

ALBERT LUDWIGS UNIVERSITÄT FREIBURG

TECHNISCHE FAKULTÄT

# PicoC-Compiler

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

BACHELORARBEIT

---

*Abgabedatum:* 28<sup>th</sup> 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
0.1 Fehlermeldungen . . . . .	1
Literatur	A

# Abbildungsverzeichnis

# Codeverzeichnis

0.1	Beispiel für typische Fehlermeldung mit 'found' und 'expected' . . . . .	2
0.2	Beispiel Fehlermeldung langgestrichte fehlermeldung . . . . .	2
0.3	Beispiel für Fehlermeldung mit mehreren erwarteten Tokens . . . . .	3
0.4	Beispiel für Fehlermeldung ohne expected . . . . .	3

# Tabellenverzeichnis

1	Syntaktische Fehlermeldungen . . . . .	1
2	Semantische Fehlermeldungen . . . . .	1
3	Laufzeit Fehlermeldungen . . . . .	2

# Definitionsverzeichnis

# Grammatikverzeichnis



## 0.1 Fehlermeldungen

Die **Fehlermeldungen**, die der **PicoC-Compiler** ausgeben kann sind in den Tabellen 1, 2 und 3 und eingeteilt nach den Kategorien **Syntax**, **Semantik** und **Laufzeit** aus Unterkapitel 0.1.

Fehlertyp	Beschreibung
UnexpectedCharacter	Der <b>Lexer</b> ist auf eine <b>unerwartete Zeichenfolge</b> gestossen, die von keinem <b>Pattern</b> erkannt wird.
UnexpectedToken	Der <b>Parser</b> hat ein <b>unerwartetes Token</b> erhalten, dass in dem <b>Kontext</b> in dem er sich befand <b>nicht vorkommen</b> konnte.
UnexpectedEOF	Der <b>Parser</b> hat in dem <b>Kontext</b> in dem er sich befand ein <b>Token erwartet</b> , aber die <b>Eingabe endete</b> abrupt.

Tabelle 1: Syntaktische Fehlermeldungen

Fehlertyp	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>Structtyp</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> .
ReDeclarationDefinition	Ein Bezeichner <sup>a</sup> 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.
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> .
TooLargeLiteral	Der <b>Wert</b> eines Literals ist <b>größer</b> als $2^{31} - 1$ oder <b>kleiner</b> als $-2^{31}$ .
NotExactlyOneMainFunction	Das Programm besitzt <b>keine</b> oder <b>mehr als eine</b> <b>main-Funktion</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.
MissingReturn	Wenn eine Funktion, die ihrem <b>Prototyp</b> zufolge einen <b>Rückgabewert</b> hat, der nicht vom <b>Datentyp</b> <code>void</code> ist (z.B. <code>int fun() { ... }</code> ) als <b>letztes Statement</b> kein <b>return-Statement</b> hat, dass einen Wert des <b>entsprechenden Datentyps</b> zurückgibt <sup>b</sup> .

<sup>a</sup> Z.B. von einer **Funktion** oder **Variable**.

<sup>b</sup> Der **entsprechende Datentyp** müsste auf das Beispiel von davor `void fun(int arg[2]) { ... }` bezogen z.B. `return 42` sein.

Tabelle 2: Semantische Fehlermeldungen

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

Tabelle 3: Laufzeit Fehlermeldungen

Eine **Fehlermeldung** fängt immer mit einem **Header** an, bei dem sich an den Fehlermeldungen des **GCC** orientiert wurde. Nacheinander stehen im **Header** der **Dateiname**, die **Position** des Fehlers in der Datei in der das fehlerhafte Programm steht, die **Fehlerart** und ein **Beschreibungstext**. In Code 0.1 ist eine **typische Fehlermeldung** zu sehen.

Unter dem **Header** wird ein kleiner **Ausschnitt des Programmes** um die Stelle herum an welcher der **Fehler aufgetreten** ist angezeigt. Die Kommandozeilenoptionen `-l` und `-c`, welche in Tabelle ?? erläutert werden könnten in diesem Zusammenhang interessant sein.

Das Symbol `~` bzw. eine Folge von `~` kennzeichnet das **Token**, welches an der Stelle des Fehlers **vorgefunden** wurde und das Symbol `^` soll einen **Pfeil** symbolisieren, der auf eine **Position zeigt** an der ein **anderes Token**, ein **anderer Datentyp** usw. **erwartet** worden wäre und **in der Zeile darunter** eine **Beschriftung** an sich hängen hat, die konkret angibt, was dort eigentlich **erwartet** worden wäre.

```
./tests/error_wrong_written_keyword.picoc:7:4: UnexpectedToken: Expected e.g. 'while', found
↳ 'wile'.
  do {

    } wile (True);
    ^~~~~
    'while'
  }
```

Code 0.1: Beispiel für typische Fehlermeldung mit 'found' und 'expected'

Bei **Fehlermeldungen**, deren Auftreten seine **Ursache** an einer **anderen Stelle im Code** hat, wird einmal ein **Header** mit **Programmausschnitt** für die **Stelle** an welcher der **Fehler aufgetreten** ist erstellt und ein weiterer **Header** mit **Programmausschnitt** für die Stelle welche die **Ursache** für das Auftreten dieses Fehlers ist.

```
./tests/error_redefinition.picoc:6:6: Redefinition: Redefinition of 'var'.
void main() {
  int var = 42;
  int var = 41;
  ~~~
}
./tests/error_redefinition.picoc:5:6: Note: Already defined here:

void main() {
  int var = 42;
  ~~~
  int var = 41;
}
```

Code 0.2: Beispiel Fehlermeldung langgestreckte fehlermeldung

```
./tests/error_unexpected_eof.picoc:4:13: UnexpectedEOF: Unexpected end-of-file, expected e.g.  
↳ reti comment or '}' or '(' or 'print' or '*' or 'if'.  
  
void main() {  
    ^  
    reti comment or '}' or '(' or 'print' or '*' or 'if'
```

**Code 0.3:** *Beispiel für Fehlermeldung mit mehreren erwarteten Tokens*

```
./tests/error_unknown_identifier_ref.picoc:5:18: UnknownIdentifier: Identifier  
↳ 'unknown_identifier' wasn't declared yet.  
  
void main() {  
    int var = 42 + unknown_identifier;  
                ~~~~~  
}  
}
```

**Code 0.4:** *Beispiel für Fehlermeldung ohne expected*

# Literatur