Cezary Grunwald

Projekt TKOM

Język do operacji na listach.

**Gramatyka i składnia języka:**

**<letter>** ::= ‘a’ | ... | ’z’ | ‘A’|...|’Z’;

**<digit>** ::= ‘0’ | ... | ‘9’;

**<id>::=** <letter>+;

**<number> ::=** <digit>+;

**<compilation\_unit> ::=** <function\_defs> , <program>;

**<program> ::=** “PROGRAM” , “{“ , <operation>\* , “}” “;” ;

**<function\_defs> ::=** “FUNCTIONS” , “{“ , <function\_def>\* , ”} “ , “;”;

**<assignment> ::=** <id> , “=” , <arithmetic\_operation> | <list> | <list\_element> | <number> | <function\_call> | <id> , “;”;

**<operation> ::=** <numerical\_var\_dec> | <list\_var\_dec> | <function\_call> , “;” | <function\_def> | <if\_statement> | <loop> | <return\_op> | <assignment> | <arithmetic\_operation>;

**<arithmetic\_operation> ::=** <id> | <value> | <list> | <function\_call> , “+” | “-“ | “\*” | “/”, <id> | <value> | <list> | <function\_call>;

**<return\_op> ::=** “return” , <return\_arg> “;” ;

**<list\_var\_dec> ::=** “list”, <id>, “=” <arithmetic\_operation> | <list> | <function\_call> , “;”;

**<numerical\_var\_dec> ::=** “number”, <id> , “=” , <arithmetic\_operation> | <number> | <function\_call> | <list\_element> “;”;

**<list> ::=** ‘{’ <number> (‘ ,’ , < number >)\* , ‘}’ ;

**<list\_element> ::=** <id>, ‘[‘ < number > ‘]’;

**<value> ::=** <number> | <list\_element> ;

**<if\_statement> ::=** “if”, <condition>, <then\_block>, “;” , <else\_block>;

| “if”, <condition>, <then\_block>;

**<then\_block>::=** “then”,”{“ (<operation> | <return\_op>)\* ”, ”}“ , “;“;

**<then\_block> ::=** “else”,”{“ (<operation> | <return\_op>)\* ”, ”}“ , “;“;

**<condition>** ::= “(“ , <elementary\_condition> , ((“!&&”| “&&” ) <elementary\_condition> )\*, “)“;

**<var>** ::= <id> | <value> | <list> | <function\_call>;

**<elementary\_condition> ::=** :

(<var> , (“==” | “!=“ | “>” | “<” | “>=” | “<=” , <var>);

**<function\_def>**

::= “@” , <id>, “(“ ,”)”, “{“ , <operation>\* , “}”, “;’

| “@” , <id>, “(“ ,<function\_call\_arg>(“,”,<function\_call\_arg>)\*, ”)”, “{“ , <operation>\* , “}”, “;’

**<function\_call>**

::= <id>, “(“, <function\_call\_arg> (“,”, <function\_call\_arg>)\* “)”

| <id> ,”(“,”)”;

**<function\_def\_arg> ::=** ((<number>, <id>) | (<list>,<id>));

**<function\_call\_arg> ::** = <list>

| <value>

| <id>

| <function\_call>

| <arithmetic\_operation>;

**<loop>** ::= “loop”, “(“ <value> “)” , “{“ , <operation>\* , “}” “,”;

**<return\_arg>** ::= <arithmetic\_operation> | <id> | <value> | <list> | <function\_call>;

W programie zostały zaimplementowane następujące funkcje:

- append(id listy , wartość) – dodaje element do listy,

- appendAt(id listy,wartość, indeks) – dodaje element do listy na podany indeks

- delete (id listy , wartość) – usuwa pierwsze wystąpienie elementu na liście,

- deleteFrom(id listy, indeks) – usuwa element z danego miejsca,

- deleteAll(id listy, wartość) – usuwa wszystkie wystąpienia elementu w liscie,

- sublist(id listy, start, koniec) – zwraca listę z zakresu start, koniec z listy podanej jako argument,

- equal(zmienna, zmienna) – porównuje dwie zmienne. Zwraca liczbę 0 lub 1. W porównywaniu list sprawdza czy listy składają się z tych samych elementów,

- sort (id listy) – zwraca posortowaną rosnąco listę,

- sortRev(id listy) – zwraca listę posortowaną malejąco,

- print(wartość) – loguje na konsolę i/lub do pliku wartość.

- przeciążony operator „+” - zwraca dodane do siebie dwie liczby lub złączone dwie listy.

- przeciążony operator „–„ zwraca odjęte dwie liczby lub odejmuje z listy podanej jako argument peirwszy elementy podane w liście drugiej i zwraca wynik

- przeciążony operator „\*” - mnoży przez siebie dwie liczby, lub mnoży każdy element listy przez liczbę i zwraca wynik.

- przeciążony operaotr „/” – analogicznie do powyższego.

Wszystkie funkcje wbudowane wywoływane są tak samo jak funkcje definiowane przez użytkowanika. Dlatego argumenty mogą przyjąć następujące postaci: Dla argumentu lista może to być id\_zmiennej lub bezpośrednio wpisana lista. Dla argumentu liczba może to być: bezpośrednio wpisana liczba lub Id\_zmiennej, lub element listy, lub wynik operacji arytmetycznej.

**Wejście i wyjście**

Podczas uruchamiania użytkownik ma do dyspozycji następujące komendy:

-h – pomoc,

-l – logowanie do pliku,

-c – logowanie do konsoli,

-t - logowanie stackTrace,

-s - ścieżka do pliku z kodem źródłowym,

-w – ścieżka do pliku wynikowego – domyślnie „/txt/data\_czas.txt”

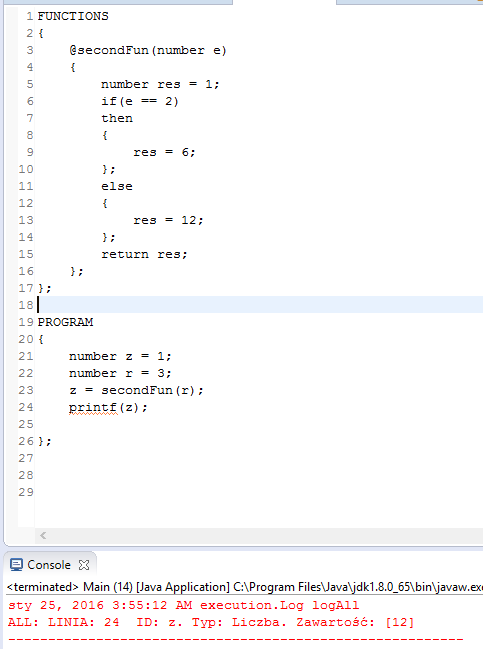
Błędy parsowania lub błędy wykonania są zapisywane do pliku lub konsoli w zależności od wybranych opcji. Jeśli wybrano logowanie StackTrace będzie on zapisywany do pliku z dopiskiem „\_STACKTRACE\_”.

**Struktura**

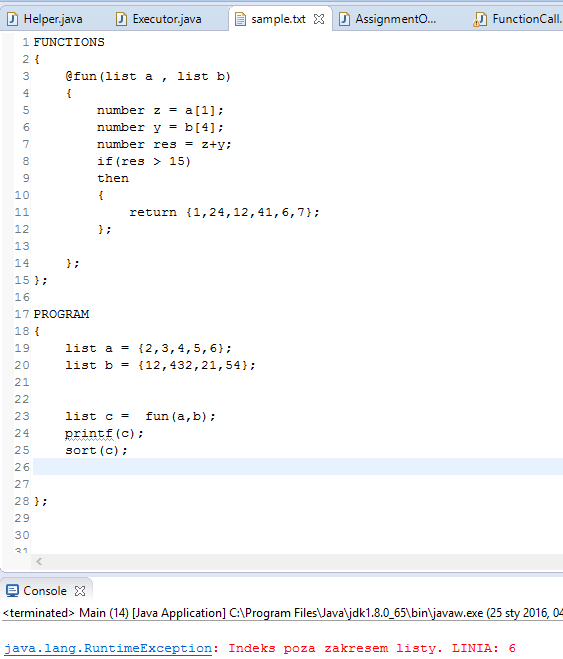
Program został napisany w języku Java z wykorzystaniem bibliotek: ANTLR v4 jako biblioteka parsująca oraz JArgs do obsługi parametrów uruchomienia. Zmienne liczbowe są przechowywane jako Integer natomiast zmienne listowe jako typ ArrayList<Integer>. Podczas działania programu przy pomocy biblioteki parsujacej zbierane są operacje znajdujące się tylko w strukturze „PROGRAM”. Następnie są one dodawane na kolejkę operacji i po zakończeniu parsowania kolejno przetwarzane przez program.

Przykłady testowe:

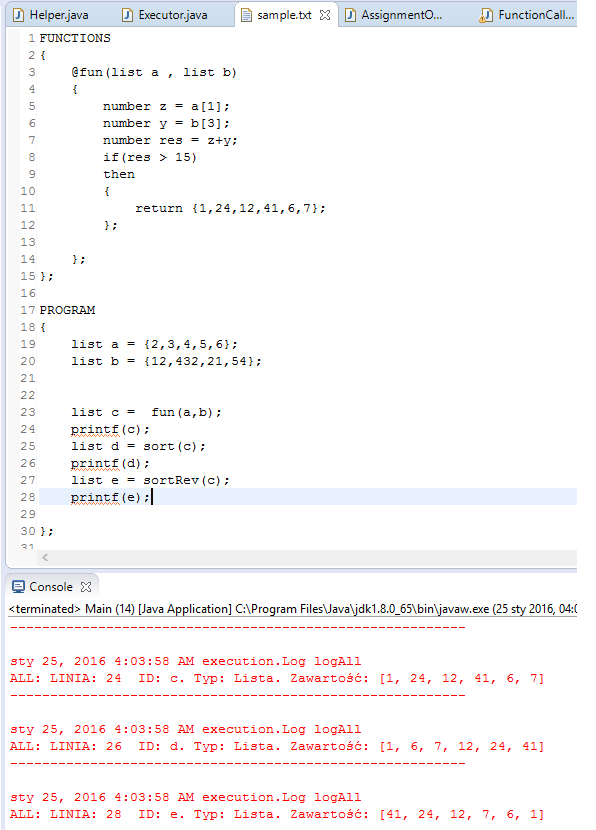
Przykładowy przebieg wykonania 1:



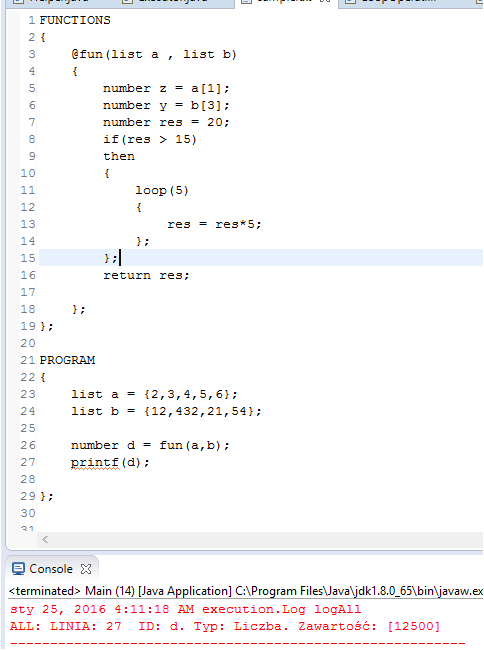
Przykładowy przebieg wykonania 2:



Przykładowy przebieg wykonania 3:



Przykładowy przebieg wykonania 4:



Przykładowy przebieg wykonania 5:

