Dokumentation

Gliederung

Fehlermeldungen

Kategorien

Fehlerkategorie	Beschreibung
`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.

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

Tabelle 2: Fehlerarten, die zur Laufzeit auftreten.

Fehlermeldungen

Kategorien

Fehlerkategorie	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 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

Fehlerkategorie	Beschreibung
`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.

Bedienung des PicoC-Compilers

Kommandozeilenargumente `<cli-options>`

Kommandozeilen-option	Beschreibung	Standard-wert
`-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. 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` (`true` im Shell-Mode und für den most_used-Befehl)

Bedienung des PicoC-Compilers

Kommandozeilenargumente `<cli-options>`, Teil 2

Kommandozeilen-option	Beschreibung	Standard-wert
`-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`-Option aktivert ist, wird der Zustand der virtuellen RETI-CPU vor und nach jedem Befehl angezeigt.	`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`

Bedienung des PicoC-Compilers

Kommandozeilenargumente `<cli-options>`, Teil 3

Kommandozeilen-option	Beschreibung	Standard-wert
`-h, --help`	Zeigt die Dokumentation, welche ebenfalls unter Link gefunden werden kann im Terminal an. Mit der `--color`-Option kann die Dokumentation mit farblicher Hervorhebung im Terminal angezeigt werden.	`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.	`false`, most_used: `true`
`-t, --thesis`	Filtert für die Codebeispiele in der schriftlichen Ausarbeitung der Bachelorarbeit bestimmte Kommentare in den Abstrakten Syntaxbäumen heraus, damit alles übersichtlich bleibt.	`false`

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options>, Teil 4

Kommandozeilen-option	Beschreibung	Standard-wert
`-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`, most_used: `true`
`-B`, `--process-begin`	Setzt die relative 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

Bedienung des PicoC-Compilers

Shell-Mode

- Starten: `> picoc_compiler`.
- Kompilieren: `> compile <cli-options> "<seq-of-stmts>"` (`cpl`).
  - automatisch in `main`-Funktion eingefügt: `void main(){<seq-of-stmts>}`.
- Kompilieren, meist genutzt:
  `> must_used <cli-options> "<seq-of-stmts>"` (`mu`).
- Beenden: `> quit`.
- Dokumentation: `> help` (`?`).

Bedienung des PicoC-Compilers

Shell-Mode

- Multiline-Command: weitere Zeile mit ↩ und mit `;` terminieren.
- Farben toggeln: `> color_toggle` (`ct`).
- Cursor bewegen: ←, →.
- Befehlshistorie: ↑, ↓.
- Autovervollständigung: `Tab`.

Bedienung des PicoC-Compilers

Shell-Mode

- 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`.

Bedienung des PicoC-Compilers

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>`.
- Zustände vor / nach Befehl ansehen: `Tab`, `⇑ −Tab`.
- Beenden: `q`, `Esc`.
- Spezielle Einstellungen: `/interp_showcase.vim`.
- Neovim: `:help`, `:Tutor`.

Bedienung des PicoC-Compilers

Show-Mode

- Fenster minimieren / maximieren: `m`, `M`.
- Alle Fenster gleich aufteilen: `E`.
- Kommentare toggeln: `C`.
- (Relative) Zeilennummern toggeln: `N`, `R`.
- Zeile farbig markieren: `1`, ..., `9`.
- Farbig markierte Zeilen verstecken / wieder einblenden: `H`.
- Farbig markierte Zeilen entfernen `D`.
- Weiteres Fenster öffnen: `S`.

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

- 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

- 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`.

Tests

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

Testkategorien

Testkategorie	Beschreibung	Anzahl
`basic`	Grundlegende Funktionalitäten des PicoC-Compilers.	23
`advanced`	Spezialfälle und Kombinationen verschiedener Funktionalitäten des PicoC-Compilers.	20
`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