**Компајлер за Микро Јаву**

**Душан Стијовић**

2017/0145

Садржај

[Кратак опис пројекта 3](#_Toc61829248)

[Опис команди: 4](#_Toc61829249)

[Кратак опис тестова 6](#_Toc61829250)

[Кратак опис новоиведених класа 7](#_Toc61829251)

## Кратак опис пројекта

Пројекат “компајлер за Микро Јаву“ се састојао из неколико фаза и циљ му је био да нас научи како тече процес развоја компајлера и како да направимо компајлер за неки језик.

Фазе које су биле обухваћене пројектом су:

* Лексичка анализа
* Синтаксна анализа
* Семантичка анализа
* Генерисање кода

Приликом лексичке анализе научили смо да користимо алат JFlex за генерисање лексера, односно препознавача текста.

Приликом синтаксне анализе научили смо да користимо алат CUP за генерисање парсера. Наш задатак је био да напишемо граматику а алат је на основу тога направио парсер. Такође смо користећи надоградњу алата CUP коју поседује наш факултет генерисали апстрактно синтаксно стабло заједно са класама које смо користили приликом писања граматике.

Фаза семантичке анализе се састојала од обилска апстрактног синтаксног стабла помоћу visitor-a, и попуњавања табеле симбола одговарајућим конструктима језика. У овој фази смо користили готову имплементацију табеле симбола. У овој фази осим попуњавања табеле симбола смо проверавали све сематичке и контексне услове нашег језика.

Фаза семантичке анализе се састојала од обиласка апстракног синтаксног стабла помоћу visitor-a и генерисање објектног кода. За генерисање кода смо користили табелу симбола коју смо попунили у претходној фази. За писање асемблерских инструкција смо користили класу Code. Класа Code садржи методе које нам олакшавају писање асемблерског кода.

## Опис команди:

Како би Компајлер исправно функционисао потребно је направити класе које су потребне за његов рад.

1. Прво је потребно направити класу Yylex.java која представља лексер, односно која скенира улазни текст и генерише токене.
   * Ово се ради помоћу build targeta “LexerGen“ који се налази у build.xml фајлу.
2. Затим је потребно направити класу MJParser.java која представља парсер.
   * Ово се ради помоћу build targeta “Parser Gen” који се налази у build.xml фајлуу.
3. Затим је потребно позвати buil target “repackage” који се налази у build.xml фајл.
   * Ово је потребно урадити како бисмо преместили класе које смо генерисали помоћу алата у одговарајући директоријум.
4. На крају је потребно превести све класе заједно како бисмо добили exe фајл за наш компајлер.
   * Ово се ради build targetom "compile" који се налази у build.xml фајлу.
   * Циљ compile позива све горње циљеве.
   * Уколико не желите да користите горње циљеве можете користити командну линију тако што ћете аргументе из циљева написати у командној линији.
5. Сада је потребно превести програм написан на Микро Јави.
   * То се може или помоћу eclips-а или помоћу командне линије. Наш програм прима два аргумента. Имe улазног фајла који преводи и име излазног фајла у који ће уписати објектни код за преведени програм.

**java Compiler.class program.mj program.obj**

Подразумевано се подаци који се добијају приликом процеса компајлирања(лексичка анализа, парсирање, семантичка анализа) исписују на стандардни излаз. Ту су укључене и грешке. Осим на стандардни излаз подаци се уписују и у лог фајлове. Уколико желите да преусмерите ове податке у неки други фајл потребно је да додате још неке аргументе у командну линију.

За преусмеравање исписа података у неки фајл **1>ime\_izlaznog\_fajla.out**

За преусмеравање грешака у неки фајл **2>ime\_izlaznog\_fajla\_za\_greske.err**

**java Compiler.class program.mj program.obj 1>ime\_izlaznog\_fajla.out**

**2>ime\_izlaznog\_fajla\_za\_greske.err**

1. Сада је потребно покренути програм. Ово се ради помоћи класе Run или помоћу build targeta “runObj” који се налази у build.xml фајлу.
   * Уколико се користи командна линија:
     1. java Run.class program.obj
   * Уколико нам је потребно да видимо садржај можемо користити алат **disasm** који прима објектни фајл и исписује асемблерске наредбе које се налазе у објектном фајлу.
   * Може се покренути циљем “dsasm” или командном линијом. У командној линији је потребно покретнути клау disam.class.

**Napomena:**

Ukoliko koristite komandnu liniju I zelite da pokrenete jar fajl a ne klasu onda je potrebno da stavitr jar opciju virtuelnoj masini.

java -jar imejara.jar argumentijara

Уколико преводите вашу класу без build targeta potrebno je da dodate na class path sve potrebne jar-ove.

Prevođenje i pokretanje klase proba koja koristi neki jar (primer je uzet sa materijala sa predavanja)

$ javac -cp .:mj-runtime-1.1.jar proba.java

$ java -cp .:mj-runtime-1.1.jar proba tmp.obj

$ java -cp .:mj-runtime-1.1.jar rs.etf.pp1.mj.runtime.disasm tmp.obj

Da bismo izvršili tmp.obj program mikrojava interpreterom

(-debug da se prikaže u svakom koraku izvršena naredna i stanje estacka):

$ java -jar mj-runtime-1.1.jtime.Run -debug tmp.obj

## Кратак опис тестова

Тестови су подељени по фазама.

За фазу синтаксне анализе тестови садрже неколико примера неисправне синтаксе у којима је наш парсер успео да се опорави од грешке и наставио дање парсирање.

За фазу синтаксне анализе тестови садрже како исправне тако и несисптавне контексне услове.

За фазу генерисања кода тестови проверавају да ли конструкти које имамо у језику функционишу на начин који очекујемо.

## Кратак опис новоиведених класа

**• ErrorMessage.java**

* Помоћна класа која служи за испис грешака. У скучају да је нека грешка после које не можемо наставити даље извршавање позива се и System.exit() и тиме се завршава извршавање нашег програма.

**• SemanticPass.java**

* Класа која служи за семантичку проверу програма који се компајлира. Провера се врши обиласком апстрактног синтаксног стабла тако што смо проширили класу VisitorAdaptor.java и имплементирали методе за сваки чвор стабла који нам је потребан.

**• CodeGenerator.java**

* Класа која служи за генерисање кода програма који је прошао синтаксну и семантичку проверу. Генерисање кода се врши обиласком апстрактног синтаксног стабла тако што смо проширили класу VisitorAdaptor.java и имплементирали методе за сваки чвор стабла коко нам је потребан.

**• Compiler.java**

* Класа која је задужена за компајлирање програма. Прво проверава да ли је одговарајући број улазних параметара. Затим позива парсер како би извршио лексичку и синтаксну анализу нашег програма. Уколико је улазни програм прошао синтаксну анализу компајлер позива семантичку анализу помоћу класе SemanticPass.java. Ако је програм прошао семантичку анализу онда компајлер позива генерисање кода помоћу класе CodeGenerator.java. Приликом генерисања кода прави се фајл који садржи објектним код који ће се извршавати у виртуелној машини за Микро Јаву.