Základy programování (IZP)

Třetí počítačové cvičení

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

03.10.2016

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>

- **Přehlašování termínu laboratoří**: **3. – 9. 10. 2016**

Důležité informace

- 1. projekt – Práce s textem
  - 31.10. Obhajoba projektu
  - 6. 11. Odevzdání projektu do WISu
  - Jméno souboru: **proj1.c**

- **Jednoduché programy**
  - Řešení všech máte na wiki stránkách…
  - … ale když už jsme tady, vyzkoušíme je aspoň trochu samostatně
  - Pokud nebudete vědět, jak se zapisuje nějaká programová konstrukce (proměnná, podmínka, atd.), zapište si do zdrojového souboru komentář
    - `// tady se budou vypisovat dva řetězce`

Získávání nápovědy

- Nápovědu můžete získat
  - z **internetu**: vygooglit název příkazu + c
    - Dobrý zdroj: http://www.cplusplus.com/
  - pomocí **příkazu `man`** (linux)
    - např. pokud chcete získat informace o funkci **`printf`**, zadejte:

```
man 3 printf
```

- **Kniha**
  - Herout, P. Učebnice jazyka C.
    - Mohla by ještě být v knihovně

Hello world

- Napište program, který na obrazovku vypíše pozdrav světu
- Určitě si ho pamatujete z předchozího cvičení
- Kostra (může vypadat nějak takto, ostatní dnes sami):

```
#include <stdio.h>

// funkce main()
// výpis pozdravu
// návratová hodnota
```

- 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;`

- Základní pojmy
  - **Inicializace**

- Základní pojmy
  - **Inicializace**
    - Nastavení hodnoty proměnné nebo konstanty

- Základní pojmy
  - **Inicializace**
    - Nastavení hodnoty proměnné nebo konstanty
  - **Příkaz**

- Základní pojmy
  - **Inicializace**
    - Nastavení hodnoty proměnné nebo konstanty
  - **Příkaz**
    - Definuje činnost, kterou program vykoná (např. výpis textu na obrazovku)

- Základní pojmy
  - **Inicializace**
    - Nastavení hodnoty proměnné nebo konstanty
  - **Příkaz**
    - Definuje činnost, kterou program vykoná (např. výpis textu na obrazovku)
  - **Operátory**

- Základní pojmy
  - **Inicializace**
    - Nastavení hodnoty proměnné nebo konstanty
  - **Příkaz**
    - Definuje činnost, kterou program vykoná (např. výpis textu na obrazovku)
  - **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

PŘÍKLADY

- V příkladech budeme používat **jedinou knihovní funkci `printf()`**
- Funkce se nachází v knihovně **`stdio.h`**

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

- **Použití:**

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

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

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

- Napište program, který:
  - Vypíše na obrazovku **číslo 42** (%d)
  - Deklaruje **proměnnou typu `int`** (může se jmenovat libovolně)
  - K deklarované proměnné **přičte hodnotu 3**
  - **Opět vypíše** novou hodnotu na obrazovku (%d)

Jednoduché podmínky (`if`)

- Základní pojmy
  - podmíněný příkaz
  - **`if(podmínka) příkaz; else příkaz`**
- Napište program, který
  - Deklaruje a inicializuje **proměnnou typu `int`** na hodnotu 42
  - K hodnotě této proměnné **přičte 3**
  - **Pokud** je hodnota proměnné větší než 44, program o tomto nějak informuje uživatele.
  - **Pokud** je hodnota proměnné větší než 45, program o tomto informuje uživatele.
    V opačném případě program ohlásí, že zadaná hodnota není větší než 45.

Řetězcové literály (konstanty)

- Základní pojmy
  - Řetězec

- Napište program, který
  - Deklaruje a inicializujte **dva řetězcové literály**
    - datový typ **`char*`**, můžete je inicializovat na cokoli
  - **Vypíše** jejich obsah (%s)
  - Očísluje je (%d) a znovu je vypíše

  - Deklaruje a definuje **proměnnou typu `int` na 1**
  - Vypíše první literál a očíslujte ho **`int`** proměnnou
  - Inkrementuje tuto proměnnou (zvýší její hodnotu o 1)
  - Vypíše druhý literál a očísluje ho **`int`** proměnnou

Argumenty programu

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

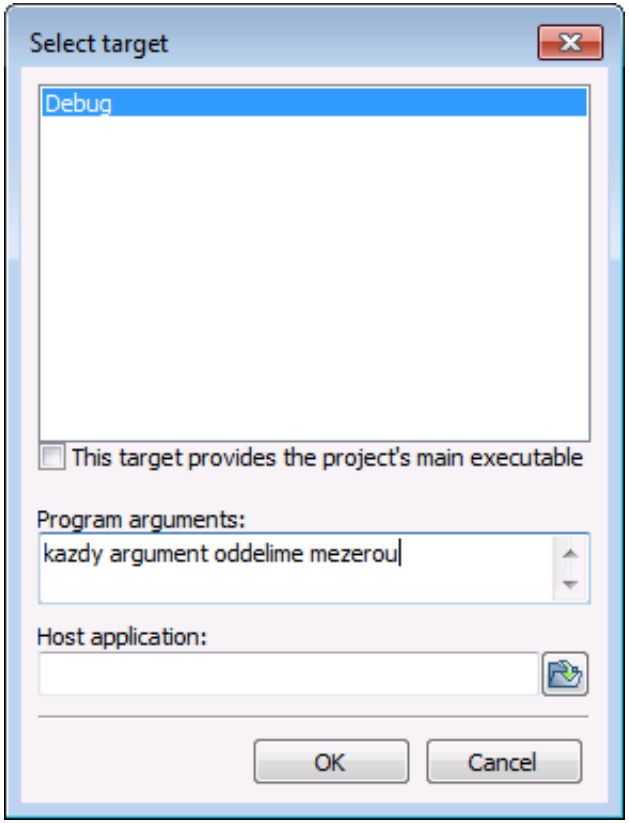
```
int main(int argc, char* argv[])
{
  // argc – počet argumentů
  // argv – jednotlivé argumenty, argv[0]
  // (název souboru s programem) }
```

- 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**

Výpis argumentů programu

- Napište program, který vypíše
  - **první tři argumenty**, se kterými byl program spuštěn
  - **počet argumentů** programu

- Tentokrát společně ☺

- Základní pojmy
  - **Cyklus** (počítaný, nepočítaný)

- Základní pojmy
  - **Cyklus** (počítaný, nepočítaný)
    - Cyklem myslíme opakování určité části programu
    - **Počítaný**: známý počet průchodů (**`for`**)
    - **Nepočítaný**: neznámý počet průchodů (**`while, do-while`**)

Jak vytvořit cyklus?

- Bude to pro vás jedna z nejčastějších činností, takže je asi dobré to dostat do krve

- Použijeme příklad z wiki a na něm si ukážeme, jak se cyklus tvoří
  - Budeme opět **vypisovat všechny argumenty**
  - Nejdříve vypíšeme **název programu** a **počet argumentů**
  - Poté inicializujeme **proměnnou typu `int`**, kterou budeme indexovat jednotlivé proměnné
  - **Pokud** bude **`argc`** větší než hodnota proměnné typu **`int`**, **vypíšeme** obsah na obrazovku (**`argv[]`**) a **inkrementujeme** proměnnou
  - A úplně stejně pro ostatní argumenty …

- Zjevný problém u předchozího příkladu
  - Mnohokrát opakujeme **stejný kód**

- **Řešení: opakující se kód zkopírujeme do cyklu**

- Zjevný problém u předchozího příkladu
  - Mnohokrát opakujeme **stejný kód**

- **Řešení: opakující se kód zkopírujeme do cyklu**

```
// inicializace
while (argc > i)
{
   printf("argument %d: %s\n",i,argv[i]);
   i = i+1; // i++;
}
```

Druhá možnost

- Na wiki máte jedno možné řešení, zkusme to jinak (pomocí cyklu **`for`**)
- Cyklus **`for`** se hodí v případech, kdy víme, kolikrát se má cyklus provést
  - **`argc`**

Druhá možnost

- Na wiki máte jedno možné řešení, zkusme to jinak (pomocí cyklu **`for`**)
- Cyklus **`for`** se hodí v případech, kdy víme, kolikrát se má cyklus provést
  - **`argc`**

```
for (int i = 1; i < argc; i++)
{
   printf("argument %d: %s\n",i,argv[i]);
}
```

- Podrobněji na dalších cvičeních a přednáškách ☺

Děkuji za pozornost

Cykly a převody mezi nimi

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

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

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'`
- **Pozor**: `char pole[5];`
  - Není zde místo pro ukončovací nulu (`'\0'`)
- **Pozor**: je nutné hlídat meze pole!