Programování v jazyce C

Mikroprocesorová technika a embedded systémy

doc. Ing. Tomáš Frýza, Ph.D.

září 2018

Obsah přednášky

Obsah přednášky

Základy algoritmizace

Definice (Algoritmus)

Algoritmus je posloupnost operací, které řeší daný úkol v konečném počtu kroků. Jestliže se dílčí kroky algoritmu provádějí jeden po druhém, označujeme algoritmus jako sekvenční. Vykonávají-li se některé kroky algoritmu současně, hovoříme o algoritmu paralelním.

Požadované vlastnosti každého algoritmu:

správnost	výsledek algoritmu musí být správný
resultativnost	po konečném počtu kroků dospěje k řešení
konečnost	algoritmus se nezacyklí
determinovanost	v každém kroku je jednoznačně určen způsob pokračování práce algoritmu
obecnost	algoritmus lze použít pro řešení obecné úlohy, tj. že nepopisujeme jen jeden specifický případ
opakovatelnost	spouštíme-li tentýž algoritmus se stejnými daty, výsledek je vždy stejný

Způsoby zapsání algoritmů

- Způsoby zapsání algoritmů:
  - slovní/textové vyjádření
  - matematický zápis
  - grafické vyjádření
  - program (instrukce daného mikroprocesoru)

- Slovní vyjádření:

  Dílčí kroky algoritmu popisujeme větami v přirozeném jazyce. Např. recept: *...do XY g hladké mouky vlejte XY ml mléka...*

- Matematický zápis:

  Posloupnost matematických vyjádření. Jednotlivé kroky se zapisují v podobě tzv. pseudokódu, příp. PDL (Program Description Language). Jednoznačný a přehledný zápis dílčích kroků. Snadný přepis do programovacího jazyka a snadná modifikovatelnost.

```
1:  Read (a, b)
2:  If a or b is not a number:
3:      Print (Error)
4:      Exit
5:  Else:
6:  If a > b:
7:      x1 = a
8:      x2 = b
9:  Else:
10:     x1 = b
11:     x2 = a
12: Print (x1, x2)
```

Ukázka: Seřazení dvou zadaných čísel podle velikosti:

1	načtení dvou vstupních hodnot
2–4	test, zda se jedná o čísla a nikoliv o písmena
6–8	pokud je první číslo větší, ulož jeho hodnotu do proměnné x1, druhé pak do x2
9–11	v opačném případě ulož první číslo do x2 a druhé do x1
12	zobraz obě čísla od většího po menší

Grafické formy zápisu algoritmů: vývojový diagram

- Vývojový diagram:

  Symbolický, přehledný, algoritmický jazyk, který se používá pro názorné zobrazení algoritmu
  Tvoří ho obrazce/značky, do kterých se zapisují jednotlivé operace symbolickou formou. Tvary značek jsou dány normami
  Posloupnost značek je dána lomenými čarami. Vývojový diagram se čte od shora dolů
  Např. nástroj: yEd Graph Editor (`http://www.yworks.com/en/products/yfiles/yed/`)

- Hlavní komponenty vývojového diagramu:
  - start/konec algoritmu: první a poslední krok algoritmu
  - zpracování dat: transformace dat (např. početní operace)
  - podprogram: volání jiného algoritmu; po jeho dokončení se pokračuje původním algoritmem
  - vstup/výstup: naštení potřebných vstupních dat; uložení dat
  - rozvětvení programu: větvení základe podmínky. Je -li splněna, pokračuje se větví označenou + (někdy také: 1, TRUE), v opačném případě větví - (0, FALSE)
  - čárové spojnice značek

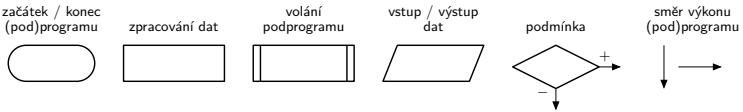

Figure: Hlavní značky používané ve vývojových diagramech

Programovací jazyk

Definice

Programovací jazyk je způsob zápisu algoritmů pro zpracování na počítači. Zápis konkrétního algoritmu ve zvoleném programovacím jazyce se nazývá program

- Programovací jazyk je soubor pravidel pro zápis algoritmů
- Základní dělení programovacích jazyků:
  - nižší programovací jazyky, kdy je nutné znát instrukční sadu procesoru a mnohdy i HW, na kterém bude program vykonáván
  - vyšší programovací jazyky s vyšším stupněm abstrakce; jsou překládány kompilátorem (C, ...), nebo spouštěny interpretem (PHP, Python, ...)

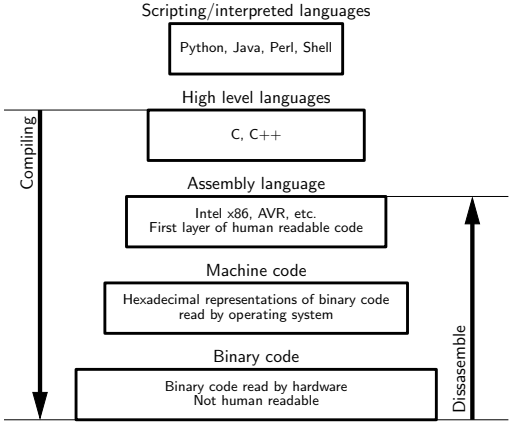

Figure: Programovací vrstvy

Vyšší programovací jazyky

- Vyšší programovací jazyky (high-level programming languages) jsou podstatně srozumitelnější, protože zavádějí větší stupeň abstrakce a nejsou závislé na konkrétním procesoru
- Příkazy jsou tvořeny pomocí klíčových slov, jejichž význam musí být převeden do strojového kódu
- Podle způsobu překladu se programovací jazyky dělí na **kompilované** nebo **interpretované**

Kompilované

- Celý kód je nejprve přeložen a až potom může být spuštěn
- Běh takovéto aplikace je rychlejší, protože kompilátor (překladač, anglicky compiler) předem převádí zdrojový textový kód na spustitelný strojový kód
- Např. C, ...

Interpretované

- Kód se nepřekládá dopředu, ale překlad se řeší až při požadavku na spuštění
- Vykonávání takového programu je tedy pomalejší
- Výhodou je ale nezávislost aplikace na platformě i hardwaru
- Např. Basic, PHP, Python, Perl, ...

Další způsoby programování

- Strukturované programování:
  - sekvenční provádění instrukcí a procedur
  - podstatou je rozdělíme algoritmů na dílčí úkoly interpretované s využitím strukturovaných datových typů a řídicích struktur jako jsou posloupnosti příkazů, větvení, cykly
  - např. jazyk C, Pascal

- Objektové programování:
  - přínosem je větší strukturovanost a modularita vytvářeného programu
  - založeno na objektech (data + metody) a vztazích mezi nimi
  - objektově orientovaný přístup tvorby programu je charakterizován třemi základními vlastnostmi:

  (1) obalení (zapouzdření): realizováno novým datovým typem objekt; obalením můžete rozumět obalení datových položek řídícími strukturami (metodami), které zajišťují přístup k datovým položkám

  (2) dědičnost (inheritance): možnost vytvářet nové objekty jako potomky již existujících objektů; přebírat od nich datové položky a metody, modifikovat je či upřesňovat

  (3) polymorfismus (mnohotvarost)možnost pojmenovat metodu jedním jménem a tato metoda může být společná pro různé objekty ve stromové hierarchii, i když pro každý objekt v této hierarchii se bude chovat různě

- Vizuální programování:
  - Většinou objektové a současně strukturované programování
  - Velké množství často používaných objektů (tlačítka, dialogová okna, práce se soubory) je již připraveno a myší je přetahujeme do vytvářeného programu

Základní pojmy programování

- **Syntaxe** příkazu popisuje, jak tento příkaz bezchybně použít
- **Sémantika** popisuje význam příkazů
- **Proměnná** (variable) je objekt, který má pevně stanovené označení a nese určitou hodnotu, která se může v průběhu programu měnit
- Konstanta (constant) je pojmenovaný objekt určité hodnoty, která je v celém programu neměnná
- Přiřazení hodnoty proměnné je proces "naplnění" proměnné výrazem
- Výrazy mohou obsahovat konstanty, proměnné, aritmetické operátory (např. `+`, `-`, `*`, `/`), logické operátory (např. `&`, `|`, `^`), kulaté závorky, ...

- Řízení běhu programu pomocí podmínek a cyklů
- **Podmínka** je logický výraz, jehož hodnotou je pravda nebo nepravda (podmínka platí nebo neplatí; je splněna, nebo není splněna)
- **Cyklus** provádí část kódu opakovaně zpravidla s jinými hodnotami proměnných. Počet opakování je řízen podmínkou, příp. v kombinaci s čítacím indexem
- Funkce/podprogramy jsou používány k tvorbě přehlednějšího strukturovaného kódu. Představují uzavřené celky kódu k danému účelu. Je umožněno opakované volání s různými parametry

Hlavní zásady při psaní programů

- Kód musí být srozumitelný nejen pro autora!
- **Coding rules**: striktní pravidla programovacího jazyka (syntaxe, příkazy, ...)
- **Coding guidelines**: pravidla srozumitelná pro všechny programátory (tým, firma, ...)
- Některá Coding guidelines:
  - Každý zdrojový soubor začíná popisnou "hlavičkou"
  - Odsazujte kód i komentáře (s ohledem na odlišné textové editory, používejte mezery místo tabulátorů)
  - Počet znaků na řádek pod cca 80 znaky. Mnohem jednodušší pro čtení
  - Používejte prázdné řádky pro tvorbu "čitelnějšího" kódu
  - Používejte popisné názvy proměnných, funkcí, ...
  - Všechny názvy ideálně anglicky. Používejte jen písmena A, B, ..., Z, a, ..., z, číslice 0, ..., 9 a podtržítko "_"

```
/*********************************************
* main00_sol.c
*
* Tomas Fryza, Brno University of Technology
* Date/Time updated: Fri Oct 21 16:01:55 2016
*
*   Target MCU : ATmega16
*   Description: Binary counter with delay
*/

#include <avr/io.h>      // definition file
#define F_CPU 1000000UL // clock frequency
#include <util/delay.h> // delay library

/*********************************************
* Main function
*/
int main(void)
{
    * setup I/O port */
    DDRB  = 0xff;        // set output direction
    PORTB = 255;         // turn off all LEDs

    * forever loop */
    while (1) {
        _delay_ms(50);  // wait for 50 ms
        PORTB = PORTB-1;// change binary counter
    }

    return 0;
}
```

Ukázka programu v jazyce C

```
/**
*********************************************************************
* @file main03_sol.c
*
* @brief  Control of TWI bus
* @author Tomas Fryza
* @date   Thu Nov 17 19:15:55 CET 2016
*
* Counter on TWI bus expander controlled by 16-bit timer interrupt
*/

#include <avr/io.h>             // definition file for MCU ATmega16
#include "twi_h.h"              // TWI library

char temp = 64;                 // counter value

int main(void)
{
    twi_init();                 // initialization of TWI bus
                                // prescaler = 64 (260 ms)
    TCCR1B = TCCR1B | ((0<<CS12)+(1<<CS11)+(1<<CS10));
    TIMSK  = TIMSK | (1<<TOIE1);// enable overflow interruption
    sei();                      // enable all interrupts

    while (1);
    return 0;
}
```

- blokový komentář popisující obsah zdrojového souboru
- příkazy (direktivy) pro průběh kompilace
- deklarace proměnné `temp`
- hlavní funkce programu `main()`
- volání uživatelské funkce `twi_init()`
- příkazy jazyka C

Vývoj jazyka C

- Jazyk C byl vytvořen v Bellových laboratořích, USA; autoři Dennis MacAlistair Ritchie & Brian Kernighan

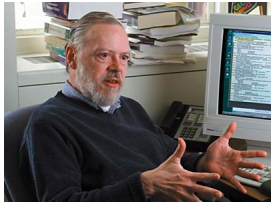

Figure: Dennis MacAlistair Ritchie (⋆9. 9. 1941, †12. 10. 2011)

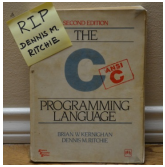

Figure: D.M. Ritchie, B. Kernighan. *The C Programming Language*. 1978

1978 První publikování učebnice jazyka C; jazyk vytvořen již dříve

C89 V roce 1989 byl programovací jazyk C byl poprvé standardizován americkou společností ANSI (American National Standards Institute); označení C89, též ANSI C

- ANSI C definuje strukturu/syntaxi jazyka i standardní knihovny

C90 Standard byl v roce 1990 převzat také mezinárodní společností ISO (International Organization for Standardization); označení C90

C99 V roce 1994 započala práce na revizi standardu, která vyústila ve standard C99

- Aktuální verze: ISO/IEC 9899:2018, cca 500 stran, 200,-CHF, viz `https://www.iso.org/standard/74528.html`

Proces tvorby spustitelného programu

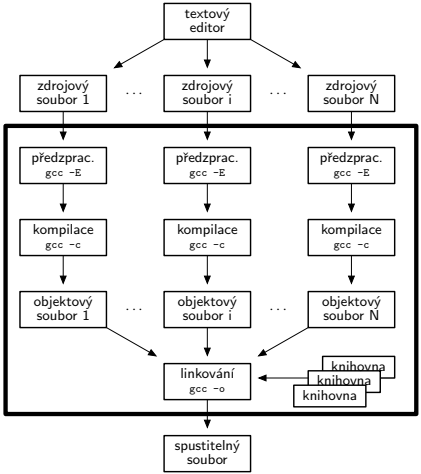

Figure: Postup překladu aplikace s více moduly v jazyce C

- Aplikace v C se skládá z modulů, které obsahují dva textové soubory: zdrojový `*.c` (anglicky source file) a tzv. hlavičkový `*.h` (header file)
- Hlavičkový soubor typicky obsahuje jiné vkládané hlavičkové soubory, definice uživatelských funkcí a konstant. U jednoduchých aplikací lze hlavičkový soubor vynechat a jeho obsah přímo vepsat do zdrojového souboru `*.c`
- Preprocesor: vynechání komentářů, vykonají se direktivy pro překladač (typ. vloží se přidružené soubory, zpracují se makra, ...). Výsledkem je textový (zdrojový) soubor
- **Kompilátor**: samotný překlad textových souborů do tzv. Object code. Jedná se o relativní kód (absolutní adresy proměnných a funkcí nemusí být ještě známy). Pro každý programový modul jeden objektový soubor
- **Linker**: Spojení všech objektových souborů (včetně knihoven) do jednoho spustitelného kódu. Relativní adresy jsou nahrazeny absolutními

Obsah přednášky

Základy syntaxe jazyka C

- Jazyk C je *free-form* formát, tj. zdrojový kód lze psát mnoha způsoby
- Standard jazyka C definuje určitá slova jako "klíčová" (též vyhrazená, rezervovaná). Ty mají v programovacím jazyce speciální význam a nelze je používat jako jména funkcí, proměnných, konstant, ...
- Některé klíčová slova jazyka C ve verzi ANSI C:

```
break case char else float for if int return
signed sizeof switch unsigned void while
```

- Jazyk C rozlišuje malá a velká písmena, je tzv. case sensitive. Rozlišuje např. `temp` $\neq$ `Temp` $\neq$ `TEMP`
- Příkazy jsou ukončeny středníkem

```
char temp = 10;
printf("Result is %d", temp);
```

- Komentáře jsou pasáže kódu, které kompilátor nezpracovává. Komentovat lze tzv. "do bloku" mezi symboly "/*" a "*/" a u většiny kompilátorů také "do řádku" za symboly "//"

```
/*
Tato část kódu slouží k slovnímu popisu
chodu funkce, nebo její části
*/
temp = a << 2;       // toto už je komentář
```

- Dílčí bloky (funkce, cykly, těla podmínek, ...) jsou uzavřeny do složených závorek "{" a "}"

```
int main(void)
{                    // začátek funkce main
    ...
}                    // konec funkce main
```

- Program v jazyce C je členěn do funkcí. Vždy je funkce `main()`, která se spouští jako první

Obsah přednášky

Deklarace proměnných

- Proměnná je pojmenované místo v paměti, kde lze uchovat jednu hodnotu určité velikosti
- Každá proměnná je určitého (předem definovaného) typu
- Obecná deklarace (vytvoření) proměnné:

```
typ identifikátor;
typ identifikátor = hodnota;     // již během deklarace lze proměnné přiřadit hodnotu
```

- Ukázky deklarací proměnných pomocí standardních typů jazyka C:

```
char temp;                       // 8bitové celé číslo
char real = 100;
int  result;                     // "větší" celé číslo
float ratio = 2.25;              // reálné číslo
```

- Identifikátor proměnné (též jméno, název proměnné) je jedinečný v rámci daného bloku/programu
- Podle dostupnosti rozlišujeme proměnné na globální a lokální

Deklarace proměnných (pokrač.)

Globální proměnné

- Takovou proměnnou deklarujeme mimo tělo jakékoliv funkce; zpravidla na začátku zdrojového souboru
- Hodnota proměnné je tak dostupná pro všechny funkce programu

Lokální proměnné

- Proměnná je deklarována na začátku funkce a její hodnota je dostupná pouze uvnitř této funkce
- Proměnná je alokována v paměti při spuštění funkce
- Po opuštění funkce, proměnná ztrácí svou hodnotu (uvolní paměť)

- Identifikátor je tvořen písmeny, číslicemi a podtržítkem. Nepoužívat háčky a čárky, příp. jinou diakritiku. Identifikátor začíná písmenem nebo podtržítkem
- Používejte smysluplné identifikátory!
- Ukázky použití identifikátorů:

```
/* srozumitelné */
volume = 2 * pi * radius * high;

/* nesrozumitelné */
r5 = r0 * r1 * r1 * r2;
```

- Hlavní styly jak pojmenovávat *delší* identifikátory: dělit slova pomocí podtržítka "_", nebo spojovat slova s prvním velkým písmenem:

```
/* underscore style */
char test_value_id;

/* camel style */
unsigned int testValueId = 5;
```

Vybrané datové typy proměnných v jazyce C

`char` Celočíselný typ; velikost alokované paměti je 1 byte

`int` Celočíselný typ; velikost alokované paměti závisí na procesoru

`long` Celočíselný typ; 32 bitů

`float` Desetinná čísla ve formátu s plovoucí řádovou čárkou; velikost 32 (`float`) a 64 bitů (`double`)

- Pozn.: Na typ `int`, `long` a `double` lze dále aplikovat kvalifikátor `long`. Při deklaraci se umístí před samotný typ
- Pozn.: Na typ `int` lze naopak aplikovat kvalifikátor `short`

- Specifické pro jazyk C je, že velikost úplně všech datových typů není standardem pevně definována. Nicméně, vždy platí, že:

  `sizeof(char)` = 1 (tj. 1 byte)
  `sizeof(char)` $\leq$ `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)` $\leq$ `sizeof(long long)`

  - Pozn.: Výraz `sizeof()` vypočte počet bytů (nikoliv bitů) datového typu nebo proměnné, tj. velikost alokované paměti

- Velikost typu závisí na procesoru. Např. na 16bitovém procesoru bude mít `int` velikosti 16 bitů, na 64bitovém procesoru může mít i 64 bitů

- Pozn.: Konkrétní rozsahy datových typů lze najít ve standardní knihovně `limits.h`

Velikost alokované paměti pro vybrané datové typy

- Výpis zdrojového souboru `main.c`:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    printf("%d ", sizeof(char));
    printf("char\n");

    printf("%d ", sizeof(short int));
    printf("short int\n");

    printf("%d ", sizeof(int));
    printf("int\n");

    printf("%d ", sizeof(long int));
    printf("long int\n");

    printf("%d ", sizeof(float));
    printf("float\n");

    /* end of main function */
    return 0;
}
```

- Na pozici dvojznaku `%d` vypíše funkce `printf` celočíselnou (desítkovou) hodnotu uvedenou za řetězcem
- Formátovací znak `\n` zajistí vložení nového řádku
- Výpis spuštěné aplikace:

```
1 char
2 short int
4 int
8 long int
4 float
```

Vybrané datové typy proměnných (pokrač.)

- Na celočíselné typy (`char`, `short int`, `int` a `long`) lze aplikovat modifikátory `signed` a `unsigned`. Ty se opět uvádí při deklaraci před samotný typ proměnné

`unsigned`: Proměnná obsahuje pouze nezáporná celá čísla

`signed`: Proměnná obsahuje jak kladná tak i záporná celá čísla; toto je defaultní hodnota a není třeba ji při deklaraci uvádět

```c
#include <stdio.h>

int main()
{
    char a = 200;
    unsigned char b = 200;
    signed short int c = -33000;

    printf("char a = 200: ");
    printf("%d\n", a);

    printf("unsigned char b = 200: ");
    printf("%d\n", b);

    printf("signed short int c = -33000: ");
    printf("%d\n", c);

    return 0;
}
```

- Víme, že `char` je vždy 8bitový, proto `signed char` reprezentuje hodnoty od -128 do 127, avšak rozsah `unsigned char` je od 0 do 255
- Kompilátor nás během překladu upozorní na přetečení přiřazovaných hodnot, nicméně překlad dokončí:

```
char a = 200: -56
unsigned char b = 200: 200
signed short int c = -33000: 32536
```

Použití celočíselných datových typů dle knihovny `stdint.h`

```c
/* standard input/output library */
#include <stdio.h>
/* standard integer library */
#include <stdint.h>

/* main function */
int main()
{
    int8_t  a = 200;
    char    b = 200;
    uint8_t c = 200;
    unsigned char d = 200;

    printf("int8_t je ekvivalent char\n");
    printf("%d, %d\n\n", a,b);

    printf("uint8_t je ekvivalent unsigned char\n");
    printf("%d, %d\n", c,d);

    /* end of main function */
    return 0;
}
```

- Standard C99 definuje v knihovně `stdint.h` také jiné, výstižnější označení celočíselných typů:

```
int8_t int16_t int32_t uint8_t
uint16_t uint32_t
```

- Výstup aplikace:

```
int8_t je ekvivalent char
-56, -56

uint8_t je ekvivalent unsigned char
200, 200
```

Aritmetické a bitové operace

Table: Aritmetické operace v jazyce C

Operace	Operand
Násobení	*
Dělení	/
Zbytek po celočíselném dělení	%
Sčítání	+
Odečítání	-
Inkrementace	++
Dekrementace	--

- Možný zkrácený zápis operací, kdy proměnná je současně argumentem a je do ni přiřazena i hodnota výsledku:

```
a += 3;      // a = a+3;
b -= 2;      // b = b-2;
c *= 5;      // c = c*5;
d /= a;      // d = d/a;
```

Table: Bitové operace v jazyce C

Operace	Operand
Jednotkový doplněk (negace bitů)	$\sim$
Bitový posuv doleva	<<
Bitový posuv doprava	>>
Logický součin AND	&
Logický součet OR	|
Exkluzivní součet EX-OR	$\wedge$

- Ukázka možných zkrácených zápisů operací:

```
a |= 3;      // a = a|3;
b &= 2;      // b = b&2;
c ^= 5;      // c = c^5;
d <<=2;      // d = d<<2;
```

Obsah přednášky

Výstup na obrazovku

- Textový výstup na obrazovku se typicky provádí pomocí funkce `printf()` z knihovny `stdio.h`. Obecná syntaxe příkazu je:

```
printf("vystupni retezec", hodnota1, hodnota2, ...);    // vystupni retezec obsahuje formatovaci znacky
```

```
/* standard input/output library */
#include <stdio.h>
/* standard integer library */
#include <stdint.h>

/* main function */
int main()
{
    uint8_t a = 20, b = 30;
    uint8_t c;

    c = a+b;

    printf("soucet = 50\n");
    printf("soucet = %d\n", 50);
    printf("soucet = %d\n", c);
    printf("soucet = %d\n", a+b);
    printf("%d+%d  = %d\n", a,b,c);
    printf("soucet = %d (dek) a 0x%x (hexa)\n", c,c);

    /* end of main function */
    return 0;
}
```

- Různé způsoby výpisů hodnot:

```
soucet = 50
soucet = 50
soucet = 50
soucet = 50
20+30  = 50
soucet = 50 (dek) a 0x32 (hexa)
```

Formátovací značky výstupního řetězce a zobrazení speciálních znaků

- Speciální formátovací značky určují formu vložení hodnot do výstupního řetězce funkce `printf()`
- Nejpoužívanější formátovací značky jsou:
  - `%d` Celé číslo v desítkové soustavě
  - `%x` Celé číslo v šestnáctkové soustavě (malá písmena `a`, `b`, `c`, `d`, `e`, `f`)
  - `%X` Celé číslo v šestnáctkové soustavě (velká písmena `A`, `B`, `C`, `D`, `E`, `F`)
  - `%f` Zobrazení desetinného čísla typu `float` nebo `double`
  - `%c` Zobrazení nebo načtení jednoho znaku
  - `%s` Zobrazení textového řetězce
  - `%%` Zobrazení znaku procenta
- Tzv. escape sekvence jsou příkazy, které umožňují vložit do výstupního řetězce `printf()` zvláštní znaky
- Nejpoužívanější znaky jsou:
  - `\n` Přesun kurzoru na začátek nového řádku
  - `\r` Přesun na začátek aktuálního řádku
  - `\t` Přesun na následující tabelační pozici
  - `\'` Zobrazení apostrofu
  - `\"` Zobrazení uvozovek
  - `\b` Přesun kurzoru o jednu pozici doleva

Formátovací značky výstupního řetězce a zobrazení speciálních znaků (pokrač.)

- Výpis zdrojového souboru `main.c`:

```c
/* standard input/output library */
#include <stdio.h>
/* standard integer library */
#include <stdint.h>

/* main function */
int main()
{
    uint8_t a = 65, b = 107;
    float c = 3.14;

    printf("%%d:\t a = %d\t b = %d\n", a,b);
    printf("%%x:\t a = %x\t b = %x\n", a,b);
    printf("%%X:\t a = %X\t b = %X\n", a,b);
    printf("%%c:\t a = %c\t b = %c\n\n", a,b);

    printf("%%f:\t c = %f\n", c);
    printf("%%5.2f:\t c = %5.2f\n\n", c);

    printf("%%s:\t %s\n", "Hello th**\b\bere!");

    /* end of main function */
    return 0;
}
```

- Výstup aplikace:

```
%d:     a = 65   b = 107
%x:     a = 41   b = 6b
%X:     a = 41   b = 6B
%c:     a = A    b = k

%f:     c = 3.140000
%5.2f:  c =  3.14

%s:     Hello there!
```

Obsah přednášky

Podmínky v jazyce C: `if`

- Podmínka umožňuje větvit chod programu do dvou směrů a to na základě pravdivosti logického výrazu, který je argumentem podmínky
- Podmínky v jazyce C prostřednictvím příkazu `if`, příp. `switch`

```
if (a == N) {    // je-li a = N, vykonej tento
    ...          // kód
}
else if (a > N) { // je-li a různé od N
    ...          // a současně a > N, pak vykonej
                 // tuto část kódu
}
else {           // pokud nebyla splněna žádná
    ...          // z podmínek, vykonej tento kód
}
```

- Je-li a rovno `N`, pak vykonej část kódu ve složených závorkách. Pozor, při testování rovnosti jsou DVĚ rovnítka. Podmíněný kód je uzavřen do **složených závorek**
- Klíčové slovo `else` označuje "v opačném případě", tj. pokud předchozí podmínka nebyla splněna. Lze kombinovat s další podmínkou `if`. Kombinace `else if` může být použita libovolněkrát
- Klíčové slovo `else` bez další podmínky doplňuje poslední podmínku `if` a smí být použito jen jednou

```
if (b <= 10) {   // je-li b menší nebo rovno 10
    ...          // vykonej tento kód
}
```

- Podmínka nemusí nutně obsahovat konstrukci `if`, `else if` a `else`. Obecně stačí jen `if`. Nejčastěji se však používá kombinace `if` a `else`

Podmínky v jazyce C: `if` (pokrač.)

- Podmínky lze kombinovat pomocí relací AND (a současně) a OR (nebo). V jazyce C se tyto operátory zapisují dvojitým znakem && a ||

```
if (a == 10 && b <= 5) {
    ...             // vykonej tento kód, pokud se
                    // a rovná 10 A SOUČASNĚ je b
}                   // maximálně rovno 5
else if (a > 15 || b > 20) {
    ...             // vykonej tento kód, pokud je
                    // a větší než 15 NEBO pokud je
}                   // b větší než 20
```

Table: Relační operandy v jazyce C

Operace	Operand
Rovno	==
Je různý od	!=
Menší než	<
Menší nebo rovno	<=
Větší než	>
Větší nebo rovno	>=
Logické AND	&&
Logické OR	||

Ukázka podmínky v jazyce C: `if`

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int a, b;

    printf("Zadej dve cela cisla:\n");
    scanf("%d %d", &a, &b);

    if (a == b) {
        printf("Hodnoty se rovnaji: %d\n", a);
    }
    else if (a < b) {
        printf("%d < %d\n", a, b);
        printf("a = %d\nb = %d\n", a, b);
    }
    else {
        printf("%d > %d\n", a, b);
        printf("a = %d\nb = %d\n", a, b);
    }

    /* end of main function */
    return 0;
}
```

- Výpis spuštěné aplikace:

```
Zadej dve cela cisla:
3
7
3 < 7
a = 3
b = 7
```

Podmínky v jazyce C: `switch`

- Příkaz `switch` umožňuje testovat rovnost proměnné prostřednictvím seznamu konkrétních případů/hodnot case
- Obecná syntaxe podmínky `switch`:

```
switch (proměnná) {
    case hodnota1:
        ...      // je-li proměnná == hodnota1
        break;   // ukončení switch

    case hodnota2:
        ...      // je-li proměnná == hodnota2
        break;   // ukončení switch

    /* lze použít libovolný počet case, každý
       je ale ukončen příkazem break */

    default:
        ...      // vykoná se pokud není
                 // splněna žádná z uvedených
                 // podmínek
}
```

- Počet testovaných případů case není omezen. Za příkazem case vždy následuje testovaná hodnota a dvojtečka
- Pokud je proměnná rovna testované hodnotě, vykonají se všechny následující přiřazení až po příkaz `break`
- Pozor, pokud není uveden `break` vykonají se i následující přiřazení, což bývá častá chyba!
- Dobrou programátorskou praxí je uvádět příkaz `default` pro specifikování přiřazení, která se mají vykonat v případě, že proměnná není rovna ani jedné z testovacích hodnot. Část `default` se vždy uvádí jako poslední a neobsahuje tak `break`

Ukázka podmínky v jazyce C: `switch`

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    char znamka = 'B';

    switch (znamka) {
        case 'A':
            printf("Vyborne!\n");
            break;
        case 'B':
        case 'C':
            printf("Dobre\n");
            break;
        case 'D':
        case 'E':
            printf("Prospel\n");
            break;
        default :
            printf("Neprospel\n");
    }
    printf("Hodnoceni: %c\n", znamka);

    /* end of main function */
    return 0;
}
```

- Výpis spuštěné aplikace:

```
Dobre
Hodnoceni: B
```

Cyklus v jazyce C: `for`

- Příkaz `for` umožňuje opakovat tělo cyklu určitý počtekrát
- Obecná syntaxe cyklu `for` v jazyce C:

```
for (počátek; podmínka; změna) {
    ...          // tělo cyklu
}
```

```
for (i = 0; i < 8; i++) {
    ...          // opakuj 8x
                 // i = 0, 1, 2, ..., 6, 7
}
```

```
for (i = 80; i >= 10; i = i - 10) {
    ...          // opakuj 8x
                 // i = 80, 70, ..., 20, 10
}
```

- Úvodní přiřazení (`počátek`) se vykoná pouze jedenkrát. Umožňuje deklarovat a inicializovat čítací index
- Následně se testuje druhá část (`podmínka`). Pokud je pravdivá, vykoná se tělo cyklu. Pokud je podmínka nepravdivá, tělo cyklu se nevykoná a běh programu pokračuje příkazy následující za `for`
- Po vykonání těla cyklu se chod programu vrátí na část `změna` a tu provede. Umožňuje to aktualizovat stav čítacího indexu. Následuje opět vyhodnocení `podmínky`
- Pozn.: Z cyklu lze kdykoliv vyskočit příkazem `break`. Příkazem `continue` ihned skočíme na testování následující podmínky cyklu

Ukázka cyklu v jazyce C: `for`

- Zdrojový kód aplikace:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int i;

    for (i = 10; i < 20; i = i + 2 ) {
        printf("hodnota indexu: %d\n", i);
    }

    /* end of main function */
    return 0;
}
```

- Výpis spuštěné aplikace:

```
hodnota indexu: 10
hodnota indexu: 12
hodnota indexu: 14
hodnota indexu: 16
hodnota indexu: 18
```

Cyklus v jazyce C: `while`

- Příkaz `while` umožňuje vykonávat přiřazení (nebo skupinu přiřazení) dokud je splněna daná podmínka
- Tělo cyklu je spuštěno/vykonáno nulakrát nebo vícekrát, pokud je podmínka pravdivá
- Obecná syntaxe cyklu `while` v jazyce C:

```
while (podmínka) {
    ...          // tělo cyklu
}
```

- Podmínkou cyklu může být výraz, nebo hodnota. Splněnou (pravdivou) podmínku v jazyce C představuje libovolná nenulová hodnota
- Tělo cyklu se vykonává, dokud je podmínka pravdivá. Pokud je podmínka nepravdivá, tělo cyklu se nevykoná a běh programu pokračuje příkazy následující za `while`

Ukázka cyklu v jazyce C: `while`

- Zdrojový kód aplikace:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int i = 10;

    printf("zacatek\n");
    while (i < 20) {
        printf("hodnota promenne: %d\n", i);
        i++;
    }
    printf("konec\n");

    /* end of main function */
    return 0;
}
```

- Výpis spuštěné aplikace:

```
zacatek
hodnota promenne: 10
hodnota promenne: 11
hodnota promenne: 12
hodnota promenne: 13
hodnota promenne: 14
hodnota promenne: 15
hodnota promenne: 16
hodnota promenne: 17
hodnota promenne: 18
hodnota promenne: 19
konec
```

Pozn.: Nekonečný cyklus, kdy podmínka opakování je vždy splněna

```c
for (;;) {
    // tělo cyklu
}
```

```c
while (1) {
    // tělo cyklu
}
```

- Všechny tři výrazy v cyklu `for` jsou nepovinné. Není zde žádná podmínka, ani čítací index, který by opakování těla cyklu omezoval
- Tělo cyklu `while` se spouští/vykonává, dokud je hodnota v závorce různá od nuly, proto se zde spouští neustále

Obsah přednášky

Jednorozměrná pole, vektory: deklarace

- Pole je datová struktura, která seskupuje prvky stejného typu. K jednotlivým prvkům lze přistupovat přímo, nezávisle na ostatních
- Indexy se zadávají v hranatých závorkách " []"
- Stejně jako proměnné i pole je nutné v jazyce C před použitím deklarovat. Obecná deklarace pole bez inicializace:

```
typ_pole identifikátor[velikost_pole];
```

- Příklad jednorozměrného pole s názvem `znamky`, které obsahuje pět prvků typu `char`:

```
char znamky[5];
```

- Obecná deklarace pole s naplněním defaultních hodnot zadávaných ve složených závorkách:

```
typ_pole identifikátor[velikost_pole] =
    {prvek0, prvek1, ...};
```

- Příklad jednorozměrného pole s názvem `znamky` i s naplněním hodnot:

```
char znamky[5] = {1, 2, 3, 4, 5};
char znamky[5] = {0};
```

- Pozn.: Pokud vytváříme pole, kde rozměr přesně odpovídá počtu inicializovaných hodnot, pak nemusí být velikost uvedena.

```
float koeficienty[] = {-3.14, 5.67, 123.45};
char text[] = "Jak se mate?";
```

Jednorozměrná pole, vektory: přístup k prvkům

- Ukázka aplikace s polem:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int i;
    int a[10];

    for (i = 0; i < 10; i++) {
        a[i] = 2 * i;
    }

    for (i = 0; i < 10; i++) {
        printf("prvek %d = %d\n", i, a[i]);
    }

    /* end of main function */
    return 0;
}
```

- K jednotlivým prvkům pole se přistupuje pomocí indexu v hranatých závorkách, které jsou uvedeny za identifikátorem pole
- Uvnitř závorek je celé číslo, odpovídající indexu prvku v daném poli
- Prvky pole jsou v jazyce C indexovány od nuly, tj. první prvek má index 0, druhý 1, atd.
- Výpis spuštěné aplikace:

```
prvek 0 = 0
prvek 1 = 2
prvek 2 = 4
prvek 3 = 6
prvek 4 = 8
prvek 5 = 10
prvek 6 = 12
prvek 7 = 14
prvek 8 = 16
prvek 9 = 18
```

Vícerozměrná pole

- Vícerozměrná pole lze v jazyce C tvořit přidáním více hranatých závorek, např. u dvourozměrného pole: " [] []"
- Stejně jako u jednorozměrných polí, jsou veškeré prvky vícerozměrných polí stejného typu
- Obecná deklarace vícerozměrného pole; `vel1`, atd. jsou rozměry pole v jednotlivých rozměrech:

```
typ_pole identifikátor[vel1][vel2]...;
```

- Příklad dvou a třírozměrného pole:

```
int matice[9][16];
char video[300][400][24];
```

- Obecná deklarace pole s naplněním defaultních hodnot; za rovnítkem následují hodnoty v blocích uzavřených do složených závorek:

```
typ_pole identifikátor[vel1][vel2]... =
    {hodnoty_v_blocích};
```

- Příklad deklarace vícerozměrného pole s inicializací:

```
char m[2][3] = {{1, 2, 3}, {7, 8, 9}};
```

- Prvky pole jsou v paměti uloženy ve stejném pořadí, jako při inicializaci. Tj. hodnoty z předchozího příkladu jsou uloženy takto:

```
m[0][0]     1
m[0][1]     2
m[0][2]     3
m[1][0]     7
m[1][1]     8
m[1][2]     9
```

Vícerozměrná pole: přístup k prvkům

- Zdrojový kód aplikace:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int i, j;
    int a[3][4];

    /* set all values */
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 4; j++) {
            a[i][j] = (i+1) * j;
        }
    }

    /* display all values */
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 4; j++) {
            printf("souradnice %d x %d: \t%d\n",↩
                  i, j, a[i][j]);
        }
    }

    /* end of main function */
    return 0;
}
```

- K jednotlivým prvkům pole se přistupuje pomocí několikanásobného použití indexu v hranatých závorkách
- Uvnitř závorek je celé číslo, odpovídající indexu prvku v daném rozměru
- Nezapomeňte: Prvky pole jsou v jazyce C indexovány od nuly, tj. první prvek má index 0, druhý 1, atd.
- Výpis spuštěné aplikace:

```
souradnice 0 x 0:       0
souradnice 0 x 1:       1
souradnice 0 x 2:       2
souradnice 0 x 3:       3
souradnice 1 x 0:       0
souradnice 1 x 1:       2
souradnice 1 x 2:       4
souradnice 1 x 3:       6
souradnice 2 x 0:       0
souradnice 2 x 1:       3
souradnice 2 x 2:       6
souradnice 2 x 3:       9
```

Obsah přednášky

Členění zdrojového kódu do funkcí

- Pro větší přehlednost (a efektivitu) zdrojového kódu, se aplikace v jazyce C dělí do menších celků: do funkcí
- Tři zdroje funkcí: standardní funkce jazyka C, funkce ze stažených knihoven, moje vlastní funkce
- Funkce jsou samostatné části programu, které mohou (ale nemusí) vracet hodnotu a mohou (ale nemusí) mít vstupní parametry
- Hlavní funkcí programu je vždy `main()`
- Uživatelské funkce:
  - Každá funkce musí být **deklarována** (jakýsi krátký předpis, jak funkce *vypadá*)
  - Každá funkce musí být **definována** (kompletní kód celé funkce)
  - Každá funkce může být volána prostřednictvím jiných funkcí

Deklarace funkce

- Deklarací rozumíme specifikaci uživatelské funkce ve zdrojovém kódu
- Informace z deklarace využívá překladač pro ověření, zda-li je funkce v programu použita korektně
- Deklarace postupně obsahuje:
  (1) typ návratové hodnoty (max. jedna hodnota)
  (2) název funkce (tzv. identifikátor funkce)
  (3) v kulaté závorce uvedené typy všech vstupních parametrů (může být několik, každý jiného typu, oddělených čárkou)
  (4) středník
- Je zvykem, deklarovat funkce na začátku zdrojového souboru, nebo v hlavičkovém souboru, který je do zdrojového vložen direktivou `#include`

- Obecná syntaxe deklarace:

```
navratovy_typ nazev_funkce(typ1, typ2, ...);
```

- Pokud funkce nemá žádné parametry, jako jediný parametr se explicitně uvádí klíčové slovo `void`
- Pokud funkce nemá žádnou návratovou hodnotu, uvádí se před názvem funkce klíčové slovo `void`
- Příklady deklarací funkcí:

```
/* funkce checkId má jeden parametr typu char
   a návratovou hodnotu typu int */
int checkId(char);

/* funkce displayVal má dva parametry float
   a double a žádnou návratovou hodnotu */
void displayVal(float, double);

/* funkce setupPort nemá žádný vstupní
   parametr a jeho návratová hodnota je typu
   int */
int setupPort(void);
```

Definice uživatelských funkcí

- Každá funkce musí být definována; byť né nutně ve stejném zdrojovém souboru
- Definicí funkce rozumíme sestavení *těla funkce*, tj. posloupnost přiřazení a příkazů, které se vykonají po volání dané funkce
- Ke vstupním typům z deklarace jsou přidány identifikátory (názvy) parametrů/proměnných a samotné tělo funkce je uzavřené do **složených závorek**
- Za definicí funkce není středník
- Ukončení funkce a současně nastavení návratové hodnoty se provádí příkazem `return`. Ten může být použit na libovolném místě v těle funkce

- Tělo každé funkce je definováno odděleně, tj. nevkládat jednu funkci do druhé:

```
/* definice hlavní funkce */
int main()
{
    ...     // tělo hlavní funkce
}           // není zde středník

/* definice uživatelské funkce */
float complexSum(double a, double b)
{
    ...     // tělo uživatelské funkce
    return temp;
}           // konec těla funkce complexSum
```

Volání funkce

- Volání funkce znamená její spuštění
- Volání se provádí prostřednictvím názvu funkce, včetně kulatých závorek, do kterých se uvedou identifikátory hodnot, které chceme funkci předat. Při volání se již neuvádějí datové typy:

```
/* Volání funkce findMax se dvěma parametry
   a a b. Návratovou hodnotu ulož do proměnné
   c */
c = findMax(a, b);
```

- Voláme-li funkci, která nemá parametry, uvedeme za názvem funkce pouze prázdné kulaté závorky:

```
/* funkce showHeaser nemá žádný vstupní
   parametr, ale závorky být musí */
id = showHeader();
```

- Voláme-li funkci, která nemá výstupní hodnotu, před názvem funkce nic neuvádíme:

```
checkValues(a, b, c);
```

- Funkci lze volat i v rámci parametrů jiné funkce:

```
/* Funkce findMax je volána jako parametr
   funkce printf. Tj. nejprve se vykoná
   funkce findMax a její výsledek je použit
   pro funkci printf */
printf("max hodnota: %d\n", findMax(a, b));

/* Podmínkou if se testuje návratová hodnota
   jiné funkce: bit_is_set */
if (bit_is_set(reg, id) == 1) {
    ...          // tělo podmínky
}
```

Příklad definice a volání funkce

- Zdrojový kód aplikace:

```c
/* standard input/output library */
#include <stdio.h>

/* deklarace uživatelské funkce findMax */
int findMax(int, int);

/* main function */
int main()
{
    int a, b, c;

    printf("Zadej dve cela cisla:\n");
    scanf("%d %d", &a, &b);

    c = findMax(a, b);
    printf("Vetsi hodnota je: %d\n", c);

    /* end of main function */
    return 0;
}
```

- Zdrojový kód aplikace (pokračování):

```c
/* definice uživatelské funkce findMax */
int findMax(int x1, int x2)
{
    int temp;   // deklarace lokální proměnné

    if (x1 >= x2) {
        temp = x1;
    }
    else {
        temp = x2;
    }

    return temp;// návratová hodnota funkce
}
```

- Výpis spuštěné aplikace:

```
Zadej dve cela cisla:
3
6
Vetsi hodnota je: 6
```

"Charakter" programovacích jazyků