Compilers 2021/2022

Projekt 2

Anna Kelm
110455

12 września 2022

1 Cel

Celem projektu jest stworzenie programu obliczającego wyrażenie arytmetyczne (kalkulator). Założenia:

- liczby są typu `int`,
- liczbę może poprzedzać znak `-`,
- parser ignoruje białe znaki między liczbami i operatorami oraz na początku i końcu wyrażenia,
- dostępne operatory to `+, -, *, /, jednoargumentowy -`,
- hierarchia operatorów to:
  1. `jednoargumentowy -`,
  2. `*, /`,
  3. `+, -`.

2 Gramatyka

Symbol E oznacza docelowe wyrażenie arytmetyczne, a symbol num - liczbę naturalną (ciąg cyfr).

$$E \rightarrow TE'$$

$$E' \rightarrow +TE'|-TE'|\varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT'|/FT'|\varepsilon$$

$$F \rightarrow G|-G$$

$$G \rightarrow num|(E)$$

Symbole terminalne to $+ - / * ()$ oraz num, gdzie num odpowiada wyrażeniu regularnemu '\d+'.

3 Różnica względem typowych parserów arytmetycznych

- możliwe jest wystąpienie operatora `jednoargumentowy -` między operatorem dwuargumentowym i liczbą bez oddzielenia nawiasem,
- wielokrotna konkatenacja znaków `+,-`, np. "+++", "+-" nie jest poprawna (oprócz "+-" i "–" przed liczbą),
- następujące złożenia znaków nie są poprawne: "**" (potęgowanie) i "//" (dzielenie całkowitoliczbowe).

4 Wybór typu implementowanego parsera

Gramatyka jest spójna i nie zawiera lewostronnej rekurencji, zatem zdecydowałam się na parser LL(1) sterowany tablicą.

	FIRST	*FOLLOW*
E	$-, num, ($	$\$,)$
E'	$+, -, \varepsilon$	$\$,)$
T	$-, num, ($	$\$, +, -)$
T'	$*, /, \varepsilon$	$\$, +, -,)$
F	$-, num, ($	$\$, +, -, *, /$
G	$num, ($	$\$, +, -, *, /$

Tabela 1: Zbiory *FIRST* i *FOLLOW*.

	+	-	*	/	(	)	num	$
E		T, E'			T, E'		T, E'	
E'	+, E', T	T, -, E'				ε		ε
T		T', F			T', F		T', F	
T'	ε	ε	T', F, *	/, T', F		ε		ε
F		G, -			G		G	
G					E,), (		num	

Tabela 2: Tablica parsingu.

5 Parsowanie i ewaluacja wyrażenia arymetycznego – algorytm

1. Na wejściu programu wczytywany jest tekst - wyrażenie arymetyczne.
2. Skanowanie i tokenizacja tekstu – wyjściem jest lista tokenów, odpowiadających symbolom terminalnym. Funkcja odpowiadająca za tokenizację tekstu skanuje tekst – znak po znaku – traktując nieprzerwany ciąg cyfr jako liczbę. Białe znaki są pomijane. Tokeny odpowiadają symbolom terminalnym z gramatyki.
3. Parsowanie i budowanie drzewa wyprowadzenia z użyciem tablicy parsingu.
   - Do przechowywania kolejno rozpoznanych symboli używany jest stos. Na początku stos zawiera symbol *E*.
   - Pojedynczy token z listy jest konsumowane, a symbol na wierzchu stosu jest zastępowany przez symbole z tabeli parsingu na przecięciu wiersza symbolem wierzchu stosu i kolumny z tokenem. Jeśli symbol jest terminalny i odpowiada tokenowi, jest on po prostu usuwany ze stosu.
   - Wraz z każdym symbolem umieszczonym na stosie, rozbudowywane jest drzewo wyprowadzenia.
   - Jeśli dla pary (symbol ze stosu, token) nie ma reguły w tabeli parsingu lub symbol terminalny z wierzchu stosu jest różny od tokenu, wtedy lista tokenów nie reprezentuje wyrażenia arymetycznego generowanego przez podaną gramatykę.

Jeśli parsowanie zakończy się sukcesem, ma miejsce:

(a) Rekurencyjne przekształcenie drzewa do listy pozostawiając jedynie symbole terminalne i redukując zbędne poziomy zagnieżdżenia.
(b) Rekurencyjna transformacja listy z wyrażeniem do postaci prefiksowej.
(c) Rekurencyjna ewaluacja wyrażenia.

4. Wypisane wyniku lub komunikatu o niezgodności wyrażenia z gramatyką.

6 Struktura projektu

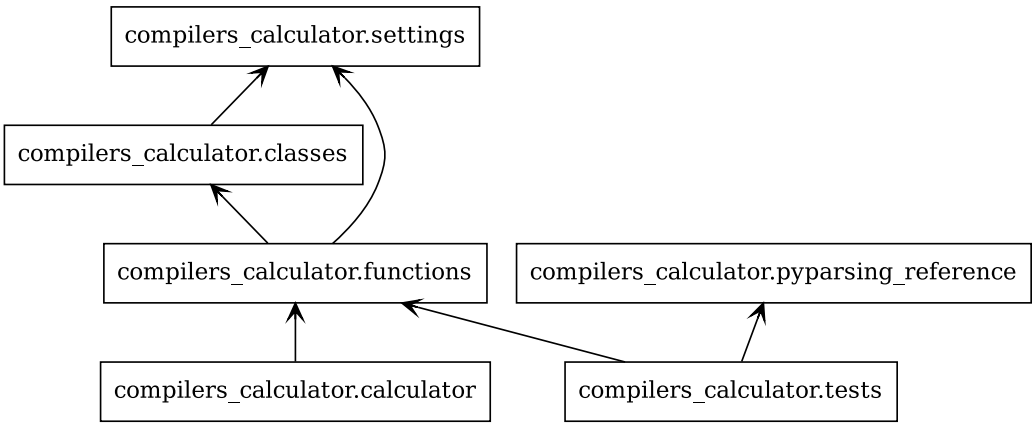
Projekt został wykonany w języku Python 3.8. Spoza pakietu standardowego zostały użyte pakiety:

- `parameterized` – do wielokrotnego uruchamiania pojedynczych testów z wieloma wartościami parametrów wejściowych,
- `pyparsing` – do stworzenia parsera "referencyjnego" na potrzeby testów jednostkowych.

6.1 Pliki

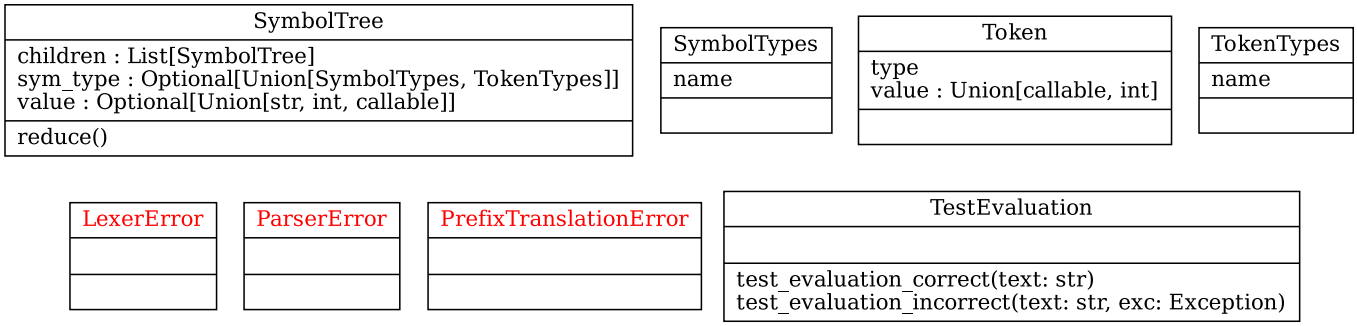
- calculator.py – moduł umożliwiający podanie wyrażenia i jego ewaluację poprzez wiersz polecenia,
- classes.py – klasy dla tokenu i drzewa symboli,
- dokumentacja.pdf – opis projektu,
- functions.py – funkcje wykonujące tokenizację, parsowanie i ewaluację oraz klasy błędów,
- pyparsing_reference.py – parser referencyjny wygenerowany pakietem `pyparsing`.
- README.md – opis pobrania i uruchomienia,
- requirements.txt – wymagane pakiety spoza biblioteki standardowej Pythona.
- settings.py – tabela parsingu, zmienne globalne projektu, klasy dziedziczące z `Enum`,
- tests.py – testy jednostkowe względem parsera wygenerowanego pakietem `pyparsing`.

6.2 Diagram zależności między pakietami w projekcie



Rysunek 1: Diagram zależności między składnikami pakietu.

6.3 Diagram klas



Rysunek 2: Diagram klas stworzonych w projekcie.