Jakub Bryl - 293085
TKOM - 20L

Opis projektu

Projekt ma na celu wykonanie interpretera prostego języka który składnią ma przypominać język Scala. Język ten ma być wyposażony w zmienne z zasięgiem,podstawową konstrukcje sterującą (instrukcja warunkowa) oraz możliwość definiowania funkcji i klas. Powinny być również obsługiwane wyrażenia matematyczne i logiczne uwzględniające priorytet operatorów.

Wymagania funkcjonalne

1. Odczytywanie, parsowanie i analiza skryptów zapisanych w plikach tekstowych
2. Kontrola poprawności wprowadzonych danych oraz zgłaszanie błędów wykrytych na każdym z etapów analizy.
3. Poprawne wykonywanie poprawnie zapisanych instrukcji w pliku skryptowym
4. Możliwość definiowania własnych zmiennych, funkcji.
5. Wykonywanie wyrażeń logicznych uwzględniając priorytet operatorów ((), &&, ||, ==, !=)
6. Wykonywanie wyrażeń matematycznych uwzględniając priorytet operatorów ((), +, -, /, *)
7. Możliwość używania instrukcji warunkowych
8. Silna typizacja (<- Tego nie ma. Typy są dynamiczne, mimo to użytkownik przy pisaniu skryptu musi zapisywać jakiego typu oczekuje.)
9. Możliwość definicji własnej klasy - definiowanie nazwy, listy parametrów oraz metod do niej należących.
10. Możliwość tworzenia instancji obiektów na podstawie zdefiniowanych klas.

Wymagania niefunkcjonalne

1. Po uruchomieniu aplikacji z niepoprawnymi parametrami powinno nastąpić poinformowanie użytkownika o możliwych prawidłowych parametrach startowych
2. Komunikaty o błędach powinny wskazywać gdzie błąd wystąpił oraz co jest jego przyczyną
3. Prezentowanie wyników wykonania skryptów w przejrzysty sposób

Środowisko

Projekt będzie zaimplementowany w języku Scala, a do testów jednostkowych wykorzystana zostanie biblioteka ScalaTest.

Obsługa programu

Program będzie aplikacją konsolową uruchamianą wraz z parametrem reprezentującym ścieżkę do pliku ze skryptem do interpretacji. Wynik działania programu oraz ewentualne informacje o błędach będą wyświetlane na standardowym wyjściu.

Sposób uruchomienia:

<w repozytorium jest Dockerfile więc można uruchamiać w kontenerze bez instalacji sbt>

> sbt

[sbt] > run [path_to_script]

Sposób realizacji

Program będzie złożony z modułów odpowiedzialnych za kolejne fazy analizy plików wejściowych oraz moduł wykonujący wygenerowane na podstawie analizy instrukcje.

Moduły główne:

1. FileHandler- moduł zapewniający operacje odczytywania pliku.
2. Lexer - odpowiedzialny za utworzenie tokenów na podstawie pliku wejściowego. Będzie pobierał od FileHandlera kolejne znaki aż do odczytania całej sekwencji odpowiadającej jednemu z akceptowanych tokenów języka(np. “else”).
3. Parser - moduł sprawdzający czy kolejno otrzymane od Lexera tokeny są ułożone zgodnie z gramatyką języka i na ich podstawie tworzy drzewo składniowe.
4. Interpreter - ma za zadanie wykonywanie instrukcji zawartych w drzewie uzyskanym na poprzednich etapach analizy. Wykorzystuje wzorzec projektowy Wizytatora.
5. Na tym poziomie mogą pojawić się błędy spowodowane odwoływaniem się do nieistniejących wartości lub związane z wartościami zmiennych(np. dzielenie przez zero) lub redefinicją zmiennych/klas/funkcji.

Przewidywane moduły pomocnicze

1. Moduł obsługi błędów - moduł w sposób czytelny prezentujący użytkownikowi błędy otrzymane w modułach głównych

W interpreterze zaimplementowałem tylko jedną funkcję wbudowaną:

1. print(x) - wypisuje x + ‘\n’

Przykład skryptu:

```
class List(head: Int, tail: List) {
  def get(x: Int): Int = {
    if (x <= 0)
      return this.head
    if (this.tail != nil)
      return this.tail.get(x-1)
    else
      return nil
    }

  def add(x: Int): List =
    return List(x, this)

  def length(): Int = {
    if (this.tail == nil) return 1
    else return 1 + this.tail.length()
  }
}
```

```
var lst: List = List(4, List(3,List(4,List(5,nil))))
def haha(): Unit =
    print(“haha”)

print(lst.add(4))
haha()
```