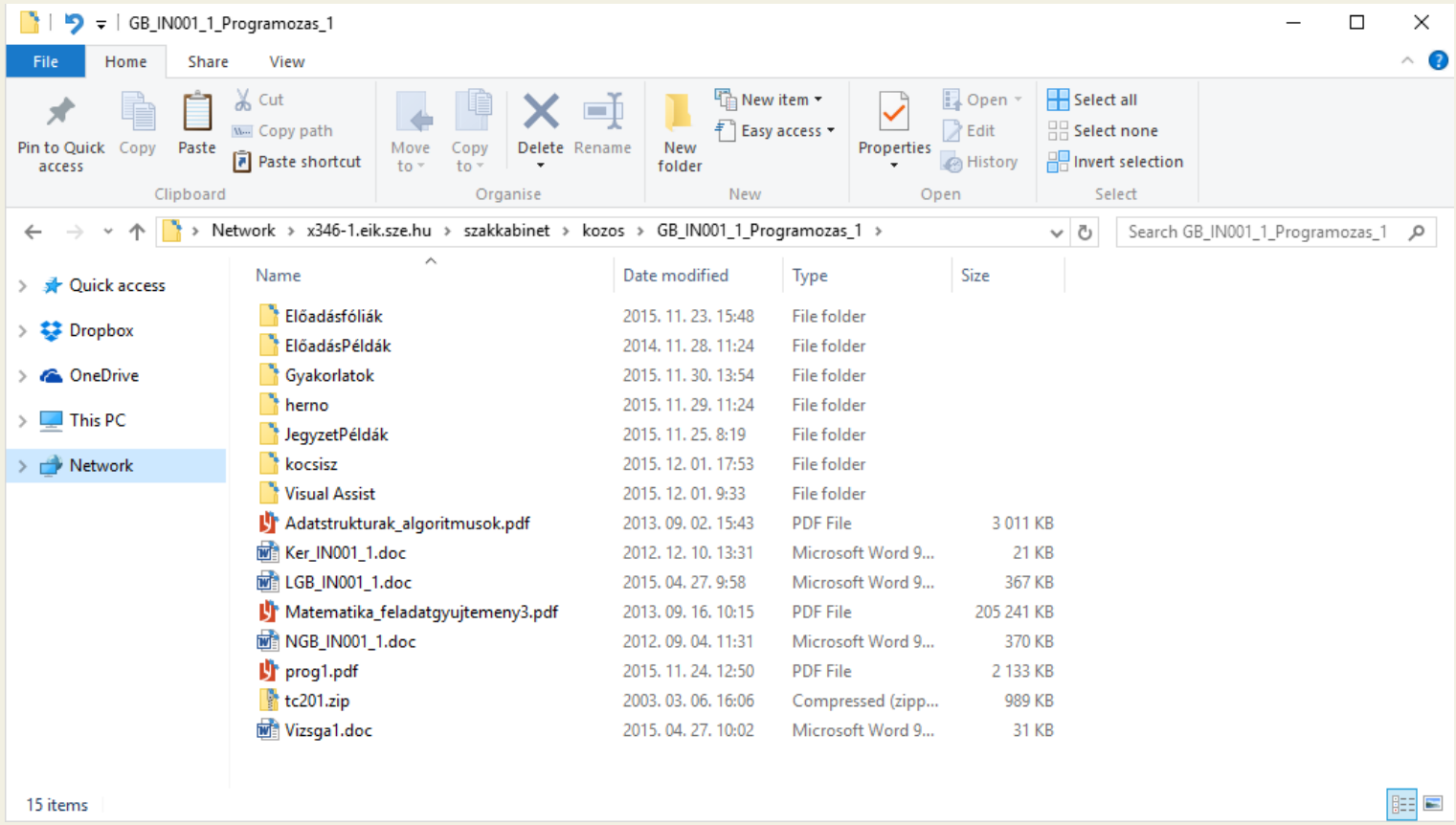
- Felhasználó név: EIK
- Jelszó nincs

- L:\ - írási jog, vizsgán ide kell a végső kódnak felkerülnie
- D:\ - írási jog
- C:\temp - ide érdemes dolgozni
- K:\GB_IN001_1 – vizsgán, órán ezek a feladatok használhatóak

K meghajtó

\\fs-kab.eik.sze.hu\szakkabinet\kozos\GB_IN001_1_Programozas_1

\\fs-kab.eik.sze.hu\szakkabinet\kozos\GKNB_INTM023 - Programozas alapjai



A félév

1-5. hét	Alapismeretek. Számítógép, fájlok. Előfeldolgozás, fordítás, kapcsoló-szerkesztés és végrehajtás. Strukturált, hierarchikus programtervezés és programozás és programtervezés alapjai. Függvények, paraméterezés, formális és aktuális paraméterek. Deklaráció és definíció. Bemenet, kimenet.
6-8. hét	Adattípusok, konstansok. Karakterláncok, stringek kezelése, bemenet ellenőrzése, konverzió.
9-10. hét	Utasítások, előfeldolgozó direktívák. Rendezés és keresés tömbben.
11-14. hét	A program általános szerkezete Precízebb inputellenőrzés

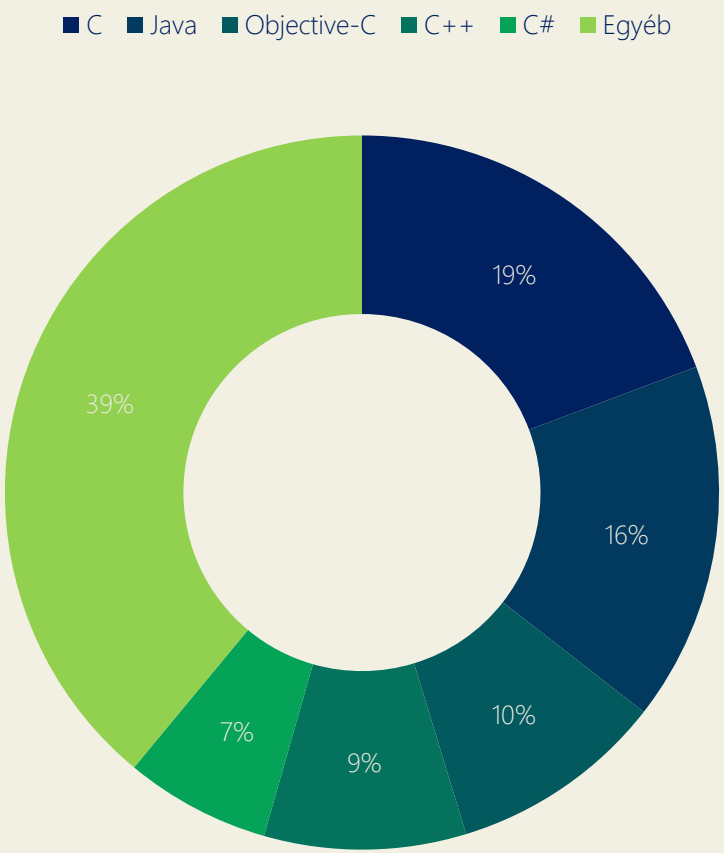
Követelmények

- Vizsga
  - Vizsgaidőszakban
    Alapfeladat 2es, minden plusz feladat után egy jeggyel jobb az érdemjegy, de az alapfeladatnak működnie kell szintaktikai hiba nélkül
- ZH nincs
- Katalógus nincs

Miért éppen C?

- Előnyök
  - » népszerű és elterjedt - “hordozható”
  - » kevés szemantikai kényszer - elegáns szintaxis
  - » sok ráépülő további nyelv
    - (C++, Objective-C, Java, C#, stb.)
- Hátrányok
  - » kevés szemantikai kényszer - elegáns szintaxis
  - » lapos tanulási görbe

Programozási nyelvek népszerűsége

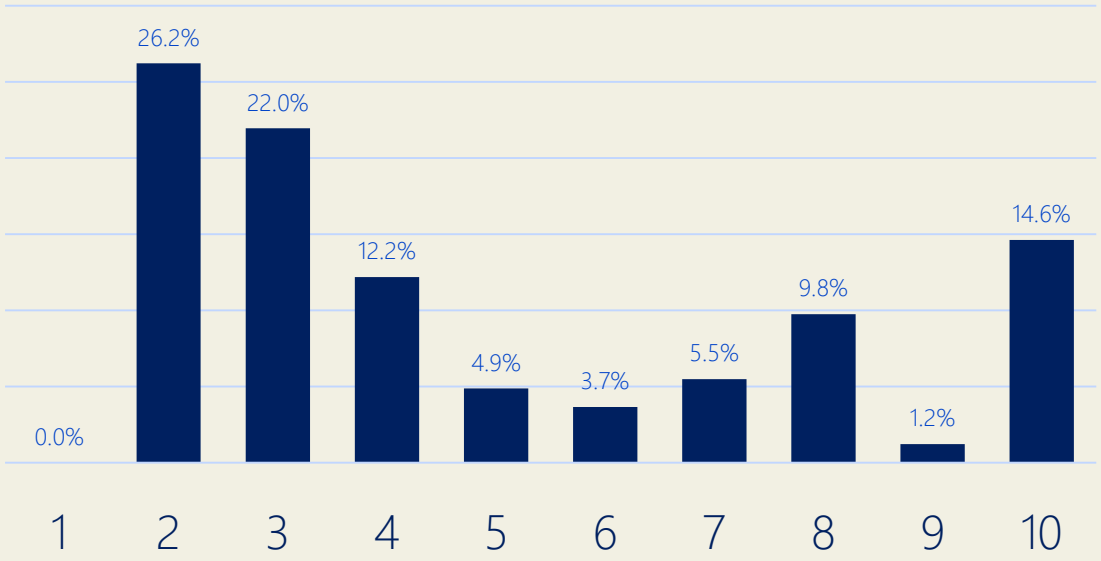


	2017	2012	2007	2002	1997	1992	1987
C	2	1	2	2	1	1	1
C++	3	3	3	3	2	2	5
C#	4	4	7	13	-	-	-
Python	5	7	6	11	27	-	-
Visual Basic .NET	6	17	-	-	-	-	-
PHP	7	6	4	5	-	-	-
JavaScript	8	9	8	7	23	-	-
Perl	9	8	5	4	4	10	-
Assembly	10	-	-	-	-	-	-
COBOL	25	28	17	9	3	9	9
Lisp	31	12	15	12	9	4	2
Prolog	32	31	26	16	20	11	3
Pascal	114	15	21	97	8	3	4
Java	1	2	1	1	15	-	-

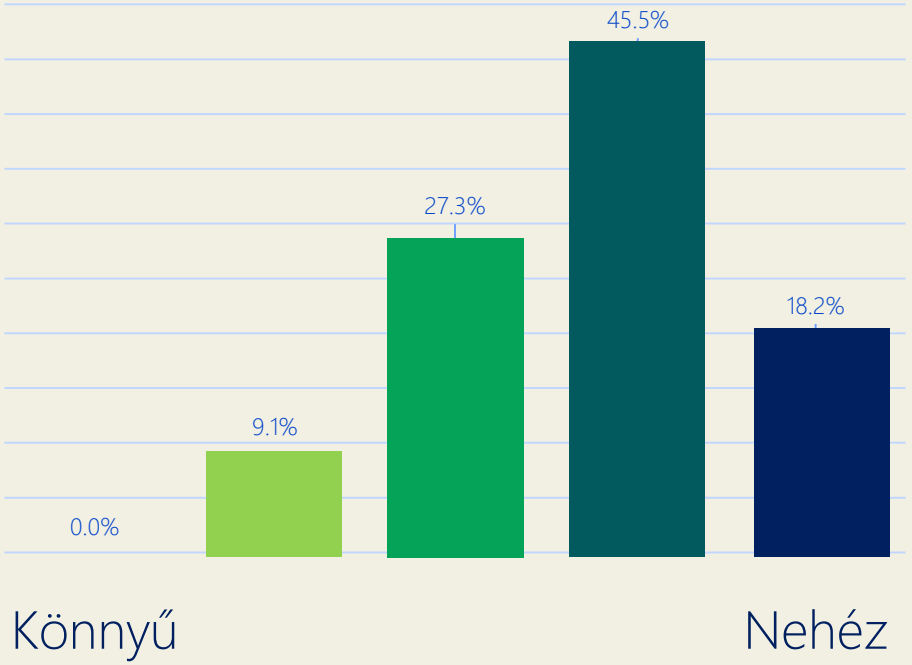
Forrás: http://www.tiobe.com (2017)

Hallgatói statisztikák

Ennyit órát foglakozott Programozás I. tanulással a tanórákon kívül



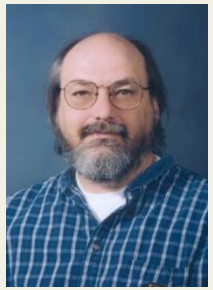
A tárgy nehézsége



* A felmérést 2016-ban a tárgyat sikeresen teljesítők között végeztem, a válaszadás anonim volt.

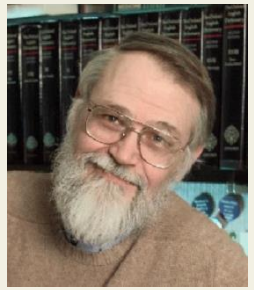
C történelem

- 1970 B - Bell Labs, Ken Thompson
- 1972 C - Bell Labs, Dennis Ritchie
- 1973 UNIX kernel C-ben átírva
- 1978 Brian Kernighan & Ritchie: The C Programming Language
- 1990 ANSI C



Thompson

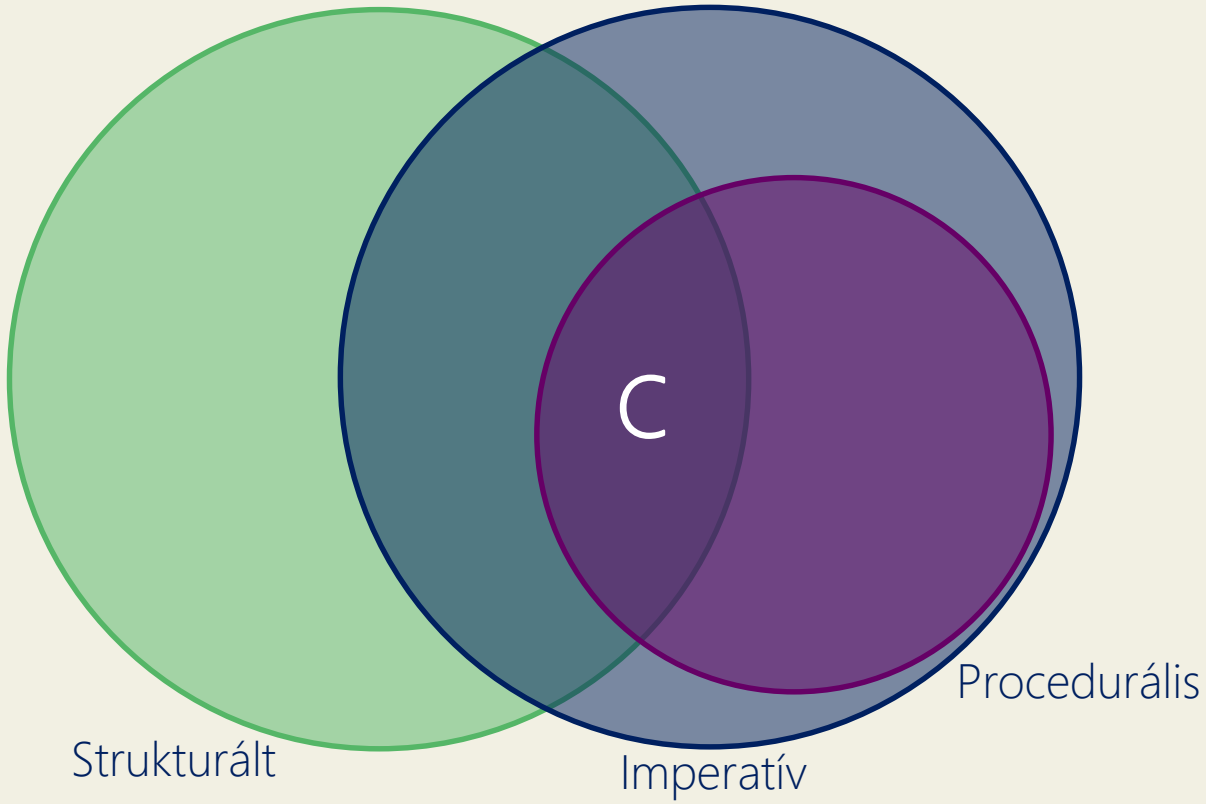
Ritchie



Kernighan

Programozási paradigmák

A C nyelv strukturált, imperatív, procedurális nyelv.



- Data-driven
- Declarative
  - » Constraint
  - » Dataflow
  - » Functional
- **Imperative**
  - » Literate
  - » **Procedural**
- Non-structured
- **Structured**
  - » Block-structured
  - » Modular
  - » Object-oriented (OOP)
  - » Recursive

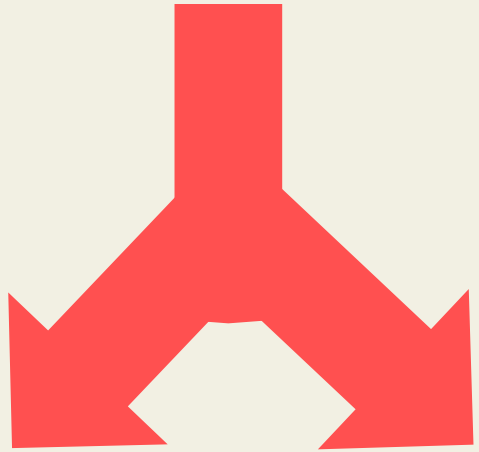
Programozási paradigmák

- **Procedurális programozás**ról beszélünk, ha a programozási feladat megoldását egymástól többé kevésbé független alprogramokból (procedure) építjük fel
- **Strukturált programozás** során elvégzendő feladatot kisebb, egymáshoz csak meghatározott módon kapcsolódó részfeladatokra bontjuk, ezeken belül ugyanígy részfeladatok határozandók meg, amelyek teljes egészében megvalósítják az őket magába foglaló nagyobb részfeladatot, és így tovább. A feladat szekvenciára, szelekcióra és iterációra osztható.

Egymás utáni végrehajtás, elágazás, ismétlődés

Szekvencia

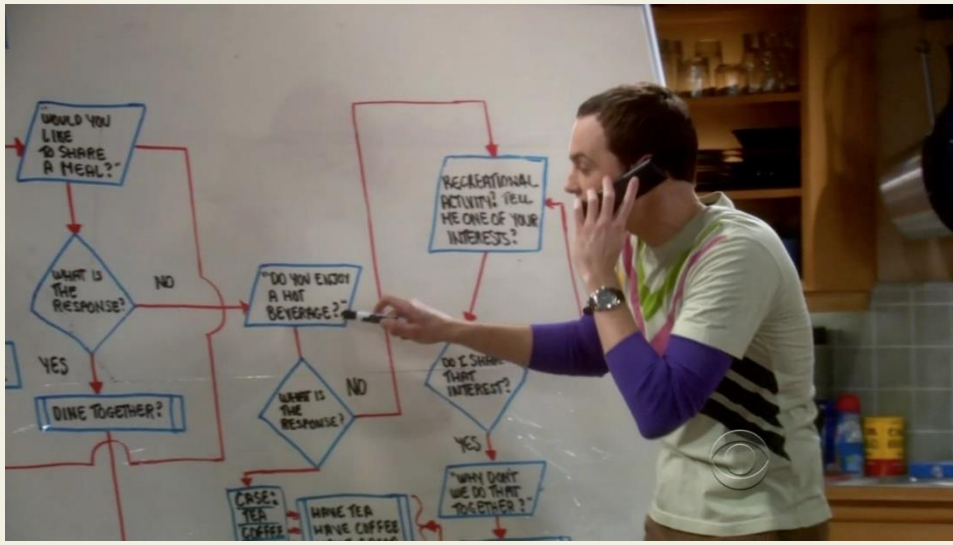
Szelekció



Iteráció

Algoritmus

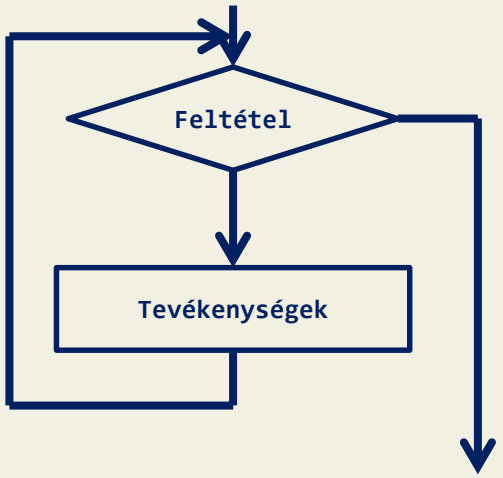
- Algoritmus
  - » lépésről lépésre definiált eljárás
  - » valamilyen probléma megoldására
  - » véges számú definiált szabály, utasítás
  - » véges lépésszám
    - ismétlődő utasítások



*The Big Bang Theory

Algoritmus

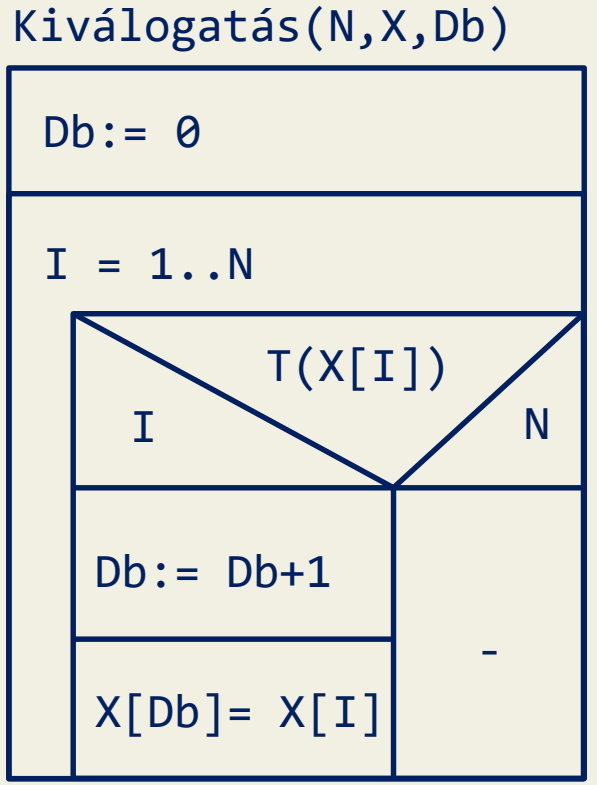
- Az algoritmus megadható:
  - » folyamatábra (flowchart)
  - » struktogram (structogram)
  - » pszeudokód vagy leíró nyelv (pseudocode)
  - » programkód (code)



Folyamatábra

```
void buborekRendez(double szamok[], int meret) {
  int i, j;
  // egyre rövidebb tömbrészletek ciklusa
  for (i = meret-1; i > 0; --i)
    // egymás utáni párok ciklusa
    for (j = 0; j < i; ++j)
      if (szamok[j+1] < szamok[j]){
         double temp = szamok[j];
         szamok[j] = szamok[j+1];
         szamok[j+1] = temp;
      }
}
```

Programkód



Struktogram

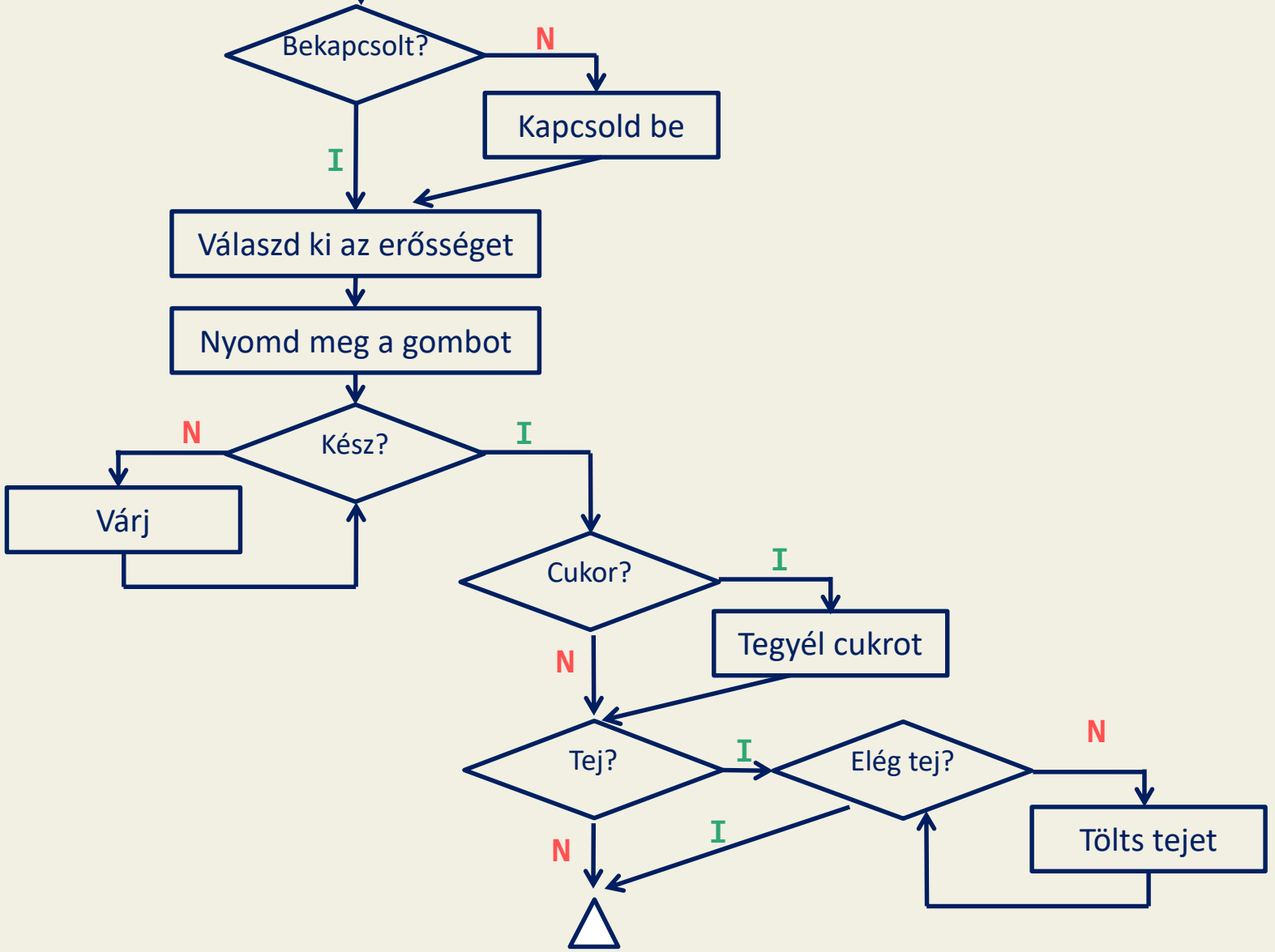
Pszeudokód - kávéfőzés

```
1.  Ha nincs bekapcsolva a kávéfőző,
2.  ⟶ kapcsold be a kávéfőzőt.
3.  Válaszd ki az erősséget.
4.  Nyomd meg az indítógombot.
5.  Amíg kész nincs,
6.  ⟶ várj.
7.  Ha szeretnél bele cukrot,
8.  ⟶ tegyél bele.
9.  Ha szeretnél bele tejet,
10. ⟶ amíg nem elég,
11. ⟶⟶ tölts tejet a kávéba.
12. Kész.
```

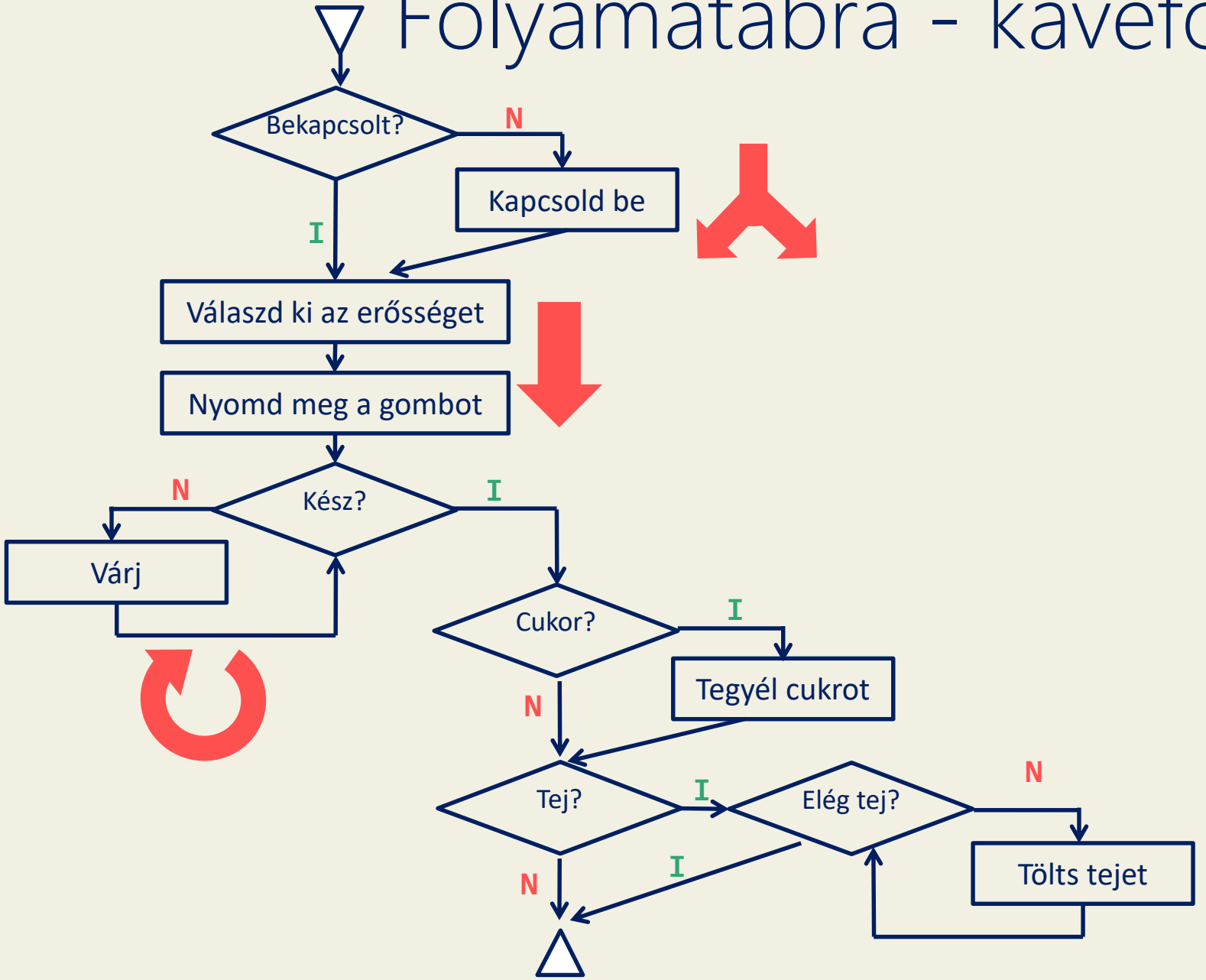
Pszeudokód - kávéfőzés

```
1.  Ha nincs bekapcsolva a kávéfőző,
2.  ⟶ kapcsold be a kávéfőzőt.
3.  Válaszd ki az erősséget.
4.  Nyomd meg az indítógombot.
5.  Amíg kész nincs,
6.  ⟶ várj.
7.  Ha szeretnél bele cukrot,
8.  ⟶ tegyél bele.
9.  Ha szeretnél bele tejet,
10. ⟶ amíg nem elég,
11. ⟶⟶ tölts tejet a kávéba.
12. Kész.
```

Folyamatábra - kávéfőzés

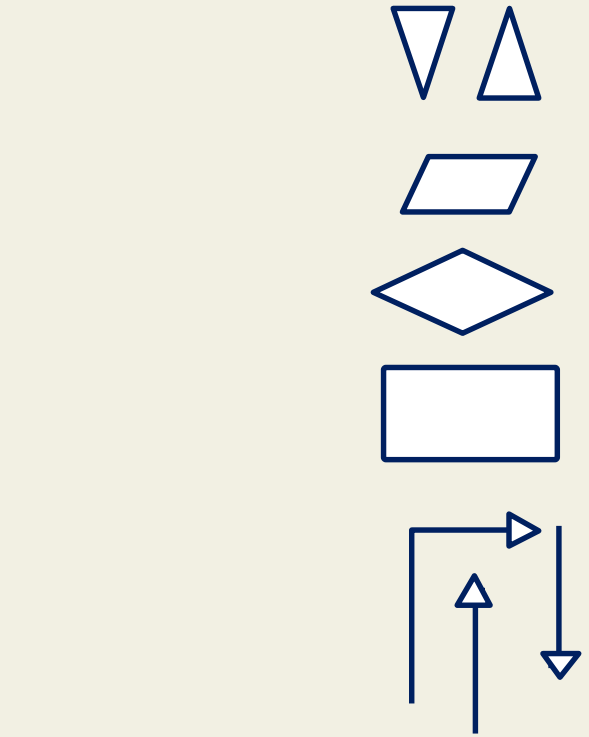


Folyamatábra - kávéfőzés



ANSI folyamatábra

- Belépés / kilépés
- Input / output
- Döntés
- Utasítás / utasítások
- Folyamatok iránya



Háromszög

Parallelogramma

Rombusz

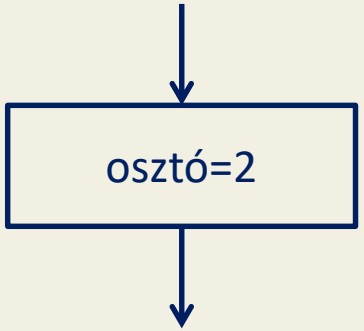
Téglalap

Nyíl

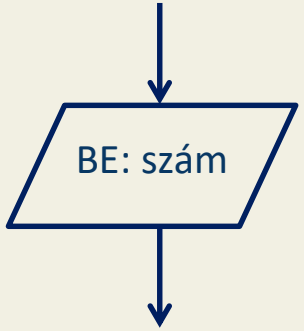
(ANSI - American National Standards Institute -**ANSI Standard Flowchart**)

Folyamatábra

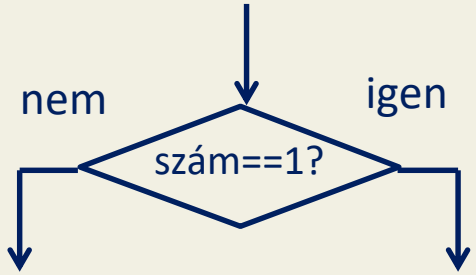
Kezdő/végpont



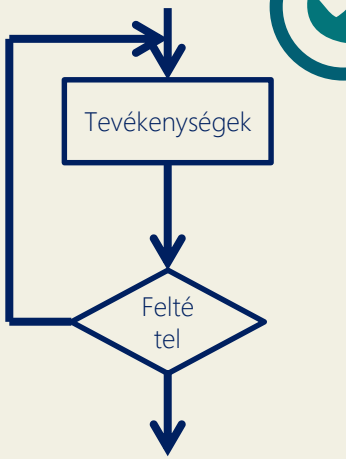
Művelet

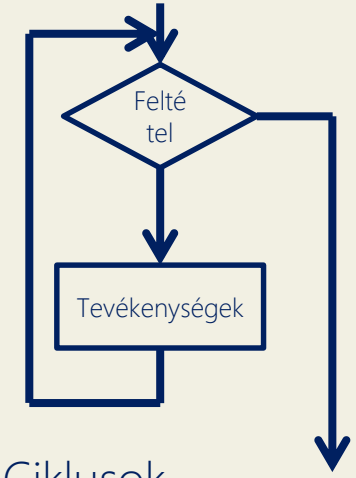


Felhasználói interakció



Elágazás



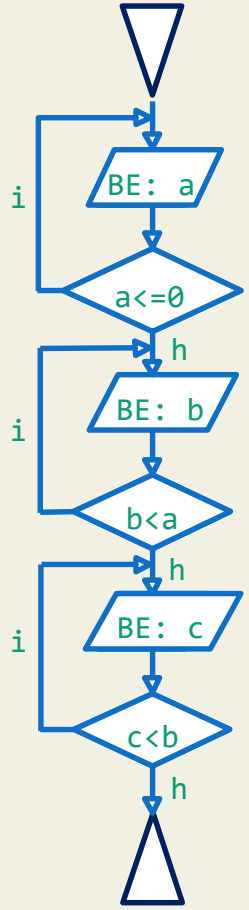


Ciklusok

Folyamatábra

Példa
Adjon meg három pozitív valós számot növekvő sorrendben!

(ANSI - American National Standards Institute)
(ISO - International Organization for Standardization)
(ANSI Standard Flowchart)

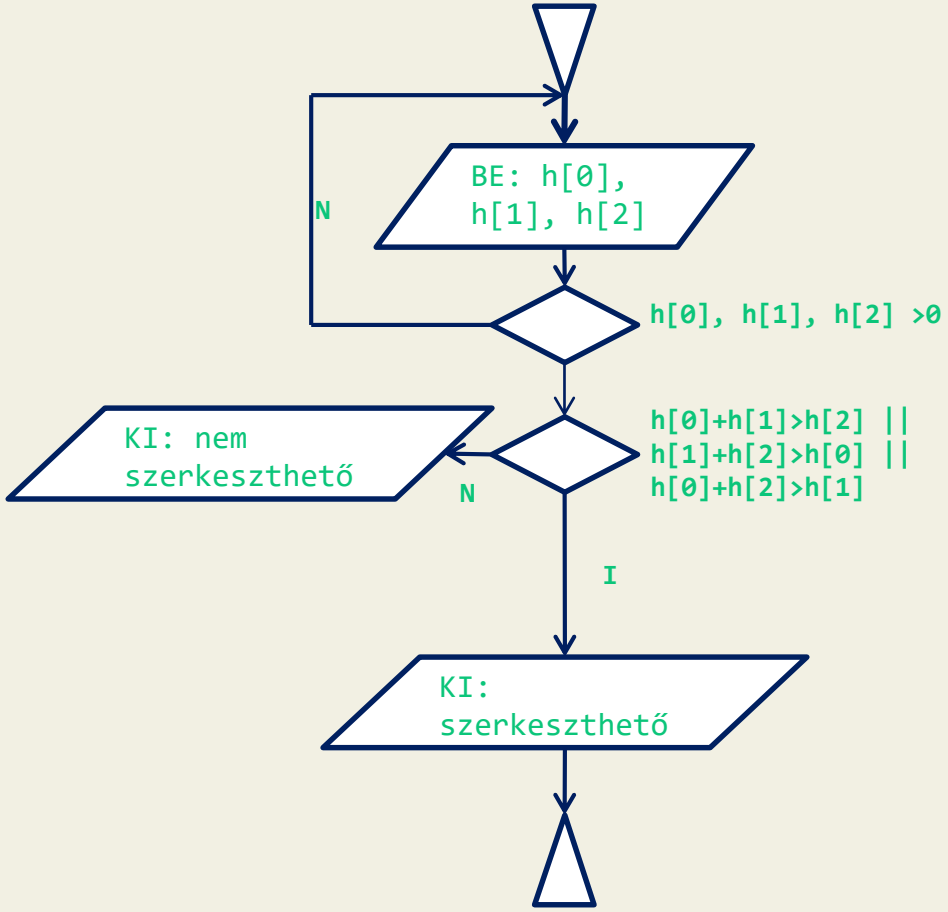


Folyamatábra

Állapítsa meg, hogy a háromszög a megadott oldalhosszakkal megszerkeszthető-e!

Első körben csak azt nézzük, hogy a 3 érték nagyobb-e, mint 0, és amint ilyen érték jött, vizsgáljuk a megszerkeszthetőséget.

A háromszög megszerkeszthető, ha bármely két oldal hosszának összege nagyobb, mint a harmadik oldal hossza.



Funkció	Azonosító	Típus
A háromszög oldalainak hosszát tároló tömb	h[3]	Valós elemekből álló háromelemű tömb

Önálló feladat

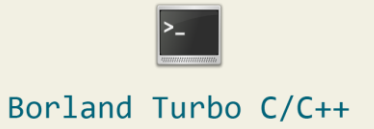
- Beolvasandó egy pozitív egész szám, n. Számítsuk ki az n! (faktoriális) értéket, azaz az egész számok szorzatát 1-től n-ig!

$$n! = 1 * 2 * 3 * \dots * (n-1) * n$$
$$4! = 24$$

- Alakítsa át az előző feladat megoldását úgy, hogy az növekményes ciklust tartalmazzon!

Integrált fejlesztői környezetek (IDE)

Borland Turbo C/C++



Code::Blocks

Microsoft Visual C++ 6.0

Visual Studio 2010

Visual Studio 2015

Az IDE tartalmazza

- a programkód szerkesztésére alkalmas szövegszerkesztőt
- a programmodulokból gépi kódot létrehozó fordítót (compiler)
- a gépi kód modulokból futtatható programot létrehozó linkert
- a programhibák felderítését megkönnyítő debugger rendszert
- és számos kényelmi funkciót.

ANSI C adattípusok

Típus	Méret bájtban (minimum)	Méret bitben (minimum)	Határok:
Egész			
char	1	8	-128 – 127
unsigned char	1	8	0 – 255
int	2	16	-32768 – 32767
unsigned int	2	16	0 – 65535
long	4	32	-2.147.483.647 - 2.147.483.647
unsigned long	4	32	0 – 4.294.967.295
Lebegőpontos			
float	4	32	$\pm 3.4*10^{-38}$ - $\pm 3.4*10^{+38}$ 6–7 decimális jegy pontosság
double	8	64	$\pm 1.7*10^{-308}$ - $\pm 1.7*10^{+308}$ 15–16 decimális jegy pontosság
long double	10	80	$\pm 1.2*10^{-4932}$ - $\pm 1.2*10^{+4932}$ 19 decimális jegypontosság

Fun fact

- 2015-ben a YouTube-on módosítani kellett a 32 bites változót, ami videók nézettségét mutatja, mivel egy videó közelébe ért a 2 milliárdos nézettségnek (milliárd, angolul billion = 10^9).
- A hivatalos bejegyzésben ezt írják: "We never thought a video would be watched in numbers greater than a 32-bit integer (=**2,147,483,647** views), but that was before we met PSY. "Gangnam Style" has been viewed so many times we had to upgrade to a 64-bit integer (9,223,372,036,854,775,808)!"

YouTube

Első programunk

```c
//Első program
#include <stdio.h>
void main(){
    printf("Hello World\n");
    getchar();
}
```

Második programunk

```c
//Második program
#include <stdio.h>
void main(){
 int i;
 i = 8;
 printf("Osszeg: %d\n", 1+2+3); //kiíratás
 printf("i: %d", i); //kiíratás
 getchar(); //billentyűre vár
}
```

Elöltesztelő ciklusok (pretest loops)

```
int alfa, beta;
for (alfa = 4; alfa <= 6; alfa = alfa + 1){
  printf("%d ", alfa);
}
// a { } ebben az esetben elhagyhatóak
printf("\n");

beta = 4;
while (beta <= 6){
  printf("%d ", beta);
  beta = beta + 1;
}
```

```
4 5 6
4 5 6
```

A ciklusokról később még lesz szó.

A két loop, a fenti módon használva ekvivalens, használjuk azt, amelyiket az adott helyzetben jobban átlátunk. A **for** ciklust nevezik növekményes ciklusnak is.

Gyakorló feladatok

1. `200 180 160 140 120 100 80 60 40`

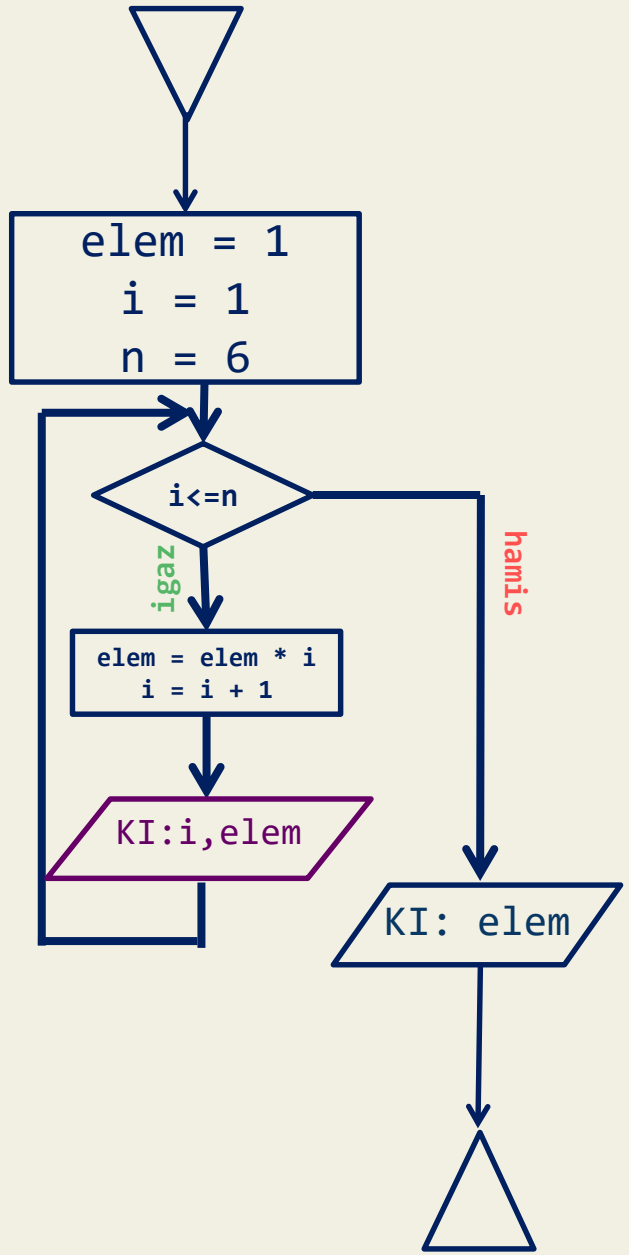
Írjon programot, amely 200-tól számol vissza 40-ig 20-asával.
(**`for`** és **`while`** ciklussal is)

2. `100-101 80-81 60-61 40-41`

Írjon programot, amely 100-tól számol vissza 40-ig 20-asával, de mindig kiírja az 1-el nagyobb számot is
(**`for`** és **`while`** ciklussal is)

Faktoriális

- *Az előző órai feladat, módosítva*: Adott egy pozitív egész szám, n. Számítsuk ki az $n!$ (faktoriális) értéket, azaz az egész számok szorzatát 1-től n-ig!
+ Íjuk ki az egyes lépések iterációját is.



Faktoriális

$$5! = 1 * 2 * 3 * 4 * 5 = 120$$

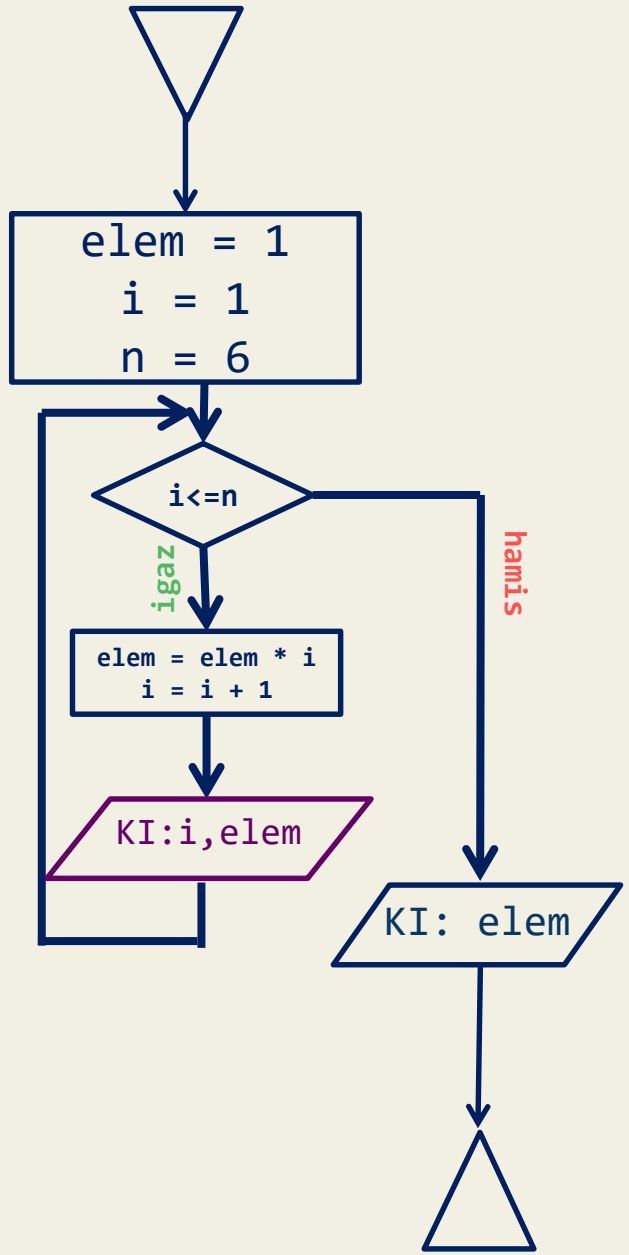
$$n! = 1 * 2 * 3 * \ldots * (n-1) * n$$

```
elem = 1;
i = 1;
while (i <= n){
      elem = elem * i;
      i = i + 1;
}
```

n	n!
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Faktoriális

```c
#include <stdio.h>
void main(){
  int i, n, elem; //deklaráció
  elem = 1; //inicializálás
  i = 1; //inicializálás
  n = 6; //inicializálás
  while (i <= n){ //ciklus kilépési feltétel
     elem = elem * i;
     i = i + 1;
     printf("i = %d, elem = %d\n", i, elem); //+
  }
  printf("%d", elem); //kiíratás
  getchar(); //billentyűre vár
}
```



Fahrenheit / Celsius

- Készítsen fahrenheit - celsius átváltását bemutató programot.

$$T_{C^\circ} = (T_{F^\circ} - 32) \times \frac{5}{9}$$

$$T_{F^\circ} = T_{C^\circ} \times \frac{9}{5} + 32$$

F°	C°
0	-17,8
20	-6,7
32	0
40	4,4
212	100
300	148,9

Fahrenheit / Celsius feladatok

Alap feladat: Fahrenheit - Celsius átváltó.

1. A Fahrenheit fok növekedjék `0`-tól `300`-ig `20`-asával!
2. A Fahrenheit fok `300`-tól `0`-ig csökkenjen `30`-asával!
3. A Fahrenheit `0`-tól `3000`-ig növekedjen `20`-asával! Az eredmény a képernyőn fejléccel ellátva lapozhatóan jelenjen meg!
4. Legyen ugyanaz a feladat, mint az előző pontban, de az eredmény a képernyőn fejléccel ellátva nem csak előre-, hanem előre-hátra lapozhatóan jelenjen meg.

Fahrenheit / Celsius [1]

- Írja ki táblázatos formában a Celsius és a hozzátartozó Fahrenheit értékeket, a Celsius fok növekedjék 0-tól 300-ig 20-asával!

$$T_{C^\circ} = (T_{F^\circ} - 32) \times \frac{5}{9}$$

$$T_{F^\circ} = T_{C^\circ} \times \frac{9}{5} + 32$$

Fahrenheit	Celsius
0	-17.8
20	-6.7
40	4.4
60	15.6
80	26.7
100	37.8
120	48.9
140	60.0
160	71.1
180	82.2
200	93.3
220	104.4
240	115.6
260	126.7
280	137.8
300	148.9

Fahrenheit / Celsius [1]

```c
#include <stdio.h>
void main() {
        int fahr;
        float also, felso, lepes;
        fahr = 0;
        also = 0; felso = 300; lepes = 20;
        printf("\nFahrenheit - Celsius  atszmitas\n\n");
        printf("Fahrenheit - Celsius\n");
        printf("--------------------\n");
        while (fahr <= felso){
                printf("%10d %10.1f\n", fahr, ((5.0 / 9.0)*(fahr - 32.0)));
                fahr = fahr + lepes;
        }
        getchar();
}
```

Fahrenheit / Celsius [1]

```
#include <stdio.h>
void main() {
      float fahr, also, felso, lepes;
      also = 0;
      felso = 300;
      lepes = 20;
      printf("\nFahrenheit - Celsius  atszmitas\n\n");
      printf("Fahrenheit - Celsius\n");
      printf("---------------------\n");
      for (fahr = also; fahr <= felso; fahr = fahr + lepes)
        printf("%10.0f%10.1f\n", fahr, (5.0 / 9.0)*(fahr - 32));
      getchar();
}
```

Fahrenheit / Celsius [2]

```
#include <stdio.h>
void main() {
      float fahr, also, felso, lepes;
      also = 0;
      felso = 300;
      lepes = 30;
      printf("\nFahrenheit - Celsius  atszmitas\n\n");
      printf("Fahrenheit - Celsius\n");
      printf("---------------------\n");
      for (fahr = felso; fahr >= also; fahr = fahr - lepes)
        printf("%10.0f%10.1f\n", fahr, (5.0 / 9.0)*(fahr - 32));
      getchar();
}
```

Fahrenheit / Celsius [3]

```c
#define ALSO 0
#define FELSO 3000
#define LEPES 20
#define LAPSOR 20
#include <stdio.h>

void main(){
  int fahr, sor;
  sor = LAPSOR;
  printf("\nFahrenheit - Celsius atszamitas:\n");
  for (fahr = ALSO; fahr <= FELSO; fahr += LEPES) {
    if (sor == LAPSOR) {
      printf("\n\n\nFahrenheit - Celsius\n");
      printf("--------------------\n");
      sor = 0;
    }
    sor++;
    printf("%10d%10.1f\n",fahr,(5./9.)*(fahr - 32));
    if (sor == LAPSOR) {
      printf("\nA tovabbitashoz nyomjon ENTERt!");
      while (getchar() != '\n');
    }
  }
}
```

Lapozás

Fahrenheit / Celsius [5] [6]

5. A Fahrenheit **`0`**-tól **`580`**-ig növekedjen **`20`**-asával! Az eredmény a képernyőn fejléccel ellátva két oszloppárban oszlopfolytonosan haladva jelenjen meg, azaz a bal oldali oszloppár `0`-val, a jobb oldali viszont `300`-zal kezdődjék és mindegyik oszloppár **`20`**-asával haladjon!

6. Ugyanez, csak sorfolytonosan, tehát a baloldali oszlop `0`-val kezdődik a jobboldali pedig `20`.

Fahrenheit / Celsius [5] [6]

Fahrenheit	- Celsius	Fahrenheit	- Celsius
0	-17.8	300	148.9
20	-6.7	320	160.0
40	4.4	340	171.1
60	15.6	360	182.2
80	26.7	380	193.3
100	37.8	400	204.4
120	48.9	420	215.6
140	60.0	440	226.7
160	71.1	460	237.8
180	82.2	480	248.9
200	93.3	500	260.0
220	104.4	520	271.1
240	115.6	540	282.2
260	126.7	560	293.3
280	137.8	580	304.4

Fahrenheit	- Celsius	Fahrenheit	- Celsius
0	-17.8	20	-6.7
40	4.4	60	15.6
80	26.7	100	37.8
120	48.9	140	60.0
160	71.1	180	82.2
200	93.3	220	104.4
240	115.6	260	126.7
280	137.8	300	148.9
320	160.0	340	171.1
360	182.2	380	193.3
400	204.4	420	215.6
440	226.7	460	237.8
480	248.9	500	260.0
520	271.1	540	282.2
560	293.3	580	304.4

Fahrenheit / Celsius [6]

```c
#define ALSO 0
#define FELSO 580
#define LEPES 20
#include <stdio.h>
void main() {
	int fahr;
	char c;
	c = ' ';
	printf("Fahrenheit - Celsius Fahrenheit - Celsius\n");
	printf("---------------------------------------\n");
	for (fahr = ALSO; fahr <= FELSO; fahr = fahr + LEPES) {
		printf("%10d%10.1f%c", fahr, (5.0 / 9.0)*(fahr - 32), c);
		if (c == ' ') c = '\n'; else c = ' ';
	}
}
```

Fahrenheit / Celsius [7]

7. Maradva a sorfolytonos megjelentetésnél kérjük be előbb a kijelzendő oszloppárok számát ellenőrzött inputtal! Az oszloppárok száma 1, 2, 3 vagy 4 lehet. A még kijelzendő felső érték ennek megfelelően 280, 580, 880 vagy 1180. Az eredmény a képernyőn fejléccel ellátva az előírt számú oszloppárban jelenjen meg úgy, hogy 20 a lépésköz!

```
maxOszlop = getchar() - '0';
```

Fahrenheit / Celsius [7]

```
Fahrenheit - Celsius átszámítás
Oszlop-párok száma(1-4)? 3
Fahrenheit  Celsius Fahrenheit  Celsius Fahrenheit  Celsius
-----------------------------------------------------------
         0    -17.8         20     -6.7         40      4.4
        60     15.6         80     26.7        100     37.8
       120     48.9        140     60.0        160     71.1
       180     82.2        200     93.3        220    104.4
       240    115.6        260    126.7        280    137.8
       300    148.9        320    160.0        340    171.1
       360    182.2        380    193.3        400    204.4
       420    215.6        440    226.7        460    237.8
       480    248.9        500    260.0        520    271.1
       540    282.2        560    293.3        580    304.4
       600    315.6        620    326.7        640    337.8
       660    348.9        680    360.0        700    371.1
       720    382.2        740    393.3        760    404.4
       780    415.6        800    426.7        820    437.8
       840    448.9        860    460.0        880    471.1
Press any key to continue . . .
```

Stepik

- ZH
- https://stepik.org/
- Regisztráció

Példa (Stepik) 1. (ZH1)

Beolvasandó a szabvány bemenetről (kb. billentyűzet) pontosan 3 darab, egymástól szóközökkel elválasztott, formailag helyes szám, melyek közül az első kettő valós, a harmadik pozitív egész. Az első szám egy mértani sorozat első tagja, a második a kvóciens. A harmadik szám annak a tagnak a sorszáma, ahányadikat a programnak ki kell számítania, majd meg kell jelenítenie a szabvány kimeneten három tizedes pontossággal. Semmi mást ne jelenítsen meg! Ügyeljen rá, hogy az, és csak az jelenjen meg a kimeneten, amit a feladat előír!

1. példa
Bemenet: **1.0 2.0 8**
Kimenet: **128.000**

2. példa
Bemenet: **1 2 8**
Kimenet: **128.000**

3. példa
Bemenet: **1 2 9**
Kimenet: **256.000**

4. példa
Bemenet: **2.1 -0.5 4**
Kimenet: **-0.263**

5. példa
Bemenet: **1.23456 10 3**
Kimenet: **123.456**

6. példa
Bemenet: **-1.2 -3.4 5**
Kimenet: **-160.360**

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

int main() {
  double t1, q;
  int n;
  scanf("%lf %lf %d", &t1, &q, &n);
  printf("%.3f", t1*pow(q, n - 1));
  return 0;
}
```

Példa (Stepik) 2. (ZH1)

Beolvasandó a szabvány bemenetről (kb. billentyűzet) pontosan 10 darab, egymástól szóközökkel elválasztott, formailag helyes egész szám. Állapítsa meg, hogy a számok mértani sorozatot alkotnak-e! (Egy mértani sorozatban a másodiktól kezdve bármelyik tag és az azt megelőző tag hányadosa állandó.) Az eredménytől függően jelenítse meg az **igen** vagy a **nem** üzenetet! Semmi mást ne jelenítsen meg! Ügyeljen rá, hogy az, és csak az jelenjen meg a kimeneten, amit a feladat előír!

1. példa

Bemenet: `-44 -44 -44 -44 -44 -44 -44 -44 -44 -44`

Kimenet: **igen**

2. példa

Bemenet: `1 2 4 8 16 32 64 128 256 512`

Kimenet: **igen**

3. példa

Bemenet: `-65 -65 -585 585 3510 10530 737 -4422 -13 79`

Kimenet: **nem**

```c
#include <stdio.h>
int main() {
  int elozo, kovetkezo, q, i;
  scanf("%d", &elozo);
  scanf("%d", &kovetkezo);
  q = kovetkezo / elozo;
  for (i = 2; i<10; i++) {
    elozo = kovetkezo;
    scanf("%d", &kovetkezo);
    if (kovetkezo / elozo != q) {
      printf("nem");
      return 0;
    }
  }
  printf("igen");
  return 0;
}
```

Példa (Stepik) 3. (ZH1)

Beolvasandó a szabvány bemenetről (kb. billentyűzet) egy legfeljebb 50 karakter hosszú sztring, ami az angol ábécé kis- és nagybetűit és szóközöket is tartalmazhat. Írja ki a sztringben szereplő betűk ábécében elfoglalt helyét. A kis- és nagybetűket is kezelnie kell a programnak! A számokat szóközzel válassza el egymástól! A sztringben szereplő szóközök helyén és a kimenet végén tegyen sortörést!

1. példa

Bemenet: **ABC cba Pelda mondat XyZ**

Kimenet:

1 2 3
3 2 1
16 5 12 4 1
13 15 14 4 1 20
24 25 26

```c
#include <stdio.h>

int main() {

  char str[51] = "";
  int i;
  fgets(str, 50, stdin);
  for (i = 0; str[i] != '\0'; i = i + 1) {
    if (str[i] >= 'a' && str[i] <= 'z')
      printf("%d ", str[i] - 'a' + 1);
    else if (str[i] >= 'A' && str[i] <= 'Z')
      printf("%d ", str[i] - 'A' + 1);
    if (str[i] == ' ')
      printf("\n");
    }
  return 0;
}
```

Példa (Stepik) 4. (ZH1)

Beolvasandó a szabvány bemenetről (kb. billentyűzet) pontosan 50 darab, egymástól szóközökkel elválasztott, formailag helyes egész szám. Állapítsa meg, hogy a megadottak között mi a *legkisebb* érték! Semmi mást ne jelenítsen meg! Ügyeljen rá, hogy az, és csak az jelenjen meg a kimeneten, amit a feladat előír!

1. példa

Bemenet: **-12 33 25 32 62 -44 -88 48 30 -21 52 -93 -77 74 -1 -18 -48 29 -17 -78 39 45 -1 41 40 58 -21 86 -28 -41 58 34 7 59 94 17 42 38 87 82 -400 -2 -69 48 -30 -38 -84 -91 79 -47**

Kimenet: **-400**

```c
#include <stdio.h>
int main() {
  int szam, i, min;
  for (i = 0; i < 50; i++) {
    scanf("%d", &szam);
    if (i == 0)
      min = szam;
    if (szam < min)
      min = szam;
  }
  printf("%d", min);
  return 0;
}
```

Példa (Stepik) 5. (ZH2)

Beolvasandó a szabvány bemenetről (kb. billentyűzet) pontosan 6 darab, egymástól szóközökkel elválasztott, formailag helyes egész szám. Írja ki a szabvány kimenetre a számokat **növekvő sorrendben**. Semmi mást ne jelenítsen meg! Ügyeljen rá, hogy az, és csak az jelenjen meg a kimeneten, amit a feladat előír!

1. példa

Bemenet: **168 60 93 167 0 146**
Kimenet: **0 60 93 146 167 168**

2. példa

Bemenet: **12 130 168 999 91 131**
Kimenet: **12 91 130 131 168 999**

```c
#include <stdio.h>

void buborekRendezJav(int szamok[], int meret) {
int i, j, voltCsere = 1;
// egyre rövidebb tömbrészletek ciklusa
  for (i = meret - 1; i>0 && voltCsere; --i) {
  voltCsere = 0;
    for (j = 0; j<i; ++j) {
      if (szamok[j + 1] < szamok[j]) {
        int temp = szamok[j];
        szamok[j] = szamok[j + 1];
        szamok[j + 1] = temp;
        voltCsere = 1;
      }}}}

int main() {
  int szam[6], i, min;
  for (i = 0; i < 6; i++)
    scanf("%d", &szam[i]);
  buborekRendezJav(szam, 6);
  for (i = 0; i < 6; i++)
    printf("%d ", szam[i]);
  return 0;
}
```

Vezérlő szerkezetek

- Program: utasítások rendezett halmaza
  - » végrehajtás - folyamat
- Programozási szerkezetek
  - » szekvencia
  - » feltételes elágazások
  - » ciklusok
  - » eljárás absztrakció

Szekvencia

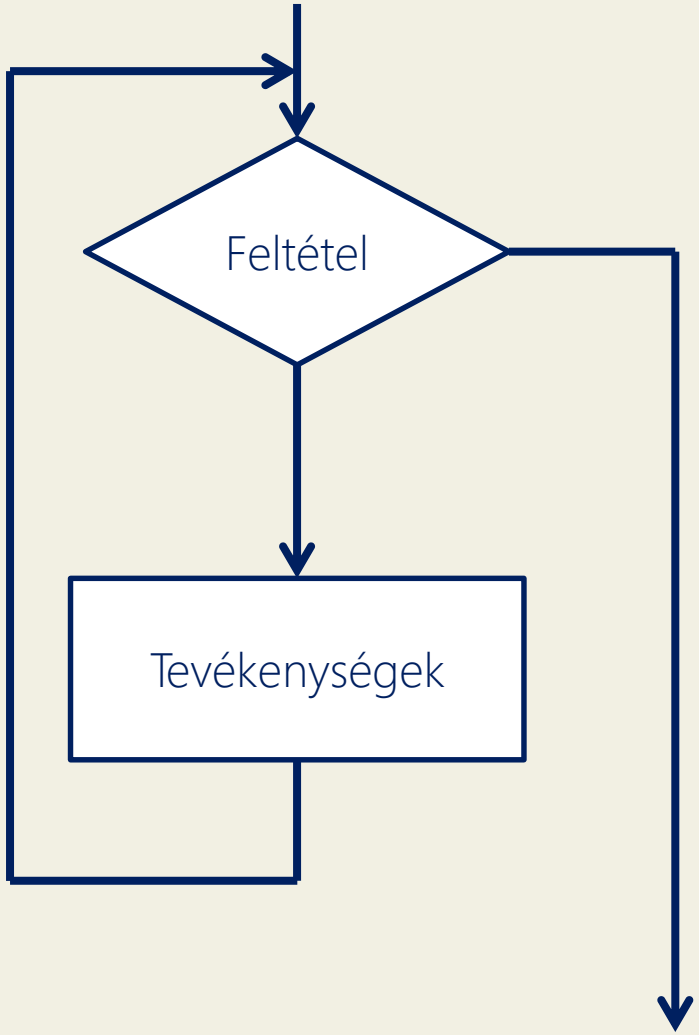
Szekvencia alatt egymás után végrehajtott utasításokat értünk.

```
utasítás 1;
   utasítás 2;
   ...
   utasítás n;
```

```
utasítás példa:
   a = 5;                       //értékadás
   osszeg = x + y;              //értékadás
   a = a + 4;                   //értékadás
   printf("Ez egy program"); //függvényhívás
```

Elöltesztelő ciklus (pretest loop)

- Az előltesztelő ciklus a feltételt a ciklustörzsbe lépés *előtt* ellenőrizi.
- Annyiszor hajtódik végre, ahányszor a *feltétel* teljesül.
- A végrehajtás száma lehet 0 is, ha a törzs egyszer sem hajtódik végre.
- A feltételnek előbb-utóbb hamissá kell válnia, különben *végtelen ciklus* (infinite loop) keletkezik.



Elöltesztelő ciklus (pretest loop)

```
int alfa, beta;
for (alfa = 4; alfa <= 6; alfa = alfa + 1){
  printf("%d ", alfa);
}
// a { } ebben az esetben elhagyhatóak
printf("\n");

beta = 4;
while(beta <= 6){
  printf("%d ", beta);
  beta = beta + 1;
}
```

```
4 5 6
4 5 6
```

A két loop, a fenti módon használva ekvivalens, használjuk azt, amelyiket az adott helyzetben jobban átlátunk. A **for** ciklust nevezik növekményes ciklusnak is.

For ciklus

A for beállít egy kezdőértéket, adott feltételig ismétli a ciklusmagot, illetve növeli a megadott módon a növekményt. A for ciklus 3 paramétere közül tetszőlegesen elhagyhatunk.

```
for(kezdőérték; feltétel; növekmény){
     utasításblokk
  }
```

Példa:

```
  for(i = 1; i <= 100; i = i + 1)
     printf("C programozás");
```

While ciklus - elöltesztelő

amíg a feltétel igaz - utasítás(ok)

```
while(feltétel_igaz){
      utasításblokk
   }
```

Példa:

```
beta = 4;
while(beta <= 8){
   printf("%d ", beta);
   beta = beta + 1;
}
```

```
4 5 6 7 8
```

While

amíg a feltétel igaz - utasítás(ok)

```
while(feltétel){
     utasításblokk
  }
```

```
kifejezés = true;
```

Végtelen ciklus	0 iteráció
while(kifejezés)	while(!kifejezés)
while(kifejezés == 1)	while(kifejezés == 0) while(kifejezés != 1)

Do While ciklus - hátultesztelő

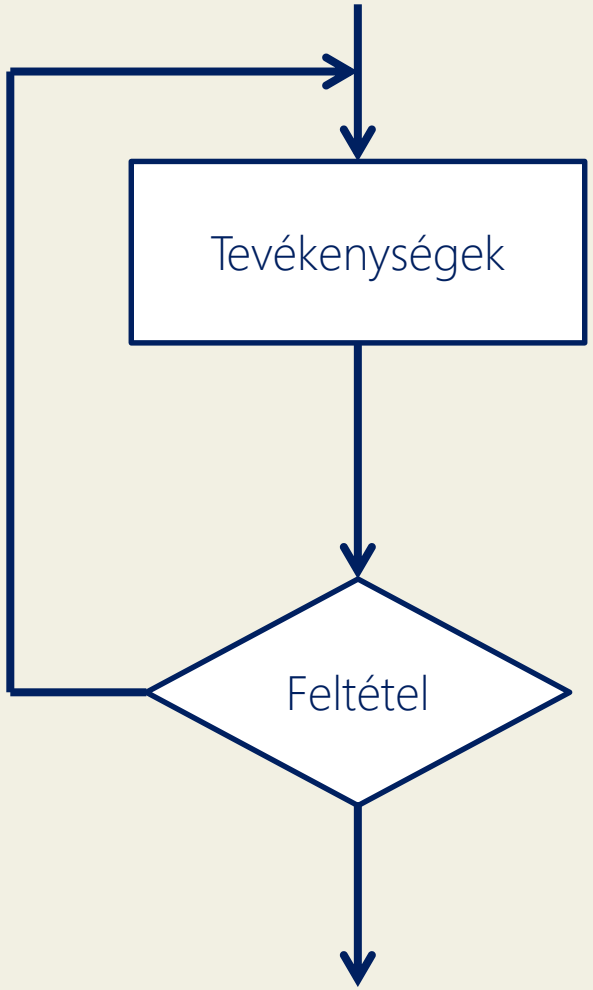
utasítás(ok) - amíg a feltétel igaz

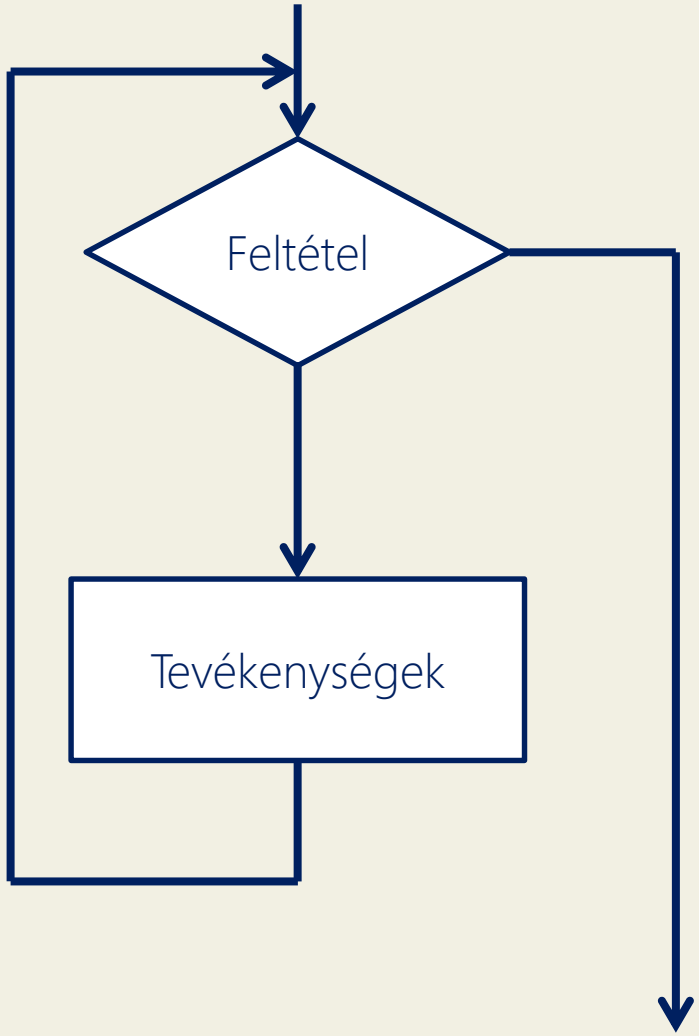
```
do{
      utasításblokk
}while(feltétel);
```

Példa:

```
  i=1;
  do{
      printf("C programozás");
      i = i + 1;
  }while(i<=100);
```

Do..While vs. While
(pretest loop vs. posttest loop)





Hogyan írhatunk **`while`** ciklussal **`do..while`**-t?

```
do_work();  //akár több utasítás
while (feltétel) {
    do_work(); //ugyanaz, mint fent
}
```

Szelekció

```
ha a feltétel igaz
     utasítás(ok)
```

```
if()
```

```
ha a feltétel igaz
     utasítás(ok)
különben
     utasítás(ok)
```

```
if()
else
```

```
ha a feltétel 1 igaz
     utasítás(ok)
különben ha a feltétel 2 igaz
     utasítás (ok)
...
különben ha a feltétel n igaz
     utasítás(ok)
különben
     utasítás(ok)
```

```
if()
else if()
else if()
else if()
else if()
...
else
```

Szelekció példa

ha az összeg kisebb, mint 0

    írd ki, hogy "Az összeg negatív."

különben ha az összeg 0

    írd ki, hogy "Az összeg 0."

különben (minden más esetben)

    írd ki, hogy "Az összeg pozitív."

```
if(osszeg<0)
        printf("Az összeg negatív.");
else if(osszeg==0)
        printf("Az összeg 0.");
else
        printf("Az összeg pozitív.");
```

Boolean logika

- Q: 0 is false and 1 is true, right?
- A: 1.

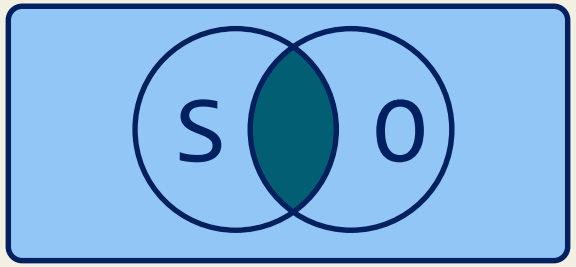
```
#define FALSE 0 // hamis
#define TRUE  1 // igaz - 1, de sokszor minden, ami nem 0
```

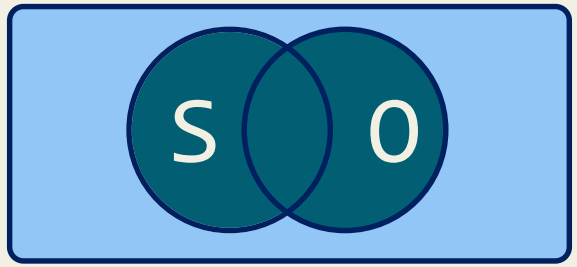
Eselbrücke *

1gaz hømis

* Német nyelvterületen a memorizálást segítő "gondolati kapcsot" jelenti

Boolean logika - Venn diagramok, igazságtáblák





!-nem

a	!a
0	1
1	0

&&-és

&&	0	1
0	0	0
1	0	1

Az ÉS (**AND**) logikai értéke csak abban az esetben IGAZ, ha <u>minden</u>, a műveletben részt vevő kifejezés logikai értéke IGAZ.

||-vagy

||	0	1
0	0	1
1	1	1

A VAGY (**OR**) művelet logikai értéke akkor IGAZ, ha a műveletben részt vevő kifejezések logikai értékének <u>bármelyike</u> IGAZ.

Printf

- `#include <stdio.h>`
- `printf("Szöveg %d", szam);`
- Formázott kiírás

%i or %d	int (egész típus)
%c	char
%f	float (lebegőpontos)
%lf	double (lebegőpontos)
%s	string

Printf

```
printf("%d\n", 7);
printf("%3d\n", 7);
printf("%03d\n", 7);
```

```
7
  7
007
```

Printf

```
printf("%f\n", 3.141592);
printf("%10f\n", 3.141592);
printf("%10.3f\n", 3.141592);
printf("%10.2f\n", 3.141592);
```

```
3.141592
  3.141592
     3.141
      3.14
```

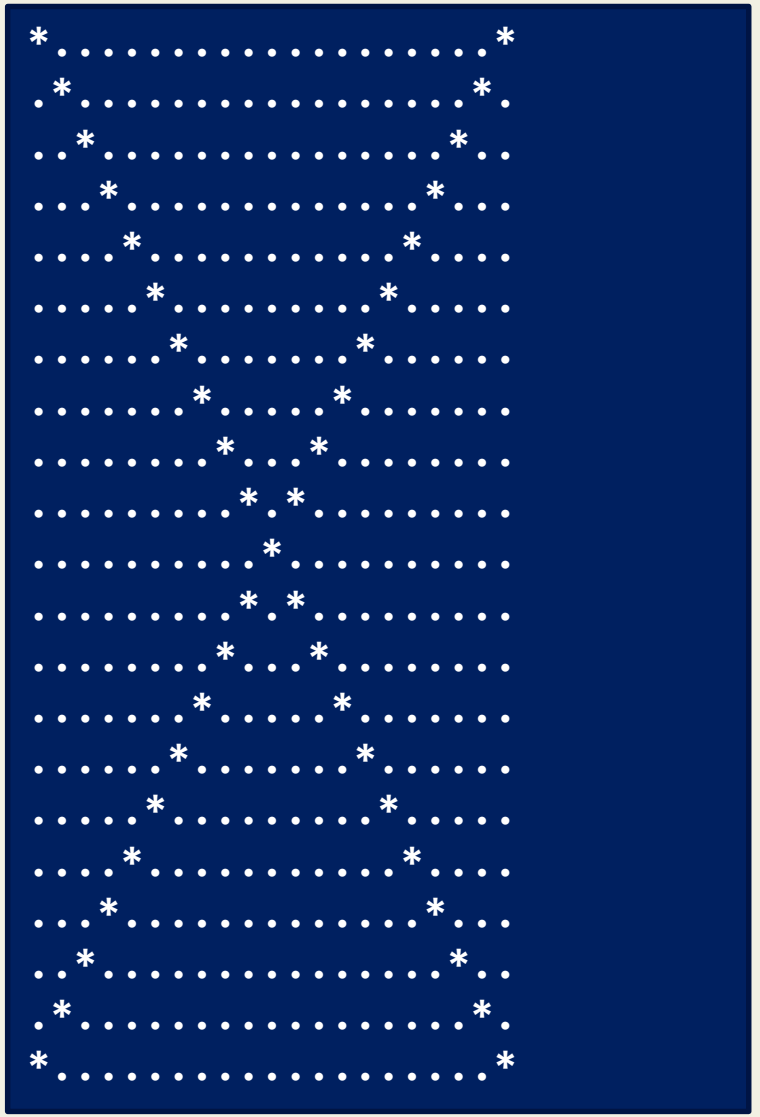
Printf

Szekvencia:	Funkció:	Karakter:
\0	karakterlánc vége	NUL
\a	fütty	BEL
\b	visszatörlés	BS
\t	vízszintes tab	HT
\n	új sor	LF
\v	függőleges tab	VT
\f	lapdobás	FF
\r	kocsi vissza	CR
\"	macskaköröm	"
\'	aposztróf	'
\?	kérdőjel	?
****	fordított per jel	\
\ooo	max. 3 oktális számjegy	bármi
\xhhh	max. 3 hexadecimális számjegy	bármi
\Xhhh	max. 3 hexadecimális számjegy	bármi

For ciklus gyakorlás - csillag

Készítsen programokat, melyek a karakteres üzemmódú képernyőn egy 21x21-es ablakában csillag karakterekkel megjelentetnek:

- egy keresztet a 10. sor és 10. oszlop feltöltésével!
- a főátlót (bal felső sarokból a jobb alsóba menőt)!
- a mellékátlót (a másik átlót)!
- egyszerre mindkét átlót, azaz egy X-et!



For ciklus gyakorlás - háromszög

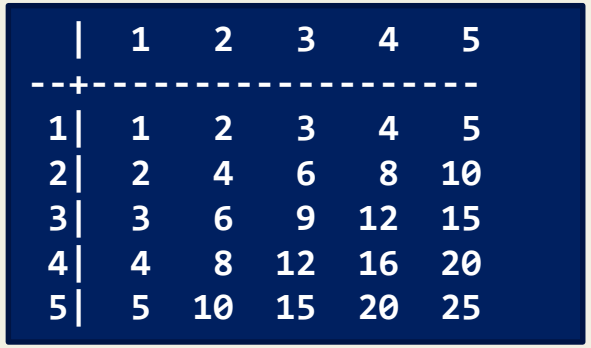
- Írjon programot, amely bekér a felhasználótól egy számot (n), és utána egy akkora háromszöget rajzol a képernyőre **"A"** betűkből, hogy annak éppen n sora van! Például n=5 esetén az oldalt látható ábra keletkezzen.
- Segítség:

```
int n;
char ch;
printf("Hany sor legyen? ");
ch = getchar();
n = ch - '0';
```

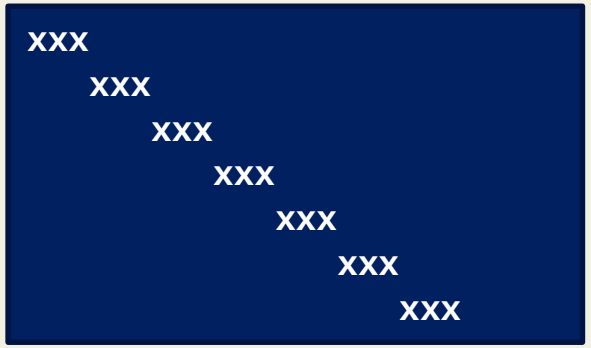
```
    A
   AAA
  AAAAA
 AAAAAAA
AAAAAAAAA
```

```
    A
   BBB
  CCCCC
 DDDDDDD
EEEEEEEEE
FFFFFFFFFFF
```

Egyéb gyakorlófeladatok



```
  |  1   2   3   4   5
--+--------------------
 1|  1   2   3   4   5
 2|  2   4   6   8  10
 3|  3   6   9  12  15
 4|  4   8  12  16  20
 5|  5  10  15  20  25
```



```
xxx
   xxx
      xxx
         xxx
            xxx
               xxx
                  xxx
```

```
#####
 ###
  #
```

```
\     /
 \   /
  \ /
```

```
  ooooooo
 ooooooo
ooooooo
```

```
VVV
 VV
  V
```

For ciklus gyakorlás - csillag

```
#include <stdio.h>
#define MERET 20 // konstans
void main() {
  int i,j;
  for (i=0; i<=MERET; i++) {
    for (j=0; j<=MERET; j++)
      if (j==i || j==1 )
                printf("*");
       else printf(".");
    printf("\n");
  }
  getchar();
}
```

For ciklus gyakorlás - háromszög

```c
#include <stdio.h>
void main(){
  int n, sor, szokoz, a;
  char ch;
  printf("Hany sor legyen? ");
  ch = getchar();
  n = ch - '0';
  // sorok ciklusa
  for (sor = 0; sor < n; sor = sor + 1) {
    // a sor elején kell valamennyi szóköz
    for (szokoz = 0; szokoz < n - sor; szokoz = szokoz + 1)
      printf("%d", szokoz); // helyette szóköz
    // utaná valahány darab 'A'
    for (a = 0; a < sor * 2 + 1; a = a + 1)
      printf("%c", sor + 'A');
    // sor vége
    printf("\n");
  }
}
```

```
Hany sor legyen? 9

012345678A
01234567BBB
0123456CCCCC
012345DDDDDDD
01234EEEEEEEEE
0123FFFFFFFFFFF
012GGGGGGGGGGGG
01HHHHHHHHHHHHHH
0IIIIIIIIIIIIIIII
```

Ciklusok gyakorlás

- Írja ki a páratlan számokat 1-től 20-ig!

```
Az aktuális számunk 1!
Amíg a szám kisebb vagy egyenlő, mint 20
  Írja ki a számot!
  Írjon ki egy szóközt!
  Növelje a számot kettővel!
```

```
Az aktuális számunk 1!
Amíg a szám kisebb vagy egyenlő, mint 20
  Ha a szám nem osztható kettővel (sz%2)
    Írja ki a számot!
    Írjon ki egy szóközt!
  Növelje a számot eggyel!
```

- Írjuk ki két pozitív egész szám legnagyobb közös osztóját!
  Induljunk ki a kisebb számból, és csökkentsük míg osztó nem osztója mindkét számnak.

Ciklusok gyakorlás

- Írjon C nyelvű programot, amely fordítási időben definiált egy **n** pozitív egész számhoz (n <= 25) kiírja a hozzá tartozó úgynevezett latin négyzetet a képernyőre. Ez az **1** és **n** közötti egész számok permutációinak egy adott sorozata.
  Például **n==5** esetén a latin négyzet: (segítség: **%** maradékos osztás)

```
1 2 3 4 5
2 3 4 5 1
3 4 5 1 2
4 5 1 2 3
5 1 2 3 4
```

Char

```
0          - NULL karakter ('\0')

32         - space (' ')

48 - 57    - '0' - '9'

65 - 90    - 'A' - 'Z'

97 -122    - 'a' - 'z'
```

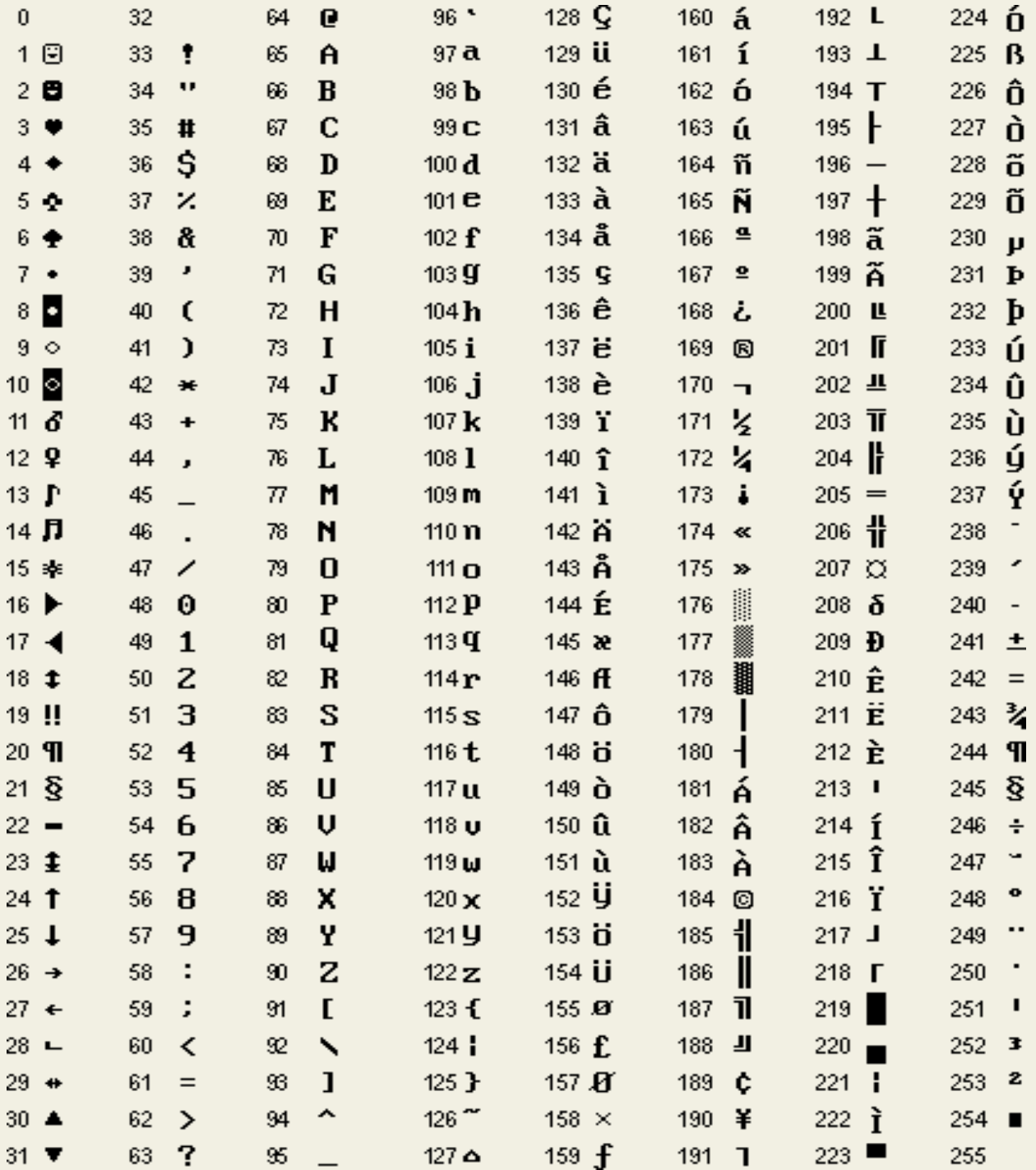
(97 – 65 = 32 – a különbség nagy és kis betűk közt)

Char

```
c = 'A';
c = 65;       // 'A' ASCII kódja
c = '\x41';   // a hexadecimalis ábrázolás \x-el kezdődik
```

http://www.wolframalpha.com/input/?i=0x41

ASCII


0 32 64 @ 96 ` 128 Ç 160 á 192 └ 224 Ó
1 33 ! 65 A 97 a 129 ü 161 í 193 ┴ 225 ß
2 34 " 66 B 98 b 130 é 162 ó 194 ┬ 226 Ô
3 35 # 67 C 99 c 131 â 163 ú 195 ├ 227 Ò
4 36 $ 68 D 100 d 132 ä 164 ñ 196 ─ 228 õ
5 37 % 69 E 101 e 133 à 165 Ñ 197 ┼ 229 Õ
6 38 & 70 F 102 f 134 å 166 ª 198 ã 230 µ
7 39 ' 71 G 103 g 135 ç 167 º 199 Ã 231 Þ
8 40 (72 H 104 h 136 ê 168 ¿ 200 ╚ 232 þ
9 41) 73 I 105 i 137 ë 169 ® 201 ╔ 233 Ú
10 42 * 74 J 106 j 138 è 170 ¬ 202 ╩ 234 Û
11 43 + 75 K 107 k 139 ï 171 ½ 203 ╦ 235 Ù
12 44 , 76 L 108 l 140 î 172 ¼ 204 ╠ 236 ý
13 45 - 77 M 109 m 141 ì 173 ¡ 205 ═ 237 Ý
14 46 . 78 N 110 n 142 Ä 174 « 206 ╬ 238 ¯
15 47 / 79 O 111 o 143 Å 175 » 207 ¤ 239 ´
16 48 0 80 P 112 p 144 É 176 ░ 208 ð 240 -
17 49 1 81 Q 113 q 145 æ 177 ▒ 209 Ð 241 ±
18 50 2 82 R 114 r 146 Æ 178 ▓ 210 Ê 242 =
19 51 3 83 S 115 s 147 ô 179 │ 211 Ë 243 ¾
20 52 4 84 T 116 t 148 ö 180 ┤ 212 È 244 ¶
21 53 5 85 U 117 u 149 ò 181 Á 213 ı 245 §
22 54 6 86 V 118 v 150 û 182 Â 214 Í 246 ÷
23 55 7 87 W 119 w 151 ù 183 À 215 Î 247 ¸
24 56 8 88 X 120 x 152 ÿ 184 © 216 Ï 248 °
25 57 9 89 Y 121 y 153 Ö 185 ╣ 217 ┘ 249 ¨
26 58 : 90 Z 122 z 154 Ü 186 ║ 218 ┌ 250 ·
27 59 ; 91 [123 { 155 ø 187 ╗ 219 █ 251 ¹
28 60 < 92 \ 124 ¦ 156 £ 188 ╝ 220 ▄ 252 ³
29 61 = 93] 125 } 157 Ø 189 ¢ 221 ¦ 253 ²
30 62 > 94 ^ 126 ~ 158 × 190 ¥ 222 Ì 254 ■
31 63 ? 95 _ 127 ⌂ 159 ƒ 191 ┐ 223 ▀ 255

For ciklus 1

```
for(i = 0; i < 40; i = i + 1){
    printf("%4d", i);
}
```

```
   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
  16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
  32   33   34   35   36   37   38   39
```

For ciklus 2 - ASCII kódokok kiírása

```
for(i = 32; i < 154; i = i + 1){
    printf("%c ", i);
}
```

```
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G
H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ ⍰ Ç ü é â ä ů ć ç ł ë Ő ő î Ź Ä Ć É Ĺ Í ô ö Ľ Ľ Ś
ś Ö
```

ASCII karakterek listázása

```c
#include <stdio.h>
void main() {
  int i;
  // '0' = 48, '1' = 49, '2' = 50
  for (i = '0'; i <= 255; i = i + 1){
    printf("  %3d %c", i, i);
      if (i % 10 == 0)
        printf("\n");
  }
  getchar();
}
```

```
 48 0   49 1   50 2
 51 3   52 4   53 5   54 6   55 7   56 8   57 9   58 :   59 ;   60 <
 61 =   62 >   63 ?   64 @   65 A   66 B   67 C   68 D   69 E   70 F
 71 G   72 H   73 I   74 J   75 K   76 L   77 M   78 N   79 O   80 P
 81 Q   82 R   83 S   84 T   85 U   86 V   87 W   88 X   89 Y   90 Z
 91 [   92 \   93 ]   94 ^   95 _   96 `   97 a   98 b   99 c  100 d
101 e  102 f  103 g  104 h  105 i  106 j  107 k  108 l  109 m  110 n
111 o  112 p  113 q  114 r  115 s  116 t  117 u  118 v  119 w  120 x
121 y  122 z  123 {  124 |  125 }  126 ~  127 ⌂  128 Ç  129 ü  130 é
131 â  132 ä  133 ů  134 ć  135 ç  136 ł  137 ë  138 Ő  139 ő  140 î
141 Ź  142 Ä  143 Ć  144 É  145 Ĺ  146 Í  147 ô  148 ö  149 Ľ  150 Ĺ
151 Ś  152 ś  153 Ö  154 Ü  155 Ť  156 ť  157 Ł  158 ×  159 č  160 á
161 í  162 ó  163 ú  164 Ą  165 ą  166 Ž  167 ž  168 Ę  169 ę  170 ¬
171 ź  172 Č  173 ş  174 «  175 »  176 ░  177 ▒  178 ▓  179 │  180 ┤
181 Á  182 Â  183 Ě  184 Ş  185 ╣  186 ║  187 ╗  188 ╝  189 Ż  190 ż
191 ┐  192 └  193 ┴  194 ┬  195 ├  196 ─  197 ┼  198 Ă  199 ă  200 ╚
201 ╔  202 ╩  203 ╦  204 ╠  205 ═  206 ╬  207 ¤  208 đ  209 Đ  210 Ď
211 Ë  212 ď  213 Ň  214 Í  215 Î  216 ě  217 ┘  218 ┌  219 █  220 ▄
221 Ţ  222 Ů  223 ▀  224 Ó  225 ß  226 Ô  227 Ń  228 ń  229 ň  230 Š
231 š  232 Ŕ  233 Ú  234 ŕ  235 Ű  236 ý  237 Ý  238 ţ  239 ´  240
241 ˝  242 ̨  243 ˇ  244 ˘  245 §  246 ÷  247 ¸  248 °  249 ¨  250 ·
251 ű  252 Ř  253 ř  254 ■  255
```

Getchar

A standard könyvtárban számos olyan függvény van, amelyekkel egy időben egy karakter olvasható, a legegyszerűbb közülük a **getchar** függvény. Minden egyes hívásakor a **getchar** függvény a szövegáramból beolvassa a következő karaktert és annak értékét adja vissza a hívó függvénynek. Ennek megfelelően a

```
ch = getchar();
```

végrehajtása után a **ch** változó a bemenő szöveg következő karakterét fogja tartalmazni. A karakterek általában a terminálról (billentyűzetről) érkeznek.

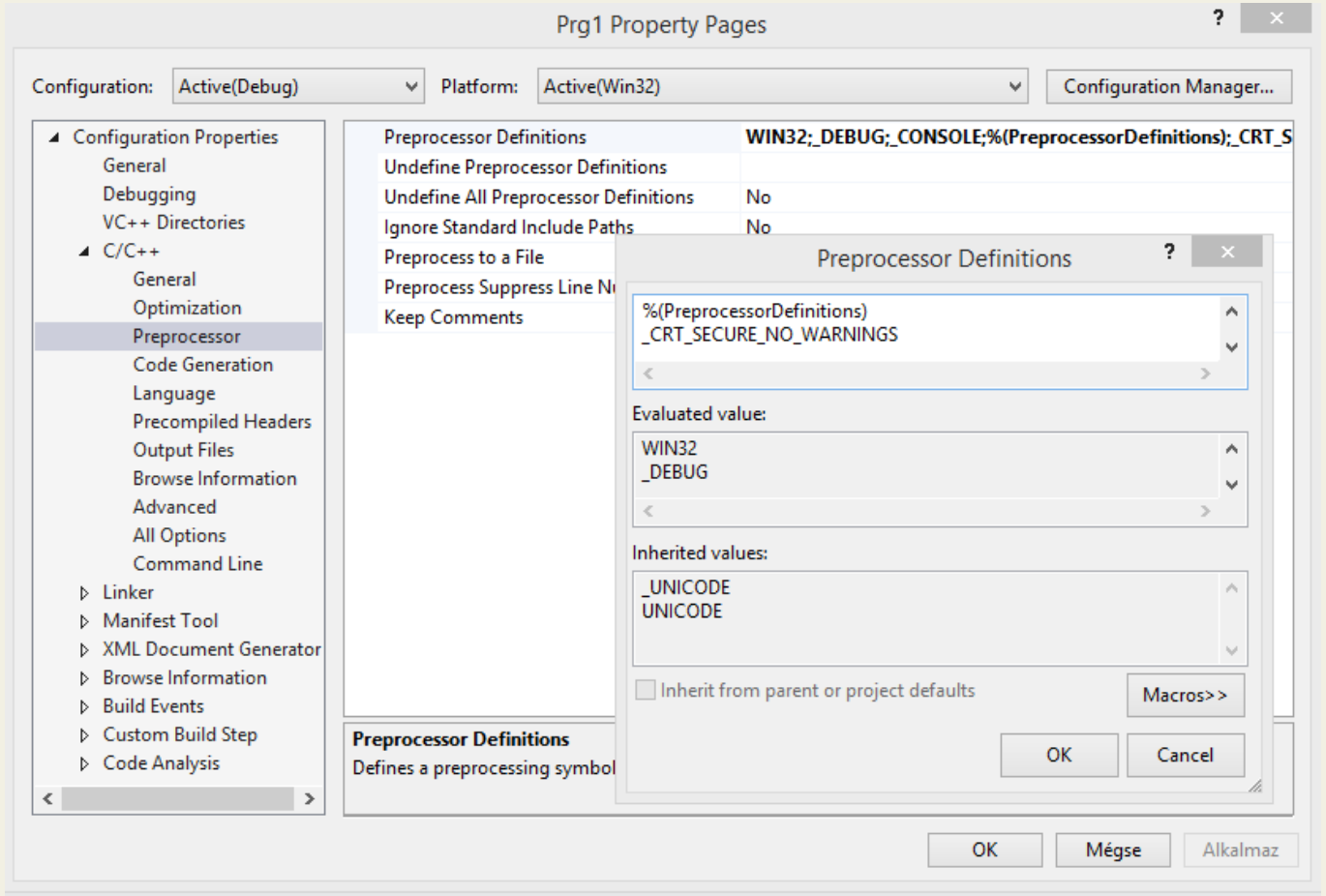
Getch

Hasonlít a **`getchar`** működésére, de **echó nélkül** olvas be egyetlen karaktert a billentyűzetről, és ezt szolgáltatja a hívónak. A bejövő karakter rögtön rendelkezésre áll, és nincs pufferezés soremelés karakterig. Funkció vagy nyíl billentyű leütésekor a függvényt kétszer kell hívni, mert az első hívás **`0`**-val vagy **`224`**-el tér vissza, és a második szolgáltatja az aktuális billenytű kódját. A rutinnak nincs hibás visszatérése. Bár **nem szabványos**, de szinte minden operációs rendszerben rendelkezésre állnak kisebb-nagyobb eltérésekkel a `<conio.h>` fejfájl bekapcsolása után.

getch

Project » Properties » Configuration properties » C/C++ » Prepocessor

```
_CRT_SECURE_NO_WARNINGS
_CRT_NONSTDC_NO_WARNINGS
```



fgets

- Egy teljes sornyi szöveg beolvasására.

- ```
  char str[201];
  fgets(str, 200, stdin);
  printf("{%s}", str);
  ```
  ```

scanf

- Mutatókat használ!
- Nincs inputellenőrzés!
- ```
  char  str[201] = "";
  int   dec;
  float flo;
  scanf("%d", &dec);
  scanf("%f", &flo);
  scanf("%s",  str);
  printf("%d %.2f %s", dec, flo, str);
  ```
  ```

ASCII kódokok kiírása - while ciklus és `getch()`

```c
#include <stdio.h>
#include <conio.h>
void main(){
  int ch;
  printf("ESC billenytuig
          karakterkodok kiiratasa\n\n");
  while ((ch = getch()) != 27) // 27 = ESC
  {
    printf("%d", ch);
    if (ch == 0 || ch == 224)
      printf(", %d", getch());
    printf("\n");
  }
  printf("ESC %d\n", ch);
}
```

A következő kódrészlet **ESC** billentyű lenyomásáig írja ki a billentyűzet karaktereiknek kódjait. Bizonyos speciális billentyűk és kombinációik (pl.: **F1**, **F2**, nyilak, **CTR+ALT+Z**) lenyomásakor két érték kerül a billentyűzetpufferbe, ezek a billentyűk kettős kóddal rendelkeznek (pl.: **0,77 224,77**)

_CRT_SECURE_NO_WARNINGS

_CRT_NONSTDC_NO_WARNINGS

Slash - Backslash

http://en.wikipedia.org/wiki/Slash_(punctuation)

Slash - /

`C:\folder\backslash.txt` vagy a `'\n'`

Backslash - \

Függvények

- A függvény a program alapvető része, bizonyos problémákat oldhatunk meg vele, lehet neki paramétert átadni és szolgáltathat visszatérési értéket.
- Miért jó?
  - » Strukturált kód
  - » Duplikáció elkerülése
  - » Újrahasználható kód
  - » Stb.

Függvények

```
típus   függvénynév   (formális-paraméterlista)  {függvény-test}
```

```
void    main          (void)                     {printf("Hello");}
```

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

```
int     getline       (char s[],int lim)
```

```
{
int c,i;
for(i=0;i<lim && (c=getchar())
!='\n' && c!=EOF;i++) s[i]=c;
s[i]=0;
while(c!='\n' && c!=EOF)
c=getchar();
return i;
}
```

Függvény - Kerület

Visszatérési érték

Formális-paraméterek

Függvény-deklaráció

```
double teglalapKerulet(double a, double b){
    return 2*(a+b);
}
```

Függvény-hívás

```
printf("%f", teglalapKerulet(4, 5.2));
```

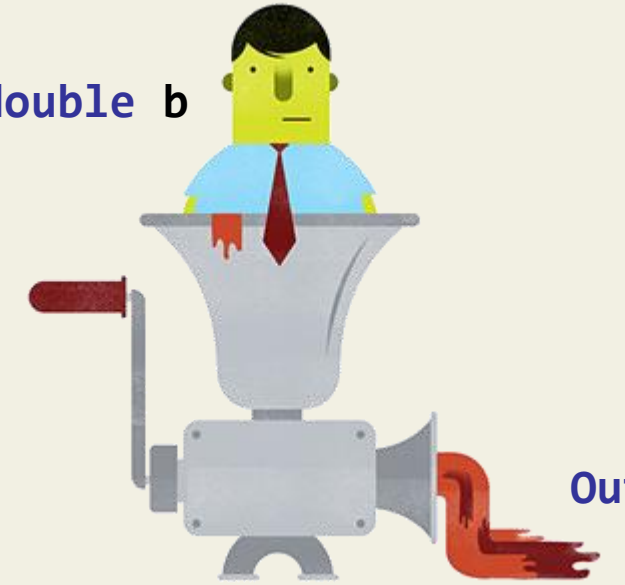
Aktuális-paraméterek

a = 4.0
b = 5.2

Függvény - Kerület

```
double teglalapKerulet(double a, double b){
     return 2*(a+b);
}
```

Intput: double a, double b



Output: double

Függvény - PrimE

```
int primE(int x){
  int osz = 2;
  if (x < 4)
    return 1;
  while (osz*osz <= x){
    if (!(x%osz))
      return 0;
    ++osz;
    if (!(osz & 1))
      ++osz;
  }
  return 1;
}
```

```
printf("%d - %d", primE(88), primE(103));
```

Természetesen van hatékonyabb algoritmus prímkeresésre.

Függvények gyakorlás

- Írjon `int max(int sz1, int sz2)` prototípusú függvényt, amely a két szám értéke közül, a nagyobbikat adja visszatérési értékként.

Függvények gyakorlás

- Írjon `int elojel(float sz1, float sz2)` prototípusú függvényt, amelyik visszatérési értékeként megadja két számról, hogy egyezik-e az előjelük!

getline / mygetline

```
// Karakterek beolvasasa EOF-ig vagy Enterig a getline fgv deklarálása
int getline(char s[],int lim){
   int c,i;

   for(i = 0; i < lim && (c = getchar()) != '\n' && c!=EOF; i++)
      s[i] = c;

   s[i] = '\0'; // tömb lezárasa

   while(c != '\n' && c!=EOF)
      c=getchar(); // puffer ürítése
   return i; // visszateresi értek: string hossza
}
```

Program Getline-nal

```
void main(){
  char string[51]="";
  if(getline(string, 50) != 0)
   printf("Stringünk: [%s] \n", string);
  else
   printf("A stringünk hossza 0");
  getch();
}
```

Függvények gyakorlás - négyzetes

- Írjon olyan függvényt, amely kiírja az első n természetes számot és azok négyzetét! A függvény prototípusa legyen: `void negyzKiir(int n)`.
- Például a `negyzKiir(8)` hívásra a jobb oldalon látható eredményt írja ki.

```
1^2= 1
2^2= 4
3^2= 9
4^2= 16
5^2= 25
6^2= 36
7^2= 49
8^2= 64
```

Feladat négyzetes

```
#include <stdio.h>
void negyzKiir(int n){
      int szam;
      for (szam = 1; szam <= n; szam = szam + 1)
            printf("%d^2= %d\n", szam, szam*szam);
}

void main(){
      negyzKiir(8);
      getchar();
}
```

Függvények gyakorlás

- Készítsen egy
  `float atlag(float a, float b, float c)`
  prototípusú függvényt, amely 3 szám átlagát adja visszatérési értékként.
- A feladat nehezítése, hogy csak a pozitív értékekből számol átlagot, a negatív értékeket figyelmen kívül hagyja.

Tippek, trükkök - Visual C++

Ctrl+U - kisbetűssé konvertálja a forráskód kijelölt részeit

Ctrl+Shift+U - nagybetűssé

Ctrl+K, Ctrf+F - formázza a forráskód kijelölt részeit

Ctrl+K, Ctrf+C - kommentez

Ctrl+K, Ctrf+U - törli a kommentezést

Random - Véletlenszám

A véletlenszám-generátor használata: program elején inicializálni kell (**`srand`**) egyszer, és utána a **`rand()`** ad egy véletlen számot. A **`%100`** hatására **`0..99`** között lesz, ehhez **`1`**-et adva kapjuk az **`1..100`** tartományt. A működéshez a `stdio.h`, `time.h`, `stdlib.h` headerek szükségesek.

```
srand(time(NULL));
r = rand()%100 + 1;
```

Random - Véletlenszám

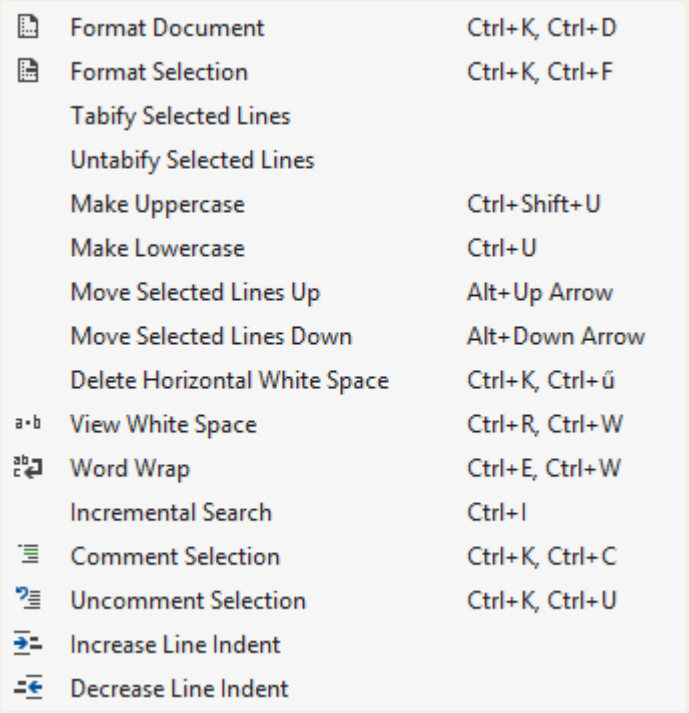
```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define MAX 20

void main(){
	int r = 0, i;
	srand(time(NULL));
	for(i=0; i<20; i++){
		r = (rand() % MAX);
		printf("%d\n", r);
	}
	getchar();
}
```

Formázás

```c
#define ALSO 0
#define FELSO 3000
#define LEPES 20
#define LAPSOR 20
#include <stdio.h>
void main(){
	int fahr, also, felso;
	char c;
	fahr = ALSO;
	c = 'e';
	printf("\nFahrenheit - Celsius átszámítás\n\n");
	while (c != 'v'){
		if (c == 'e'){
			also = fahr;
			felso = also + LEPES*LAPSOR;
			if (felso > FELSO){
				felso = FELSO;
				also = FELSO - LEPES*LAPSOR;
			}
		}
		else if (c == 'h'){
			felso = also - LEPES;
			also = felso - LEPES*LAPSOR;
			if (also < ALSO){
				also = ALSO;
				felso = ALSO + LEPES*LAPSOR;
			}
		}
		printf("Fahrenheit - Celsius\n");
		printf("--------------------\n");
		for (fahr = also; fahr <= felso; fahr = fahr + LEPES)
			printf("%10d%10.1f\n", fahr, (5. / 9.)*(fahr - 32));
		printf("\nv - vege, e - elore, h - hatra: ");
		while ((c = getchar()) != 'e' && c != 'h' && c != 'v');
	}
}
```

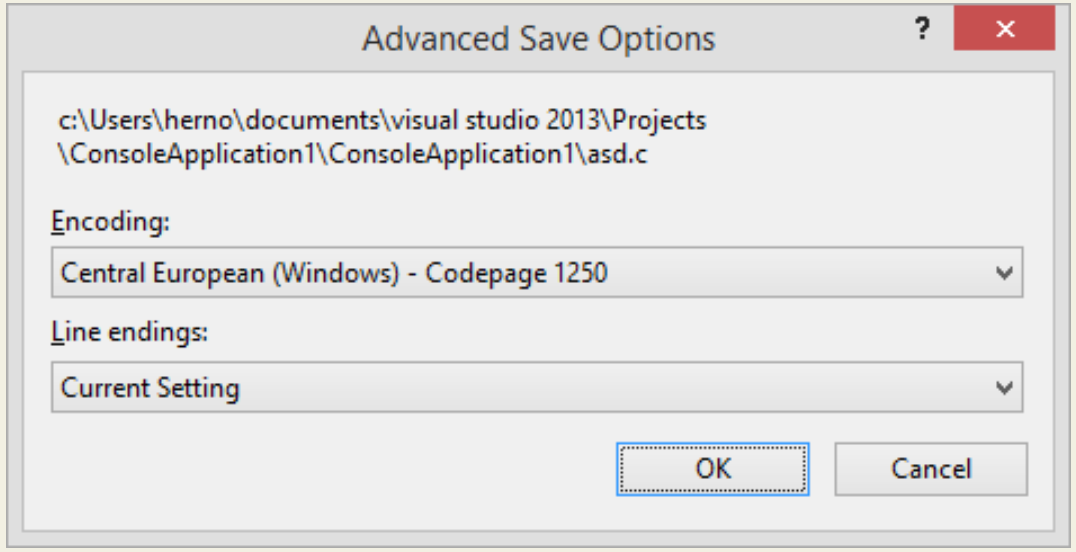
Edit » Advanced » Format Document
» Format Selection

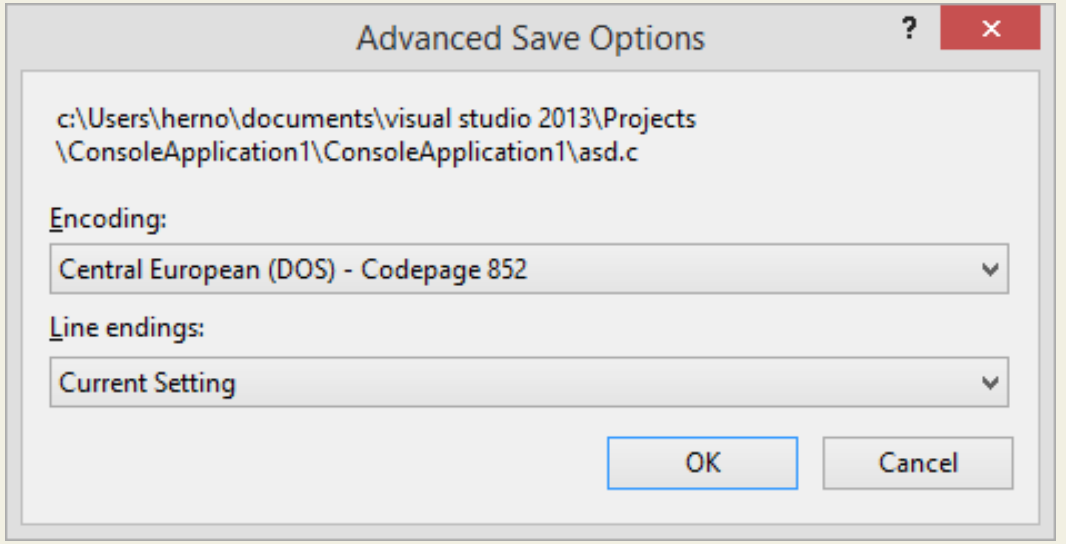
Ctr+K Ctr+D
Ctr+k Ctr+F

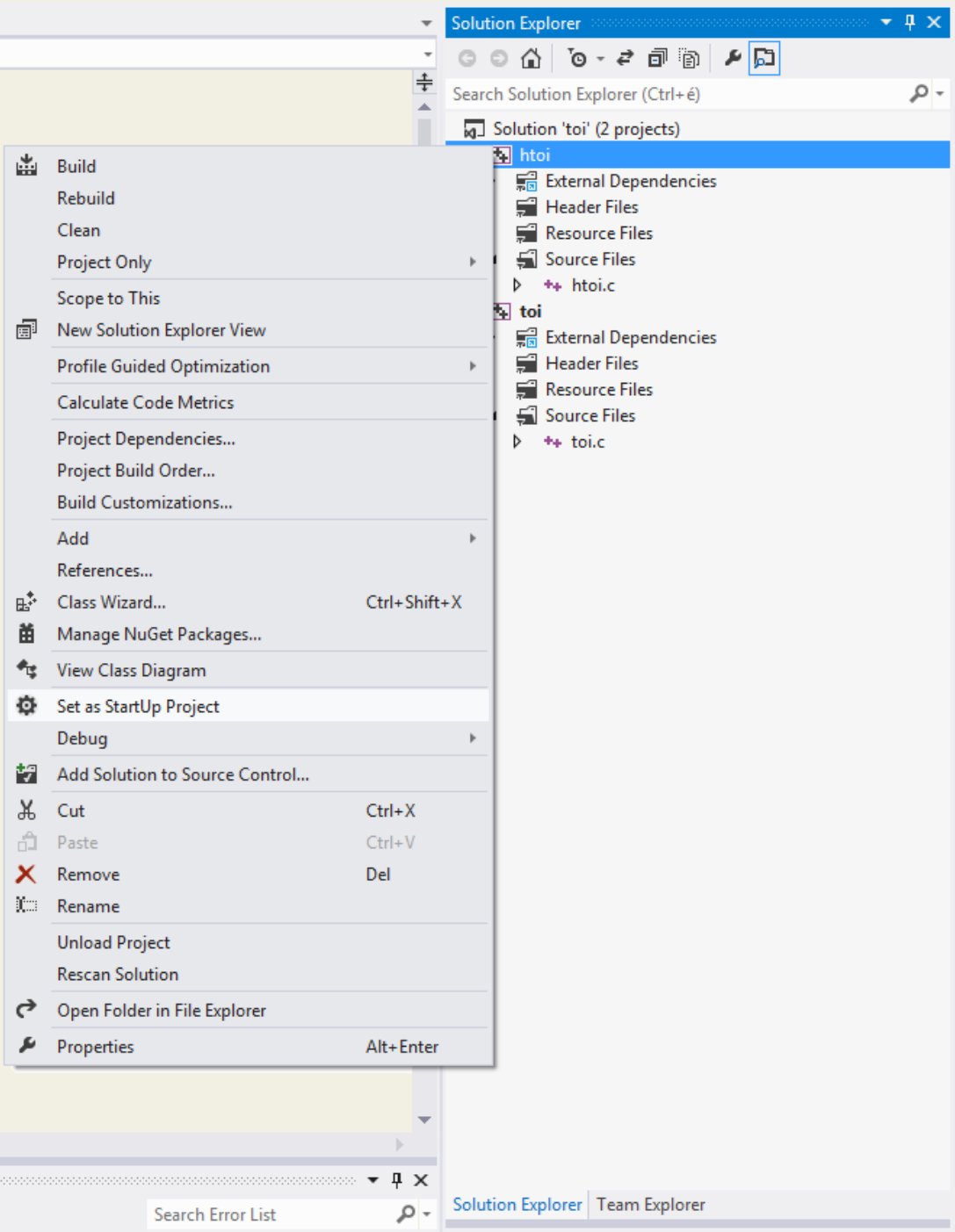


Codepage

- CP1250(Win) vs CP852(DOS)







Projektek

- Több Projekt egy Solution
- Set as StartUp Project

Összehasonlítás vs. Értékadás

Összehasonlítás:

```
if(i == 1)
  printf("i egyenlő 1el");
```

Értéke: TRUE v. FALSE

Értékadás:

```
i = 1;    //i értéke egy lett
```

Összehasonlítás vs. Értékadás

Operátorok

```
a == b
a != b
a > b
a < b
a >= b
a <= b
```

Operátorok

```
a = b
a++
```

Operátorok

```
a + b
a - b
a * b
a / b
a % b    Modulo (maradék)
a++
a--
a && b   a és b
a || b   a vagy b
!a       nem a
```

++i vs i++

```
i = 1;
j = ++i;
```

(**i** értéke 2, **j** értéke 2)

i értéke a növelés után

- prefix operátor

```
i = 1;
j = i++;
```

(**i** értéke 2, **j** értéke 1)
Hiszen még akkor kapott értéket, mikor i értéke egy volt.
i értéke a növelés előtt

- postfix operátor

Fizzbuzz

Írjon C programot, amely elszámol 1-től 100-ig, minden hárommal osztható szám helyett Fizz, 5-tel osztható szám helyett Buzz, de minden hárommal és öttel osztható szám helyett Fizzbuzz szöveget ír ki.

Fizzbuzz

Írjon C programot, amely elszámol 1-től 100-ig, minden hárommal osztható szám helyett Fizz, 5-tel osztható szám helyett Buzz, de minden hárommal és öttel osztható szám helyett Fizzbuzz szöveget ír ki.

```c
#include <stdio.h>
int main(){
	int i = 1;
	for (i; i <= 100; i++){
		if (i % 15 == 0)
			printf("FizzBuzz");
		else if (i % 3 == 0)
			printf("Fizz");
		else if (i % 5 == 0)
			printf("Buzz");
		else
			printf("%d", i);
		printf("\n");
	}
	getchar();
}
```

e: a természetes logaritmus alapszáma

- Az e=2,7182818… matematikai konstans előállítható az alábbi képlettel:

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} \ldots\ldots$$

- Írjon programot, amely kiszámolja ezt az első 20 taggal! A faktoriális nagy szám lehet. Tárolja **`double`** típusú változóban.
  - » Figyeljünk rá, hogy **`0!=1`** és is **`1!=1`**. (**`5!=120`**)

e: a természetes logaritmus alapszáma

```c
#include <stdio.h>

void main() {
  double e;    // e tárolása
  double fact; // Faktoriális tárolása
  int i;       // Ciklusváltozók

  e = 0;
  fact = 1;
  for (i = 1; i < 20; i = i + 1) {
    e = e + 1 / fact;  // Hozzáadja e-hez
    fact = fact*i;
  }

  printf("e = %f", e);  // Kiírja

  getchar();
}
```

Elég egy ciklus is!

Előfordító

```
#include <stdio.h>
#include <conio.h>
#define MINHOSSZ 10
#define MAXHOSSZ 50
```

A C előfordító a C programozási nyelv tényleges fordítása előtt végrehajtott speciális program. Az előfordító felel más forrásfájlok kezeléséért, felhasználható szimbólumok és makrók definiálására illetve a feltételes fordítás lehetővé tételéért.

Feladat 06_1.C Betűk leszámlálása

Készítsen programot, mely a szabvány bemenetet olvassa EOF-ig.
Megállapítandó és kijelzendő, hogy hány A, B, C, stb. karakter érkezett.
A kis- és nagybetűk között nem teszünk különbséget! A betűkön kívüli többi karaktert tekintsük egy kategóriának, s ezek darab számát is jelezzük ki!

Tömbök

Deklarálása

- `típus név  [hossz];`
- `int   tomb[5] = {0,0,0,0,0};`

Értékadás

```
tomb[1] = 100;
```

Kiíratás

```
for (i = 0; i < 5; i++)
  printf("%d ", tomb[i]);
```

```
0 100 0 0 0
```

Tömbök

hossz	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	
7	'A'	'l'	'm'	'a'	'\0'	#	ä		Karakteres
6	3.14	4.65	43.55555	0	3232.22	212.22			Lebegőpontos
5	0	100	0	0	0				Egész

```
char KarakterTomb1 [7] = {'A', 'l', 'm', 'a', '\0'};
char KarakterTomb2 [7] = "Alma";
i=0;
while (KarakterTomb1[i] != '\0')
   {
      printf("%c ", KarakterTomb1[i]);
      i++;
   }
printf("\n %s", KarakterTomb2);
```

Karaktertömbök

Hibás:

- `KarakterTomb1 = "Hello!";`

Helyes: (string.h)

- `strcpy(KarakterTomb1, "Hello!");`

Nincsenek string operátorok, de vannak string kezelő függvények.

Karaktertömb bejárása

- **0.** elemtől a végéig
  ```
  for(i=0; karTomb[i]!='\0'; i++)
  ```
- A végétől a **0.** elemig
  ```
  for(i=strlen(karTomb)-1; i>=0; i--)
  ```

				strlen(k)-1	strlen(k)
[0]	[1]	[2]	[3]	[4]	[5]
'A'	'l'	'm'	'a'	'\0'	

EOF

- End Of File
- Negatív érték (-1) rendszerfüggő
- Ctrl+D – Linux
- Ctrl+Z+Enter - Windows

Betűk számolása

```
while ((k = getchar()) != '\n')
 if (k >= 'a' && k <= 'z')
  betu[k -'a']++;
 else if (k >= 'A' && k<= 'Z')
  betu[k - 'A']++;
 else
  egyeb++;
```

Feladat 06_2.C: Név keresése

- Készítsen programot, mely neveket olvas a szabvány bemenetről EOF-ig vagy üres sorig! Megállapítandó egy fordítási időben megadott névről (pl. `"Jani"`), hogy hányszor fordult elő a bemeneten! A feladat megoldásához készítendő egy int `strcmp(char s1[], char s2[])` függvény, mely két karakterlánc összehasonlítását végzi, és egyezés esetén nullát szolgáltat visszaadott értékül!
- E függvényt később fejlesszük úgy tovább, hogy működése egyezzék a szabvány könyvtári változatéval!

Strcmp

```
int strhasonlit(char s1[], char s2[]){
   int i = 0;
   while(s1[i] != 0 && s1[i] == s2[i])
     i++;
   return s1[i] - s2[i];
}
```

Nevek megszámolása

```
while(getline(s,MAX)!=0)
 if(strhasonlit(s, NEV) == 0)
  db++;

printf("%d darab", db);
```

string.h

Függvény	Leírás
strcmp(s1, s2)	Két string összehasonlítására használható, eredménye 0, ha a két string azonos.
strcat(s1, s2)	Két string összefűzésére alkalmas.
strcpy(s1, s2)	Az első stringbe másolja a második tartalmát
strlen(s)	A string hosszát adja vissza
strrev(s)	Megfordítja a stringet
strchr(s1,k)	Megnézi, hogy az s1 stringben először hol fordul elő a k karakter

Részletesen: http://en.wikipedia.org/wiki/C_standard_library

strcmp

```
if(strcmp(karTomb1, karTomb2) == 0)
     printf("Ugyanaz mind2 string.");
else
     printf("%s != %s", karTomb1, karTomb2);
```

Stringek összevetése, nullát ad vissza, ha egyeznek.

strcat

```c
char str1[50] = "Hello ";
char str2[50] = "World!";

strcat(str1, str2);
printf("%s", str1);
```

Stringek összefűzése.

strcpy

```
char karTomb [50] = "teszt";

strcpy(karTomb, "Alma");
printf("%s", karTomb);
```

String értékadás, illetve másolás.

strlen

```
for(i=0; i < strlen(karTomb); i++)
  printf(" %c \n", karTomb[i]);
```

Az **strlen** a karaktertömb hosszát adja vissza.

Feladat 07_2.c Karakterlánc megfordítása

- Készítsen egy **`void reverse(char s[])`** függvényt, mely a saját helyén megfordítja a paraméter karakterláncot!

Feladat 07_2.c Karakterlánc megfordítása

- A feladat azt kérte, hogy a függvény „saját helyén megfordítsa meg a paraméter karakterláncot”.
- Eddig úgy tudtuk a változók *érték szerint* (pass by value) adódnak át a függvényeknek. Ez azt jelenti, hogy nem maga a változó, hanem annak értéke adódik át a függvénynek. A tömbök (és a mutatók) kivételt képeznek ez alól, hiszen *referencia szerint* (pass by reference), tehát itt a változó memóriacíme adódik át a függvények. Ennek a C nyelvben történelmi okai vannak és erről bővebben tanulunk a mutatók témakörnél.

reverse

```
void reverse(char s[]){
   int i, j;
   char kar;
   for(i = 0, j = strlen(s)-1; i<j; ++i, --j){
     kar  = s[i];
     s[i] = s[j];
     s[j] = kar;
   }
}
```

reverse

	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
	a	b	c	d	x	6	7	8	9	'\0'
[i=0]	9	b	c	d	x	6	7	8	a	'\0'
[i=1]	9	8	c	d	x	6	7	b	a	'\0'
[i=2]	9	8	7	d	x	6	c	b	a	'\0'
[i=3]	9	8	7	6	x	d	c	b	a	'\0'

Debug

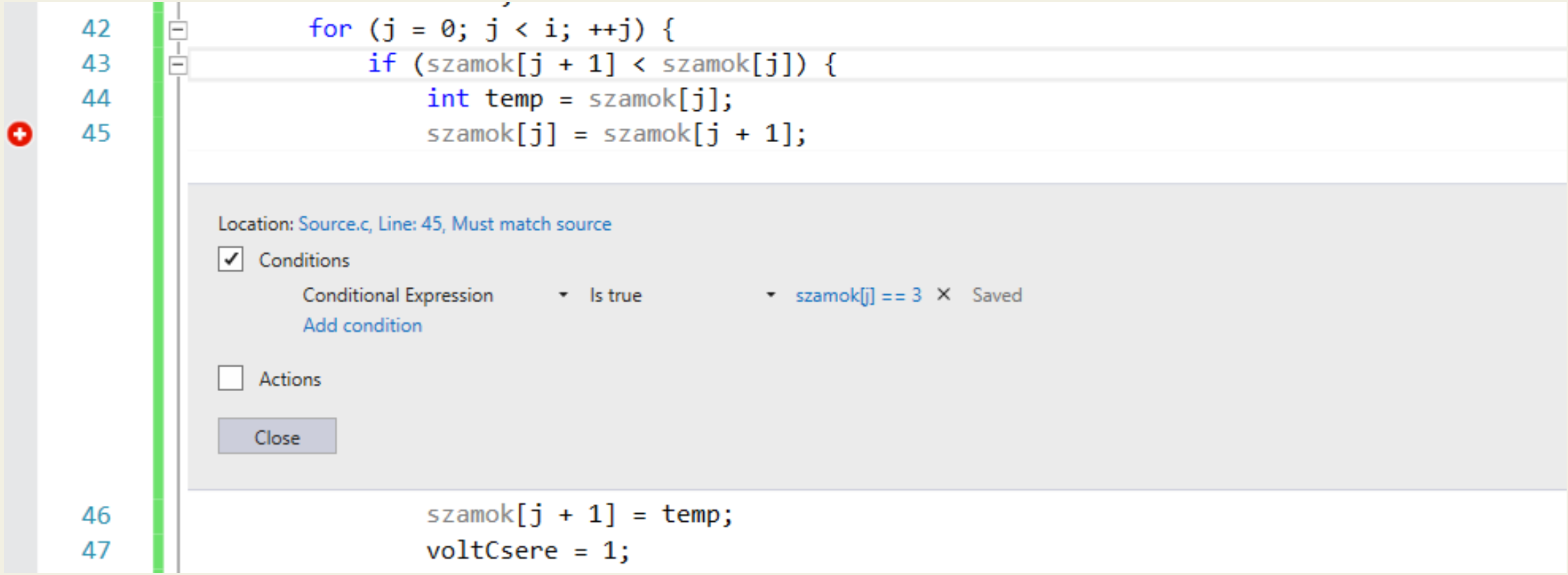
- A fejlesztett program vagy alkalmazás logikai hibáinak kiküszöbölése.
- Breakpoint
- Step Into (F11)
- Step Over (F10)

"Debugging is like being the detective in a crime movie where you are also the murderer."

- Filipe Fortes

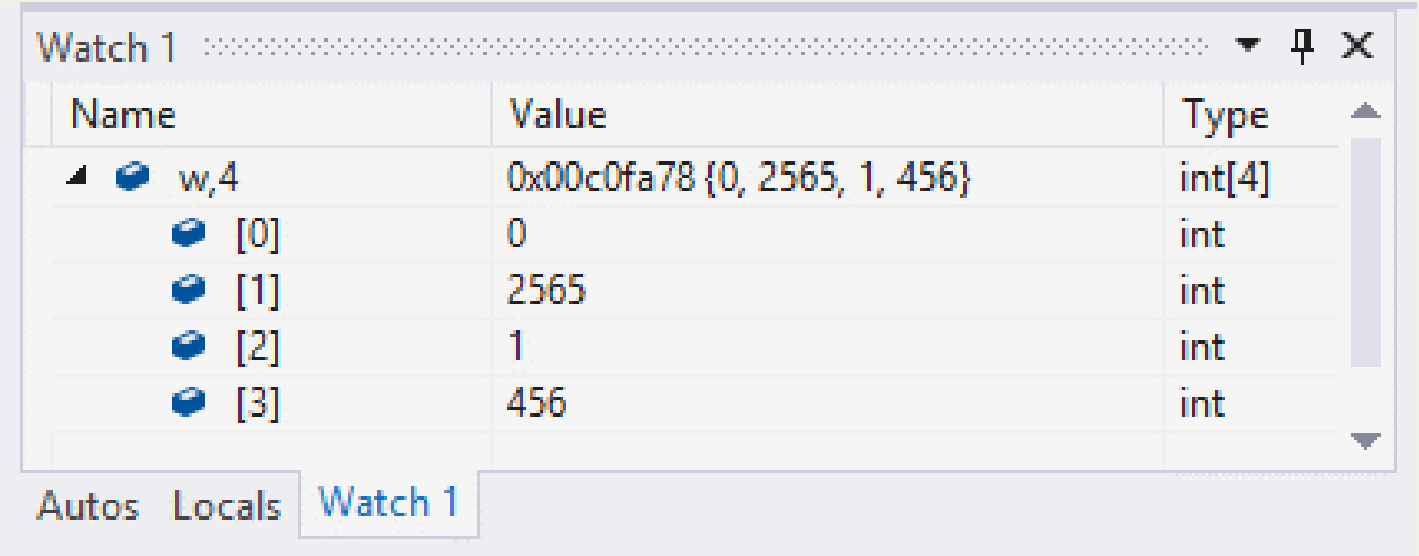
Debug

Breakpoint feltétellel



Debug

- Tömbök értékeinek figyelésesnél (Watch) csak az első értéket írja ki alapesetben. A **`tombneve, tömbhossza`** kifejezést megadva a teljes tömbhosszra kiírja az értékeket.
- Például, ha a tömb neve **`w`**, hossza **`4`**, akkor **`w, 4`** kifejezéssel lehet kijeleztetni.



Feladat Egyszerű cézár kódolás

- Készítsen egy `void cezar(char mit[])` függvényt, mely egyszerű cézár kódolással látja el a karakterláncot! Az `"A"` karaktert eggyel növeli, így `"B"`-t kapunk a `"B"`-ből `"C"` lesz stb.
- Fejlessze tovább a feladatot, úgy, hogy `"Z"` után újra `"A"` következzen és az eltolás mértéke is változtatható legyen: `void cezar(char mit[], int mennyivel).`

Eredeti	**Alma**	**Almabor**	**ABCDE**	**Jani**
Cézár kódolt	**Bmnb**	**Bmnbcps**	**BCDEF**	**Kboj**

```
Adjon stringet, amit cezar-kodoloni kell:

Alma almafa almabor 123 ABCdef!

Eredeti:   [Alma almafa almabor 123 ABCdefXYZ!]
3 eltolas: [Dopd dopdid dopderu 123 DEFghiBCD!]
```

Feladat Névkiíratás1

Deklaráljon egy tömböt a saját nevének, majd írja ki a páratlan sorszámú (indexű) betűket! Ha a ciklusváltozó kettesével halad, nagyon figyelni kell arra, hogy a legnagyobb tömbindexet ne lépje túl!

Sorszám	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Karakter	'K'	'i'	's'	' '	'T'	'a'	's'	'\0'

Feladat Névkiíratás1

```
strcpy(karTomb, "Horvath Erno");
for(i=0; karTomb[i]!='\0'; i++)
    if(i%2 == 0)
        printf("%c ", karTomb[i]);
```

Feladat Feltölt

Készítsen egy olyan,
`void feltolt(char s[], int db, char mivel)` prototípusú függvényt, amely az **s** karakterlánc első **db** indexű elemét kicseréli a **mivel** karakterre.

				strlen(s)-1	strlen(s)
[0]	[1]	[2]	[3]	[4]	[5]
'A'	'l'	'm'	'a'	'\0'	

Feladat Stringek

- Készítsem programot, mely üres sorig a beolvasott stringeket
  - » Nagybetűssé konvertálva
  - » Egy értékkel eltolva (a->b, b->c, c->d stb)

  a beírt érték alá írja ki.

Feladat Számkitaláló

Készítsen egy számkitaláló programot! A program kitalál véletlenszerűen egy pozitív egész számot (1 és 1000 között), a felhasználó pedig addig tippel, amíg meg nem találja a keresett számot. A program minden tipp után megmondja, hogy a felhasználó tippje kisebb vagy nagyobb a keresett értéknél. Ha eltalálta, akkor pedig azt. Ilyenkor egyúttal be is fejeződik a program futása.

- Segítség: **`getline`**, **`toi`**, **`szame`**, **`do while`**

Feladat 07_1.C Szorzatpiramis

Készítsen szorzatpiramist! A piramis első sora álljon 100 és 115 közötti egész számokból! A következő sorokat úgy kapjuk, hogy a számokat páronként összeszorozzuk.

Tehát a második sor számai: 100*101=10100, 102*103=10506, ..., 114*115=13110

A harmadik sor pedig a következő: 10100*10506=106110600, 10920*11342=126854640, ..., 12656*13110=165920160

Figyeljünk arra, hogy az eredményül kapott számokat mindig a lehető legkisebb helyen tároljuk, de a számoknak és a közbenső eredményeknek nem szabad csonkulniuk! (ANSI C adattípusok)

Feladat 07_1.C Szorzatpiramis

Szorzatpiramis: első szint

```
( 100)*( 101) =  10100
( 102)*( 103) =  10506
( 104)*( 105) =  10920
( 106)*( 107) =  11342
( 108)*( 109) =  11772
( 110)*( 111) =  12210
( 112)*( 113) =  12656
( 114)*( 115) =  13110
```

Szorzatpiramis: második szint

```
( 10100)*( 10506) =  106110600
( 10920)*( 11342) =  123854640
( 11772)*( 12210) =  143736120
( 12656)*( 13110) =  165920160
```

Szorzatpiramis: harmadik szint

```
( 106110600)*( 123854640) =  13142290163184000
( 143736120)*( 165920160) =  2384872002817920 0
```

Szorzatpiramis: csúcs

```
( 13142290163184000)*(2384872002817920 0) ~ 31342679863086874000000000000000000
```

Feladat 09_1.C Toi számkonverzó

- Készítsen int **`toi(char s[])`** függvényt, mely tizenegynél kisebb alapú számrendszerbeli számjegyekből álló karakterláncból előállít egy egész számot!
- A számrendszer alapja legyen fordítási időben változtatható!

Algoritmus

- Billentyűzetről karakterlánc beolvasása -**`getline`**
- Karakterlánc ellenőrzése – megnézzük, számot adott-e be a felhasználó – karakterek ‘0’ és ‘9’ közé esnek (csak 10es számrendszer)
- Karakterlánc átalakítása számmá

Karaktertömb számmá

Háromezer-hatszáznegyven

	[0]	[1]	[2]	[3]	[4]	[5]
Karakter	'3'	'6'	'4'	'0'	'\0'	
Szám	51	54	52	48	0	

↓ ↓ ↓ ↓

3*1000 6*100 4*10 0*1

51-48=3 54-48=6 52-48=4 48-48=0

toi

```
int toi(char s[]){

  int i, eredmeny = 0;
  for(i=0; s[i]!=0; ++i)
     eredmeny = eredmeny * ALAP + s[i] - '0';
  return eredmeny;

}
```

toi

- `printf("%d", '0'); //48`

- ASCII
  - `48 = '0'`
  - `49 = '1'`
  - `50 = '2'`

  - `...`
  - `65 = 'A'`

toi

```
for(i=0; s[i]!=0; ++i)
      eredmeny = eredmeny * ALAP + s[i] - '0';
```

"1264"

[0] >> 0 * 10 + 1 = 1

[1] >> 1 * 10 + 2 = 12

[2] >> 12* 10 + 6 = 126

[3] >> 126*10 + 4 = 1264

szám-e » egész szám ellenőrzése

```
//"Az s string ALAP alapú szám-e?" kérdésre ad logikai választ a függvény
int szame(char s[]){
  int i;
  // Ha 10-es szamrendszert kellene vizsgálni isdigit()
  for(i=strlen(s)-1; i && s[i]>='0' && s[i]<ALAP+'0'; --i); //!!!
  if(s[i]>='0' && s[i]<ALAP+'0')
      return 1;
  else
      return 0;
}
```

			strlen(s)-1	strlen(s)	
[0]	[1]	[2]	[3]	[4]	[5]
'3'	'6'	'4'	'0'	'\0'	
'3'	**'B'**	'4'	'0'	'\0'	

stlib.h

Függvény	Leírása
atoi	Stringet konvertál integerré, a fehér karaktereket figyelmen kívül hagyva. –32768 és 32767 között
atol	Stringet konvertál long int típussá, a fehér karaktereket figyelmen kívül hagyva. –2,147,483,648 és +2,147,483,647 között
atof	Stringet konvertál lebegőpontos double-é, a fehér karaktereket figyelmen kívül hagyva. $\pm 1.7*10^{-308}$ 15–16 decimális jegy pontosság

Részletesen: http://en.wikipedia.org/wiki/C_standard_library

atoi

```
// string --> decimalis
int atoi(char st[]){
  int decimal = 0;
  int i = 0;
  char neg=0;
  while(st[i]==' ' || st[i]=='\t' || st[i]=='\n') ++i;
  if(st[i]=='-') ++neg;
  if(st[i]=='-' || st[i]=='+') ++i;
  for(;st[i];++i) decimal = decimal*10+st[i]-'0';
  if(neg) decimal = -decimal;
  return decimal;
}
```

atoi

```
char string[]="-2147";
printf("Ez egy sima string:  %s \n", string);
printf("Ez decimalis szam :  %d", atoi(string));
```

```
Ez egy sima string: -2147
Ez decimalis szam : -2147
```

atof

```c
#include <stdio.h>    //printf
#include <stdlib.h>   //atof
#include <math.h>     //sin

void main (){
     double n;
     char pi[20] = "3.1415926535";
     n = atof(pi);
     printf("Az n erteke:   %f \n" , n );
     printf("Az n negyzete: %f \n" , pow(n, 2) );
     getchar();
}
```

lebeg-e » lebegőpontos szám ellenőrzése

```
int lebege(char s[]){  // Az s karakterlánc lebegőpontos numerikus-e?
  int i=0, kezd;
  // A karakterlánc elején levő fehér karakterek átlépése
  while(isspace(s[i]))++i;
  // Az előjel elintézése
  if(s[i]=='+'||s[i]=='-')++i;
    kezd=i;
  // Az egész rész intézése
  while(isdigit(s[i]))++i;
  if(s[i]=='.') ++i;
  // A tört rész
  while(isdigit(s[i]))++i;
  if(i==kezd||kezd+1==i&&s[kezd]=='.') return 0;
  // Kitevő rész
  if(s[i]=='E'||s[i]=='e'){
    ++i;
    if(s[i]=='+'||s[i]=='-')++i;
    if(!isdigit(s[i]))return 0;
    while(isdigit(s[i]))++i;}
  if(s[i]) return(0);
  else return(1); }
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
' '	' '	'-'	'3'	'0'	0		
'1'	'.'	'8'	'9'	'6'	'3'	'7'	0

Személynév ellenőrzése

```
// Az string név-e? kérdésre ad logikai választ
int nevEllenorzes(char s[]){
  int i = 0;
  // a ciklus addig fut, míg betű vagy szóköz jön
  while ( isalpha(s[i]) || isspace(s[i]) )
    i++;
  // az utolsó karakter a lezáró 0,
  // és legalább 4 betűs tehát végig megfelelt
  if (s[i] == '\0' && i >= 4)
    return 1;
  // nem felelt meg a név formai követelményeinek
  else
    return 0;
}
```

A függvény csak annyit ellenőriz, hogy a név legalább 4 karakterből álljon, és csak szóközt, illetve betűt tartalmazzon.
Ennél természetesen részletesebb ellenőrzést is el lehet készíteni.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
'a'	'b'	0					
'#'	'2'	0					
'a'	'b'	'c'	'd'	'1'	0		
'a'	'b'	'c'	'd'	'e'	0		
'K'	'i'	's'	' '	'T'	'a'	's'	0
' '	' '	'a'	'a'	'a'	'a'	'a'	0

Rendszámtábla ellenőrzése

Írjon függvényt, amely egy string validációját végzi el. A formátum egyezzen meg a leggyakoribb magyar rendszámtábla formátumával. Eszerint 7 karakter hosszúságú legyen, ennek elején három (nagy vagy kis) betű után egy kötőjel és végül három szám következzen.
A függvény legyen

```
int rendszamEllAlap(char s[]).
```

Megoldás: `rendszamEllAlap.c`

Rendszámtábla ellenőrzése

```
int rendszamEllAlap(char s[]) {
  int i;
  for (i = 0; i < 3; i++) // betűk ellenőrzése
    if (!isalpha(s[i]))
      return 0;
  if(s[i] != '-') // kötőjel
    return 0;
  for (i = 4; i < 7; i++) // számok ellenőrzése
    if (!isdigit(s[i]))
      return 0;
  // az utolsó karakter a lezáró 0, helyes
  if (s[i] == '\0')
    return 1;
  // nem felelt meg a formai követelményeinek
  else
    return 0;
}
```

Rendszámtábla ellenőrzése

Fejlessze tovább az előző feladatban írt függvényt, úgy, hogy a 3 betű, kötőjel, 3 szám mellett a 3 betű, három szám és a 3 betű, space, 3 szám formátumot is fogadja el.

```
SZE-001
SZE 001
SZE001
```

A függvény legyen `int rendszamEllBov(char s[])`.

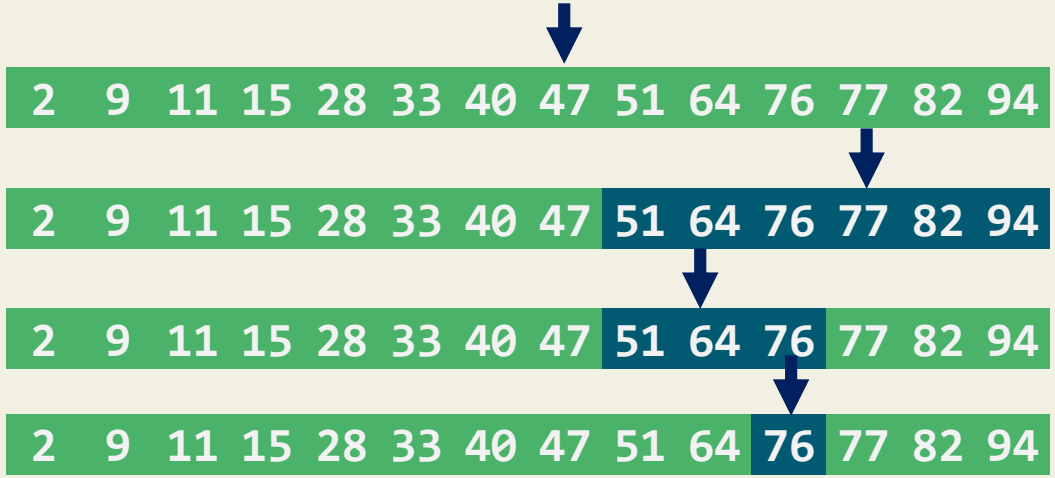
Lineáris keresés

Szeretnénk megállapítani, hogy egy adott (például **double** típusú) tömbben hol fordul elő egy adott szám. Ha nem tudunk semmit a tömbről, a **linKeres** függvény visszaadja az első előfordulás indexét, illetve **-1**-et, amennyiben nem találja.

```
#include <stdio.h>
#define MERET 6
int linKeres(double szamok[], int meret, double mit){
  int i;
  for (i = 0; i < meret; ++i)
  if (szamok[i] == mit)
    return i;
  return -1;
}
void main(){
  double sz[MERET] = {9.4, 2.1, 5.8, 9.9, 4.2, 3.6};
  printf("%d. hely", linKeres(sz, MERET, 5.8));
  getchar();
}
```

Ha rendezett a tömb » bináris keresés

```
int binKeres(double szamok[], int meret, double mit) {
  int min, max, kozep;
  min = 0;      // határok
  max = meret - 1;
  kozep = (min + max) / 2;
  while (min <= max && szamok[kozep] != mit) {
    if (szamok[kozep] < mit)
      min = kozep + 1;   // középtől jobbra
    else
      max = kozep - 1;   // középtől balra
    kozep = (min + max) / 2;
  }
  // miért állt meg a ciklus?
  return min <= max ? kozep : -1;
}
```



Feladat 10_5.C Tömb átlaga, rendezése

Készítsen programot, mely egész számokat kér be, meghatározza átlagukat, minimumukat, maximumukat és rendezi őket növekvő sorrendbe! Az egészek száma csak futás közben dől el. Csak formailag helyes egész számok fogadhatók el. A program elsőnek kérje be, hány egész szám lesz, s csak azután a számokat magukat!

Tömb átlaga, rendezése

- Módosítsa a feladatot úgy, hogy lebegőpontos számokat is elfogadjon és a programból `"kilep"` string érkezésekor mindig kilépjen.

Algoritmusok hatékonysága

Az O(n) "nagy ordó" azt írja le, hogy a műveletek (algoritmus) elvégzéshez szükséges idő hogyan függ a bemeneti adatok számától.

Hatékonyság	
$O(1)$	konstans (jó)
$O(\log n)$	logaritmikus
$O(n)$	lineáris
$O(n^2)$	négyzetes
$O(x^n)$	exponenciális (rossz)

Rendezések

- Buborék rendezés: egyszerű, nem túl hatékony (bubble sort)
- Gyorsrendezés: átlagos esetben gyors, a legrosszabb esetben lassú (quick sort)
- Shell rendezés
- Kupacrendezés
- Közvetlen kiválasztásos rendezés (selection sort)
- Közvetlen beszúró rendezés
- … és még sok másik rendezés

- http://en.wikipedia.org/wiki/Sorting_algorithm
- http://youtu.be/t8g-iYGHpEA (What different sorting algorithms sound like)
- http://algo-visualizer.jasonpark.me/#path=sorting/bubble/basic

Buborékrendezés

1) Hasonlítsuk össze az első két elemet. Ha nincsenek jó sorrendben, cseréljük meg.
2) Hasonlítsuk össze a második párt (második és harmadik elem). Esetleg csere.
3) Folytassuk így a tömb végéig.
4) A legnagyobb elem ezáltal a tömb végére kerül, még akkor is, ha legelöl volt. Az már a végleges helye.
5) Csináljuk meg ugyanezt még egyszer, a tömb elejétől az utolsó előttiig. Az utolsóhoz már nem kell nyúlni, hiszen az a legnagyobb.
6) Aztán ugyanezt megint, de az utolsó kettőhöz már nem nyúlunk stb.

Buborékrendezés

```
void buborekRendez(double szamok[], int meret) {
  int i, j;
  //egyre rövidebb tömbrészletek ciklusa
  for (i = meret-1; i > 0; --i)
    // egymás utáni párok ciklusa
    for (j = 0; j < i; ++j)
      if (szamok[j+1] < szamok[j]){
         double temp = szamok[j];
         szamok[j] = szamok[j+1];
         szamok[j+1] = temp;
      }
}
```

32	65	21	87	43
32	21	65	87	43
32	21	65	87	43
32	21	65	43	87
21	32	65	43	87
21	32	65	43	87
21	32	43	65	87
21	32	43	65	87
21	32	43	65	87
21	32	43	65	87

A buborékrendezés javítása

A buborékrendezés hatékonysága javítható azzal, ha megjegyezzük, hogy a vizsgált tömbrészletnél volt-e csere (**voltCsere** változó).

```
void buborekRendezJav(double szamok[], int meret) {
  int i, j, voltCsere=1;
  // egyre rövidebb tömbrészletek ciklusa
  for (i=meret-1; i>0 && voltCsere; --i){
    voltCsere=0;
    for (j=0; j<i; ++j){
      if (szamok[j+1] < szamok[j]){
        double temp=szamok[j];
        szamok[j]=szamok[j+1];
        szamok[j+1]=temp;
        voltCsere=1;
      }
    }
  }
}
```

32	65	21	87	43
21	65	32	43	87
21	32	43	65	87
21	32	43	65	87

Közvetlen kiválasztásos rendezés

Megkeresi a rendezetlen tömbrészlet legkisebb elemét, és az elejére rakja.

```
void kozvetlenRendez(double szamok[], int meret){
  int i, j, minIndex;
  for(i = 0; i < meret -1; i++){
    minIndex = i;
    for(j = i + 1; j < meret; j++)
      if(szamok[j] < szamok[minIndex])
        minIndex = j;
    if(minIndex != i){
      double temp = szamok[minIndex];
      szamok[minIndex] = szamok[i];
      szamok[i] = temp;
    }
  }
}
```

32	65	21	87	43
21	65	32	87	43
21	32	65	87	43
21	32	43	87	65
21	32	43	65	87

Forrás: https://infoc.eet.bme.hu/jegyzet.php

Közvetlen kiválasztásos rendezés

1) Jegyezzük meg, hogy a tömb első eleme helyére akarjuk betenni a tömb legkisebb elemét!
2) Keressük meg a legkisebb elemet! Először feltételezzük, hogy az első elem a legkisebb, és ezzel hasonlítjuk össze sorra a tömb elemeit.
3) Ha találunk kisebbet, akkor ettől kezdve ezzel hasonlítunk.
4) Ismételjük a 3. lépést a tömb végéig! Így megtaláltuk a legkisebbet.
5) Cseréljük ki a megjegyzett elemet a legkisebbre, ha nem a megjegyzett volt a legkisebb!
6) Jegyezzük meg, hogy az előzőleg megjegyzett utáni tömbelem helyére akarjuk becserélni a maradék tömbből a legkisebbet, és ismételjük a lépéseket a 2.-tól, míg a megjegyzett tömbelem az utolsó nem lesz!
7) A tömb rendezett.

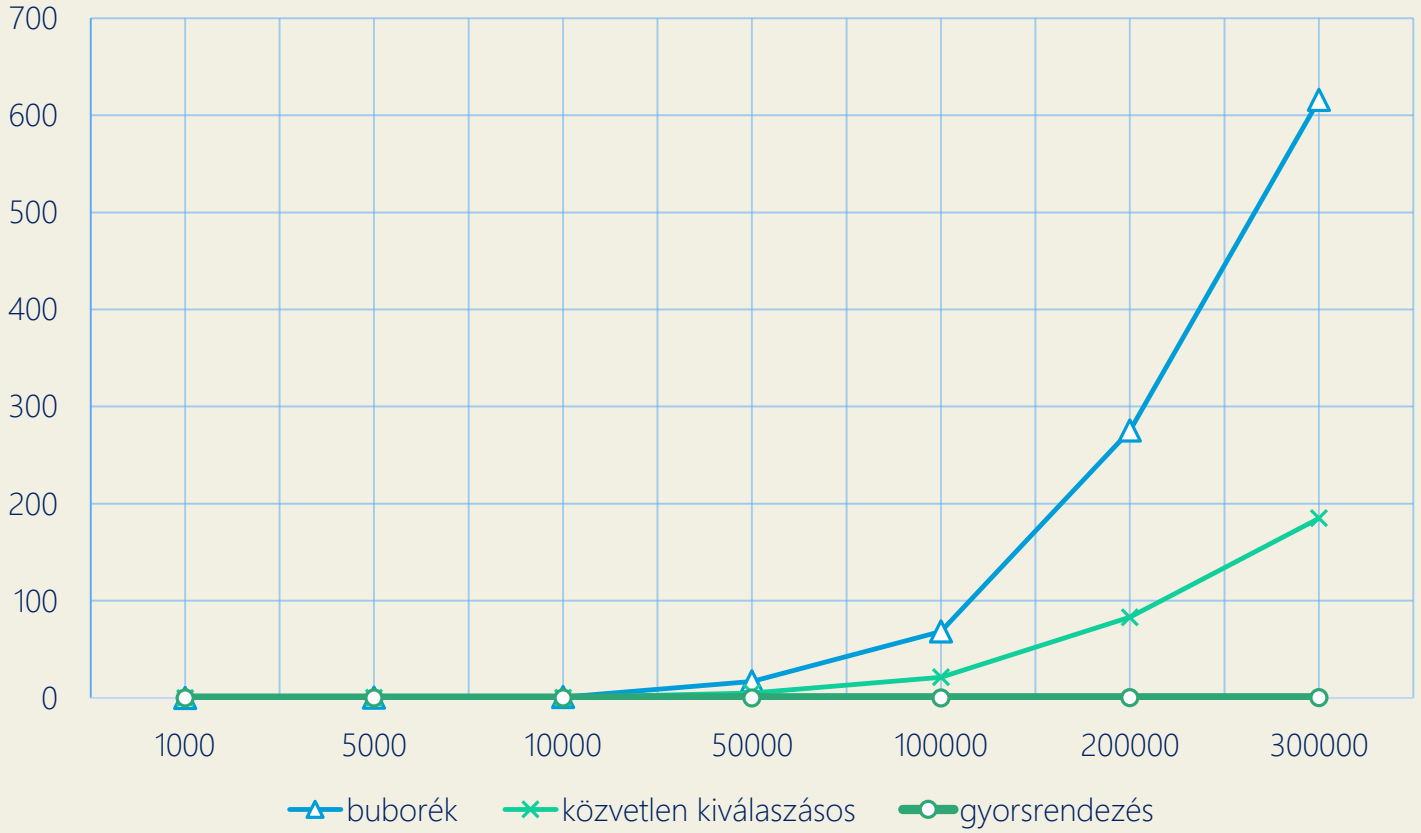
Forrás: https://infoc.eet.bme.hu/jegyzet.php

Rendezések hatékonysága cserék alapján, n elemű tömbre

rendezés	összehasonlítás			cserék		
	max	átlag	min	max	átlag	min
javított buborék	n^2	n^2	n	n^2	n^2	0
közvetlen kiválasztás	n^2	n^2	n^2	n	n	0
gyorsrendezés	n^2	n·logn	n·logn	n^2	n·logn	0
kupacrendezés	n·logn	n·logn	n·logn	n·logn	n·logn	n·logn

Forrás: https://infoc.eet.bme.hu/

Rendezések hatékonysága (futási idő)



elemszám	1000	5000	10000	50000	100000	200000	300000
buborék	0	0,174	0,72	16,86	68,28	275,328	616,008
közvetlen kiválasztásos	0	0,054	0,216	5,1	21,144	82,944	185,064
gyorsrendezés	0	0	0	0,012	0,024	0,06	0,078

Decimális, hexadecimális és bináris számok

Dec	Hex	Bin
0	0	0000 0000
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
4	4	0000 0100
5	5	0000 0101
6	6	0000 0110
7	7	0000 0111
8	8	0000 1000
9	9	0000 1001
10	A	0000 1010
11	B	0000 1011
12	C	0000 1100
13	D	0000 1101
14	E	0000 1110
15	F	0000 1111

Dec	Hex	Bin
16	10	0001 0000
17	11	0001 0001
18	12	0001 0010
19	13	0001 0011
20	14	0001 0100
21	15	0001 0101
22	16	0001 0110
23	17	0001 0111
24	18	0001 1000
25	19	0001 1001
26	1A	0001 1010
27	1B	0001 1011
28	1C	0001 1100
29	1D	0001 1101
30	1E	0001 1110
31	1F	0001 1111

Hexadecimális számok

- Pl. MAC cím: **`01:23:45:67:89:ab`**
- Pl. CAN üzenetek: **`225 08 00 A2 00 00 00 00 00 CF`**
- Pl. színek: **`#ffea80`**
- Pl. memória címek: **`*(0x0012DFAF)`**
- Könnyen számolható, akár fejben is binárisról
- Elterjedt számábrázolás

Hexadecimális:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimális:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Feladat 10_1.C Bitműveletek

- Készítsen **`unsigned rotl(unsigned)`** függvényt, mely visszaadja 1 bittel balra forgatva paramétere értékét!
- Készítsen **`unsigned rotr(unsigned)`** függvényt, mely 1 bittel jobbra forgatva adja vissza paramétere értékét!
- Készítsen **`unsigned tobbrotl(unsigned, unsigned)`** függvényt, mely az első paramétere értékét a másodikban megkapott bitszámmal forgatja balra!
- Készítsen **`unsigned tobbrotr(unsigned, unsigned)`** függvényt, mely az első paramétere értékét a másodikban megkapott bitszámmal forgatja jobbra!
- A függényeket kipróbáló programban célszerű binárisan megjelentetni a forgatni kívánt unsigned forgatás előtti és utáni állapotát, akár egy **`void BitekKi(unsigned)`** függvény segítségével.

Biteltolás (bit shift)

```
i = 14;       //1110
j = i >> 1; //0111 = 7 // fele a 14-nek

dec 5225 =bin 0001010001101001
  5225 >> 0 = 0001010001101001       // 5225
  5225 >> 1 = 0000101000110100       // 2612
  5225 >> 2 = 0000010100011010       // 1306
  5225 >> 3 = 0000001010001101       // 653
  5225 >> 4 = 0000000101000110       // 326
```

Bitforgatás (bit rotation, circular shift)

	decimális	bináris
Eredeti érték:	32769	00000000000000001000000000000001
Egy bittel balra:	65538	00000000000000010000000000000010
Négy bittel jobbra:	536875008	00100000000000000001000000000000
Egy bittel jobbra:	268437504	00010000000000000000100000000000
Négy bittel balra:	32769	00000000000000001000000000000001

További bitműveletek

```
  11001110
| 10011000
= 11011110
```

```
  11001110
& 10011000
= 10001000
```

Symbol	Operator
&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR
<<	left shift
>>	right shift
~	one's complement (unary)

Feltételes kifejezés

Értéke egy logikai feltételtől függ, jobbértékként használható
Gyakorlatilag egy logikai elágazás utasítás tömörebb írásmódja.

```
kifejezés ? kifejezésHaIgaz : kifejezésHaHamis
```

```
strcpy(ezString, ezSzam < 0 ? "neg" : "pos");
```

```
kisebb = a < b ? a : b;
```

Lásd még: háromoperandusú operátor, **?:** , ternary operator, conditional operator, inline if, Elvis operator

Kifejezések

- `if(ezSzam != 0)`
- `if(ezSzam)`

false 0
true 1 (vagy egyéb szám)

- `if(ezSzam == 0)`
- `if(!ezSzam)`

- `while(!getline(ezString, MAX) || !szame(ezString));`
  - » Addig fut, míg a az üres sor vagy pedig nem szám az input (ezString)
  - » Az első számnál, ami nem üres sor továbblépünk a következő utasításra

Feladat Összead, while

- Kérjen be egymás után **a** illetve **b** értékeket. A program mindaddig fusson, amíg **a** és **b** értékére nem érkezik tízes számrendszerbeli egész szám és **a** és **b** összege nem lesz egyenlő tízzel.
- Segítség:
  - `getline`
  - `szame`
  - `atoi`
  - `do.. while`

`osszead.c`

limits.h

Tartalmazza konstansként azokat a limiteket, definíciókat, amik a legtöbb változó leírására alkalmasak. Az értékek implementációfüggőek, de nem kisebb tartományba esőek, mint amit a C követelményként előír.

```
#include <limits.h>
…
printf("INT_MIN:\t%12d\n", INT_MIN);
printf("INT_MAX:\t%12d\n", INT_MAX);
printf("CHAR_MIN:\t%12d\n", CHAR_MIN);
printf("CHAR_MAX:\t%12d\n", CHAR_MAX);
```

math.h

Függvény	Leírás
abs	Abszolútérték
sqrt	Gyökvonás
sin	Szinusz
cos	Koszinusz
tan	Tangens
pow(2,3)	Hatványozás 2^3
round	Kerekítés

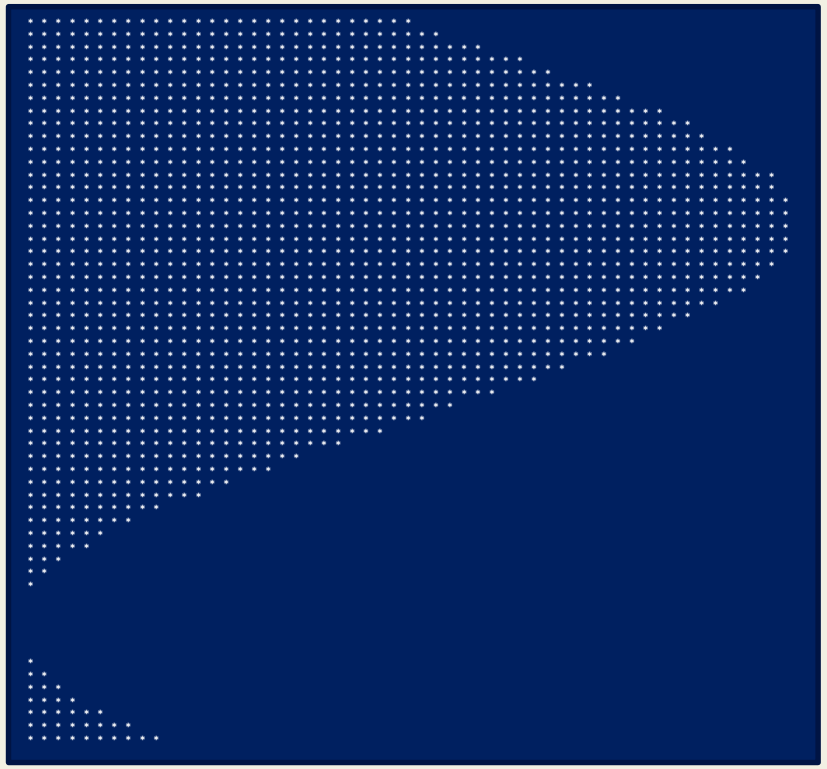
Részletesen: http://en.wikipedia.org/wiki/C_standard_library

Feladat szinusz görbe

- Rajzoljuk ki a szinusz alakzatot a `<math.h>` segítségével.
- Segítség:
  - » for ciklus
  - » típuskényszerítés (int)
  - » negatív értékek
  - » **`cmd`** » tulajdonságok » elrendezés sorok és oszlopok mérete (80x25)

Feladat szinusz görbe

```
#include <stdio.h>
#include <math.h>
void main(){
  int i, j;
  double d = 0;
  for(i=0; i<100; i++){
    for(j=0; j<(int)((28*(sin(d)))+28); j++)
      printf("-");
      d=d+0.1;
      printf("\n");
   }
  getchar();
}
```



Feladat másodfokú egyenlet

Kérjen be három nullától különböző valós számot (**`a,b,c`** együtthatók) szabvány bemenetről. Amennyiben nem számot kapunk, természetesen kérjen új értéket. Amennyiben értékek alapján alkotott másodfokú egyenletnek (**$ax^2+bx+c=0$**) van valós gyöke írja ki az eredményt, különben pedig **`"Az egyenletenek nincs valos gyoke"`** szöveget. Az egyenletnek akkor van valós gyöke, ha a négyzetgyökjel alatt álló diszkrimináns nemnegatív.

+feladat: A program ismételje üres sorig vagy EOF-ig a másodfokú egyenletek megoldását. Üres sor természetesen bármelyik együttható helyett érkezhet.

Feladat másodfokú egyenlet

$ax^2+bx+c=0$

$$x1,x2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Négyzet:
pow `#include <math.h>`
Gyök:
sqrt `#include <math.h>`
String valós számmá:
atof `#include <stdlib.h>`
Valós (lebegőpontos) szám-e:
lebege

Diszkrimináns számolása:

```
diszkr = pow(b,2) - 4*a*c;
```

```
x1=(-b + sqrt(diszkr))/(2*a);
x2=(-b - sqrt(diszkr))/(2*a);
```

Példa:
$4x^2+8x-5=0$
x1=0.5 x2=-2,5

Beolvasás példák

- Beolvasás » az első üres sornál kilép a ciklusból:
  - » `while(getline(input, MAX)) {..}`
  - » `while((getline(input, MAX)) != 0) {..}`
- Beolvasás » kilép, ha üres a sor vagy **abc** a string
  - » `while(  getline(input, MAX) &&  strcmp(input, "abc")) {..}`
  - » `while(!(!getline(input, MAX) || !strcmp(input, "abc")))){..}`
- Beolvasás » kilép, ha üres a sor vagy valós szám
  - » `while(  getline(input, MAX) &&  lebege(input)) {..}`

Beolvasás példák

Beolvasás » 3 valós nemnulla szám tömbbe olvasása

```
while(db<3){
   printf("%c: ", db + 'a');
   getline(input, MAX);
   if(atof(input)!=0)
     tomb[db++]=atof(input);
   }
```

Beolvasás » 3 valós nemnulla szám tömbbe olvasása

```
printf("%c: ", db + 'a');
while(getline(input, MAX) && db<3 ){
   if(atof(input)!=0)
     tomb[db++]=atof(input);
   if(db<3)
     printf("%c: ", db + 'a');
}
```

Beolvasás példák

Beolvasás » érvénytelen értékig fogad el a, b, c.. értékeket

```
do{
  printf("%c: ",'a'+ t++);
  getline(sor,MAX);
}while(lebege(sor));
printf("Nem valos szam");
```

Beolvasás » a valós szám beolvasása és számmá konvertálása

```
do{
   printf("a: ");
   getline(sor,MAX);
}while(!lebege(sor));
printf("a ereke %f", atof(sor));
```

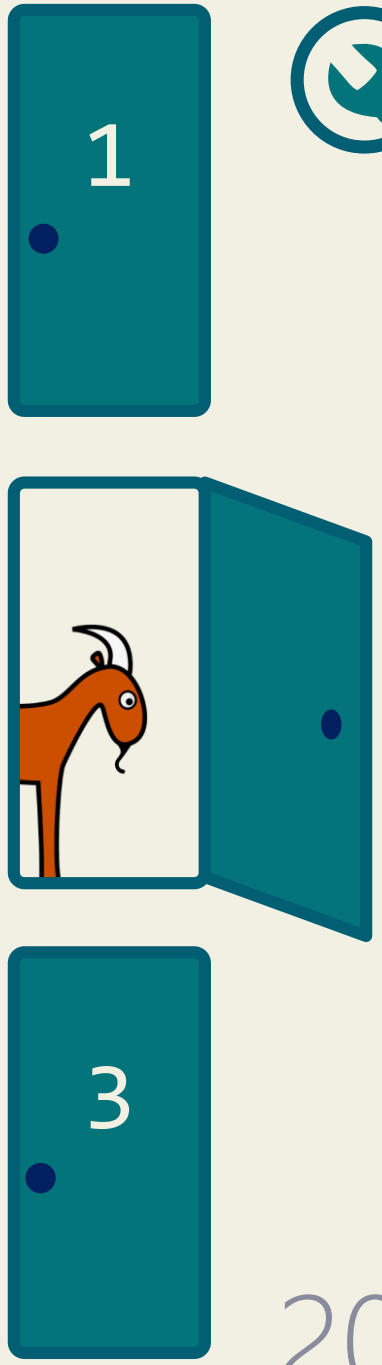
Feladat - Beolvasás gyakorlás

- **`Gyak01:`** Olvasson be 5.5-nél nagyobb valós számokat üres sorig vagy hibás bemenetig
- **`Gyak02:`** Olvasson be hibás bemenetig stringeket, amelyeknek második karaktere **`'a'`**

Feladat 11_2.c indexe

Készítsen **`int indexe(char s[], char t[])`** és **`int indexu(char s[], char t[])`** függvényeket, melyek meghatározzák és visszaadják a t paraméter karakterlánc **`s`** karakterláncbeli első, illetve utolsó előfordulásának indexét! Egy a függvényeket felhasználó tesztprogram írja ki a képernyőre egy adott próbakarakterlánc összes elofőrdulását is a billentyűzetről érkező sorokban! A próbakarakterlánc összes előfordulásának közlését - később visszatérve - oldjuk meg mutató segítségével is

Monty Hall-probléma



Egy TV-s játékosnak mutatnak három csukott ajtót, amelyek közül kettő mögött egy-egy kecske van, a harmadik mögött viszont egy vadonatúj autó. A játékos nyereménye az, ami az általa kiválasztott ajtó mögött van. Azonban a választás meg van egy kicsit bonyolítva. Először a játékos csak rámutat az egyik ajtóra, de mielőtt valóban kinyitná, a műsorvezető a másik két ajtó közül kinyit egyet, amelyik mögött nem az autó van (a játékvezető tudja, melyik ajtó mögött mi van), majd megkérdezi a játékost, hogy akar-e módosítani a választásán. A játékos ezután vagy változtat, vagy nem, végül kinyílik az így kiválasztott ajtó, mögötte a nyereménnyel. A paradoxon nagy kérdése az, hogy érdemes-e változtatni, illetve hogy számít-e ez egyáltalán.
A válasz, hogy mindig érdemes váltani, ez azonban annyira ellentmond a józan észnek, hogy a problémát paradoxonnak tekinthetjük.

Forrás: https://hu.wikipedia.org/wiki/Monty_Hall-paradoxon

Monty Hall-probléma

Írjunk C nyelvű programot, amely a Monty Hall problémát szimulálja. A felhasználótól kérjünk egy iterációszámot, majd ennek megfelelő véletlenszerű felállás mellett válasszuk a cserét, és végül mutassuk meg, mekkora a nyerési arány. (Minél nagyobb a szám, annál közelebb lesz a 66,6%-hoz)

```
Hany iteracio legyen: 20
Vegrehajtunk 20 iteraciot

[K][A][K] 1. valasztottuk, 3 megmutatott Cserevel nyert!
[K][A][K] 1. valasztottuk, 3 megmutatott Cserevel nyert!
[K][K][A] 2. valasztottuk, 1 megmutatott Cserevel nyert!
[K][K][A] 3. valasztottuk, 2 megmutatott :(
[K][A][K] 2. valasztottuk, 3 megmutatott :(
[A][K][K] 2. valasztottuk, 3 megmutatott Cserevel nyert!
[K][A][K] 2. valasztottuk, 3 megmutatott :(
[K][K][A] 2. valasztottuk, 1 megmutatott Cserevel nyert!
[K][K][A] 3. valasztottuk, 2 megmutatott :(
[K][K][A] 1. valasztottuk, 2 megmutatott Cserevel nyert!
[A][K][K] 3. valasztottuk, 2 megmutatott Cserevel nyert!
[K][A][K] 2. valasztottuk, 3 megmutatott :(
[K][K][A] 1. valasztottuk, 2 megmutatott Cserevel nyert!
[K][A][K] 1. valasztottuk, 3 megmutatott Cserevel nyert!
[K][K][A] 2. valasztottuk, 1 megmutatott Cserevel nyert!
[A][K][K] 3. valasztottuk, 2 megmutatott Cserevel nyert!
[K][A][K] 2. valasztottuk, 3 megmutatott :(
[A][K][K] 1. valasztottuk, 3 megmutatott :(
[A][K][K] 3. valasztottuk, 2 megmutatott Cserevel nyert!
[A][K][K] 3. valasztottuk, 2 megmutatott Cserevel nyert!

Nyeresiarany: 65.00%
```

Megoldás: **MontyHall1.c**

```
//Ellenőrzött input
változó=rossz érték;
while(változó nem jó értékű){
    printf(Mit kérünk, milyen határokkal, mértékegységben stb.);
    getline(s, MAX);
    if(egesze(s)) változó=atoi(s);
<else printf(Üzenet a formai hibáról);> }

//Ha a változó bármilyen értékű lehet, akkor esetleg indokolt lehet az ok logikai változó használata:
do{
    printf(Mit kérünk);
    getline(s, MAX-1);
if(ok=egesze(s)) változó=atoi(s);
else printf(Üzenet a formai hibáról); }
while(!ok);
//Természetesen ezeken kívül is még több jó megoldás lehetséges.
```

Goto

- "Ugrálást" tesz lehetővé a kódban
- Nehezen átlátható kódot eredményez
- Rontja a hibafelderítést

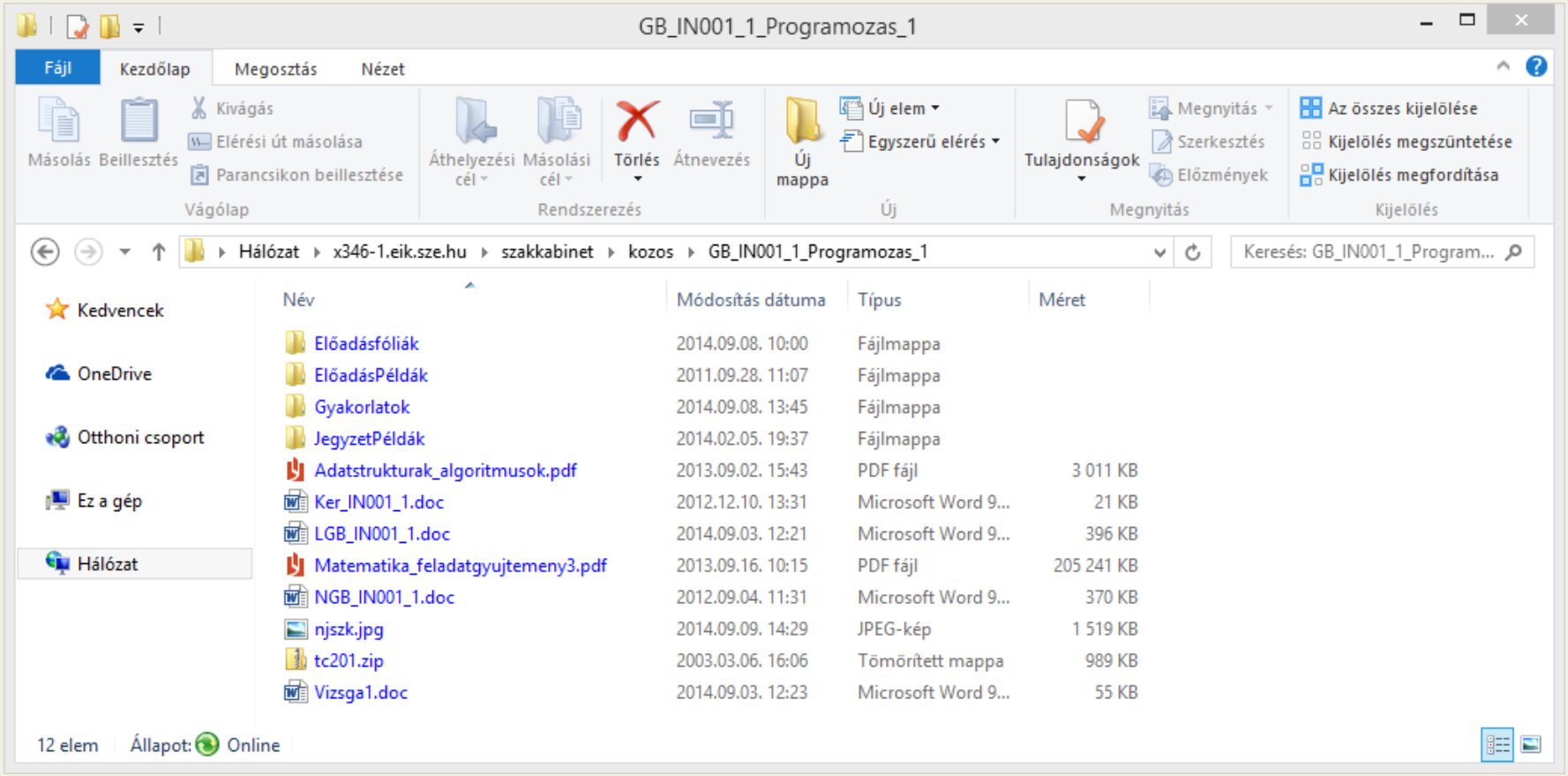
- A strukturált programozás alapelve szerint *szekvenciából, elágazásból* és *ciklusból* - matematikailag bizonyítottan - fel lehet építeni a programot ezért, a goto utasítást
- **ne használjuk!**
- Go To Statement Considered Harmful

Legfontosabb header fájlok

`<stdio.h>`	Standard I/O műveletek (pl. **`printf, getchar`**)
`<stdlib.h>`	További standard műveletek, memória (pl. **`atof, atoi`**)
`<string.h>`	String műveletek (pl. **`strlen, strcpy`**)
`<math.h>`	Matematikai műveletek (pl. **`sin, pow`**)
`<time.h>`	Időkezelés (pl. **`time, clock`**)
`<ctype.h>`	Karakterek kezelése (pl. **`isdigit, tolower`**)
`<limits.h>`	Makrók, az alap típusok méretei (pl. **`INT_MIN, CHAR_MAX`**)

K meghajtó

\\x346-1.eik.sze.hu\szakkabinet\kozos\GB_IN001_1_Programozas_1



Felhasznált irodalom

- **A C programozási nyelv** - Az ANSI szerinti változat. **B. W.** Kernighan - **D. M. Ritchie**; Műszaki Könyvkiadó, 1995
- **Programozás C nyelven** - Pere László, Kiskapu 2003
- **Programozási feladatok** és algoritmusok Turbo C és C++ nyelven Benkő Tiborné, Benkő László, ComputerBooks 1997
- **A programozás alapjai -** Pohl László, Budapest, 2010
- **Programozás I-II. C programnyelv -** Bauer Péter, Universitas-Győr Kht., 2005
- infoc.eet.bme.hu