Základy programování v C++ - 5. cvičení

Zuzana Petříčková

22. října 2018

Přehled

Už jsme probírali

- proměnné a datové typy
  - číselná aritmetika, operátory, knihovna **cmath**
  - implicitní a explicitní přetypování
- rozdělení programu na podprogramy (funkce)
  - význam
  - definiční deklarace, informativní deklarace
  - volání
- další konvence při psaní programů:
  - komentáře, odsazení, lokální proměnné
  - rozdělení programu do více souborů
- Příkazy pro řízení běhu programu
  - podmíněné bloky (podmínky)
  - cykly (smyčky)

Domácí úkol – bylo

Minule:

- Vytvořte "povídací" program, který umí řešit v oboru reálných čísel kvadratickou rovnici v základním tvaru $ax^2 + bx + c = 0$
  - Postupně načte koeficienty a, b, c ze standardního vstupu.
  - Spočte diskriminant a oba kořeny (pomocí pomocných funkcí).
  - Vypíše výsledky.
- Zkuste program rozdělit do více souborů.

Rozšíření:

- Program ošetří speciální případy (např. situaci, kdy rovnice nemá řešení v R, kdy má jeden dvojnásobný kořen, případ $a = 0$).
- Program bude načítat zadání a vypisovat výsledky v cyklu (ukončeném uživatelem).
- Program bude kontrolovat vstup od uživatele (že zadal číslo).

Řízení běhu programu

- funkce
- podmíněné bloky (podmínky)
- cykly (smyčky)

Řízení běhu programu - Podmíněné bloky

Jeden příkaz:

```
if (testovaci_podminka)
prikaz1;
```

Více příkazů:

```
if (testovaci_podminka)
{
prikaz1;
        prikaz2;
        ...
}
```

testovaci_podminka = výraz, který lze převést na logickou hodnotu

Řízení běhu programu - Podmíněné bloky

IF - ELSE:

```
if (testovaci_podminka)
prikaz1;
else
        prikaz2;
```

Více příkazů:

```
if (testovaci_podminka)
prikaz1;
else
{
        prikaz2;
        prikaz3;
        ...
}
```

Řízení běhu programu - Podmíněné bloky

IF - ELSE IF - ELSE:

```
 if (testovaci_podminka1)
prikaz1;
 else if (testovaci_podminka2)
prikaz2;
 else
      prikaz3;
```

- opět lze použít i bloky příkazů

Řízení běhu programu - Podmíněné bloky

Testovací podmínky

- boolovský výraz
- libobolný výraz, který lze převézt na typ **bool** (čísla, znaky, ukazatele)

Relační operátory (operátory pro porovnání)

- == ... rovnost (NE pro racionální čísla)
- != ... nerovnost (NE pro racionální čísla)
- < ... je menší
- <= ... je menší nebo rovno
- > ... je větší
- >= ... je větší nebo rovno

Řízení běhu programu - - Podmíněné bloky

Logické operátory (operátory pro porovnání)

- && ... logické AND
- || ... logické OR
- ! ... logická negace

Příklad

```
if (a == 0 || b == 0)
    cout << "Jedno z cisel je 0.";
else
    cout << "Cisla jsou nenulova.";
```

Řízení běhu programu - Cykly

Cyklus WHILE:

- část kódu, která se provádí vícekrát
- o dalším opakování rozhoduje splnění podmínky
- počet opakování může a nemusí být znám předem

Jeden příkaz:

```
while (testovaci_podminka)
prikaz1;
```

Řízení běhu programu - Cyklus WHILE

Jeden příkaz:

```
while (testovaci_podminka)
prikaz1;
```

Více příkazů:

```
while (testovaci_podminka)
{
prikaz1;
        prikaz2;
        ...
}
```

testovaci_podminka = výraz, který lze převést na logickou hodnotu

Řízení běhu programu - Cyklus WHILE

Příklad:

```
...
int i = 0;
while (i <= 10)
{
    cout << i << " " << i*i << endl;
    i++;
}
...
```

Řízení běhu programu - Cyklus DO-WHILE

```
do
{
prikaz1;
        prikaz2;
        ...
}
while (testovaci_podminka)
```

Příklad: funkce, která načte číslo v daném rozsahu hodnot
int nacti_cislo (int minimum, int maximum)

Řízení běhu programu - Cyklus DO-WHILE

Příklad: funkce, která načte číslo v daném rozsahu

```
int nacti_cislo (int minimum, int maximum)
{
    int n;
    cout << "Zadej cislo:" << endl;
    do
    {
        if (!(cin >> n))
            ...;
        if (n < minimum || n > maximum)
            cout << "Neplatne zadani. Zkus to znovu:" << endl;
    }
    while (n < minimum || n > maximum);
    return n;
}
```

Řízení běhu programu - cyklus WHILE

Příklady na procvičení WHILE:

1. Napište a zavolejte funkci **long long faktorial(int n)**, která pro zadané celé číslo n spočte a vrátí $n!$.
2. Napište a zavolejte funkci **double mocnina(double x, int n)**, která spočte a vrátí x^n.
3. Napište a zavolejte funkci **double euler(int n)**, která spočítá odhad hodnoty eulerovy konstanty $e = lim_{n \to \infty}(1 + \frac{1}{n})^n$ spočtením $(1 + \frac{1}{n})^n$ pro zadané n.
4. Napište a zavolejte funkci **int najdi(double x)**, která najde nejmenší n takové, že $x <= 2^n$.

Řízení běhu programu

Příklad 1: řešení

```
long long faktorial (int n)
{
    long long f = 1;
    while (n > 0)
    {
        f *= n;
        n--;
    }
    return f;
}

int main()
{
    int n = nacti_cislo(0,20);
    cout << "Faktorial je " << faktorial(n) << endl;
    return 0;
}
```

Řízení běhu programu

Příklad 2: řešení

```
double mocnina (double x, int n)
{
    double y = 1;
    while (n > 0)
    {
        y *= x;
        n--;
    }
    return y;
}
int main()
{
    int n = nacti_cislo(0,100);
    double x;
    cin >> x;
    cout << "Mocnina je " << mocnina(x,n) << endl;
    return 0;
}
```

Řízení běhu programu

Příklad 3, 4: řešení

```
double euler(int n)
{
    double x = (1 + 1.0 / n);
    return mocnina(x, n);
}


int najdi(double x)
{
    int n = 0;
    while (x > mocnina(2,n))
      n++;
    return n;
}
```

Problém racionálních čísel

Příklad

- Kód, který v cyklu WHILE přičítá k proměnné f typu double inicializované na 0 hodnotu 0.1. Cyklus ukončí, jakmile f bude rovno 1.

```
...
double f = 0.0;
while(f != 1.0)
{
    f += 0.1;
}
...
```

→ zacyklí se

- Racionální čísla jsou na počítači reprezentována nepřesně → místo operátorů -="a "!="testovat, zda jsou čísla dostatečně blízko.

Problém racionálních čísel

Příklad

- Kód, který v cyklu WHILE přičítá k proměnné f typu double inicializované na 0 hodnotu 0.1. Cyklus ukončí, jakmile f bude rovno 1.

```
...
double eps = 1e-5;
...
double f = 0.0;
while(abs(f-1.0) > eps)
{
  f += 0.1;
}
...
```

Řízení běhu programu - Cyklus FOR

Jeden příkaz:

```
for ( pocatecni_prikaz; podminka; krok )
    prikaz1;
```

Více příkazů:

```
 for ( pocatecni_prikaz; podminka; krok )
{
    prikaz1;
    prikaz2;
    ...
}
```

Řízení běhu programu - Cyklus FOR

FOR-cyklus:

```
for ( pocatecni_prikaz; podminka; krok )
{
    prikaz1;
    prikaz2;
}
```

Ekvivalentní kód:

```
pocatecni_prikaz;
while ( podminka )
{
    prikaz1;
    prikaz2;
    krok;
}
```

Řízení běhu programu - Cyklus FOR

Příklad:

- typické použití: provedení pevného počtu opakování:

```
for (i = 0; i <= 10; i++)
    cout << i << " " << i*i << endl;
for (i = 10; i > 0; 10; i--)
    cout << i << " " << i*i << endl;
```

faktoriál:

```
for (int i = 0; i < 26; i++)
    cout << "Faktorial " << i << " je " << faktorial(i) << en
```

Řízení běhu programu - cyklus FOR

Příklady na procvičení FOR:

1. Napište for-cyklus, který vypíše všechna sudá čísla od 10 do 25.
2. Napište a zavolejte funkci **long long faktorial1(int n)**, která pro zadané celé číslo n spočte a vrátí $n!$.
3. Napište a zavolejte funkci **double mocnina1(double x, int n)**, která spočte a vrátí x^n.
4. Napište a zavolejte funkci **void delitele(int n)**, která vypíše všechny dělitele čísla n.

Příklad 2: řešení

```
pocatecni_prikaz;
while(podminka)
{
   prikaz1;
   krok;
}
```

```
for(pocatecni_prikaz;podminka;krok)
    prikaz1;
```

```
long long faktorial (int n)
{
    long long f = 1;
    int i = 1;
    while (i <= n)
    {
        f *= i;
        i++;
    }
    return f;
}
```

```
long long faktorial1 (int n)
{
    long long f = 1;
    for (int i = 1; i <= n; i++)
    {
        f *= i;
    }
    return f;
}
```

Řízení běhu programu

Příklad 1: řešení

```
...
for (int i = 10; i <= 25; i+=2)
        cout << i << endl;
...
```

Řízení běhu programu

Příklad 3: řešení

```
double mocnina1 (double x, int n)
{
    double y = 1;
    for (int i = 1; i <= n; i++ )
    {
        y *= x;
    }
    return y;
}
int main()
{
    int n = nacti_cislo(0,100);
    double x;
    cin >> x;
    cout << "Mocnina je " << mocnina1(x,n) << endl;
    return 0;
}
```

Řízení běhu programu - skoky

break:

- Ukončí cyklus, program následuje prvním příkazem za cyklem

continue:

- Ukončí aktuální iteraci cyklu a započne novou

Příklad

```
int a;
for (int i=0; i<10; i++)
{
    cin >> a;
    if (a < 0)
      continue;       // break;
    cout << a;
}
```

Řízení běhu programu - skoky

Příklad: Funkce načítá čísla na vstupu, dokud uživatel nezadá celé číslo v daném rozsahu. Funkce používá nekonečný cyklus.

```
int nacti_cislo1 (int minimum, int maximum)
{
    int n;
    while (true)
    {
        cout << "Zadej cislo:" << endl;
        cin >> n ;                          // Potencialni problem
        if (n >= minimum && n <= maximum)
            break;
        cout << "Neplatne zadani. Zkus to znovu:" << endl;
    }
    return n;
}
```

Řízení běhu programu - skoky

Příklad - pokračování: Funkce si poradí se situací, kdy se načtení čísla nepodařilo - snaží se situaci "opravit".

```
int nacti_cislo1 (int minimum, int maximum)
{
    int n;
    while (true)
    {
        cout << "Zadej cislo:" << endl;
        if (!(cin >> cislo))
        {
          cin.clear();
          cin.ignore(256,'\n');
          cout << "Zadany vstup neni cele cislo. Zkus to znovu.";
          continue;
        }
        if (n >= minimum && n <= maximum)
            break;
        cout << "Neplatne zadani. Zkus to znovu:" << endl;
    }
    return n;
}
```

Řízení běhu programu - skoky

Příklad - pokračování: Funkce si poradí se situací, kdy se načtení čísla nepodařilo - ukončí program.

```
#include <string>
...
void chyba(string text)
{
    cerr << text << endl;
    exit(EXIT_FAILURE); // ukonci program
}
int nacti_cislo2 (int minimum, int maximum)
{
   int n;
   while (true)
   {
      cout << "Zadej cislo:" << endl;
      if (!(cin >> cislo))
      {
          chyba("nacti_cislo2: Zadany vstup neni cele cislo.");
          cislo = minimum;
          break;
      }
      ...
   }
```

Shrnutí:

Jak reagovat na chybu (např. ve vstupu od uživatele)?

1. Vypsat informaci o chybě.
2. Reagovat na chybu:
   - snažit se chybu opravit.
     "Zadej znova..."
     "cislo = minimum;"
   - Ukončit program
   - Chybu ignorovat → zákeřné

Rekurze

Rekurzivní funkce

- funkce, která "volá sama sebe"ještě před svým dokončením

Příklad: faktoriál

- $f(n) = n! = n(n-1)(n-2)...2 \cdot 1, \quad n \in N$

Faktoriál - rekurzivní definice:

- $f(0) = 1$
- $f(n) = n! = n \cdot (n-1)! = n \cdot f(n-1); \quad n > 1$

Rekurzivní funkce

Příklady na procvičení REKURZE:

1. Napište a zavolejte funkci **long long faktorial2(int n)**, která pro zadané celé číslo n spočte a vrátí $n!$.
2. Napište a zavolejte funkci **double mocnina2(double x, int n)**, která spočte a vrátí x^n.
3. Napište a zavolejte funkci **int fibonacci(int n)**, která vypíše n-tý člen Fibonacciho posloupnosti.
4. **(pro pokročilé)** Napište a zavolejte funkci **int fibonacci1(int n)**, která vypíše n-tý člen Fibonacciho posloupnosti bez využití rekurze.

Řízení běhu programu

Další příklady

1. Napište a zavolejte funkci **int secti_cisla()**, která sčítá čísla ze vstupu, dokud uživatel nezadá 0. Vrátí výsledek. Pokud uživatel nezadá celé číslo (udělá překlep ap.), požádá ho program o opravu zadání.

Rekurze

Příklad 1, 2: řešení

```
long long faktorial2(int n)
{
    if (n <= 0)
        return 1;
    return n*faktorial2(n-1);
}

double mocnina2(double x, int n)
{
  if (n <= 0)
      return 1;
  return x*mocnina2(x,n-1);
}
```

Rekurze

Příklad 3: řešení Fibonacci - rekurzivní definice

- $f(0) = 0$
- $f(1) = 1$
- $f(n) = f(n-1) + f(n-2); \quad n > 1$

```
int fibonacci(int n)
{
    if (n <= 0)
      return 0;
    if (n == 1)
      return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

Domácí úkol 1

- Napište a zavolejte funkci **int ciferny_soucet(int n)**, která vrátí ciferný součet čísla n ($157 \rightarrow 13 = 1 + 5 + 7$).

Domácí úkol 2

Kvadratická rovnice - pokračování

- Vytvořte "povídací" program, který umí řešit v oboru reálných čísel kvadratickou rovnici v základním tvaru $ax^2 + bx + c = 0$
  - Postupně načte koeficienty a, b, c ze standardního vstupu.
  - Spočte diskriminant a oba kořeny (pomocí pomocných funkcí).
  - Vypíše výsledky.
- Zkuste program rozdělit do více souborů.

Rozšíření:

- Program vrátí řešení rovnice i pro speciální případy (např. v situaci, kdy rovnice má řešení v oboru komplexníích čísel, kdy má jeden dvojnásobný kořen, případ, kdy degraduje na lineární rovnici ap.).
- Program bude načítat zadání a vypisovat výsledky v cyklu (ukončeném uživatelem).
- Program bude kontrolovat vstup od uživatele (že zadal číslo).