Univerzitet u Beogradu, Elektrotehnički fakultet

Projekat iz predmeta Programski Prevodioci 1

IR4PP1

Kompajler za jezik Mikrojava

Student:  
Dino Hadžiahmetović, 2015/0197

Beograd, februar 2021

1. *Opis Projekta*

Potrebno je realizovati kompajler za programski jezik Mikrojava. Kompajler omogućava prevođenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvršava na odgovarajućoj virtuelnoj mašini. Sintaksno i semantički ispravni programi su definisani MJ specifikacijom (mikrojava\_2020\_2021\_jan.pdf).

Prevođenje se sastoji iz četiri faze:

1. Leksička analiza
2. Sintaksna analiza
3. Semantička analiza
4. Generisanje koda
5. Leksička analiza

Leksički analizator je implementiran pisanjem .flex specifikacije. Skener treba da prihvati fajl sa izvornim kodom napisanim na jeziku Mikrojava i da vrati skup tokena izdvojenih iz izvornog koda, koji se dalje razmatraju u okviru sintaksne analize. Detektovane su i obrađene sledeće strukture: identifikatori, konstante, ključne reči, operatori, komentari.

Ukoliko se naiđe na leksičku grešku, leksički analizator je prijavljuje i nastavlja sa obradom ulaznog programa.

1. *Sintaksna analiza*

Na osnovu specifikacije jezika napisana je gramatika i implementiran parser za programe pisane na jeziku Mikrojava. Gramatička specifikacija je napisana u .cup fajlu.

Parser ima zadatak da utvrdi da li izdvojeni tokeni formiraju gramatički ispravne naredbe. Ako je parsiranje uspešno, na kraju rada parsera se prikazuje sintaksno stablo i generišu klase za sve uvedene smene u kodu, dok u slučaju nailaska na grešku parser ispisuje poruku o grešci i vrši oporavak od greške. Oporavak od greške se vrši u slučaju:

* neispravno definisane globalne promenljive
* neispravne konstrukcije dodele
* neispravne deklaracije formalnog parametra funkcije
* neispravnog logičkog izraza unutar if konstrukcije

1. *Semantička analiza*

Kod Semantičke analize vrši se provera konteksnih uslova i ažuriranje tabele simbola. Semantička analiza se vrši obilaskom sintaksnog stabla formiranog prilikom sintaksne analize. U ovoj fazi se implementira klasa SemanticAnalyzer koja proširuje klasu VisitorAdapter. Klasa SemanticAnalyzer redefiniše određene metode klase VisitorAdabpter, odnosno posećuje one čvorove sintaksnog stabla koji su relevantni za semantičku analizu.

U ovoj fazi se vrši provera ispravnosti koda. Proverava se da li su korišćeni simboli definisani, da li postoje višestruke definicije, proverava se kompatibilnost tipova pri naredbi dodele, ternarnarnog operanda i učesnika u aritmetičkoj operaciji, u ovoj fazi se takođe vrši detektovanje:

- simboličkih konstanti  
- globalnih promenljivih  
- lokalnih promenljivih  
- poziva globalnih funkcija  
- pristup elementu niza  
- korišćenje formalnog argumenta funkcije

1. *Generisanje koda*

Prilikom generisanja koda vrše se transformacije sintaksno i semantički ispravnog sintaksnog stabla u bajtkod za isvršno okruženje za Mikrojava virtuelnu mašinu.

Generisanje koda takođe se vrši obilaskom apstraktnog sintaksnog stabla koje je nastalo kao rezultat sintaksne analize i zadovoljilo semantičku proveru.

Implementirana je klasa CodeGenerator koja proširuje klasu VisitorAdapter i u njoj redefiniše metode za obilazak čvorova sintaksnog stabla koji su relevantni za generisanje koda.

Izlaz generatora koda je izvršni .obj fajl za Mikrojava virtuelnu mašinu

*Prevođenje*

1. Lexer Generator – pokretanjem ove konfiguracije se na osnovu .flex fajla generiše klasa Yylex
2. MJTest – testira lexer
3. Parser Generator – gramatika je napisana u mjparser.cup fajlu i pokretanjem te konfiguracije generiše se klasa MJParser, generišu se klase koje odgovaraju čvorovima sintaksnog stabla. Pre pokretanja Parser Generator-a, da ne bi došlo do konflikata, treba obrisati sadržaj paketa rs.ac.bg.etf.pp1.ast i nakon pokretanja treba osvežiti projekat
4. Compiler – pokreće main metodu klase Compiler, koja poziva sve 4 faze obrade nad ulaznim programom koji se prosleđuje kao prvi argument komandne linije i kreira objektni fajl koji se nalazi na putanji koja je prosleđena kao drugi argument komandne linije, ispisuje tabelu simbola i ukoliko je parsiranje uspešno, ispisuje i sintaksno stablo i zahtevane podatke iz semantičke analize
5. Run – pokreće prevedeni bajt kod koji se nalazi u odgovarajućem .obj fajlu koji se prosleđuje kao argument komandne linije, takođe možemo proslediti i ručku „-debug“ nakon koje možemo videti stanje steka i ulazne parametre pri izvršavanju svake komande
6. Disasm – pokretanjem ove konfiguracije možemo videti izgenerisani bajtkod
7. RunTest – poziva main metodu klase Compiler i može pozvati Disasm i Run kofiguraciju (sa i/ili bez „-debug“ ručke)

*Test primeri*

1. test/test302.mj - javni test za B nivo
2. test/error.mj – program koji prikazuje rad oporavka od grešaka
3. test/HelloWorld.mj – prikazuje rad nekih obrađenih funkcionalnosti