**Mikrojava kompajler**

Programski prevodioci 1  
Februar 2023.

Profesor: Dragan Bojić

Asistenti: Maja Vukasović i Mihajlo Ogrizović

Student: Maja Nišević 0340/2018

**OPIS PROJEKTNOG ZADATKA**

Projektni zadatak predstavlja realizaciju kompajlera za programski jezik Mikrojava. Postoje četiri jasno definisane faze rada kompajlera, navedene redosledom izvršavanja:

1. leksička analiza
2. sintaksna analiza
3. semantička analiza
4. generisanje koda

Prva faza, faza leksičke analize, korišćenjem JFlex alata, prepoznaje lekseme ulaznog programa i za svaki vraća token.

Sintaksna analiza koristi alat CUP da generiše LALR(1) parser baziran na gramatici koju sastavljamo. Proizvod sintaksne analize jeste apstraktno sintaksno stablo. Ukoliko se neka instrukcija ne poklapa sa ovim sintaksnim stablom, pokušava se oporavak od greške da bi parser nastavio rad i prijavio svaku grešku na koju naiđe.

Semantička analiza koristi gorenavedeno sintaksno stablo da utvrdi da li su te sintaksno ispravne smene i semantički ispravne, što radi obilaskom čvorova našeg stabla redefinisanim visit metodama. Takođe ima cilj da popuni tabelu simbola i proveri da li postoje problemi višestruke definicije.

Četvrta faza, generisanje koda, takođe obilazi čvorove sada i sintaksno i semantički ispravnog stabla. Rezultat je izvšni kod.

**KOMANDE**

Naš build.xml fajl ima nekoliko različitih definisanih targeta za rad našeg kompajlera, odvojenih po fazama.

1. Za leksičku analizu koristimo target LexerGen. LexerGen ima cilj da korišćenjem našeg lexer.flex fajla, kojim smo definisali lekseme, generiše uz pomoć Flex alata klasu Yylex.java. Yylex je zapravo naš lekser. Ovde koristimo jar fajl Jflex.jar.
2. U slučaju da se LexerGen ispravno izvrši, sledeći target je ParserGen. On generiše naš parser korišćenjem cup\_v10k.jar jar fajla. Kako se ovim naše nove klase ne nalaze u ispravnim folderima, pozivamo repackage target.
3. Compile nam na osnovu ovih targeta pravi izvršni fajl, koristeći dodatno i jar fajl symboltable-1-1.jar za semantičku analizu.
4. Za pokretanje programa imamo targete runObj, disasm, i debugObj. debugObj nam dodatno ispisuje stanja steka prilikom svake instrukcije, dok disasm ispisuje instrukcije redom. Ovde se koristi mj-runtime-1.1.jar. Dodatno, ovi targeti su parametrizovani tako da se mogu dodati njihov argument file, kome je default vrednost postavljena na test/test301.mj, odnosno javni test za A nivo.

Imena naših .mj i .obj fajlova se dodaju kao argumenti komandne linije koje obrađuje klasa Compiler.java.

**TESTOVI**

Folder test sadrži .mj i .obj fajlove koji nam služe za testiranje projekta. Oni su:

1. SyntaxErrorTest.mj koji predstavlja prikaz oporavka od greške u svim situacijama navedenim za nivo A
2. SyntaxCorrectTest.mj koji ima ispravnu sintaksu ali ne i ispravnu semantiku, pošto neke od smena nisu implementirane za A nivo
3. SemanticErrorTest.mj, odnosno prikaz obrade semantičkih grešaka
4. SemanticCorrectTest.mj, test sa ispravnom sintaksom i semantikom
5. Test301.mj, javni test za A nivo
6. Program.mj, pomoćni test

Takođe sadrži readInput.txt koji koristimo za read instrukciju Mikrojave umesto stdin (kada se pokreće iz build.xml a ne iz Run.class), kao i .obj fajlove za testove koji mogu da se generišu.

**NOVE KLASE U IZRADI PROJEKTA**

**SemanticAnalyzer.java:** Ima cilj da obilaskom čvorova implementira sva semantička pravila, i obrađuje greške u slučaju da na njih naiđe. Takođe popunjava tabelu simbola.

**CodeGenerator.java:** Generiše kod koji je semantički i sintaksno ispravan. Obilaskom stabla korišćenjem redefinisanih metoda gura u naš kod-bafer odgovarajuće instrukcije, služeći se klasom Code.

**Compiler.java:** Naša ulazna tačka, klasa sa main metodom kojoj dajemo argumente komandne linije, koji je upućuju na .mj i .obj fajlove. Prvo se odavde poziva sintaksna i semantička analiza, a onda u slučaju da je ispravno (errorDetected je false) poziva CodeGenerator.

**MyTab.java**: Klasa izvedena iz Tab.java. Glavna promena joj je uvođenje bool tipa u tabelu simbola, i samim tim redefinicija metoda init() i dump(). Redefiniše dump() jer nam je potreban obilazak tabele koji prepoznaje bool tip, zato koristimo MyDumpSymbolTableVisitor.

**MyDumpSymbolTableVisitor.java:** Izvedena iz DumpSymbolTableVisitor zarad ispravnog ispisivanja i obilaska bool tipa (videti MyTab.java).