Elektrotehnički fakultet u Beogradu

Katedra za računarsku tehniku i informatiku

Projekat iz predmeta Programski prevodioci 1

* **Kompajler za Mikrojavu** -

Student: Jovana Janković 0586/17 , Smer: SI1. Kratak opis projektnog zadatka

Cilj projektnog zadatka je realizacija kompajlera za programski jezik Mikrojava. Jezik Mikrojava predstavlja jezik sličan Javi, ali je dosta pojednostavljen.

Mikrojava program, prilikom kompajliranja, prolazi kroz sledeće faze u okviru datog projekta:

**1)** Leksička analiza – prepoznaje jezičke lekseme iz programa i vraća tokene koji se koriste dalje u sintaksnoj analizi. Ukoliko je lekser detektovao grešku u programu, on će je prijaviti samom korisniku.

**2)** Sintaksna analiza – prilikom sintaksne analize koristi se gramatika koja je realizovana u okviru projektnog zadatka. Cilj sintaksne analize je da utvrdi da li tokeni dobijeni od leksičkog analizatora mogu da formiraju gramatički ispravne sekvence. Ukoliko postoji greška prilikom parsiranja, parser će se za odredjene greške oporaviti, a za neke neće. Cilj svakog programa je da dobije pozitivan odgovor od parsera, ali ukoliko to ne bude slucaj, parser će prikazati gde postoji greška u programu.

**3)** Semantička analiza – ova faza prevodjenja ima cilj da utvrdi da li su ispoštavani svi kontekstni uslovi dati u specifikaciji Mikrojava jezika. Semantička analiza je realizovana obilaženjem čvorova stabla koji su dobijeni sintaksnom analizom. Ukoliko je program semantički ispravan, prelazi se na sledeću fazu, a ukoliko nije, potrebno je da korisnik popravi delove programa u kojima postoji greška, a o kojima će nas obavestiti semantički analizator.

**4)** Generisanje koda – Do ove faze se stiglo ukoliko je program i sintaksno I semantički ispravan. Ova faza je takodje realizovana obilaskom stabla koje je nastalo kao posledica sintaksne analize. Generator koda pretvara sintaksno i semantičko ispravne programe u Mikrojava bajtkod koji će se izvršavati na virtuelnoj mašini za Mikrojavu.

2. Opis korišćenih alata i biblioteka

- U leksičkoj analizi korišćen je alat – **JFlex**

- U sintaksnoj analizi korišćen je – **AST-CUP** generator, korišćena je biblioteka **cup\_v10k.jar**.

- U semantičkoj analizi korišćena je biblioteka – **symboltable-1-1.jar**

- U generisanju korišćena je biblioteka – **mj-runtime-1.1.jar**

3. Opis komandi korišćenih u projektu

**1)**Leksička analiza – U ovoj fazi se kreira klasa **MJlexer.java**. Naziv te klase je specificiran u fajlu **mjlexer.flex** naredbom **%class MJLexer.** Kako bi se kreirala klasa MJlexer.java od fajla mjlexer.flex, treba pokrenuti main metodu klase **JFlex.Main**. U argumentima konfiguracije **LexerGenerator** u argumentima programa neophodno je da stoji sledeća direktiva:

**-d src\rs\ac\bg\etf\pp1 spec\mjlexer.flex**

**2)**Sintaksna analiza – U ovoj fazi se kreira klasa **MJParser.java** kao i apstraktno sintaksno stablo(**AST**). Kako bi se kreirala klasa MJParser.java i sintaksno stablo, treba pokrenuti main metodu klase **java\_cup.Main.** U argumentima konfiguracije **ParserGenerator** u argumentima programa neophodno je da stoji sledeca direktiva:

**-destdir rs\ac\bg\etf\pp1 -dump\_states -parser MJParser -ast rs.ac.bg.etf.pp1.ast -buildtree ..\spec\mjparser.cup**

**3)**Semantička analiza- Ova faza se izvršava pozivom main metode klase **Compiler.java**. U konfiguraciji **Compiler/Compile MJSource** u argumentima programa postoje dva argumenta, gde prvi argument predstavlja program napisan u Mikrojava programu(.mj), a drugi predstavlja objektni fajl(.obj). Primer argumenata:

**test/program.mj test/program.obj**

**4)**Generisanje koda – Nakon što se završila semantička analiza, nastao je objektni fajl Mikrojava programa u kom se nalazi bajtkod. On je spreman za izvršavanje na Mikrojava virtuelnoj mašini. To je moguće uraditi pokretanjem konfiguracije **Run,** koja poziva main metodu **rs.etf.pp1.mj.runtime.Run.** Argumenti programa u konfiguraciji su sledeće **-debug test\program.obj**

Program.obj predstavlja objektni fajl koji će izvrsiti Mikrojava virtuelna masina. Prilikom ovog pokretanja, korisniku se prikazuje izgled steka, instrukcije i vrednosti adresa na kojima se nalaze te instrukcije.

Moguće je i pokrenuti izvršavanje programa kroz konfiguraciju **Disasm**. Ova konfiguracija zove main metodu klase **rs.etf.pp1.mj.runtime.disasm.** Tada postoji jedan argument, a to je putanja do objektnog fajla - **test\program.obj.**

Prilikom ovog pokretanja vide se samo instrukcije i adrese na kojima se instrukcije nalaze.

4. Opis test primera

Prilikom izrade programa, korišćeni su testovi za proveru ispravnosti koda, koji su radjeni za svaku fazu projekta. Oni se sastoje od ispravnih programa, ali i neispravnih, kako bi se utvrdilo da li su obradjene sve situacije napisane u Mikrojava specifikaciji jezika.

Na ovaj način, detaljno su testirane sve 4 faze. Test sa nazivom “test301.mj”, predstavlja relevantan test(javni test) kojim se testiraju sve funkcionalnosti programa trazeni za ovaj nivo projektnog zadatka.

“**neispravni\_test\_primeri\_sintaksna\_analiza.mj**” i “**neispravan\_test\_sintaksa.mj**” sadrži neispravan kod kojim su se najpre testirali neispravni delovi koda kako bi se utvrdilo da li su sve situacije obradjene na adekvatan nacin.

“**test\_semantika.mj**” I “**test\_semantika\_2.mj**” sadrže relevantne test primere za semantičku analizu.

“**test\_generisanje\_koda.mj**” i “**test\_ternarni.mj**” sadrže razne primere kojima je testirana faza generisanja koda. U ovim testovima se nalaze ispravni kodovi koji sadrže veliki broj različitih situacija i primera.

5. Kratak opis novouvedenih klasa

**1)**SemanticAnalyzer.java – klasa u kojoj je napisana cela implementacija semantičke analize. Obuhvata visit metode cvorova stabla čijim obilaskom se utvrdjuje semantička ispravnost programa.

**2)**CodeGenerator.java – klasa u kojoj je napisana celokupna implementacija za generisanja koda. Obuhvata visit metode čvorova stabla. U visit metodama zapravo dolazi do generisanja koda.

**3)**Compiler.java – klasa čijim pokretanjem zapravo vršimo leksičku, sintaksnu, semantičku analizu i generisanje koda. Ona sadrži main metodu i ulazne tačke programa. Kao prvi argument komandne linije dostavlja se naziv Mikrojava programa, a kao drugi argument naziv objektnog fajla.

**4)**CounterVisitor.java – pomoćna klasa koju koristi klasa CodeGenerator. U njoj je implementiran deo koji računa broj formalnih parametara funkcije i lokalnih promenljivih funkcije, kako bi te podatke iskoristila instrukcija enter.

**5)**MyVisitor.java – pomoćna klasa za ispis tabele simbola, dodata je zbog proširenja radi ispisa bool tipa!