ComPyler

Niklas Dillenberger

Ziel des Projekts

- Funktionierender Lexer für alle Tokens der Sprache ✅
- Funktionierender Parser für alle Lexer Tokens ✅
- Definition von Funktionen ✅
- Definition von Variablen ✅
- Definition von If Bedingung ✅
- Auswertung von Mathematischen Operationen (✅)
- Auswertung von Operationen auf Variablen (✅)
- Ausgeben von Werten auf der Konsole ✅
- Errors werden erkannt und dem User ausgegeben (✅)

Was kann der Compiler

Statische Typ Analyse in einer nicht statisch getypten Sprache

Berücksichtigung von Scopes

Operationen auf definierten Datentypen

Error Codes mit Zeile und Spalte des Fehlers

Variablen

If Abfragen

print

Vorgehen der Pipeline:

script.dillen -> Lexer -> Parser -> AST -> Python Code -> Python Byte Code

Überblick über das Framework

PLY:

- Gerüst für Lexer und Parser
- Basiert auf yacc
- Eignet sich gut für single-pass Compiler
- Gut Dokumentiert
    - aber einige stellen, die evtl. veraltet sind…

Übersicht über die Grammatik

Tokens werden definiert und nach bedarf noch bearbeitet:

```
6   tokens = (
7       #Newline
8       'newline',
9       # Keywords
10      'VAR',
11      'FUNC',
12      'RETURN',
13      'IF',
14      # Identifiers
15      'IDENTIFIER',
16      'TYPE',
17      # Symbols
18      'COLON',
```

```
Niklas Dillenberger
def t_FLOAT(t):
    r'-?\d+\.\d+'
    return t

Niklas Dillenberger
def t_INT(t):
    r'-?\d+'
    return t

Niklas Dillenberger *
def t_STR(t):
    r'"[^"]*"'
    return t

Niklas Dillenberger
def t_BOOL(t):
    r'True|False'
    return t
```

Die Tokens werden per Regex gelesen und returned.

Übersicht über den Parser

- Bottom up Auswertung
- Regeln werden durch Funktionen dargestellt
- Am Ende wird ein AST vom Parser ausgegeben
- Ähnlich wie ANTLR

Syntax der Sprache

- Variablen:
  - var [identifier]:[type] = [value];
- Funktionen:
  - func [identifier]:[return type] ([[identifier]:[type], [identifier]:[type], ...]) { fn_block};
- If-Blöcke:
  - if (boolean expression) { fn_block};
- Unterstützte Typen:
  - int, float, bool, list, str, (dict)

Limitierungen

- Alle Parser Regeln geben IMMER einen String zurück
  - macht Typisierung umständlich…
- Python Code richtet sich nach der Einrückung
  - Parsen von Code sehr unintuitiv und schwer
- “Typisierung” findet nur auf Compiler ebene statt
  - Statische Typen existieren nicht im Byte Code
- Line Tracking aufgrund des Bottom Up ansatzes sehr schwer…

Beispiele

```
var x:int = 4;
var x:str = "sdfr";
print(x);

func test: str (c:int, v:str) {
    var k:float = 3.2;
    if (1 == 2) {
        var d:int = 3;
    };
};
```

```
x = 4
print(str(x))

def test(c: int, v: str) -> str:
    k = 3.2
    if False:
        d = 3
```

Nicht erreichte Ziele

Variablen neu zuweisen

Variablen nicht immutable (können aber eh nicht neu zugewiesen werden 😎)

Loops

Enums

Viele Bugs aufgrund der Indentation…

Danke fürs Zuhören