Универзитет у Београду – Електротехнички факултет

**Програмски преводиоци 1 - 13С114ПП1**



Пројекат

**Компајлер за Микројаву**

Професор: **др. Драган Бојић**

Асистенти: **маст. инж. Маја Вукасовић**

**маст. инж. Михајло Огризовић**

фебруар 2023. године

**Кратак опис поставке задатка**

Циљ пројектног задатка је реализација компајлера за програмски језик Микројава. Сама спецификација језика Миркојава може се видети на предметној страници у одељку „Домаћи задатак“.

Компајлер пружа могућност превођења синтаксно и семантички исправних Микројава програма у **Микројава бајткода** који је могуће извршавати на виртуелној машини за Микројаву. Цела синтакса и семантика је описана у документу „mikrojava\_2022\_2023\_jan.pdf“ који се може наћи на страници предмета.

Постоје четири фазе приликом превођења програма: Лексичка анализа, Синтаксна анализа, Семантичка обрада и Генерисање кода.

**Лексичка анализа**

Лексички анализатор пролази кроз улазни фајл (програм) и препознаје токене у виду кључних речи као и преко регуларних израза. За сваки токен се проверава да ли је дефинисано правило. Уколико се пронађе одговарајући токен, лексички анализатор ће вратити симбол за тај токен. То се дешава све док се не стигне до краја фајла или док не настане нека лексичка грешка, тј лексер не успе да упари неки део из улазног фајла са својим изразима и кључним речима. Уколико се препозна све, прелази се на следећу фазу, синтаксну анализу.

Улазни фајл ове фазе је mjlexer.flex.

**Синтаксна анализа**

Синтаксни анализатор добија токене који су изгенерисани у лексичкој анализи и да провери да ли токени из изворног кода формирају исправне граматичке сентенце. Уколико се открије грешка, потребно је покушати извршење **опоравка од грешке до неког сигурног симбола**. Уколико то није могуће, не наставља се на фазу семантичке обраде, јер постоји грешка у граматици програма. Уколико се не детектује ни једна грешка, генерише се апстрактно синтаксно стабло.

**Семантичка обрада**

Претходно генерисано апстрактно синтаксно стабло се анализира и детаљно проверава. Обрада се извршава писањем „визиторских“ метода за конкретне чворове у стаблу и приликом обиласка се проверава да ли је неко од семантичких правила датих у спецификацији Микројаве нарушено и није испоштовано. Уколико се утврди да је било неких неправилности, не наставља се са генерисањем кода.

**Генератор кода**

Генерисање кода служи да се, поново писањем виситорских метода, генерише одговарајући низ инструкција тако да се резултат може покретати на виртуелној машини Микројаве и да осликава написани код који је био анализиран у претходним фазама.

Коришћени алати и фајлови:

* **JFlex.jar** – коришћено приликом лексичке анализе
* **cup\_v10k.jar –** коришћено приликом генерисања апстрактног синтаксног стабла
* **symboltable-1-1.jar –** коришћено приликом генерисања АСТ
* **mj-runtime-1.1.jar –** коришћено приликом „распакивања“ објектног фајла

**Команде за покретање и превођење**

Приликом покретања је потребно искористити **build.xml** фајл који у себи садржи правила за покретање, као и међусобне зависности правила како би се покретање извршило како треба. Приликом измене у првој или другој фази, потребно је покренути правило **compile**. Након тога, покретањем класе Compiler, која ће касније бити објашњена, генерише се објектни фајл који је потребан. Уколико желимо да га извршимо, покрећемо правило **runObj,** које ће нам приказати који су излази за одређене улазе и сам производ ивршавања фајла. Могуће је поставити **-debug** опцију која ће нам приказати стање стека који се користи за складиштење операнада и резултата инструкција после сваке наведене инструкције, као и њихове адресе и операнде које користе.

**Кратак опис приложених тест примера**

* **test301.mj** – Тест пример који је дат као пример уз поставку домаћег задатка за јануарски и фебруарски рок 2023. године
* **test302.mj** – Тест пример који је дат као пример уз поставку домаћег задатка за јануарски и фебруарски рок 2023. године
* **test303.mj** – Тест пример који је дат као пример уз поставку домаћег задатка за јануарски и фебруарски рок 2023. године
* **testAleksandar.mj** – Тест пример који детаљно испитује све функционалности које треба подржати за почетни (А) ниво пројекта. (Велико хвала Александру Ивановићу!)
* **prilog2.mj** – Тест пример који је дат уз поставку самог домаћег задатка, као прилог уз домаћи уз који је приложен и очекивани излаз.
* **program.mj** – садржи неки од наведених тестова који се тренутно тестира (због једноставности покретања)
* **programExampleMJ.mj** – Тест пример приказан у документу спецификације језика Микројава
* **myProgramFromStart.mj** – Тест пример приказан у документу спецификације језика Микројава исправљен јер садржи семантичке грешке које нису у складу са спецификацијом
* **programAtStart.mj** - Тест пример тестира једноставне доделе, функције и испис, коришћен при почетним фазама пројекта, више се није користио

**Кратак опис новоуведених класа**

**Пакет test.rs.ac.bg.etf.pp1**

* **MJTest** – Класа која тестира лексички анализатор и исписује препознате токене.
* **MJParserTest** – Класа која тестира лексички анализатор и синтаксни анализатор, проверава да ли је дошло до грешке током анализе и исписује грешку уколико јесте. Покреће program.mj
* **Compiler** – Покреће program.mj (први аргумент при позиву) и ради све четири фазе пројекта. На крају генерише program.obj фајл (други аргумент при позиву)
* **RuleVisitor** - \*не користи се\* приликом обиласка стабла бележи уколико се десио позив исписивања или декларисања променљиве. То се сада ради у класи **SemanticAnalyzer**.
* **SemanticAnalyzer** – Класа која садржи имплементације визиторских метода за АСТ стабло програма који се проверава и броји велики број позива и декларисања: променљиве, константе, исписа, коришћења променљиве, коришћења константи...
* **CodeGenerator** – Врши се обилазак стабла програма који се проверава и ради се смештање инструкција у бафер који ће представљати стек помоћу којег се генерише .obj фајл.
* **CounterVisitor** – Помоћна класа која има бројач формалних параметара и декларисања променљивих.