
ALBERT LUDWIGS UNIVERSITÄT FREIBURG

TECHNISCHE FAKULTÄT

PicoC-Compiler

Übersetzung einer Untermenge von C in den Befehlssatz der RETI-CPU

BACHELORARBEIT

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreuung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis	I
Codeverzeichnis	II
Tabellenverzeichnis	III
Definitionsverzeichnis	IV
Grammatikverzeichnis	V
1 Motivation	1
1.1 RETI-Architektur	2
1.2 PicoC	2
1.3 Eigenheiten der Sprache C	2
1.4 Gesetzte Schwerpunkte	3
1.5 Richtlinien	3
1.6 Zu dieser Arbeit	3
Literatur	A

Abbildungsverzeichnis

1.1	Schritte zum Ausführen eines Programmes mit dem GCC	1
1.2	Stark vereinfachte Schritte zum Ausführen eines Programmes	1

Codeverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

1.1	Deklaration	2
1.2	Definition	2
1.3	Allokation	3
1.4	Initialisierung	3
1.5	Scope	3
1.6	Call by value	3
1.7	Call by reference	3

Grammatikverzeichnis

1 Motivation

Als Programmierer kommt man nicht drumherum einen **Compiler** zu nutzen, er ist geradezu **essentiell** für den Beruf oder das Hobby des Programmierens. Selbst in der Programmiersprache L_{Python} , welche als interpretierte Sprache bekannt ist, wird das in der Programmiersprache L_{Python} geschriebene Programm vorher zu **Bytecode** kompiliert, bevor dieser von der **Python Virtual Machine (PVM)** interpretiert wird.

Compiler, wie der **GCC**¹ oder **Clang**² werden üblicherweise über eine **Commandline-Schnittstelle** verwendet, welche es für den Benutzer **unkompliziert** macht ein Programm, dass in der Programmiersprache geschrieben ist, die der Compiler kompiliert³ zu **Maschinencode** zu kompilieren.

Meist funktioniert das über schlichtes und einfaches **Angeben der Datei**, die das Programm enthält, welches kompiliert werden soll, z.B. im Fall des **GCC** über `> gcc file.c -o machine_code`⁴. Als Ergebnis erhält man im Fall des **GCC** die mit der Option `-o` selbst benannte Datei `machine_code`, welche dann zumindest unter **Unix** über `> ./machine_code` **ausgeführt** werden kann, wenn das **Ausführungsrecht** gesetzt ist. Das gesamte gerade erläuterte Vorgehen ist in Abbildung 1.1 veranschaulicht.

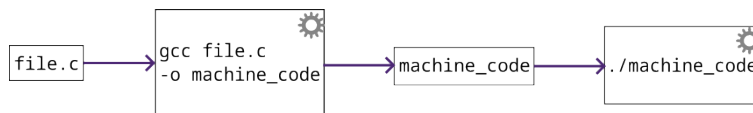


Abbildung 1.1: Schritte zum Ausführen eines Programmes mit dem GCC

Der ganze Kompilervorgang kann, wie er in Abbildung 1.2 dargestellt ist zu einer Box abstrahiert werden. Der Benutzer gibt ein **Programm** in der Sprache des Compilers rein und erhält **Maschinencode**, den er dann im besten Fall in eine andere Box hineingeben kann, welche die passende **Maschine** oder den passenden **Interpreter** in Form einer **Virtuellen Maschine** repräsentiert, der bzw. die den **Maschinencode** ausführen kann.

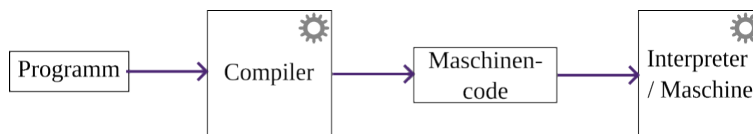


Abbildung 1.2: Stark vereinfachte Schritte zum Ausführen eines Programmes

¹GCC, the GNU Compiler Collection - GNU Project.

²clang: C++ Compiler.

³Im Fall des **GCC** und **Clang** ist es die Programmiersprache L_C .

⁴Bei **mehreren Dateien** ist das ganze allerdings etwas komplizierter, weil der **GCC** beim Angeben aller `.c`-Dateien nacheinander `gcc file_1.c ... file_n.c` nicht darauf achtet doppelten Code zu entfernen. Beim **GCC** muss am besten mittels einer **Makefile** dafür gesorgt werden, dass jede Datei einzeln zu **Objectcode** (Definition ??) kompiliert wird. Das Kompilieren zu **Objectcode** geht mittels des Befehls `gcc -c file_1.c ... file_n.c` und alle **Objectdateien** können am Ende mittels des **Linkers** mit dem Befehl `gcc -o machine_code file_1.o ... file_n.o` zusammen gelinkt werden.

Der Programmierer muss für das Vorgehen in Abbildung 1.2 **nichts** über die **Theoretischen Grundlagen des Compilerbau** wissen, noch wie der Compiler **intern** umgesetzt ist. In dieser Bachelorarbeit soll diese **Compilerbox** allerdings geöffnet werden und anhand eines eigenen im Vergleich zum **GCC** im Funktionsumfang **reduzierten Compilers** gezeigt werden, wie so ein Compiler **unter der Haube** stark vereinfacht funktionieren könnte.

Die konkrete **Aufgabe** besteht darin einen sogenannten **PicoC-Compiler** zu implementieren, der die **Programmiersprache** L_{PicoC} , welche eine Untermenge der Sprache L_C ist⁵ in eine zu **Lernzwecken** prädestinierte, **unkompliziert** gehaltene **Maschinensprache** L_{RETI} kompilieren kann. Im Unterkapitel 1.1 wird näher auf die **RETI-Architektur** eingegangen, die der Sprache L_{RETI} zu Grunde liegt und im Unterkapitel 1.2 wird näher auf die auf die Sprache L_{PicoC} eingegangen, welche der **PicoC-Compiler** zur eben erwähnten Sprache L_{RETI} kompilieren soll.

1.1 RETI-Architektur

Die **RETI-Architektur** ist eine zu Lernzwecken für die Vorlesungen P. D. C. Scholl, „Betriebssysteme“ und P. D. C. Scholl, „Technische Informatik“ entwickelte **32-Bit** Architektur, die sich vor allem durch ihre einfache Zugänglichkeit kennzeichnet und deren **Maschinensprache** als Zielsprache des **PicoC-Compilers** hergenommen wurde. In der Vorlesung P. D. C. Scholl, „Technische Informatik“ wird die **grundlegende RETI-Architektur** erklärt und in der Vorlesung P. D. C. Scholl, „Betriebssysteme“ wird diese Architektur erweitert, sodass diese mehr darauf angepasst ist, dass auch komplexere Konstrukte, wie ein **Betriebssystem**, **Interrupts**, **Funktionen** usw. auf nicht zu komplizierte Weise implementiert werden können.

In der **RETI-Architektur** ist alles simpel gehalten, es gibt Register, deren Bedeutungen in Tabelle ?? genauer erklärt werden, da manche dieser Register später Erwähnung finden und es gibt Maschinenbefehle für deren Bedeutung allerdings auf die Vorlesung ?? zu verweisen ist, da diese Maschinenbefehle zwar später vorkommen, aber ihre konkrete Aufgabe. Für die genauen Implementierungsdetails ist allerdings auf die Vorlesungen P. D. C. Scholl, „Technische Informatik“ und P. D. C. Scholl, „Betriebssysteme“ zu verweisen.

Der **Aufbau** der Maschinensprache ist in Grammatik ?? dargestellt.

1.2 PicoC

Der **Aufbau** der Sprache ist in Grammatik ?? dargestellt.

1.3 Eigenheiten der Sprache C

Definition 1.1: Deklaration

a

^aP. D. P. Scholl, „Einführung in Embedded Systems“.

Definition 1.2: Definition

a

^aP. D. P. Scholl, „Einführung in Embedded Systems“.

⁵Die der **GCC** kompilieren kann.

Definition 1.3: Allokation*a*^aThiemann, „Einführung in die Programmierung“.**Definition 1.4: Initialisierung***a*^aThiemann, „Einführung in die Programmierung“.**Definition 1.5: Scope***a*^aThiemann, „Einführung in die Programmierung“.**Definition 1.6: Call by value***a*^aBast, „Programmieren in C“.**Definition 1.7: Call by reference***a*^aBast, „Programmieren in C“.

1.4 Gesetzte Schwerpunkte

Die **Laufzeit** ist bei Compilern zwar vor allem in der Industrie **nicht unwichtig**, aber bei **Compilern**, verglichen mit **Interpretern** weniger zu gewichten, da ein Compiler bei einem fertig implementierten Programm nur **einmal** Maschinencode generieren muss und dieser Maschinencode danach fortan ausgeführt wird. Beim einem **Compiler** ist daher eher zu priorisieren, dass der kompilierte **Maschinencode** möglichst **effizient** ist.

Beim **PicoC-Compiler** wurde eher darauf Wert gelegt **sauberen, strukturierten Code** zu schreiben, den die Studenten sogar selber verstehen könnten und eine **unkomplizierte Bibliothek mit guter Dokumentation**⁶, nämlich das **Lark Parsing Toolkit**⁷ für das **Parsen** zu verwenden. Vor allem, da zu erwarten ist, dass der **PicoC-Compiler** vielleicht in einigen anderen Projekten eingebunden werden könnte, ist es von **Vorteil** bei der Notwendigkeit kleiner **Erweiterungen**, diese Erweiterungen **unkompliziert** durchführen zu können.

1.5 Richtlinien

1.6 Zu dieser Arbeit

⁶Welcome to Lark's documentation! — Lark documentation.⁷Lark - a parsing toolkit for Python.

Literatur

Online

- *clang: C++ Compiler*. URL: <http://clang.org/> (besucht am 29.07.2022).
- *GCC, the GNU Compiler Collection - GNU Project*. URL: <https://gcc.gnu.org/> (besucht am 13.07.2022).
- *Welcome to Lark's documentation! — Lark documentation*. URL: <https://lark-parser.readthedocs.io/en/latest/> (besucht am 31.07.2022).

Vorlesungen

- Bast, Prof. Dr. Hannah. „Programmieren in C“. Vorlesung. Vorlesung. Universität Freiburg, 2020. URL: <https://ad-wiki.informatik.uni-freiburg.de/teaching/ProgrammierenCplusplusSS2020> (besucht am 09.07.2022).
- Scholl, Prof. Dr. Christoph. „Betriebssysteme“. Vorlesung. Vorlesung. Universität Freiburg, 2020. URL: https://abs.informatik.uni-freiburg.de/src/teach_main.php?id=157 (besucht am 09.07.2022).
- — „Technische Informatik“. Vorlesung. Vorlesung. Universität Freiburg, 3. Aug. 2022. (Besucht am 03.08.2022).
- Scholl, Prof. Dr. Philipp. „Einführung in Embedded Systems“. Vorlesung. Vorlesung. Universität Freiburg, 2021. URL: <https://earth.informatik.uni-freiburg.de/uploads/es-2122/> (besucht am 09.07.2022).
- Thiemann, Prof. Dr. Peter. „Einführung in die Programmierung“. Vorlesung. Vorlesung. Universität Freiburg, 2018. URL: <http://proglang.informatik.uni-freiburg.de/teaching/info1/2018/> (besucht am 09.07.2022).

Sonstige Quellen

- *Lark - a parsing toolkit for Python*. 26. Apr. 2022. URL: <https://github.com/lark-parser/lark> (besucht am 28.04.2022).