Institut für Informatik
der Universität München
Prof. Dr. M. Hofmann
Dr. J. Johannsen, Dr. M. Lange

WS 03/04

22.1.04

# Übungen zur Vorlesung
# **Informatik I**

Blatt 12

Abgabe der Hausaufgaben spätestens am 2.2.04, 11:00 Uhr. Programmieraufgaben über
`http://miles.tcs.informatik.uni-muenchen.de/info1/abgabe.php`,
schriftliche Aufgaben auf Papier im Briefkasten in der Theresienstraße 39, 1. Stock. Notieren Sie Namen, Matrikelnummern und Ihre Übungsgruppe auf den Blättern. Bearbeitung in Gruppen zu max. 3 Personen ist zulässig. Besprechung der Aufgaben in den Übungen ab 9.02.04.

Im Rahmen dieses Übungsblattes werden Sie ein kleineres Projekt bestehend aus mehreren Modulen programmieren. Das Ziel ist ein Programm, welches einen arithmetischen Ausdruck einliest, ggf. normalisiert, den Syntaxbaum grafisch darstellt und den Ausdruck klammersparend ausgibt.

Einiges ist bereits programmiert, u.a.

- Die Signatur `syntax.mli` enthält Definitionen der Datenstrukturen für arithmetische Ausdrücke und deren Syntaxbäume.
- Das Archiv `provided.cma` enthält u.a. kompilierte Versionen der folgenden Module:
  - `Parse` stellt eine Funktion zur Verfügung, welche aus einer Liste von Token einen Syntaxbaum erstellt.
  - `Graph` enthält Funktionen zur grafischen Darstellung.
- Die Datei `main.ml` enthält das Hauptprogramm, welches den Ausdruck einliest, usw. Es erwartet einen oder zwei Parameter auf der Kommandozeile. Sind zwei gegeben, muss einer der beiden `-n` sein. Dieser gibt an, dass der Ausdruck normalisiert werden soll. Der andere Parameter ist der arithmetische Ausdruck. Aufruf z.B. als `main -n "(23-x)*(54+y)"`.
- Die Datei `Makefile` ist nützlich, um die einzelnen Module automatisch zu kompilieren und zu linken. Die Benutzung erfolgt folgendermassen. Stellen Sie sicher, dass es die Datei `.depend` gibt und rufen Sie `make depend` auf. Dies prüft Abhängigkeiten der einzelnen Module untereinander. Rufen Sie dann `make` auf. Dies kompiliert das Hauptprogramm und alle dazu nötigen Module. Mit `make clean` können Sie bereits kompilierte Dateien wieder löschen.
- Die Datei `main` enthält eine (unter Linux) ausführbare Version der Projektes. Diese können Sie benutzen, um sich an Beispielen die Funktionalität zu verdeutlichen.

Auf der WWW-Seite der Vorlesung finden Sie einen Link zu einem Verzeichnis mit allen Dateien, die Sie für dieses Projekt brauchen. Sie können die Dateien einzeln oder als ZIP- oder TAR-Archiv herunterladen.

Schauen Sie sich alle Dateien an. Diese enthalten in Kommentaren Erklärungen zu den bereitgestellten Funktionen und zeigen auch an, was Sie noch programmieren müssen.

Halten Sie sich auf jeden Fall an die in den Signaturen angegebenen Typen und erklären Sie in Kommentaren Ihre Funktionen!

**Programmieraufgabe P-47** (`lex.ml`): **6 Punkte**
In diesem Modul muss noch die Funktion `lex : string -> Syntax.token list` implementiert werden, die aus einem eingegebenen String die Liste der Tokens, die die in dem String vorhandenen syntaktischen Objekte darstellen, erzeugt.

Dabei sollen neben den Klammern und Operatorsymbolen nur Leerzeichen, Kleinbuchstaben und Ziffern vorkommen, jedes andere Zeichen löst die Ausnahme `IllegalSymbol` aus. Jeder maximale Teilstring, der nur aus Buchstaben und Ziffern besteht und mit einem Buchstaben beginnt, ist ein Variablenname.

Beispiel: `lex "+)x2y1 65("` ergibt die Ausgabe:
`[TokOp Plus; TokR; TokVar "x2y1"; TokNum 65; TokL]`

**Programmieraufgabe P-48** (`norm.ml`): **8 Punkte**
In diesem Modul sind zwei Funktionen zu implementieren:

- Die Funktion `normalise : Syntax.expr -> Syntax.expr` soll einen arithmetischen Ausdruck vereinfachen, indem beispielswiese ein Produkt von Summen ausmultipliziert wird, oder Teilausdrücke, die nur Zahlkonstanten enthalten, ausgerechnet werden. Wie weit Sie die Ausdrücke vereinfachen, bleibt dabei Ihnen überlassen. Für die schönste Vereinfachungsfunktion gibt es als Sonderpreis eine Tafel französischer Bitterschokolade und eine Siegerehrung in der Vorlesung.
- Die Funktion `string_of_expr : Syntax.expr -> string` nimmt einen Syntaxbaum, und gibt den dadurch dargestellten Ausdruck als Zeichenkette zurück. Dabei sollten Klammern nur dort gesetzt werden, wo sie wirklich nötig sind, also `3*(x+y)+12*z-13` statt `((3*(x+y))+(12*z))-13`.

**Programmieraufgabe P-49** (`visualise.ml`): **6 Punkte**
In diesem Modul muss noch die Funktion `draw_expr : Syntax.expr -> Graph.point -> Graph.point -> Graph.drawitem list` implementiert werden. Diese nimmt einen Syntaxbaum und zwei geometrische Punkte und liefert eine Liste über dem Typ `drawitem`. Dabei wird ein Knoten des Syntaxbaumes mit Beschriftung $x$ und den Unterbäumen $l$ und $r$ zu einem Textfeld mit dem Inhalt $x$ und zwei Linien von diesem Textfeld zu den beiden Unterbäumen.

Die beiden Argumente vom Typ `point` geben die linke untere und die rechte obere Ecke eines Rechtecks an, in das der Baum (von der Funktion `draw_it` im Modul `graph.ml`) gezeichnet wird. Aus diesen Koordinaten müssen Sie

- das Rechteck gedanklich in drei Teile teilen: einen für das Textfeld und jeweils einen für den linken und den rechten Teilbaum. Dafür können Sie z.B. die Höhe und die Grösse der Teilbäume bestimmen, um das Rechteck proportional aufzuteilen.
- Koordinaten für das Textfeld sowie für die Endpunkte der beiden Linien bestimmen.

Für die schönste grafische Ausgabe gibt es als Sonderpreis eine Packung Haribo Colorado, ebenfalls mit Siegerehrung.