AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Stworzenie kompilatora do języka LOLCODE za pomocą narzędzia ANRLv4

Nikita Grygoriev, Mikołaj Ogarek,
Zuzanna Śmiech, Małgorzata Barnach

OPIS ZAKRESU PRACY

Celem danego projektu było napisanie kompilatora do języka LOLCODE. Do wspomagania pisania kompilatora było wykorzystane narzędzie ANTLRv4, które tworzy parser i lekser na podstawie plików gramatyki (o rozszerzeniu .g4). Zakres prac obejmował tworzenie pliku gramatyki, który powinien był posiadać zdefiniowane reguły analizatora leksykalnego i składniowego oraz napisanie "listenera" (programu który dostarcza możliwość poruszania się drzewem składniowym), ponieważ narzędzie ANTLR nie generuje go.

Gramatyka

Pliki gramatyki powinne mieć strukturę:

reguła : **wyrażenie regularne opisujące regułę** ;

Reguła może mieć dowolny identyfikator pod warunkiem, że jest on unikalny w ramach gramatyki. (np. program, loop, comment).

Wyrażenie regularne może się składać z:

- innych reguł
- '+' oznacza 1 lub więcej wyrażeń podanych przed danym symbolem
- '|' alternatywa (stosuje się między dwoma wyrażeniami)
- '.' dowolny symbol
- ' ' ciąg znaków znajdujący się w cudzysłowie pojedynczym
- itd

Gramatyka jest niezbędna dla ANTLR do stworzenia podprogramów Java **lolcodeLexer** oraz **lolcodeParser**. Pliki te są dostępne do edycji, ale jej nie wymagają, ponieważ tuż po stworzeniu przez ANTLR są gotowe do użytku. Trzecim plikiem wytwarzanym przez ANTLR jest plik listenera, na którym głównie skupialiśmy się podczas dalszej pracy.

Listener

Ten podprogram służy do poruszania się drzewem składniowym i składa się z metod abstrakcyjnych **enter**Nazwa reguły i **exit**Nazwa reguły. Jedynym ograniczeniem nałożonym przez listener jest obiekt kontekstu **ctx**. Przez dany obiekt jest dostarczany dostęp do wszystkich reguł składniowych występujących w programie do danego momentu (np. nazwy zmiennych, struktur danych, operacji matematycznych). Reszta implementacji jest dowolna. W danym projekcie pamięć programu została zaimplementowana za pomocą mapy (kluczem są nazwy zmiennych, wartością - ich wartości) i stosu (na górze stosu jest przechowywana ostatnia wartość zmiennej lub wynik operacji matematycznej).

KRÓTKI STEP-BY-STEP

Korzystając z artykułu How to Create a Programming Language Using ANTLR4, możemy stworzyć prosty język składający się z kilku reguł typu przypisywanie liczby całkowitej do zmiennej, dodawanie dwóch liczb całkowitych i wyświetlanie wartości zmiennej. Następnie, po otrzymaniu odpowiednich plików wynikowych z wykorzystaniem ANTLR4, należy zaimplementować możliwość dodawania i odczytywania z mapy zmiennych w pliku listenera. Na sam koniec, należy stworzyć klasę **Main** w której będzie definiowany lekser, następnie dany lekser będzie przekazany do konstruktora parsera i na końcu za pomocą metody **.addParseListener** listener będzie dodany do parsera. Po wykonaniu powyższych kroków prosty kompilator dowolnego języka jest gotowy do użytku. Żeby to sprawdzić na dowolnym programie wystarczy wpisać w terminalu polecenie:

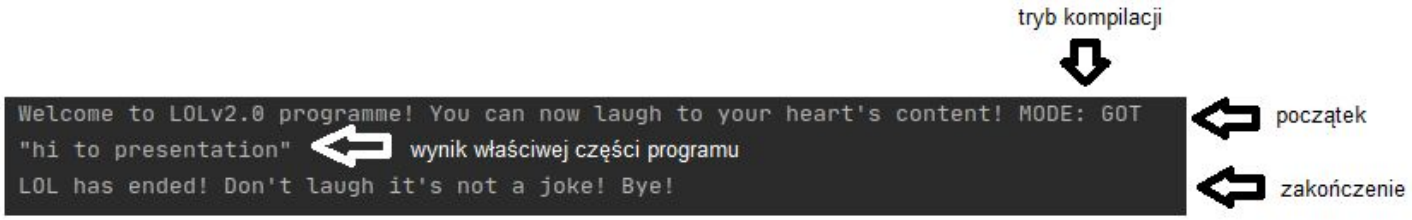
```
java   nazwa listenera   plik z kodem w nowym języku
```

PRZYKŁADY

lolProgram1.lol

Input:
HAI GOT
VISIBLE "hi to presentation" MKAY?
KTHXBYE

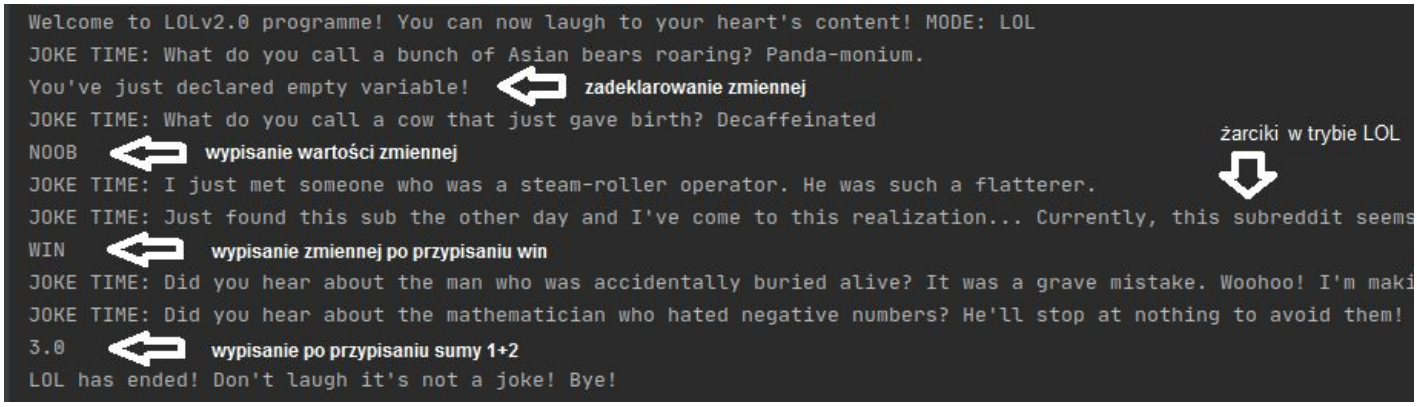
Output:



lolProgram2.lol

Input:
HAI LOL
I HAS A zmienna
VISIBLE zmienna MKAY?
zmienna R WIN
VISIBLE zmienna MKAY?
zmienna R SUM OF 1 AN 2
VISIBLE zmienna MKAY?
KTHXBYE

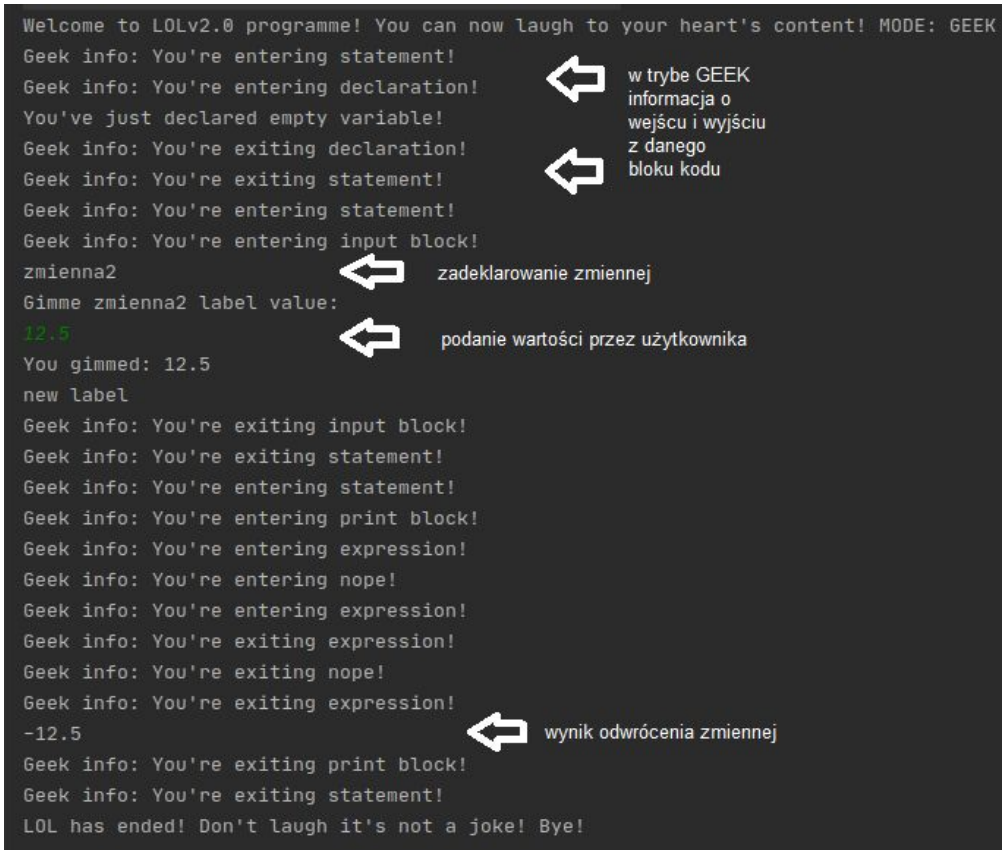
Output:



lolProgram3.lol

Input:
HAI GEEK
I HAS A zmienna2
GIMMEH zmienna2
VISIBLE NOT zmienna2 MKAY?
KTHXBYE

Output:



Program został skompilowany w trybie GEEK, więc przed wejściem i po wyjściu z każdej części kodu pojawia się odpowiednia informacja.

lolProgram4.lol

Input:
HAI GOT
I HAS A zmienna ITZ "text"
VISIBLE BOTH OF 12 AN WIN MKAY?

VISIBLE DIFF OF BOTH OF 12 AN WIN AN 1.0 MKAY?
VISIBLE SUM OF DIFF OF BOTH OF 12 AN WIN AN "l" AN zmienna MKAY?
KTHXBYE

Output:

```
Welcome to LOLv2.0 programme! You can now laugh to your heart's content! MODE: GOT
You've decalred: zmienna with value: "text"
WIN
0.0
4.0
LOL has ended! Don't laugh it's not a joke! Bye!
```

Wynikiem *BOTH OF 12 AN WIN* jest WIN, ponieważ zarówno WIN jak i 12 nie jest nullem. Wynikiem *DIFF OF BOTH OF 12 AN WIN AN 1.0* jest 0.0, ponieważ WIN jest odpowiednikiem 1.0 . W działaniach zmienne do których przypisane są wartości tekstowe zamieniane są na ich długości, więc wynikiem *SUM OF DIFF OF BOTH OF 12 AN WIN AN "l" AN zmienna* jest 4.0, czemu odpowiada działanie ((1.0 i 1.0) - 1.0) + 4.0 .

lolProgram5.lol

Input:
HAI GOT
BTW "elko tu jest komentarz"
OBTW
SUM OF 12 AN 16
TLDR
KTHXBYE

Output:

```
Welcome to LOLv2.0 programme! You can now laugh to your heart's content! MODE: GOT
ICYMI: there is comment! IDC but IMO you should read it.
ICYMI: there is comment! IDC but IMO you should read it.
LOL has ended! Don't laugh it's not a joke! Bye!
```

Program skompilowany w trybie GOT, więc wypisuje się jedynie informacja o komentarzu.

lolProgram6.lol

Input:

```
HAI GOT
O RLY?
YA RLY
BOTH SAEM 1 AN 2
VISIBLE "you are in first if" MKAY?
MEBBE DIFFRINT 3 AN 3
VISIBLE "you are in else if" MKAY?
NO WAI
VISIBLE "you are in else" MKAY?
OIC
KTHXBYE
```

Output:

```
Welcome to LOLv2.0 programme! You can now laugh to your heart's content! MODE: GOT
"you are in else"
LOL has ended! Don't laugh it's not a joke! Bye!
```

Przykład użycia if. Pierwszy if zostaje pominięty, ponieważ warunek (1==2) zawsze będzie nieprawdziwy. Drugi warunek (3!=3) również jest nieprawdziwy, więc następuje wypisanie *you are in else*.