Základy programování (IZP)

Čtvrté počítačové cvičení

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

4. týden

Organizace

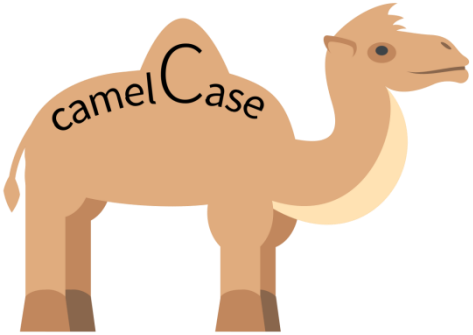
- Projekty
  - Odevzdání 31.10.
- Otázky?

Náplň cvičení

- ovládat datové typy, proměnné a výrazy
- ovládat práci s poli (sekvenční průchod)
- porozumět **funkcím** a řetězcům
- pochopit význam **vnořených cyklů**

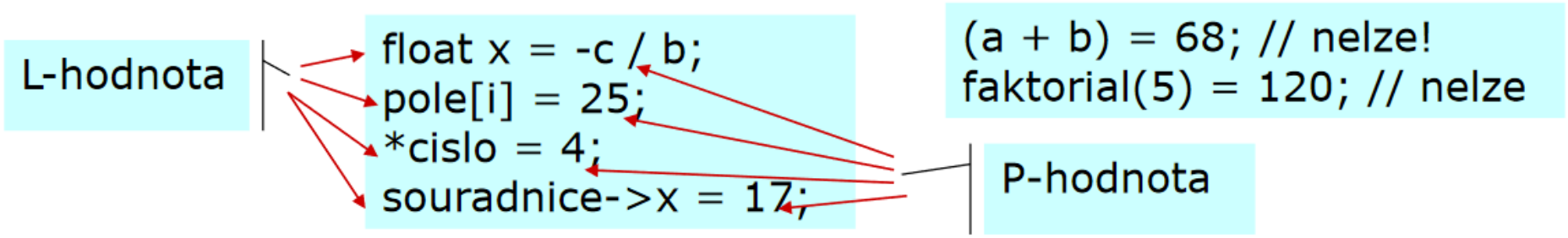
Formátování

- **Formátování zdrojového kódu**
- **Pojmenování (identifikátor)**
  - To, jak je proměnná pojmenovaná, **závisí pouze na programátorovi**
  - **ALE proměnné** pojmenované `test12345`, `mojenejuzasnejsipromenna` **apod. asi nebudou úplně nejvhodnější**
  - varianta_dlouheho_id (snake case), variantaDlouheholD (camel case)
  - Indexy: `i, j, k, l, …`
  - Řetězce: `str, s, …`
  - Znaky: `c, ch, …`
  - Konstanty: `KONSTANTA` (velká písmena)
  - Datové typy: `TArray, ColorSpace, …` (později)



Opakování

- **L-hodnota**
  - je objekt v paměti, kterému lze přiřadit hodnotu
- **P-hodnota**
  - výraz, který má vždy hodnotu a který vystupuje na pravé straně přiřazovacího operátoru



Proměnná – deklarace, definice, inicializace

- Několik příkladů

```
int i;
char s[] = "Hello";
char c1 = '\0', c2; // ne
char s2[10], c;

i = 2;
c2 = s[i]; // jaký znak získáme?
```

Opakování: operátor čárka

- Vyhodnocuje se zleva doprava a celkový výraz nabývá hodnoty nejpravějšího podvýrazu.

```
int a = 1, b = 2, c = 3, d = 4;
int vysledek = (a+b, c+d);
int vysledek2 = a+b, c+d;
```

- Jaký bude výsledek?
- NEPOUŽÍVAT!
- priorita ”=” > priorita ”,”

Priorita operátorů

Priorita	**Operátory**	**Asociativita**	**skupina op.**
1.	() [] -> .	zleva doprava	primární
2.	! ~ ++ -- + - (typ) * & **sizeof**	zprava doleva	unární
3.	* / %	zleva doprava	multiplikativní
4.	+ -	zleva doprava	aditivní
5.	<< >>	zleva doprava	posuny
6.	< <= > >=	zleva doprava	relační
7.	== !=	zleva doprava	relační
8.	&	zleva doprava	bitový součin
9.	^	zleva doprava	exkl. bit. souči.
10.	|	zleva doprava	bitový součet
11.	&&	zleva doprava	logický součin
12.	||	zleva doprava	logický součet
13.	?:	zprava doleva	ternární podm.
14.	= += -= *= /= %= >>= <<= &= |= ^=	zprava doleva	přiřazení
15.	,	zleva doprava	op. čárka

Datový typ pole – řetězce

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!
- **Pozor**: `scanf("%s", retezec); //chybí zde &, už se jedná o adresu!!`
- `Printf("%s", retezec); // normálně`

- **Řetězcový literál** (neboli **konstanta)**
  - Inicializujeme

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

  - **Nelze měnit za běhu programu**

- **Nekonstantní řetězec**
  - Inicializujeme (na případně prázdný řetězec)

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

  - nebo na řetězec, který již obsahuje slovo

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

  - **Lze měnit za běhu programu**

break & continue

- **break**
  - Při vnořování cyklů ukončí break jen nejbližší cyklus, ve kterém je použit.
  - **Příklad: zobrazí čísla 0 až 5 pětkrát**
- **continue**
  - Příkaz continue si vynutí nové vyhodnocení podmínky cyklu, přičemž se přeskočí všechny příkazy mezi ním a koncem těla cyklu.

```
for(int i=0; i<5; ++i)
{
   for(int j=0; j<100; ++j)
   {
        printf("%d", j);
        if(j==5) break;
   }
   printf("\n"); // break skočí sem
}
```

```
for(int x=0; x<100; ++x)
{
   continue;
   //nikdy se neprovede
   printf("%d ", x);
}
```

FUNKCE

Funkce

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

Funkce

- Základní program může vypadat například takto:

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

- Obsahuje kód výše nějaké funkce???

Funkce

- Jak do programu přidáme funkce, které jsou součástí standardu?

Funkce

- Jak do programu přidáme funkce, které jsou součástí standardu?

```
#include<nazev_hlavickoveho_souboru>
```

- **Později** se naučíme, jak vytvářet a přidávat vlastní hlavičkové soubory

Funkce

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

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

  - int - datový typ, který funkce vrací
    - 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**

Funkce – definice: hlavička + tělo

Co chceme získat?

Název funkce

Co funkce potřebuje pro výpočet?

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

Funkce – použití v programu

```
#include<stdio.h>
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;
}
```

Funkce – jednoduchý příklad

- Napište funkci s prototypem

```
float sum(float a, float b);
```

- Funkce bude vracet součet dvou čísel typu **`float`**
- Výsledek funkce vypište ve funkci **`main`**
- V programu se funkce volá např.:

```
//soucet cislic 5 a 4, vysledek v res
float res = sum(5,4);
```

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

Další jednoduchá funkce

- Funkce s prototypem

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

vrací absolutní hodnotu menšího čísla

Další jednoduchá funkce

- Načtěte jméno a příjmení (2 řetězce s omezenou délkou, délku si zvolte sami) a zjistěte, zda obsahují pouze písmena anglické abecedy.
- Použijte funkci s prototypem

```
int is_alpha(char c);
```

funkce pro detekci písmene

- Porovnání: isalpha v `<ctype.h>`

Funkce - práce s řetězci

- Napište funkci **`strlen_m`**, která bude počítat délku řetězce a tuto délku vrací
- Hlavička

```
int strlen_m(char str[]);
```

- Implementace

```
int strlen_m(char str[]) {
  …
  return delka_retezce_str;
}
```

- Nápověda: řetězec končí znakem \0

Další funkce (DÚ)

- Dále implementujte:
  - Funkci 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);
```

  - Funkci 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);
```

Funkce pro práci s polem čísel

- 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);
```

Vnořené cykly

- Načtěte dvě pole číslic o velikosti 5 a určete počet společných číslic (pozice nehraje roli).
  - předpokládejme, že pole **neobsahují duplicity.**
  - Vylepšit o možnost duplicit (např. pro 1,1,5,5,2 a 5,5,1,1,3 je v výsledek 2).
  - Zjednodušit tento program pomocí funkce, která pro zadané pole a číslo vrátí 0/1 podle toho, jestli se číslo v poli nachází.

Příklad:

Arr1[] = {1,4,3,2,9}

Arr2[] = {0,2,3,1,10}

Výsledek: 3

	1	4	3	2	9
0	ne	ne	ne	ne	ne
2	ne	ne	ne	ano	ne
3	ne	ne	ano	ne	ne
1	ano	ne	ne	ne	ne
10	ne	ne	ne	ne	ne

Funkce pro práci s množinami

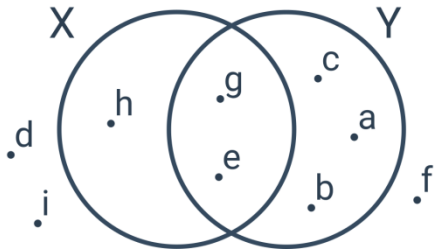
- Uvažujeme množiny čísel s fixní maximální velikostí (zatím nemáme dostatečnou znalost pro efektivnější reprezentaci).

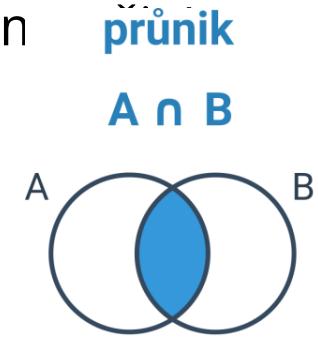
```
isInSet(int arr[], int len, int value);
```

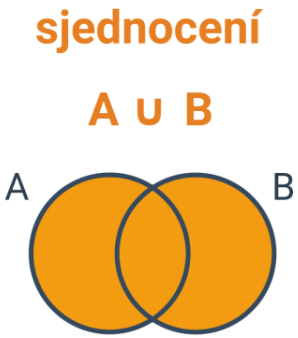
- **Ověření, že pole reprezentuje množinu, tj. každý prvek se vyskytuje jen jednou)**

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

- **Dále pro rychlejší:** printIntersection, printUnion, printProduct (průnik, sjednocení a kartézský součin n







Děkuji za pozornost