Karlsruher Institut für Technologie
Lehrstuhl für Programmierparadigmen
Compilerpraktikum WS 2012/13
Dozent: Prof. Dr.-Ing. G. Snelting
Betreuer: Andreas Zwinkau zwinkau@kit.edu

Übungsblatt 3 | Ausgabe: 31.10.2012 | Besprechung: 7.11.2012

Aufgabe 1: *Parser*

1.1 *SLL(k)-Grammatik*

- Die Grammatik aus dem Sprachbericht lässt sich nicht direkt mit rekursiven Abstieg implementieren. Warum?
- Formen Sie die Grammatik in eine geeignete SLL(k)-Grammatik um und geben Sie diese ab. Wie groß ist k?

1.2 *Parser*

Entwickeln Sie aus der Syntaxbeschreibung im Sprachbericht von MiniJava einen Parser. Dieser soll für beliebige Textdateien entscheiden können, ob die Syntax der Datei der Sprache MiniJava entspricht. Eine Semantik und Typprüfung hat selbstverständlich noch nicht zu erfolgen.

Wenden Sie den Parser auf die Beispielprogramme an und entscheiden Sie welche Programme eine korrekte Syntax haben. Notieren Sie den ersten aufgetretenen Fehler bei Programmen mit unkorrekter Syntax. Welche dieser Fehler lassen sich einfach beheben?

1.3 *Testen*

Erstellen Sie jeweils 5 (nicht notwendigerweise sinnvolle) Programme mit korrekter Syntax und 5 mit inkorrekter Syntax. Tauschen Sie Ihre Beispiele mit den anderen Gruppen aus und vergleichen Sie die Ergebnisse.

Aufgabe 2: *Zusatzaufgaben*

Im Rahmen eines universitären Praktikums können immer nur die wichtigsten Aspekte eines Compilers betrachtet werden. Oft gibt es aber viele weitere interessante Themen in Theorie und Praxis bei denen sich eine tiefergehende Beschäftigung lohnt. Mit den Zusatzaufgaben wollen wir einige Anregungen in diese Richtung liefern. Die Zusatzaufgaben sind natürlich vollkommen freiwillig.

2.1 *Fehlermeldungen*

Ein wichtiger Punkt in der Praxis ist die Qualität der Fehlermeldungen. Gute Fehlermeldungen wirken sich meist direkt auf die Produktivität von Entwicklern aus, da er Fehler schneller lokalisieren, verstehen oder Programmierfehler vermeiden kann.

- Die meisten Compiler haben zusätzliche Kategorien von Meldungen wie „Warnungen“, „Fehler“ oder „fatale Fehler“. Was sind die Unterschiede?
- Dem Benutzer sollten Hilfestellungen zur Lokalisierung der Fehler gegeben werden. Dazu sind Informationen wie Zeilennummer, Spalte oder auch Beginn und Ende der Fehlerstelle nützlich.
- Fehler treten in verschiedenen Phasen des Compilers auf. Die meisten Nutzer bevorzugen Fehlermeldung aber in der Reihenfolge wie sie im Programm auftreten.
- Gibt es weitere wichtige Aspekte für gute Fehlermeldungen?

2.2 *Unterschiedliche Zeichensätze*

Programme werden heutzutage meist weltweit von internationalen Teams entwickelt und die Benutzerschnittstellen auf lokale Gegebenheiten angepasst. Compiler sind hier meist recht unkritisch da wenig mit dem Benutzer interagiert werden muss. Das Wichtigste hier ist wohl die Unterstützung verschiedener Zeichensätze. In den verschiedenen Ländern, Kulturen und leider auch Betriebssystemen haben verschiedene Zeichensätze durchgesetzt. Zwar wird ASCII heutzutage so gut wie überall unterstützt, viele Sprachen benötigen aber weitere Zeichen die in ASCII nicht enthalten sind. Typische ASCII-Erweiterungen im europäischen Raum sind ISO8859-15. Heutzutage wird meist der Unicode Standard mit 2^{24} möglichen Zeichen eingesetzt, der alle relevanten Schriftsysteme unterstützt.

- Wo spielen verschiedene Zeichensätze bei einem Compiler eine Rolle?
- Sollten Zeichenketten in der Sprache erweiterte Zeichen enthalten dürfen?
- Sollten Bezeichner erweiterte Zeichen enthalten dürfen?
- Wie kodiert man Bezeichner und Zeichenketten innerhalb des Compilers, welchen Code erzeugt man und wie ist die Eingabe kodiert?
- Wie erkenne ich die Kodierung der Eingabe?
- Werden bei der lexikalischen Analyse Automaten eingesetzt, so führen die 2^{24} Zeichen des Unicode Standards zu extrem großen Tabellen. Was kann man dagegen tun?