5. Reference, konstantní metody, přetížené funkce a operátory

Předávání parametrů hodnotou a referencí

Způsob kterým se obvykle předávají parametry funkcím (nebo metodám) se nazývá předávání hodnotou. To se využívá hlavně při předávání parametrů základních typů (`int`, `char`, `double`, ...).

Při předávání parametrů hodnotou se parametry funkce chovají jako lokální proměnné, pro které se alokuje potřebná paměť a zkopírují se do nich předávané hodnoty.

V jazyce C++ je také možné předávat parametry referencí. V takovém případě nedochází k alokaci paměti pro nové proměnné, ale pouze jména parametrů jsou použita pro přístup k proměnným které byly do funkce předány.

Parametry, které mají být předávány referencí musí mít mezi typem a jménem parametru uvedený znak **&**.

```
// Funkce swapNumbers() zameni obsah dvou
// ciselnych promennych

// Parametry jsou predavany referenci
void swapNumbers(int &n1, int &n2)
{
  int n3 = 0;  // Pomocna promenna
  n3 = n2;
  n2 = n1;
  n1 = n3;
}

int main()
{
  int a = 3, b = 7;

  cout << a << ", " << b;  // Vypise: 3, 7
  swapNumbers(a, b);
  cout << a << ", " << b;  // Vypise: 7, 3

  // Pokud by funkce swapNumbers()
  // neprijimala parametry referenci, ale
  // hodnotou, k zadne zamene hodnot
  // v promennych a, b by nedoslo a
  // vypsala by se cisla 3, 7
  return 0;
}
```

Předávání objektových parametrů konstantní referencí

Předávání proměnných referencí používáme především v následujících dvou situacích:

1) Pokud potřebujeme aby uvnitř volané funkce nebo metody mohla být modifikována hodnota předávané proměnné (např. jako v příkladu funkce `swapNumbers(int &a, int &b)`).

2) Pokud chceme pouze zajistit vyšší efektivitu programu při předávání objektových proměnných, protože při použití referencí nebude docházet k alokaci paměti pro parametr a kopírování hodnoty do něj

Případ 2) se používá v situacích, kdy nebude hodnota předávané proměné nebyla modifikována (narozdíl od případu 1)). Abychom zaručili, že skutečně nedojde ke změně hodnoty předávané proměnné, deklarujeme parametr jako konstantní použitím klíčové slova **`const`**.

Hodnoty parametrů předaných konstantní referencí nelze měnit (překladač by ohlásil chybu).

```
// Nekde na zacatku je definovana trida
Circle

class Circle
{
  public:
    void testNonConstant(Circle &circ);
    void testConstant(const Circle &circ);
    int x, y;
}

void Circle::testNonConstant(Circle &circ)
{
  circ.x = 12;  // Tohle bude fungovat,
  // protoze circ je nekonstantni parametr
}

void Circle::testConstant(
                  const Circle &circ)
{
  circ.x = 12; // Tohle nebude fungovat,
  // prekladac ohlasi chybu, protoze
  // circ je konstantni parametr
}
```

Konstantní metody

Všechny metody, které nemění hodnotu členských dat (tj. proměnných třídy) deklarujeme jako konstantní uvedením klíčového slova `const`.

Pro konstantní parametry předávané metodám (nebo funkcím) smíme volat pouze tyto konstantní metody.

```
class Circle
{
  public:
    void nonConstantMethod();
    void constantMethod() const;
    void test(const Circle &circ);
    int x, y; // Verejne cleny tridy
}

void Circle::nonConstantMethod()
{
  x = 12;  // Tato metoda muze menit sva
 // clenska data protoze neni konstantni
}

void Circle::constantMethod() const
{
  // Tato konstantni metoda nemuze menit
  // sva clenska data
  x = 12;  // Prekladac nahlasil chybu !!!
}
```

POKRACOVANI NA DALSI STRANE ...

```
   ... POKRACOVANI Z PREDCHOZI STRANY STRANY

void Circle::test(const Circle &circ)
{
  // Pro konstantni parametr muzeme volat
  // konstantni metodu:
  circ.constantMethod();

  // Nemuzeme vsak volat nekonstantni
  // metodu (pro konstantni parametr)
  circ.nonConstantMethod();
  // Prekladac by zde ohlasil chybu !!!
}
```

Přetížení funkcí a metod

V jazyce C++ lze definovat více funkcí/metod se stejným jménem ale různým počtem nebo typem parametrů. Takovéto funkce/metody se nazývají přetížené.

Je-li funkce volána, překladač analyzuje parametry které funkci předáváme a podle toho vybere odpovídající funkci/metodu.

```
class Cicle
{
 public:
   void setColor(int c);
   void setColor(const string &colorName);
   void setValues(int ax, int ay);
   void setValues(int ax, int ay, int c);
};

void Circle::setColor(int c)
{
  color = c;
}

void Circle::setColor(
       const string &colorName)
{
  if (colorName == "red") color = 19;
  else if ...
}

void Cicle::setValues(int x, int y)
{
  x = ax;
  y = ay;
}

void Cicle::setValues(int x, int y, int c)
{
  x = ax;
  y = ay;
  color = c;
}

    POKRACOVANI VE VEDLEJSIM SLOUPCI ...
```

```
   ... POKRACOVANI Z PREDCHOZIHO SLOUPCE

int main()
{
  Circle circ;
  // Bude se volat prvni metoda setColor()
  circ.setColor(3);
  // Bude se volat druha metoda setColor()
  circ.setColor("red");

  //Bude se volat prvni metoda setValues()
  circ.setValues(150, 230);
  //Bude se volat druha metoda setValues()
  circ.setValues(150, 230, 3);
  return 0;
}
```

Přetížení konstruktorů

Podobně jako funkce a metody lze přetížit i konstruktory..

```
class Circle
{
public:
  Circle();
  Circle(int ax, int ay);
  Circle(int ax, int ay, int c);
};

Circle::Circle()
{
  x = 0;
  y = 0;
  color = 1;
}

Circle::Circle(int ax, int ay)
{
  x = ax;
  y = ay;
  color = 1;
}

Circle::Circle(int ax, int ay, int c)
{
  x = ax;
  y = ay;
  color = c;
}

int main()
{
  // Bude se volat prvni konstruktor
  Circle circ1;
  // Bude se volat druhy konstruktor
  Circle circ2(150, 230);
  // Bude se volat treti konstruktor
  Circle circ3(150, 230, 3);
  return 0;
}
```

Přetížené operátory

Jazyk C++ obsahuje interní definice operátorů pro základní datové typy (`int`, `double`, `char` ...).

V C++ je kromě toho možné definovat operátory pro uživatelské typy, tj. pro objektové proměnné.

Operátory se definují podobně jako funkce (a metody) pouze místo názvu funkce použijeme klíčové slovo **`operator`** a za ním uvedeme příslušný znak operátoru.

Přetížit lze prakticky všechny dostupné operátory, nejčastěji se přetěžují: =, ==, !=, <, >, <=, >=, [], <<, >>, +, -, *, /.

V praxi používáme přetížené operátory pouze vzácně, jejich nadměrné používání může program znepřehlednit (zejména tam, kde nelze význam operátoru snadno odhadnout z kontextu).

```
class Vector2D
{
  public:
    Vector2D();
    double getX() const;
    double getY() const;
    void set(double ax, double ay);
  private:
    double x, y;
};

// Operator vrati skalarni soucin vektoru
double operator*(const Vector2D &v1,
                 const Vector2D &v2)
{
  return (v1.getX() * v2.getX()
        + v1.getY() * v2.getY());
}

int main()
{
  Vector2D v1, v2;
  double result = 0;

  // Bude se volat operator*
  result = v1 * v2;

  return 0;
}
```

Přetížené operátory jako metody

Operátory lze implementovat také jeko metody.

Volá se vždy metoda objektu na levé straně od operátoru a jako argument se mu předá objekt na pravé straně.

```
class Vector2D
{
public:
  // Tady bude deklarace konstruktoru,
  // dalsich metod a promennych atd.
double operator*(const Vector2D &v) const;
};

    POKRACOVANI VE VEDLEJSIM SLOUPCI ...
```

```
    ... POKRACOVANI Z PREDCHOZIHO SLOUPCE

double Vector2D::operator*(
        const Vector2D &v) const
{
  return (x * v.getX() + y * v.getY());
}

int main()
{
  Vector2D v1, v2;
  double result = 0;

// Bude se volat operator* objektu v1
// a jako parametr se mu preda objekt v2
  result = v1 * v2;
  return 0;
}
```

Přetížené operátory proudů

```
// Operator pro zapis promenne typu
// Vector2D do proudu. Zapisovanou
// promennou predavame konstantni
// referenci, proud os vsak predavame
// nekonstantni referenci, protoze pri
// zapisu se meni stav vnitrnich
// promennych proudu
ostream& operator<< (ostream &os,
                     const Vector2D &v)
{
  os << v.getX() << " " << v.getY()
     << endl;
  return os;
}

// Operator pro cteni promenne typu
// Vector2D z proudu. Parametr v musi
// byt nekonstantni reference, protoze
// se budou menit jeho hodnoty (nactene
// hodnoty souradnic)
istream& operator>> (istream &os,
                     Vector2D &v)
{
  double ax = 0.0, ay = 0.0;
  os >> ax >> ay;
  v.set(ax, ay);
  return os;
}

int main()
{
  Vector2D v1, v2, v3;

  // Vola se vyse definovany operator <<
  cout << v1 << v2 << v3;

  // Vola se vyse definovany operator >>
  cin >> v1 >> v2 >> v3;

  return 0;
}
```

Dodržujte následující pravidla

- Parametry objektových typů předávejte jako konstantní referenci, pokud neexistují důvody pro jiný přístup. Toto platí i pro řetězcové proměnné typu `string`.
- Parametry základních typů (`int`, `double` ...) předávejte obvyklým způsobem (tj. hodnotou a nekonstantní), pokud neexistují důvody pro jiný přístup.
- Všechny metody ve třídách, které nemění hodnoty datových členů třídy, definujte jako konstantní
- Operátory implementuje přednostně jako metody třídy. Pouze tam kde to není možné je implementujte jako funkce, např. pokud je operandem nalevo od operátoru neobjektová proměnná (`int`, `double`, ...) nebo pokud je jím objekt třídy, která je zabudovaná v knihovně a nelze do ní tedy přidávat metody/operátory (např. třídy proudů).

Úloha 1 — 1 bod

Upravte program z úlohy 2 z předchozího cvičení následujícím způsobem:

- Ve třídě `Graphic` implementujte metodu `getColorNumberFromString()`, která bude přijímat název barvy jako parametr (black, blue, green, red, yellow) a vracet číslo barvy.
- Ve třídách `Circle` a `FilledCircle` přidejte další variantu metody `setValues()`, která bude místo čísla barvy (a barvy výplně) přijímat text z názvem barvy (původní variantu metody však také ponechte).
- V každé ze tříd `Graphic`, `Circle`, `FilledCircle` a `Rectangle` implementujte dva konstruktory, jeden bez parametrů a jeden s parametry, které budou nastavovat hodnoty členů tříd.
- Všechny objektové parametry budou do metod předávány jako konstantní reference (týká se i parametrů typu string). Všechny metody, které nemění hodnotu datových členů třídy definujte jako konstantní metody.
- Program upravte tak, aby byl schopen načítat soubor *graphic2.dat* (v adresáři */home/martinp/C3220/data/*), který se od *graphic1.dat* liší tím, že jsou v něm barvy specifikovány formou textu namísto čísel.

Úloha 2 — 3 body

Vytvořte program pro práci s 3D vektory. V programu definujte třídu `Vector3D`, která bude obsahovat souřadnice vektoru `x`, `y`, `z` (typu `double`). Dále bude obsahovat následující metody:

- Konstruktor bez parametrů `Vector3D()` a s parametry `Vector3D(double ax, double ay, double az)`
- Metody `getX()`, `getY()`, `getZ()` pro získání hodnot souřadnic a metodu `set(double ax, double ay, double az)` pro jejich nastavení.
- Metodu `readValues()`, která si od uživatele vyžádá zadání tří souřadnic a načte je a metodu `printValues()`, která vypíše souřadnice vektoru. Obě metody budou přijímat parametr typu `string`, který se uživateli vypíše před tím než se načtou/vypíší souřadnice.
- Operátor * pro skalární součin a operátor + pro vektorový součet.
- Funkci `swapVectors()`, která zamění dva vektory (hodnoty jejich souřadnic). Bude se jednat o samostatnou funkci, ne o metodu.
- Objektové parametry předávejte do metod (vč. operátorů) jako konstantní reference (týká se i parametrů typu `string`). Všechny metody, které nemění hodnotu datových členů třídy definujte jako konstantní metody. Operátory implementujte ve třídách, pokud je to možné.

Program si od uživatele vyžádá souřadnice dvou vektorů a na obrazovku vypíše souřadnice vektorového součtu a dále hodnotu skalárního součinu vektorů. Pak zavolá funkci `swapVectors()` pro ty zadané vektory a vypíše jejich souřadnice.

Poznámka:

V případě objektových proměnných je možné přiřadit jednu proměnnou do druhé (pomocí operátoru =) podobně jako u proměnných základních typů. V takovém případě je automaticky kopírován každý datový člen objektové proměnné.

Úloha 3 — nepovinná, 2 body

Předchozí program modifikujte tak, že v něm implementujete operátory >> a << pro načítání/zápis do proudu. Program načte ze souboru *vectors.dat* (v adresáři */home/martinp/C3220/data/*) souřadnice dvou vektorů a do výstupního souboru zapíše souřadnice vektorového součtu těchto dvou vektorů. Jména vstupního a výstupního souboru budou specifikovány na příkazovém řádku. Pro načítání a zápis do souboru využijte implementované operátory.