
Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Programski prevodioci 1
Nastavnik: doc. dr Dragan Bojić
Asistent: dipl.ing. Nemanja Kojić
Školska: 2010/2011.
Ispitni rok: jun 2011.
Datum: 11.04.2010.

Projekat

- Kompajler za Mikrojavu -

Važne napomene: Pre čitanja ovog teksta, **obavezno** pročitati opšta pravila predmeta i pravila vezana za izradu domaćih zadataka! Pročitati potom ovaj tekst **u celini i pažljivo**, pre započinjanja realizacije ili traženja pomoći. Ukoliko u zadatku nešto nije dovoljno precizno definisano ili su postavljeni kontradiktorni zahtevi, student treba da uvede razumne pretpostavke, da ih temeljno obrazloži i da nastavi da izgrađuje preostali deo svog rešenja na temeljima uvedenih pretpostavki. Zahtevi su namerno nedovoljno detaljni, jer se od studenata očekuje kreativnost i profesionalni pristup u rešavanju praktičnih problema. **Srećan rad!**

1. Uvod

U okviru datog projektnog zadatka potrebno je realizovati kompajler za Mikrojavu koji omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji može da se izvršava na Mikrojava virtuelnoj mašini. Sintaksno i semantički ispravni Mikrojava programi su definisani specifikacijom i eventualnim izmenama u toj specifikaciji u okviru teksta projektnog zadatka. Dati projekat se izvodi u dve faze.

Cilj prve faze projekta je da se omogući parsiranje sintaksno ispravnih i sintaksno neispravnih programa. 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 na kraju obavestiti korisnika o uspešnosti parsiranja. Međutim, ukoliko se vrši parsiranje programa sa sintaksnim greškama, potrebno je izdati adekvatno objašnjenje o detektovanoj sintaksoj greški, izvršiti oporavak i nastaviti parsiranje.

U drugoj fazi projekta nastavlja se razvoj kompajlera za Mikrojava programe kroz nadogradnju parsera sa ciljem da se omogući semantička analiza i prevodjenje programskog koda napisanog na Mikrojavi u izvršni oblik za odabrano izvršno okruženje.

Nakon uvodnog dela sledi opis zadataka i zahteva za prvu fazu izrade projekta. Svi relevantni pomoćni materijali za izradu projekta se mogu pronaći na sajtu predmeta ili u okviru sekcije Prilog ovog dokumenta.

2. Specifikacija projektnih zadataka

U okviru prve faze projekta potrebno je realizovati dve komponente kompajlera: leksički i sintaksni analizator. Sledi detaljan opis zahteva za konstrukciju pomenutih komponenata.

I Izmene u programskom jeziku Mikrojava

- 1) Omogućiti pisanje komentara u više redova. Takvi komentari počinju kombinacijom znakova `/*` i završavaju se kombinacijom znakova `*/`.
- 2) U Mikrojavu se uvodi tip podataka `string`, koji reprezentuje znakovne nizove. Instance znakovnih nizova se alociraju na heap-u. Konstante znakovnih nizova se navode kao niz karaktera u okviru dovstrukih navodnika (npr. `"mikrojava pp1"`). Nad podacima tipa `string` moguće je vršiti operaciju nadovezivanja korišćenjem operatora `+`. Dva podatka tipa `string` se nadovezuju tako što se napravi novi podatak tipa `string`, čija je dužina jednaka zbiru dužina stringova koji se spajaju, a zatim se taj `string` popuni znakovima iz izvorišnih stringova – prvo iz levog operanda, a zatim iz desnog. Sledi primer operacija koje se mogu sprovoditi nad podacima tipa `string`.

```
string s1 = "mikrojava";  
string s2 = "pp1";  
string s3 = s1+" - " + s2; // "mikrojava - pp1"
```

Potrebno je realizovati/implementirati odgovarajuću (globalnu, ugrađenu) funkciju za ispis stringova na standardni izlaz. Potpis funkcije treba da bude: **void print(string s);**

- 3) Unutrašnje klase treba da imaju konstruktore. Ukoliko se ne navede nijedna definicija konstruktora, kompajler generiše podrazumevani konstruktor bez argumenata i sa praznim telom. Prilikom pravljenja objekata unutrašnjih klasa, neophodno je pozvati odgovarajući konstruktor klase.

Primer:

```
class X { int p; X(int k) { p=k; } }  
...  
int p = 4;  
X x1 = new X(3), x2 = new X(p+2);  
X x3 = new X; // ovo nije korektno!  
X x4 = new X(); // ovo nije korektno, zbog X(int)!
```

- 4) Uvode se 4 nova bitska operatora za aritmetičko pomeranje: `<<`, `>>`, `<<=`, `>>=`. Dati operatori su binarni operatori. Operandi su tipa `int`. Levi operand može da bude ili promenljiva ili konstanta i predstavlja vrednost koju treba pomerati. Drugi operand je obavezno konstanta i njegova vrednost predstavlja broj bitskih mesta za koliko se vrši pomeranje i mora obavezno biti pozitivna vrednost. Vrednost izraza sa operatorima `<<` i `>>` je odgovarajuća pomerena vrednost, pri čemu se vrednost levog operanda ne menja. Sa druge strane, vrednost izraza sa operatorima `<<=` i `>>=` je takođe pomerena vrednost za odgovarajući broj mesta, pri čemu se vrši dodela dobijene vrednosti levom operandu. U tom slučaju, levi operand mora biti promenljiva.
- 5) Uvodi se ternarni operator.
Sintaksa: `Bool_Expr ? Expr1 : Expr2`

Semantika: Izračunava se vrednost logičkog izraza. Ukoliko je ta vrednost TRUE, vrednost izraza sa ternarnim operatorom je vrednost izraza Expr1, u suprotnom je to vrednost izraza Expr2.

II Leksička analiza (4 poena)

Potrebno je realizovati leksički analizator (skener) Mikrojava fajlova. Skener prihvata fajl u kome se nalazi izvorni Mikrojava kod i deli ga na tokene. Tip tokena se vraća pri eksplicitnom pozivu leksičkog analizatora (poziv `next_token()` na Javi, ili `yylex()` na c++u). U slučaju leksičke greške analizator vraća token greške (`sym.INVALID`) i nastavlja analizu do kraja ulaznog fajla.

Izvorni kod se piše prema pravilima koja su definisana unijom pravila iz specifikacije Mikrojave (<http://ir4pp1.etf.rs/Predavanja/mikrojava.pdf>) i novih, odnosno redefinisanih pravila koja su navedena u prethodnom pasusu I.

Student može po svojoj volji izabrati da domaći uradi u jednoj od sledećih varijanti:

- A) Implementacija na javi korišćenjem alata jflex. Interfejs skenera prema parseru mora biti standardni jflex-cup interfejs. Za više informacija, pogledati primer mini domaćeg u vežbama na sajtu predmeta. Moraju se koristiti jdk 1.6.x i aktuelne verzije jflex i cup alata kao što je opisano u primerima na vežbama. Korišćenje integrisanih okruženja (Eclipse ili Netbeans) nije zabranjeno, ali prevedeni kod mora da može da se pokrene iz komandne linije na računaru koji je podešen kao što je opisano u JFlex primerima u okviru vežbi.
- B) Implementacija na c++u korišćenjem alata flex. Interfejs prema parseru mora odgovarati parseru generisanom byacc alatu. Dodatna uputstva su u Prilogu 1. Mora se koristiti C++ kompajler koji je deo Visual Studio 2005. Prevedeni program mora biti unmanaged konzolna aplikacija koja se pokreće iz komandne linije.

III Sintaksna analiza (12 poena)

Potrebno je definisati LALR(1) gramatiku i implementirati sintaksni analizator (parser) za programe napisane na MJ. U slučaju uspešnog parsiranja ulaznog fajla parser na kraju rada na standardnom izlazu prikazuje poruke o broju prepoznatih jezičkih elemenata (kako je definisano u zadatku) i na kraju poruku o uspešnom parsiranju (vidi Prilog 4). U slučaju nailaska na sintaksnu grešku parser prijavljuje poruku na standardni izlaz greške (`System.err` u slučaju jave, odnosno `cerr` u slučaju c++a) koja sadrži **broj linije ulaznog programa** (videti realizaciju u primeru mini domaćeg sa sajta predmeta), opis greške, vrši oporavak od greške i nastavlja sa parsiranjem ostatka fajla. U slučaju pojave leksičke greške greška se ispisuje na izlazu greške i ignoriše u sintaksoj analizi.

Sintaksa MJ fajla je data u specifikaciji Mikrojave, a jedan predlog LALR(1) gramatike Mikrojave, koja se može ali i ne mora koristiti u Prilogu 3. Na datu specifikaciju potrebno je dodati oporavak od grešaka. Koristeći kreiranu gramatiku potrebno je generisati parser obavezno koristeći alat CUP (u slučaju korišćenja jave) ili alat byacc (u slučaju c++a).

Ne koristiti opciju precedence u CUP odnosno .y fajlu.

Potrebno je uraditi sledeće stavke:

1. [4 poena] Napisati gramatiku koja opisuje sintaksno ispravne MJ programe.
 - a. Napisati CUP, odnosno byacc specifikaciju sintaksnog analizatora koji prepoznaje MJ fajlove i implementirati odgovarajući CUP, odnosno byacc parser.
 - b. Gramatika mora da sadrži neterminale koji su nazvani isto kao u MJ specifikaciji uz eventualne dodatne neterminale.
 - c. Elementi koje treba zasebno prebrojavati su standardne jezičke konstrukcije MikroJave:

- i. Deklaracije klasa
 - ii. Metode glavne klase
 - iii. Deklaracije globalnih promenljivih prostog tipa
 - iv. Deklaracije promenljivih tipa string.
 - v. Deklaracije globalnih konstanti
 - vi. Deklaracije globalnih nizova
 - vii. Deklaracije lokalnih promenljivih u okviru tela main metoda
 - viii. Iskazi u okviru tela main metoda (osim poziva funkcija)
 - ix. Poziva funkcija u okviru tela main metoda.
2. [4 poena] U gramatiku iz prethodne tačke dodati smene i akcije za oporavak od grešaka. Sintaksne greške koje parser prepoznaje i akcije kojima se oporavlja su:
- a. Neispravna definicija
 - i. globalne promenljive ili globalne konstante => ignorisati karaktere do ;
 - ii. unutrašnje klase => ignorisati karaktere do }
 - iii. lokalnih promenljivih => ignorisati karaktere do ; ili {
 - b. Neispravna konstrukcija
 - i. Iskaza dodele => ignorisati karaktere do ;
 - ii. Bloka naredbi (u { } zagradama) => ignorisati karaktere do prve }
 - iii. Logičkog izraza unutar if ili while konstrukcije => ignorisati karaktere do)
 - iv. Parametara u pozivu funkcije => ignorisati karaktere do)
 - v. Izraza u indeksu => ignorisati karaktere do]
3. U klasu generisanog parsera dodati main metod koji pokreće parsiranje nad MJ fajlom čije se ime zadaje u komandnoj liniji (nalik na CUP primer – mini domaći zadatak sa vežbi).
4. [4 poena] Za potrebe testiranja rada implementiranog parsera:
- a. Kreirati ulazne fajlove koji sadrže sve sintaksno ispravne kombinacije elemenata MJ gramatike(npr. funkcija bez formalnih parametara, sa jednim formalnim parametrom, sa više formalnih parametara).
 - b. Kreirati ulazne fajlove koji sadrže sve kombinacije grešaka koje su navedene u tački 2 zahteva.

IV Semantička analiza (12 poena)

Potrebno je proširiti sintaksni analizator iz prethodnog poglavlja da vrši semantičku analizu MJ fajlova i ažurira tabelu simbola. Obezbediti sledeće dodatne funkcionalnosti:

a. Tabela simbola

Potrebno je kompletirati zadatak implementaciju tabele simbola koja obezbeđuje čuvanje informacija o simbolima nađenim u MJ fajlu koji se analizira. Nije dozvoljeno menjati dati kod, već samo dopisivati novi.

- i. Studenti imaju parni broj indeksa treba da koriste implementaciju tabele simbola u vidu neuređene (jednostruko ili dvostruko) ulančane liste, a oni sa neparnim brojem implementaciju u vidu steka sa markerima početaka opsega. Na primer, indeks 05/234 znači da student implementira tabelu simbola kao ulančanu listu. Studenti koji koriste c++ treba ne moraju voditi računa o dealokaciji objekata iz memorije. Od pomoći može biti kompletna java implementacija tabele simbola sa jednostruko ulančanim listama koja se nalazi na sajtu predmeta u okviru arhive domaćih zadataka. Nacrti zahtevanih implementacija mogu se pronaći u materijalima sa predavanja.
- ii. Potrebno je implementirati metod **init()** u klasi Tab koji formira sadržaj opsega "universe"(globalni opseg koji sadrži naziv glavne klase i nazive preddefinisanih funkcija i tipova) i otvara ovaj opseg (na samom početku procesiranja MJ programa). Potrebno je implementirati i druge metode na mestima koja su označena u programskom kodu.

b. Semantička analiza

Dodavanjem semantičkih akcija u parser potrebno je omogućiti unos simbola u tabelu simbola i detektovati korišćenje simbola u programu po sledećim pravilima:

- i. Potrebno je obraditi deklaracije programa, konstanti, globalnih i lokalnih promenljivih (prostog tipa, nizova i string-ova), globalnih metoda (članova klase koja reprezentuje ceo program) i deklaracije unutrašnjih klasa.
- ii. Potrebno je detektovati svako naknadno korišćenje pomenutih objekata, proveriti da li objekat postoji u tabeli simbola i da li je ispravnog tipa. Potrebno je proveravati i formalne parametre za metode. **Ispisati odgovarajuću poruku na standardni izlaz (vidi Prilog 4 za format poruke).**
- iii. Na način opisan u prethodnoj tački (ii) treba obraditi sledeće vrste iskaza i izraza MJ(za iskaze koji se ne obrađuju, ne menjati gramatiku, nego samo izostaviti semantičku obradu):

```
Statement = Designator ("=" Expr | "(" [ActPars] ")" | "++" | "- -") ";"
| "if" "(" Condition ")" Statement ["else" Statement]
| "return" [Expr] ";"
| "while" "(" Condition ")" Statement
| "read" "(" Designator ")" ";"
| "print" "(" Expr ["," number] ")" ";"
| "{" {Statement} "}".
| "new" Type | "new" Type "[" Expr "]"
| "new" Type "(" ")" | "new" Type "(" ActualArgs ")"
```

1. Uslovni izrazi (Condition) mogu biti samo oblika CondFact.
2. Obradivati stvarne parametre u pozivu funkcija (mogu biti ili prostog tipa, tipa nizova ili stringovi).
3. Sintaksni element pod nazivom Designator može se odnositi samo na promenljivu prostog tipa, niz ili polje objekta unutrašnje klase.
4. Izrazi (expr) treba da se obrade kompletno (uključujući i izraze sa novouvedenim operatorima).

Na kraju izvršavanja programa (po povratku iz parsera) pozivom funkcije **dump()** ispisati sadržaj tabele simbola. Primeri izlaza navedeni su u Prilogu 4.

V Generisanje koda (12 poena)

Potrebno je proširiti semantički analizator iz prethodnog poglavlja da na osnovu ispravnog mikrojava koda generiše mikrojava bajt kod koji se može izvršiti pomoću mikrojava virtuelne mašine. Koristiti priložene klase **Code**, **disasm** i **Run**, za generisanje koda, ispis generisanog koda i izvršavanje (interpretiranje) prevedenog programa. Korišćenje ovih klasa opisano je u trećem delu primera mini domaćeg. Potrebno je obraditi sve elemente definisane u tački b u prethodnom poglavlju (semantička analiza).

3. Napomene u vezi sa izradom i odbranom rešenja

Elementi rešenja su sledeći:

1. Prpratna dokumentacija u obliku Word dokumenta MJProjekat.doc koji treba da se nalazi u korenom direktorijumu rešenja i da sadrži:

- a) naslovnu stranu,
- b) kratak opis postavke zadatka od nekoliko rečenica,
- c) detaljan opis komandi za generisanje java/c++ koda alatima, prevođenje koda kompajlerom, pokretanje i testiranje rešenja,
- d) spisak poruka o grešaka koje kompajler može da generiše,
- e) kratak opis sopstvenih klasa (van onih već zadatih) ako su korišćene, izveštaj koji daju jflex i cup, odnosno flex i byacc pri prevođenju specifikacionih fajlova, sa komentarom za svaki eventualni parserski konflikt kako nastaje i zašto nije razrešen
- f) kratak opis priloženih test primera (ne uključivati ulaze niti izlaze testiranja u izveštaj).

2. Izvorni i prevedeni programski kod za java varijantu mora da sledi direktorijumsku strukturu koja je opisana u vežbama. Dakle moraju se poslati .flex i .cup fajlovi, svi izgenerisani i rukom pisani .java fajlovi koji čine rešenje i odgovarajući prevedeni .class fajlovi. U korenu rešenja treba da se nalaze i .jar arhive cup i flex alata. Analogna pravila važe i za c++ rešenje. **Treba sve zahteve ugraditi u jedinstven parser (samo jedan CUP fajl).**

3. U posebnom folderu **test** treba da se nalaze svi ulazni test fajlovi sa ekstenzijom .MJ, kao i odgovarajući izlazni fajlovi koji su rezultat testiranja, sa istim imenom kao ulazni fajl, ali sa ekstenzijom .out za standardni izlaz i .err za izlaz greške,. Uputstvo: Pri pokretanju programa standardni izlaz može se preusmeriti u fajl izlaz.out ako se na komandnoj liniji navede **>izlaz.out**, a izlaz greške se preusmerava sa **2>izlaz.err**.

Pravila za izradu i odbranu domaćeg zadatka

1. Domaći zadatak se radi individualno i brani usmeno, pre ispita. Uspešna odbrana domaćeg je uslov za izlazak na ispit. Ukoliko se na odbrani utvrdi **nedozvoljena** saradnja između studenata prilikom izrade domaćeg, u moguće posledice osim ponovne izrade domaćeg spada i trajno dobijanje negativnih poena koji će biti uključeni u zbir za konačnu ocenu.
2. Odbrana se organizuje posle roka predaje domaćeg, prema naknadnom obaveštenju. Detalji oko postavljanja rešenja na računar za odbranu/donošenje sopstvenog računara biće naknadno precizirani.
2. Po potrebi će ulazni test fajlovi biti pokretani na odbrani domaćeg.
3. Na odbrani će, pored samog rešenja, biti proveravano i poznavanje rada sa alatima jflex, CUP, ako je rađenja java varijanta, odnosno flex/byacc za c++ varijantu.

4. Prilog

Uputstvo za realizaciju c++ verzije leksičkog analizatora

Pri izradi analizatora, bilo da se koristi flex ili se manuelno implementira konačni automat potrebno je pridržavati se sledećih pravila:

1. Analizirati specifikaciju MikroJave i identifikovati sve leksičke elemente koji se pojavljuju u njoj (ključne reči, promenljive, konstante). Sve leksičke elemente predstaviti kao konstante u fajlu `yy.tab.h`, koji sadrži niz definicija koje odgovaraju simboličkim imenima tokena i vrednostima klasa tih tokena, na primer:

```
#define LP 1
#define ID 2
```
2. Ovaj fajl uključiti direktivom `#include "yy.tab.h"` u datoteku sa izvornim kodom analizatora.
3. Leksički analizator je u vidu funkcije (u slučaju korišćenja flex-a ona se generiše automatski):

```
int yylex()
```

pri čemu svaki poziv procesira standardni ulaz od mesta gde se stalo u prethodnom pozivu ove funkcije, i vraća klasni deo tekućeg tokena (koristiti simbolička imena definisana u fajlu `yy.tab.h`). U slučaju da se došlo do kraja ulaza, potrebno je vratiti vrednost 0.
4. Vrednosni deo prepoznatog tokena postavlja se u globalnu promenljivu `yylval`. Ovu promenljivu treba deklarirati da je tipa `YYSTYPE`, na primer:

```
#define YYSTYPE int
YYSTYPE yylval;
```

U ovom primeru vrednost tokena je celobrojna. Složeniji primer, gde neki tokeni vraćaju celobrojnu vrednost, a drugi string, bio bi:

```
struct YYSTYPE { int x; char id[100]; };
#define YYSTYPE struct YYSTYPE
YYSTYPE yylval;
```


Prilog 2 - Transformisanje gramatike

1. U slučaju da je potrebno napisati smenu u kojoj se neki pojam ponavlja jednom ili više puta, odgovarajuća smena se može uraditi na sledeći način:

$\text{Parameter_list} \rightarrow \text{Parameter_list Parameter} \mid \text{Parameter}$

Gde je Parameter_list neterminal koji opisuje jedno ili više pojavljivanja objekta Parameter , dok je Parameter objekat koji treba da se ponavlja jednom ili više puta.

2. U slučaju da se grupa različitih objekata pojavljuje jednom ili više puta može se koristiti sledeći oblik smene:

$\text{Parameter_list} \rightarrow \text{Parameter_list Parameter_part} \mid \text{Parameter_part}$
 $\text{Parameter_part} \rightarrow \text{Parameter1} \mid \text{Parameter2} \mid \text{Parameter3} \mid \dots$

Gde su Parameter1 , Parameter2 , ... Tipovi objekata iz grupe koji se pojavljuju jednom ili više puta.

3. U slučaju da se neki objekat opciono pojavljuje u nekoj smeni smena se razdvaja na dve smene. Prvu koja ima traženi objekat i drugu koja ga ne sadrži. Primer takve smene je:

$\text{Funkcija} \rightarrow \text{ImeFunkcije (Parameter_list)} \mid \text{ImeFunkcije ()}$

Druga varijanta je da se uvede prazna smena (za prazne smene u CUPu samo na mestu gde bi stajala desna strana smene napisati komentar `/* epsilon */`).

4. U slučaju da se neki objekat može ponavljati nula ili više puta u nekoj smeni koristi se kombinacija pravila iz tački 1. i 2.

$\text{Funkcija} \rightarrow \text{Ime (Parameter_list)} \mid \text{Ime ()}$
 $\text{Parameter_list} \rightarrow \text{Parameter_list Parameter} \mid \text{Parameter}$

U prikazanoj smeni parametri funkcije se mogu pojaviti jednom ili više puta ali i ne moraju. Druga varijanta bi bila:

$\text{Funkcija} \rightarrow \text{Ime (Parameter_list)}$
 $\text{Parameter_list} \rightarrow \text{Parameter_list Parameter} \mid \text{/* epsilon */}$

Ova varijanta ima tu prednost da se ne multiplicitiraju smene za neterminal Funkcija .

Prilog 3 - Primer LALR(1) gramatike za MikroJavu

Primer gramatike je dat u listingu 1. Jedini specijalni simboli su \rightarrow (početak nove smene) i $|$ (razdvajanje opcija smena) svi ostali simboli kao što su “;”, “)” i slično predstavljaju ulazne simbole. Gramatika nije potpuno testirana i postoji mogućnost da ima logičkih grešaka.

Pogledati takođe u primeru mini domaćeg kako je konvertovana EBNF gramatika iz postavke zadatka u CUP gramatiku (neka rešenja su zgodnija nego u ovoj gramatici, npr. smene za methodDecl nisu multiplicirane kao u ovoj gramatici). Ovo je samo primer na koji se treba ugledati prilikom izrade domaćeg zadatka. Relevantna je specifikacija jezika i specifikacija izmena u samom jeziku.

```
Program  $\rightarrow$  CLASS IDENTIFIKATOR declaration_list
           {
               method_declaration_list
           }
           |
           CLASS IDENTIFIKATOR
           {
               method_declaration_list
           }
           CLASS IDENTIFIKATOR declaration_list
           {
               }
           |
           CLASS IDENTIFIKATOR
           {
               }
```

```
declaration_list  $\rightarrow$  declaration_list declaration_part
                   |
                   declaration_part
```

```
declaration_part  $\rightarrow$  ConstDecl ;
                   |
                   VarDecl ;
                   |
                   ClassDecl
```

```
Type  $\rightarrow$  IDENTIFIKATOR
```

```
rhs  $\rightarrow$  NUMBER | CHAR_CONST
```

```
ConstDecl  $\rightarrow$  FINAL Type IDENTIFIKATOR = rhs
```

// VarDecl predstavlja deklaraciju grupe promenljivih

// Sastoji se od tipa promenljivih i liste imena promenljivih odvojenih zarezima

```
VarDecl  $\rightarrow$  Type var_list
```

```
var_list  $\rightarrow$  var_list , var_part | var_part
```

```
var_part  $\rightarrow$  IDENTIFIKATOR
           |
```

IDENTIFIKATOR []

```

ClassDecl →      CLASS IDENTIFIKATOR
                {
                    local_field_list
                }
                |
                CLASS IDENTIFIKATOR
                {
                    }

local_field_list →      local_field_list VarDecl ;
                        |
                        VarDecl ;

method_declaration_list → method_declaration_list MethodDecl
                        |
                        MethodDecl

return_type →      Type | VOID

MethodDecl →
    return_type IDENTIFIKATOR ( FormPars ) local_field_list
        { stmt_list }
    |
    return_type IDENTIFIKATOR ( FormPars )
        { stmt_list }
    |
    return_type IDENTIFIKATOR ( FormPars ) local_field_list
        { }
    |
    return_type IDENTIFIKATOR ( FormPars )
        { }
    |
    return_type IDENTIFIKATOR ( ) local_field_list
        { stmt_list }
    |
    return_type IDENTIFIKATOR ( )
        { stmt_list }
    |
    return_type IDENTIFIKATOR ( ) local_field_list
        { }
    |
    return_type IDENTIFIKATOR ( )
        { }

FormPars →      parameter parameter_list
                |
                parameter

parameter →      Type IDENTIFIKATOR
                |

```

Type IDENTIFIKATOR[]

```

parameter_list → parameter_list , parameter
                |
                , parameter

stmt_list      → stmt_list Statement
                |
                Statement

Statement      → Matched
                |
                Unmatched

Unmatched      → IF ( Condition ) Statement
                |
                IF ( Condition ) Matched ELSE Unmatched
                |
                WHILE ( Condition ) Unmatched
                Matched → Designator = Expr;
                |
                Designator ( );
                |
                Designator ( ActPars );
                |
                Designator ++;
                |
                Designator --;
                |
                BREAK;
                |
                RETURN;
                |
                RETURN Expr;
                |
                READ (Designator);
                |
                PRINT ( Expr );
                |
                PRINT ( Expr , NUMBER );
                |
                { }
                |
                { stmt_list }
                |
                IF ( Condition ) Matched ELSE Matched
                |
                WHILE ( Condition ) Matched

```

Designator →	IdentExpr_list
IdentExpr_list →	IdentExpr_list . IDENTIFIKATOR IdentExpr_list [Expr] IDENTIFIKATOR
Relop →	== <> < <= > >=
Addop →	+ -
Mulop →	* / %
ActPars →	expr_list
expr_list →	expr_list , Expr Expr
Expr →	Term_list - Term_list
Term_list →	Term_list Addop Term Term
Term →	Factor_list
Factor_list →	Factor_list Mulop Factor Factor
Factor →	Designator (ActPars) Designator NUMBER CHAR_CONST NEW Type NEW Type [Expr] (Expr)
Condition →	CondTerm OR CondTerm_list

		CondTerm
ORCondTerm_list →	ORCondTerm_list OR	CondTerm
		OR CondTerm
CondTerm →	CondFact AND	CondFact_list
		CondFact
ANDCondFact_list →	ANDCondFact_list &&	CondFact
		&& CondFact
CondFact →	Expr Relop	Expr

Listing 1. Primer gramatike za MikroJavu.

Prilog 4 - Primeri izlaza

Ulazni program:

```
class P
{
    final int size = 10;
    int pos[];
    {
        void main()
        {
            int x, i;
            char x;
            { //----- Initialize val
                x.i=1;
                pos = new int[size];
                i = 0;
                while (i < size) {
                    pos[i] = 0;
                    i++;
                }
                //----- Read values
                read(x);
                while (x >= 0) {
                    if (x < size) {
                        pos[x]++;
                    }
                    read(x);
                }
            }
        }
    }
}
```

Referentni izlaz kompajlera za navedeni program je dat u nastavku. Prijave grešaka (prikazano podebljano) treba da idu na standardni izlaz greške, ostalo na standardni izlaz.

=====SEMANTICKA OBRADA=====

Greska na 7: x vec deklarisan

Pretraga na 9(x), nadjeno Var x: int, 0, 1

Greska na 9 Polja klase nisu podrzana

Greska na 9(i) nije nadjeno

Pretraga na 10(pos), nadjeno Var pos: Arr of int, 0, 1

Pretraga na 10(size), nadjeno Con size: int, 10, 1

Pretraga na 11(i), nadjeno Var i: int, 0, 1

Pretraga na 12(i), nadjeno Var i: int, 0, 1

Pretraga na 12(size), nadjeno Con size: int, 10, 1

Pretraga na 13(pos), nadjeno Var pos: Arr of int, 0, 1

Pretraga na 13(i), nadjeno Var i: int, 0, 1

Pretraga na 14(i), nadjeno Var i: int, 0, 1

Pretraga na 17(x), nadjeno Var x: int, 0, 1

Pretraga na 18(x), nadjeno Var x: int, 0, 1

Pretraga na 19(x), nadjeno Var x: int, 0, 1

Pretraga na 19(size), nadjeno Con size: int, 10, 1

Pretraga na 20(pos), nadjeno Var pos: Arr of int, 0, 1

Pretraga na 20(x), nadjeno Var x: int, 0, 1

Pretraga na 22(x), nadjeno Var x: int, 0, 1

=====SINTAKSNA ANALIZA=====

1 classes

1 methods in the program

0 global variables

1 global constants

1 global arrays

3 local variables in main

13 statements in main

2 function calls in main

=====SADRZAJ TABELE SIMBOLA=====

(Level 0)

Type int: int, 0, 0

Type char: char, 0, 0

Con eol: char, 10, 0

Con null: Class, 0, 0

Meth chr: char, 0, 1 [Var i: int, 0, 1]

Meth ord: int, 0, 1 [Var ch: char, 0, 1]

Meth len: int, 0, 1 [Var arr: Arr of notype, 0, 1]

Prog P: notype, 0, 0

[Con size: int, 10, 1]

[Var pos: Arr of int, 0, 1]

[Meth main: notype, 0, 0 [Var x: int, 0, 1][Var i: int, 0, 1]]

5. Zapisnik revizija

Ovaj zapisnik sadrži spisak izmena i dopuna ovog dokumenta po verzijama.

Verzija 1.1

Strana	Izmena