УНИВЕРЗИТЕТ У БЕОГРАДУ ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Програмски преводиоци 1

Предметни професор:

проф. др Драган Бојић

Предметни асистенти:

ас. мс Маја Вукасовић

ас. мс Михајло Огризовић

Студент:

Немања Митровић 0288/2018

Одсек:

Рачунарска техника и информатика

Кратак опис поставке задатка

Циљ пројектног задатка је реализација компајлера за програмски језик Микројаву. Компајлер омогућава превођење синтаксно и семантички исправних Микројава програма у Микројава бајткод који се извршава на виртуелној машини за Микројаву. Програмски преводилац за Микројаву има четири основне функционалности: лексичку анализу, синтаксну анализу, семантичку анализу и генерисање кода.

1. Лексички анализатор

Лексички анализатор треба да препознаје језичке лексеме и врати скуп токена издвојених из изворног кода, који се даље разматрају у оквиру синтаксне анализе. Лексички анализатор је имплементиран писањем .flex спецификације у фајлу *mjlexer.flex*. Спецификација .flex се трансформише у имплементацију лексера на програмском језику Јава коришћењем алата JFlex. Лексички анализатор пролази кроз текст улазног фајла и проналази токене на основу .flex спецификације и ти токени ће се користити касније у фази синтаксне анализе. У случају да неку секвенцу знакова са улаза не може да упари ни са једном лексичком структуром лсексички анализатор треба да пријави грешку.

1. Синтаксни анализатор

Синтаксни анализатор има задатак да утврди да ли издвојени токени из изворног кода програма могу да формирају граматички исправне сентенце. Синтаксна анализа се имплементира писањем ЛАЛР(1) граматике на основу спецификације језика и та граматика је написана у фајлу *mjparser.cup*. За имплементацију парсера користи се генератор синтаксних анализатора AST-CUP који представља локално развијено проширење алата CUP за рад са синтаксним стаблима. Синтаксни анализатор покушава да сваки токен из фазе лексичке анализе упари са неким делом граматике и тако редукује смене граматике, а у случају да то није могуће изврши опоравак од грешака за сигурне симболе: , и ; а у случају да није могућ опоравак да пријави грешку у фази синтаксне анализе. У случају успеха формира се апстрактно синтаксно стабло које се користи у наредној фази.

1. Семантички анализатор

Семантички анализатор се формира на основу апстрактног синтаксног стабла које је настало као резултат синтаксне анализе. Семантичка анализа се реализује имплементацијом метода visit() за посећивање чворова апстрактног синтаксног стабла у класи SemanticAnalyzer.java. У фази семантичке анализе врши се ажурирање табеле симбола и додавање нових симбола у табелу када се препозна симболичка константа, глобална променљива или локална променљива и врши се провера контекстних услова на основу којих се пријављује грешка у фази семантичке анализе. У случају успеха то значи да је улазни програм лексички, синтаксно и семантички исправан и спреман за фазу генерисања кода.

1. Генератор кода

Генератор кода преводи синтаксно и семантички исправне програме у извршни облик за извршно окружење Микројава виртуелне машине. У овој фази се такође врши имплементација метода visit() које служе за посећивање чворова апстрактног синтаксног стабла и то се ради у класи *CodeGenerator.java.* У тој класи се на многим местима користи класа *Code.class* која је доступна у библиотеци *mj-runtime-1.1.jar*.

Опис команди за генерисање кода алатима, превођење кода компајлером, покретање и тестирање решења

У оквиру пројекта постоји *build.xml* скрипта која служи за генерисање класа и која се састоји из неколико target-a.

* Таргет *delete* служи да приликом сваког новог извршавања ове скрипте да се генерисане класе из прошлог извршавања скрипте буду избрисане.
* Таргет *lexerGen* служи да се позове главна метода *JFlex.jar* библиотеке и уз то се користи опција да се преусмери где ће се сачувати изгенерисана класа *Yylex.java* и такође се прослеђује путања до спецификације *mjlexer.flex.*
* Таргет *parserGen* служи да се позове *cup\_v10k.jar.* Аргументом *destdir* наглашавам у који ће директоријум да се изгенеришу фајлови. Аргумент *-ast* служи да се наведе име пакета у коме ће бити изгенерисане класе које су потребне за изградњу синтаксног стабла. Аргументом *-buildtree* ће генерисани парсер имати могућност динамичког генерисања стабла. Након покретања овог таргета биће изгенерисане класе *MJParser.java* и *sym.java.*
* Таргетом *repackage* се постиже да се у класама које су потребне за изградњу синтаксног стабла да се све референце на пакет src.rs.ac.bg.etf.pp1.ast замене са rs.ac.bg.etf.pp1.ast.
* Таргетом *compile* се постиже да се за изгенерисане класе изврши њихово компајлирање.
* Таргетом *disasm* се постиже да се позове алат *disasm* из библиотеке *mj-runtime-1.1.jar* који омогућава да се изгенерисани објектни фајл преведе у текстуални облик који садржи мнемонике МикроЈава инструкција.
* Таргетом *runObj* се постиже да се покрене извршавање објектног фајла који је изгенерисан.

У класи *MJParserTest.java*  се врши покретање изгенерисаних класа. Прво се отвара фајл где је изворни код који треба да се преведе, затим се прави reader за тај фајл, затим се прави лексички анализатор коме се прослеђује тај reader и након тога се прави парсер коме се прослеђује лексички анализатор и започиње се парсирање. Затим се иницијализује табела симбола и покреће семантичка анализа. Након тога, у случају да се није десила синтаксна или семантичка грешка прелази се на фазу генерисања кода у којој ће се code buffer напунити програмским кодом и тај програмски код се преписује у излазни фајл.

Кад год се изврши измена везана за лексичку анализу или синтаксну анализу односно за фајлове mjlexer.flex или mjparser.cup потребно је покренути build.sh скрипту за таргет compile. Затим је потребно покренути МЈParserTest.java или у случају да се мења нешто везано за семантичку анализу или за генерисање кода односно за фајлове SemanticAnalyzer.java или CodeGenerator.java такође је потребно покренути МЈParserTest.java. На крају је потребно покренути build.sh скрипту за таргет compile за покретање извршавања објектног фајла.

Опис приложених тест примера

У свим примерима се за read() инструкцију користи као улаз фајл *input.txt.*

test301.mj – тест за тестирање функционалности за А ниво за испитни рок у јулу.

test301JanFeb.mj – тест за тестирање функционалности за А ниво за испитни рок у јануару и фебруару.

prvitest.mj – тест који се користи за тестирање неких основних функционалности, за убацивање препознатих симбола у табелу симбола, за тестирање пријављивања грешке у случају да се два пута декларише променљива са истим именом.

proba.mj – тест који служи за проверу свих смена написане граматике.

matrice.mj – тест који се користи за детаљно тестирање имплементације матрица.