

Programozási nyelvek 1

Szathmáry László
Debreceni Egyetem
Informatikai Kar

5. előadás

- const, fordítás, hibakeresés, stílus
- sztringek

(utolsó módosítás: 2023. márc. 13.)

2022-2023, 2. félév



Tömb átadása paraméterként

- Amikor átadunk egy tömböt paraméterként, akkor érdemes átadni a tömb méretét, ill. magát a tömböt. Érdemes ezt a sorrendet követni, vagyis előbb a méretet adjuk át, aztán magát a tömböt.
- Amikor egy tömböt átadunk paraméterként, akkor MINDIG gondoljuk át, hogy a tömböt milyen céllal adjuk át.
 - Azt akarom, hogy a meghívott függvény módosítsa a tömbömet.
 - Csak át akarom adni a tömböt feldolgozásra, de nem szeretném, ha a függvény bármit is módosítana a tömbön.

Tömb átadása paraméterként

```
1  #include <stdio.h>
2
3  void szoroz_10_zel(int n, int tomb[])
4  {
5      for (int i = 0; i < n; ++i)
6      {
7          tomb[i] *= 10;    // tomb[i] = tomb[i] * 10;
8      }
9  }
10
11 int main()
12 {
13     int szamok[4] = { 1, 4, 8, 3 };
14     int meret = 4;
15
16     szoroz_10_zel(meret, szamok);
17     printf("%d\n", szamok[0]);
18     printf("%d\n", szamok[1]);
19     printf("%d\n", szamok[2]);
20     printf("%d\n", szamok[3]);
21
22     return 0;
23 }
24
```

Tömb átadása paraméterként

```
1  #include <stdio.h>
2
3  int sum(int n, const int tomb[])
4  {
5      int osszeg = 0;
6      for (int i = 0; i < n; ++i)
7      {
8          osszeg += tomb[i];
9      }
10
11     return osszeg;
12 }
13
14 int main()
15 {
16     int szamok[4] = { 1, 4, 8, 3 };
17     int meret = 4;
18
19     int osszeg = sum(meret, szamok);
20     printf("osszeg: %d\n", osszeg);
21     //
22     printf("%d\n", szamok[0]);
23     printf("%d\n", szamok[1]);
24     printf("%d\n", szamok[2]);
25     printf("%d\n", szamok[3]);
26
27     return 0;
28 }
29
```

`const` = *read-only*

A változó kezdőértéket kap az inicializáció során, majd a változó értéke nem változhat meg (*immutable*).

Ez mire jó?

Kapunk egy garanciát arra, hogy a változó értéke nem fog megváltozni.

A változó védve van a véletlen módosításoktól. Ha mégis módosítani szeretnénk, akkor fordítási hibát kapunk.

const

```
1  #include <stdio.h>
2
3  int terület(const int a, const int b)
4  {
5      return a * b;
6  }
7
8  int main()
9  {
10     const int a = 3;
11     const int b = 2;
12
13     const int t = terület(a, b);
14     printf("terület: %d\n", t);
15
16     return 0;
17 }
18
```

Tetszőleges változó ellátható a *const* módosító jelzővel.

Vagyis nem csak a formális paraméterlistán használható!

Ha egy változó értékét szeretnénk megvédeni a módosításoktól, akkor használjuk bátran!

Tipp: miután megírtuk a programunkat, nézzük át ismét, s gondoljuk át, hogy miket lenne érdemes *const* -tá tenni. Mi az, aminek végleges az értéke, ami biztosan nem módosul (vagy legalábbis nem kellene módosulnia)?

Rust példa

```
5
6 fn terület(a: i32, b: i32) -> i32
7 {
8     return a * b;
9 }
10
11 fn main()
12 {
13     let mut a: i32 = 3;
14     let b: i32 = 2;
15
16     a = 4;
17     b = 5;    // Hiba! b immutable
18
19     let t = terület(a, b);
20     println!("terület: {}", t);
21 }
22
```

Rust-ban a *read-only* mód az alapértelmezés.

Ha valamit módosíthatóvá akarunk tenni, akkor azt külön jeleznünk kell.

A modernebb programozási nyelvekben lévő jó ötleteket nyugodtan át lehet venni. Lehet őket „backportolni” C-be.

Kotlin példa

```
sum.kt
1 fun sum(x: Int, y: Int): Int
2 {
3     x = 5;    // Hiba! x read-only
4     return x + y
5 }
6
7 fun main()
8 {
9     println(sum(1, 2))
10 }
11
```

Kotlin-ban a formális paraméterek alapértelmezetten *read-only* módban vannak! A hívás pillanatában kezdőértéket kapnak, ami utána nem módosítható.

Ha mégis módosítani akarjuk valamelyik paramétert, akkor arról készítünk egy módosítható másolatot.

A modernebb programozási nyelvekben lévő jó ötleteket nyugodtan át lehet venni. Lehet őket „backportolni” C-be.

Fordítás

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello world\n");
6
7     return 0;
8 }
```

```
$ gcc hello.c
$ ./a.out
```

Leegyszerűsítve:



Fordítás

```
1  #include "prog1.h"
2  #include <stdio.h>
3
4  int main()
5  {
6      string s = get_string("Mi a neved?\n");
7
8      printf("Hello %s!\n", s);
9
10     return 0;
11 }
12
```

```
$ gcc hello1.c prog1.c
```

```
$ ./a.out
```

Fordítás

A fordítás lépései részletesen:

- előfeldolgozás (*preprocessing*) [C forrás → C forrás]
- fordítás (*compiling*) [C forrás → tárgykód]
 - assembly kód generálása [C forrás → assembly forrás]
 - assembly kód fordítása [assembly forrás → tárgykód]
- kapcsolatszerkesztés (*linking*) [tárgykód(ok) → futtatható EXE]

Fordítás - előfeldolgozás

```
#include "prog1.h"
#include <stdio.h>

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```

Fordítás - előfeldolgozás

```
#include "prog1.h"
#include <stdio.h>

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```

Fordítás - előfeldolgozás

```
string get_string(const string prompt);  
#include <stdio.h>  
  
int main()  
{  
    string s = get_string("Mi a neved?\n");  
  
    printf("Hello %s!\n", s);  
  
    return 0;  
}
```

Fordítás - előfeldolgozás

```
string get_string(const string prompt);  
#include <stdio.h>  
  
int main()  
{  
    string s = get_string("Mi a neved?\n");  
  
    printf("Hello %s!\n", s);  
  
    return 0;  
}
```

Fordítás - előfeldolgozás

```
string get_string(const string prompt);  
int printf(string format, ...);  
  
int main()  
{  
    string s = get_string("Mi a neved?\n");  
  
    printf("Hello %s!\n", s);  
  
    return 0;  
}
```

Fordítás - előfeldolgozás

```
...
string get_string(const string prompt);
int printf(string format, ...);
...

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```


Fordítás - fordítás

```
1  .file "hello1.c"
2  .text
3  .section .rodata
4  .LC0:
5  .string "Mi a neved?\n"
6  .LC1:
7  .string "Hello %s!\n"
8  .text
9  .globl main
10 .type main, @function
11 main:
12 .LFB0:
13 .cfi_startproc
14 pushq %rbp
15 .cfi_def_cfa_offset 16
16 .cfi_offset 6, -16
17 movq %rsp, %rbp
18 .cfi_def_cfa_register 6
19 subq $16, %rsp
20 leaq .LC0(%rip), %rdi
21 call get_string@PLT
22 movq %rax, -8(%rbp)
23 movq -8(%rbp), %rax
24 movq %rax, %rsi
25 leaq .LC1(%rip), %rdi
26 movl $0, %eax
27 call printf@PLT
28 movl $0, %eax
29 leave
30 .cfi_def_cfa 7, 8
31 ret
32 .cfi_endproc
33 .LFE0:
34 .size main, .-main
35 .ident "GCC: (Arch Linux 9.2.1+20200130-2) 9.2.1 20200130"
36 .section .note.GNU-stack,"",@progbits
37
```

C kód



assembly kód



tárgykód

Fordítás - fordítás

```
00010111111000111011001100011010011111
10101000111000011000101001100100110000
11001101010000001000110111010101000111
00000000111110010101100110001001001101
01000101010000010000010110111010110011
01100110010000100010110001000110101100
11000100011101000110111010111101000011
11111011000011010101100101011000010101
11100000100001001011010110001001111011
11001001100000010111000001010100100000
11111110001110010111011001111110100000
01011101000100101010101011000100100111
1101010111110111010110111110100011011000
01011110100000100000110111011011110001
100101101011111110000110011101110110100
11001010001010111111100010111011000011
10100011000011010111001111111000101111
00101110010010011111000111100000000001
```

C kód



assembly kód



tárgykód

Házi feladat:

Egy ilyen szemléltető
ábra generálása.

Fordítás - kapcsolatszerkesztés

```
#include "prog1.h"
#include <stdio.h>

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```

Fordítás - kapcsolatszerkesztés

```
#include "prog1.h"
#include <stdio.h>

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```

Fordítás - kapcsolatszerkesztés

```
#include "prog1.h"  
#include <stdio.h>  
  
int main()  
{  
    string s = get_string("Mi a neved?\n");  
  
    printf("Hello %s!\n", s);  
  
    return 0;  
}
```

Fordítás - kapcsolatszerkesztés

```
#include "prog1.h"
#include <stdio.h>

int main()
{
    string s = get_string("Mi a neved?\n");

    printf("Hello %s!\n", s);

    return 0;
}
```

Fordítás - kapcsolatszerkesztés

hello1.c

prog1.c

stdio.c

Fordítás - kapcsolatszerkesztés

tárgykódok

hello1.c



0101000
1010010
1001001
001...

prog1.c



1100101
0010100
0101001
110...

stdio.c



0010100
1001001
0001111
001...

Fordítás - kapcsolatszerkesztés

tárgykódok

hello1.c



0101000
1010010
1001001
001...

prog1.c



1100101
0010100
0101001
110...

stdio.c



0010100
1001001
0001111
001...

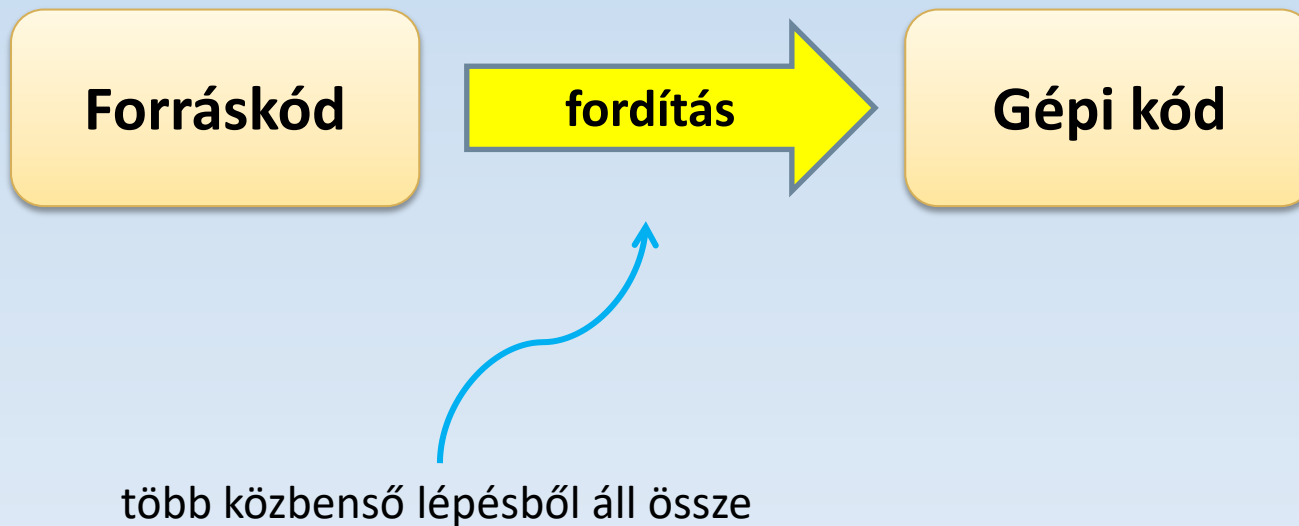
kapcsolat-
szerkesztés

futtatható EXE

...0100100100100010110101011
11011010100010010101010100
10010111010101000100010010
010101001001010...

Fordítás

Leegyszerűsítve:



Hibakeresés (debugging)



A **bug** a számítógépes **programhiba** elterjedt elnevezése.

Hibák esetén beszélhetünk

- szintaktikai, és
- szemantikai

hibákról.

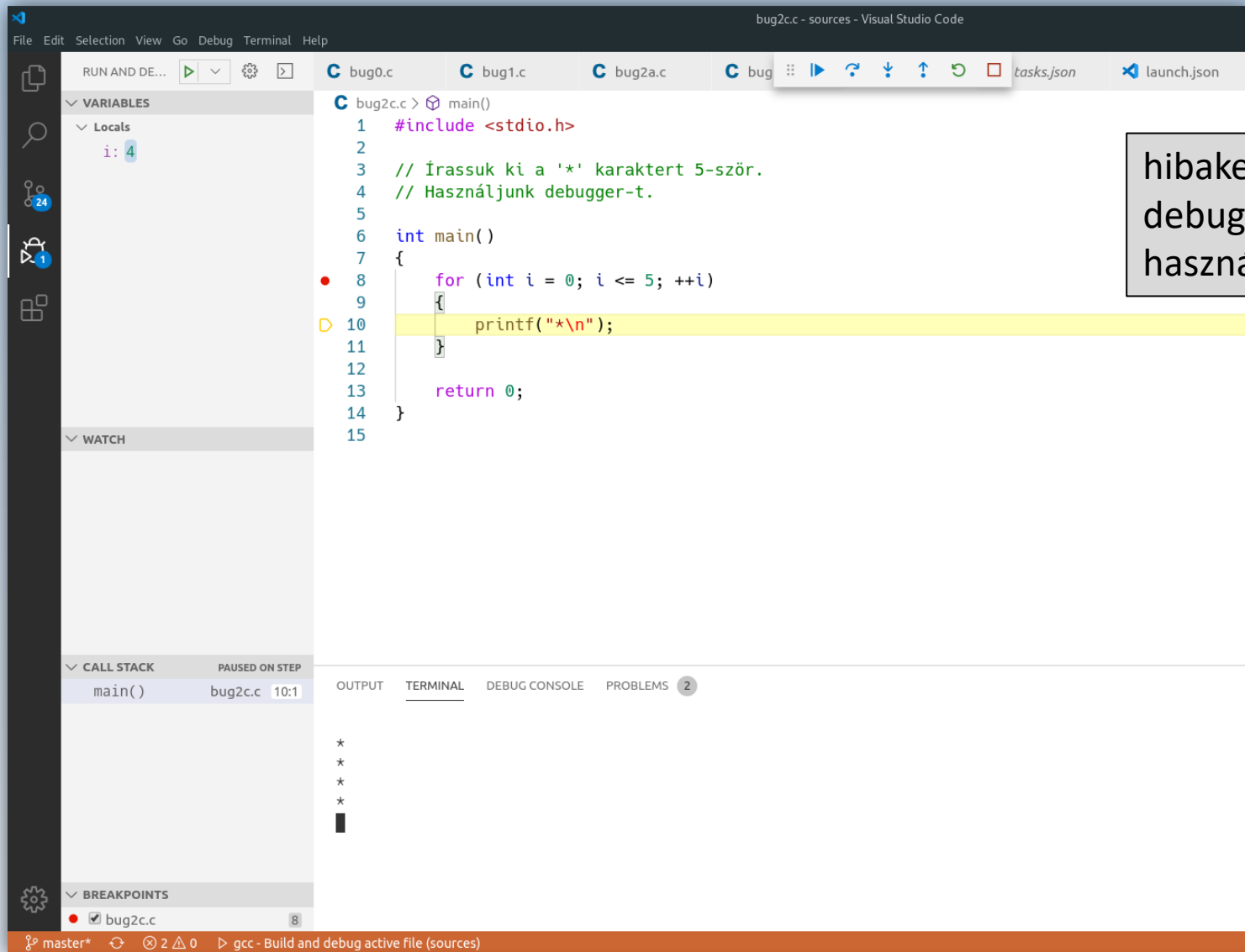
Hibakeresés

Hibakeresési módszerek:

- `printf()` használata (legegyszerűbb módszer, debug információk kiírása)
- debugger használata (ez lehet parancssoros vagy grafikus felülettel rendelkező)

Hibakeresés

hibakeresés
debugger
használatával



```
bug2c.c - sources - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
RUN AND DE... [play] [stop] [refresh] [step over] [step into] [step out] [step back] [step forward] [run and debug] [tasks.json] [launch.json]
VARIABLES
  Locals
    i: 4
WATCH
CALL STACK
  PAUSED ON STEP
    main() bug2c.c 10:1
BREAKPOINTS
  bug2c.c 8
bug2c.c
1 #include <stdio.h>
2 // Írassuk ki a '*' karaktert 5-ször.
3 // Használjunk debugger-t.
4
5
6 int main()
7 {
8     for (int i = 0; i <= 5; ++i)
9     {
10         printf("*\n");
11     }
12
13     return 0;
14 }
15
OUTPUT
*
*
*
*
*
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
master* 2/0 gcc - Build and debug active file (sources)
```

Stílus

„A stílus maga az ember.”

Georges-Louis Leclerc de Buffon

Az, hogy megírok egy programot és az lefut helyesen, az egy dolog.

Törekedjünk arra, hogy maga a forráskód is **szép** legyen. Egy program elkészítése egy szellemi termék, egy mérnöki munka. Nem mindegy, hogy mit adunk ki a kezünk közül!

```
1 #include <stdio.h>
2 int main(){
3     printf("hello\n");
4     return 0;}
5
```



```
1 #include <stdio.h>
2
3 int main( )
4 {
5     printf("hello\n");
6
7     return 0;
8 }
```

Stílus

- gondoljunk magunkra
- gondoljunk másokra
- gondoljunk arra, hogy egy programot karban is kell tartani

Ha majd elmennek dolgozni, akkor valószínűleg a cégnél is lesz egy *style guide*, amit minden programozónak követnie kell.

Nekünk is van egy *style guide*-unk, a félév során ehhez próbáljuk tartani magunkat ([link](#), github).

Stílus

linter: olyan szoftver, amely a forráskódot elemzi. Célja: hibák, bug-ok felderítése, stílusbeli problémák jelzése.

A style50 nevű stíuselemző telepítése:

```
$ pip3 # győződjünk meg róla, hogy ez fent van  
  
$ pip3 install style50 --user -U  
$ sudo apt install astyle # ez is kell hozzá
```

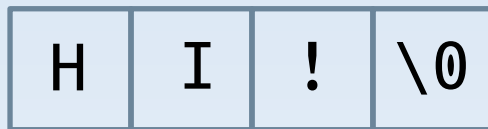
Használata:

```
$ style50 program.c
```


Sztringek

- Egy sztring valójában egy karaktereket tartalmazó tömb.
- C-ben egy karakter (char) mérete 1 byte.
- Ami különlegessé teszi: az utolsó karaktere után szerepel egy speciális karakter (' \0 '), ami a sztring végét jelzi.

```
string s = "HI!";
```

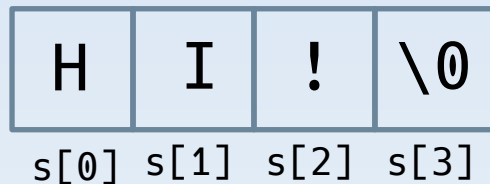


s

Sztringek

- Egy sztring valójában egy karaktereket tartalmazó tömb.
- C-ben egy karakter (char) mérete 1 byte.
- Ami különlegessé teszi: az utolsó karaktere után szerepel egy speciális karakter (`'\0'`), ami a sztring végét jelzi.

```
string s = "HI!";    // a mem.-ban 4 byte-ot foglal
```



Sztringek

```
string nevek[4];  
nevek[0] = "Anna";  
nevek[1] = "Bea";  
nevek[2] = "Cecil";  
nevek[3] = "David";
```

A nevek[0]	n	n	a	\0	B nevek[1]
e	a	\0	C nevek[2]	e	c
i	l	\0	D nevek[3]	a	v
i	d	\0			

Sztringek


0	<u>NUL</u>	16	<u>DLE</u>	32	<u>SP</u>	48	0	64	@	80	P	96	`	112	p
1	<u>SOH</u>	17	<u>DC1</u>	33	!	49	1	65	A	81	Q	97	a	113	q
2	<u>STX</u>	18	<u>DC2</u>	34	"	50	2	66	B	82	R	98	b	114	r
3	<u>ETX</u>	19	<u>DC3</u>	35	#	51	3	67	C	83	S	99	c	115	s
4	<u>EOT</u>	20	<u>DC4</u>	36	\$	52	4	68	D	84	T	100	d	116	t
5	<u>ENQ</u>	21	<u>NAK</u>	37	%	53	5	69	E	85	U	101	e	117	u
6	<u>ACK</u>	22	<u>SYN</u>	38	&	54	6	70	F	86	V	102	f	118	v
7	<u>BEL</u>	23	<u>ETB</u>	39	'	55	7	71	G	87	W	103	g	119	w
8	<u>BS</u>	24	<u>CAN</u>	40	(56	8	72	H	88	X	104	h	120	x
9	<u>HT</u>	25	<u>EM</u>	41)	57	9	73	I	89	Y	105	i	121	y
10	<u>LF</u>	26	<u>SUB</u>	42	*	58	:	74	J	90	Z	106	j	122	z
11	<u>VT</u>	27	<u>ESC</u>	43	+	59	;	75	K	91	[107	k	123	{
12	<u>FF</u>	28	<u>FS</u>	44	,	60	<	76	L	92	\	108	l	124	
13	<u>CR</u>	29	<u>GS</u>	45	-	61	=	77	M	93]	109	m	125	}
14	<u>SO</u>	30	<u>RS</u>	46	.	62	>	78	N	94	^	110	n	126	~
15	<u>SI</u>	31	<u>US</u>	47	/	63	?	79	O	95	_	111	o	127	<u>DEL</u>

Decimal ASCII Chart


<https://asciichart.com>

Sztringek

A C standard könyvtárának felhasználóbarát leírása:

 CS50 Programmer's Manual

Manual pages for the C standard library (and the CS50 Library), with student-friendly annotations.

 frequently used in CS50

ctype.h

`isupper` - character classification functions
`tolower` - convert uppercase or lowercase
`toupper` - convert uppercase or lowercase

<https://man.cs50.io>

Sztringek

Parancssori argumentumok

```
#include <stdio.h>

int main()
{
    printf("hello\n");

    return 0;
}
```

Sztringek

Parancssori argumentumok

```
#include "prog1.h"  
#include <stdio.h>  
  
int main(int argc, string argv[])  
{  
    printf("hello\n");  
    return 0;  
}
```

Sztringek

Parancssori argumentumok

```
$ ./a.out hello world
```



```
argv[0] argv[1] argv[2]
```


Sztringek

Parancssori argumentumok

```
#include <stdio.h>

int main()
{
    printf("hello\n");
    return 0;
}
```

ha nincs szükségem
a parancssori argumentumokra

```
#include "prog1.h"
#include <stdio.h>

int main(int argc, string argv[])
{
    printf("hello\n");
    return 0;
}
```

ha szükségem van
a parancssori argumentumokra

Kilépési kód

```
#include <stdio.h>

int main( )
{
    printf("hello\n");

    return 0;
}
```

- 0 kilépési kód (exit code): minden rendben volt, a program rendesen befejezte a működését
- 0-tól eltérő kilépési kód: valamilyen hiba történt

Kilépési kód

```
1  #include "prog1.h"
2  #include <stdio.h>
3
4  int main(int argc, string argv[])
5  {
6      if (argc != 2)
7      {
8          printf("Hiba! Adj meg egy paramétert!\n");
9          return 1;
10     }
11
12     printf("hello %s\n", argv[1]);
13
14     return 0;
15 }
```

A 9. sorban egy `exit(1);` hívás szebb lenne.
Az `exit()` függvény az `stdlib.h`-ban található meg.

Házi feladat

- A K & R-féle „C Bibliában” nézzék át azokat a részeket, amikről szó volt az előadáson.
- Juhász István jegyzetéből nézzék át azokat a fogalmakat, amikről szó volt az előadáson ([link](#)).
- C Style Guide tanulmányozása ([link](#))
- Mi a *linter*? ([link](#), wikipedia)
- A jövő heti gyakorlatra megírt forráskódjainkat nézzük át a style50 programmal is. A programokat próbáljuk megszépíteni.
- A CS50 manual-ban olvassunk utána az `exit()` függvénynek.

Szorgalmi

- Elolvasni: Bug ([link](#), wikipedia)
- Elolvasni: „Le style est l'homme même” ([link](#), wikipedia)