# 8. Načítání a zápis PDB souboru

## Třída `string`

Typ `string` není základním vestavěným typem ale je implementován jako třída ve standardní knihovně C++.

Třída `string` obsahuje některé užitečné metody:

`length()` - vrátí počet znaků v řetězci
`size()` - totéž jako length()
`operator[](int pos)` – vrátí znak na pozici pos
`substr(int pos, int n)` – vrátí podřetězec dlouhý `n` znaků začínající na pozici `pos`
`c_str()` - vrátí řetězec jak je používan v C, tj. typ `char*` (např. pro předání řetězce funkcím které akceptují pouze klasické řetězce `char*` a nepodporují `string`)

Podrobnější informace lze nalézt na:
http://www.cplusplus.com/reference/string/string/

Uvedené metody nekontrolují platnost řetězce, tj. pokud do proměnné typu `string` nepřiřadíme žádný řetězec, je řetězec nedefinován a pokus o práci s ním vede k chybě programu.

Uvedené metody nekontrolují velikost řetězce, tj. před zavoláním metody musíme zkontrolovat velikost řetězce.

## Formátování výstupu

Pro formátování výstupů používáme tzv. manipulátory. Pro použití manipulátorů je třeba v záhlaví zdrojového souboru deklarovat `#include <iomanip>` a using namespace std;

Manipulátory se používají ve spojení s operátorem <<

Manipulátory nastavují formátování a různé parametry pro načítání vstupu a výstupu.

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

int main()
{
    // Vypise cele cislo zarovnane
    // doprava, minimalne 5 znaku,
    // (zleva se doplni mezery)
  cout << right << setw(5) << 234 << endl;

    // Nasledujici prikaz vypise realne
    // cislo s platnosti na 2
    //desetinna mista, tj. 456.15
  cout << fixed << setprecision(2)
       << 456.15738 << endl;

  return 0;
}
```

## Manipulátory k I/O formátování

Následující manipulátory lze použít pro formátování výstupu:

`fixed` – výstup reálného čísla ve formátu 123.45
`scientific` – výstup reálného čísla ve formátu1.235E2
Výchozí chování je takové, že se vybere *fixed* nebo *scientific* tak aby vypsaná přesnost byla co nejvyšší.

`left` – zarovnává výpis do leva
`right` – zarovnává výpis do prava
`setw(n)` - nastaví minimální počet vypisovaných znaků n pro nejbližší operaci výpisu (pro celá čísla nebo řetězce).
`setfill(c)` – nastaví použití znaku c pro vyplnění výpisu pokud je šířka (nastavená pomocí setw()) větší než je vyžadováno.
`setprecision(n)` - nastavuje počet číslic za desetinnou tečkou (pro výpis *fixed* a *scientific* se vypíše přesně `n` cifer za desetinnou tečkou a případně doplní zprava nulami, při výchozím/implicitním nastavení se jedná pouze o maximální celkový počet cifer před a za desetinnou tečkou).

Následující manipulátory lze použít při načítání ze vstupu:

`skipws` – nastaví přeskakování bílých znaků (mezera, tabelátor, konec řádku) při načítání (toto je výchozí nastavení)
`noskipws` – nastaví že bílé znaky nebudou přeskakovány
`ws` – načítá bílé znaky tak dlouho dokud nenarazí na nebílý znak

Podrobný výčet manipulátorů se nachází na:
http://www.cplusplus.com/reference/iostream/manipulators/

```
ATOM      7  CG2 THR A   1      18.159  11.531   6.124  1.00 10.28           C
ATOM  ····7··CG2·THR·A···1·····18.159··11.531···6.124··1.00·10.28·········C
1   5    10   15   20   25   30   35   40   45   50   55   60   65   70   75
```

## Načítání PDB souboru v C++

PDB formát se používá pro ukládání struktur biomolekul v PDB databázi www.rcsb.org.

Data PDB souboru jsou organizována jako *fixed format*, tj. každá položka má přesně udanou pozici na řádku.

Dokumentace k PDB formátu je dostupná na: http://www.wwpdb.org/docs.html

Každý řádek PDB souboru obsahuje na začátku 6 znaků identifikujících typ dat na daném řádku.

Souřadnice atomů se načítají z řádků ATOM (pro standardní residua) a HETATM (pro nestandardní residua).

Načítání PDB souboru v C++ je podobné jako v C, ale využívají se proměnné typu `string` a odpovídající metody (např. `substr()`, `length()`), proudy `stringstream` a jejich metody (operátor >>, `str()`, `clear()`, `fail()`.

Zápis PDB souboru v C++ je podobný jako v C, zapisuje se do proudu pomocí operátoru << a pro formátování se používají manipulátory (`fixed`, `left`, `right`, `setw()`, `setprecission()`).

### Struktura záznamu ATOM a HETATM

| | |
|---|---|
| 1 – 6 | Jméno záznamu ATOM nebo HETATM (6 znaků) |
| 7 – 11 | Pořadové číslo atomu (celé číslo) |
| 12 | Mezera |
| 13 – 16 | Jméno atomu (řetězec max. 4 znaky) |
| 17 | Alternativní pozice atomu (znak) |
| 18 – 20 | Jméno residua (řetězec max. 3 znaky) |
| 21 | Mezera |
| 22 | Identifikátor protein. řetězce (znak) |
| 23 – 26 | Pořadové číslo residua v sequenci (celé číslo) |
| 27 | i_code - Kód indikující vložení residua (znak) |
| 28 – 30 | Tři mezery |
| 31 – 38 | Souřadnice x v Å (reálné číslo) |
| 39 – 46 | Souřadnice y v Å (reálné číslo) |
| 47 – 54 | Souřadnice z v Å (reálné číslo) |
| 55 – 60 | Occupancy (reálné číslo) |
| 61 – 66 | Teplotní faktor (reálné číslo) |
| 77 – 78 | Symbol prvku (řetězec max. 2 zn. zarov. doprava) |
| 79 – 80 | Formální náboj na atomu (2 znaky ve tvaru 2+, 1-) |

## Třídy pro načítání PDB souboru

Informace načítané z řádků ATOM a HETATM ukládáme to třídy `Atom`, která obsahuje datové členy pro uložení informace o jednom atomu a odpovídající metody.

```
  // Pro moznost vypisu na standardni
  // vystup (cout) vlozime:
#include <iostream>
  // Pro praci se souborovymi proudy
  // (ifstream a ofstream) vlozime:
#include <fstream>
  // Pro praci s retezcovymi proudy
  // (istringstream) vlozime:
#include <sstream>
  // Pro praci s manipulatory vlozime:
#include <iomanip>
  // Pro praci s kontejnetem vector<>
  // vlozime:
#include <vector>
  // Pro primy pristup ke jemnum
  // promennych a funkci standardni
  // knihovny deklarujeme jmenny
  // prostor std:
using namespace std;

class Atom
{
  // Cleny tridy Atom
};

int main(int argc, char *argv[])
{
}
```

## Třída `Atom`

```
class Atom
{
  public:
      // Nasledujici staticke konstatntni
      // promenne slouzi k nastaveni
      // hodnoty v recordType (viz. nize)
    static const int RECORD_UNKNOWN = 0;
    static const int RECORD_ATOM = 1;
    static const int RECORD_HETATM = 2;
    // Konstruktor
    Atom();

    // Metoda pro nacteni radku
    void readLine(const string &line);

    // Metoda pro zapis radku do PDB
    // souboru
    void writeLine(ofstream &ofile);

    // Metoda pro vypis testovacich
    // informaci o atomu na stand. vystup
    void print();

    // Zde mohou byt dalsi metody, napr.
    // pro pristup k datovym clenum
    // napr. getAtomNumber(),
    // getAtomName(), getRresidueName()

  private:
    int recordType;
    int atomNumber;
    string atomName;
    char alternateLocation;
    string residueName;
    char chainId;
    int residueNumber;
    char iCode;
    double coordX; // souradnice x
    double coordY; // souradnice y
    double coordZ; // souradnice z
    double occupancy;
    double tempFactor;
    string elementName;
    string formalCharge;
      // Nasledujici promenne indikuji,
      // zda-liByla nactena occupancy a
      // teplotni faktor
    bool isOccupancy;
    bool isTempFactor;
};
```

## Načítání řádků z PDB souboru

```
  string line;
  string recordName;
  ifstream ifile;
  Atom *atom = 0;
  vector<Atom*> atomsContainer;
  string inputPdbFileName =
                          "1jxy_noal.pdb";

  ifile.open(inputPdbFileName.c_str());
  if (ifile.fail())
  {
  cout << "Could not open input file!\n";
    return 1;
  }

  while (!ifile.eof())
  {
    getline(ifile, line);
    if (line.length() >= 6)
    {
      // Zkopirujeme prvnich 6 znaku do
      // recordName
      recordName = line.substr(0, 6);

      if (recordName == "ATOM  " ||
          recordName == "HETATM")
      {
        atom = new Atom;

        // Metoda readLine() nacte z radku
        // vsechna data pro dany atom
        atom->readLine(line);

        // Atom vlozime do kontejneru
        atomsContainer.push_back(atom);
      }
    }
  }
  ifile.close();
```

## Načtení záznamu ATOM a HETATM

```
// Funkce prijima jako argument radek
// obsahujici zaznam ATOM nebo HETATM
void Atom::readLine(const string &line)
{
  string recordName;
  string s;  // pomocna retezcova promenna
  istringstream sstream;

  // Radek by mel mit minimalne 53 znaku
  // aby obsahoval vsechny povinne udaje
  if (line.length() < 53)
  {
    cout<<"The line is too short!" <<endl;
    return;
  }

  // Zkopirujeme prvnich 6 znaku do
  // recordName
  recordName = line.substr(0, 6);
  if (recordName == "ATOM  ")
    recordType = RECORD_ATOM;
  else if (recordName == "HETATM")
    recordType = RECORD_HETATM;
  else return;  // Neni-li to ATOM ani
    // HETATM nelze pokracovat

  // Budeme nacitat cislo atomu
  // 5 znaku od pozice 6 se zkopiruje do s
  s = line.substr(6, 5);
  // Do retezcoveho proudu nastavime
  // retezec s, ze ktereho se bude cist
  sstream.str(s);
  // Odstranime pripadny chybovy stav
  // proudu z pripadneho predchoziho cteni
  sstream.clear();
  // Nacitame cislo atomu
  sstream >> atomNumber;
  if (sstream.fail())
  {
    cout << "Error reading file!" << endl;
    return;
  }

  // Nyni nacteme jmeno atomu
  // Z retezce line se zkopiruji 4 znaky
  // od pozice 12 do atomName
  atomName = line.substr(12, 4);

  // Nacteme alternate location indicator
  // Zkopiruje se znak na pozici 16
  alternateLocation = line[16];

  // Pokracovani na dalsim sloupci ...
```

```
// Pokracovani metody readLine()
// z predchoziho sloupce

  // Zde je treba doplnit kod pro nacteni
  // dat do promennych residueName,
  // chainId, residueNumber, iCode

  // Nacteme souradnice x, y, z
  s = line.substr(30, 24);
  sstream.str(s);
  sstream.clear();
  sstream >> coordX >> coordY >> coordZ;
  if (sstream.fail())
  {
    cout << "Error reading file!" << endl;
    return;
  }

  // Nacteme occupancy, ale pouze pokud je
  // radek delsi nez
  if (line.length() >= 60)
  {
    s = line.substr(54, 6);
    sstream.str(s);
    sstream.clear();
    sstream >> occupancy;
    if (!sstream.fail())
      isOccupancy = true;
  }

  // Zde je treba doplnit kod pro nacteni
  // tempFactor, elementName, formalCharge
  // Nesmime zapomenout predtim vzdy
  // zkontrolovat delku radku
}
```

## Zápis do PDB souboru

```
  // Nekde ve funkci main() nebo ve vhodne
  // funkci/metode bude umisten kod
  // ktery otevre vystupni soubor a potom
  // bude pro jednotlive atomy volat
  // nasledujici metodu

void Atom::writeLine(ofstream &ofile)
{
  // Zapiseme jmeno zazn. ATOM nebo HETATM
  if (recordType == RECORD_ATOM)
    ofile << "ATOM  ";
  else if (recordType == RECORD_HETATM)
    ofile << "HETATM";
  else
    return;

  // Zapiseme cislo atomu, 5 znaku
  // zarovnanych do prava
  ofile << right << setw(5) << atomNumber;

  // Zde je v PDB vzdy mezera
  ofile << ' ';

  // Zapiseme jmeno atomu, 4 znaky
  // zarovnane do leva
  ofile << left << setw(4) << atomName;

  // Zapiseme jeden znak alternate
  // location
  ofile << alternateLocation;

  // Zapiseme jmeno residua, 3 znaky
  // zarovnane do leva
  ofile << left << setw(3) << residueName;

  // Zde je v PDB vzdy mezera
  ofile << ' ';

  // Zapiseme znak identifikujici retezec
  ofile << chainId;

  // Zapiseme cislo residua, 4 znaky
  // zarovnane do prava
  ofile << right << setw(4) <<
           residueNumber;

  // Zapiseme jeden znak insert code
  ofile << iCode;

  // Zde jsou v PDB vzdy 3 mezery
  ofile << "   ";

 // Pokracovani na dalsim sloupci ...
```

```
  // Pokracovani metody writeLine()
  // z predchoziho sloupce

  // Zapiseme kartezske souradnice atomu
  // zarovnane doprava s fixnim poctem
  // cislic za desetinnou teckou
  // (manipulator fixed), pocet
  // desetinnych mist bude 3 a celkovy
  // pocet zapsanych znaku je 8
  // (vcetne desetinne tecky)
  ofile << right << fixed
        << setprecision(3);
  ofile << setw(8) << coordX;
  ofile << setw(8) << coordY;
  ofile << setw(8) << coordZ;

  // Zapiseme occupancy nebo mezery
  if (isOccupancy)
    ofile << right << fixed
          << setprecision(2) << setw(6)
          << occupancy;
  else
    ofile << "      ";

  // Zapiseme teplotni faktor
  if (isTempFactor)
    ofile << right << fixed
          << setprecision(2) << setw(6)
          << tempFactor;
  else
    ofile << "      ";

  // Zde je v PDB souboru vzdy 10 mezer
  ofile << "          ";

  // Zapiseme jmeno prvku
  ofile << right << setw(2) <<elementName;

  // Zapiseme formalni naboj
  ofile << left << setw(2) <<formalCharge;

  // Zapiseme znak konce radku
  ofile << endl;
}
```

## Třída `Application`

V jazyce C++ často používáme třídu `Application`, která je hlavní třídou programu a soustřeďuje nejdůležitější data a metody.

Ve funkci `main()` pak pouze vytvoříme objekt typu `Application` a zavoláme jeho metodu `run()`, která obsahuje veškerý hlavní kód.

```
class Application
{
  public:
    Application();

    // Destruktor, v nem je treba uvolnit
    // dynamicky alokovanou pamet
    ~Application();

    // V metode run() bude veskery hlavni
    // kod, ktery jsme drive umistovali
    // do funkce main()
    int run(int argc, char *argv[]);
    bool readPdbFile();
    void writePdbFile();
  private:
    string inputPdbFileName;
    string outputPdbFileName;
    vector<Atom*> atomsContainer;
};

int main(int argc, char *argv[])
{
  Application app;
  return app.run(argc, argv);
}
```

## Dodržujte následující pravidla

- V konstruktoru vždy inicializujte datové členy třídy (obvykle se inicializují jen proměnné základních typů, proměnné objekt. typů obvykle není třeba inicializovat, protože jsou inicializovány ve svých konstruktorech).
- V destruktoru vždy uvolněte dynamicky alokovanou paměť (týká se jen situace, kdy ukazatele ukazující na tuto paměť jsou členy třídy).
- Program vytvářejte postupně, nejdříve vytvořte definice tříd a metody, jejich těla ponechejte prázdná a program přeložte. Potom přidávejte kód, vždy po dokončení důležité části program přeložte a otestujte.

## Úloha 1 6 bodů

Vytvořte program který načte atomy z PDB souboru (řádky ATOM a HETATM) a zapíše je do jiného souboru.

- Jméno vstupního a výstupního souboru bude specifikováno na příkazovém řádku
- Program otestujte se souborem /home/mprokop/C3220/data/1jxy_noal.pdb
- Objekty typu `Atom` alokujte vždy dynamicky (pomocí operátoru new) a ukládejte do kontejneru `vector<>`
- V programu použijte třídu `Application`
- V destruktoru třídy `Application` nezapomeňte uvolnit paměť alokovanou pro atomy a vyprázdnit kontejner
- Chybové hlášky upravte tak, aby nahlásily na kterém řádku ve vstupním PDB souboru chyba nastala

**Nápověda:**

Vložte hlavičkové soubory (viz. příklad v sekci "Načítání PDB").

Definujte třídu `Atom` jak je uvedeno v příkladu v sekci "Třída Atom".

Vytvořte metod třídy Atom, jejich obsah prozatím ponechejte prázdný. Vytvořte funkci `main()` a program zkompilujte.

V konstruktoru `Atom()` inicalizujte proměnné třídy: `recordType` inicializujte hodnotou RECORD_UNKNOWN, číselné proměnné (int, double) inicializujte 0, znakové proměnné (char) je v tomto případě vhodné inicializovat znakem mezery. Řetězcové proměnné je možné inicializovat prázdným řetězcem, v tomto případě je však vhodnější je inicializovat mezerami odpovídající šířce záznamu (tj. `atomName` inicializujeme řetězcem o 4 znacích, `residueName` o 3 znacích a `elementName` a `formalCharge` o 2 znacích). Proměnné `isOccupancy` a `isTempFactor` a inicializujte hodnotou `false`.

Ve funkci `main()` zapište kód vycházející z příkladu v sekci "Načítání řádků z PDB souboru". Na konci vypište počet atomů v kontejneru a v cyklu volejte mrtodu `print()` pro každý atom. V metodě `print()` implementujte kód pro jednoduchý výpis hlavních vlastností atomu (číslo atomu, jméno atomu, souřadnice, symbol prvku).

Implementujte metodu `readLine()` podle příkladu uvedeného v sekci "Načtení záznamu ATOM a HETATM". Do příkladu je však dále potřeba implementovat načtení hodnot `residueName`, `chainId`, `residueNumber`, `iCode`, `tempFactor`, `elementName`, `formalCharge`. Nezapomeňte, že v PDB speifikaci jsou pozice záznamů číslovány od 1, zatímco znaky řetězce jsou v C/C++ číslovány od 0.

Implementujte metodu `writeLine()` podle příkladu uvedeného v sekci "Zápis do PDB souboru". Ve funkci `main()` otevřete výstupní soubor a volejte metodu `writeLine()` pro každý atom v kontejneru. Funkčnost otestujte se souborem *1jxy_noal.pdb*, zapisovaný soubor by měl být zcela totožný s načítaným souborem (toto ověřte unixovým příkazem *diff*).

Program nyní předělajte tak, aby obsahoval třídu `Application` jak je ukázáno na příkladu v sekci "Třída `Application`".

V konstruktoru inicializujte proměnné třídy (např. přiřaďte jména souborů do proměnných `inputPdbFileName` a `outputPdbFileName`).

V destruktoru uvolněte dynamicky alokovanou paměť (vymažte každý atom v kontejneru a vynulujte velikost kontejneru – viz. minulé cvič., příklad v sekci "Kontejner ukazatelů").

Kód pro načtení PDB souboru přesuňte z původní funkce `main()` do metody `readPdbFile()`, nezapomeňte odstranit lokální proměnné `atomsContainer` a `inputPdbFileName`, protože místo nich se budou používat obdobné proměnné, které jsou členy třídy `Application`. Metoda bude vracet hodnotu `false`, pokud nebylo načtení úspěšné, jinak vrátí true.

Kód pro zápis PDB souboru přesuňte do `writePdbFile()`.

V metodě `run()` otestujte počet argumentů (nezapomeňte, že první argument je název programu, takže argc je vždy rovno minimálně 1). Pokud nebyl žádný argument specifikován, ukončete program. Pokud byl specifikován jeden argument (název vstupního PDB souboru), načtěte PDB soubor a na výstop vypište testovací informace. Pokud byly specifikovány dva argumenty, pak navíc zapište PDB soubor (jeho jméno je dáno druhým argumentem). Nezapomeňte zkopírovat argumenty z `argv[]` do proměnných `inputPdbFileName` a `outputPdbFileName`.

Abychom mohli při chybě vypisovat číslo řádku, je potřeba v metodě `Application::readPdbFile()` přidat celočíselnou proměnnou jejíž hodnota ze vždy zvětší po načtení řádku o 1 a tu předávat do metody `Atom::readLine()`, která potom její hodnotu využije při výpisu chybové zprávy.