Pokročilé programování v jazyce C pro chemiky

(C3220)

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;
}
```

Manipolátory k I/O formátování

- Následující manipulátory lze použít pro formátování výstupu:

  `fixed` - výstup reálného čísla ve formátu 123.45

  `scientific` – výstup reálného čísla ve formátu 1.2345E2

  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).

Manipulátory k I/O formátovaání

- 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/

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

```
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
```

1 - 6	"ATOM "	
7 - 11	serial	Atom serial number
13 - 16	name	Atom name
17	altLoc	Alternate location indicator
18 - 20	resName	Residue name
21		
22	chainID	Chain identifier
23 - 26	resSeq	Residue sequence number
27	iCode	Code for insertion of residues
28 - 30		
31 - 38	x	Coordinates for X in Angstroms
39 - 46	y	Coordinates for Y in Angstroms
47 - 54	z	Coordinates for Z in Angstroms
55 - 60	occupancy	Occupancy
61 - 66	tempFactor	Temperature factor
77 - 78	element	Element symbol, right-justified
79 - 80	charge	Charge on the atom

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` - datové členy

```
class Atom
{
  public:
      // Tady budou verejne metody
  private:
    int recordType;     // Rozliseni radku ATOM a HETATM
    int atomNumber;
    string atomName;
    char alternateLocation;
    string residueName;
    char chainId;              // Znak identifikujici retezec
    int residueNumber;         // Cislo residua
    char iCode;
    double coordX, coordY, coordZ;  // Kartezske souradnice atomu
    double occupancy;
    double tempFactor;
    string elementName;
    string formalCharge;
    bool isOccupancy;         // Byla nactena occupancy?
    bool isTempFactor;      // Byl nacten teplotni faktor?
};
```

Třída Atom – metody a statické členy

```
class Atom
{
  public:
      // Nasledujici staticke konstatntni promenne slouzi k nastaveni
      // hodnoty v recordType
    static const int RECORD_UNKNOWN = 0;
    static const int RECORD_ATOM = 1;
    static const int RECORD_HETATM = 2;
  public:
      // Konstruktor – zde inicializujeme vsechny datove cleny
      // Retezce je vhodne inicializovat prazdnym retezcem "" nebo
      // v tomto pripade lepe mezerami, tj. pro jmeno atomu 4 mezery,
      // pro jemno residua 3, pro jmeno prvku 2
    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 standardni vystup
    void print();

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

Načítání řádků z PDB souboru

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

ifile.open(inputPdbFileName.c_str());
  // Zde musi byt osetreni chybneho otevreni souboru

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

Načtení záznamu ATOM a HETATM – část 1

```
void Atom::readLine(const string &line)
{
  string recordName, s;  // s je pomocna retezcova promenna
  istringstream sstream;

  if (line.length() < 53) { cout<<"Prilis kratky radek!"<<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

    // Nacteme cislo atomu
  s = line.substr(6, 5);  // 5 znaku od pozice 6 se zkopiruje do s
  sstream.str(s);   // Do retezcoveho proudu nastavime retezec s
  sstream.clear();  // Odstranime pripadny chybovy stav proudu
  sstream >> atomNumber;   // Nacitame cislo atomu
  if (sstream.fail()) { /*Nahlasime chybu a vyskocime z metody.*/ }

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

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

Načtení záznamu ATOM a HETATM – část 2

```
  // Pokracovani metody readLine() z predchozi stranky

  // Zde se nactou data do promennych residueName, chainId,
  // residueNumber, iCode

    // Nacteme souradnice - nacteme vsechny tri najednou
  s = line.substr(30, 24);
  sstream.str(s);
  sstream.clear();
  sstream >> coordX >> coordY >> coordZ;
  if (sstream.fail()) { /*Nahlasime chybu a vyskocime, vratime false*/ }

    // Nacteme occupancy
  if (line.length() >= 60)  // Pouze pokud je radek delsi nez 60 znaku
  {
    s = line.substr(54, 6);
    sstream.str(s);
    sstream.clear();
    sstream >> occupancy;
    if (!sstream.fail())
      isOccupancy = true;  // Occupancy byla uspesne nactena
      // Occupancy je nepovinna, takze i pri nenacteni muzeme pokracovat
  }

  // Zde nacteme tempFactor, elementName, formalCharge
  // Nezapomeneme predtim vzdy zkontrolovat delku radku
}   // Konec metody readLine()
```

Zápis do PDB souboru – část 1

```
  // Nekde ve funkci main nebo 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 zaznamu ATOM nebo HETATM
  if (recordType == RECORD_ATOM)
    ofile << "ATOM  ";
  else if (recordType == RECORD_HETATM)
    ofile << "HETATM";
    // Zapiseme cislo atomu, 5 znaku zarovnanych do prava
  ofile << right << setw(5) << atomNumber;
  ofile << ' ';   // Zde je v PDB vzdy mezera
    // 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;
  ofile << ' ';   // Zde je v PDB vzdy mezera
  ofile << chainId;  // Zapiseme jeden znak identifikujici retezec
    // Zapiseme cislo residua, 4 znaky zarovnane do prava
  ofile << right << setw(4) << residueNumber;
  ofile << iCode;  // Zapiseme jeden znak insert code
  ofile << "   ";   // Zde jsou v PDB vzdy 3 mezery
    // Pokracovani na dalsi strance
```

Zápis do PDB souboru – část 2

```
  // Pokracovani metody write_line() z predchozi stranky

    // 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 (vc. desetinne tecky)
  ofile << right << fixed << setprecision(3);
  ofile << setw(8) << coordX;
  ofile << setw(8) << coordY;
  ofile << setw(8) << coordZ;

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

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

  ofile << "          ";  // Zde je v PDB souboru vzdy 10 mezer
  ofile << right << setw(2) << elementName;
  ofile << left << setw(2) << formalCharge;

  ofile << endl;    // Zapiseme znak konce radku
} // Konec metody writeLine()
```

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();
    ~Application(); // Destruktor, zde se uvolni alokovana pamet
      // 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 objektových 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.

Cvičení

1. 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

6 bodů