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

Aktualisiert am:
7. 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

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

Tabelle 1: Fehlerarten in der Lexikalischen und Syntaktischen Analyse.

`DivisionByZero`	Wenn bei einer Division durch 0 geteilt wird (z.B. `var / 0`).

Tabelle 2: Fehlerarten, die zur Laufzeit auftreten.

Fehlermeldungen

Kategorien, Teil 2

`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 Verbundstyp (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.
`ReDeclarationOrDefinition`	Ein Bezeichner von z.B. einer Funktion oder Variable, 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.
`TooLargeLiteral`	Der Wert eines Literals ist größer als $2^{31} - 1$ oder kleiner als -2^{31}.
`NoMainFunction`	Das Programm besitzt keine oder mehr als eine `main`-Funktion.

Tabelle 3: Fehlerarten in den Passes.

Fehlermeldungen

Kategorien, Teil 3

`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.
`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.
`WrongReturnType`	Wenn eine Funktion, die ihrem Prototyp zufolge einen Rückgabewert hat, der nicht mit dem dem Datentyp übereinstimmt, der von einer `return`-Anweisung zurückgegeben wird.

Tabelle 4: Fehlerarten in den Passes, Teil 2.

Fehle

Kategc

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

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

Ko

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

Instant-Mode

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

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

Instant-Mode

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

Kom

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

nt-Mode

ndozeilenargumente <cli-opts> für den Compiler, Teil 4

Komma
o

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

Instant-…e

Kommandozeilenargumente <cli-opts> für den Interpreter

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

nt-Mode

ndozeilenargumente <cli-opts> für den Interpreter, Teil 2

Komma
op

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

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>`

Ko

`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

`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