

Funkce s proměnným počtem argumentů

Při každém volání funkce jsou hodnoty skutečných parametrů kopírovány na zásobník (paměť používaná pro uložení lokálních proměnných a parametrů funkcí).

Pokud známe adresu a typ (velikost paměti potřebné k jejímu uložení), můžeme přistoupit také k následujícímu parametru.

Funkce s proměnným počtem parametrů musí mít alespoň jeden pevný parametr a musí být určeno, kalik parametrů v paměti následuje (je znám počet, nebo je použita nějaká zarážka) a jaké jsou typy těchto parametrů.

Tyto informace mohou být zadány následujícími způsoby:

- počet argumentů a jejich typy předány formátovacím řetězcem (např. `printf`)
- předpokládáme typ parametrů a funkci je předán jejich počet (např. `suma`, `průměr`)
- předpokládáme typ parametrů a máme určenou zarážku (např. práce s řetězcí)

1 Deklarace

Používá se výpustka `...`, která následuje po povinných argumentech.

```
typ nazev(typ povinny, ...)
```

Příklad použití:

```
double prumer(int pocet, double prvni, ...)
{
    /* Telo funkce */
}
```

```
prum = prumer(5, 1.2, 3.4, 5.6, 7.8, 9.0);
```

2 Zpracování parametrů v těle funkce

Díky knihovně `stdarg.h` nemusíme znát, jak je implementován zásobník.

V knihovně je implementován:

- typ `va_list`, který se používá k uložení parametrů v zásobníku
- makro `va_start()`
- makro `va_arg()`
- makro `va_end()`

V těle funkce je nutné deklarovat alespoň jednu proměnnou typu `va_list` a tu nastavit na první z nepovinných argumentů pomocí makra `va_start()`.

`va_start()` bere jako argument identifikátor nastavované proměnné typu `va_list` a identifikátor posledního povinného argumentu.

```
va_list parametry;
```

```
va_start(parametry, posledni_povinny);
```

Jednotlivé parametry pak získáváme prostřednictvím makra `va_arg()`, jehož argumenty jsou ukazatele na zásobník a očekávaný datový typ dalšího parametru.

Vyhodnocení `va_arg()` má za následek i posunutí ukazatele na zásobníku.

```
cislo = va_arg(parametry, double);
```

Pokud jsme získali všechny nepovinné argumenty, je potřeba ukončit práci s ukazatelem na zásobník pomocí makra `va_end()`.

```
va_end(parametry);
```

3 Příklad

Celý kód na výpočet průměru by mohl vypadat následovně:

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

double prumer(int pocet, ...)
{
    va_list parametry;
    va_start(parametry, pocet);
    double soucet = 0;
    int i;

    for(i = 0; i < pocet; i++)
    {
        soucet = soucet + va_arg(parametry, double);
    }
    va_end(parametry);

    return soucet/pocet;
}

int main(int argc, char* argv[])
{
    printf("%f \n", prumer(5, 1.2, 3.4, 5.6, 7.8, 9.0));
    return 0;
}
```

4 Cvičení

1. Napište funkci, která vrací nejmenší z předaných celočíselných parametrů. Funkce bere jako první argument počet předaných celých čísel.
2. Napište funkci `komplexni suma(int pocet, ...)`, která vypočítá součet předaných komplexních čísel. Počet sčítaných čísel je určen pevným parametrem `pocet`, za nímž pak ve volání funkce následují hodnoty, které má funkce sčítat. Pro práci s komplexními čísly je nutné vytvořit strukturovaný datový typ `komplexni`.

Argumenty příkazové řádky

Funkce `main()` je funkce, která je spuštěna při startu programu. I této funkci mohou být předány argumenty, s níž je funkce zavolána. Jsou to 2 argumenty:

- `argc` = argument count = počet argumentů s kolika je příkaz spuštěn
- `argv` = argument vector = ukazatel na pole znakových řetězců obsahujících argumenty. Vždy jeden parametr v jednom řetězci.

Hlavička funkce `main()` pak vypadá takto:

```
int main(int argc, char *argv[]);
```

Jelikož je funkce spuštěna automaticky, o jejich naplnění se stará zavaděč operačního systému. Tedy jsou naplněny už při spouštění. Motivací k využití parametrů funkce `main()` je, že ne vždy je vhodná interakce s uživatelem při běhu programu a možnost spouštění programu s parametry nám poskytuje nástroj k tomu, jak spouštět úlohy dávkově.

Příkladem by mohl být program na sčítání 2 čísel. Nám doposud známý způsob je takový, že v kódu `main` použijeme funkci `scanf`, díky které získáme od uživatele 2 čísla, která následně sečteme a vypíšeme výsledek.

Způsob, o kterém mluvíme teď, je takový, že zkompilujeme program (tím vytvoříme např. soubor `program.exe`), ten pak z příkazové řádky spustíme se 2 parametry (`program.exe 2 3`). Tyto 2 parametry se ve funkci sečtou a vypíše se výsledek.

Při tomto spuštění (`program.exe 2 3`) bude v proměnné `argc` uložena hodnota 3 a první tři položky `argv` budou obsahovat tyto řetězce:

```
argv[0] = "program.exe"
```

```
argv[1] = "2"
```

```
argv[2] = "3"
```

Jednotlivé parametry jsou odděleny mezerami. Pokud bychom potřebovali, aby argument obsahoval mezeru, pak jej napíšeme do uvozovek.

Spustíme-li tedy program následujícím způsobem:

```
program.exe 2 "3 4"
```

Budou jednotlivé prvky pole `argv` následující:

```
argv[0] = "program.exe"
```

```
argv[1] = "2"
```

```
argv[2] = "3 4"
```

5 Příklad

Ukážeme si to na jednoduchém příkladu.

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;

    for(i=0; i<argc; i++)
    {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Takto definovaná funkce po spuštění vypíše celé pole `argv`, tedy název programu a všechny argumenty, se kterými byl volán. Každý z nich bude na novém řádku.

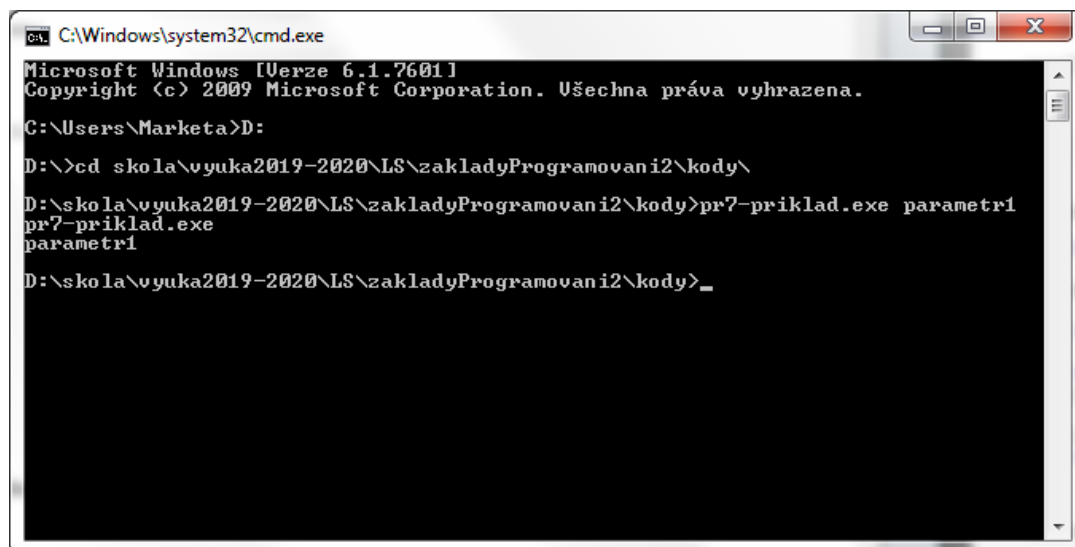
5.1 Spuštění

Po přeložení kódu se ve složce, kde je uložený vytvoří soubor, který se jmenuje stejně jako kód s příponou `.exe`.

Spusťte příkazovou řádku (spustit → `cmd`).

Přesuňte se pomocí příkazu `cd` do složky, kde máte soubor uložený.

A spusťte tím, že napíšete název souboru a parametry.



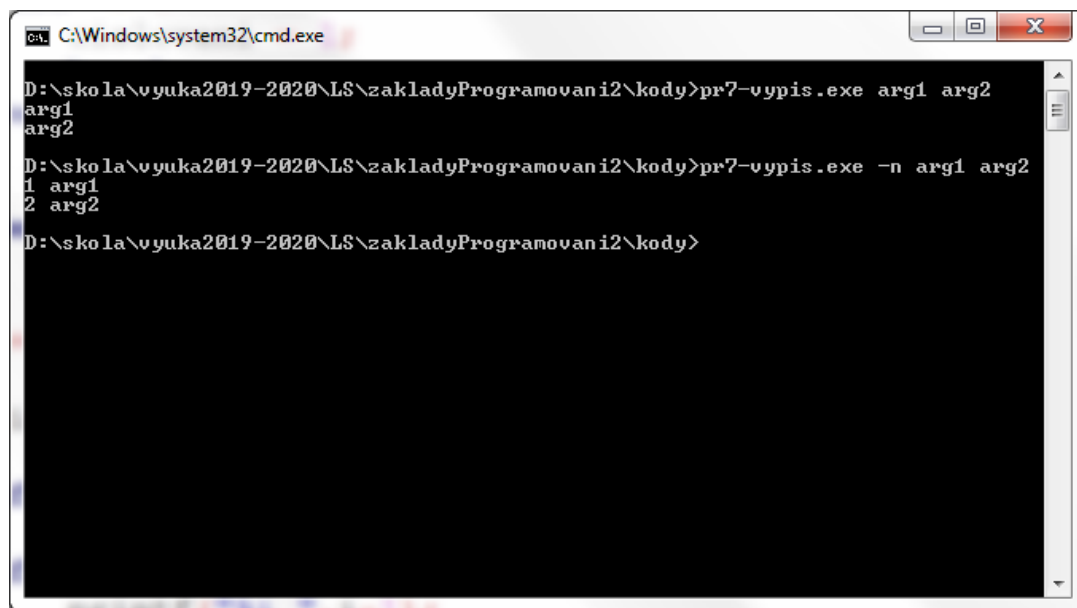
Upravte kód tak, aby vypisoval pouze předané argumenty.

6 Příklad 2

Běžnou konvencí programů v jazyce C (obzvláště pod systémem UNIX) je, že argument, který začíná znaménkem `-` uvádí volitelný parametr.

Upravíme tedy předchozí kód tak, že nebudeme pokud bude prvním argumentem -n, tak se jednotlivé argumenty očísloví (tento argument se nevypíše).

Výpisy by vypadaly následovně:



```
C:\Windows\system32\cmd.exe
D:\sko la\vyuka2019-2020\LS\zakladyProgramovani2\kody>pr7-vypis.exe arg1 arg2
arg1
arg2
D:\sko la\vyuka2019-2020\LS\zakladyProgramovani2\kody>pr7-vypis.exe -n arg1 arg2
1 arg1
2 arg2
D:\sko la\vyuka2019-2020\LS\zakladyProgramovani2\kody>
```

7 Cvičení

1. Modifikujte předchozí příklad tak, aby rozlišoval volitelné argumenty -n (vypíše čísla řádků) a -o (přidá před každý argument odrážku).
2. Napište program **soucet**, který sečte čísla zadaná v příkazové řádce. Každé číslo je samostatný argument. Například:
soucet 2 3 4
vrátí 9
Pozor! Jednotlivé argumenty jsou textové řetězce. Ty je potřeba převést na čísla.
3. Napište program **vyraz** vyhodnocující výpočty zapsané v obrácené polské notaci (postfixová notace) a zadané z příkazové řádky, kde každý operand nebo operátor je samostatným argumentem. Například: **vyraz 2 3 4 + ***
vypočítá výraz $2 \cdot (3 + 4)$
Nápověda: Pro jednoduchost budeme používat jen operace +, -, * a / a budeme předpokládat, že berou vždy 2 argumenty. Postup, jak se výraz vyhodnocuje je například na wikipedii.
4. Napište program **nejdelší**, který vrátí nejdelší (obsahující nejvíce znaků) ze svých argumentů. Například: **nejdelší ahoj jak se mas**
vypíše ahoj

8 Povinná úloha

Tuto úlohu je potřeba odevzdat do konce měsíce března (tedy do 31.3. 2020), pokud z jakéhokoliv důvodu (nemoc, pomoc jiným, ...) nemůžete tuto úlohu v řádném termínu splnit, napište mi a domluvíme se na prodloužení tohoto termínu.

Úlohu odevzdávejte jako soubor s příponou .c na e-mail marketa.trneckova@gmail.com

(případně marketa.trneckova@upol.cz).

Napište funkci `my_printf()`, která se bude chovat obdobně jako funkce `printf()`. Bude mít 1 povinný argument - řetězec, který může obsahovat formátovací sekvence - a libovolný počet nepovinných argumentů.

Formátovací sekvence:

- *i nahradí celým číslem, který byl předán jako nepovinný argument
- *c nahradí znakem
- *f nahradí číslem s desetinnou čárkou

Pozor! funkce `va_arg()`, pracuje jen se základními datovými typy. Znak (`char`), je potřeba načíst jako `int` a přetypovat na `char`

```
(char)va_arg(parametry, int)
```

a desetinná čísla je potřeba načíst jako typ `double`

```
va_arg(parametry, double)
```

Uvnitř funkce je možné použít funkci `printf()` a jiné funkce, které jsme používali.

Není potřeba provádět kontrolu, zda je funkce volána s korektními vstupy.