Informatyka, 3 rok

Wydział EAIiIB

Teoria Kompilacji i Kompilatory

Projekt zaliczeniowy

**KALKULATOR**

**Krzysztof Misiak**

**Filip Pasternak**

# Opis działania kalkulatora

Celem projektu jest przedstawienie działania prostego kalkulatora.

Funkcjonalności kalkulatora :

* przetwarzanie podstawowych działań algebraicznych (dodawanie, odejmowanie , mnożenie, dzielenie, potęgowanie, negacja)
* wczytywanie i wypisywanie działań w konsoli
* wczytywanie i zapisywanie działań do pliku
* wyświetlanie błędu w przypadku nieprawidłowego działania wejściowego
* operacje na liczbach rzeczywistych
* obsługa funkcji trygonometrycznych
* obsługa zmiennych

Projekt realizowany jest w języku Java.

# Opis gramatyki

Produkcje używanej gramatyki według notacji Backusa-Naura

<wejscie> ::= linia

|<wejscie><linia>

<linia> ::= ‘\n’

| <wyrazenie> ‘\n’

| <identyfikator> = <wyrazenie> ‘\n’

<trygonometria> ::= sin(<wyrazenie>)

| cos<wyrazenie>

| tg(<wyrazenie>)

| ctg(<wyrazenie>)

<wyrazenie> ::= <liczba>

| <identyfikator>

| <trygonometria>

| <wyrazenie> + <wyrazenie>

| <wyrazenie> – <wyrazenie>

| <wyrazenie> \* <wyrazenie>

| <wyrazenie> / <wyrazenie>

| – <wyrazenie>

| <wyrazenie> ^ <wyrazenie>

| ( <wyrazenie> )

<zero> ::= 0

<cyfra niezerowa> ::= 1

| 2

| 3

| 4

| 5

| 6

| 7

| 8

| 9

<cyfra> ::= <zero>

| <cyfra niezerowa>

<ciag cyfr> ::= <cyfra>

| <cyfra><ciag cyfr>

<calkowita> ::= <cyfra>

| <cyfra niezerowa><ciag cyfr>

<liczba> ::= <calkowita>

| <calkowita> . <ciag cyfr>

<identyfikator> ::= a

|(…)

|z

|A

|(…)

|Z

|<identyfikator><identyfikator>

|<identyfikator><cyfra>

# Skaner

## Opis skanera

Skaner został skonstruowany tak, aby na podstawie opisanej w poprzednim rozdziale gramatyki wyodrębniać z otrzymanego na wskazanym wejściu łańcucha znaków tokenów charakterystycznych dla kalkulatora. Tokeny reprezentowane są przez instancje klasy Java\_cup.runtime.**Symbol**, która to posiada pola ‘sym’ i value’ oznaczające odpowiednio rodzaj tokena (określone w interfejsie **sym**) i opcjonalna wartość(dotyczy takich tokenów jak liczba czy identyfikator, wówczas **value** reprezentuje wartość liczbową lub łańcuch znaków).

Skaner ten umożliwia wprowadzanie na wejściu komentarzy w notacji języka Java (komentarz liniowy i blokowy) oraz JavaDoc. Napotykając takowe ignoruje je, dzięki czemu nie mają wpływu na kod wynikowy.

## Sposób tworzenia

Do stworzenia skanera wykorzystane zostało narzędzie JFlex, które pozwala na generowanie analizatorów leksykalnych w języku Java w oparciu o plik z rozszerzeniem ‘.flex’. Plik ten dzieli się na trzy części (oddzielone znakiem %%).

Pierwsza zawiera kod w języku Java, który zostanie wstawiony bez zmian do plików wynikowych (.java). Umożliwia stworzenie dodatkowych elementów potrzebnych w programie (np. bufory, klasy) czy też dołączenie dodatkowych plików, bibliotek. Została użyta do stworzenia zaimportowania pakietu **java\_cup.runtime**  i **java.util**.

Druga sekcja to miejsce na definiowanie makr dla skanera. Polega to na podaniu nazwy makra i odpowiadającego mu wyrażenia regularnego. Makra te można wykorzystać w kolejnej części pliku flex opisanej niżej. W drugiej części można wykorzystać także dodatkowe opcje dla skanera lub poźniejszego parsera, określać stany w jakich może znajdować się skaner. W naszym przypadku użyty został parametr **class** określający klasę skanera, parametry **line i kolumn** umożliwiające wyłapywanie lokalizacji danego tokena, a także parametr **cup**, który jest wręcz nieodzowny do współpracy z późniejszym parsere.

Ostatnia sekcja pliku flex jest kluczowa z punktu widzenia skanera. Określa się tutaj jego konkretne zachowanie w zależności od dopasowanego wzorca. Dla poszczególnych wzorców podane są instrukcje w języku Java, które podejmowane są w momencie wykrycia.

# Parser

## Opis Parsera

Parser został utworzony z wykorzystaniem narzędzia CUP. Na podstawie pliku parser.cup generowana jest klasa java, w której zawarte są wygenerowane metody oraz metody odziedziczone po interfejsie java\_cup.runtime.Parser. Parser został tak utworzony, by korzystać z wcześniej stworzonej klasy Scanner. W konstruktorze ustawiany jest obiekt konkretnego skanera, który generuje symbole(tokeny). Jako drugi parametr przyjmuje obiekt klasy Writer, który pełni rolę opakowania strumienia wyjściowego i pozwala na ujednolicenie mechanizmu prezentacji wyniku działania niezależnie od rodzaju strumienia wyjściowego.

W tej klasie jest też definiowana gramatyka, symbole terminalne oraz nieterminalne, a także priorytety poszczególnych działań, w tym nawiasowanie.

## Sposób tworzenia

Plik parsera składa się z kilku części:

* parser code – część kodu dołączona do wynikowej klasy
* init with – częśc kodu odpowiadająca za inicjalizację
* scan with – część kodu odpowiadająca za wczytywanie kolejnych tokenów
* definicja symboli terminalnych, nieterminalnych i ich priorytetów (określone rodzaje tokenów zapisane są w interfejsie **sym**
* gramatyka – definicja reguł gramatyki i odpowiadająca im reakcja po stronie Javy

# Graficzny Interfejs Użytkownika

Interfejs graficzny stworzony został z użyciem biblioteki **Swing**. Okno programu podzielone jest na trzy główne części

* Przestrzeń tekstowa dla kodu wejściowego dla kompilatora.
* Przestrzeń tekstowa, w której prezentowany jest wynik działania kompilatora.
* Przestrzeń zarządzająca, w której użytkownik może wybrać interesujące go opcje oraz rozpocząć kompilację.

Program umożliwia odczyt z dysku pliku tekstowego zawierające wyrażenia arytmetyczne zgodne z podaną wcześniej gramatyką. Można także tworzyć wyrażenia ‘na bieżąco’ korzystając z udostępnionego pola. Podobnie sprawa wygląda z wynikiem działania kompilatora. Użytkownik może zdecydować czy chce zapisać wynik do pliku, czy może wyświetlić w oknie programu.