Funkce a vnořené cykly

IZP-cv04

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

11. října 2024

Funkce

Funkce

Zdrojový kód programu se skládá z podprogramů (funkcí):

- Funkce nám umožňují část programu volat opakovaně a s různými parametry.
- Hlavička funkce obsahuje návratový typ, identifikátor a seznam parametrů.
- Funkce končí příkazem – `return` – který definuje její návratovou hodnotu.

Jazyk C rozlišuje mezi deklarací a definicí funkce:

- Deklarace – říká jak funkci budeme volat, je tvořena hlavičkou a středníkem.
- Definice – říká co funkce bude dělat, je tvořena hlavičkou a tělíčkem.

```
1 //deklarace funkce
2 int soucet(int x, int y); //NavratovyTyp Identifikator (SeznamParametru);
3
4 //definice funkce
5 int soucet(int x, int y)  //NavratovyTyp Identifikator (SeznamParametru)
6 {                         //zacatek tela funkce
7     int vysledek = x + y; //vypocet
8     return vysledek;      //predani navratove hodnoty (konec funkce)
9 }                         //konec tela funkce
```

Deklarace funkcí

Funkci musíme deklarovat nebo definovat dříve než ji budeme volat:

- Funkce deklarujeme proto abychom nemuseli řešit vzájemné pořadí jejich definic.
- U složitějších programů, sestavovaných z více souborů, deklarace funkcí píšeme do samostatných hlavičkový souborů s příponou .h.

```
1  int main()                    //hlavicka funkce main (zacatek programu)
2  {                             //"warning: implicit declaration of function"
3      int x = soucet(10,20);    //CHYBA -- volana funkce soucet jeste
4                                //  nebyla deklarovana ani definovana
5      printf("%i\n", x);        //vypis
6      return 0;                 //konec funkce main (konec programu)
7  }
8
9  int soucet(int x, int y)      //hlavicka funkce soucet
10 {
11     return x + y;             //konec funkce soucet
12 }
```

Návratový typ

Návratový typ funkce je datový typ hodnoty kterou funkce vrací:

- Pokud funkce žádnou hodnotu nevrací, její návratový typ je – `void` –.
- U funkcí typu – `void` – můžeme vynechat příkaz – `return` –.

```
1  void prazdnaFunkce()  //funkce bez navratove hodnoty (a bez parametru)
2  {
3      //return;         //prikaz return bez navratove hodnoty lze vynechat
4  }
```

Návratovou hodnotu můžeme použít i jako parametr jiné funkce:

```
5  int soucet(int x, int y)     //definice
6  {
7      return x + y;            //konec funkce soucet
8  }
9
10 int main()                   //hlavicka funkce main (zacatek programu)
11 {
12     printf("%i + %i = %i\n", 10, 20, soucet(10,20)); //navratova hodnota
13                              //funkce soucet je parametrem funkce printf
14     return 0;                //konec funkce main (konec programu)
15 }
```

Vyzkoušejte si:

- Ze vstupu si načtěte dvě celá čísla a implementujte funkce které spočítají jejich největší společný dělitel a nejmenší společný násobek.

```
1 int nejvetsiDelitel (int a, int b);
2 int nejmensiNasobek (int a, int b);
```

- Obě funkce musejí ověřit platnost parametrů (obě čísla musejí být kladná), a pro neplané parametry vrátí nulu.

Například:

- (4, 6) => delitel = 2, nasobek = 12
- (5, 7) => delitel = 1, nasobek = 35
- (3, -3) => delitel = 0, nasobek = 0

Šablona

```c
#include <stdio.h>                              //vlozeni knihovny

int nejvetsiDelitel (int a, int b);             //deklarace funkce
int nejmensiNasobek (int a, int b);             //deklarace funkce

int main()                                      //zacatek programu
{
    int x, y;                                   //vytvor promenne
    scanf("%i %i", &x, &y);                     //nacti do nich vstup

    int delitel = nejvetsiDelitel(x, y); //zavolej prvni  funkci
    int nasobek = nejmensiNasobek(x, y); //zavolej druhou funkci

    printf("delitel = %i\n", delitel);      //vypis
    printf("nasobek = %i\n", nasobek);      //vypis

    return 0;                                   //konec programu
}
```

Pravdivostní hodnoty

Funkce mohou vracet i pravdivostní hodnotu datového typu – `bool` – (boolean):

- Používání tohoto datového typu vyžaduje knihovnu stdbool.h.

```
1  #include <stdbool.h>     //knihovna pro pravdivostni hodnoty
2  bool jeKladne(int x)     //definice funkce
3  {
4      if (x > 0)           //pokud je cislo vetsi jak nula
5          return true;     //vracime hodnotu TRUE
6      else                 //jinak
7          return false;    //vracime hodnotu FALSE
8  }
9
10 int main()               //hlavicka funkce main (zacatek programu)
11 {
12     int x;               //cele cislo "x"
13     scanf("%i", &x);     //ze vstupu do "x" nacteme hodnotu
14     if (jeKladne(x))     //pokud volana funkce vrati TRUE
15         printf("cislo %i je kladne\n", x);   //vypis
16     else                 //jinak
17         printf("cislo %i neni kladne\n", x); //vypis
18     return 0;            //konec funkce main (konec programu)
19 }
```

Předávání polí

Pole a jeho délku funkci předáváme jako dva samostatné parametry:

- V parametrech funkce pole označujeme prázdnými hranatými závorkami ([ ]).

```
1  int sumaPole(int delka, int pole[]) //definice funkce
2  {
3      int suma = 0;                   //pocatecni hodnota sumy je 0
4      for (int i = 0; i < delka; i++) //pro vsechny prvky pole
5          suma += pole[i];            //k sume pricteme prvek
6      return suma;                    //konec funkce, vracime sumu
7  }
8
9  int main()                          //zacatek programu
10 {
11     int arr[3] = {10, 20, 30};      //pole tri celych cisel "arr"
12     int suma = sumaPole(3, arr);    //pole predavame BEZ hranatych zavorek
13     printf("suma = %i\n", suma);    //vypis
14     return 0;                       //konec programu
15 }
```

Pole nelze použít jako návratový typ:

```
16 int[] funkce(int delka, int pole[]);//CHYBA -- nelze vracet pole
```

Na procvičení

Vyzkoušejte si:

- Vytvořte si pole znaků a ze vstupu do něj načtěte řetězec.
- Implementujte funkci která ověří jestli řetězec obsahuje alespoň jedno malé písmeno, a funkci která ověří jestli všechny znaky jsou malá písmena.

```
1 bool alesponJednoMale (char str[]);
2 bool vsechnaMala (char str[]);
```

- Vytvořte si inicializované pole několika desetinných čísel.
- Implementujte funkce které spočítají jejich aritmetický a geometrický průměr.

```
3 float aritmetickyPrumer (int n, float arr[]);
4 float geometrickyPrumer (int n, float arr[]);
```

Například:

- ("Hello") => Alespon jeden znak je male pismeno
  => Alespon jeden znak neni male pismeno
- (1.0, 2.0, 3.0, 4.0, 5.0) => aritmeticky = 3.000000
  => geometricky = 2.605171

Šablona

```
1  int main()                                  //zacatek programu
2  {
3      // ???                                  //vytvorime pole znaku
4      // ???                                  //nacteme do nej retezec
5
6      if (alesponJednoMale(str))              //pokud funkce vratila true
7          printf("alespon jedno male\n");     //vypis
8      else                                    //jinak (funkce vratila false)
9          printf("zadna mala\n");             //vypis
10
11     if (vsechnaMala(str))                   //pokud funkce vratila true
12         printf("vsechna mala\n");           //vypis
13     else                                    //jinak (funkce vratila false)
14         printf("alespon jedno nemale\n");   //vypis
15
16     float arr[5] = {1.0,2.0,3.0,4.0,5.0};   //vytvarime pole cisel
17     float a = aritmetickyPrumer(5,arr);     //volame funkci
18     float g = geometrickyPrumer(5,arr);     //volame funkci
19     printf("aritmeticky = %f\n", a);        //vypis
20     printf("geometricky = %f\n", g);        //vypis
21     return 0;                               //konec programu
22 }
```

Vnořené cykly

Cykly můžeme vzájemně vnořovat (stejně jako podmínky):

- V každém cyklu obvykle iterujeme přes jinou proměnnou.

```
1  for (int i = 0; i < 4; i++)            //pro "i" od 0 do 3
2  {
3      for (int j = 0; j < 4; j++)        //pro "j" od 0 do 3
4      {
5          printf("i=%i, j=%i\n", i, j); //vypis hodnotu "i" a "j"
6      }
7  }
```

Pokud tělo cyklu obsahuje jen jeden příkaz tak jeho závorky můžeme vynechat:

- Vnořený cyklus nebo podmínka jsou považovány za jeden strukturovaný příkaz.

```
8  for (int i = 0; i < 4; i++)            //pro promennou "i" od 0 do 3
9      for (int j = 0; j < 4; j++)        //pro promennou "j" od 0 do 3
10         printf("i=%i, j=%i\n", i, j); //vypis hodnotu "i" a "j"
```

Cyklus můžeme předčasně ukončit příkazem – `break` – (zlom):

```
1  for (int i = 0; i < 4; i++) //pro "i" od 0 do 3
2  {
3      if (i == 2)             //pokud je "i" rovno 2
4          break;              //ukonci cyklus
5      printf("%i ", i);       //vypis hodnotu "i"
6  }
7  printf("\n");               //vypis konec radku
```

Zbytek těla cyklu můžeme přeskočit příkazem – `continue` – (pokračuj):

```
8  for (int i = 0; i < 4; i++) //pro "i" od 0 do 3
9  {
10     if (i == 2)             //pokud je "i" rovno 2
11         continue;           //preskoc zbytek tela cyklu
12     printf("%i ", i);       //vypis hodnotu "i"
13 }
14 printf("\n");               //vypis konec radku
```

Přerušení vnořeného cyklu

Pokud příkaz – `break` – nebo – `continue` – použijeme uvnitř těla vnořeného cyklu tak se vnitřní cyklus přeruší ale vnější cyklus bude pokračovat:

```
1  int main()                          //hlavicka funkce main
2  {
3      for(int i = 0; i < 4; i++)      //pro "i" od 0 do 3
4      {
5          for(int j = 0; j < 4; j++)  //pro "j" od 0 do 3
6          {
7              printf("X");            //vypis znak "X"
8              if (i == j)             //pokud se "i" rovna "j"
9              {
10                 break;              //ukonci vnitrni cyklus
11             }
12         }
13         printf("\n");               //vypis konec radku
14     }
15     return 0;                       //konec funkce main
16 }
```

Na procvičení

Vyzkoušejte si:

- Vytvořte si dvě různě dlouhá inicializovaná pole několika celých čísel.
- Implementuje funkce pro následující množinové operace:

```
1 bool jeVMnozine(int delka, int pole[], int cislo);
2 bool jeMnozina(int delka, int pole[]);
3 void vypisPrunik(int delkaA, int poleA[], int delkaB, int poleB[]);
4 void vypisSjednoceni(int delkaA, int poleA[], int delkaB, int poleB[]);
```

- U obou dvou polí ověřte jestli to jsou množiny, a pokud ano, tak vypište jejich průnik a sjednocení.
- Ve funkcích pro průnik a sjednocení **NEVYTVÁŘEJTE** novou množinu, pouze vypište odpovídající prvky – na pořadí prvků nezáleží.
- Funkce `vypisSjednoceni` by měla volat funkci `jeVMnozine`.

Například:

- (1 2 3 4, 0 2 4 6 0) => Pole A je mnozina
  => Pole B neni mnozina
- (1 2 3 4, 0 2 4 6 8) => Pole A je mnozina
  => Pole B je mnozina
  => Prunik je: 2 4
  => Sjednoceni je: 1 2 3 4 0 6 8

```c
int main()                                  //zacatek programu
{
    int poleA[4] = {1,2,3,4};               //prvni inicializovane pole
    int poleB[5] = {0,2,4,6,8};             //druhe inicializovane pole

    if (jeMnozina(4,poleA))                 //overujeme ze poleA je mnozina
        printf("pole A je mnozina\n");      //pokud ano, vypis
    else                                    //jinak
        printf("pole A neni mnozina\n");    //pokud ne, vypis

    if (jeMnozina(5,poleB))                 //overujeme ze poleB je mnozina
        printf("pole A je mnozina\n");      //pokud ano, vypis
    else                                    //jinak
        printf("pole A neni mnozina\n");    //pokud ne, vypis

    if (jeMnozina(4,poleA) && jeMnozina(5,poleB)) //pokud jsou obe mnoziny
    {
        vypisPrunik(4,poleA,5,poleB);       //vypisujeme jejich prunik
        vypisSjednoceni(4,poleA,5,poleB);//vypisujeme jejich sjednoceni
    }
    return 0;                               //konec programu
}
```