Informační a komunikační technologie

Funkce v C

David Weber
Kabinet K13

weber3@spsejecna.cz

Příklad na úvod

```
// Print array
for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
}
printf("\n");
// Square array elements
for (int i = 0; i < size; i++) {
    arr[i] *= arr[i];
}
// Print array
for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
}
```

V čem je problém?

- Kód je funkční, ale část pro výpis pole je zde uvedena **dvakrát**.

```
for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
}
```

- Opakující kód je nepraktický ✗
  - budeme-li chtít změnit nějakou jeho část, musíme změnu provést všude
- Použijeme funkci ✓

Co je to funkce?

- Obecně se jedná o část programu, kterou je možné opakovaně "vyvolat" v různých místech programu.
- Motivace pro použití:
  - odstranění opakování kódu v programu,
  - rozklad složitých problémů na jednodušší,
  - znovupoužití v jiných programech, např. formou knihoven (to až ve druháku ).

Struktura funkce

```
navratovy_typ jmeno(parametry) {
    <implementace>
}
```

- U funkce je třeba specifikovat:
  - návratový datový typ,
  - jméno (identifikátor),
  - parametry,
  - tělo (implementace).
- Funkce uvádíme mimo tělo funkce `main`.

Funkce bez parametrů

- Nejjednodušší typ funkce.
- Klíčové slovo `void` (prázdný datový typ), do závorek `(void)` nebo nechat prázdné, tj. `()`

```
void greet(void) {
    printf("Hello World!");
}
```

- **Doporučení:** funkci uvádějte nad `main`!
- Samotná definice funkce nic nedělá ⇒ je třeba ji tzv. **zavolat**.

Volání funkce

- Je třeba specifikovat, kde v programu se má daná funkce provést.

```c
void greet(void) {
    printf("Hello World!");
}

int main(void) {

    greet();    // Prints out "Hello World!"

    return 0;
}
```

Funkce s parametry

- Často budeme chtít činnost funkce zobecnit (výstup funkce bude na něčem záviset).
- $\Rightarrow$ k tomu použijeme tzv. **parametry**.
- Do kulatých závorek `()` uvádíme výčet parametrů (jejich datový typ a název), které funkce přijímá.

```
void fce(<datovy_typ> var1, <datovy_typ> var2, ...) {
    ...
}
```

Příklad I (funkce s parametrem)

```c
void square(float x) {
    printf("Druha mocnina x je %g", x*x);
}

int main(void) {

    float input;
    scanf("%f", &input);

    square(input);

    return 0;
}
```

Příklad II (funkce s více parametry)

Funkce pro výpočet aritmetického průměru celých čísel $a, b \in \mathbb{Z}$:

```
void average(int a, int b) {
    float avg = (a + b) / 2;
    printf("%g", avg);
}

int main(void) {
    float input1, input2;
    scanf("%d %d", &input1, &input2);

    average(input1, input2);

    return 0;
}
```

Úloha I

Předělejte program pro výpis faktoriálu, tj.

$$n! = n(n-1)(n-2)\cdots 2 \cdot 1,$$

do funkce. (Viz domácí úkol 🙂)

Funkce s návratovou hodnotou I

- **Problém:** s hodnotou, kterou funkce vypočte, nemůžeme dále pracovat (pouze ji na konci vypisujeme).
- Např. u funkcí `pow`, `sqrt`, ...(viz knihovna **math**) jsme mohli s vypočtenou hodnotou dále počítat

```
float hypotenuse = sqrt(pow(x, 2) + pow(y, 2));
printf("%f", hypotenuse);
```

- $\Rightarrow$ využijeme tzv. **návratovou hodnotu**.

Funkce s návratovou hodnotou II

- Klíčové slovo `return`.
- Před názvem funkce musíme dále uvést datový typ návratové hodnoty funkce.
- Např. funkce pro výpočet faktoriálu vracející výsledek:

```
int factorial(int n) {
    int fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
```

Úloha II

Napište vlastní funkci pro výpočet n-té mocniny čísla $x \in \mathbb{R}$. Funkce bude přijímat parametry x, n a bude vracet x^n. Nesmíte použít funkci `pow`. 🙃

Co se děje uvnitř?

- Při volání funkce program vykoná její kód a následně se vrátí do místa jejího volání, odkud pokračuje.
- $\Rightarrow$ jak ale ví, odkud má pokračovat? Jaké byly stavy proměnných při volání?

Příklad

- Program začne ve funkci `main`, kde inicializuje proměnné `a` a `b`,
- při zavolání funkce `mult` je navrácena hodnota $x \cdot y$ a vrátíme se zpět do místa volání (tj. do funkce `main`),
- $\Rightarrow$ je třeba vědět, do které části kódu se přesunout! $\Rightarrow$ návratová adresa.

```
int mult(int x, int y) {
    return x*y;
}
int main(void) {
    int a = 10;
    int b = 47;
    printf("%d", mult(a,
    b));
    return 0;
}
```

Zásobník

- **Návratová adresa** – adresa v paměti, odkud má program pokračovat po skončení volané funkce.
- Při volání funkce a návratu z ní se využívá tzv. **zásobník** (angl. *stack*).
- Do něj se ukládá (mimo jiné) **při volání funkce** *návratová adresa*, *předávané parametry* ***volané*** *funkce*.
- Dále jsou zde ukládány i *hodnoty lokálních proměnných* při deklaraci.

Rozdělení paměti

- **Kód programu** – zkompilovaný (strojový) kód programu
- **Statická data** – mají svou velikost přesně a neměně určenou v okamžiku překladu zdrojového textu (globální proměnné)
- **Zásobník** – lokální proměnné, argumenty funkcí, návratová hodnota funkce, *návratová adresa*
- **Halda** – dynamická paměť, alokována za běhu programu (např. v případě pole)

Příklad (stav zásobníku)

```
float sqr(float x) {
    return x*x;
}
float mult(int x, int y) {
    int c = 30;
    return c*sqr(x)*sqr(y);
}
int main(void) {
    int a = 20;
    float b = 23.5f;
    printf("%d", mult(a, b));
    return 0;
}
```

Otázky?