Kompilatory i teoria kompilacji

Dokumentacja

Własny język programowania - POLang

Adrianna Kopeć
Sylwia Kwiatkowska
Anna Nagi

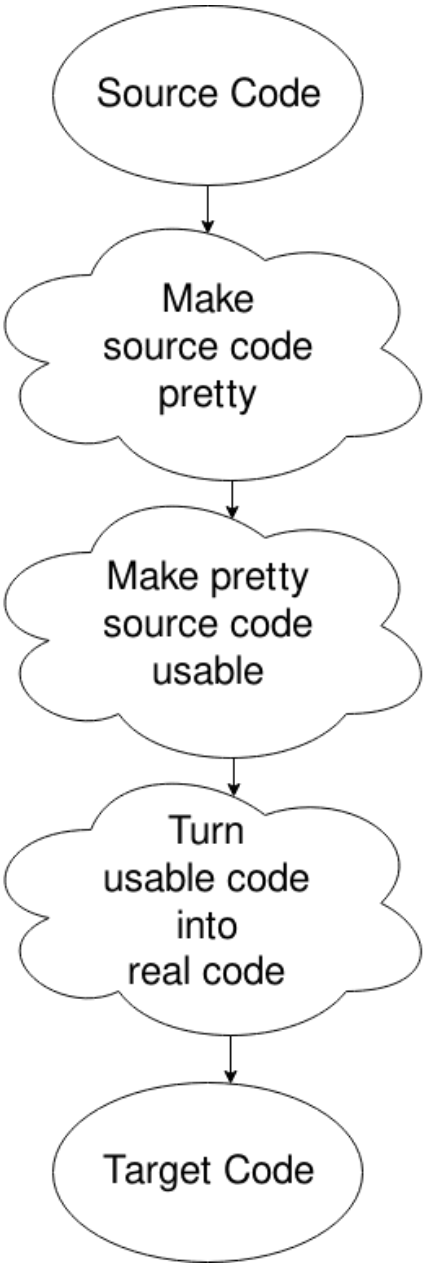
Spis Treści

Opis projektu

Celem projektu jest stworzenie własnego języka programowania. Nasz język wyróżnia się przede wszystkim zastosowaniem polskich słów kluczowych (*keywords*). Stworzony przez nas język wzorowany jest składnią Javy. Cechuje się silnym, statycznym typowaniem, a strukturę programu tworzy klasa, której najważniejszym elementem jest funkcja start() - punkt wejścia do programu. Język jest tłumaczony na język Python, co umożliwia uruchomienie kodu po translacji poprzez odpowiadający mu plik .py.

Teoria tworzenia własnego języka programowania

Podstawą tworzenia własnego języka jest zdefiniowanie jego gramatyki. Na jej podstawie dokonuje się analizy składniowej programu, a także określa jego semantykę, co pozwala uznać dany kod za prawidłowy oraz wykonać odpowiednie akcje.



Rys. 1 Fazy tworzenia kodu, źródło: https://dev.to/

Składnię programu można podzielić na dwie części: leksykalną i gramatyczną.

Lekser
Odpowiada za analizę leksykalną, czyli przetwarzanie sekwencji znaków kodu źródłowego w sekwencję tokenów. Token jest małą jednostką języka, która reprezentuje leksem i wprost go kategoryzuje - jest odpowiednikiem części mowy w języku naturalnym.

Reguły gramatyki definiujące składnię leksykalną określają zbiór możliwych sekwencji znakowych, które tworzą pojedyncze tokeny. Na ich podstawie lekser dokonuje tokenizacji - przetwarza oddzielone tzw. “białymi znakami” ciągi znaków i dla każdego z nich produkuje token, bądź ogłasza błąd analizy leksykalnej.

Parser

Odpowiada za analizę gramatyczną, czyli sprawdza składnię i buduje strukturę danych np. drzewo parsowania. Reguły parsingu reprezentowane są za pomocą wyrażeń regularnych i określają komponenty operacji jak i ich kolejność. Zazwyczaj praca parsera polega na przetworzeniu listy tokenów dostarczonej przez lekser do struktury drzewa, podczas którego sprawdza czy dane wyrażenie jest dopuszczalne.

Proces tworzenia naszego języka

Pierwszym etapem prac nad naszym językiem było stworzenie jego gramatyki, czyli zdefiniowanie reguł dla leksera i parsera. Znajduje się ona w pliku **Lang.g4**.

Następnie na jej podstawie, za pomocą narzędzia ANTLR4 został wygenerowany lexer (**LangLexer.py**) oraz parser (**LangParser.py**)

w języku Python. Parser jest rodzaju **LL**(*), czyli lewostronny parser z podglądem dowolnej liczby symboli.

Za pomocą narzędzia **ANTLR TestRig** (**Grun**) możliwe było testowanie napisanych reguł dla naszego języka z możliwością wizualizacji powstałego drzewa parsowania.

Ostatnim etapem było stworzenie translatora naszego języka na język Python. Głównym programem jest **Main.py**, który generuje drzewo parsingu na podstawie pliku wejściowego napisanego w języku "NAZWA".

Drzewo przeglądane jest od lewej do prawej przez obiekt klasy ParseTreeWalker, który generuje zdarzenia m.in. wejście czy wyjście z danego węzła. Sygnały te są następnie interpretowane przez Listener zaimplementowany w **LangVarListener.py**, który odpowiedzialny jest za przetłumaczenie poszczególnych wyrażeń na ich odpowiedniki w Pythonie.

Program generuje plik z rozszerzeniem .py, którego uruchomienie odpowiada działaniu kodu w naszym języku.

Instalacja języka i pierwszy program

Pobranie kodu z repozytorium

Aby używać naszego języka programowania należy pobrać repozytorium znajdujące się na GitHubie:
https://github.com/skwiatkowska/compilers-project

Instalacja Pythona 3 i biblioteki ANTLR4

Python 3: https://www.python.org/downloads/
ANTLR4: https://www.antlr.org/download.html

Generowanie leksera i parsera za pomocą biblioteki ANTLR4 odbywa się poprzez komendę:
antlr4 Grammar.g4,
gdzie Grammar.g4 to plik z regułami gramatycznymi
Podając flagę -o, można wybrać folder, w którym zostaną umieszczone wygenerowane pliki, natomiast flagą -Dlanguage=xxx można wyspecyfikować język, w którym te pliki będą napisane.

Przykład testowania gramatyki za pomocą narzędzia grun:
grun Grammar rule -gui,
gdzie Grammar to plik z regułami gramatycznymi, podany bez rozszerzenia .g4, a rule to reguła gramatyczna, którą chcemy przetestować.

Flaga -gui umożliwia przedstawienie drzewa parsingu w formie graficznej, co ułatwia analizę poprawności gramatyki.
Po uruchomienie powyżej komendy w konsoli należy wpisać lub wkleić fragment kodu z odpowiednią regułą do przetestowania.

Pierwszy program - “Hello world!”

Zapisanie przykładowego kodu

Należy wkleić poniższy kod do notatnika i zapisać go w pliku o rozszerzeniu “.txt”.

```
klasa Klasa[
        start[
                pisz(“Hello world!”);
        ]
]
```

Zmiana nazwy w pliku Main.py

Następnie w pobranym wcześniej pliku Main.py należy zmienić nazwę pliku wejściowego z test1.txt na nazwę naszego pliku i uruchomić program Main.py

Uruchomienie wygenerowanego programu w Pythonie

W folderze, w którym jest pobrany projekt powinien pojawić się nowy plik o rozszerzeniu “.py”. Należy uruchomić go za pomocą Pythona.

Program powinien wyświetlić “Hello world!”.

Składnia języka i przykłady użycia

Typy zmiennych akceptowane przez język

Liczba całkowita

'Calkowita' - integer

Liczba zmiennoprzecinkowa

'Rzeczywista' - double

Ciąg znaków

'Napis' - string

Typ logiczny

'TypLogiczny' - boolean

Możliwe typy logiczne:
'prawda' - true
'falsz' - false

Deklarowanie i definiowanie zmiennych

Deklarowanie zmiennej

Dokonujemy jej za pomocą słowa kluczowego określającego typ zmiennej.

Definiowanie

Dokonujemy jej za pomocą słowa kluczowego określającego typ zmiennej i nadania zmiennej wartości

Rozdzielanie bloków kodu

Dokonujemy go za pomocą nawiasów kwadratowych:
'[' ... ']' - block of code

Podstawowe słowa kluczowe

Wyświetlanie na ekranie

'pisz' - print

Tworzenie głównej funkcji programu

'start' - main function

Tworzenie innych funkcji

'funkcja' - function

Zwracanie wartości przez funkcję

'zwroc' - return

Tworzenie klasy

'klasa' - class

Używamy do tego celu słowa kluczowego *klasa*. Blok tworzący klasę musi być rozdzielony za pomocą nawiasów kwadratowych.

Operatory

Operator koniunkcji

'oraz' - and

Operator alternatywy

'lub' - or

Operator nierówności

'!=?=' - not equal

Operator równości

'=?=' - equal

Operator większości / i równości

'>?' - greater
'>=?' - greater or equal

Operator mniejszości / i równości

'<?' - less
'<=?' - less or equal

Operator przypisania wartości

'<=' - assign value

Kontrola przebiegu programu

Instrukcje warunkowe: if / else if / else

'jezeli' - if

'inaczejJezeli' - else if

'inaczej' - else

Pętle

'dlaKazdego' - for

'wZakresie' - in range

'dopoki' - while

Komentarze

Komentarz jednolinijkowy

'***' - comment

Komentarz wielolinijkowy

'#/#' ...'#/#' - multi-line comment

Przykładowy program

Przykładowy kod umieszczony w pliku “test1.txt”

```
*** komentarz jednolinijkowy

#/#
komentarz
wielolinijkowy
#/#

klasa Klasa[
	Napis nazwa1;
	Calkowita num1;
	Calkowita num2 <= 51;
	Tablica Napis tablica1 <= {"a", "b", "c"};
	funkcja fun2(Calkowita a, Calkowita b) zwroc Napis;


	funkcja fun1(Calkowita i, Napis s) zwroc Calkowita[
		Napis nazwa <= s;
		pisz(nazwa);
		Calkowita num2 <= i + 1;
		zwroc (num2);
	]

	funkcja fun2(Calkowita a, Calkowita b) zwroc Napis[
		jezeli(a <? b)[
			zwroc("1");
		]
		inaczejJezeli(b =?= a)[
			zwroc("2");
		]
		inaczej[
			zwroc("3");
		]
	]

	funkcja fun3()[
		dlaKazdego (i wZakresie 0..5) [
			pisz(i);
		]
	]
```

```
        funkcja fun4()[
                Calkowita x <= 0;
                dopoki (x <? 10) [
                        pisz(x);
                        x <= x+1;
                ]
        ]


        start[
                num1 <= 68;
                nazwa1 <= fun2(3, 5);
                pisz(nazwa1);
                fun3();
                num1++;
        ]
]
```

Wygenerowany plik Pythona:

```
num2 = 51

tablica1 = ["a", "b", "c"]


def fun1(i, s):
        nazwa = s
        print(nazwa)
        num2 = i + 1
        return num2


def fun2(a, b):
        if a < b:
            return "1"
        elif b == a:
            return "2"
        else:
            return "3"
```

```
def fun3():
        for i in range(0, 5):
            print(i)


def fun4():
        x = 0
        while x < 10:
           print(x)
           x = x + 1


def main():
        num1 = 68
        nazwa1 = fun2(3, 5)
        print(nazwa1)
        fun3()
        num1 += 1


if __name__ == '__main__':
        main()
```

Wynik działania programu:

```
1
0
1
2
3
4
```

Podział obowiązków podczas tworzenia projektu

Stworzenie gramatyki, wygenerowanie lexera, parsera i drzewa parsowania - Sylwia Kwiatkowska

Stworzenie listenera, za pomocą którego język jest tłumaczony na Pythona - Adrianna Kopeć, Anna Nagi

Napisanie dokumentacji - Adrianna Kopeć, Anna Nagi