Interpreter własnego języka: sgm-lang

Szymon Borowy
Marcin Kozak
Grzegorz Nieużyła

Zakres funkcjonalności języka

1. Wspierane typy zmiennych: `int`, `float`, `string`, `bool` → `mrINTernational,boatWhichFloat,stringiBoi,bool`
2. Komentarze → `#`
3. Parsowanie wyrażeń matematycznych składających się z `+`, `-`, `*`, `/`, `%` oraz nawiasów `()`
4. Relacje `==`, `<`, `>`, `<=`, `>=`
5. Instrukcje warunkowe `IF`
6. Pętla `WHILE → youSpinMeRound()`
7. Wypisywanie → `showMeYourGoods()`

Tokenizer – opis działania

Tokenizer/Skaner/Lekser - wyodrębnia podstawowe symbole języka (tokeny). Usuwa białe znaki i komentarze.

Wejście: kod źródłowy (plik)
Wyjście: lista tokenów

Schemat działania:

1. Usuń komentarze
2. Wstaw odstępy i podziel na słowa (rozdziel: `2+3` → `2 + 3`, zostaw: `==`),
3. Dla każdego słowa:
    a. Sprawdź czy jest słowem kluczowym
    b. Sprawdź czy jest wartością typu danych `(2, true, 4.34, "ala ma kota")`
    c. Sprawdź czy jest poprawnym identyfikatorem
    d. Jeśli żadne z powyższych - wyrzuć wyjątek

Parser

Parser - tworzy odpowiednie drzewo składniowe na podstawie listy tokenów.

Wejście: lista tokenów
Wyjście: struktura drzewiasta

Schemat działania:

Tworzy Abstract Syntax Tree przechodząc kolejno po liście tokenów.

Przykładowe wyjście:

[CompoundToken.DATA_TYPE -> DataType.INT name: a == (ASSIGN, None) == [0 <> (SUB, None) <> 5]],[(PRINT, None) print start],[while (WHILE, None) exp([None -> None name: a <> (LESS, None) <> 10]) <[None -> None name: a == (ASSIGN, None) == [None -> None name: a <> (ADD, None) <> 1]],[(PRINT, None) print None -> None name: a]>],[(PRINT, None) print end]

Generator

Generator - dokonuje przekładu kodu z AST na ByteCode

Wejście: struktura drzewiasta (AST)

Wyjście: lista operacji

Schemat działania:

Przechodzenie po węzłach drzewa i przekładanie ich na kolejne operacje .

Bytecode

Bytecode - reprezentuje liniowy ciąg instrukcji zrozumiałych dla interpretera

Instrukcja:

- **Opcode -** kod operacji (m.in. LOAD, STORE, PRINT, JMP, JMP_IF, ADD, SUB, MUL, PUSH, POP)
- **Parametr** (opcjonalnie)
  - **Typ parametru** - stała lub offset
  - **Wartość**

Operacje matematyczne/logiczne pobierają parametry ze stosu i wynik odkładają na wierzch stosu

Program - lista instrukcji

Maszyna wirtualna i interpreter

Maszyna wirtualna - symulacja procesora wykonującego bytecode

Składa się z:

- Stosu
- Listy zmiennych
- Wskaźnika instrukcji (IP)
- Pamięci ROM z kodem programu

Interpreter - wykonuje program krok po kroku zmieniając stan maszyny wirtualnej

Możliwości rozbudowy

- Możliwość definiowania funkcji
- Dodanie kolekcji (listy, zbiory, słowniki)
- Pobieranie danych od użytkownika

Podział pracy

- Marcin → Tokenizer
- Szymon → Parser
- Grzegorz → Interpreter Bytecodu

Prezentacja działania