

# Čtvrté 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



- První projekt – otázky, diskuse?
  - **Deadline: 17.10.2025, 23:59**
  - **Za každý započatý den zpoždění: -1b**
  - **Odevzdání do StudISu, jste přihlášení**
  - **Testování na serveru merlin**
- Úhlednost zdrojového kódu II
- Opakování základních pojmů
- Funkce
- Cykly II (vnořené cykly)
- Funkce pro práci s množinami

- Funkce (např. main) by měly být komentovány tak, aby bylo možno snadno vygenerovat dokumentaci k programu
- Stručný příklad:

```
/** (nebo *!)  
 * @param argc Number of arguments  
 * @param argv Individual arguments  
 * @return Program exit code  
 */  
int main(int argc, char* argv[])  
{ ... }
```

\* nepovinné

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

```
int idx;  
char *str = "Help";  
char cha1 = '\\0', cha2;  
char *str2, abc;  
  
idx = 2;  
cha2 = str[idx]; // jaký znak získáme?
```

# FUNKCE

- Funkce umožňují dekomponovat program do menších celků, které vykonávají nějakou „funkci“ 😊
- Například porovnání dvou řetězců (funkce `strcmp`), načtení řetězce ze standardního vstupu (funkce `scanf`) a podobně
- Někáká demopozice (tj. rozdělení do funkcí) bude **nutnou součástí prvního projektu**

- Už víme, že jednoduchý program může vypadat například takto:

```
#include<stdio.h>
int main() {
    // cokoli
    return 0;
}
```

- Obsahuje tento kód nějaké funkce?

- Jak do programu přidáme funkce, které jsou součástí standardu jazyka C (např. C11)?

```
#include<nazev_hlavickoveho_souboru>
```

- **Později** se naučíme, jak vytvářet a přidávat vlastní hlavičkové soubory
- Vlastní funkce můžeme vytvářet i v rámci jednoho souboru se zdrojovým kódem.

- Obecně může **deklarace funkce** vypadat např.

```
int max(int a, int b); // hlavička
```

- **int** - datový typ, který funkce vrátí
  - pokud nic nevrací, použijeme typ **void**
- **max** – název funkce (musí být unikátní v rámci všech použitých hlavičkových souborů)
- **int a, int b** – parametry, které funkce očekává
  - všimněte si datových typů
  - proměnné **a, b** jsou deklarovány a inicializovány na hodnotu, se kterou je funkce volána
  - **lze je použít v těle funkce**

Co chceme získat?

Název funkce

Co funkce potřebuje pro výpočet?

```
int max(int a, int b)
{
    // Co musí funkce
    provést, aby dosáhla
    výsledku
}
```

- Definice funkce

```
#include<stdio.h>
// PŘED PRVNÍM ZAVOLÁNÍM FCE
int max(int a, int b) { // hlavicka
// implementace
    return promenna_typu_int; // vraci
    typ integer (výsledek) }

int main() {
    int m = max(5,3);
    return 0;
}
```

- Deklarace funkce, definice funkce jinde (více zdrojových souborů)

```
#include<stdio.h>
int max(int a, int b); // deklarace
int main() {
    int m = max(5,3);
    return 0;}

// definice (kdekoli v programu)
int max(int a, int b) {
// implementace
    return promenna_typu_int;}
```

- Napište funkci

```
float sum(float a, float b) {tělo}
```

- Funkce bude vracet součet dvou čísel typu **float**
- Výsledek funkce vypište ve funkci **main**
- Ve funkci **main** se funkce zavolá např.:

```
//soucet cislic 5 a 4, vysledek ve vys  
float vys = sum(5.0, 4.0);
```

- Volání funkce je **výraz**
- Specifikátor pro výpis **%f**

- Funkce s prototypem

```
int my_crazy_min(int a, int b);
```

vrací **absolutní hodnotu menšího čísla**

- Funkce s prototypem

```
int my_max(int a, int b);
```

vrací **hodnotu většího čísla**

- **Řetězcový literál (neboli konstanta)**

- Inicializujeme

```
char *s = "Hello";
```

- **Nelze měnit za běhu programu**

- **Nekonstantní řetězec**

- Inicializujeme na prázdný řetězec nebo na řetězec, který již obsahuje slovo

```
char str[6] = "Hello"; // proc 6???
```

```
char str[] = "Hello"; // automaticky
```

- **Lze měnit za běhu programu**

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

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

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];`

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!

- Napište funkci `strlen_m`, která bude počítat délku řetězce a tuto délku vrátí

- Deklarace

```
int strlen_m(char *str);
```

- Implementace

```
int strlen_m(char *str) {  
    ...  
    return delka_retezce_str;  
}
```

- **Nápověda: řetězec končí znakem `\0`**

- Funkce pro hledání prvního výskytu znaku **ch** v řetězci **str**
  - Vrací index znaku nebo -1, pokud nebyl znak nalezen

```
int my_strchr(char *str, char ch);
```

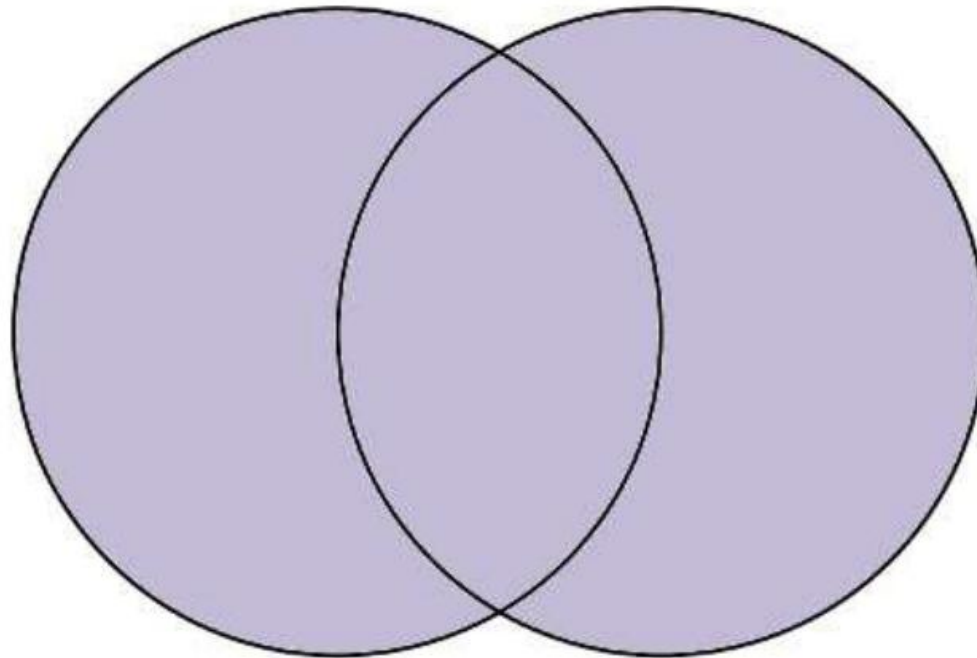
- Funkce pro hledání posledního výskytu znaku **ch** v řetězci **str**
  - Vrací index znaku nebo -1, pokud nebyl znak nalezen

```
int my_strrchr(char *str, char ch);
```

Cykly II

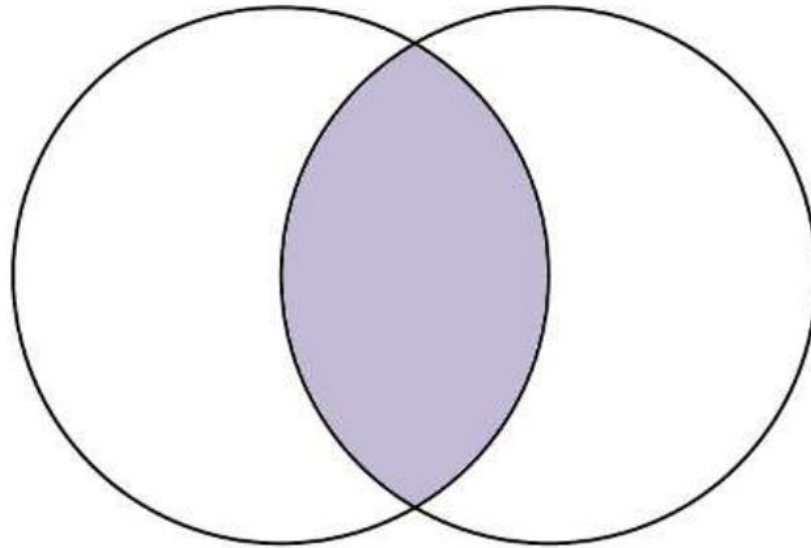
# VNOŘENÉ CYKLY

- Implementujte následující funkce
  - **isInSet** (prvek je v množině)
  - **isSet** (ověření, zda je pole množina – každý prvek se vyskytuje jen jednou)
  - **printUnion** (vypíše sjednocení množin)
  - **printIntersection** (vypíše průnik množin)
  - **printProduct** (vypíše kartézský součin množin)



$$A \cup B = \{x; x \in A \vee x \in B\}$$

$$x \in A \cup B \iff x \in A \vee x \in B$$



$$A \cap B = \{x; x \in A \wedge x \in B\}$$

$$x \in A \cap B \iff x \in A \wedge x \in B$$

$$A \times B = \{[x, y]; x \in A \wedge y \in B\}$$

$$[x, y] \in A \times B \iff x \in A \wedge y \in B$$

$$[x, y] \notin A \times B \iff x \notin A \vee y \notin B$$

- Implementujte následující funkce
  - **isInSet** (prvek je v množině)
  - **isSet** (ověření, zda je pole množina – každý prvek se vyskytuje jen jednou)
  - **printUnion** (vypíše sjednocení množin)
    - Návrátový typ **void**
  - **printIntersection** (vypíše průnik množin)
    - Návrátový typ **void**
  - **printProduct** (vypíše kartézský součin množin)
    - Návrátový typ **void**
- Jak budou vypadat definice (hlavičky) funkcí?

# DALŠÍ PŘÍKLADY K PROCVIČENÍ

- Načtete dvě pole číslic o **délce 5** a určete počet společných číslic.
  - **Varianta 1:** pole neobsahují duplicitu
  - **Varianta 2:** pole mohou obsahovat duplicitní číslice (např. pro [1,1,5,5,2] a [5,5,1,1,3] je v výsledek 2)
  - **Varianta 3:** zjednodušení programu pomocí funkce, která vrátí pro pole a číslo 0/1 podle toho, jestli se v poli nachází

- Funkce pro práci s polem čísel (počet čísel v poli předejte parametrem)

- Funkce vrátí největší číslo v poli celých čísel

```
int get_max(int arr[], int len);
```

- Funkci, která vrátí součet všech čísel v poli celých čísel

```
int get_sum(int arr[], int len);
```

- Funkci, která porovná prvky ve dvou polích typu int a vrátí 1, pokud jsou všechny prvky v poli arr1 menší než prvky v poli arr2

```
int smaller_than(int arr1[], int arr2[],  
int len);
```

- Napište funkci, která vrátí řetězcový literál odpovídající skloňování slova hodina

```
char* hodiny2str(int hodin);
```

- Např. pokud `hodin==1`, funkce vrátí hodina apod.
- Napište funkci, která vrátí řetězcový literál odpovídající skloňování slova minuta

```
char* minuta2str(int minut);
```

- Např. pokud `minut==1`, funkce vrátí minuta apod.

Děkuji za pozornost

- **Základní styly:**

- llvm

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

- Mozilla

- [https://developer.mozilla.org/en-US/docs/Mozilla/Developer\\_guide/Coding\\_Style](https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style)

- Gnu

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

# HLEDÁNÍ ARGUMENTU

- 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 nebo použít funkce `strtod`, `strtod` (později)

- Nutné hlavičkový soubor: `#include <string.h>`
- Lexikografické porovnání řetězců `s1` a `s2`

```
int strcmp(char* s1, char* s2);
```

- Funkce vrací číslo (`int`)
  - `== 0` pokud jsou řetězce `s1` a `s2` stejné
  - `> 0` pokud je řetězec `s1` lexikograficky větší než `s2`
  - `< 0` pokud je řetězec `s1` lexikograficky menší než `s2`
- A [další](#)

- Složený příkaz: **`if (podmínka) {...} else {...}`**
  - Pokud **je** podmínka pravdivá: provede se větev **`if`**
  - Pokud **není** podmínka pravdivá: provede se větev **`else`**
- **Jak tedy zjistíme, že jsou dva řetězce shodné?**
  - 1) Musíme zavolat funkci **`strcmp()`**
  - 2) Musíme v podmínce ověřit, jak porovnání dopadlo
- 2 způsoby:

```
if (strcmp("ahoj", "ahoj") == 0) {...}
```

```
int pom;  
pom = strcmp("ahoj", "ahoj"); // lze také...  
if (pom == 0) {...}
```

- Napište program, který zjistí, **zda byl zadán** argument **-help**
  - Pokud ano, vypište libovolný text na obrazovku
- Zároveň vypište, **na jakém indexu** se argument **-help** nachází
- Zkuste hledání argumentu implementovat do funkce, která bude vracet index argumentu

```
int findArgH(int c, char* a[]);  
findArgH(argc, argv); // volání
```

- **Nápověda**

```
int strcmp(char* s1, char* s2);
```

- Funkce vrací 0, pokud jsou řetězce shodné, pro kontrolu argumentů využijte cyklus

**ČAS**

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

int main()
{
    // globalni cas
    time_t now = time(NULL);
    // lokalni cas
    struct tm *datetime = localtime(&now);
    // pristup ke slozкам pomoci ->
    printf("Year = %d\n", datetime->tm_year +
1900);
    return 0;}

```

- Napište funkci, která vrátí řetězcový literál odpovídající skloňování slova hodina

```
char* hodiny2str(int hodin) ;
```

- Např. pokud `hodin==1`, funkce vrátí hodina apod.
  - Napište funkci, která vrátí řetězcový literál odpovídající skloňování slova minuta
- ```
char* minuta2str(int minut) ;
```
- Např. pokud `minut==1`, funkce vrátí minuta apod.
  - Poznámka: lze i `return "ahoj" ;`

```
$ ./cas  
Je prave 9 hodin a 3 minuty.  
$ date  
Fri  5 Oct 09:03:14 CEST 2018
```

# KALKULAČKA II

- Jistě si pamatujete na kalkulačku z minulého cvičení 😊
- Dnes vyzkoušíme implementaci pomocí standardního vstupu
- Schopnosti aplikace si přizpůsobte sami
- Vstup načítejte pomocí funkce `scanf` s formátovacím řetězcem `%s`
- Načtená slova zpracovávejte pomocí funkce `sscanf`  
`sscanf(nactene_slovo, "%lf", &cislo)`
- Výsledky tiskněte na standardní výstup pomocí funkce `printf` se specifikátorem `%g`
- **Snažte se, aby program neměl žádnou funkci delší než 25 řádků. Žádný řádek nesmí být delší než 80 znaků.**

# NAČÍTÁNÍ SLOV ZE STANDARDNÍHO VSTUPU (STDIN)

- **Načítání slov ze standardního vstupu (stdin)**
- Možnosti:
  - Po znacích: např. funkce `getchar()`
  - Celý řádek: funkce `scanf`

`scanf()` **vrací:**  
**cislo** = počet správně načtených  
formátovacích řetězců  
**cislo**  $\leq$  **0** při chybě

```
int cislo = scanf("%format", kam);
```

- **Načítání slov ze standardního vstupu (stdin)**
- Možnosti:
  - Po znacích: funkce `getchar ()`
  - Celý řádek: funkce `scanf`
- Napíšeme program, který načte slovo o maximální délce **10 znaků a vypíše ho**
- Bude slova načítat, dokud nenarazí konec souboru (**EOF = End Of File**)
  - Windows: CTRL-Z                      Linux: CTRL-D

```
char slovo[?]; // co tu bude? 😊  
int code = scanf("%10s", slovo);
```

# ZPRACOVÁNÍ ARGUMENTŮ PŘÍKAZOVÉ ŘÁDKY

- Následující příklad se může hodit v projektech
- Knihovna `#include <stdbool.h>`
  - Poté můžete používat datový typ `bool`
    - Pravdivostní hodnoty `true/false`
    - Číselně: `false == 0`, `true != 0` (cokoli jiného než 0)
    - `bool pravdivaPodminka = true;`
    - `bool nepravdivaPodminka = false;`
  - V podmínce

```
if (pravdivaPodminka) { ... }
```

```
if (!nepravdivaPodminka) { ... }
```

- A na závěr ještě operátory
  - **Aritmetické:** unární, binární
  - **Logické:** && (AND), || (OR)
  - **Relační:** ==, !=, >, <, >=, <=

| A | B | AND | OR |
|---|---|-----|----|
| 0 | 0 | 0   | 0  |
| 0 | 1 | 0   | 1  |
| 1 | 0 | 0   | 1  |
| 1 | 1 | 1   | 1  |

- Výpis chybových hlášení → standardní chybový

```
fprintf(stderr, "Chyba! \n");
```

- → **Toto je potřeba v projektech!**
- A teď konečně příklad 😊

- Napište program, který bude kontrolovat zadané argumenty příkazové řádky
  - Pokud **nebyl zadán žádný argument**, vypíše **chybu** na **stderr**
  - Pokud byl zadán **argument -h** (většinou nápověda), vypíše na standardní výstup řetězec **Napoveda**
  - Pokud byl zadán **argument -o**, vypíše na standardní výstup libovolný jiný řetězec
  - Argumenty **-h a -o nelze použít zároveň** (v tom případě chyba, výpis na **stderr**)
  - Program může přijímat **pouze argumenty - h a - o**, v opačném případě chyba
- Pokuste se **oddělit zpracování argumentů od výpisů**

```
• bool hflag = false; ... if (hflag) ...
```