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

Betreung:
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

Codeverzeichnis

Tabellenverzeichnis

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 **Lexer** ist auf eine **unerwartete Zeichenfolge** gestossen, die von keinem **Pattern** erkannt wird.
`UnexpectedToken`	Der **Parser** hat ein **unerwartetes Token** erhalten, dass in dem **Kontext** in dem er sich befand **nicht vorkommen** konnte.
`UnexpectedEOF`	Der **Parser** hat in dem **Kontext** in dem er sich befand bestimmte **Token erwartet**, aber die **Eingabe endete** abrupt.

Tabelle 1: *Syntaktische Fehlermeldungen*

Fehlertyp	Beschreibung
`UnknownIdentifier`	Es wird ein Zugriff auf einen **Bezeichner** gemacht (z.B. `unknown_var + 1`), der noch **nicht deklariert** und ist daher **nicht** in der **Symboltabelle** aufgefunden werden kann.
`UnknownAttribute`	Der **Structtyp** (z.B. `struct st {int attr1; int attr2;}`) auf dessen Attribut im momentanen Kontext zugegriffen wird (z.B. `var[3].unknown_attr`) besitzt das **Attribut** (z.B. `unknown_attr`) auf das zugegriffen werden soll **nicht**.
`ReDeclarationDefinition`	Ein Bezeichner[a] der bereits **deklariert** oder **definiert** ist (z.B. `int var`) wird **erneut** deklariert oder definiert (z.B. `int var[2]`). Dieser Fehler ist leicht festzustellen, indem geprüft wird ob das **Assoziative Feld** durch welches die **Symboltabelle** umgesetzt ist diesen **Bezeichner bereits als Schlüssel** besitzt.
`ConstAssign`	Wenn einer intialisierten **Konstante** (z.B. `const int const_var = 42`) ein **Wert zugewiesen** wird (z.B. `const_var = 41`). Der **einzige Weg**, wie eine Konstante einen Wert erhält ist bei ihrere **Initialisierung**.
`TooLargeLiteral`	Der **Wert** eines Literals ist **größer** als $2^{31} - 1$ oder **kleiner** als -2^{31}.
`NotExactlyOneMainFunction`	Das Programm besitzt **keine** oder **mehr als eine** `main`-Funktion.
`PrototypeMismatch`	Der **Prototyp** einer **deklarierten** Funktion (z.B. `int fun(int arg1, int arg2[3])`) stimmt nicht mit dem **Prototyp** der späteren **Definition** dieser Funktion (z.B. `void fun(int arg1[2], int arg2) { }`) überein.
`ArgumentMismatch`	Wenn die **Argumente** eines **Funktionsaufrufs** (z.B. `fun(42, 314)`) nicht mit dem **Prototyp** der Funktion die aufgerufen werden soll (z.B. `void fun(int arg[2]) { }`) nach **Datentypen** oder **Anzahl Argumente bzw. Parameter** übereinstimmt.
`MissingReturn`	Wenn eine Funktion, die ihrem **Prototyp** zufolge einen **Rückgabewert** hat, der nicht vom **Datentyp** `void` ist (z.B. `int fun() {...}`) als **letztes Statement** kein `return`-Statement hat, dass einen Wert des **entsprechenden Datentyps** zurückgibt[b].

[a] Z.B. von einer **Funktion** oder **Variable**.
[b] 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 **Division** durch 0 geteilt wird (z.B. `var / 0`).

Tabelle 3: *Laufzeit Fehlermeldungen*

In Code 0.1 ist eine **typische Fehlermeldung** zu sehen. 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**.

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 konkrett angibt, was dort eingentlich **erwartet** worden wäre.

```
1./tests/error_wrong_written_keyword.picoc:7:4: UnexpectedToken: Expected e.g. 'while', found
  ↪  'wile'.
2  do {
3
4  } wile (True);
5    ^~~~~
6    'while'
7}
```

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

Bei **Fehlermeldungen**, wie in Code 0.2, die ihre **Ursache** an einer **anderen Stelle im Code** haben, wird einmal ein **Header** mit **Programmauschnitt** für die **Stelle** an welcher der **Fehler aufgetreten** ist erstellt und ein weiterer **Header** mit **Programmauschnitt** für die **Stelle** welche die **Ursache** für das Auftreten dieses Fehlers ist.

```
1./tests/error_redefinition.picoc:6:6: Redefinition: Redefinition of 'var'.
2void main() {
3  int var = 42;
4  int var = 41;
5      ~~~
6}
7./tests/error_redefinition.picoc:5:6: Note: Already defined here:
8
9void main() {
10  int var = 42;
11      ~~~
12  int var = 41;
13}
```

Code 0.2: *Beispiel Fehlermeldung langgestrechte fehlermeldung*

Bei manchen Fehlermeldungen, wie in Code 0.3 ist es **garnicht möglich** mit ~ ein Token an der **Stelle** zu markieren, an welcher der **Fehler vorgefunden** wurde, da z.B. beim `UnexpectedEOF`-Fehler das **Ende der Programmes** erreicht wurde, wo es **kein sichtbares Token** gibt, welches man markieren könnte. Des Weiteren ist in Code 0.3 interessant, dass mehrere Tokens

```
1./tests/error_unexpected_eof.picoc:4:13: UnexpectedEOF: Unexpected end-of-file, expected e.g.
  ↪ reti comment or '}' or '(' or 'print' or '*' or 'if'.
2
3void main() {
4             ^
5             reti comment or '}' or '(' or 'print' or '*' or 'if'
```

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

```
1./tests/error_unknown_identifier_ref.picoc:5:18: UnknownIdentifier: Identifier
  ↪ 'unknown_identifier' wasn't declared yet.
2
3void main() {
4  int var = 42 + unknown_identifier;
5                 ~~~~~~~~~~~~~~~~~~
6}
```

Code 0.4: *Beispiel für Fehlermeldung ohne expected*

Literatur