UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

PROGRAMSKI PREVODIOCI 1 (13E114PP1)

DOMAĆI ZADATAK – MICROJAVA COMPILER

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:

prof. dr Dragan Bojić

as. ms Maja Vukasović

as. ms Mihajlo Ogrizović

Student:

Aleksandar Ivanović 2013/0010

Beograd, januar 2023.

SADRŽAJ

1. 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.

2. OPIS KOMANDI ZA GENERISANJE JAVA KODA ALATIMA, PREVOĐENJE KODA KOMPAJLEROM, POKRETANJE I TESTIRANJE REŠENJA

2.1. Generisanje java koda alatima

Generisanje leksera se pokreće korišćenjem lexerGen ant targeta ili preko komandne linije java -jar lib\JFlex.jar -d src/rs/ac/bg/etf/pp1 spec\mjflexer.flex

Generisanje parsera se pokreće korišćenjem parserGen ant targeta ili preko komandne linije java -jar lib\cup_v10k.jar -expect 0 -destdir src/main/java/rs/ac/bg/etf/pp1 -parser MJParser -dump_states -buildtree spec\mjparser.cup

2.2. Prevođenje koda kompajlerom

Prevođenje koda kompajlerom se vrši pokretanjem main metode klase MJCompiler sa 2 ulazna argumenta koji predstavljaju putanju do izvršnog koda MJ programa i putanju do objektnog fajla koji treba biti generisan, na primer: src/test/resources/test_semantic_analysis.mj src/test/resources/foobar.obj

2.3. Pokretanje

Definisana su dva ant targeta run i runDebug koji pokreću kod u release odnosno debug režimu pokretanjem klase Run biblioteke lib\mj-runtime-1.1.jar and objektnim fajlom koji se prosledi kao argument komandne linije.

2.4. Testiranje rešenja

Definisane su klase MJLexerTest i MJParserTest za potrebe testiranja.

3. OPIS PRILOŽENIH TEST PRIMERA

Svi testovi imaju objašnjenje šta je očekivan ulaz i izlaz unutar samih testova.

3.1. test_A_successful.mj

Testira funkcionalnosti predviđene za nivo A. Input/output sa standardnog ulaza izlaza. Inkrementiranje I dekrementiranje designatora. Operator za unpack niza.

3.2. test_error_recovery_A.mj

Testira oporavak od greške pri definisanju globalne proomenljive i pri operatoru dodele.

3.3. Test_semantic_analysis.mj

Testira semantičku analizu.

4. OPIS NOVOUVEDENIH KLASA

4.1. ActualParametersStack

Implementacija steka liste promenljivih koja se koristi za čuvanje parametara sa kojima se pozivaju funkcije. Potrebno je držati stek lista promenljivih jer parametar funkcije može biti rezultat funkcije. Na primer: result = f(f(4, 5), f(5, 4));

4.2. BranchJumpAddressStack

Implementacija steka liste adresa korišćenih za skokove pri uslovnom grananju u kodu (if, else).

4.3. Disassemble

Kopija disasm klase iz mj-runtime-1.1.jar korišćena za potrebe lakšeg testiranja kako bi se u isto vreme i generisao objektni kod i ispisale njegove instrukcije.

4.4. ErrorMessageGenerator

Pomoćna klasa koja apstrahuje generisanje poruka koje ispisuju ostale klase poput SemanticAnalzer ili CodeGenerator. Definiše enum MessageType koji predstavlja tip poruke i statičku metodu generateMessage(MessageType messageType, Object… params) gde se korišćenjem promenljivog broja parametara (Object…) prosleđuju odgovarajući parametri potrebni za generisanje datog tipa poruke.

4.5. InstructionHelper

Definiše statičke metode getOperatorCode koje prihvataju AdditionOperator ili MultiplicationOperator ili RelationalOperator i vraćaju kod njihove instrukcije u MicroJava VM.

Na primer: getOperatorCode(AdditionOperator(“+”)) će vratiti Code.add.

4.6. LexerUtils

Klasa koja se koristi u testu za Lexer gde se putem refleksije izvlače imena leksema iz Leksera radi lepšeg ispisivanja prilikom testiranja.

4.7. LoopJumpAddressStack

Implementacija steka liste adresa korišćenih za skokove pri korišćenju petlji (for, foreach, continue, break).

4.8. OperatorHelper

Definiše statičke metode getOperatorCode koje prihvataju AdditionOperator ili MultiplicationOperator ili RelationalOperator i vraćaju njihovu stringovnu reprezentaciju

Na primer: za leksemu AdditionOperator(“-“) getOperatorCode će vratiti “-“.