Projekt języka z wbudowanym typem macierzowym – dokumentacja końcowa

Dominik Orliński

Spis treści

[Opis funkcjonalności 1](#_Toc514778020)

[Formalna specyfikacja 2](#_Toc514778021)

[Opis modułów i sposobu ich realizacji 3](#_Toc514778022)

[Obsługa błędów 3](#_Toc514778023)

# Opis funkcjonalności

Język, który stworzyłem jest prosty językiem opartym na uproszczonej składni języka C. Język zawiera dwa wbudowane typy - typ int (zmienne całkowite) oraz typ matrix (zmienne typu macierzowego). Język ten pozwala między innymi na :

* użycie instrukcji warunkowej if
* utworzenie pętli while
* tworzenie zmiennych publicznych (widocznych w całym programie)
* tworzenie zmiennych prywatnych (widocznym w określonym bloku)
* definiowanie prostych funkcji zwracających wartość
* tworzenie bloków (wyodrębnianie zmiennych)
* dodawanie komentarzy w kodzie
* wykonywanie operacji arytmetycznych na zmiennych
* dostęp do elementu macierzy
* wypisywanie wartości zmiennych na standardowe wyjście

Przykłady typu macierzowego :

- macierz 1 x 1 - macierz 1 x 5 - macierz 3 x 4

Na zmiennych typu int można wykonywać operacje dodawania, odejmowania, mnożenia, dzielenia. Na zmiennych typu matrix można wykonać operacje dodawania i odejmowania pod warunkiem, że liczba kolumn macierzy A równa się liczbie kolumn macierzy B oraz liczba wierszy macierzy A równa się liczbie wierszy macierzy B. Mnożenie macierzy jest dozwolone jeśli liczba kolumn macierzy A jest równa liczbie wierszy macierzy B (zakładając kolejność A \* B). Dzielenie macierzy nie jest możliwe. Nie można mieszać typów podczas wykonywania działań arytmetycznych. Elementem macierzy nie może być macierz. Nie wolno przypisywać wartości do pojedynczej komórki macierzy.

W moim języku wartości są wyrażane:

* za pomocą Identyfikatora
* wprost poprzez wartości
* poprzez wartość zwracaną z funkcji
* poprzez element macierzy

# Przykładowy kod i omówienie

Przykładowy kod i komentarze w oddzielnym dokumencie (example.txt).

# Formalna specyfikacja

Poniżej przedstawiam podział symboli leksykalnych (tokenów) :

1. Słowa kluczowe :
   * “ main ” – główna funkcji, od której rozpoczyna się wykonanie skryptu (de facto nie jest to słowo kluczowe, ale bez tej funkcji nie uruchomimy programu)
   * ” function ” – tworzenie funkcji
   * ” matrix ” – tworzenie zmiennej o typie macierzowym
   * ” int ” – tworzenie zmiennej o typie int
   * ” return ” – zwracanie wartości z funkcji
   * ” while ” – wykonanie pętli (iteracji)
   * ” if ” – instrukcja warunkowa
   * ” else ” – rozszerzenie instrukcji warunkowej
   * ” public ” – określanie zmiennej jako lokalną
   * ” private ” – określanie zmiennej jako globalną
2. Operatory
   * Arytmetyczne : ” - ”, ” + ”, ” \* ”, ” / ”
   * Logiczne : „ ! ”, „ & ”, „ | ”,
   * Przypisania : „ := ”
   * Porównania : „ = ”, „ < ”, „ > ”, „ <= ”, „ >= ”, „ != ”
   * Dostępu do elementu macierzy ‘tiger’ : „ tiger[indeks,indeks] ”
3. Symbole specjalne :
   * Nawiasowanie : „ ( ”, „ ) ”, „ { ”, „ } ”
   * Komentowanie : „ // komentarz” ‘# komentarz #’
   * Terminator instrukcji : „ ; ”
4. Separatory : spacja, tablulacja, znak nowej linii
5. Identyfikatory : dowolne ciągi znaków nie będące słowami kluczowymi np. „main”, „tmp”, „Foo”, „\_myFoo”
6. Stałe liczbowe : np. „12345”, „-123”
7. Napisy (ciagi znaków) : np. „ala ma kota”

# Opis modułów i sposobu ich realizacji

Source - jest modułem odczytującym pojedyncze znaki z wejścia. Odczyt jest realizowany znak po znaku, tylko na żądanie modułu Scanner. Moduł source dodatkowo trzyma pozycję kursora tak aby możliwe było zlokalizowanie błędu.

Scanner - odpowiada za tokenizację (lekseryzację). Odczytuje kolejne tokeny na żądanie modułu Parser. Moduł ten zgłasza dwa rodzaje błędów - zbyt długie identyfikatory i zbyt długie stałe liczbowe.

Parser - dokonuje analizy składniowej, rozbiera cały plik pod względem znaczeniowym (semantycznym). Tworzy odpowiednie struktury danych, z których korzysta Executor. Moduł Parser współpracuje z modułem Scanner prosząc go o kolejne tokeny. Wszystkie błędy, które zgłasza powodują wyjątek i przerwania wykonanie programu. Podczas tego etapu zgłaszanych jest najwięcej błędów. Są to wszystkie błędy konstrukcji, składni. nierozpoznania słowa kluczowego, braku odpowiednich tokenów na odpowiednich miejscach.

Executor - jest odpowiedzialny za wykonanie programu. Jest całkowicie odłączony od źródła i leksera. Korzysta tylko ze struktur pozostawionych przez moduł parsera. Rozpoczyna wykonanie od funkcji o nazwie ‘main’, która musi znaleźć się w programie.

Każdy z modułów znajduje się w osobnym pliku.

# Obsługa błędów

Błędy wykrywane są na każdym etapie analizy. Etapami, podczas których pojawia się najwięcej błędów jest prasowanie i analiza semantyczna. Zgłaszanie błędów na każdym etapie pozwala na przerwanie analizy błędnego kodu jak najszybciej i nie wykonywaniu zbędnych operacji.

Typy błędów zgłaszane przez każdy z modułów :

Source – brak możliwości otwarcia pliku

Scanner – błędne (nierozpoznane) znaki, zbyt długi identyfikator, za duża stała

Parser – błędy składniowe typu „oczekiwano symbolu } otrzymano symbol ;”

Executor – błędy wykonania np. przypisanie zmiennej matrix do zmiennej int, wykonanie niedozwolonej operacji dzielenia na macierzach, brak zadeklarowania zmiennej, odnoszenie się do zmiennej poza zasięgiem widoczności