Jazyk C

Funkce

Mgr. Markéta Trnečková, Ph.D.

Palacký University, Olomouc

Funkce

- předem definované
- uživatelsky definované

Funkce

- vstupní data – argumenty funkce
- návratová hodnota

```
typ jmeno(typ1 a1, typ2 a2, ..., typn an)
```

- funkce, která nic nevrací – procedura

Funkce – příklad

Example

```
/* Vypocet mocniny s prirozenym exponentem */
double mocnina(double zaklad, int exponent){
    double vysledek = 1.0;
    int i;

    for(i = 0; i<exponent; i++){
        vysledek *= zaklad;
    }
    return vysledek;
}
```

```
double y = mocnina(2,3);
```

„Přetypuješ každý parametr předávaný funkci na očekávaný typ, pokud toho typu již není, ačkoli jsi přesvědčen o zbytečnosti svého úsilí. Odplata krutá totiž nečekaně zaskočí ty, kdo opomenou.“

(Henry Spencer)

Funkce – deklarace

Example

```
void funkceA(){
    funkceB();
}

void funkceB(){
    funkceC();
}

void funkceC(){
    funkceA();
}
```

Funkce – deklarace

Example

```c
double mocnina(double zaklad, int exponent);
/* pripadne */
double mocnina(double, int);
```

Rekurzivní funkce

Example

```
#include <stdio.h>
int fakt(int n){
    if(n <= 0)
        return 1;
    return n * fakt(n - 1);
}

int main(){
    int n = 10;
    printf("Faktorial cisla %d je roven %d", n, fakt(n));
    return 0;
}
```

Předávání argumentů funkci

Example

```c
#include <stdio.h>

void zmen_cislo_na_5(int cislo){
    cislo = 5;
}

int main(){
    int cislo = 4;
    zmen_cislo_na_5(cislo);
    printf("%d ", cislo);
    return 0;
}
```

Předávání argumentů funkci

- Předávání argumentů hodnotou
- Předávání argumentů odkazem

Předávání argumentů odkazem

Example

```
int deleni(int a, int b, int *r){
    /* r - zbytek po deleni */
    *r = a % b;
    return a / b;
}

int main(){
    int x, y;
    /* x = 2, y = 3*/
    x = deleni(13, 5, &y);
    return 0;
}
```

Pole jako argument funkce

Example

```c
#include <stdio.h>

void vynuluj_pole(int p[], int velikost){
    int i;
    for(i = 1; i < velikost; i++){
        pole[i] = 0;
    }
}

int main(){
    int pole[5] = {1, 2, 3, 4, 5};
    vynuluj_pole(pole, 5); /* Vsechny prvky rovny 0 */
    return 0;
}
```

Rozsah platnosti proměnných

- lokální proměnné
- globální proměnné

Example

```
int funkceA(){
    int vysledek = 1;
    /* Zbytek funkce*/
}

int funkceB(){
    int vysledek = 10;
    /* Zbytek funkce*/
}
```

Rozsah platnosti proměnných

Example (Jaký bude výsledek tohoto programu?)

```c
int foo = 5; /* Globalni promenna*/

void funkceA(){
    int foo = 6;
    printf("%d ", foo);
}

void funkceB(){
    foo = 7;
    printf("%d ", foo);
}

int main(){
    printf("%d ", foo);
    funkceA();
    funkceB();
    printf("%d ", foo);
    return 0;
}
```

Paměťové třídy

- `auto`
- `extern`
- `static`
- `register`

Paměťové třídy – auto

```c
void func(){
    int j;
    ...
}
/* ma stejny vyznam jako */
void func(){
    auto int j;
    ...
}
```

Paměťové třídy – extern

```c
/* Soubor 1 */
typ promenna; /* definice/deklarace */

/* Soubor 2 */
extern typ promenna; /* deklarace */
```

Paměťové třídy – static

```
void f(){
    int x = 0;
    static int y = 0;

    y++;
    x++;
}
```

Paměťové třídy – register

```
register typ promenna;
```

Typové modifikátory

- `const`
- `volatile`

```
pametova_trida typovy_modifikator typ nazev_promenne;
```

Funkce s proměnným počtem argumentů

Výpustka – tři tečky `...`

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

- počet argumentů a jejich typy jsou předány formátovacím řetězcem (např. jako u `printf()`),
- předpokládáme typ parametrů a funkci je předán jejich počet,
- předpokládáme typ parametrů a máme určenou zarážku (např. práce s řetězci).

Funkce s proměnným počtem argumentů

Example

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

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

Funkce s proměnným počtem argumentů

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

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

cislo = va_arg(parametry, double);

ve_end(parametry);
```

Funkce s proměnným počtem argumentů

Example

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

double prumer(int pocet, ...){
    double soucet = 0;
    int i;
    va_list parametry;
    va_start(parametry, pocet);
    for(i = 0; i < pocet; i++){
        soucet += va_arg(parametry, double);
    }
    va_end(parametry);
    return soucet / pocet;
}

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

Parametry příkazové řádky

- `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.

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

Parametry příkazové řádky – Příklad

- Součet dvou čísel – `soucet.exe`
- Spuštění v konzoli – `soucet.exe 2 3`
- `argc` $= 3$
- `argv`:
  `argv[0] = ''soucet.exe''`
  `argv[1] = ''2''`
  `argv[2] = ''3''`

Parametry příkazové řádky – Příklad

Example

```c
#include <stdio.h>

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

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

Parametry příkazové řádky – Příklad

Ukazatele na funkce

```
typ (*nazev)(typ1, typ2, ..., typn);
```

Example

```
double polynomA(double x){
    return 3*x*x + 2*x + 1;
}

double (*ptr_polynom)(double) = polynomA;
```

```
typedef double (*PTR_FUN)(double);

PTR_FUN ptr_polynom = polynomA;
```

Ukazatele na funkce

Přiřazení adresy ukazateli

```
ptr_polynom = polynomA;
```

Volání funkce

```
v = ptr_polynom(-1);

/* pripadne */
v = (*ptr_polynom)(-1);
```

Ukazatel na funkci jako parametr funkce

funkce vyšších řádů

```
int *map(int (*fce)(int), int *vstup, int pocet){
    /* telo funkce */
}
```

Volání funkce

```
/* Funkce vracejici treti mocninu prvku x */
int na3(int x){
    return x * x * x;
}

pole_vysledku = map(na3, pole_vstupni, velikost_pole);
```

Pole ukazatelů na funkce

```c
/* Deklarace */
int (*pole_fci[5])(int);

/* Deklarace + inicializace */
int(*pole_fci[5])(int) = {na0, na1, na2, na3, na4};

/* Volani */
vysledek = pole_fci[1](-1);
```

Cvičení

1. Napište program, který nalezne všechna čtyřciferná čísla, která jsou dělitelná číslem, které dostaneme jako sumu čísla tvořeného první a druhou číslicí a čísla tvořeného třetí a čtvrtou číslicí. Například číslo 3417 je dělitelné číslem `34 + 17`. Vhodné části algoritmu realizujte pomocí funkcí.
2. Napište program, který pro dané přirozené číslo spočítá jeho rozdíl od nejbližšího většího prvočísla.
3. Napište program, který pro dané přirozené n spočítá sumu:

$$1! + (1 + 2)! + (1 + 2 + 3)! + ... + (1 + 2 + ... + n)!$$

4. Naprogramujte funkci, která bere jako argument dvě proměnné a provede výměnu hodnot těchto dvou proměnných.
5. Naprogramujte funkci, která jako argument bere pole čísel a vrátí (nové) pole jejich druhých mocnin (tedy prvek na indexu `i` vráceného pole bude mít hodnotu druhé mocniny prvku na indexu `i` vstupního pole).

Cvičení

6. Napište funkci, která bere dva argumenty (`text` a `podretezec`). Funkce v daném textovém řetězci `text` vyhledá první výskyt zadaného podřetězce `podretezec`. Funkce vrací ukazatel na první znak nalezeného podřetězce nebo konstantu `NULL`, pokud podřetězec `podretezec` nebyl nalezen.
7. Napište funkci `my_printf()`, která se bude chovat obdobně jako funkce `printf()`. Bude mít jeden 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 desetinným číslem.

   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()`.

Cvičení

8. Napište funkci, která vrací nejmenší z předaných celočíselných parametrů. Funkce bere jako první argument počet předaných celých čísel.
9. Napište program `soucet`, který sečte celá čísla zadaná v příkazové řádce (terminálu). Každé číslo je samostatný argument. Například: `soucet 2 3 4` vrátí 9. Pozor! Jednotlivé argumenty jsou textové řetězce. Ty je potřeba převést na čísla.(Můžete použít funkci `atoi()`, která je součístí standardní knihovny `stdlib`.)
10. Napište program `nejdelsi`, který vrátí nejdelší (obsahující nejvíce znaků) ze svých argumentů předaných přes příkazovou řádku (terminál). Například: `nejdelsi ahoj jak se mas`
vypíše `ahoj`
11. Naprogramujte funkce `na0`, `na1`, `na2`, `na3` a `na4`, vytvořte pole, do kterého tyto funkce uložíte, a vyzkoušejte si práci s tímto polem.

Cvičení

12 Dopište následující program tak, že proměnné `i` a `j` definujete nejprve jako globální `int` a pak lokální `register int`. Porovnejte rychlost obou programů.

Example

```
for (i = 0; i < MAX; i++){
    for(j = 0; j < MAX; j++){
        i = i;
        j = j;
    }
}
```