Základy programování (IZP)

Šesté laboratorní cvičení

Vysoké učení technické v Brně, Fakulta informačních technologií v Brně
Božetěchova 2, 612 66 Brno

Gabriela Nečasová (inecasova@fit.vutbr.cz)

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**
- 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

Náplň cvičení

- **Diskuze k prvnímu projektu**
- **Opakování pojmů**
- **Řídicí struktury a jejich převod**
  - Převod mezi cykly

DISKUZE K PRVNÍMU PROJEKTU

1. projekt – rady, doporučení

- Testujte na serveru `merlin`
- Překládejte i s přepínačem `-Werror`
- **Důsledně dodržujte formát výstupu, projekt bude hodnocen automaticky**

- Pro převod z řetězce na číslo se dá s výhodou použít funkce `strtol`

Funkce vrací číslo > 0 nebo 0

- Zkontroluje, zda je znak **c** bílý znak

```
int isspace(int c);
```

- Zkontroluje, zda je znak **c** číslice

```
int isdigit(int c);
```

- Zkontroluje, zda je znak **c** písmeno

```
int isalpha(int c);
```

- Více informací o **ctype.h**
  - **man ctype.h**
  - Internet

OPAKOVÁNÍ POJMŮ

- **Funkce**

- **Funkce**
  - Funkce sdružuje příkazy v jeden celek.
  - Jedna z funkcí je funkce `main()` **a musí být v programu vždy uvedena**.
  - Je vhodné program **rozdělit na funkce**, které odpovídají jednotlivým činnostem, které má program vykonávat.

- **Deklarace funkce (prototyp funkce, hlavička)**

- **Deklarace funkce (prototyp funkce, hlavička)**
  - Deklaruje funkci před jejím použitím a před tím než je definována.

```
navratTyp JmenoFce(arg1, arg2,…,argN);
```

- **Deklarace funkce (prototyp funkce, hlavička)**
  - Deklaruje funkci před jejím použitím a před tím než je definována.

```
navratTyp JmenoFce(arg1, arg2,…,argN);
```

- **Poznámka**: Funkce `main()` prototyp nepotřebuje
  – jedná se o funkci, která má zvláštní postavení
  - Každý spustitelný program ji musí obsahovat
  - Její návratový typ je vždy `int`

- **Poznámka**: Funkce `main` má **0 nebo 2 parametry**
  - **0 parametrů**: nepotřebujeme pracovat s argumenty příkazové řádky

```
int main() {…}
```

  - **2 parametry**: potřebujeme argumenty příkazové řádky

```
int main(int argc, char* argv[]) {…}
```

- **Poznámka**: **Nezáleží na pojmenování parametrů**, důležité jsou pouze jejich datové typy! Uvedené pojmenování je ale konvence, která se používá všude

- **Definice funkce (implementace funkce)**

- **Definice funkce (implementace funkce)**
  - Uvedení toho, co má funkce dělat
  - Hlavička funkce + tělo funkce

```
navratTyp JmenoFce(arg1, arg2,…,argN)
{
    // tělo: co má funkce provádět…
}
```

- **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()`**

- **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;`

- **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;`

- **Deklarace**

- **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**

- **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
```

- **Příkaz**

- **Příkaz**
  - Příkaz definuje činnost, kterou program vykoná (výpis textu na obrazovku, opakování části programu, atp.).

- **Řídicí konstrukce (struktury) programu**

- **Řídicí konstrukce (struktury) programu**
  - Části programovacího jazyka, které řídí, jakým směrem se bude výpočet ubírat.
  - Patří sem:
    - **Funkce**
    - **Příkazy**: složený příkaz, podmíněný příkaz, cykly, příkazy skoku

- **Výraz**

Opakování pojmů

- **Výraz**
  - Konstrukce jazyka, která má hodnotu (nějakého datového typu)

- **Výraz × příkaz**
  - **Výraz**: představuje dále použitelnou hodnotu
  - **Příkaz**: jeho úkolem je vykonat nějaký kód

- **Operátory**

- **Operátory**
  - **Aritmetické**

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**: && (AND), || (OR), ! (NOT)

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

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**: && (AND), || (OR), ! (NOT)

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

  - **Relační**

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**: && (AND), || (OR), ! (NOT)

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

  - **Relační**: ==, >, <, >=, <=

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**: && (AND), || (OR), ! (NOT)

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

  - **Relační**: ==, >, <, >=, <=
  - **Bitové**

- **Operátory**
  - **Aritmetické**:
    - unární (+, -, speciální: ++, --),
    - binární (+, -, *, /, %, =)
  - **Logické**: && (AND), || (OR), ! (NOT)

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

  - **Relační**: ==, >, <, >=, <=
  - **Bitové**: výsledkem je číselná hodnota (ne logická!)
    & (bit. AND), **|** (bit. OR), **~** (bit. NOT), **^** (bit. XOR), **<<** (bit. posun vlevo), **>>** (bit. posun vpravo)

- **Homogenní/heterogenní datová struktura**

- **Homogenní/heterogenní datová struktura**
  - **Homogenní**: všechny komponenty struktury **jsou stejného** datového typu
  - Příklad: pole → `int moje_pole[6];`

  - **Heterogenní**: komponenty struktury **nejsou stejného** datového typu
  - Příklad: struktura (později ☺)

- **Cyklus**

- **Cyklus**
  - Cyklem myslíme opakování určité části programu.
  - Rozeznáváme dva základní druhy cyklů:
    - se **známým počtem průchodů (iterací) (`for`)**
    - s **neznámým počtem průchodů (iterací) (`while`, `do-while`)**

- **Příkaz break**
  - umožňuje ukončit cyklus v libovolném místě těla cyklu, zpracování programu pokračuje příkazem následujícím za cyklem

- **Příkaz continue**
  - vynutí nové vyhodnocení podmínky cyklu, přičemž se přeskočí všechny příkazy mezi ním a koncem těla cyklu

Jednoduché cykly

PŘÍKLAD 1

Příklad 1 – cykly

- Se **známým počtem** průchodů – jaký cyklus?
  - Vzestupně - interval [0;n)
  - Vzestupně - interval [a;b]
  - Sestupně - interval [a;b)
  - Vzestupně ob jedno - interval [a;b]
  - Po jednotlivých znacích řetězce
  - Po jednotlivých znacích načtených ze **`stdin`**

Příklad 1 – cykly

- Se **známým počtem** průchodů – cyklus **for**
  - Vzestupně - interval [0;n)
  - Vzestupně - interval [a;b]
  - Sestupně - interval [a;b)
  - Vzestupně ob jedno - interval [a;b]
  - Po jednotlivých znacích řetězce
  - Po jednotlivých znacích načtených ze **`stdin`**

Časté chyby

- Pozor na = a ==

```
for(int i=0; i=5; i++){ … }
```

- Pozor na pořadí jednotlivých částí cyklu **for**

```
for(int i=0; i++; i<5){ … }
for(int i=5; i--; i>0){ … }
```

Převody mezi cykly

PŘÍKLAD 2

Příklad 2 – převody mezi cykly

- Napište program, který **projde všechny argumenty** příkazové řádky a zjistí, zda byl zadán **argument `–h`**
  - Pokud byl zadán, ihned ukončete provádění cyklu
  - Použijte cyklus **`while`** a příkaz **`break`**

Příklad 2 – převody mezi cykly

- Napište program, který **projde všechny argumenty** příkazové řádky a zjistí, zda byl zadán **argument `-h`**
  - Pokud byl zadán, ihned ukončete provádění cyklu
  - Použijte cyklus **`while`** a příkaz **`break`**

- Modifikace 1: převeďte s cyklem **`while`** bez použití **`break`** a **`continue`**

Příklad 2 – převody mezi cykly

- Napište program, který **projde všechny argumenty** příkazové řádky a zjistí, zda byl zadán **argument `-h`**
  - Pokud byl zadán, ihned ukončete provádění cyklu
  - Použijte cyklus **`while`** a příkaz **`break`**

- Modifikace 1: převeďte s cyklem **`while`** bez použití **`break`** a **`continue`**

- Modifikace 2: přepište cyklus **`while`** na **`for`**

Příklad 2 – převody mezi cykly

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

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

Příklad 2 – převody mezi cykly

- Napište program, který **projde všechny argumenty** příkazové řádky a zjistí, zda byl zadán **argument `-h`**
  - Pokud byl zadán, ihned ukončete provádění cyklu
  - Použijte cyklus **`while`** a příkaz **`break`**

- Modifikace 1: převeďte s cyklem **`while`** bez použití **`break`** a **`continue`**

- Modifikace 2: přepište cyklus **`while`** na **`for`**

Převody mezi cykly II

PŘÍKLAD 3

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h				

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e			

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l		

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o
2	w				

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o
2	w	o			

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o
2	w	o	r		

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o
2	w	o	r	l	

Příklad 3 - Převody mezi cykly II

- Zjistěte, zda **libovolný argument** programu **obsahuje písmeno h**.
  - Do **proměnné i** uložte **pozici argumentu** obsahující písmeno h
  - Do **proměnné j** uložte **pozici písmene** v daném argumentu

```
./prog hello world
```

argv[1] = hello argv[2] = world

Index argumentu i	argv[i][0]	argv[i][1]	argv[i][2]	argv[i][3]	argv[i][4]
1	h	e	l	l	o
2	w	o	r	l	d

Z minulého cvičení

DOMÁCÍ ÚKOLY

Implementuje funkci, která ověří, že **číslo `x` patří do intervalu [`a`,`b`]**.

Funkce vrátí

- 1 pokud se číslo **`x`** nachází v intervalu [**`a`,`b`**]
- 0 jinak

Implementujte funkci, která **ověří, zda je číslo dělitelné jiným číslem beze zbytku**. Pokud ano, funkce vrací 1, jinak 0.

Nápověda: k řešení použijte operátor modulo (zbytek po celočíselném dělení), v C **%**

10 % 5 = 0

11 % 6 = 5

atp…

Implementujte funkci, která **vypočítá absolutní hodnotu** čísla typu **`double`** a vypočítanou absolutní hodnotu vrátí.

Implementujte funkci, která vybere **maximální hodnotu** z pole typu `int`, které má `n` členů. Funkce maximální hodnotu vrátí.

Napište funkci, která vypočítá **n-tý prvek Fibonacciho posloupnosti**.

Fibonacciho posloupnost je definována:

f(0) = 0

f(1) = 1

f(n) = f(n-1) + f(n-2)

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, atd.

Děkuji Vám za pozornost!