Praktikum C-Programmierung 2026

Crashkurs Compiler

Jannek Squar

2026-07-08

Scientific Computing
Universität Hamburg

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Outline

Table of Contents

- Übersetzt Quell-Sprache in Ziel-Sprache
- Ziel-Sprache ist erstmal beliebig:
  - Maschinencode
  - andere High-level-Sprache
  - Zwischencode: Intermediate Representation (IR)
  - aber auch: PDF, Grafik
- Transpiler
  - Compiler + Transformation
  - Modifiziert Quellcode auf gleichem Sprachlevel

  $\rightarrow$ Source-to-source Compiler

Warum interessiert uns das?

Blick unter die Haube notwendig?

- Kurz: In der Regel eigentlich nicht nötig
- Aber: U.U. wichtiges Arbeitsfeld
  - Compiler-Fehler sind schwer zu finden (Bsp.: AMD-Compiler auf LUMI)
  - Fortgeschrittene, maßgeschneiderte Optimierungen
  - Neue Hardware
  - Domain-specific Languages (DSLs)
  - Forschung

Hier:

- Betrachtung anhand von LLVM Clang
- Gnu Compiler Collection (GCC) ist ähnlich

Table of Contents

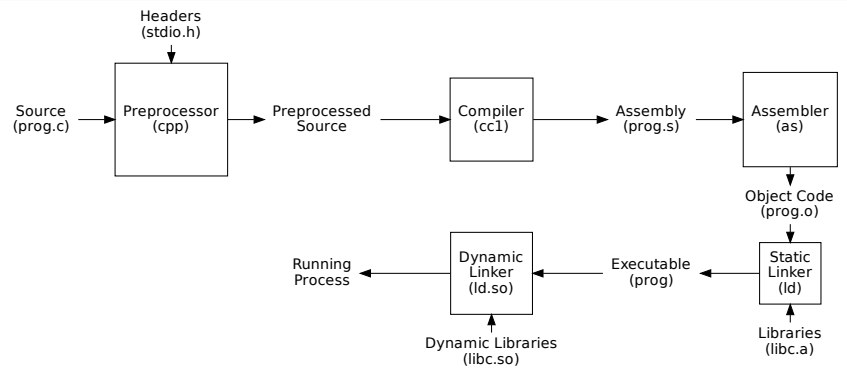


Abbildung 1: Stationen vom Source-Code zur Ausführung [Tha20]

Compiler Komponenten

Preprocessor

- Ersetzt Makros
- Fügt Header-Dateien ein
- Entfernt Kommentare

Compiler

- Scanner
- Parser
- Semantische Checks
- Optimierungen
- Code Generator

Compiler Komponenten

Preprocessor

- Ersetzt Makros
- Fügt Header-Dateien ein
- Entfernt Kommentare

Compiler

- Scanner
- Parser
- Semantische Checks
- Optimierungen
- Code Generator

Preprocessor

- Ersetzt Makros
- Fügt Header-Dateien ein
- Entfernt Kommentare

Compiler

- Scanner
- Parser
- Semantische Checks
- Optimierungen
- Code Generator

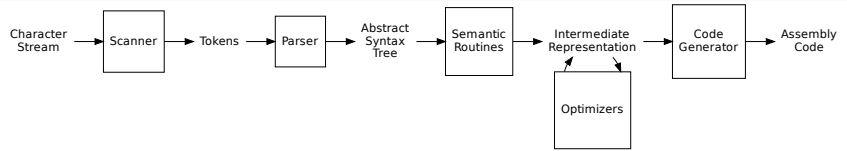
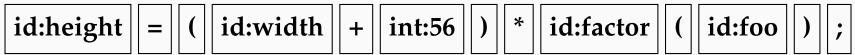


Abbildung 2: Compiler Toolchain [Tha20]

Source-Code:

```
1  height = (width+56) * factor(foo);
```

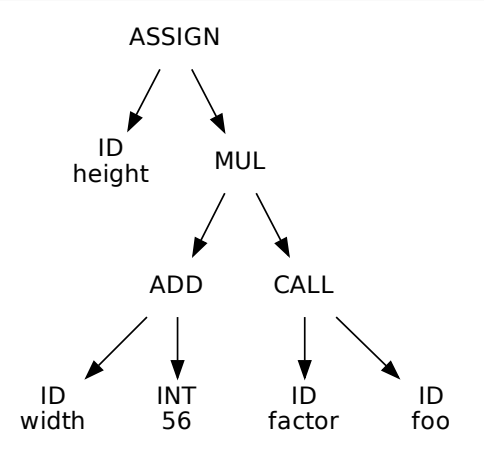
Tokenized:



- Überprüfung der Korrektheit anhand Grammatik-Regeln
- Ableitung AST durch Grammatikregeln

Abstract Syntax Tree (AST)

id:height	=	(	id:width	+	int:56	)	*	id:factor	(	id:foo	)	;



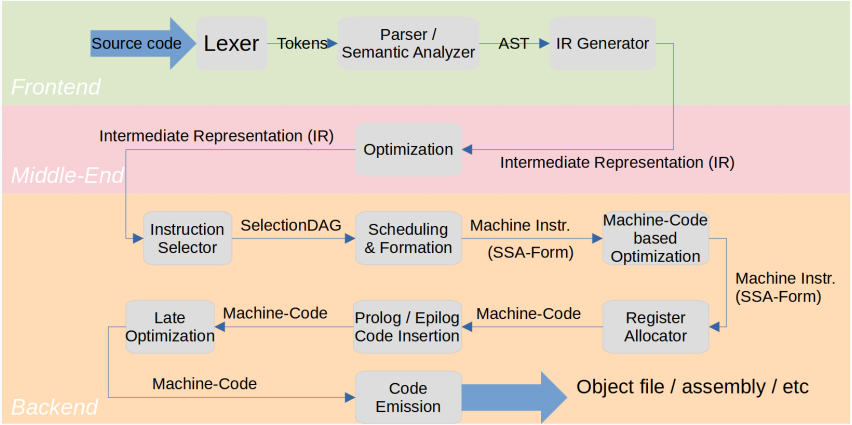


Abbildung 3: Compiler Toolchain Ausschnitt [Win22]

Notwendigkeit IR

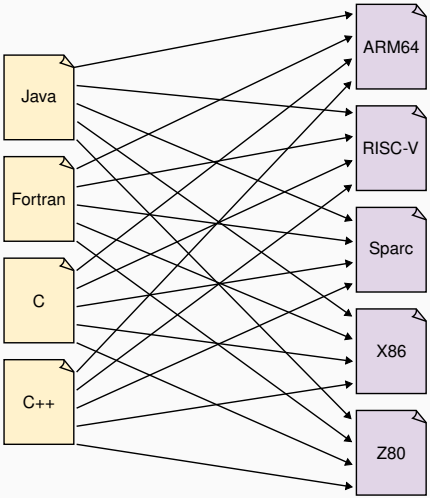


Abbildung 4: Ohne IR

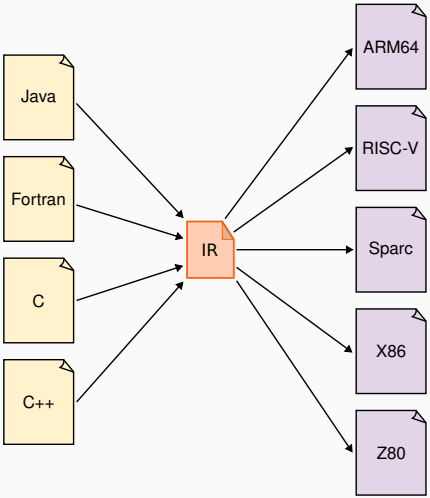


Abbildung 5: Mit IR

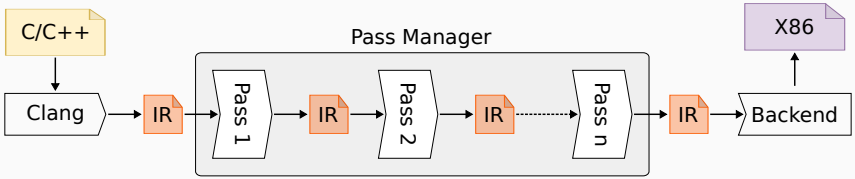


Abbildung 6: Optimierungen auf IR-Ebene

```c
#include <stdlib.h>

int main(int argc, char **argv) {
  int result = 0;
  int counter = 0;

  if (argc == 1)
    counter = 1;
  else
    counter = atoi(argv[1]);

  for (int i = 0; i < counter; i++) {
    result += 1;
  }
  return result;
}
```

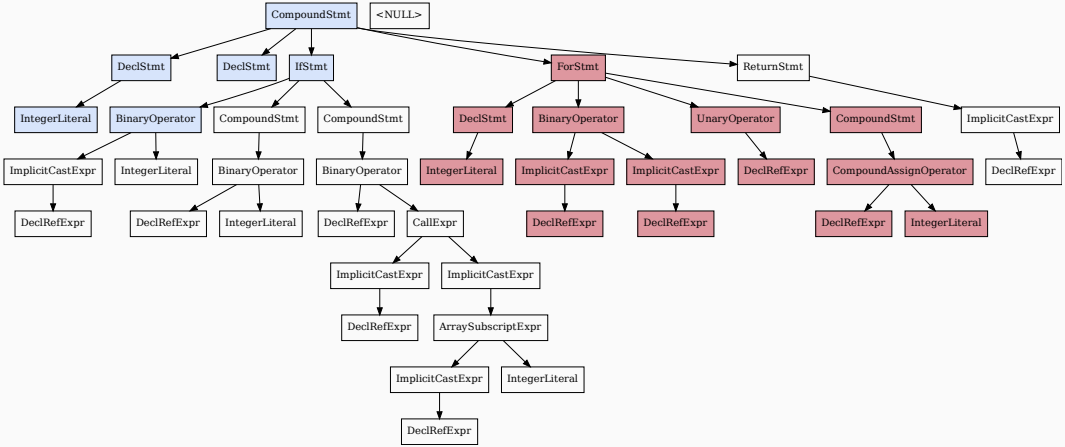


Abbildung 7: AST Visualisierung [Squ23]

Optimizer

```
39  for.cond:   |\label{lst:trivial_branch_loop_realistic_ir:for}|                                    ; preds =
    ↪  %for.inc, %if.end
40    %3 = load i32, i32* %i, align 4
41    %4 = load i32, i32* %iteration_count, align 4
42    %cmp1 = icmp slt i32 %3, %4 |\label{lst:trivial_branch_loop_realistic_ir:binary_op}|
43    br i1 %cmp1, label %for.body, label %for.end
44
45  for.body:                                         ; preds = %for.cond
46    %5 = load i32, i32* %result, align 4
47    %add = add nsw i32 %5, 1
48    store i32 %add, i32* %result, align 4
49    br label %for.inc
50
51  for.inc:   |\label{lst:trivial_branch_loop_realistic_ir:unary_op}|                                ;
    ↪  preds = %for.body
52    %6 = load i32, i32* %i, align 4
53    %inc = add nsw i32 %6, 1
54    store i32 %inc, i32* %i, align 4
55    br label %for.cond, !llvm.loop !4 |\label{lst:trivial_branch_loop_realistic_ir:unary_op_end}|
```

Code Generator

```
19  .LBB0_3:
20          mov     dword ptr [rbp - 28], 0
21  .LBB0_4:
22          mov     eax, dword ptr [rbp - 28]
23          cmp     eax, dword ptr [rbp - 24]
24          jge     .LBB0_7
25          mov     eax, dword ptr [rbp - 20]
26          add     eax, 1
27          mov     dword ptr [rbp - 20], eax
28          mov     eax, dword ptr [rbp - 28]
29          add     eax, 1
30          mov     dword ptr [rbp - 28], eax
31          jmp     .LBB0_4
```

Table of Contents

Beispiel aus der Forschung

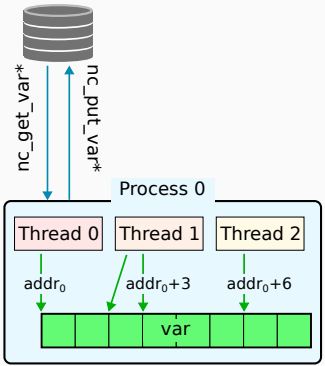


Abbildung 8: Vorher [Squ23]

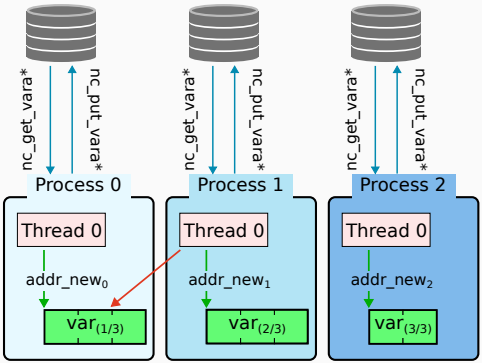


Abbildung 9: Nachher [Squ23]

Douglas Thain,
Introduction to Compilers and Language Design,
2nd ed., Independently Published,2020.
https://dthain.github.io/books/compiler/compilerbook.pdf

Linda Torczon and Keith Cooper,
Engineering a Compiler,
2nd ed., Morgan Kaufmann,2007.
https://github.com/lighthousand/books

Literatur

[Squ23] Jannek Squar. ***Compiler-assisted distribution of OpenMP code for improved scalability.*** PhD thesis, University of Hamburg, Germany, 2023.

[Tha20] Douglas Thain. ***Introduction to Compilers and Language Design.*** Independently Published, 2 edition, 2020.

[Win22] Hannes Winkler. **Clang/llvm overview.** Online, July 2022. Last accessed: 2023-06-10.