**Elektrotehnički fakultet**

**Univerzitet u Begoradu**



Projekat iz predmeta Programski Prevodioci

**Kompajler za Mikrojavu**

Marta Avramović 0591/17

jun 2021.

Postavka zadatka

Cilj projektnog zadatka je realizacija kompajlera za programski jezik Mikrojavu. Kompajler omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvršava na virtuelnoj mašini za Mikrojavu. Sintaksno i semantički ispravni Mikrojava programi su definisani specifikacijom [MJ].

Programski prevodilac za Mikrojavu ima četiri osnovne funkcionalnosti: leksičku analizu, sintaksnu analizu, semantičku analizu i generisanje koda.

Leksički analizator treba da prepoznaje jezičke lekseme i vrati skup tokena izdvojenih iz izvornog koda, koji se dalje razmatraju u okviru sintaksne analize. Ukoliko se tokom leksičke analize detektuje leksička greška, potrebno je ispisati odgovarajuću poruku na izlaz.

Sintaksni analizator ima zadatak da utvrdi da li izdvojeni tokeni iz izvornog koda programa mogu formiraju gramatički ispravne sentence. Tokom parsiranja Mikrojava programa potrebno je na odgovarajući način omogućiti i praćenje samog procesa parsiranja na način koji će biti u nastavku dokumenta detaljno opisan. Nakon parsiranja sintaksno ispravnih Mikrojava programa potrebno je obavestiti korisnika o uspešnosti parsiranja. Ukoliko izvorni kod ima sintaksne greške, potrebno je izdati adekvatno objašnjenje o detektovanoj sintaksnoj grešci, izvršiti oporavak i nastaviti parsiranje.

Semantički analizator se formira na osnovu apstraktnog sintaksnog stabla koje je nastalo kao rezultat sintaksne analize. Semantička analiza se sprovodi implementacijom metoda za posećivanje čvorova apstraktnog sintaksnog stabla. Stablo je formirano na osnovu gramatike implementirane u prethodnoj fazi. Ukoliko izvorni kod ima semantičke greške, potrebno je prikazati adekvatnu poruku o detektovanoj semantičkoj grešci.

Generator koda prevodi sintaksno i semantički ispravne programe u izvršni oblik za odabrano izvršno okruženje Mikrojava VM. Generisanje koda se implementira na sličan način kao i semantička analiza, implementacijom metoda koje posećuju čvorove.

Projekat je odrađen za osnovni A nivo.

Pokretanje Koda

Projekat je razvijen i podešen za izvršavanje u Eclipse razvojnom okruženju.

Postupak pokretanja:

1. Program koji se prevodi nalazi se u fajlu program.mj u okviru foldera test

2. Najpre je potrbno pokrenuti kompajler kao java aplikaciju. Naziv kompajlera: MJTestCompile.java

3. Kompajler ce izgeneristai program.obj fajl koji se moze pokrenuti putem build.xml fajla u okviru targeta RunObj.

4. Postoji i opcija debugovaja obejktonog fajla i ona se pokrece puteam targeta debugObj.

Buid.xlm

<?xml version="1.0" encoding="UTF-8"?>

<project name="MJCompiler" default="compile" basedir=".">

<target name="delete">

<delete>

<fileset dir="src/rs/ac/bg/etf/pp1">

<exclude name="util/Log4JUtils.java"/>

<exclude name="test/Compiler.java"/>

<exclude name="test/CompilerError.java"/>

<exclude name="RuleVisitor.java"/>

<exclude name="SemanticAnalyzer.java"/>

<exclude name="NewSymbolTable.java"/>

<exclude name="CodeGenerator.java"/>

<exclude name="MJTestCompile.java"/>

</fileset>

</delete>

<delete>

<fileset dir="spec">

<exclude name="mjlexer.lex"/>

<exclude name="mjparser.cup"/>

</fileset>

</delete>

</target>

<target name="lexerGen" depends="delete">

<java jar="lib/JFlex.jar" fork="true">

<arg value="-d"/>

<arg value="./src/rs/ac/bg/etf/pp1"/>

<arg value="spec/mjlexer.lex"/>

</java>

</target>

<target name="parserGen" depends="lexerGen">

<java jar="lib/cup\_v10k.jar" fork="true">

<arg value="-destdir"/>

<arg value="src/rs/ac/bg/etf/pp1"/>

<arg value="-ast"/>

<arg value="src.rs.ac.bg.etf.pp1.ast"/>

<arg value="-parser"/>

<arg value="MJParser"/>

<arg value="-dump\_states"/>

<arg value="-buildtree"/>

<arg value="spec/mjparser.cup"/>

</java>

</target>

<target name="repackage" depends="parserGen">

<!-- Replaces all of the references to the old package name in files in the "src" directory -->

<replace dir="src" value="rs.ac.bg.etf.pp1.ast" token="src.rs.ac.bg.etf.pp1.ast" summary="true"/>

</target>

<target name="compile" depends="repackage">

<javac srcdir="src/rs/ac/bg/etf/pp1" includeantruntime="false">

<classpath>

<pathelement path="lib/JFlex.jar"/>

<pathelement path="lib/cup\_v10k.jar"/>

<pathelement path="lib/log4j-1.2.17.jar"/>

<pathelement path="lib/symboltable-1-1.jar"/>

<pathelement path="lib/mj-runtime-1.1.jar"/>

</classpath>

</javac>

</target>

<target name="disasm" >

<java classname="rs.etf.pp1.mj.runtime.disasm">

<arg value="test/program.obj"/>

<classpath>

<pathelement location="lib/mj-runtime-1.1.jar"/>

</classpath>

</java>

</target>

<target name="runObj" depends="disasm" >

<java classname="rs.etf.pp1.mj.runtime.Run">

<arg value="test/program.obj"/>

<classpath>

<pathelement location="lib/mj-runtime-1.1.jar"/>

</classpath>

</java>

</target>

<target name="debugObj" depends="disasm" >

<java classname="rs.etf.pp1.mj.runtime.Run">

<arg value="test/program.obj"/>

<arg value="-debug"/>

<classpath>

<pathelement location="lib/mj-runtime-1.1.jar"/>

</classpath>

</java>

</target></project>

Test primeri

Greske:

// Test301

program test301

const int nula = 0;

const int jedan = 1;

const int pet = 5 a;

int niz[];

\_ nizch[];

{

void main()

char bodovi;

{

bodovi = 0;

bodovi++;

bodovi = bodovi + jedan;

bodovi = bodovi \* pet;

bodovi--;

print(bodovi);

niz = new int[3];

niz[nula] = jedan;

niz[1] = 2;

niz[niz[jedan]] = niz[niz[0]] \* 3;

bodovi = niz[2]/niz[0];

print(bodovi);

print(niz[2]);

nizch = new char[3];

nizch[0] = 'a';

nizch[jedan] = 'b';

nizch[pet - 3] = 'c';

print(nizch[1]);

print(nizch[jedan \* 2]);

bodovi = bodovi + (pet \* jedan - 1) \* bodovi - (3 % 2 + 3 \* 2 - 3);

print(bodovi);

read(bodovi);

if(bodovi%2 == 0) bodovi = bodovi + 10;

if(false) print(bodovi + 20);

else print(bodovi);

}

}

Bez gresaka:

// Test301

program test301

const int nula = 0;

const int jedan = 1;

const int pet = 5;

int niz[];

char nizch[];

{

void main()

int bodovi;

{

bodovi = 0;

bodovi++;

bodovi = bodovi + jedan;

bodovi = bodovi \* pet;

bodovi--;

print(bodovi);

niz = new int[3];

niz[nula] = jedan;

niz[1] = 2;

niz[niz[jedan]] = niz[niz[0]] \* 3;

bodovi = niz[2]/niz[0];

print(bodovi);

print(niz[2]);

nizch = new char[3];

nizch[0] = 'a';

nizch[jedan] = 'b';

nizch[pet - 3] = 'c';

print(nizch[1]);

print(nizch[jedan \* 2]);

bodovi = bodovi + (pet \* jedan - 1) \* bodovi - (3 % 2 + 3 \* 2 - 3);

print(bodovi);

read(bodovi);

if(bodovi%2 == 0) bodovi = bodovi + 10;

if(false) print(bodovi + 20);

else print(bodovi);

}

}

Opis Impementiranih klasa

1. NewSymbolTable.java – Posto u okviru date Symbol tabele nije postojao bool tip, ova klasa je samo extendovala klasu Tab i implementirala tip koji je nedostajao.

2. SemanticAnalayzer.java – Klasa koja je mozak semantičke analize. Ima implementiranu celokupnu logiku, detektovanje i obradu gresaka koji se mogu pojaviti u Sematničkoj analaizi.

3.CodeGenerator.java - Klasa koja implemnetira generisanje koda. Vrlo je slična klasi semantičke analize, jer svoju funkcionalnost ostavruje kroz visitor metode i generise kod.

4.MJTestCompile.java – Kompajlerksa klasa, pokrece sve faze prilikom izvrsavanja i generise program.obj fajl. Takodje je zaduzena i za ispis gresaka, printovanje rezultata svih faza.