Pokročilé programování
v jazyce C pro chemiky
(C3220)

Vstup a výstup v C++, reference, přetěžování funkcí

Reference

- Reference jsou speciálním typem proměnných, které samy neuchovávají žádnou hodnotu, jen odkazují na jiné proměnné
- Reference jsou v podstatě ukazatele (jako v C), které se ale automaticky dereferencují (není třeba používat * a ->)
- Reference se definují přidáním znaku **&** mezi typ a jméno proměnné
- Reference je v okamžiku vytvoření pevně svázána s odkazovanou proměnnou, nelze ji „odpojit“ či „přesměrovat“

```
int a = 1, b = 2;
int &r = a;

a = 3;
cout << r;          // Vypise se cislo 3

// Aktualni hodnota promenne b bude zkopirovana do promenne,
// na kterou odkazuje reference r (tedy do a)
r = b;

b = 4;
cout << r;          // Vypise se cislo 2
```

Předávání parametrů hodnotou

- Způsob, kterým se obvykle v C/C++ předávají parametry funkcím (nebo metodám) se nazývá předávání hodnotou
- Při zavolání funkce se parametry funkce chovají jako lokální proměnné dané funkce, pro které se alokuje potřebná paměť a zkopírují se do nich předávané hodnoty
- Pokud měníme hodnoty parametrů uvnitř funkce, žádným způsobem tím neovlivníme hodnoty proměnných, které byly předávány do funkce (argumentů)

```
  // V pameti se vytvori nova promenna n, do ktere se
  // zkopiruje predavana hodnota (v tomto pripade 1)
void myFunction(int n)
{
  n = 5;
    // Do n se priradi 5, po opusteni funkce vsak promenna n
    // zanikne a jeji hodnota se navzdy ztrati
}

int main()
{
  int a = 1;
  myFunction(a);
  cout << a;          // Vypise se cislo 1
  return 0;
}
```

Předávání parametrů odkazem

- Při předávání parametrů odkazem (referencí) 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 **&**

```
  // V pameti se nevytvari nova promenna, pouze nova
  // reference n je pouzita pro praci s puvodni promenou a
void myFunction(int &n)
{
  n = 5; // Hodnota 5 se v tuto chvili ulozi do puvodni promenne a
}

int main()
{
  int a = 1;
  myFunction(a);
  cout << a;          // Vypise se cislo 5
  return 0;
}
```

Předávání parametrů referencí - příklad

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

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

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

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

  return 0;
}
```

Předávání parametrů referencí

- Reference používáme také v případech, kdy předávaná proměnná zabírá v paměti hodně místa a vytváření její kopie (při předávání hodnotou) by bylo spojeno s velkými paměťovými a výpočetními nároky
- Toto se týká zejména objektových proměnných které předáváme téměř vždy referencí
- Některé objektové proměnné kopírovat vůbec nelze (například tehdy, když reprezentují nějaký externí objekt dle principu RAII)

```
// Nasledujici metoda je ze cviceni 2, uloha 4
void Circle::setAverageCircle(Circle &c1, Circle &c2)
{
  // Pouzitim referenci predejdeme zbytecnemu kopirovani circ1/2 do c1/2
  x = (c1.getCentreX() + c2.getCentreX()) / 2;
  y = (c1.getCentreY() + c2.getCentreY()) / 2;
  radius = (c1.getRadius() + c2.getRadius()) / 2;
}

int main()
{
  Circle circ1(250, 250, 80, 7);
  Circle circ2(250, 250, 80, 3);
  Circle circ3(0, 0, 0, 19);

  circ3.setAverageCircle(circ1, circ2);
  return 0;
}
```

Předávání 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 ke zbytečnému kopírování
- Případ 2) se používá v situacích, kdy nebude hodnota předávané proměnné modifikována (na rozdíl od případu 1)). Abychom zaručili, že skutečně nedojde ke změně hodnoty předávané proměnné, deklarujeme parametr jako konstantní referenci použitím klíčové slova **`const`**

```
void Circle::setAverageCircle(const Circle &circ1, const Circle &circ2)
{
  // Zde bude kod metody
}
```

Předávání parametrů konstantní referencí

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

```
class Circle
{
  public:
    void testNonConstant(Circle &circ);
    void testConstant(const Circle &circ);
    int x, y; // Verejne cleny tridy (jen pro potreby tohoto prikladu)
}

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

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

Proudy pro standardní vstup a výstup

- V jazyce C++ provádíme textový vstup a výstup prostřednictvím tzv. datových proudů
- Datové proudy jsou ve skutečnosti specializované třídy dostupné ve standardní knihovně jazyka C++, které poskytují metody a operátory pro zápis a čtení dat
- Pro práci se standardními vstupy a výstupy je třeba v záhlaví zdrojového souboru deklarovat `#include <iostream>` a `using namespace std;`
- Standardní knihovna obsahuje následující třídy pro standardní vstup a výstup:
  - `ios` – základní třída, na níž jsou založeny vstupní a výstupní proudy
  - `istream` – pro vstupní operace
  - `ostream` – pro výstupní operace
- Ve standardní knihovně jsou definovány následující proměnné:
  - `istream cin` – pro standardní vstup (z klávesnice)
  - `ostream cout` – pro standardní výstup (na obrazovku)
  - `ostream cerr` – pro chybový výstup, bez bufferování (na obrazovku)
  - `ostream clog` - pro chybový výstup (na obrazovku)

Operátory pro vstup a výstup

- Pro výstup do proudu používáme operátor **<<**, pro vstup **>>**
- Tyto operátory lze použít pouze pro čtení/zápis proměnných základních datových typů (`int`, `double`, `char`) a typu `string`

```
#include <iostream>
using namespace std;


int main()
{
  int i = 5;
  double a = 10;
  string str;

  cout << "Hodnota promenne i: " << i << endl;
  cout << "Hodnota promenne a: " << a << endl;

  cin >> str;
  cout << "Retezec je: " << str << endl;

  return 0;
}
```

Vstup a výstup objektových proměnných

- Operátory **<<** a **>>** nelze přímo použít pro čtení/zápis celých objektových proměnných
- Pro objektové proměnné musíme číst/zapisovat jednotlivé členy třídy samostatně

```
#include <iostream>
using namespace std;

// Na zacatku je definovana trida Circle – viz. minule cviceni

void Circle::readValues()
{
  cin >> x >> y >> radius >> color;
}

int main()
{
  Circle circ;
  circ.readValues();
  cout << circ.getCentreX() << circ.getCentreY() << circ.GetColor();

  // Nasledujici by nefungovalo:
  cin >> circ;
  cout << circ;
  return 0;
}
```

Souborový vstup a výstup

- Pro zápis do souboru a čtení ze souboru používáme následující třídy:
  `fstream` – pro čtení i zápis téhož souboru
  `ifstream` – pro vstupní souborové operace
  `ofstream` – pro výstupní souborové operace
- Tyto třídy jsou odvozeny ze základních tříd pro standardní vstup a výstup `ios`, `istream`, `ostream`
- Pro práci se souborovými vstupy a výstupy je třeba v záhlaví zdrojového souboru deklarovat `#include <fstream>` a `using namespace std;`

Zápis do souboru

- Pro výstup do souboru vytvoříme proměnnou typu `ofstream`
- Soubor otevřeme metodou `open()`, které předáme jméno souboru. Stejného výsledku dosáhneme i předáním jména souboru konstruktoru (v definici proměnné).
- Úspěšnost otevření souboru ověříme pomocí logického operátoru `!`, případně ekvivalentní metody `fail()`. Oboje vrací pravdivou hodnotu, pokud bylo otevření souboru neúspěšné
- Pro zápis dat používáme operátor `<<`
- Soubor můžeme uzavřít pomocí metody `close()`. Neučiníme-li tak, zavře ho destruktor třídy fstream (princip RAII)

```
#include <fstream>
using namespace std;

int main()
{
  ofstream ofile;           // Definujeme promennou proudu
  ofile.open("test.dat");   // Otevreme soubor
  If (!ofile) {             // Nebo take: if (ofile.fail())
    cout << "Nelze otevrit soubor!\n";
    return 1;
  }
  ofile << "Tento text se zapise do souboru" << endl;
  ofile.close(); // Nemusi byt, nechceme-li testovat uspesnost
  return 0;
}
```

Čtení ze souboru

- Pro čtení dat ze souboru vytvoříme proměnnou typu `ifstream`
- Jméno souboru předáme konstruktoru nebo metodě `open()`
- Úspěšnost otevření souboru ověříme pomocí ! nebo `fail()`
- Pro čtení dat používáme operátor `>>`
- Soubor můžeme ručně uzavřít pomocí metody `close()`

```
#include <fstream>
using namespace std;

int main()
{
  double a1 = 0.0, a2 = 0.0;
  ifstream ifile("test.dat"); // Muzeme pouzit i open()/close()
  if (!ifile)
  {
    cout << "Nelze otevrit soubor!\n";
    return 1;
  }
  ifile >> a1 >> a2;
  return 0;
}
```

Diagnostika I/O chyb

- Po otevření souboru a také po načtení nebo zápisu znaku potřebujeme zjistit, zda byla operace úspěšná. K tomu používáme následující metody:

  `fail()` – vrací **true**, pokud byla operace neúspěšná

  `eof()` – vrací **true**, pokud předchozí operace dosáhla konce souboru, bez ohledu na úspěšnost (týká se jen čtení)

  `good()` – vrací **true**, pokud byla poslední operace úspěšná a proud je dále použitelný (tedy nenastal `fail()` ani `eof()`)

```
ifstream ifile("test.dat");
if (ifile.fail())  // testujeme uspesnost otevreni souboru
{
  cout << "Nelze otevrit soubor!\n";
  return 1;
}

double a = 0;
ifile >> a;   // Nacitani hodnoty do promenne a
if (ifile.fail())  // testujeme uspesnost nacteni
{
  cout << "Nepodarilo se nacist hodnotu!" << endl;
}
```

Zrušení chybového stavu proudu

- Pokud byla předchozí operace neúspěšná (tj. `fail()` vrátí pravdivou hodnotu) je další načítání/zápis z/do proudu blokováno, tj. selžou všechny operace, které se o to pokusí
- Tento chybový stav musíme odstranit voláním metody `clear()`, teprve potom můžeme provádět další operace čtení a zápisu

```
double a;
string s;
ifstream ifile("test.dat");

// Nasledujici pokus o nacteni cisla bude neuspesny (napr. protoze
// ve vstupnim souboru se nachazeji neciselne znaky)
ifile >> a;
if (ifile.fail()) {
  cout << "Chyba pri cteni cisla." << endl;
  // Volanim clear() zrusime chybovy stav proudu
  ifile.clear();
}

// Nyni nacteme text. Pokud bychom predtim nezavolali clear(),
// zadny text by se nenacetl, protoze proud by byl v chybovem stavu.
ifile >> s;
cout << "Nacteny text: " << s << endl;
```

Proudy pro vstup a výstup z/do řetězce

- Data lze také načítat nebo zapisovat do řetězce, tj. proměnné typu `string`. K tomu používáme následující třídy:
  - `stringstream` – pro načítání/zápis z/do řetězce
  - `istringstream` – pro načítání z řetězce
  - `ostringstream` – pro zápis do řetězce
- Tyto třídy jsou odvozeny ze základních tříd pro standardní vstup a výstup `ios`, `istream`, `ostream`
- Pro práci se souborovými vstupy a výstupy je třeba v záhlaví zdrojového souboru deklarovat `#include <sstream>` a `using namespace std;`

Proudy pro vstup a výstup z/do řetězce

- Pro načítání dat z řetězce vytvoříme proměnnou typu `istringstream`, pro zápis `ostringstream`, a při inicializaci jim předáme jméno řetězcové proměnné
- Operátor `>>` používáme pro čtení, operátor `<<` pro zápis
- K diagnostice úspěšnosti načítání opět používáme `!` či metodu `fail()`

```
#include <sstream>
using namespace std;

int main()
{
  double a1 = 0.0, a2 = 0.0, a3 = 0.0;
  string s = "3.24 1.2 5.7";

  istringstream sstream(s);

  sstream >> a1 >> a2 >> a3;

  if (sstream.fail()) {
    cout << "Chyba pri nacitani z retezcoveho proudu" << endl;
  }
  return 0;
}
```

Proudy pro vstup a výstup z/do řetězce

- Pokud chceme načítat z jiného řetězce, můžeme použít stejný proud, do kterého nastavíme nový vstupní řetězec pomocí metody `str()`
- Poté zavoláme metodu `clear()`, abychom zrušili případnou chybu z předchozího načítání

```
#include <iostream>
#include <sstream>
using namespace std;

int main()
{
  double a1 = 0.0, a2 = 0.0, a3 = 0.0;
  double b1 = 0.0, b2 = 0.0, b3 = 0.0;
  string s1 = "3.24 1.2 5.7";
  string s2 = "2.0 4.5 6.0";
  istringstream sstream(s1);

  sstream >> a1 >> a2 >> a3;    // Ted nacitame z retezce s1
  sstream.str(s2);              // Do proudu nastavime retezec s2
  sstream.clear();              // Zrusime pripadny chybovy stav
  sstream >> b1 >> b2 >> b3;    // Ted nacitame z retezce s2

  return 0;
}
```

Načítání souboru po řádcích

- Pro načtení celého řádku z proudu používáme funkci `getline()`, která z proudu načte jeden řádek a přiřadí ho do řetězcové proměnné typu `string` předané jako druhý argument
- `getline()` vrací úspěch, pokud se podařilo něco načíst
- Tuto řetězcovou proměnnou nastavíme do příslušného řetězcového proudu metodou `set()` a načítáme

```
string fileName = "soubor.dat";
double a1 = 0.0, a2 = 0.0;
string s;
istringstream sstream;
ifstream ifile(fileName);
if (ifile.fail())
  { cout << "Nelze otevrit soubor: " << fileName << endl; return;}

while (getline(ifile, s)) { // Nacte jeden radek z proudu ifile do retezce s
  sstream.str(s);  // Proud sstream bude nacitat z retezce s
  sstream.clear(); // Zrusime predchozi chybovy stav
  sstream >> a1 >> a2;
}

if (ifile.fail() && !ifile.eof()) {
  // Neco se pokazilo a nebyl to jen konec souboru
  cout << "Chyba pri nacitani souboru!" << endl;
}
```

Datový proud jako argument funkce

- Proudovou proměnnou předáváme do funkcí vždy odkazem (referencí) – všechny vstupní a výstupní operace totiž musí probíhat nad stejným objektem proudu, ne jeho kopiemi vzniklými předáváním hodnotou

```
// Proud predavame do funkce odkazem (formou reference)
void read(istringstream &istream)
{
  double a1 = 0.0, a2 = 0.0;
  istream >> a1 >> a2;
}

int main()
{
  string s = "2.13  5.67";
  istringstream sstream;

  sstream.str(s);
  sstream.clear();
  read(sstream);  // Funkci predavame proud sstream, tady bez &

  return 0;
}
```

Přetížení funkcí a metod

- V jazyce C++ lze definovat více funkcí/metod se stejným jménem ale různým počtem nebo typem parametrů
- Takovéto funkce se nazývají přetížené (*overloaded*)
- Je-li funkce volána, překladač analyzuje předávané argumenty a podle nich vybere odpovídající funkci/metodu

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

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

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

int main()
{
  Circle circ;
  circ.setColor(3);          // Vola se prvni metoda
  circ.setColor("red");      // Vola se druha metoda
  return 0;
}
```

Přetížení funkcí - příklad

```
class FilledCicle
{
  public:
    void setValues(int ax, int ay);
    void setValues(int ax, int ay, int c);
    void setValues(int ax, int ay, int c, int fc);
};


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

void FilledCicle::setValues(int ax, int ay, int c)
{ setValues(ax, ay); color = c; }

void FilledCicle::setValues(int ax, int ay, int c, int fc)
{ setValues(ax, ay, c); fillColor = fc; }

int main()
{
  FilledCircle circ;
  circ.setValues(150, 230, 3);      // Vola se druha metoda
  circ.setValues(150, 230, 3, 17);  // Vola se treti metoda
  circ.setValues(150, 230);         // Vola se prvni metoda

  return 0;
}
```

Přetížení konstruktorů

- Velmi často se přetěžování užívá pro konstruktory

```
class Circle
{
public:
  Circle();
  Circle(int ax, int ay);
  Circle(int ax, int ay, int c);
private:
  int x = 0, y = 0, color = 1;
};

Circle::Circle() {} // vse dostane vychozi hodnoty

Circle::Circle(int ax, int ay) : x(ax), y(ay) {} // vychozi jen barva

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

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

Přetížené operátory

- Jazyk C a 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 téměř všechny dostupné operátory, nejčastěji se přetěžují operátory: `=`, `==`, `!=`, `<`, `>`, `<=`, `>=`, `[]`, `<<`, `>>`, `+`, `-`, `*`, `/`.
- V praxi vytváříme vlastní přetížené operátory pouze v odůvodněných případech, jejich nadužívání může program znepřehlednit (zejména tam, kde nelze význam operátoru snadno odhadnout z kontextu, např. „co vrací operátor + aplikovaný na dva objekty typu Clovek?“)

Přetížené operátory – příklad

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

// Operator vrati skalarni soucin dvou 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;

  result = v1 * v2;       // Tady se zavola operator*

  return 0;
}
```

Přetížené operátory jako metody

- Operátory pracující nad objekty implementujeme přednostně jako metody
- Volá se vždy metoda objektu na levé straně operátoru a jako argument se mu předá objekt na pravé straně (kromě unárních operátorů jako `!`, `->`, `++` a `--`, ty žádný druhý objekt nemají)

Přetížené operátory jako metody

```
class Vector2D
{
public:
  double operator*(const Vector2D &v) const;
  bool operator!(void) const;
};

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

bool Vector2D::operator!(void) const
{
  return (x == 0 && y == 0);
}

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

  result = v1 * v2;    // Tady se zavola operator* objektu v1 a jako parametr
                       // se mu preda objekt v2, tedy: result = v1.operator+(v2);

  if (!v1) {           // Tady se zavola operator! objektu v1: if (v1.operator!())
    cout << "Vektor v1 je nulovy!" << endl;
  }

  return 0;
}
```

Přetížené operátory proudů

```
// Na zacatku je definovana trida Vector2D

// 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; // Proud musime vzdy vratit, aby slo operace retezit
}

// Operator pro cteni promenne typu Vector2D z proudu
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;
  cout << v1 << v2 << v3; // Vola se vyse definovany operator <<
  cin >> v1 >> v2 >> v3;  // Vola se vyse definovany operator >>
  return 0;
}
```

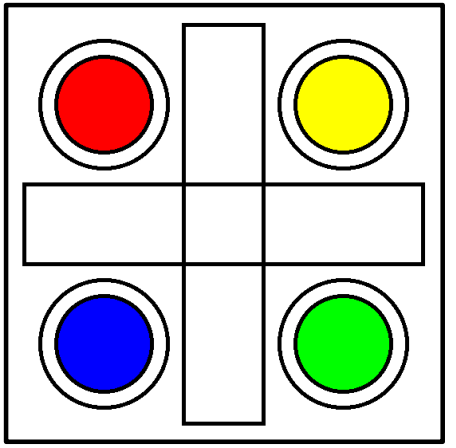
Dodržujte následující pravidla

- Pro práci se souborovými proudy nezapomeňte vložit hlavičkový soubor *fstream* a pro práci s řetězcovými proudy soubor *sstream*
- Nepoužívejte v programu globální proměnné, pokud to není nezbytně nutné. Upřednostněte lokální proměnné a ty předávejte do funkcí/metod. Proměnné související s funkcí nějakého objektu umístěte do příslušné třídy.
- Parametry objektových typů předávejte jako konstantní referenci, pokud neexistují důvody pro jiný přístup. Toto platí i pro řetězcové proměnné typu `string`. Proměnné proudů předávejte vždy nekonstantní referencí.
- Parametry základních typů (`int`, `double`, ...) předávejte obvyklým způsobem (tj. hodnotou a nekonstantní), pokud neexistují důvody pro jiný přístup.
- 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).

Cvičení – 1. část

1. Vytvořte program vycházející z programu v úloze 2 z minulého cvičení. Program uzpůsobte tak, že místo interaktivního vstupu bude údaje o zobrazovaném objektu načítat ze souboru. V souboru bude uveden typ obrazce, který se bude kreslit, souřadnice x, y a případně další potřebné hodnoty (poloměr pro kružnici, šířka a výška pro obdélník, poloměr a číslo barvy kružnice, číslo barvy výplně pro vyplněnou kružnici). Tyto hodnoty budou odděleny mezerami. V souboru bude uveden pouze jeden řádek s jedním objektem (například: *circle 150 150 60 3*). Jméno vstupního souboru bude programu předáno jako parametr na příkazovém řádku. Otestujte se soubory *circle.dat*, *rectangle.dat*, *filled_circle.dat* v adresáři */home/tootea/C3220/data/* **2 body**

2. Vytvořte program, který bude ze souboru načítat grafické objekty. Soubor bude obsahovat více řádků a každý řádek bude obsahovat všechny informace o jednom grafickém objektu ve stejném formátu, jako soubory z předchozí úlohy. Až po načtení celého souboru se všechny tyto objekty vykreslí v jednom okně. Na řádku s klíčovým slovem `window` bude specifikována velikost okna. Program otestujte se souborem */home/tootea/C3220/data/shapes1.dat* (měl by se vykreslit stejný obrazec jako na obrázku, nezáleží na pořadí vykreslování jednotlivých objektů). **3 body**



Úloha 1 – nápověda

- Do tříd Shape, `Circle`, `Rectangle` a `FilledCircle` implementujte metodu `void readFile(ifstream &istream)`, která bude vypadat podobně jako metoda `readValues()`, ale bude načítat data ze souborového proudu, který jí bude předán jako argument.
- Načítání souboru implementujte ve funkci `main()`, jak je uvedeno v příkladu v sekci “Načítání ze souboru”. Ze souboru načtěte řetězec do řetězcové proměnné a podle obsahu této proměnné (“circle”, “rectangle”, “filled_circle”) vytvořte objektovou proměnnou příslušného typu a zavolejte její metodu `readFile()` a následně volejte metodu `printValues()` a pak `draw()`
- Pro snazší vytváření objektů můžete všem třídám přidat prázdné bezparametrové konstruktory (v kombinaci s vhodnými inicializačními hodnotami v definici datových položek), pak lze psát jen `Circle c;` místo `Circle c(0, 0, 0, 0);`
- Argumenty předané programu na příkazovém řádku získáme stejně jako v jazyce C, tj. funkce main() bude přijímat dva parametry `main(int argc, char *argv[])`, kde `argc` obsahuje počet předaných parametrů zvětšený o 1, pole `argv[]` obsahuje seznam parametrů (`argv[0]` obsahuje název souboru s programem, teprve `argv[1]` obsahuje první předaný parametr.)

Úloha 2 – nápověda

- Metodu `readFile()` ve všech objektech modifikujte tak, že bude přijímat proud `istringstream` namísto `ifstream`, tj: `void readFile(istringstream &istream)`. Na začátek souboru nezapomeňte vložit hlavičkový soubor *sstream*.
- Ve funkci `main()` definujte proměnnou pro vektor kružnic `vector<Circle> circles`. Podobně definujte vektory `rectangles` a `filledCircles`.
- Vstupní soubor načítejte v cyklu po řádcích, jak je uvedeno v příkladu v sekci “Načítání souboru po řádcích”. Další postup je podobný jako v předchozí úloze, tj. načtete první slovo a podle toho rozhodnete, který objekt se bude načítat. Použijte lokální proměnnou příslušného typu, na ní zavolejte readFile() k načtení parametrů objektu a připravený objekt vložte do příslušného vektoru pomocí `push_back()`. Pokud je na začátku řádku uvedeno slovo “window”, načtou se rozměry okna (do lokálních proměnných).
- Po načtení celého souboru (a jeho uzavření) dojde k vykreslení všech objektů. Nejdříve se otevře okno (jeho šířka a výška jsou dány hodnotami ve vstupním souboru na řádku „window“) a potom se v cyklu vykreslí všechny načtené kružnice (tj. zavolá se `circle.draw(dev)` pro každou položku `circle` v poli). Potom se totéž provede pro čtverce a vyplněné kružnice. Nakonec program čeká na stisknutí Enter a pak okno zavře.
- Program vylepšete použitím třídy `Drawing`, která bude mít podobnou roli jako v úloze 3 z předchozího cvičení, konkrétně bude obsahovat vektory objektů `circles, filledCircles` a `rectangles` a proměnné pro šířku a výšku okna. Dále bude obsahovat metodu pro načtení dat ze souboru `readFile(string fileName)`, která přijímá jméno souboru jako parametr a metodu `draw()`, která otevře okno a vykreslí do něj všechny grafické objekty. Ve funkci `main()` vytvořte objekt typu `Drawing` a zavolejte jeho metodu `readFile()` a potom `draw()`.

Cvičení – 2. část

3. Upravte program z úlohy 2 následujícím způsobem:
   - Ve třídě Shape implementujte statickou 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 s názvem barvy (původní variantu metody však také ponechte).
   - V každé ze tříd Shape, `Circle`, `FilledCircle` a `Rectangle` implementujte dva konstruktory, jeden bez parametrů (ponechávající výchozí hodnoty určené v těle třídy) a jeden s parametry, které bude ukládat do členů dané třídy
   - 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 `shapes2.dat` (v adresáři `/home/tootea/C3220/data/`), který se od `shapes1.dat` liší tím, že jsou v něm barvy specifikovány formou textu místo čísel. **2 body**

Cvičení – 3. část

4. 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()` (výchozí nulové souřadnice) 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 navíc přijímat parametr typu `string`, který se uživateli vypíše před tím než se načtou/vypíší souřadnice.
   - Operátor * pro skalární součin a operátor + pro součet vektorů.
   - Samostatnou funkci `swapVectors()` (ne metodu), která zamění dva vektory (hodnoty jejich souřadnic)
   - 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 pokud možno jako metody.

   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 zadané vektory a vypíše jejich souřadnice.

2 body

Cvičení – 4. část

5. Předchozí program modifikujte tak, že v něm implementujete operátory **>>** a **<<** pro načítání/zápis do proudu. Program načte ze souboru `vectors.dat` (v adresáři `/home/tootea/C3220/data/`) souřadnice dvou vektorů a do výstupního souboru zapíše souřadnice součtu těchto dvou vektorů. Jména vstupního a výstupního souboru budou specifikována na příkazovém řádku. Pro načítání a zápis do souboru využijte implementované operátory. **nepovinná, 2 body**