Základy programování (IZP)

Třetí 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.vutbr.cz

3. týden

Organizační okénko

- Projekty
  - Odevzdání 31.10.
- Otázky?

Náplň cvičení

- ovládat aritmetické, relační a logické výrazy (včetně odpovídajících datových typů)
- porozumět **znakům** (ASCII) a **řetězcům** v souvislosti s poli
- ovládat **pole** a jeho **sekvenční zpracování**

- **=> Vše na WIKI**

Číselné proměnné a konstanty

- 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;`

Číselné proměnné a konstanty

- 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;`
  - **Přiřazení (=) vs Porovnání (==)**

Číselné proměnné a konstanty

- Základní pojmy
  - Inicializace
    - Nastavení hodnoty proměnné nebo konstanty
  - Příkaz
    - Definuje činnost, kterou program vykoná (např. výpis textu na obrazovku)
  - **Definice**
    - Příkaz, který přidělí proměnné určitého typu jméno a paměť
  - **Deklarace**
    - Příkaz, který pouze udává typ proměnné a její jméno
    - **Nepřiděluje žádnou paměť!**

Operátory

- **Aritmetické**: unární, binární
- **Relační**: ==, !=, >, <, >=, <=
- **Logické**: && (AND), || (OR)

A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

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

- **BONUS:**
  - Jaký je rozdíl mezi double a float?
    - Zkuste 0.1 + 0.1 + 0.1 + 0.1 … 0.1 = ?

Datový typ pole

Skutečná adresa	1634	1638	1642	1646	1650	1654
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	1638	1642	1646	1650	1654
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'

- **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`

Řetězec: otázky

Skutečná adresa	1634	1635	1636	1637	1638	1639
Index	0	1	2	3	4	5
Hodnota	'h'	'e'	'l'	'l'	'o'	'\0'

- Jaký je rozdíl mezi `"x"` a `'x'`? Jakou mají velikost?

- Opakování (`for` a `while`):
  - Jak projdeme pole?

Poznámka: porovnání dvou řetězců

- Lze řetězce porovnávat takto?

```
char str1[] = "Ahoj";
char str2[] = "Ahoj";

if (str1 == str2) {
/// ?????
}
```

- Co porovnáváme?
- Řešení: funkce **`strcmp`**

Úkol: Řetězce

- Deklaruje a inicializujte proměnnou **typu řetězec**
  - datový typ **`char[]`**, délka řetězce **100** znaků
- **Načtěte** jeho hodnotu hodnotu (příkaz **`scanf("%100s",…)`**)
- **Vypište** obsah (**`%s`**)

- Dále: zjistěte **délku řetězce** a délku vypište

Úkol: Znaky v řetězci

- Napište program, který v řetězci (obsah opět pomocí funkce *scanf*) spočítá, kolik obsahuje **písmen** a **číslic** (tzv. alfanumerických znaků).
  - Dvě proměnné: jedna pro počet písmen, druhá pro počet číslic

- Pozn.: ASCII

- Pozn.: ctype.h - knihovna nad znaky

Úkol: převod velká na malá

- V načteném řetězci převeďte **velká** písmena na **malá**
- *Např.: Hello World => hello world*

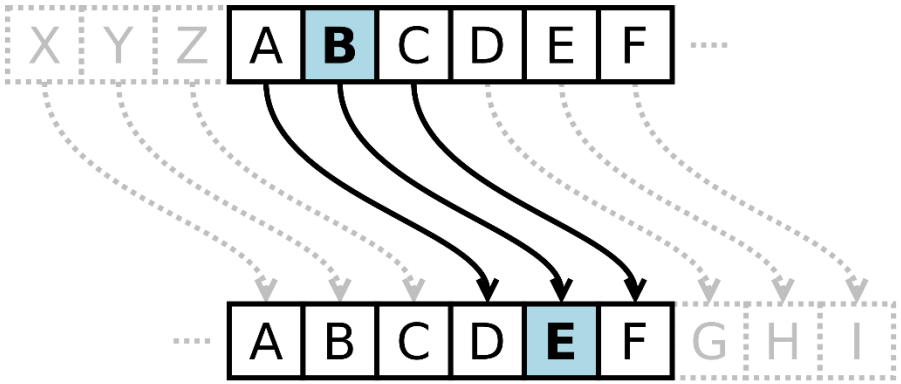
- Jaký je rozdíl mezi ’a’ a ‘A’? => **ASCII**

Úkol: nahrazení

- Nahraďte v načteném řetězci vybraný znak (první argument programu) a nahraďte **jej pomlčkou**.
  - Příklad nahrazení o:
  - *Hello World => Hell- W-rld*

- *Složitější (pouze kdo má předchozí): Caesarova šifra*
- *Př.: int posun = 3*

(protože B + 3 = E)



Úkol: Porovnání dvou řetězců

- Napište program, který porovná dva řetězce
  - **POZOR, pro porovnání řetězců nelze použít ==**
- Dva řetězce se shodují, když
  - Mají stejnou délku a
  - Všechny znaky stejné

- *Složitější: porovnání case-insensitive*

Úkol: getchar

- Načtěte celý řádek a zjistěte a vypište počet znaků v řádku pomocí getchar.

```
int ch;
while ((ch=getchar()) != EOF){
  printf("%c", ch);
}
```

- Ukončení načítání pomocí ctrl + D nebo ctrl + C

Argumenty programu

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

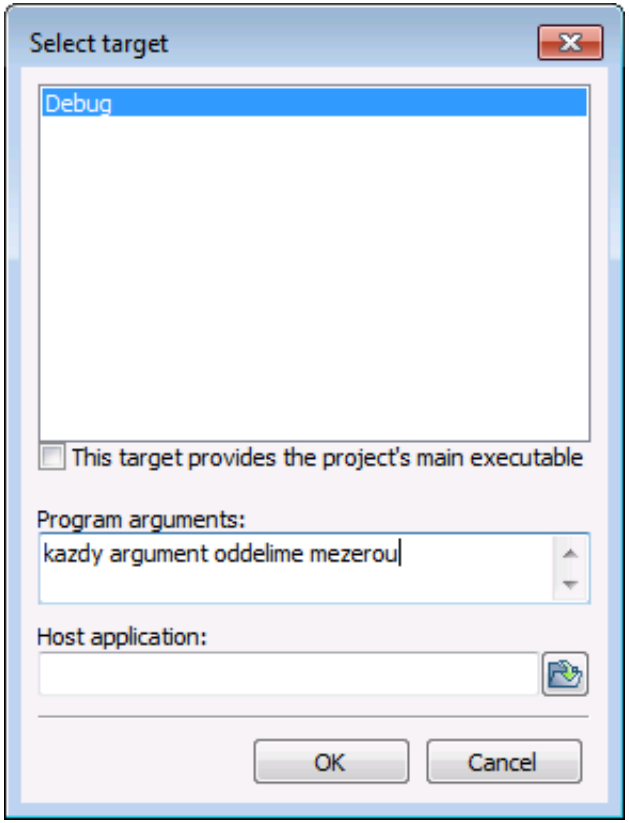
```
int main(int argc, char* argv[])
{
  // argc - počet argumentů
  // argv - jednotlivé argumenty, argv[0]
  // (název souboru s programem)
}
```

- 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

Úkol

- Vytiskněte **první** argument programu

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

- Pro `./hello 10` **argc=2**

argv[0]	argv[1]
"hello"	"10"

Řetězce – převody

- 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 **`12abcf`** není číslo, výsledek funkce `ato(i/f)` bude 12
  - Řešení: zkontrolovat, zda se řetězec skládá pouze z číslic, funkce **`strtod`**, **`strtof`** (později)

Úkol: převod na celé číslo

- Převeďte první argument programu na celé číslo a vytiskněte ho

- Nápověda:

```
#include <stdlib.h> // atoi,atof

int celeCislo = atoi("12");
float desetinneCislo = atof("3.14");
```

Děkuji za pozornost