

# PicoC-Compiler

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

Dokumentation

---

*Bitte Korrekturen mitteilen über:*

[https://github.com/matthejue/Bachelorarbeit\\_Dokumentation\\_out/issues](https://github.com/matthejue/Bachelorarbeit_Dokumentation_out/issues)

*Aktualisiert am:*

13. Oktober 2022

Universität Freiburg, Lehrstuhl für Betriebssysteme

# Gliederung

Fehlermeldungen

Instant-Mode

Shell-Mode

Show-Mode

Makefile Bedienung

Tests ausführen, verifizieren, konvertieren usw.

Testkategorien

# Fehlermeldungen

# Fehlermeldungen

## Kategorien

Fehlerkategorie	Beschreibung
UnexpectedCharacter	Der <b>Lexer</b> ist auf eine <b>unerwartete Zeichenfolge</b> gestossen, die in der Grammatik des Lexers <b>nicht abgeleitet</b> werden kann.
UnexpectedToken	Der <b>Parser</b> hat ein <b>unerwartetes Token</b> erhalten, das in dem <b>Kontext</b> in dem es sich befand in der Grammatik des Parsers <b>nicht abgeleitet</b> werden kann.
UnexpectedEOF	Der <b>Parser</b> hat in dem <b>Kontext</b> in dem er sich befand bestimmte <b>Tokens erwartet</b> , aber die <b>Eingabe endete</b> abrupt.

*Tabelle 1: Fehlerarten in der Lexikalischen und Syntaktischen Analyse.*

Fehlerkategorie	Beschreibung
DivisionByZero	Wenn bei einer <b>Division</b> durch 0 geteilt wird (z.B. <code>var / 0</code> ).

*Tabelle 2: Fehlerarten, die zur Laufzeit auftreten.*

# Fehlermeldungen

## Kategorien, Teil 2

Fehlerkategorie	Beschreibung
UnknownIdentifier	Es wird ein Zugriff auf einen <b>Bezeichner</b> gemacht (z.B. <code>unknown_var + 1</code> ), der noch <b>nicht deklariert</b> und ist daher <b>nicht</b> in der <b>Symboltabelle</b> aufgefunden werden kann.
UnknownAttribute	Der <b>Verbundstyp</b> (z.B. <code>struct st {int attr1; int attr2;}</code> ) auf dessen Attribut im momentanen Kontext zugegriffen wird (z.B. <code>var[3].unknown_attr</code> ) besitzt das <b>Attribut</b> (z.B. <code>unknown_attr</code> ) auf das zugegriffen werden soll <b>nicht</b> .
ReDeclarationOrDefinition	Ein Bezeichner von z.B. einer <b>Funktion</b> oder <b>Variable</b> , der bereits <b>deklariert</b> oder <b>definiert</b> ist (z.B. <code>int var</code> ) wird <b>erneut</b> deklariert oder definiert (z.B. <code>int var[2]</code> ). Dieser Fehler ist leicht festzustellen, indem geprüft wird ob das <b>Assoziative Feld</b> durch welches die <b>Symboltabelle</b> umgesetzt ist diesen <b>Bezeichner bereits als Schlüssel</b> besitzt.
TooLargeLiteral	Der <b>Wert</b> eines Literals ist <b>größer</b> als $2^{31} - 1$ oder <b>kleiner</b> als $-2^{31}$ .
NoMainFunction	Das Programm besitzt <b>keine</b> oder <b>mehr als eine</b> main-Funktion.

*Tabelle 3: Fehlerarten in den Passes.*

# Fehlermeldungen

## Kategorien, Teil 3

Fehlerkategorie	Beschreibung
ConstAssign	Wenn einer initialisierten <b>Konstante</b> (z.B. <code>const int const_var = 42</code> ) ein <b>Wert zugewiesen</b> wird (z.B. <code>const_var = 41</code> ). Der <b>einzige Weg</b> , wie eine Konstante einen Wert erhält ist bei ihrer <b>Initialisierung</b> .
PrototypeMismatch	Der <b>Prototyp</b> einer <b>deklarierten</b> Funktion (z.B. <code>int fun(int arg1, int arg2[3])</code> ) stimmt nicht mit dem <b>Prototyp</b> der späteren <b>Definition</b> dieser Funktion (z.B. <code>void fun(int arg1[2], int arg2) { }</code> ) überein.
ArgumentMismatch	Wenn die <b>Argumente</b> eines <b>Funktionsaufrufs</b> (z.B. <code>fun(42, 314)</code> ) nicht mit dem <b>Prototyp</b> der Funktion die aufgerufen werden soll (z.B. <code>void fun(int arg[2]) { }</code> ) nach <b>Datentypen</b> oder <b>Anzahl Argumente bzw. Parameter</b> übereinstimmt.
WrongReturnType	Wenn eine Funktion, die ihrem <b>Prototyp</b> zufolge einen <b>Rückgabewert</b> hat, der <b>nicht</b> mit dem dem Datentyp übereinstimmt, der von einer <b>return-Anweisung</b> zurückgegeben wird.

*Tabelle 4: Fehlerarten in den Passes, Teil 2.*

# Fehlermeldungen

## Kategorien, Teil 4

Fehlerkategorie	Beschreibung
DatatypeMismatch	Wenn die <b>Operation</b> und der <b>Datentyp</b> des Attributes oder Elementes auf welches in diesem <b>Kontext</b> zugegriffen wird <b>nicht</b> zueinander <b>passen</b> .

*Tabelle 5: Fehlerarten in den Passes, Teil 3.*

# Instant-Mode



# Bedienung des PicoC-Compilers

## Instant-Mode

- ▶ Kompilieren: `> picoc_compiler <cli-opts> program.picoc .`
- ▶ Interpretieren: `> picoc_compiler <cli-opts> program.reti .`

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Compiler

Kommandozeilen-option	Beschreibung	Standardwert
<code>-i, --intermediate_stages</code>	Gibt <b>Zwischenstufen</b> der Kompilierung in Form der verschiedenen <b>Tokens</b> , <b>Ableitungsbäume</b> , <b>Abstrakten Syntaxbäume</b> der verschiedenen <b>Passes</b> in Dateien mit <b>entsprechenden Dateieindungen</b> aber gleichem <b>Basisnamen</b> aus. Wenn die <code>--run</code> -Option aktiviert ist, wird der <b>Zustand der RETI</b> nach der Ausführung des <b>letzten</b> Befehls in eine Datei ausgegeben. Im <b>Shell-Mode</b> erfolgt <b>keine</b> Ausgabe in Dateien, sondern nur im <b>Terminal</b> .	<b>False</b> , most_used: <b>True</b>
<code>-p, --print</code>	Gibt alle <b>Dateiausgaben</b> auch im <b>Terminal</b> aus. Diese Option ist im <b>Shell-Mode</b> dauerhaft aktiviert.	<b>False</b> , Shell-Mode und most_used: <b>True</b>

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Compiler, Teil 2

Kommandozeilen-option	Beschreibung	Standard-wert
-v, --verbose	Fügt den verschiedenen <b>Zwischenschritten der Kompilierung</b> , unter anderem auch dem finalen RETI-Code <b>Kommentare</b> hinzu. Diese Kommentare beinhalten eine <b>Anweisung</b> oder einen <b>Befehl</b> aus einem <b>vorherigen Pass</b> , der durch die darunterliegenden Anweisungen oder Befehle <b>ersetzt</b> wurde. Wenn die --run und die --immediate_stages-Option aktiviert sind, wird der <b>Zustand</b> der virtuellen RETI-CPU <b>vor</b> und <b>nach jedem Befehl</b> ausgegeben.	False
-vv, --double_verbose	Hat <b>dieselben Effekte</b> , wie die --verbose-Option, aber bewirkt zusätzlich <b>weitere Effekte</b> . <b>PicoC-Knoten</b> erhalten bei der Ausgabe als zusammenhängende <b>Abstrakte Syntaxbäume</b> zusätzliche <b>runde Klammern</b> , sodass direkter abgelesen werden kann, wo ein Knoten anfängt und wo einer aufhört. In <b>Fehlermeldungen</b> werden <b>mehr Tokens</b> angezeigt, die an der Stelle der Fehlermeldung erwartet worden wären. Bei Aktivierung der --intermediate_stages-Option werden in den dadurch ausgegebenen <b>Abstrakten Syntaxbäumen</b> zusätzlich <b>versteckte Attribute</b> angezeigt, die <b>Informationen zu Datentypen</b> und Informationen für <b>Fehlermeldungen</b> beinhalten.	False

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Compiler, Teil 3

Kommandozeilen-option	Beschreibung	Standardwert
-h, --help	Zeigt diese <b>Dokumentation</b> mithilfe des im Betriebssystem <b>eingestellten PDF-Viewers</b> an.	False
-l, --lines	Es lässt sich einstellen, <b>wieviele Zeilen</b> rund um die Stelle an welcher ein Fehler aufgetreten ist angezeigt werden sollen.	2
-c, --color	Aktiviert <b>farbige Ausgabe</b> für <b>Fehlermeldungen</b> , <b>PicoC-</b> und <b>RETI-Code</b> , <b>Tokens</b> , <b>Ableitungsbäume</b> und <b>Abstrakte Syntaxbäume</b> der verschiedenen Passes.	False, most_used: True
-e, --example	<b>Filtert</b> für übersichtliche Codebeispiele bestimmte <b>Kommentare</b> in den Abstrakten Syntaxbäumen heraus. Diese Option wurde für die Codebeispiele in der <b>schriftlichen Ausarbeitung der Bachelorarbeit</b> implementiert.	False

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Compiler, Teil 4

Kommandozeilen-option	Beschreibung	Standardwert
-t, --traceback	Nutzt das Python Package traceback um bei Fehlermeldungen <b>Stacktraces</b> des Compilers auszugeben.	False
-d, --debug	Startet den <b>PuDB-Debugger</b> (pip install pudb) <b>vor</b> Beginn des Kompilierens oder Interpretierens.	False
-s, --supress_errors	Obwohl eine <b>Fehlermeldung</b> ausgegeben werden müsste, wird bei manchen Fehlermeldungen die Ausgabe <b>unterdrückt</b> .	False

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Interpreter

Kommandozeilen-option	Beschreibung	Standardwert
-R, --run	Führt die <b>RETI-Befehle</b> , die das Ergebnis der Kompilierung sind mit einer <b>virtuellen RETI-CPU</b> aus. Wenn die <b>--intermediate_stages</b> -Option aktiviert ist, wird eine Datei <b>&lt;basename&gt;.reti_states</b> erstellt, welche den <b>Zustand der RETI-CPU</b> nach dem <b>letzten</b> ausgeführten RETI-Befehl enthält. Wenn die <b>--verbose-</b> oder <b>--double_verbose</b> -Option aktiviert ist, wird der Zustand der RETI-CPU <b>vor</b> und <b>nach</b> jedem Befehl auch noch zusätzlich in die Datei <b>&lt;basename&gt;.reti_states</b> ausgegeben.	<b>False</b> , Show-Mode und <b>most_used</b> : <b>True</b>
-B, --process_begin	Setzt die <b>Adresse</b> , wo der <b>Prozess</b> bzw. das <b>Codesegment</b> für das ausgeführte Programm <b>beginnt</b> .	<b>3</b>
-D, --datasegment_size	Setzt die Größe des <b>Datensegments</b> . Diese Option muss mit <b>Vorsicht</b> gesetzt werden, denn wenn der Wert zu niedrig gesetzt wird, dann können die <b>Globalen Statischen Daten</b> und der <b>Stack</b> miteinander <b>kollidieren</b> .	<b>32</b>

# Instant-Mode

## Kommandozeilenargumente <cli-opts> für den Interpreter, Teil 2

Kommandozeilen-option	Beschreibung	Standardwert
-S, --show_mode	Startet den <b>Show-Mode</b> . Der <b>Show-Mode</b> zeigt eine Zeichenfolge über <b>mehrere Seiten</b> verteilt an. Standardmäßig wird dies für die <b>Zustände der RETI nach</b> und <b>vor</b> der Ausführung eines bestimmten RETI-Befehls gemacht. Der Eindruck des Debuggens kommt dadurch, dass durch Drücken entsprechender Tasten immer an die <b>richtigen Stellen</b> gesprungen wird, an denen der nächste oder vorherige Zustand <b>anfähgt</b> .	<b>False</b> , compile_show und interpret_show: <b>True</b>
-P, --pages	Setzt auf <b>wieviele Seiten</b> im <b>Show-Mode</b> eine Zeichenfolge <b>verteilt</b> werden soll.	<b>5</b>
-E, --extension	Setzt welcher <b>Dateityp</b> , der durch eine bestimmte <b>Dateiendung</b> spezifiziert ist im <b>Show-Mode</b> angezeigt werden soll.	<b>reti_states</b>

# Shell-Mode



# Bedienung des PicoC-Compilers

## Shell-Mode

- ▶ Starten: `> picoc_compiler`.
- ▶ Kompilieren: `> compile <cli-opts> "<seq-of-stmts>" ( cpl )`.
  - ▶ automatisch in main-Funktion eingefügt: `void main(){<seq-of-stmts>}`.
- ▶ Kompilieren, meist genutzt: `> must_used <cli-opts> "<seq-of-stmts>" ( mu )`.
- ▶ Kompilieren und dann Show-Mode:  
`> compile-show <cli-opts> "<seq-of-stmts>" ( cs )`.
- ▶ Interpretieren und dann Show-Mode:  
`> interpret-show <cli-opts> "<seq-of-instrs>" ( is )`.

# Bedienung des PicoC-Compilers

## Shell-Mode, Teil 2

- ▶ Beenden: `> quit`.
- ▶ Dokumentation: `> help` (`?`).
- ▶ Multiline-Command: weitere Zeile mit `↵` und mit `;` terminieren.
- ▶ Farben toggeln: `> color_toggle` (`ct`).
- ▶ Cursor bewegen: `←`, `→`.
- ▶ Befehlshistorie: `↑`, `↓`.
- ▶ Autovervollständigung: `Tab`.

# Bedienung des PicoC-Compilers

## Shell-Mode, Teil 3

- ▶ Befehlshistorie anzeigen: `> history` .
- ▶ Aktion mit Befehlshistorie ausführen `> history <opt>` .
  - ▶ Befehl erneut ausführen: `-r <cmd-nr>` .
  - ▶ Befehl editieren `-e <cmd-nr>` (Editor durch **Environment Variable** `$EDITOR` bestimmt).
  - ▶ Befehlshistorie leeren: `-c` .
  - ▶ Befehl suchen: `ctrl + r` .

# Show-Mode

# Bedienung des PicoC-Compilers

## Show-Mode

- ▶ Starten: `> picoc_compiler -S <cli-opts> program.(reti|picoc)` .
- ▶ Shell-Mode Befehle: `> cs <cli-opts> "<seq-of-stmts>"` bzw.  
`> is <cli-opts> "<seq-of-instrs>"` .
- ▶ Anzahl Seiten: `-P <num>` .
- ▶ Dateiendung der gewünschten Datei: `-E <extension>` .
- ▶ Spezielle Einstellungen: `/interp_showcase.vim` .
- ▶ Neovim: `:help` , `:Tutor` .

# Bedienung des PicoC-Compilers

## Show-Mode, Teil 2

- ▶ Zustände vor / nach Befehl ansehen: **Tab**, **↑ -Tab**.
- ▶ Beenden: **q**, **Esc**.
- ▶ Fenster minimieren / maximieren: **m**, **M**.
- ▶ Alle Fenster gleich aufteilen: **E**.
- ▶ Kommentare toggeln: **C**.
- ▶ (Relative) Zeilennummern toggeln: **N**, **R**.
- ▶ Zeile farbig markieren: **c - 1**, ..., **c - 9**.
- ▶ Farbig markierte Zeilen verstecken / wieder einblenden: **H**.

# Bedienung des PicoC-Compilers

## Show-Mode, Teil 3

- ▶ Farbig markierte Zeilen entfernen **D**.
- ▶ Weiteres Fenster öffnen: **S**.

# Makefile Bedienung



# Makefile Bedienung

## Show-Mode

- ▶ Starten für bestimmtes Programm:

```
> make show FILEPATH=<path-to-file> <more-options> .
```

- ▶ Starten für bestimmten Test in /tests:

```
> make test-show TESTNAME=<testname> <more-options> .
```

# Makefile Bedienung

## Makefile Optionen <more-options>

Kommandozeilenoption	Beschreibung	Standardwert
FILEPATH	Pfad zur Datei, die im Show-Mode angezeigt werden soll.	∅
TESTNAME	Name des Tests. Alles andere als der Basisname, wie die Dateiendung wird abgeschnitten.	∅
EXTENSION	Dateiendung, die an TESTNAME angehängt werden soll, damit daraus z.B. ./tests/TESTNAME.EXTENSION wird.	reti_states
NUM_WINDOWS	Anzahl Fenster auf die ein Dateiinhalt verteilt werden soll.	5
VERBOSE	Möglichkeit für eine ausführlichere Ausgabe die Kommandozeilenoption -v oder -vv zu aktivieren.	∅ bzw. -v für test-show
DEBUG	Möglichkeit die Kommandozeilenoption -d zu aktivieren, um bei make test-show TESTNAME=<testname> den Debugger für den entsprechenden Test <testname> zu starten.	∅

Tests ausführen, verifizieren, konvertieren usw.

# Tests

## Bedienung

- ▶ Tests in `/tests` verifizieren und ausführen: `> make test <more-options> .`
  - ▶ `/run_tests.sh`, welches **zuerst** `/extract_input_and_expected.sh`, `/convert_to_c.py` und `/verify_tests.sh` ausführt.
- ▶ Tests vom GCC verifizieren lassen: `> make verify TESTNAME=<testname> .`
  - ▶ **vorher** `/extract_input_and_expected.sh`, `/convert_to_c.py` ausgeführt.
  - ▶ `/verify_tests.sh`.

# Tests

## Bedienung, Teil 2

### ► Informationen aus Tests extrahieren:

```
> make extract TESTNAME=<testname> .
```

► **Eingabe** // in:<space-sep-values> in <program>.in, **Ausgabe** // expected:<space-sep-values> in <program>.out\_expected, **Datensegmentgröße** // datasegment:<size> **optional** in <program>.datasegment\_size.

► `/extract_input_and_expected.sh` .

# Tests

## Bedienung, Teil 3

- ▶ Testdatei erstellen, die vom GCC kompiliert werden kann:

```
> make convert TESTNAME=<testname>.
```

- ▶ `input()`s werden durch **Eingaben** in `<program>.in` ersetzt.
- ▶ `print(exp)`s werden durch `#include<stdio.h>` und `printf("%d", exp)` ersetzt.
- ▶ `/convert_to_c.py`.

# Testkategorien

# Tests

## Testkategorien

Testkategorie	Beschreibung	Anzahl
basic	Grundlegende Funktionalitäten des PicoC-Compilers.	23
advanced	Spezialfälle und Kombinationen verschiedener Funktionalitäten des PicoC-Compilers.	21
hard	Lange und komplexe Tests, für welche die Funktionalitäten des PicoC-Compilers in perfekter Harmonie miteinander funktionieren müssen.	8
example	Bekannte Algorithmen, die als gutes, repräsentatives Beispiel für die Funktionsfähigkeit des PicoC-Compilers dienen.	24
error	Fehlermeldungen testen. Keine Verifikation wird ausgeführt.	69
exclude	Aufgrund vielfältiger Gründe soll keine Verifikation ausgeführt werden.	7
thesis	Codebeispiele der schriftlichen Ausarbeitung der Bachelorarbeit, die etwas umgeschrieben wurden, damit nicht nur das Durchlaufen dieser Tests getestet wird.	28
tobias	Vom Betreuer dieser Bachelorarbeit, M.Sc. Tobias Seufert geschrieben.	1