

Třetí počítačové cvičení

Základy programování (IZP)

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Petr Veigend, veigend@fit.vut.cz



- **Zadání prvního projektu, časté chyby**
- **Úhlednost zdrojového kódu**
- **Řetězce**

ZADÁNÍ PRVNÍHO PROJEKTU

ÚHLEDNOST ZDROJOVÉHO KÓDU

- Programy by vždy (u projektů je to **povinné**) měly začínat hlavičkou, např.:

```
/**  
* IZP Prvni projekt  
* Autor: Jan Novak, xnovak00@stud.fit..  
* Datum: 02.10.2025  
* Poznamka: program nefunguje vubec  
*/
```

- Každá proměnná a složitější konstrukce v těle funkce by měla být komentována (nebo pojmenována tak, aby komentář nebyl potřeba). Napomáhá to přehlednosti kódu
- Stručný příklad (*názvy proměnných*):

```
int a = 10; // first counter
int b = a; // second counter

// comparison between two counters
// if a == b, increments a, otherwise..
if (a == b) { ... }
```

- Tvar komentářů lze přizpůsobit pro generování dokumentace

- To, jak je proměnná pojmenovaná, **závisí pouze na programátorovi, ALE ...**
- **proměnné** pojmenované `test12345`, `mojenejuzasnejsipromenna` apod. asi nebudou **úplně nejvhodnější**
- Název by měl mít **minimálně 3 znaky**
- Indexy (v projektech ne „i“): `idx`, ...
- Řetězce: `str`, ...
- Znaky: `cha`, ...
- Konstanty: `KONSTANTA` (velká písmena)
- Datové typy: `TArray`, `ColorSpace`, ... (později)

- **Nedoporučuje se**, aby název proměnné začínal **podtržítkem** - mnoho takových názvů je definováno překladačem a mají proto zvláštní význam.
 - `int _promenna`
- Názvy proměnných a funkcí, které jsou delší, než jedno slovo můžeme zapisovat
 - `velmi_dlouhy_nazev_id`
 - `velmiDlouhyNazevId`
(angl. Camel-Caps, Camel-Case)
- Vyberte si jeden styl zapisování a **držte se ho**
- Je **vhodné (nikoliv nutné)** pojmenovávat v **angličtině**



- **Základní styly:**

- llvm

- <http://llvm.org/docs/CodingStandards.html>

- Mozilla

- https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style

- Gnu

- <https://www.gnu.org/prep/standards/standards.html>

- Proměnná / konstanta
 - Deklarace
 - Inicializace
- Datový typ
- Operátory
 - Přiřazení, porovnání
 - Aritmetické, Logické
- Příkaz, výraz

• Ukazatel

- proměnné **uchovávají adresu**, která ukazuje do paměti počítače
- Vážou se k určitému datovému typu
- Definice proměnné typu ukazatel:

```
bazovy_typ *nazev_promenne;
```

```
int *int_ptr; // ukazatel na int
```

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	10	20	30	40	50	60

- **Pole**
- **Deklarace staticky**

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	10	20	30	40	50	60

- **Pole:** prvky stejného typu, spojitě místo v paměti
- **Deklarace staticky:** `int moje_pole[6];`

- **Řetězec** bude vysvětlen podrobněji na příštích cvičeních a na přednášce.
- Prozatím :
 - Řetězec se skládá ze znaků (datový typ **char**)
 - Řetězec je **ukončen speciálním znakem** (' \0 ').
Z toho vyplývá, že s tím jako programátoři musíme počítat a přidělit řetězci dostatek paměti.
 - Při načítání (funkce scanf) u řetězců **nepoužíváme &**

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	'h'	'e'	'l'	'l'	'o'	'\0'

- **Řetězce:** pole typu `char` zakončená nulovým znakem `'\0'`
- **Pozor:** `char pole[5];`
 - Není zde místo pro ukončovací nulu (`'\0'`)
- **Pozor:** je nutné hlídat meze pole!

PŘÍKLADY

- Pro připomenutí, programy budeme překládat pomocí

```
gcc -std=c11 -Wall -Wextra -pedantic  
hello.c -o hello
```

- Ve cvičeních můžete použít online prostředí <https://hub.bazar.nesad.fit.vutbr.cz/hub/home>
 - Empty environment
 - Data se neukládají, uložte si je poté sami
- Automatizace překladu: **později**

- Deklaruje proměnnou **typu řetězec**
 - datový typ **char []**, délka řetězce **100** znaků
 - **char retezec[101];**
- **Načtěte** jeho hodnotu (**není &**)
(**scanf ("%100s", retezec);**)
 - 100 – omezí načítání na 100 znaků
 - načítání přeruší bílý znak (pozor na mezery...)
- **Vypište** obsah (**%s**)

- Napište program, který vypočítá délku řetězce
 - Řetězec opět načtete pomocí funkce `scanf`, maximální délka je **100 znaků**
 - Z minula: znaky mají přiřazenu číselnou hodnotu, lze je mezi sebou porovnávat pomocí operátoru `==`
 - **Jak lze projít přes všechny znaky v řetězci?**

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	'h'	'e'	'l'	'l'	'o'	'\0'

```
for(inicIALIZACE; test; inkrementace)  
{ }
```



```
inicIALIZACE;  
while(test)  
{ inkrementace; }
```

- Napište program, který vypočítá délku řetězce
 - Řetězec opět načtete pomocí funkce `scanf`, maximální délka je **100 znaků**
 - Z minula: znaky mají přiřazenu číselnou hodnotu, lze je mezi sebou porovnávat pomocí operátoru `==`
 - **Jak lze projít přes všechny znaky v řetězci?**
Cyklus

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	'h'	'e'	'l'	'l'	'o'	'\0'

- Napište program, který v řetězci (obsah opět pomocí funkce `scanf`) spočítá, kolik obsahuje a) písmen, b) číslic
 - Dvě proměnné: jedna pro počet písmen, druhá pro počet číslic

- ASCII
- `ctype.h`

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

- Napište program, který opět načte řetězec (max. délka 100 znaků) a převede jeho jednotlivé znaky (malá písmena) na velká písmena

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

- Řetězce **nemusíme** načítat funkcí **scanf**, někdy je výhodnější použít funkci **getchar** a načíst vstup **znak po znaku**

```
char radek[100]; // pole znaku
int c; // pro jeden znak ze vstupu
int i = 0;
while ((c = getchar()) != EOF)
{
    radek[i] = c;
    i++;
}
radek[i] = '\0'; // pridani znaku \0
```

- Zatím jsme program spouštěli jako

```
./hello
```

- Ale už minule jsme např. při překladu psali např.

```
gcc -std=c99 -Wall ...
```

- Jak do programu **gcc** argumenty `-std=c99` a `-Wall` předáme?

Přidáním parametrů do definice funkce `main`

- Jednotlivé argumenty se oddělují **mezerou**
- Argumenty, se kterými byl program spuštěn, se dají získat pomocí doplnění dvou parametrů do definice funkce **main**:

```
int main(int argc, char* argv[])
{
    // argc - počet argumentů
    // argv - jednotlivé argumenty, argv[0]
    // (název souboru s programem)
}
```

- Argumenty jsou opět **pole (čeho?)**

```
int main(int argc, char* argv[])
{
    // argc - počet argumentů
    // argv - jednotlivé argumenty, argv[0]
    // (název souboru s programem)
}
```

- Pro `./hello -sum 10 20` argc=4

argv[0]	argv[1]	argv[2]	argv[3]
"hello"	"-sum"	"10"	"20"

- **PowerShell, Linux**

 - `./program arg1 arg2 arg3`

- Jednotlivé argumenty jsou od sebe odděleny **mezerou**

- Vypište první argument programu na standardní výstup

argv[0]	argv[1]	argv[2]	argv[3]
"hello"	"-sum"	"10"	"20"

```
int main(int argc, char* argv[])
{
    // argc - počet argumentů
    // argv - jednotlivé argumenty, argv[0]
    // (název souboru s programem)
}
```

- Napište program, který
 - Bude jako první argument vyžadovat znak, např. pro znak **a**

argv[0]	argv[1]
"hello"	"a"

řetězec

- *Jak převedeme takový jednoznakový řetězec na znak?*
 - `argv[1][0]`
- Program načte řetězec (opět pomocí `scanf`, max. 100 znaků).
- V tomto řetězci vyhledá znak z argumentu a všechny výskyty nahradí znakem -

- Napište program, který
 - Bude jako první argument vyžadovat číslo, např. pro číslo **52**

argv[0]	argv[1]	
"hello"	"52"	řetězec

- Jak převedeme řetězec na číslo?
 - Funkce **atoi**, **atof**

- Často je potřeba zadaný řetězec převést na číslo
- Funkce
 - `atoi` (řetězec -> celé číslo)
 - `atof` (řetězec -> desetinné číslo)

```
#include <stdlib.h> // atoi, atof

int celeCislo = atoi("12");
float desetinneCislo = atof("3.14");
```

- **POZOR**
 - Řetězec `12abc` není číslo, výsledek funkce `atoi` bude 12
 - Řešení: zkontrolovat, zda se řetězec skládá pouze z číslic, funkce `strtod`, `strtod` (později)

- Napište program, který
 - Bude jako první argument vyžadovat číslo, např. pro číslo **52**

argv[0]	argv[1]
"hello"	"52"

řetězec

- Jak převedeme řetězec na číslo?
 - Funkce **atof**, **atoi** (`#include <stdlib.h>`)
- O toto číslo se „posunou“ jednotlivé znaky v řetězci → Caesarova šifra
 - Např. pro 1: abc → bcd

- Napište program, který porovná dva řetězce
 - **POZOR, pro porovnání řetězců nelze použít ==**
- Dva řetězce se shodují, pokud
 - mají stejnou délku a
 - všechny znaky stejné

DALŠÍ PŘÍKLADY

- Deklarujte proměnnou typu řetězec
 - `char str[] = "text";`
 - Může obsahovat libovolný text
- Zjistěte, zda zadaný řetězec obsahuje znak 'a'

- Deklarujte proměnnou typu řetězec
 - `char str[] = "text";`
 - Může obsahovat libovolný text
- Spočítejte, kolikrát se v řetězci změní znaky. Ignorujte velikost písmen.
 - `text` → 3x (`t` → `e`, `e` → `x`, `x` → `t`)
 - `aAb` → 1x (`A` → `b`)
- Tento příklad se objevil na vstupním pohovoru společnosti Apple.

- Vytvořte program `c`, který bude umět základní matematické výpočty (sčítání, odčítání, násobení, dělení, druhá odmocnina, ...).
- Výpočet se zadává na příkazové řádce. Každý operand nebo operátor bude samostatný argument programu (argumenty jsou oddělené mezerou). Program vypíše výsledek na standardní výstup.
- První varianta programu bude umět zpracovávat pouze jeden operátor, další může umět zpracovávat i výrazy s více operátory.
- Pro správné sestavení programu je nutné přidat přepínač překladači `-lm`. Tedy: `gcc -std=c11 -Wall -Wextra -Werror c.c -o c -lm`.

- Jako rozšíření můžete vyzkoušet načítat připravené výpočty ze souboru a hromadně vracet výsledky

- Zkrácený příklad

```
$ ./c 1 + 2
```

```
3
```

```
$ ./c 3 / 2
```

```
1.5
```

```
$ ./c 40 - -2
```

```
42
```

```
$ ./c sqrt 25
```

```
5
```

- V Elearningu je i ukázkový Makefile

- Zpracování argumentů programu
 - Jak vybrat argument?
 - Ve kterém argumentu bude uložena operace?
 - Jaký datový typ lze použít pro určení operace?
 - Jak převádět mezi řetězcem a celým číslem?
- Implementace jednotlivých operací
 - Jak určíme, která operace byla uživatelem vybrána?
- Doporučení:
 - Datový typ **float**
 - Funkce **printf**

Děkuji za pozornost