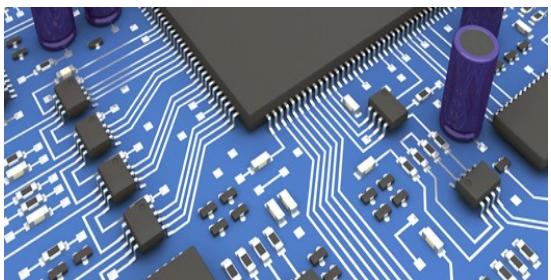


Aplikace Embedded systémů v Mechatronice



Michal Bastl

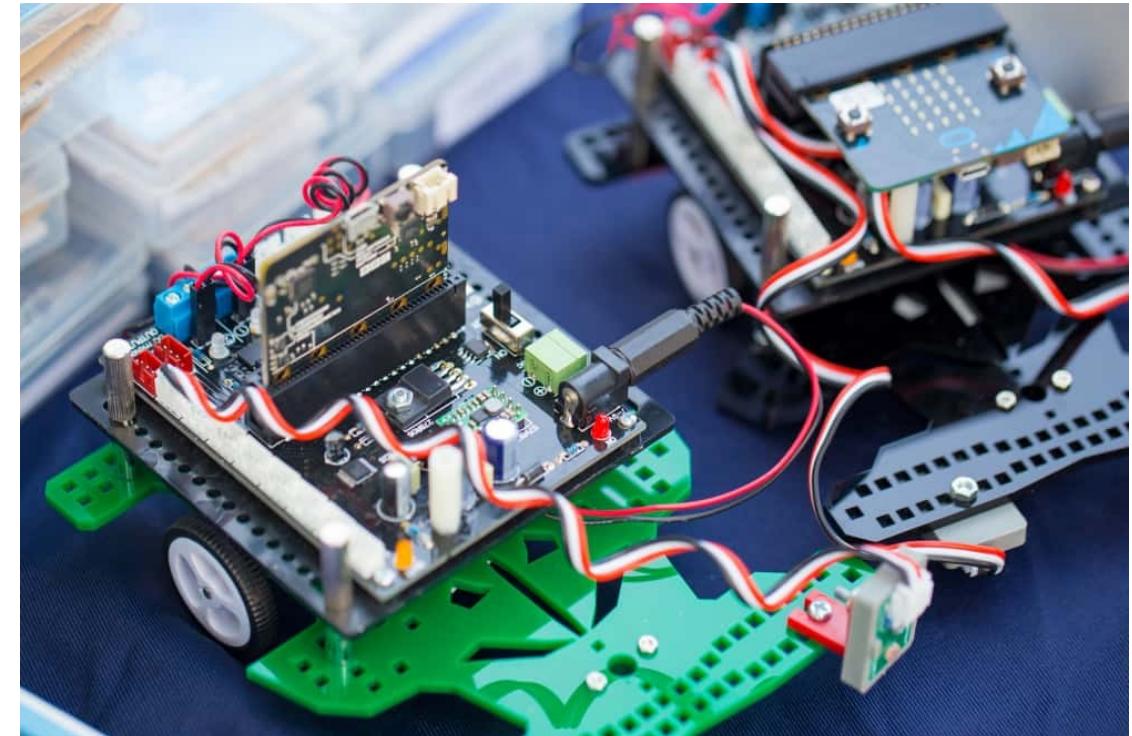
Organizace

Přednáška: Nepovinná, ale...

Cvičení: Povinné

Hodnocení předmětu 2026

- 2 domácí úlohy v průběhu semestru 2x5b
- Zápočtové testy 2x15b + 1x10b
- Semestrální projekt 50b
 - Projekt je open-ended (podobně jako 1PA)



zdroje: [github](#) BUT-FME-REV

Rozdíly mezi MATLABem a jazykem C:

C

- Kompilovaný jazyk
- Datové typy musí určit uživatel
- Přímý přístup do paměti
- málo klíčových slov
- Poměrně blízko hardwaru
- Oproti ASM je vzniklý kód čitelný (pokud byl programátor slušný : -)
- Je možné natropit chyby, které kompilátor nemůže ověřit
- Indexuje od 0

MATLAB

- Interpretovaný jazyk
- Datové typy určí interpreter
- Uživatel nemá přímý přístup do paměti
- Mnoho funkcí a klíčových slov
- Propast mezi hardwarem a programátorem je větší
- Je dobře čitelný, vývoj je rychlejší
- pracuje s pamětí sám. Nelze udělat zásadní chyby a jejich hledání je jednodušší
- Indexuje od 1

Číselné soustavy

Poziční:

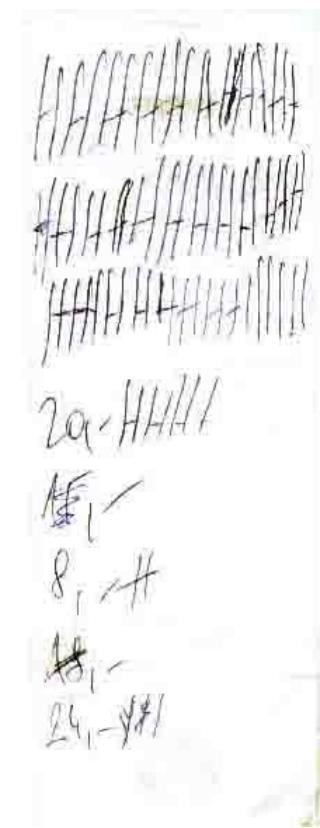
Klíčovou charakteristikou pozičních soustav je jejich základ. To je obvykle přirozené číslo větší než jedna. Váhy jednotlivých číslic jsou pak mocninami tohoto základu.

Tomuto odpovídá klasická desítková soustava.

Nepoziční

I II III IV V
VI VII VIII IX X
LCDM

I	II	III	III	III	III
1	2	3	4	5	6
			匚	ϙ	ϙ
7	8	9	10	100	1000
			 	 	
10,000	100,000	1,000,000			



Číselné soustavy

Dekadická:

c = 156

$$10^3 + 10^2 + 10^1 + 10^0$$

Binární:

c = 0b10011100

$$2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

Hexadecimální:

c = 0x9C

$$16^2 + 16^1 + 16^0$$

MCU používá pouze binární. Hexadecimální soustava se používá ke zkrácení zápisu.

Jeden znak Hexadecimální soustavy odpovídá 4 bitům.

Například:
BIN 0b11001000
HEX 0xC8

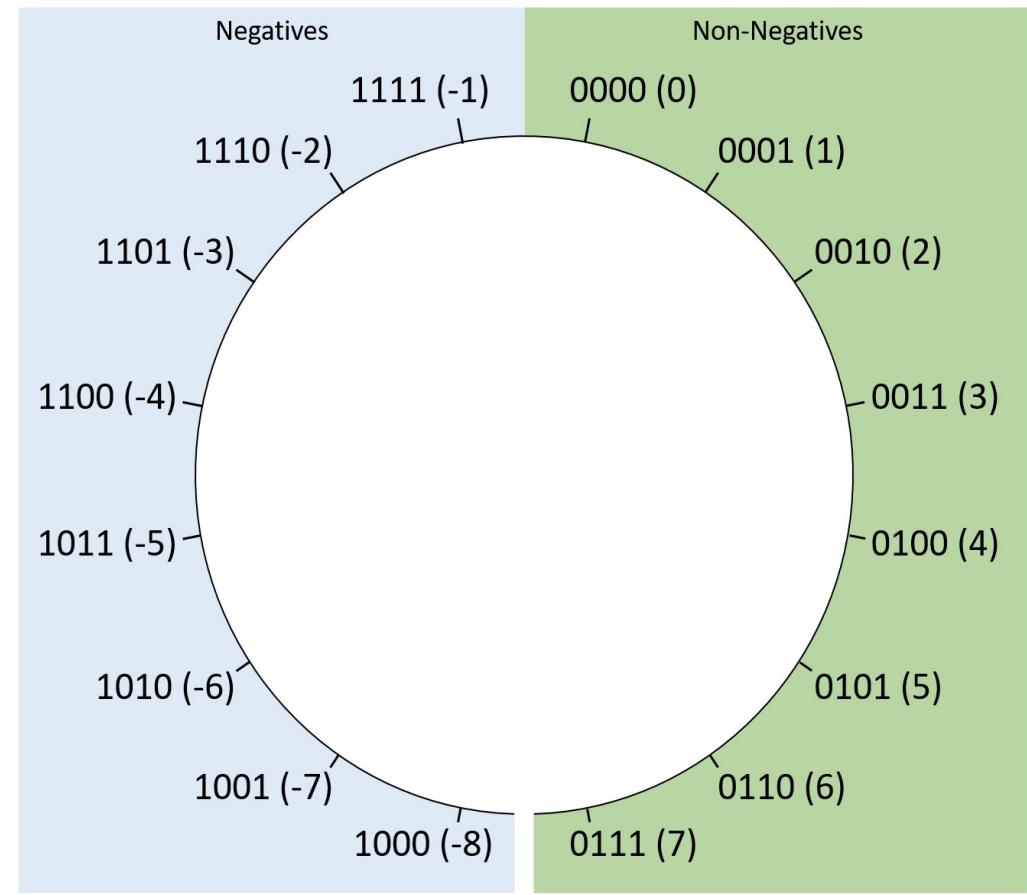
Ukázka

Reprezentace čísel a dvojkový doplněk

- Realizace znaménka je nejčastěji pomocí tzv. dvojkového doplňku.
- Nevýhodou je asymetričnost (rozsah není pro záporná a nezáporná čísla stejný)

$$\begin{array}{r} 000000000001010 \quad 10 \\ 111111111110101 \quad \text{flip all bits} \\ + 000000000000001 \quad \text{add 1} \\ \hline 111111111110110 \quad -10 \end{array}$$

$$\begin{array}{r} 111111111110110 \quad -10 \\ 000000000001001 \quad \text{flip all bits} \\ + 000000000000001 \quad \text{add 1} \\ \hline 000000000001010 \quad 10 \end{array}$$



Ukázka

Datové typy v jazyce C

Před použitím musím proměnnou deklarovat i s datovým typem.

Název proměnné nesmí začínat číslicí

Př:

```
char a, b, c;  
unsigned int i=0;  
long counter;
```

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Práce s proměnnými

- Programátor určuje datové typy, ale zároveň si musí hlídat jejich rozsah
- Proměnné je možné přetypovat (typ)
var
- Povšimněte si různých specifikátorů ve funkci printf() %d, %f, %li...

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

int main()
{
    int a=10;
    int b=3;

    float c, f;

    c = a/b;
    f = (float)a/b;

    printf("c=%f, f=%f\n", c, f);

    int16_t d = 32768;

    uint16_t e = (uint16_t)d;

    printf("d=%d, e=%d", d, e);
    return 0;
}
```

Ukázka

Rozsah datových typů je závislý na platformě

Knihovna stdint.h:

uint8_t, uint16_t, uint32_t
int8_t, int16_t, int32_t

Knihovna stdbool.h:

bool, true, false

Knihovna stdio.h

Obsahuje fci printf()

Funkce main() je entry point
(je v programu jen jednou)

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

int main(void)
{
    uint8_t a,b,c;
    uint16_t d=35000;

    a = 10;
    b = 100;
    c = a + b;

    printf("V c je %d, v d je %d", c,d);

    return(0);
}
```

operátory

Využití:

Rychlé násobení a dělení >> bitový posuv:

$A \ll n$ (násobení 2^n) ; $A \gg n$ (dělení 2^n)

Masky:

AND, OR a případně XOR

&		^
10011110	10011110	10011110
<u>00001111</u>	<u>00000001</u>	<u>00001111</u>
00001110	10011111	10010001

Symbol	Operator
&	bitwise AND
	bitwise inclusive OR
^	bitwise XOR (eXclusive OR)
<<	left shift
>>	right shift
~	bitwise NOT (one's complement) (unary)

operatory

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If $c = 5$ and $d = 2$ then, expression <code>((c==5) && (d>5))</code> equals to 0.
	Logical OR. True only if either one operand is true	If $c = 5$ and $d = 2$ then, expression <code>((c==5) (d>5))</code> equals to 1.
!	Logical NOT. True only if the operand is 0	If $c = 5$ then, expression <code>!(c==5)</code> equals to 0.

Operator	Meaning of Operator	Example
==	Equal to	<code>5 == 3</code> is evaluated to 0
>	Greater than	<code>5 > 3</code> is evaluated to 1
<	Less than	<code>5 < 3</code> is evaluated to 0
!=	Not equal to	<code>5 != 3</code> is evaluated to 1
>=	Greater than or equal to	<code>5 >= 3</code> is evaluated to 1
<=	Less than or equal to	<code>5 <= 3</code> is evaluated to 0

Operator	Example	Same as
=	<code>a = b</code>	<code>a = b</code>
+=	<code>a += b</code>	<code>a = a+b</code>
-=	<code>a -= b</code>	<code>a = a-b</code>
*=	<code>a *= b</code>	<code>a = a*b</code>
/=	<code>a /= b</code>	<code>a = a/b</code>
%=	<code>a %= b</code>	<code>a = a%b</code>



ASCII tabulka

- Zápis v jazyce C pomocí jednoduchých uvozovek '0'
- Používáme datový typ char 0..255
- V původním rozsahu má 128 znaků (rozšířená pak 255)
- Pro více znaků se používá pole char[x]

Příklad:

- Chci vypisovat číslice 0..9

```
char c = 2;
```

Knihovna stdio.h:

```
printf("cislice je %c", c + '0');
```

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

podminky if..else

//příklad podmínky
if...else

```
int a = 10;
```

```
if (a == 10){  
    //  
}  
else{  
    //  
}
```

//příklad podmínky
if...else

```
int a = 10;
```

```
if (a == 10){  
    //  
}  
else if(a > 10){  
    //  
}  
else{  
    //  
}
```

switch

- Program se řídí jednou proměnnou
- Pozor na ukončení příkazem break. Jinak by se vyhodnocovalo dále
- Default není nutně povinná
- Řídící proměnná nemůže být float nebo double

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

int main()
{
    char c;
    printf("Zadej znak:");
    c = getche();
    printf("\n");

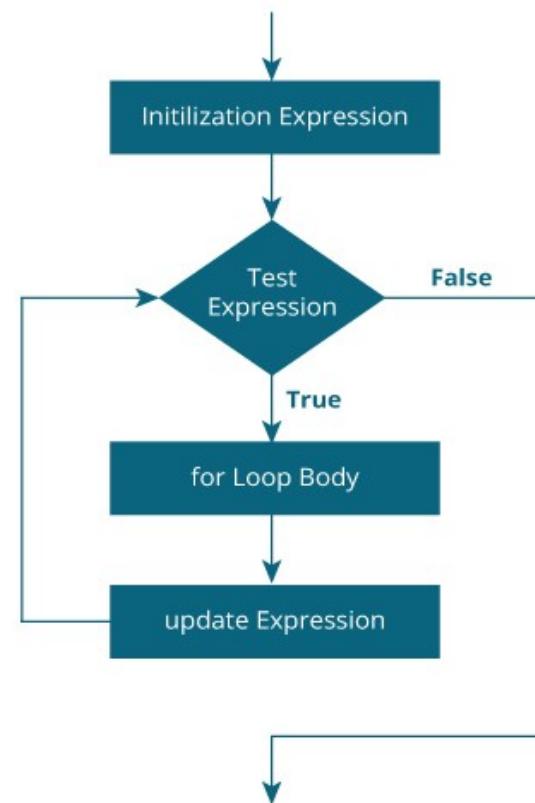
    switch (c)
    {
        case 'A':
            printf("Zadals A");
            break;
        case 'B':
            printf("Zadals B");
            break;
        case 'C':
            printf("Zadals C");
            break;
        default:
            break;
    }
    return 0;
}
```

for smyčka

//příklad for smyčky

```
int i;  
  
for (i=0; i < 10; i++){  
    printf( "Ahoj svete" );  
}
```

For smyčka se používá tehdy, kdy znám dopředu počet cyklů, které chci provést.

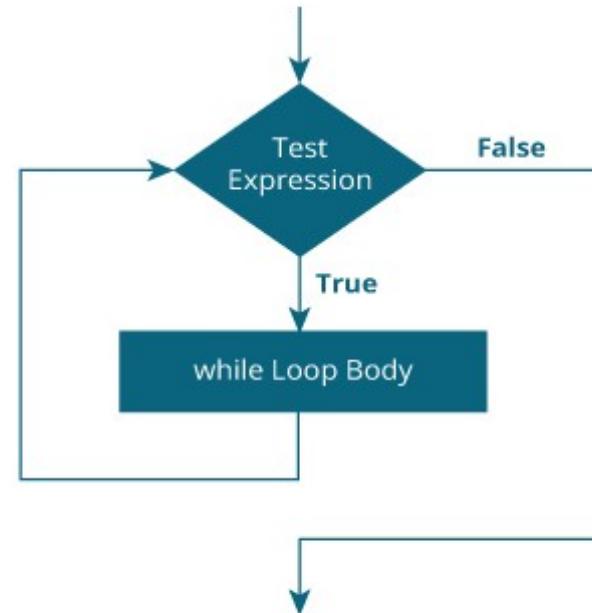


while smyčka

//příklad while
smyčky

```
int a = 10;  
  
while(a <= 100){  
    a = a * 10;  
}
```

While smyčka funguje jinak, jednoduše opakuje blok programu v jeho těle dokud platí podmínka



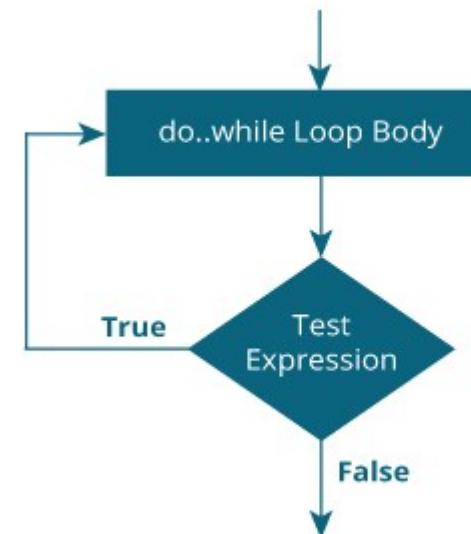
do-while smyčka

- Stejné jako while, jen se provede program a teprve potom se testuje podmínka
- Provede se tedy min. jednou

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

int main()
{
    do{
        printf("Provede se!");
    } while(0);

    return 0;
}
```



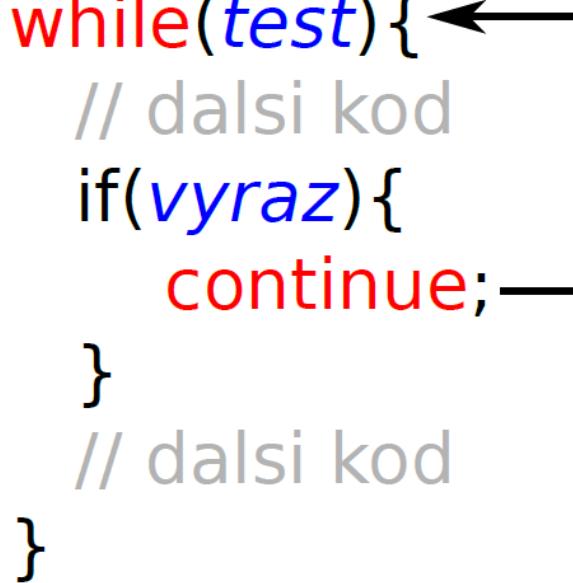
Ukázka

break, continue

```
while(test){  
    // dalsi kod  
    if(vyraz){  
        break; —  
    }  
    // dalsi kod  
}
```



```
while(test){  
    // dalsi kod  
    if(vyraz){  
        continue; —  
    }  
    // dalsi kod  
}
```



Funkce v C

- Funkce nemusí vracet hodnotu slovo void, jinak může vracet datové typy např. int, char apod... klíčové slovo return.
- Funkce může přijímat parametry (int a, char, b...) nebo žádné nemá (void)
- Překladač před použitím musí funkci „znát“, případně ví, že funkce někde existuje. Je definovaná .
- Používáme prototyp funkce před prvním použitím. Deklarujeme ji.

```
void putchar(char c);      // prototyp pozor na ;  
  
//definice funkce  
void putchar(char c){  
    bufferToSend = c;  
}
```

Čtyři verze funkce

Funkce nic nevrací a nepřijímá:

`void fun(void)`

Funkce nic nevrací a přijímá parametry:

`void fun(int a, int b)`

Funkce vrací a nepřijímá parametry:

`int fun(void)`

Funkce vrací a přijímá parametry:

`char fun(char a, int b)`

Funkce v C

Deklarace vs. Definice

```
//deklarace (prototyp)
int soucet(int a, int b);
//main
int main(void){
    int c;
    c = soucet(10, 5);
    return 0;
}
//definice
int soucet(int a, int b){
    return a + b;
}
```

```
//definice
int soucet(int a, int b){
    return a + b;
}
//main
int main(void){
    int c;
    c = soucet(10, 5);
    return 0;
}
```

Ukázka v C

```
#include <stdio.h>

// prototypy
void tisk(void);
void tisk_cisla(int cislo);

int main(void){
    char cislo;

    tisk();           // pouziti funkce
    printf("Zadej cislo:");
    cislo = getche() - '0';
    printf("\n");
    tisk_cisla(cislo); // pouziti funkce

    return 0;
}

// definice
void tisk_cisla(int a){
    printf("Cislo je %d\n", a);
}

// definice
void tisk(void){
    printf("Tisk z funkce\n");
}
```

```
#include <stdio.h>

// prototypy
int smysl_zivota(void);
int pricti_deset(int cislo);

int main(void){
    char cislo;

    printf("Smysl zivota: %d\n", smysl_zivota());
    printf("Zadej cislo:");
    cislo = getche() - '0';
    printf("\n");
    printf("Cislo +10 je: %d\n", pricti_deset(cislo));

    return 0;
}

// definice
int smysl_zivota(void){
    return 42;
}

// definice
int pricti_deset(int cislo){
    return cislo + 10;
}
```

Rekurze

- Rekurze znamená volání sebe sama v těle funkce
- Pomocí rekurze lze jednoduše vyřešit určité úlohy
- Některé problémy jsou z podstaty rekurzivní součet čísel od 1..n, faktoriál apod.
- Rekurze vede k efektivnímu zápisu, ale je náročnější na zdroje
- Stack overflow
- Pozor na podmínuku ukončení rekurze

```
#include <stdio.h>

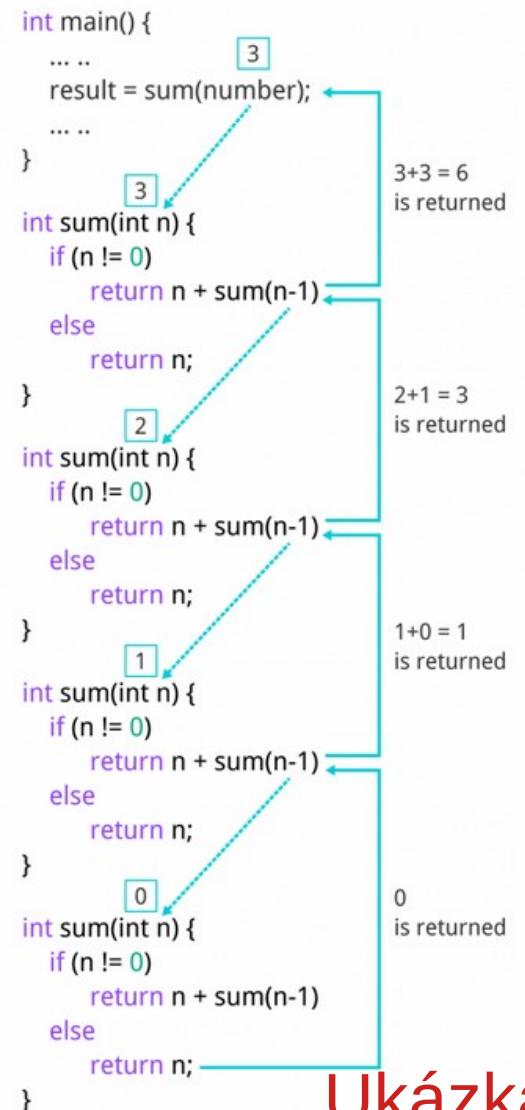
// prototypy
int sum(int n);

int main(void){
    char cislo;
    int suma;

    printf("Zadej cislo:");
    cislo = getche() - '0';
    printf("\n");
    suma = sum(cislo);           // pouziti funkce

    printf("Suma je: %d", suma);
    return 0;
}

int sum(int n){
    if(n != 0){
        return n + sum(n-1);
    }
    else{
        return 0;
    }
}
```



Ukázka

Platnost proměnných

- Proměnné v C mají určitou platnost
- Proměnné použité ve funkci existují pouze při volání funkce jsou lokální
- Při volání funkce se hodnota proměnné zkopíruje
- Globální proměnnou zavedu někde v globálním prostoru vně funkce main()
- Globální proměnné bych neměl nadužívat
- V příkladu je $a = 10$ pouze v těle funkce

```
#include <stdio.h>

//prototyp
void moje_funkce(int a);

int main(void){

    int a=0;

    moje_funkce(a);
    printf("a je: %d\n", a);

    return 0;
}

// definice
void moje_funkce(int a){
    a = a + 10;
    printf("a je: %d\n", a);
}
```

Platnost proměnných

- Proměnné v C mají určitou platnost
- Proměnné použité ve funkci existují pouze při volání funkce jsou lokální
- Při volání funkce se hodnota proměnné zkopiřuje
- Globální proměnnou zavedu někde v globálním prostoru vně funkce main()
- Globální proměnné bych neměl nadužívat
- V příkladu bude hodnota a = 10 jak ve funkci, tak v main

```
#include <stdio.h>

// globalní proměnná
int g_a;
//prototyp
void moje_funkce(void);

int main(void){

    moje_funkce();
    printf("a je: %d\n", g_a);

    return 0;
}

// definice
void moje_funkce(void){
    g_a += 10;
    printf("a je: %d\n", g_a);
}
```

Platnost proměnných

- Local scope a block scope

```
2 #include <stdio.h>
3
4 int main() {
5
6 {
7     int a;
8     a = 10;
9 }
10
11 printf("a = %d", a); // zde je a nedeklarován
12
13 return 0;
14 }
```

```
#include <stdio.h>

int main() {

    if (1){
        int a;
        a = 10;

        printf("a je: %d\n", a);
    }

    int a;
    a = 20;

    printf("a je: %d\n", a);

    return 0;
}
```

Globální/lokální/static

- V tomto případě proměnná sum přestane existovat po opoštění funkce
- Pokud chci proměnnou zachovávat musím použít klíčové slovo static
- Proměnná static je vlastně globální proměnná (existuje po celou dobu programu)
- Nemohu k ní však přistupovat z dalších funkcí

```
#include <stdio.h>

// prototyp
int suma(int a, int b, int c);

int main()
{
    int a=1, b=5, c=10;

    printf("Suma1 je: %d\n", suma(a, b, c));
    printf("Suma2 je: %d\n", suma(a, b, c));
    printf("Suma3 je: %d\n", suma(a, b, c));

    return 0;
}

int suma(int a, int b, int c){
    static int pocet=0;
    int sum;

    sum = a + b + c + pocet;
    pocet++;
    return sum;
}
```

Ukázka