



---

UNIVERZITET U BEOGRADU  
Elektrotehnički fakultet  
Katedra za računarsku tehniku i informatiku

---

# Programski Prevodioci 1

Domaći – Januar 2009

Profesor:  
**dr Dragan Bojić**

Asistent:  
**dipl ing Miloš Gligorić**

Beograd, novembar 2008.

Verzija 0.5

Izmene verzija 0.3	Opis
Zahtev 4 (prevodjenje)	Uklonjen opis za pokretanje programa

Izmene verzija 0.4	Opis
Strana 5 (pokretanje parsera)	Implementirati metodu main u okviru parsera

Izmene verzija 0.5	Opis
Strana 7 (prevodjenje)	Uklonjena je stavka koja je dvosmisleno definisala jezika u koji je potrebno preslikati MJ program

## **Zahvalnost**

Posebnu zahvalnost dugujem **Aleksandru Milićeviću** i **Saši Misailoviću** na komentarima, predlozima i strpljenju.

## **Napomene**

### ***Broj poena***

Broj poena koji nosi domaći zadatak je 40. Ukoliko se prilikom izrade zadatka koristi Apstraktno Sintaksno Stablo (AST), moguće je osvojiti dodatnih 5 poena.

### ***Slanje domaćeg***

Prijava za odbranu domaćeg vrši se od datuma objavljivanja do 30.11.2008. U cilju prijavljivanja potrebno je poslati, prazan, mejl na adresu [compilers.etf@gmail.com](mailto:compilers.etf@gmail.com), pri čemu subject ima sledeći format: PRIJAVA IME PREZIME INDEKS. Gde su IME PREZIME i INDEKS, ime, prezime i indeks studenta. Zakasnele prijave neće se uzimati u razmatranje.

Rok za predaju neće biti poznat unapred. Ono što je poznato jeste da neće biti pre 31.12.2008. Rok za predaju biće objavljen 4 dana pre datuma predaje.

Način/format predaje domaćeg zadatka biće definisan prilikom objavljivanja datuma predaje (prethodni pasus).

### ***Opšte napomene***

Kod koji studenti kreiraju potrebno je da se nalaz i paketu koji je sledećeg formata xggbbbb. Gde je x p ukoliko je parni indeks u pitanju odnosno n za neparni. gg godina upisa i bbbb broj indeksa studenta.

Kreiranje klasa za AST je proizvoljno, preporučuje se upotreba classgen alata (<http://classgen.sourceforge.net/>).

Nepoštovanje nekog od prethodnih pravila povlači negativne poene.

## Leksička i sintaksna analiza MikroJava programa

### 1. (4)

Potrebno je napraviti CUP kompatibilan leksički analizator modifikovanih MikroJava fajlova. Leksički analizator prihvata tekstualni fajl sa ekstenzijom .mj u kojem se nalazi izvorni kôd koji odgovara specifikaciji MikroJava jezika (<http://ir4pp1.etf.bg.ac.yu/Domaci/mikrojava.rar>) uz određene izmene i deli ga na leksičke jedinice – tokene (simbole). U slučaju leksičke greške analizator vraća token greške (sym.INVALID) i nastavlja analizu do kraja ulaznog fajla.

Traženi analizator bi trebalo implementirati u Javi upotrebom alata JFlex (<http://jflex.de/>).

Modifikacija MikroJava jezika sastoji se u sledećem:

- Pored postojećih komentara u jednoj liniji (//) potrebno je omogućiti i pisanje komentara u više linija ( `/*-- komentar -->` ) kao i dokumentacionih komentara ( `/** */` ).
- Potrebno je dodati strukture (struct) po definiciji jezika C. Strukture se mogu pojaviti na mestima gde i unutrašnje klase.
- Potrebno je dodati ternarni operator (?:) po definiciji jezika C.
- Potrebno je dodati switch po definiciji jezika C.

Prilikom leksičke analize potrebno je obezbediti brojanje sledećih pojavljivanja:

- (2) broj komentara u više redova u ulaznom fajlu
- (1) broj while tokena (WHILE)
- (1) broj belih znakova u ulaznom fajlu (isspace(c) funkcija jezika C)

**Važno:** pronadjene brojeve i stringove ne treba ispisivati na ekranu. Treba obezbediti da se traženi brojevi i stringovi mogu dohvatiti pozivom odgovarajuće metode. Pogledati Dodatak B (Prikupljanje rezultata).

### 2. (16)

- a) Napisati LALR(1) gramatiku koja obezbeđuje parsiranje sintaksno ispravnih, modifikovanih, MikroJava programa, ali i onih sa eventualnim sintaksnim greškama. Gramatika mora da sadrži neterminale koji su nazvani isto kao u specifikaciji MikroJava uz dodatne neterminale. Kao osnova za pisanje gramatike može se koristiti gramatika data u Dodatku C (MJ LALR(1) Gramatika), potrebno je dopuniti gramatiku odgovarajućim izmenama MJ jezika (pogledati tačku 1), na koju se dodaju akcije za oporavak od grešaka.
- b) Na osnovu gramatike iz prethodne tačke napisati CUP specifikaciju sintaksnog analizatora, tj. parsera, koji prepoznaje MikroJava fajlove i implementirati ga upotrebom CUP alata ([http://ir4pp1.etf.bg.ac.yu/Domaci/java\\_cup\\_v10k.rar](http://ir4pp1.etf.bg.ac.yu/Domaci/java_cup_v10k.rar)). Pri tome važi sledeće:
  - 01) Koristiti leksički analizator urađen u tački 1. U slučaju da postoji leksička greška u MikroJava fajlu koji se analizira, opis greške treba ispisati na izlaz i ignorisati je u sintakсноj analizi.
  - 02) Ne koristiti opciju precedence u CUP fajlu.
  - 03) Pri prepoznavanju određenih elemenata ulaznog fajla treba prebrojati sledeće:

- (2) broj struktura (struct)
- (2) broj polja unutar svake strukture
- (2) broj ternarnih operatora
- (1) if u čijem se then delu nalazi barem jedna while petlja (if može biti i sa else granom).
- (1) koliko puta je broj argumenata + broj lokalnih promenljivih isti u dve uzastopne metode.
- (1) broj klasa koje poseduju tačno 4 polja. Barem jedno polje mora biti niz.
- (1) broj relacionih operatora.
- (1) metode koje imaju tri lokalne promenljive, pri čemu je ime lokalnih promenljivih u rastućem leksikografskom poretku (pr: int a, b, c;).
- (1) broj poziva metoda koje imaju barem dva argumenta.

**Važno:** prilikom prebrojavanja ne treba vršiti ispisivanje na ekranu. Treba omogućiti da se po završetku parsiranja pozivom odgovarajuće metode mogu dohvatiti tražene vrednosti. Pogledati dodatak Dodatak B (Prikupljanje rezultata).

**04)** U slučaju nailaska na sintaksnu grešku analizator uvećava odgovarajući brojač greške, vrši oporavak od greške i nastavlja sa parsiranjem ostatka ulaznog fajla. Sintaksne greške koje parser prepoznaje i akcije kojima se oporavlja su:

- (1) Neispravna definicija polja klase. Ignorišu se svi karakteri do prvog znaka }
- (1) Neispravna konstrukcija stvarnih argumenata prilikom poziva funkcije print. Ignorišu se svi karakteri do prvog znaka )
- (1) Nekorektno kreiran relacioni operator. Ignorišu se svi karakteri do ).
- (1) Neispravna konstrukcija izraza inkrementiranja odnosno dekrementiranja. Ignorišu se svi karakteri do prvog znaka ;

**Važno:** prilikom prebrojavanja ne treba vršiti ispisivanje na ekranu. Treba omogućiti da se po završetku parsiranja pozivom odgovarajuće metode mogu dohvatiti tražene vrednosti. Pogledati dodatak Dodatak B (Prikupljanje rezultata).

Potrebno je kreirati klasu koja implementira sledeći interfejs, pokretanje parsera vrši se pozivom metode runParser:

```
package domaci.jan2009;

public interface IMain {
    /*
     * Treba predvideti da se runParser metoda moze pozivati
     * vise puta tokom izvorsavanja (obezbediti da se prilikom
     * svakog poziva ponasa kao da je prvi put pozvana)
     */
    public ResultCollector runParser(String fileName);
}
```

Pored prethodno opisanog načina za pokretanje potrebno je obezbediti i metodu main unutar klase parsera koja kao prvi argument prihvata ime fajla (radjeno na vežbama).

### Semantička analiza Mikro Java programa i prevođenje u C++ izvorni kod

Potrebno je projektovati analizator za jezik MikroJava (MJ) koji, pored leksičke i sintaksne analize, puni tabelu simbola, vrši semantičku analizu MJ programa i prevođenje MJ izvornog koda u C++ izvorni kod. Rad analizatora će biti proveravan samo na sintaksno ispravnim MJ programima. Konkretno, potrebno je obezbediti sledeće:

#### 3. (10)

##### a) (5)

Potrebno je omogućiti unos svih simbola MikroJava jezika u tabelu simbola. Tabela simbola koju je potrebno implementirati ima sličnu strukturu i logiku popunjavanja kao tabela izložena na vežbama i predavanjima (pogledati materijale sa sajta), znači i dalje postoje Object, Struct i Scope čvorovi. Implementacija je proizvoljna a kao osnova može biti ranija implemenatcija sa ulančanim listama, data na sajtu predmeta (<http://ir4pp1.etf.bg.ac.yu/Domaci/TabSimImpl.rar>). Pored unosa u tabelu simbola, potrebno je detektovati svako korišćenje simbola (pod korišćenjem se misli na upotrebu u naredbama u telu metoda), proveriti da li odgovarajući objekat postoji u tabeli simbola i ispisati poruku (koja uključuje i broj linije u kodu u kojoj je razmotreni simbol) o pronađenom simbolu ili poruku o grešci (npr. simbol nije pronađen; simbol nije tipa niza, a koristi se uz operator indeksiranja; simbol nije polje klase i slično), videti tačku b. Poruka mora biti takva da se iz nje može nedvosmisleno zaključiti koji simbol je pronađen (ime simbola, vrsta, tip).

Na kraju izvršavanja programa moguće je ispisati tabelu simbola pozivom metode printST, pogledati Dodatak B.

Kod objekata u tabeli simbola koji odgovaraju konstantama, potrebno je postavljati vrednost polja adr, jer je to vrednost konstante. Kod ostalih objekata u tabeli simbola NIJE POTREBNO dodeljivati vrednost polju adr.

##### b) (5)

Obezbediti proveru sledećih kontekstnih (semantičkih) uslova MJ jezika:

01) (2) U pozivu metode stvarni argumenti moraju po broju i tipu odgovarati formalnim argumentima u definiciji te metode.

02) (2) Posmatra se sledeća smena MikroJava gramatike:

**Factor = "new" Type "[" Expr "].**

Za prethodno navedenu smenu mora važiti:

**Expr** mora biti int.

03) (1) Posmatra se smena MJ gramatike:

**Term = Term Mulop Factor.**

**Term i Factor** moraju biti tipa int.

#### 4. (10)

Obezbediti da MJ analizator, na osnovu ulaznog MJ fajla, pravi novi fajl sa izvornim C++ kodom koji odgovara ulaznom MJ programu. Pri tome važi sledeće:

- 1) Kada se analizator pokrene za neki MJ program, on će u tekućem direktorijumu u kojem se nalazi napraviti fajl `imeGlavneMJKlase.cpp`, u koji će upisati odgovarajući C++ kod.
- 2) Pošto se MikroJava i C++ razlikuju, ne može se izvršiti 1-1 preslikavanje MJ koda u C++ kod.
- 3) Od studenta se očekuje da definiše pravila preslikavanja i dokumentuje ista. Za neka preslikavanja moguće je definisati više pravila, potrebno je usvojiti jedno od mogućih i nastaviti izgradjivanje rešenja nad usvojenim pravilom.
- 4) **SEMANTIKA MIRKO JAVA PROGRAMA PREDATOG NA PREVODJENJE MORA BITI OČUVANA!**

**Napomena:** Treba obezbediti da se dobijeni C++ kod može kompajlirati i pokrenuti.

## Dodatak B – Prikupljanje rezultata

```
package domaci.jan2009;

public interface ResultCollector {

    // ----- LEXER -----

    /**
     *
     * @return broj komentara u vise redova u ulaznom fajlu
     * @throws UnsupportedOperationException
     *         ukoliko metoda nije implementirana
     */
    public int numOfMulComments() throws UnsupportedOperationException;

    /**
     *
     * @return broj WHILE tokena u ulaznom fajlu
     * @throws UnsupportedOperationException
     *         ukoliko metoda nije implementirana
     */
    public int numOfWhileTokens() throws UnsupportedOperationException;

    /**
     *
     * @return broj belih znakova u ulaznom fajlu
     * @throws UnsupportedOperationException
     *         ukoliko metoda nije implementirana
     */
    public int numOfWhiteSpaces() throws UnsupportedOperationException;

    // ----- PARSER -----

    /**
     *
     * @return broj uvedenih struktura u ulaznom fajlu
     */
    public int numOfStructs() throws UnsupportedOperationException;

    /**
     *
     * @return broj polja za svaku uvedenu strukturu. Bitno je da broj polja,
     *         unutar niza, prati redosled deklaracija struktura unutar ulaznog
     *         fajla
     * @throws UnsupportedOperationException
     */
    public int[] numOfFields() throws UnsupportedOperationException;

    /**
     *
     * @return broj ternarnih operatora u ulaznom fajlu
     * @throws UnsupportedOperationException
     */
    public int numOfConditional() throws UnsupportedOperationException;

    /**
```



```

*
* @return broj if u cijem se then delu nalazi barem jedna while petlja
* @throws UnsupportedOperationException
*/
public int numOfIfWhile() throws UnsupportedOperationException;

/**
*
* @return broj koliko puta u ulaznom fajlu uzastopne metode imaju identican
*         broj formalnih argumenata i loklnih promenljivih
* @throws UnsupportedOperationException
*/
public int numOfArgsLocs() throws UnsupportedOperationException;

/**
*
* @return broj klasa koeje poseduju tacno 4 polja, pri cemu barem jedno
*         polje mora biti niz
* @throws UnsupportedOperationException
*/
public int numOf4Fields() throws UnsupportedOperationException;

/**
*
* @return broj relacionih operatora
* @throws UnsupportedOperationException
*/
public int numOfRelops() throws UnsupportedOperationException;

/**
*
* @return broj metoda koje imaju 3 lokalne promenljive koje su u ulaznom
*         fajlu navedene u leksikografskom poretku
* @throws UnsupportedOperationException
*/
public int numOf3Locs() throws UnsupportedOperationException;

/**
*
* @return broj poziva metoda koje imaju barem dva argumenta
* @throws UnsupportedOperationException
*/
public int numOfMethodCalls() throws UnsupportedOperationException;

// ----- GRESKE -----

/**
*
* @return broj gresaka: neispravna definicija polja klase
*/
public int numOfErrFields() throws UnsupportedOperationException;

/**
*
* @return broj gresaka prilikom poziva funkcije print
* @throws UnsupportedOperationException
*/
public int numOfErrPrint() throws UnsupportedOperationException;

/**
*
* @return broj gresaka prilikom upotrebe relacionih operatora
* @throws UnsupportedOperationException

```

```

    */
    public int numOfErrRelop() throws UnsupportedOperationException;

    /**
     *
     * @return broj gresaka prilikom konstrukcije izraza inkrementiranja odnosno
     *         dekrementiranja
     * @throws UnsupportedOperationException
     */
    public int numOfErrInc() throws UnsupportedOperationException;

    // ----- OPSTE -----

    /**
     * Resetovanje objekta
     */
    public void clear();

    /**
     * Ispisivanje tabele simbola
     */
    public void printST();
}

```

Potrebno je implementirati odgovarajuće metode za radjene stavke domaćeg.

**Napomena:** potrebno je prilikom pokretanja glavnog programa praseru izvršiti resetovanje objekta koji sakuplja rezultate izvršavanja.

## Dodatak C – MJ LALR(1) Gramatika

```

Program    = "class" ident DeclarationList "{" MethodDeclarationList "}"
           | "class" ident "{" MethodDeclarationList "}"
           | "class" ident "{" "}"
           | "class" ident DeclarationList "{" "}".

```

$$\text{DeclarationList} = \text{DeclarationList DeclarationPart} \mid \text{DeclarationPart}.$$

```
DeclarationPart = ConstDecl ";"
                | VarDecl ";"
                | ClassDecl.
```

ConstDecl = "final" Type ident "=" Rhs.

Type = ident.

```
Rhs = number
      | charConst.
```

$$\text{VarDecl} = \text{Type VarList.}$$

```
VarList = VarList "," VarPart
          | VarPart.
```

```
VarPart = ident
         | ident "[" "]"
```

```
ClassDecl = "class" ident "{" LocalFieldList "}"
           | "class" ident "{" "}".
```

```
LocalFieldList = LocalFieldList VarDecl ";"
                | VarDecl ";".
```

$$\text{MethodDeclarationList} = \text{MethodDeclarationList MethodDecl} \mid \text{MethodDecl}.$$

```
MethodDecl = ReturnType ident "(" FormPars ")" LocalFieldList "{" StmtList "}"
| ReturnType ident "(" FormPars ")" "{" StmtList "}"
| ReturnType ident "(" FormPars ")" LocalFieldList "{" "}"
| ReturnType ident "(" FormPars ")" "{" "}"
```

```

| ReturnType ident "(" ")" LocalFieldList "{" StmtList "}"
| ReturnType ident "(" ")" "{" StmtList "}"
| ReturnType ident "(" ")" LocalFieldList "{" "}"
| ReturnType ident "(" ")" "{" "}".

```

```

ReturnType = Type
| "void".

```

```

FormPars = Parameter ParameterList
| Parameter.

```

```

Parameter = Type ident
| Type ident "[" "]"".

```

```

ParameterList = ParameterList "," Parameter
| "," Parameter.

```

```

StmtList = StmtList 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 ")" ";"
| "{" "}"
| "{" StmtList "}"
| "if" "(" Condition ")" Matched "else" Matched
| "while" "(" Condition ")" Matched.

```

```

Designator = IdentExprList.

```

```

IdentExprList = IdentExprList "[" Expr "]"

```

```

        | ident
        | IdentExprList "." ident.

ActPars = ExprList.

ExprList = ExprList "," Expr
        | Expr.

Expr = TermList.

TermList = TermList Addop Term
        | Term.

Term = FactorList.
FactorList = FactorList Mulop Factor
        | Factor
        | "-" Factor.

Factor = Designator "(" ActPars ")"
        | Designator
        | Designator "(" ")"
        | number
        | charConst
        | "new" Type
        | "new" Type "[" Expr "]"
        | "(" Expr ")".

Condition = CondTerm OrCondTermList
        | CondTerm.

OrCondTermList = OrCondTermList "||" CondTerm
        | "||" CondTerm.

CondTerm = CondFact AndCondFactList
        | CondFact.

AndCondFactList = AndCondFactList "&&" CondFact
        | "&&" CondFact.

CondFact = Expr Relop Expr.

Addop = "+"
        | "-".

Mulop = "*"
        | "/".

```

		"%".
Relop	=	"=="
		"!="
		">"
		"<"
		"<="
		">=".

## **Dodatak E – Saveti za izradu domaćeg**

### **Pojava Shift/Reduce i Reduce/Reduce konflikata**

Kada u CUP specifikaciju, na gramatiku koja je provereno ispravna, dodajemo nove delove sa akcijama, može doći do Shift/Reduce i Reduce/Reduce konflikata. Npr. konflikt se može javiti kod smene:

```
class_decl ::= CLASS prog_id:class_name { : Tab.insert(class_name); : } LBRACE RBRACE
           |
           CLASS prog_id:class_name { : Tab.insert(class_name); : } LBRACE local_field_list
           RBRACE;
```

prog\_id ::= IDENT;

Dva moguća rešenja su:

#### **Rešenje 1.**

Problem se može rešiti tako što se akcija ukloni iz smene u kojoj pravi konflikt i "premesti" u smenu neterminala koji je ispred nje. Primer za ovu transformaciju je:

```
class_decl ::= CLASS prog_id LBRACE RBRACE
           |
           CLASS prog_id LBRACE local_field_list RBRACE;

prog_id ::= IDENT:class_name { : Tab.insert(class_name); : };
```

#### **Rešenje 2.**

Uvesti novi neterminal koji sadrži samo akciju koja stvara konflikt. Primer ovog rešenja je:

```
class_decl ::= CLASS prog_id action1 LBRACE RBRACE
           |
           CLASS prog_id action1 LBRACE local_field_list RBRACE;

prog_id ::= IDENT;

action1 ::= { : Tab.insert("Problem: Ovde nemamo informaciju o class_name"); : };
```

Ovo rešenje se može primeniti samo ako akcija NE zavisi od sintetizovanih atributa neterminala u originalnoj smeni (u ovom primeru od prog\_id:class\_name). Ako akcija zavisi

od njih (što je čest slučaj), onda se i akcija i neterminali od kojih zavisi stavljaju u istu smenu kao u rešenju 1.

### **Transformisanje izraza**

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:

$\text{ParameterList} = \text{ParameterList Parameter} \mid \text{Parameter}.$

ParameterList je 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:

$\text{ParameterList} = \text{ParameterList ParameterPart} \mid \text{ParameterPart}.$

$\text{ParameterPart} = \text{Parameter1} \mid \text{Parameter2} \mid \text{Parameter3} \mid \dots$

Parameter1, Parameter2, ... su 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:

$\text{Funkcija} = \text{ImeFunkcije "(" ParameterList ")" } \mid \text{ImeFunkcije "(" ")}.$

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

$\text{Funkcija} = \text{ImeFunkcije "(" ParameterList ")" } \mid \text{ImeFunkcije "(" ")}.$

$\text{ParameterList} = \text{ParameterList Parameter} \mid \text{Parameter}.$

U prikazanoj smeni argumenti funkcije se mogu pojaviti jednom ili više puta ali i ne moraju.



## Dodatak F – Test primeri

### Primer1:

```
class C
{
    void main()
    {
        int a, b, c;
        {
            print('h');
            print('e');
            print('l');
            print('l');
            print('o');

            a = 3;
            b = 6;
            c = a + b;
            print(c);
        }
    }
}
```

### Primer2:

```
class C

    class Stack {
        char id;
        int stack[];
        int sp; // pokazuje na prvu slobodnu
    }

    final int size = 10;

    Stack stack;

{

    void push(Stack s, int n)
    {
        s.stack[s.sp] = n;
        s.sp++;
    }

    int pop(Stack s)
    {
        int val;
        {
            s.sp--;
            val = s.stack[s.sp];
        }
    }
}
```

```

    return val;
}

Stack createStack(char id)
{
    Stack s;
    {
        s = new Stack;
        s.stack = new int[size];
        s.id = id;
        s.sp = 0;
        return s;
    }
}

void printStack(Stack s)
{
    int i;
    {
        i = 0;
        while (i < len(s.stack)) {
            print(s.stack[i]);
            i++;
        }
    }
}

void main()
{
    int i;
    {
        stack = createStack('w');
        push(stack, 5);
        push(stack, 10);
        push(stack, 22);
        push(stack, 30);

        printStack(stack);

        pop(stack);
        pop(stack);

        printStack(stack);
    }
}

```