Základy programování (IZP)

Čtvrté počítačové cvičení

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Gabriela Nečasová, inecasova@fit.vutbr.cz

10.10.2016

Důležité informace

- **Můj profil**: http://www.fit.vutbr.cz/~inecasova/
  - Kancelář: A221
  - Konzultační hodiny: po domluvě emailem
  - Karta Výuka → odkaz na osobní stránky:
    IZP 2016/2017 Cvičení → Materiály
- **Komunikace**: email – prosím používejte předmět:
  IZP - <předmět emailu>

Důležité informace

- **Nezapomeňte** se ve WISu přihlásit na všechny **4 termíny**:
  - 3 projekty IZP
  - 1 dokumentace ke třetímu projektu
- Pozor: přihlašujte se ke správnému asistentovi!
- Přihlašování začalo **10. 10. 2016 v 8:00**
- **PŘIHLAŠTE SE RADĚJI HNED TEĎ**
- Přihlašování končí 30. 10. 2016 ve 22:00

Důležité informace

- 1. projekt – Práce s textem
  - 31. 10. Obhajoba projektu
  - 6.11. Odevzdání projektu do WISu
  - Jméno souboru: **proj1.c**
- 2. projekt
  - Odevzdání již 27. 11. 2016
- 3. projekt
  - Odevzdání již 11. 12. 2016

- **Proměnné, datové typy, příkazová řádka**
  - Řešení všech máte na wiki stránkách…
  - … ale když už jsme tady, vyzkoušíme je aspoň trochu samostatně (jako minule)

  - Pokud nebudete vědět, jak se zapisuje nějaká programová konstrukce (proměnná, podmínka, atd.), zapište si do zdrojového souboru komentář
    - `// tady se budou vypisovat dva retezce`
    - ```
      /* komentar na vice radku
      */
      ```
      ```

- Základní pojmy
  - **Proměnná**

- Základní pojmy
  - **Proměnná**
    - Pojmenované místo v paměti, ve kterém uchováváme data.
    - Má určitý datový typ a její hodnota se **může** za běhu programu měnit.
    - Příklad: `int a=10;`

- Základní pojmy
  - **Proměnná**
    - Pojmenované místo v paměti, ve kterém uchováváme data.
    - Má určitý datový typ a její hodnota se **může** za běhu programu měnit.
    - Příklad: `int a=10;`
  - **Konstanta**

- Základní pojmy
  - **Proměnná**
    - Pojmenované místo v paměti, ve kterém uchováváme data.
    - Má určitý datový typ a její hodnota se **může** za běhu programu měnit.
    - Příklad: `int a=10;`
  - **Konstanta**
    - Pojmenované místo v paměti, ve kterém uchováváme data.
    - Má určitý datový typ, její hodnota se **nemůže** za běhu programu měnit.
    - Příklad: `const int b=10;`

- **Datový typ**

- **Datový typ**
  - Jaká **data** mohou být na pojmenovaném místě **uložena**
  - Jaké **operace** mohou být s daty prováděny
  - Např. **`int`**, **`double`**, **`float`**, **`char`**, **`bool`**, …
  - Pro určení **velikosti datového typu** můžeme s výhodou použít operátor **`sizeof()`**
    - Příklad za okamžik

- **Pojmenování (identifikátor)**
  - To, jak je proměnná pojmenovaná, **závisí pouze na programátorovi**
  - **ALE proměnné** pojmenované `test12345`, `mojenejuzasnejsipromenna` **apod. asi nebudou úplně nejvhodnější**

  - Indexy: `i, j, k, l, …`
  - Řetězce: `str, s, …`
  - Znaky: `c, ch, …`
  - 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**

- **Deklarace**
  - Deklarace proměnné **vytváří novou proměnnou**, kterou je možno použít dál v programu

```
int i; // nepřiřazena žádná hodnota
```

- **Deklarace**
  - Deklarace proměnné **vytváří novou proměnnou**, kterou je možno použít dál v programu

```
int i; // nepřiřazena žádná hodnota
```

- **Inicializace**
  - Přiřazení počáteční hodnoty **deklarované** proměnné

```
int j = 10; // inicializováno na 10

a = 60;  // NELZE, proměnná není
         // deklarována
```

- **Přiřazení** (operátor **=**)
  - Nastavení hodnoty proměnné

```
int j = 10; // inicializováno na 10

j = 60;     // nová hodnota j je 60
```

- **Porovnání** (operátor **==**)
  - Využíváme v podmínkách `if(i == 0) {…}`
  - **Pozor: neplést s přiřazovacím operátorem!**

- Několik ukázek z wiki

```
int i;
char* s = "Hello";
char c1 = '\0', c2;
char *s2, c;

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

- **Ukazatele**
  - 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
```

- Na minulém cvičení jsme už řetězce trochu zkoumali …
- … a nyní si je zopakujeme oficiálně ☺

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

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

  - **Nelze měnit za běhu programu**

Datový typ pole

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

Datový typ pole

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, spojité místo v paměti

Datový typ pole

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, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`

Datový typ pole

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, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek

Datový typ pole

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, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek
- **Velikost moje_pole:**

```
int velikost = 6*sizeof(int);
```

Datový typ pole

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, spojité místo v paměti
- **Deklarace staticky**: `int moje_pole[6];`
- **Velikost pole**: operátor `sizeof()` – velikost v bajtech
  - U polí vrací součet velikostí jeho položek
- **Velikost moje_pole:**
  `int velikost = 6*sizeof(int);`
- **Pozor**: velikost datového typu záleží na procesoru
  - `sizeof(int)` může být 2, 4 nebo 8

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'

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'`

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!

PŘÍKLADY

Výpis argumentů programu

- Na rozehřátí poslední příklad z minula, ale nejdřív argumenty trochu zopakujeme …

Argumenty programu

- Jednotlivé argumenty budeme oddělovat mezerou
- Argumenty se dají získat pomocí následující konstrukce:

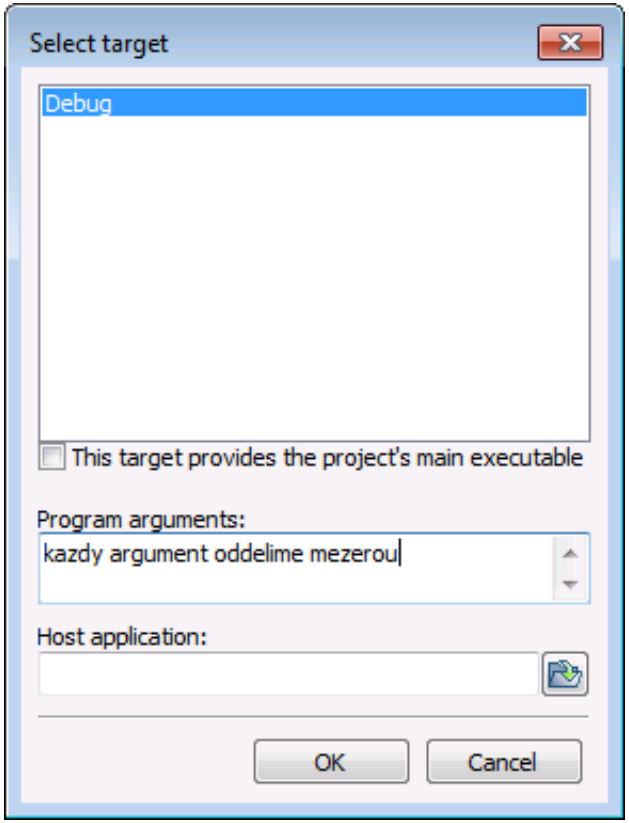
```
int main(int argc, char* argv[])
{
   // argc – počet argumentů
   // argv – jednotlivé argumenty, argv[0]
   // (název souboru s programem) }
```

- Pro `./hello –sum 10 20` **argc=4**

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

Spouštění programu s parametry

- **Code::Blocks**
  - Project → Set program‘s arguments → OK
  - Spustíme program



- **Linux**

```
./program arg1 arg2 arg3
```

- Jednotlivé argumenty jsou od sebe odděleny mezerou

PŘÍKLAD 1
VÝPIS ARGUMENTŮ PŘÍKAZOVÉHO ŘÁDKU

Výpis argumentů programu

- Napište program, který bezpečně **vypíše všechny argumenty**, se kterými byl program spuštěn
- Nápověda: Použijte cyklus (**`while`**, **`for`**)
  - Cyklus **`while`**: nezpomeňte inkrementovat řídící proměnnou v těle cyklu
  - Funkce **`printf()`** nutné **`#include <stdio.h>`**

```
printf("text");
```

```
int i = 10;
printf("cislo: %d", i);
```

```
char* retezec = "Ahoj";
printf("retezec: %s", retezec);
```

Cykly a převody mezi nimi

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

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

PŘÍKLAD 2
POROVNÁNÍ DVOU ŘETĚZCŮ

- Nutné vložit knihovnu: `#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

Funkce `strcmp()` a podmíněný příkaz

- 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:
- Lepší:

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

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

Porovnání dvou řetězců

- Napište program, který porovná **první argument programu** s **řetězcem `--help`**
  - Pokud se první argument s tímto řetězcem shoduje, vypište libovolný text na obrazovku
- Zkontrolujte, **zda byl první argument opravdu zadán**, složitější kontrolu argumentů **NEPROVÁDĚJTE**.
- Nápověda

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

- Funkce vrací 0, pokud jsou řetězce shodné

PŘÍKLAD 3
HLEDÁNÍ ARGUMENTU -H

Hledání argumentu -h

- Napište program, který zjistí, **zda byl zadán** argument **`-h`**
  - Pokud ano, vypište libovolný text na obrazovku
- Zároveň vypište, **na jakém indexu** se argument **`-h`** nachází

- Nápověda

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

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

PŘÍKLAD 4
ZPRACOVÁNÍ ARGUMENTŮ PŘÍKAZOVÉ ŘÁDKY

- **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 výstup:

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

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

Zadání

- 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)`** …

PŘÍKLAD 5
PŘEVOD ŘETĚZCE NA ČISLO

Převod řetězce na číslo

- Několik funkcí (a hlavičkových souborů), které se mohou hodit
- **Knihovna** `#include <stdlib.h>`
- **Funkce** `int atoi(const char* str);`
  - Převádí **řetězce na čísla**, vrací získané číslo
    - POZOR: je nutné funkci předat opravdu číslo, ne nesmysl

```
int cislo = atoi("123abc");
printf("prevedene cislo: %d", cislo);
// prevedene cislo: 123
```

    - Pokud předáte opravdu nesmysl, vrátí 0
    - Bezpečnější varianta: `strtol()`

Převod řetězce na číslo

- **Funkce `long int strtol (const char* str, char** endptr, int base);`**
  - **`str`** – řetězec k převodu
  - **`endptr`** – ukazatel na poslední znak, který zbyl po konverzi
    - **`if(*endptr != '\0') { // chyba }`**
  - **`base`** – základ, v tomto případě desítková soustava (tedy 10)

PŘÍKLAD 6
NAČÍTÁNÍ SLOV ZE STANDARDNÍHO VSTUPU (STDIN)

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

`scanf()` vrací:

cislo = 1 pokud se slovo načetlo

cislo <= 0 při chybě

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

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

`scanf()` vrací:

cislo = 1 pokud se slovo načetlo

cislo <= 0 při chybě

- 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);
```

Příště - první bodované cvičení

- Budu se ptát na jednoduché věci, které se budou týkat tématu cvičení nebo toho, co jsme tady už dělali
- Budete zase programovat samostatně, tak jako na tomto a předchozím cvičení
  - Pokud vám dělaly problémy dnešní příklady, využijte tento týden k procvičení
- **Hodnocení**
  - Ráda bych všem dala **ALESPOŇ 1b**, ale budu chtít vidět aktivitu a to, že jste se na cvičení připravili
  - Se samostatným příkladem případně pomůžu, ale většinu musíte zvládnout sami

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
  - Gnu
    - https://www.gnu.org/prep/standards/standards.html