Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ZBH Zentrum für Bioinformatik

Prof. Dr. M. Rarey, M. von Behren **28.11.2013**

7. Übung zur Vorlesung Grundlagen der Chemieinformatik

Abgabe bis 06.12.2013 an gci-uebung@zbh.uni-hamburg.de

Aufgabe 1: (P) (4 Punkte)

1 Einleitung

Für den Einstieg in die Programmiersprache `C++` ist es zunächst wichtig, dass Sie einige grundlegende Konzepte dieser Sprache verstehen. Der größte Unterschied zwischen `C++` und `C` ist, dass es sich bei `C++` um eine objektorientierte Programmiersprache handelt. Die folgenden Passagen sollen Ihnen zunächst einen kleinen Einblick in die Handhabung von `C++` geben und Ihnen helfen die erste Programmieraufgabe zu lösen.

2 Objektorientierte Programmierung (Klassen)

In `C++` ist es möglich, eigene Datenstrukturen zu erstellen, indem man sogenannte Klassen definiert. Diese Klassen dienen dazu, unterschiedliche Daten zu speichern und über sogenannte Methoden zur Verfügung zu stellen. Hierbei unterscheidet man zwischen `public` Methoden, die jederzeit auf einem Objekt der Klasse ausgeführt werden können und `private` Methoden, die nur innerhalb der Klasse (intern) verwendet werden können. Unter einem Objekt versteht man in der objektorientierten Programmierung ein Exemplar eines bestimmten Datentyps bzw einer bestimmten Klasse. Ein primärer Fokus beim Design von `C++` lag darauf, es zu ermöglichen Klassen-Typen zu definieren, die sich genauso natürlich verwenden lassen wie eingebaute Typen (z.B. `int, float und char`). Um komplett zu verstehen, wie Klassen definiert werden und was man alles mit ihnen machen kann, sind sehr viele Informationen nötig. Daher wird das Erstellen einer eigenen Klasse erst auf einem späteren Übungszettel behandelt. Um jedoch eine bereits existierende Klasse zu verwenden sind lediglich drei Informationen notwendig:

1. Wie ist der Name der Klasse?
2. Wo ist die Klasse definiert?

3. Welche Methoden stellt die Klasse zur Verfügung?

Beispiel: Eine Klasse die als Teil der Standard-Bibliotheken von `C++` zur Verfügung gestellt wird ist die `String`-Klasse. Die `String`-Klasse speichert intern eine Sequenz von Charactern und bietet verschiedene Methoden an, um auf diese zuzugreifen. Definiert ist diese Klasse in der Datei `string.h`, somit reicht ein Include dieser Datei aus, um die `String`-Klasse verwenden zu können. Da es sich bei `String` um eine Klasse aus der Standard-Bibliothek handelt, reicht hier sogar "`#include <string>`" aus. Um eine Übersicht zu bekommen, welche Methoden diese Klasse zur Verfügung stellt, kann man nun entweder in die Datei hinein schauen oder bei Klassen der Standard-Bibliothek in die Online Dokumentation schauen (http://www.cplusplus.com/reference/string/string/). Jede Klasse verfügt zunächst über einen Konstruktor mit dem sich eine Variable bzw. ein Objekt des Klassentyps erzeugen lässt. Um zum Beispiel ein Objekt der Klasse `String` zu erzeugen kann man schreiben: `std::string name(‘‘Programmierung’’);` Hierbei können wie bei Funktionsaufrufen in den runden Klammern Parameter übergeben werden. Die Variable `name` hat nun den Typ `String` und enthält das Wort "Programmierung". Auf dieser Variable lassen sich alle Methoden ausführen, die durch die Klasse zur Verfügung gestellt werden.
Zur Verwendung einer Methode auf einem Objekt fügt man den Namen der Funktion zusammen mit eventuellen Übergabeparametern (in runden Klammern) mit einem Punkt an den Variablennamen an. Eine Ausnahme bilden hierbei Pointer. Methoden auf Pointern, welche auf ein Objekt einer bestimmten Klasse zeigen, werden wie Elemente in einem `C-Struct` mit einem Pfeiloperator aufgerufen ("->"). Die meisten in Naomini vorkommenden Klassen wie `Molecule, Atom` und `Bond` liegen in der Regel als Pointer vor.
Beispiele:
`name.size();` - gibt die Länge des Strings als integer zurück.
`name.append(‘‘ ist super!’’);` - hängt die gegebenen Character an den `String` an.
`name.at(3);` - gibt den vierten Character des `Strings` zurück (Aufzählung beginnt bei 0).
`name.clear();` - löscht den Inhalt des `Strings`.

3 Namespaces

In `C++` Programmen ist es möglich, verschiedene Bibliotheken und die darin definierten Klassen für ein Programm zu verwenden, um die darin enthaltene Funktionalität nicht selbst implementieren zu müssen. Hierbei kann es geschehen, dass verschiedene Bibliotheken überschneidende Funktionalität enthalten oder für unterschiedliche Funktionalität derselbe Name verwendet wurde. In einem solchen Fall würde es demnach zu Konflikten kommen, da der Compiler nicht weiß, welche von zwei gleichnamigen Klassen oder Funktionen im Code gemeint ist. Um dieses Problem zu umgehen, gibt es die sogenannten Namespaces. Im Idealfall liegen alle Klassen und Funktionen einer zusammenhängenden Bibliothek im selben Namespace. So liegen zum Beispiel alle Klassen der Standard-Bibliothek im Namespace "`std`" und alle Klassen der Naomini-Bibliothek im

gleichnamigen Namespace "`Naomini`". Den Namespace gibt man vor dem Namen einer Klasse bzw. Funktion mit zwei Doppelpunkten an. Man würde also zum Beispiel bei der Verwendung der `String`-Klasse aus der Standard-Bibliothek "`std::string`" schreiben.
Beispiele:
```
std::string
std::cout
std::vector
Naomini::MoleculePtr
Naomini::AtomPtr
```
Achten Sie bei der Verwendung von Bibliotheken also darauf nicht nur den Klassennamen, sondern auch den entsprechenden Namespace dazu zu schreiben.
Tipp: Um sich den Aufwand zu sparen sehr häufig verwendete Namespaces wie in unserem Fall den Naomini Namespsace immer wieder vorweg schreiben zu müssen, können Sie am Anfang ihres Programms "`using namespace Naomini;`" schreiben und ihn dann im Folgenden weglassen. Hierbei muss jedoch beachtet werden, dass die Verwendung von mehreren `using namespace` Anweisungen dazu führen kann, dass die Namensüberschneidungen erneut auftreten.
Achtung: Niemals ein `using namespace` in einer Header-Datei verwenden.

In dieser Übung sollen Sie sich mit den Methoden der Standard-Bibliothek für Ein- und Ausgabe beschäftigen und einen ersten Eindruck von der Verwendung von Klassen und ihren Methoden erhalten.
Der Befehl für die Standardausgabe lautet `std::cout`. Die gewünschte Textausgabe sowie auszugebende Variablen lassen sich einfach mit dem "`<<`" Operator aneinander hängen. Für einen Zeilenumbruch kann man entweder ein Newline oder ein `std::endl` anhängen.
Eine besondere Form der Standardausgabe ist `std::cerr` für die Ausgabe von Fehlermeldungen. Im Gegensatz zu `std::cout` wird hierbei die Ausgabe nicht zuerst in einem Puffer zwischengespeichert. Dadurch wird sichergestellt, dass auch bei einem Programmabsturz Fehlermeldungen noch direkt ausgegeben werden.
Zur Verwendung der Standardeingabe wird der `std::cin` Befehl verwendet. Eingaben auf der Kommandozeile werden hierbei mithilfe des "`>>`" Operators direkt einer entsprechenden Variablen zugewiesen.
Tipp: Neben den grundlegenden Variablen wie `int` und `float` lassen sich auch viele Klassen der Standardbibliothek wie zum Beispiel der `std::string` direkt ausgeben. Dies hängt davon ab, ob für eine Klasse zu diesem Zweck der "`<<`" Operator implementiert wurde.
Beispiele:
```
std cin >> myIntVariable;
std::cout << ‘‘Hello World’’ << std::endl;
std::cout << ‘‘Die Zahl ist ’’ << myIntVariable << std::endl;
std::cerr << ‘‘Wrong input.‘‘ << std::endl;
```

Im Verzeichnis `Exercises/InputOutput/` finden Sie die Datei `InputOutput.cpp`, fügen Sie hier bitte Ihren Code zur Lösung der folgenden Aufgaben ein. Zum Kompilieren und Ausführen Ihres Programms benutzen Sie bitte folgende Aufrufe:

```
make ioExercise
./ioExercise MoleculeFiles/some_dud_ligands.smi
```

- In der `Main`-Funktion befindet sich ein `printf` Befehl zur Ausgabe des Namens eines geladenen Moleküls. Ersetzen Sie diesen Befehl durch ein `std::cout`.
- Fügen Sie eine Fehlermeldung ein für den Fall, dass beim Funktionsaufruf eine falsche Anzahl von Parametern übergeben wird.
- Lesen Sie über die Standardeingabe eine Zahl ein. Diese soll entweder 0 oder 1 sein (Eingabe überprüfen) und in einer Variable gespeichert werden.
- Implementieren Sie in Abhängigkeit von der Eingabe verschiedene Ausgaben. Bei der Eingabe 0 sollen alle verfügbaren Informationen über das Molekül ausgegeben werden und bei der Eingabe 1 die Eigenschaften aller Atome. Schauen Sie in die Header der `Naomini/Molecule.hpp` und der `Naomini/Atom.hpp` Klasse um herauszufinden, welche Informationen durch Methoden abgerufen werden können.

Zur Bewertung dieser Aufgabe geben Sie bitte den von Ihnen geschriebenen Code ab!

Aufgabe 2: (T) (8 Punkte)

Wie Sie bereits wissen, bietet die SMARTS Sprache als Erweiterung von SMILES die Möglichkeit, Substrukturen zu spezifizieren. Das komplette Konzept der Sprache finden Sie unter:
`http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html`
Zum Üben bietet es sich an, das Tutorial von Daylight durchzuarbeiten:
`http://www.daylight.com/dayhtml_tutorials/languages/smarts/index.html`
Bei der Erstellung von SMARTS-Ausdrücken kann es helfen, sich diese visualisiert mit dem SMARTSviewer anzuschauen: `http://smartsview.zbh.uni-hamburg.de`

1. Im Wirkstoffdesign werden molekulare Muster oft als Filter genutzt, um aus einer Datenbank eine Teilmenge von kleinen Molekülen zu identifizieren, die bestimmte Eigenschaften oder Substrukturen besitzen. Entwerfen Sie für jede der folgenden molekularen Muster einen SMARTS-String:
   - kovalente Halogenverbindungen
   - heterozyklische Ringsysteme aus drei Atomen
   - an aromatische Systeme gebundene Nitro-Gruppen

- jedes Atom, das sich in einer Seitenkette einer α Aminosäure befindet

Testen Sie die erstellten SMARTS-Ausdrücke, z. B. unter Verwendung der Webseite `http://www.daylight.com/daycgi_tutorials/depictmatch.cgi` und der nachstehenden SMILES-Strings. Geben Sie in Ihrer Lösung an, welche Moleküle von keinem Ihrer Filter erfasst werden.

```
Brc1cc(ccc1NC(=O)C1CC21CCC2)[N+](=O)[O-]
O=[N+]([O-])C1CC=CCC1c2cccnc2
CSCC[CH](C(=O)O)N
C(C)(C)CCC(F)(F)(F)
CC1OC1
CCCOc1c(N)cc([N+](=O)[O-])cc1
c1ccc2c(c1)c(c[nH]2)C[CH](C(=O)O)N
Nc1cc(Cl)c(N)c([N+](=O)[O-])c1
[Na+].[Cl-]
c1ccc2c(c1)[nH]c1ccccc21
CCC1CP1C(C)(C)C
CC1O[N+](=O)c2ccccc2C1
O=C(N)CC(N)C(=O)O
I-I
```

2. Vereinfachen Sie den folgenden SMARTS-Ausdruck soweit wie möglich, ohne die Bedeutung zu verändern `[c]-,:[c&!R0][n;!X4;X3]`

3. Beschreiben Sie das Muster, das durch den folgenden SMARTS-Ausdruck beschrieben wird, mit eigenen Worten (beide Zeilen bilden einen Ausdruck). Auf Moleküle welcher Substanzklasse passt es?
   `CC12CC([C;$(C-F),$([CH1])]3C(C1CC[C@@]2(C(=O)C[OH1])[OH1])C[C;`
   `$(C-C),$([CH1])]C4=C[C;$(C=O),$(C-[OH1])]C-,=CC34C)-,=O`

4. Erstellen Sie für den umrandeten Teil der Molekülstruktur ein SMARTS-Pattern, welches exklusiv auf diesen Teil des Moleküles passt: