Programski prevodioci 1

# 

# 

# 

# 

Tijana Djokic

# 

# 

# Postavka:

Potrebno je realizovati kompajler za programski jezik Mikrojava.

Potrebno je da kompajler radi leksicu analizu koda, sintaksnu analizu koda, semanticku analizu koda i generisanje koda za mikro java virtualnu masinu. Kompajler omogucava prevodjenje sintaksno i semanticki ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvrsava na virtuelnoj masini za Mikrojavu. Sintaksno i semanticki ispravni Mikrojava programi su definisani specifikacijom.

Leksički analizator je potrebno da prepoznaje jezičke lekseme a kao rezultat vraca se skup tokena izdvojenih iz izvornog koda, Ti tokeni ce dalje biti razmatrani u okviru sintaksne analize. Ako se tokom leksičke analize otkrije neka vrsta leksičke greške, 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.

OPIS KOMANDI ZA GENERISANJE JAVA KODA ALATIMA

(PREVODJENJE KODA KOMPAJLEROM, POKRETANJE I TESTIRANJE RESENJA)

-build.xml

-Implementiratan je interfejs Compiler. On deklariše samo jednu metodu – compile. Metoda prima kao argumente putanju do ulaznog fajla koji sadrži kod koji se kompajlira i putanju do izlaznog fajla u koji je potrebno smestiti generisani bajtkod. Povratna vrednost metode je lista CompilerError objekata. U slučaju uspešnog kompajliranja i generisanja koda, vraća se null ili prazna lista. Klasa CompilerError opisuje greške do kojih je moguće doći prilikom kompajliranja. Nabrajanje (enum) CompilerErrorType predviđa postojanje leksičke, sintaksne i semantičke greške. Prilikom kreiranja greške zadaje se i linija na kojoj je greška otkrivena, kao i opis greške.

-Za testiranje koristimo fajl program.mj gde pisemo kod koji zelimo da kompajliramo nasim kompajlerom. Nakon sto smo napisali program koji zelimo da kompajliramo korisiticemo klasu koja implementira gore spomenuti interfejs Compile. Kao rezultat dobijamo program.obj.

KRATAK OPIS PRILOZENIH TEST PRIMERA:

U prilozenim test primerima se pokusalo testirati ono sto je navedeno u zahtevima domaceg zadatka, kao i sto je bilo trazeno izmenama za avgustovski rok.

1. primer

// ovo je test primer (jedan mikro java program)

program MJProgram

int a;

int b;

char c;

{

void main() {

//c=2+2+; //sintaksna greska

//a#=0; //lexicka greska

a=a+1;

//a=a+;

// a$=1; //lexicka greska

// b=b+;

b=1;

b=2;

c=1;

c=1;

c=1;

}

}

U ovom primeru su se pokazale situacije u kojima dolazi do javljanja sintaksne i leksicke greske. Leksicke greske su se javljale iz razloga sto simboli # i $ nisu simboli koji su dozvoljeni za upotrebu i njih leksicki analizator prepoznaje kao gresku, i takodje prijavljuje.

Sintaksna greska se javlja u situaciji kada (kao sto je gore prikazano) naredba nije u skladu sa nasom gramatikom.

1. primer

// Test301

program test301

const int nula = 0;

const int jedan = 1;

const int pet = 5;

int niz[];

char nizch[];

{

void main()

int bodovi;

bool bt;

{

bodovi = 0;

bodovi++; //bodovi=1

bodovi = bodovi + jedan; //bodovi=2

bodovi = bodovi \* pet; //bodovi=10

bodovi--; //bodovi=9

print(bodovi); //9 --->tacno

niz = new int[3];

niz[nula] = jedan; //niz[0]=1

niz[1] = 2; //niz[1]=2

niz[niz[jedan]] = niz[niz[0]] \* 3; //niz[2]=6

bodovi = niz[2]/niz[0]; //bodovi=6

print(bodovi); //6 --->tacno

print(niz[2]); //6 --->tacno

nizch = new char[3];

nizch[0] = 'a';

nizch[jedan] = 'b';

nizch[pet - 3] = 'c';

print(nizch[1]);

print(nizch[jedan \* 2]); //bc --->tacno

bodovi = bodovi + (pet \* jedan - 1) \* bodovi - (3 % 2 + 3 \* 2 - 3); //6+4\*6-(1+6-3)=30-(4)=26

print(bodovi); //26 --->tacno

//read(bodovi);

bt = true;

if(bt) {

print(bodovi); //26 --->tacno

} else {

print(bodovi + 10);

}

if(bodovi > 10) {

print('\*'); //\* --->tacno

} else {

print('-');

}

}

}

Ovo je primer u kome su testiranje razne stvari: upotreba konstanti, upotreba globalnih promenljivih, upotreba lokalnih promenljivih, manipulaciju nad podacima koriscenjem razlicitih operatora. Takodje prikazana je i upotreba structure if-else. Sa strane u formi komentara su prikazane vrednost promenljivih, tako da znamo sta mozemo da ocekujemo na izlazu.

KRATAK OPIS NOVOUVEDENIH KLASA

-IFLista, ElseLista, PozicijaLista su klase koje opisuju strukuru steka i manipulaciju podacima u steku, koriscenjem lista za implementaciju steka. Ove klase cemo koristiti u CodeGeneratoru pri realizaciji skokova i cuvanju adresa.

-CompilerImpl – klasa koja se koristi za implementaciju kompajlera Compiler.