Funkce

Základy programování 1

Martin Kauer

Funkce

- Relativně samostatná část programu
- Je ji možné opakovaně volat z různých částí programu
- Má vstupy = parametry funkce
- Má výstup = návratová hodnota funkce, může být „žádná“
- Příklady: `main`, `printf`, `scanf`, …

Vytvoření funkce

- Funkci nelze definovat uvnitř jiné funkce (včetně `main`)
- Funkci je potřeba definovat dříve, než ji voláte
- Obecně:

```
typ_výsledku jméno_funkce(typ_p1 jméno_p1, ... , typ_pN jméno_pN){
    příkazy_těla_funkce
    ...
}
```

- V těle funkce se mohou využívat identifikátory parametrů
- Parametry se chovají jako lokální proměnné
- Tělo funkce může obsahovat příkaz `return` pro nastavení výsledku
- Příkaz `return` ihned funkci ukončí

Příklad vytvoření funkce

- Funkce bez návratové hodnoty:

```
void vypis_max(int prvni, int druhe){
    int max;
    max = (prvni < druhe) ? druhe : prvni;
    printf("Maximum je %d.\n", max);
    //return;
}
```

- Funkce s výsledkem:

```
int maximum(int prvni, int druhe){
    int max;
    max = (prvni < druhe) ? druhe : prvni;
    return max;
}
```

Použití funkce

- Pomocí operátoru volání funkce (kulaté závorky)
- Uvnitř závorek uvádíme hodnoty parametrů = argumenty
- Návratovou hodnotu můžeme, ale nemusíme použít
- Příklady:

```
int main(){
    int a = 5, b = 10;
    int vetsi = maximum(b, 3 * a);
    printf("Max: %d\n", maximum(vetsi, a + b));
    vypis_max(a*b, a + b);
    return 0;
}
```

Funkce pracující s polem

- Pole lze předat do funkce jako vstup
- Zatím ho ale neumíme vrátit z funkce jako výsledek
- Pozor, změny provedené v poli se projeví i mimo funkci
- Příklad:

```
void vypis_pole_cisel(int pole[], int delka) {
    for (int i = 0; i < delka; i++) printf("%i, ", pole[i]);
}

int main() {
    int cisla[] = {1, 2, -7, 14};
    vypis_pole_cisel(cisla, 4);
    return 0;
}
```

Příklad s modifikací pole

```
#define DELKA 3
void plus10(int vstup[], int pocet){
    for (int i = 0; i < pocet; i++){
        vstup[i] += 10;
    }
}

int main(){
    int pole[] = { 10, 20, 30 };
    plus10(pole, DELKA);
    for (int i = 0; i < DELKA; i++){
        printf("%d, ", pole[i]);
    }
    return 0;
}
```

Deklarace funkce

- Hodí se především u větších programů
- Je komplikované funkce definovat dříve, než jsou použity
- Někdy je toto dokonce nemožné
- Deklarace = popis funkce dostačující k tomu, aby překladač věděl, jak bude funkce používána
- Uvádí se zpravidla na začátku programu
- Udává jméno funkce, typy parametrů a návratové hodnoty
- Obecně:

```
typ_výsledku jméno_funkce(typ_p1 jméno_p1, ...);
typ_výsledku jméno_funkce(typ_p1, ...);
```

Příklad s deklarací

```
int soucet(int, int);
void novy_radek();

int soucet(int a, int b) {
    int soucet = a + b;
    printf("soucet a + b = %i", soucet);
    novy_radek();
    return soucet;
}

void novy_radek() {
   printf("\n");
}
```

Rozsah platnosti - lokální proměnná

- Definována uvnitř bloku (funkce, cyklus, …)
- Vzniká při vstupu programu do bloku
- Existuje po dobu vykonávání příkazů v bloku
- Zaniká při opuštění bloku
- Příklad:

```
// tady proměnná znak ještě neexistuje
while (text[i] != '\0'){
    char znak = text[i] - 'A' + 'a';
    printf("%c\n", znak);
}
// tady proměnná znak už neexistuje
```

Rozsah platnosti - globální proměnná

- Definována mimo jakékoli bloky
- Vzniká spuštěním programu
- Přístupná ze všech míst programu
- Lokální identifikátory mohou překrýt ty globální (i jiné lokální)
- Zaniká při ukončení programu
- Globálních proměnných používáme co nejméně
- Příklad:

```
int pocet = 10;  // proměnná přístupná odkudkoli

int main(){ ... }
...
```

Rozsah platnosti - statická proměnná

- Definujeme uvnitř bloku
- Pomocí klíčového slova `static`
- Vzniká (a je inicializována) při prvním vstupu do bloku
- Přístupná je pouze z daného bloku
- Zaniká ale až ukončením programu
- Příklad:

```
int porovnej(int prvni, int druhe){
    static int pocet = 0;
    pocet++;
    if (prvni < druhe) return -1;
    if (prvni == druhe) return 0;
    return 1;
}
```

Poznámky

- Funkce **musí** vždy vracet hodnotu daného typu, tzn. zkontrolujte všechny větve funkce. Výjimkou jsou funkce navracející `void`.

```
int foo(a)
{
     if (a > 0)
     {    ...
          return 1;
     }
     else if (a == 0)
     {    ...
          return 0;
     }
     ...
     return 1;
}
```

Poznámky

- Proměnné definované uvnitř funkce nelze vidět „zvenku“ funkce. Jejich rozsah platnosti je pouze tělo dané funkce.

```
void foo()
{
     int i = 7; //začíná rozsah platnosti proměnné i
     ...
} //končí rozsah platnosti proměnné i

int main()
{
     foo();
     i++; //proměnná i tady vůbec neexistuje
     return 0;
}
```

Cvičení

Vytvořte fci s předpisem `int search(const char input[], const char pattern[])`, která bude zleva hledat první výskyt řetězce *pattern* v řetězci *input* a vrátí index znaku řetězce *input*, kde tento podřetězec začíná. Pokud *input* neobsahuje *pattern* jako podřetězec, fce musí vrátit hodnotu **-1**.

Podmínky na vypracování (diskvalifikační chyby):

1. Fce nesmí žádným způsobem modifikovat vstupní parametry *input* a *pattern*.
2. Fce nesmí nic vypisovat na obrazovku ani obsahovat žádná blokující volání typu scanf(…); getch(); system(“pause”); apod.
3. Nesmíte používat žádné fce z knihovny string.h.

Příklady:

```
search(“abeceda”, “ceda”) -> 3
search(“abeceda”, “be”) -> 1
search(“abeceda”, “deda”) -> -1
search(“abeceda”, “”) -> 0
search(“abeceda”, “aabe”) -> -1
search(“abc”, “abcd”) -> -1
```