**Електротехнички факултет - Универзитет у Београду**

**Август 2022. године**



**Програмски преводиоци 1**

**Професор: Драган Бојић**

**Студент: Лазар Вулић 2018/0182**

**Одсек: Рачунарска техника и информатика**

# Кратак опис пројектног задатка

Циљ овог пројектног задатка била је израда једноставног **MJCompiler**-a за програмски језик **MicroJava**. Постоје 4 основне функционалности*: Лексички анализатор, Синтаксна анализа, Семантичка анализа и Генерисање кода*. Циљ је задати лексичко, синтаксно и семантички исправан програм, који ће се превести **у MicroJava бајткод**, који ће се затим извршити на **виртуалној машини** са сајта предмета. Синтакса и семантика које су коришћене при изради овог пројекта су са сајта предмета. Коришћени алат је алат **JFlex** са сајта предмета.

# 1.фаза – Лексичка анализа

Лексички анализатор пролази кроз улазни фајл који представља програм који се симулира, и препознаје токене(лексеме) у виду кључних речи наведених у фајлу: **mjlexer.flex**. За сваки токен покушава се пронаћи његов парњак, и уколико лексичка анализа прође без грешке прелази се на наредну фазу, тј. Синтаксну анализу. Уколико пак лексички анализатор пронађе токен који не одговара ниједној унапред предефинисаној језичкој лексеми, јавља се одговарајућа порука о грешци.

# 2. фаза – Синтаксна анализа

Циљ синтаксног анализатора је да провери да ли написани програм одговара граматици написаној у фајлу: **mjparser.cup**. За сваки токен који је препознат у првој фази – Лексичкој анализи, покушава се његово упаривање са одговарајућим деловима граматике, како би се одлучило о томе да ли скуп препознатих језичких лексема може да формира исправну граматику. Уколико се у другој фази – Синтаксној анализи препозна грешка, потребно је извршити опоравак од грешке до тзв. *Сигурног симбола, тј. До ; и ,* .Уколико се у другој фази, Синтаксној анализи, не детектује ниједна грешка, значи да се ради о лексички и синтаксно исправном програму. Одговарајућу поруку о синтаксно исправном програму треба приказати на излазу. Као резултат друге фазе, Синтаксне анализе, генерише се синтаксно стабло, које прати ток упаривања језичких лексема са одговарајућим деловима граматике.

# 3.фаза – Семантичка анализа

Циљ ове фазе је исправан обилазак претходно генерисаног синтаксног стабла са циљем семантичке провере исправности задатог улазног програма. У овој фази се пишу *visit(…) методе* из претходно добијеног *ast* подпакета пакета *src* унутар класе **SemanticAnalyzer.java**. Семантичка правила која се проверавају у овој фази су задата у поставци пројектног задатка, у одељку Контекстни услови. Ово је уједно и фаза у којој се ради и попуњавање табеле симбола на исправан начин. У имплементацији табеле симбола са сајта предмета *недостаје Bool тип*, па ће сходно томе он бити и додат по узору на остале примитивне типове који су нам унапред доступни.

# 4. фаза – Генерисање кода

Циљ је задати програм, за кога се сада подразумева да је лексички, синтаксно и семантички исправан, превести у облик спреман за извршавање на виртуалној машини са сајта предмета. У овој фази такође се пишу одговарајуће и релевантне *visit(…) методе* унутар класе **CodeGenerator.java**. Класа која се користи у овој фази је класа **Code.class** која нам је била на располагању из библиотека са сајта предмета.

# Команде за покретање и превођење

Након сваке направљене измене у некој од прве 2 фазе, Лексичком и Синтаксном анализатору, потребно је покренути **build.xml:compile**, који у позадини позива низ команди, најбитније и лехер и парсер. На основу тог позива, добијамо ажуриране класе унутар подпакета *ast* пакета *src*. Лехер је тај који нам производи **класу Yylex.java**, након чега парсер ради на формирању класа унутар горе споменутог пакета. Како би добили извршни фајл нашег програма потребно је покренути класу **Compile.java** из *пакета tests/rs.ac.bg.etf.pp1/* nakon чега се може покренути **build.xml:runbObjFile** што ће симулирати извршавање нашег програма на виртуалној машини. Унутар *фолдера lib/* налазе се одговарајуће библиотеке које се користе у низу таргета **у build.xml**, услед чега смо у таргету compile из build.xml и навели потребне библиотеке као *pathelement*.

# Опис класа у пројекту

* **МЈТеst.java** – из *фолдера test/rs.ac.bg.etf.pp1/* - класа којом се покреће синтаксна анализа наведеног програма.
* **Compiler.java** - из *фолдера test/rs.ac.bg.etf.pp1/* - класа која покреће све 4 фазе пројекта једну за другом и у случају успеха враћа објектни фајл program.obj за коришћени тест који се налази у фајлу **program.mj**. Добијени објектни фајл спреман је да се провуче кроз коришћену виртуалну машину. Унутар ове класе се уједно ради и проширивање табеле симбола за објектни чвор bool типа који није био имплементиран у добијеној имплементацији табеле симбола. Левел и адр поље тог објектног чвора, постављено је на вредност 1 по узору на постојеће објектне чворове уграђених примитибних типова. Обилазак стабла се ради по *принципу одоздо нагоре, с лева на десно*. У овој класи је уједно и испис табеле симбола помоћу предефинисане методе dump, као и инизијализација универзалног опсега на нивоу -1. Пребројавање глобалних и локалних симбола рађено је у самој 3.ој фази пројекта. Број глобалних променљивих се поставља као вредност **за Code.dataSize** управо унутар ове Compile.java klase.

# Опис тестова у пројекту

Тестови који су коришћени:

* **test301A.mj** – тест који је тестирао функционалности за А ниво у јануару и фебруару. Добијен је од стране предметног професора/асистената.
* **test301A\_avgust.mj** – тест који тестира функционалности за А ниво у текућем, августовском року. Добијен од стране предметног професора/асистената.
* **test301.mj** – тест који је вероватно тест за А ниво од прошле године. Добијен од стране предметног професора/асистената.
* **testic.mj** – једноставан тест који је само служио за проверу попуњавања табеле симбола и броја локалних променљивих. У њему тестирана само једна додела, и креирање низа где је као број елемената низа стављен исправан целобројни тип и неисправан тип за величину низа.
* **testic1.mj** – мој једноставан тест, који ради са низовима, наредбама за испис и читање из улазног фајла ulaz.txt из фолдера test/.
* **mojTest.mj** – тест који ради малтене исто као и претходни тест, при чему линија 5 када се откоментарише треба да нам сугерише како је рађен опоравак од грешке до ; у линији 5. Такође, уколико се откоментарише линија 6, сугерише нам како је рађен опоравак од грешке до , у линији 6. Уколико се откоментарише линија 11, сугерише нам како је рађен опоравак од грешке до ; при додели вредности.
* **test.mj** – тест који тестира малтене све граматичке смене. Нешто је остало закоментарисано, нешто закоментарисано. Буквално сам ишао редом и сваку могућу грешку за синтаксне услове проверио да ли се хвата и да ли је опис грешке добар. Такође, у испису једна грешка може проузроковати још грешака, при чему се све грешке исписују(нпр. Сабирање не инт вредности које представља noType, а затим додела тог типа у променљиву другог типа, дакле 2 грешке).
* **test2.mj** – малтене исти тест као и тест описан изнад, само са додатним проверама око лабела. Проверава шта се дешава уколико постоји вишеструка дефиниција лабеле, уколико се скаче на недифинисану лабелу. Такође провераба вишеструки узастопни скок на исту лабелу што би логички било сасвим у реду.

*Напомена 1*: у свим тестовима се исписује информација о приступу елементу низа, информација о наиласку на локални/глобални симбол, као и њихов објектни чвор.

*Напомена 2*: фајл са именом ulaz.txt користи се као фајл из кога се чита се параметар потребан за read(…) statement. У њему се мора навести најмање онолико параметара колико постоји тих наредби у улазном тесту из фајла program.mj. Буквално улаз са тастатуре преусмерен на тај улазни фајл.