Základy programování v C++ 11. cvičení
Funkce a jejich parametry
Reference
Struktury (další příklady)

Zuzana Petříčková

29. října 2019

Přehled

Funkce a jejich parametry

Předávání parametrů funkcím hodnotou

```
void funkce(int a, int b)
{
    ...
}
```

- funkce pracují s **lokální kopií** svých parametrů, tyto kopie jsou po ukončení funkce zrušeny
  → **funkce** nemůže měnit skutečné hodnoty parametrů **a** a **b**

Výjimka: parametr typu pole

- pole se předává **odkazem** (uvnitř funkce můžeme měnit hodnoty v poli)

```
void funkce(int *a, int n);
```

Funkce a jejich parametry

Předávání parametrů hodnotou a návratová hodnota

```
int funkce(int a, int b)
{
    ...
}

int jinaFunkce()
{
   int a = 5, b = 3;
   a = funkce(a,b);
}
```

- **funkce** sice nemůže měnit skutečné hodnoty parametrů **a** a **b**
- ale může mít (jednu) návratovou hodnotu

Funkce a předávání parametrů hodnotou - příklad:

```
void zmen(int x)
{
    x++;
    cout << "Hodnota promenne uvnitr je " << x << endl;
}
int zmen1(int x)
{
    x++;
    cout << "Hodnota promenne uvnitr je " << x << endl;
    return x;
}
int main()
{
    int a = 9;
    zmen(a);
    cout << "Hodnota promenne venku je " << a << endl;
    a = zmen1(a);
    cout << "Hodnota promenne venku je " << a << endl;
```

Funkce a předávání parametrů hodnotou

- Co když ale chci změnit hodnoty více proměnných?

```
void neprohod(int a, int b)
{
    int c = a;
    a = b;
    b = c;
}

int main()
{
    int x = 1, y = 2;
    neprohod(x,y); // :(
    cout << x << ' ' << y << endl;
    return 0;
}
```

Funkce a předávání parametrů hodnotou

- Co když ale chci změnit hodnoty více proměnných?

Možná řešení

1. pomocí více funkcí :(
2. **pomocí pole**

```
void prohod(int *a)
{
    int c = a[0];
    a[0] = a[1];
    a[1] = c;
}
int main()
{
    int a[2] = {1,2};
    prohod(a); // :)
    cout << a[0] << ' ' << a[1] << endl;
    return 0;
}
```

Funkce a předávání parametrů hodnotou

- Co když ale chci změnit hodnoty více proměnných?

Možná řešení

1. pomocí pole
2. **pomocí struktury**

```
struct Dvojice
{
  int a, b;
};
Dvojice prohod(Dvojice x)
{
    int c = x.a;
    x.b = x.a;
    x.a = c;
    return x;
}
```

Funkce a předávání parametrů hodnotou

- Co když ale chci změnit hodnoty více proměnných?

Možná řešení

1. pomocí pole
2. **pomocí struktury**

```
Dvojice prohod(Dvojice x)
{
    int c = x.a;
    x.b = x.a;
    x.a = c;
    return x;
}
int main()
{
    Dvojice a = {1,2};
    a = prohod(a); // :)
    cout << a.a << ' ' << a.b << endl;
    return 0;
}
```

Funkce a předávání parametrů hodnotou

- Co když ale chci změnit hodnoty více proměnných?

Možná řešení

1. pomocí pole
2. pomocí struktury
3. **pomocí tzv. referencí**
4. pomocí tzv. ukazatelů (ukážeme později)

```
void prohod(int &a, int &b)
{
    int c = a;
    a = b;
    b = c;
}
int main()
{
    int x = 1, y = 2;
    prohod(x,y); // :)
    cout << x << ' ' << y << endl;
    return 0;
```

Reference

Deklarace (definice)

- znak **&**

```
int x = 7, y = 5;
int &rx = x; // referenci inicializuji na promennou x
rx++;   // totez jako x++;
rx = y; // totez jako x = y; (rx stale odkazuje na x)
x += y;
cout << x << ' ' <<  rx << endl;
```

Omezení

- referenci je třeba inicializovat ihned při deklaraci na nějakou l-hodnotu (např. proměnnou)
- odkazovanou proměnnou nelze později změnit
  → na referenci se mohu dívat jako na nové jméno pro existující proměnnou

Reference

Reference může odkazovat i na prvek pole:

```
int a[5] = { 1,2,3,4,5 };
int &atri = a[3];
atri = 8; // totez jako a[3]=8
cout << a[3] << ' ' << atri << endl;
```

Reference

Význam

- předávání parametrů funkcí odkazem
- jiné jméno pro existující proměnnou / prvek pole
- návratový typ funkce (tzv. referenční funkce, funkce vracející l-hodnotu, ukážeme později)

Omezení

- nelze deklarovat referenci na referenci, pole referencí *a ukazatel na referenci*
- naopak lze deklarovat referenci na pole *nebo referenci na ukazatel*

Reference

Další využití reference - struktura jako parametr funkce:

```
void vypis(Zamestnanec z)
{
    ...
}
Zamestnanec zmenPlat(Zamestnanec z, int novyPlat)
{
    ...
}
```

- zbytečně se vytváří kopie struktury na zásobníku → lépe:

```
void vypis(const Zamestnanec &z)
{
    ...
}
void zmenPlat(Zamestnanec &z, int novyPlat)
{
    ...
}
```

Příklad : Zlomek

- Definujte strukturu **Zlomek** s celočíselnými složkami **citatel** a **jmenovatel** (jmenovatel bude vždy kladné číslo, případné znamenko mínus bude u čitatele).
- Napište funkce pro základní operace se zlomky (sčítání, odčítání, násobení, dělení). Funkce převedou výsledek do základního tvaru.
  Příklad: součet 2/3 a -1/6 je 1/2
- **Pomocné funkce**
  - **int nsd(int x, int y)**, která spočítá největšího společného dělitele čísel x a y.
  - **void zakladniTvar(Zlomek &x)** nebo **Zlomek zakladniTvar(const Zlomek &x)**, která převede zlomek do základního tvaru (vydělí čitatele i jmenovatele jejich největším společným dělitelem a opraví znaménko minus, aby bylo pouze u čitatele).
  - **void vypisZlomek(const Zlomek &x)**, která vypíše zlomek na konzoli.

Příklad : Zlomek

Seznam funkcí k implementaci

```
Zlomek secti(const Zlomek &x,const Zlomek &y);
Zlomek odecti(const Zlomek &x,const Zlomek &y);
Zlomek vynasob(const Zlomek &x,const Zlomek &y);
Zlomek vydel(const Zlomek &x,const Zlomek &y);
int nsd(int x, int y);
void zakladniTvar(Zlomek &z); // Zlomek zakladniTvar(const Z
void vypisZlomek(const Zlomek &x);
```

Pro struktury mohu definovat (přetížit) i operátory

klíčové slovo **operator**:

```
...
Zlomek operator + (const Zlomek& x,const Zlomek& y)
{
  return secti(x, y);
}

Zlomek operator - (const Zlomek& x,const Zlomek& y)
{
  return odecti(x, y);
}

int main()
{
  Zlomek x= {2, 3}, y ={-2, 12}, z;
  z = x + y;
  ...
}
```

Příklad : Zlomek

Euklidův algoritmus pro výpočet největšího společného dělitele (pro $x, y > 0$)

$$nsd(x, y) = \begin{cases} x, & x == y \\ nsd(x - y, y), & x > y \\ nsd(x, y - x), & x < y \end{cases}$$

Pro zvídavé: Pro struktury mohu definovat (přetížit) i operátor <<

ostream je datový typ pro výstupní datový proud (např. cout)

```
...
ostream& operator << (ostream &out, const Zlomek &z)
{
    out << z.citatel << " / " << z.jmenovatel << endl;
    return out;
}

int main()
{
   Zlomek x= {2, 3};
   cout << x;
   ...
}
```