IR4PP1

Rok za predaju(jun): **16.06.2008.**
Rok za predaju(april): **05.05.2008.**

Domaći zadatak/prvi deo
Junski rok 2007/2008

Leksička i sintaksna analiza MikroJava programa

1. (8)

Potrebno je napraviti CUP kompatibilan leksički analizator modifikovanih MikroJava fajlova. Leksički analizator prihvata tekstualni fajl sa ekstenzijom .mmj u kojem se nalazi izvorni kôd koji odgovara specifikaciji MikroJava jezika (http://ir4pp1.etf.bg.ac.yu/Domaci/mikrojava.rar) uz odredjene 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 JLex. Koristiti JLex dat na sajtu predmeta (http://ir4pp1.etf.bg.ac.yu/Domaci/JLex.rar).
Sve modifikacije MikroJave koje treba uraditi mogu se pogledati u dodatku Dodatak Modifikacije MJ !
Prilikom leksičke analize potrebno je obezbediti brojanje sledećih pojavljivanja

- (0,5) broj dokumentacionih komentara u ulaznom fajlu
- (0,5) broj komentara u više redova u ulaznom fajlu
- (1) broj pojavljivanja @author unutar dokumentacionog komentara (više @author može biti u istom redu; nema ograničenja gde se @author može naći unutar dokumentacionog komentara).
- (1) brojati pojavljivanje @see unutar dokumentacionog komentara (više @see može biti u istom red; nema ograničenja gde se @see može naći unutar dokumentacionog komentara).

Prilikom nailaska na sledeće konstrukcije treba pamtiti stringove

- (2) prilikom nailaska na @author potrebno je zapamtiti sve do kraja reda. Eventualne blanko znake pre i posle teksta koji sledi nakon @author treba ukloniti. (primer: /* neki tekst @author ja sam autor */ ; treba da vrati string “ja sam autor”.
- (3) prilikom nailaska na @see potrebno je zapamtiti prvu reč (niz ne belih karaktera). Bele karaktera treba zanemariti. (primer /* nesto @see vidi drugi tekst */ ; treba da vrati string “vidi”.

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 Lexer ZAHTEV!

2. (12)

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 MikroJave uz dodatne neterminale. Kao osnova za pisanje gramatike može se koristiti gramatika data u Dodatku Gramatika (potrebno je dopuniti gramatiku odgovarajućim izmenama navedenim u Dodatku Modifikacije MJ), 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 alata CUP (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 sintaksnoj analizi.

02) Ne koristiti opciju precedence u CUP fajlu.

03) Kako je potrebno izvršiti modifikacije Mikro Jave (pogledati Dodatak Modifikacije MJ) potrebno je prebrojati sledece:

- (1) broj pojavljivanja for petlje
- (0,5) broj enum neredbi
- (0,5) broj konstanti unutar enum naredbe (poslednje enum navedene u programu)

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 Parser ZAHTEV!

04) Pri prepoznavanju određenih elemenata ulaznog fajla treba prebrojati sledeće (ukoliko neki od zahteva nije jasan pogledati Dodatak Objašnjenje):

(RADE STUDENTI SA NEPARNIM BROJEM INDEKSA)

- (1) Pojavljivanje if else selekcije (if sa else delom!)
- (1) Operator * (množenje)
- (1) Operator – (oduzimanje)
- (1) logičkih operatora (|| i &&)
- (1) broj klasa koje poseduju > od 5 polja (pri čemu mora da važi da je broj polja (koja su niz) uvek manji od broj polja ne niz tipa)
- (1) metoda koje imaju barem 1 argument i ne više od 3 lokalne promenljive tipa niza

(RADE STUDENTI SA PARNIM BROJEM INDEKSA)

- (1) while petlje koja u telu ima barem jedan if (moze biti if ili if else)
- (1) operator + (sabiranje)
- (1) lokalne promenljive kojima prethode dve promenljive (niz tipa) a same mogu ali ne moraju biti uvedene kao niz (svaka metoda posmatra se zasebno).
- (1) broj unutrašnjih klasa koje imaju jednak broj polja nizovskog i ne niz tipa
- (1) izraza dodele nakon read metode (odmah nakon read)
- (1) broj new operatora

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 Parser ZAHTEV!

05) U slučaju nailaska na sintaksnu grešku analizator uvećava odgovarajuci 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 globalnih promenljivih. Ignorišu se svi karakteri do prvog znaka ;
- (1) Nekorektno zadat izraz za indeksiranje prilikom pristupa elementu niza
- (1) Neispravno zadat uslov u okviru while petlje ignorišu se svi karakteri do) ili {
- (1) Neispravno zadat uslov u okviru if upravljačke konstrukcije ignorišu se svi karakteri do) ili {

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 Parser ZAHTEV!

05) Ime klase koja predstavlja parser ne treba menjati, već to treba da ostane `parser`. Očekuje se da se napravljeni parser pokreće na sledeći način:
`java ime_paketa.parser imeMMJfajla.mmj`

`ime_paketa` predstavlja ime paketa u kojem bi trebalo da se nalaze napravljeni parser, kao i leksički analizator iz tačke 1. Forma tog imena će biti definisana u delu Napomene. `imeMMJfajla.mmj` je izvorni MikroJava fajl koji predajemo parseru na analizu.

Takodje potrebno je kreirati klasu koja implementira sledeći interfejs:

```
package domaci.jun2008;

public interface IMain {
      /*
       * Prvi argument je ime fajla koji treba obraditi.
       * Treba predvideti da se main metoda moze pozivati
       * vise puta tokom izvrsavanja (obezbediti da se prilkom
       * svakog poziva ponasa kao da je prvi put pozvana
       */
    public ResultCollector main(String[] args);
}
```

Kreirana klasa treba da prihvati niz argumenata (args) pokrene parser i kao rezultat vrati instancu podklase klase ResultCollector.

Napomene

Slanje domaćeg

Ime paketa u kojem treba da se nalaze leksički analizator i parser je oblika xbbbgg, gde x predstavlja p za studente sa parnim brojem indeksa, a n za studente sa neparnim brojem indeksa; bbb je broj indeksa; gg je godina upisa. Svi dodatno uvedeni java fajlovi moraju biti u istom paketu!

Folder xbbbgg u kojem se nalaze .lex fajl, .cup fajl, drugi java fajlovi neophodne za funkcionisanje leksičkog analizatora treba zapakovati u zip arhivu i poslati od 14.06.2008 do 16.06.2008(24:00) (april: 04.05 ili 05.05) na adresu compilers.etf@gmail.com. (ostale fajlove ne treba slati!).

U subject-u navesti xbbbgg po ranije opisanom formatu, zatim ime i prezime tim redom. Treba slati samo jedan mejl. Višestruko slanje se neće priznati.

U telu navesti stavke koje su radjene a donose dodatne poene (ukoliko istih ima).

Prijavljivanje za odbranu domaćeg će se vršiti na fakultetu od 17.06.2008 (april: 07.05.2008) na način koji će biti naknadno objavljen preko mejling liste.

Osnovne napomene

Svi izvorni fajlovi, osim onih koje automatski generišu alati, kao i MikroJava test primeri MORAJU BITI DETALJNO KOMENTARISANI.

Nepoštovanje nekog od navedenih pravila povlači negativne poene.

Broj poena

Broj poena koji nosi domaći zadatak je 40 (ispred svake stavke naveden je broj poena, koji nosi, unutar zagrada “()”).

Dodatak Gramatika – Primer LALR(1) gramatike za MikroJavu

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

DeclarationList    = DeclarationList DeclarationPart
                   | DeclarationPart.
```

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

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

Type  =  ident.

Rhs  =  number
      |   charConst.

VarDecl  =  Type VarList.

VarList  =  VarList "," VarPart
          |   VarPart.

VarPart  =  ident
          |   ident "[" "]".

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

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

MethodDeclarationList  =  MethodDeclarationList MethodDecl
                        |   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" "(" Codition ")" 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 B – 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:

ParameterList = ParameterList Parameter | 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:

ParameterList = ParameterList ParameterPart | ParameterPart.
ParameterPart = Parameter1 | Parameter2 | Parameter3 | ...

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:

Funkcija = ImeFunkcije "(" ParameterList ")" | 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.

Funkcija = ImeFunkcije "(" ParameterList ")" | ImeFunkcije "(" ")".
ParameterList = ParameterList Parameter | Parameter.

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

Dodatak Modifikacije MJ

Modifikacija MikroJava jezika sastoji se u sledećem:

- Pored postojećih komentara u jednoj liniji (//) potrebno je omogućiti i pisanje komentara u više linija (/* */) kao i dokumentacionih komentara (/** */).
- Potrebno je dodati for petlju po definiciji jezika C
- Potrebno je dodati enum nabrajanje po definiciji jezika C

Nabrojane konstante (enum):

```
enum enum_ime { ime_konstante = vrednost, ... }
```

For petlja:

```
for (izraz; uslov; izraz) naredba
```

Napomena: nabrojane konstante i for petlja je po definiciji jezika C !

Dodatak Lexer ZAHTEV! && Parser ZAHTEV!

```
package domaci.jun2008;

public abstract class ResultCollector {
  private static int NOT_IMPLEMENTED = -1;


  // ---------------- LEXER --------------------
  /**
   *
   * @return broj pronadjenih dokumentacionih komentara
   * u ulaznom fajlu
   */
  public int docComments() { return NOT_IMPLEMENTED; }

  /**
   *
   * @return broj viserednih komentara u ulaznom fajlu
   */
  public int mulComments() { return NOT_IMPLEMENTED; }

  /**
   *
   * @return broj pronadjenih author
   */
  public int author() { return NOT_IMPLEMENTED; }

  /**
   *
   * @return broj pronadjenih see
   */
  public int see() { return NOT_IMPLEMENTED; }

  /**
   *
   * @return niz stringova pronadjnih uz author
   * u redosledu pojavljivanja
   */
  public String[] authorText() { return null; }

  /**
   *
   * @return niz stringova pronadjenih uz see
   * u redosledu pojavljivanja
   */
  public String[] seeText() { return null; }
```

```java
// -------------- PARSER ------------------
/**
 *
 * @return broj for petlji
 */
public int forLoop() { return NOT_IMPLEMENTED; }

/**
 *
 * @return broj pronadjenih enum nabrajanja
 */
public int enumCount() { return NOT_IMPLEMENTED; }

/**
 *
 * @return broj konstanti u poslednje vidjenom
 * enum nabrajanju
 */
public int lastEnum() { return NOT_IMPLEMENTED; }

// parni indeks
/**
 *
 * @return broj while petlji koje imaju barem jedan
 * if (bilo if bilo if else)
 */
public int whileIf() { return NOT_IMPLEMENTED; }

/**
 *
 * @return  broj operatora +
 */
public int sum() { return NOT_IMPLEMENTED; }

/**
 *
 * @return broj promenljivih kojima prethode dve
 * promenljive uvedene kao niz
 */
public int specLoc() { return NOT_IMPLEMENTED; }

/**
 *
 * @return broj klasa koje poseduju isti broj
 * polja uvedenih kao niz i ne niz
 */
public int classEquals() { return NOT_IMPLEMENTED; }

/**
 *
 * @return broj izraza dodele nakon read metode
 */
public int assignRead() { return NOT_IMPLEMENTED; }

/**
```

```
     *
     * @return broj operatora new
     */
    public int newCount() { return NOT_IMPLEMENTED; }

    // neparni indeks
    /**
     *
     * @return broj pronadjenih if sa else delom
     */
    public int ifWithElse() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj pronadjenih operatora *
     */
    public int mul() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj pronadjenih operatora -
     */
    public int sub() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj pronadjenih logickih operatora && i ||
     */
    public int logic() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj klasa koje poseduju vise od 5 polja
     * pri cemu klasa poseduje vise ne niz polja
     */
    public int specClass() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj metoda koje imaju vise od 1 argumenta
     * i ne vise od 3 lokalne promenljive (koje su niz)
     */
    public int specMeth() { return NOT_IMPLEMENTED; }

    // ------------ GRESKE ---------------
    /**
     *
     * @return broj gresaka u slucaju definicije globalne promenljive
     */
    public int globErr() { return NOT_IMPLEMENTED; }

    /**
     *
     * @return broj gresaka u slucaju indeksiranja niza
     */
    public int indexErr() { return NOT_IMPLEMENTED; }
```

```
  /**
   *
   * @return broj gresaka nekorektnog uslova while petlje
   */
  public int whileErr() { return NOT_IMPLEMENTED; }

  /**
   *
   * @return broj gresaka nekorektnog uslova if petlje
   */
  public int ifErr() { return NOT_IMPLEMENTED; }


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

  /**
   * Resetovanje objekta
   */
  public void clear() {}

}
```

Potrebno je implementirati odgovarajuće metode za radjene stavke domaćeg.
Napomena: potrebno je prilikom pokretanja glavnog programa prasera izvršiti resetovanje objekta koji sakuplja rezultate izvršavanja.

Dodatak Objašnjenje – primer zahteva tačke 2.04

Broj klasa koje poseduju > od 5 polja (pri čemu mora da važi da je broj polja (koja su niz) uvek manji od broj polja ne niz tipa):

```
class klasa1 {
  int a, b, c, d, e, f;
}

class klasa2 {
  int a, b, c, d[], e[], f[];
}
```

Klasa klasa1 zadovoljava uslov dok klasa2 ne zadovoljava postavljeni uslov.

Metode koje imaju barem 1 argument i ne više od 3 lokalne promenljive nizovskog tipa:

```
void m1(int a) int a, b, c, d; { }
void m2() {}
void m3(int b) int a[], b[], c, d[]; {}
void m4(int c) int a[], b[], c[], d[]; {}
```

Metode m1 i m3 zadovoljavaju ovaj uslov dok druge dve ne zadovoljavaju.

Lokalne promenljive kojima prethode dve promenljive (niz tipa) a same mogu ali ne moraju biti uvedene kao niz:

```
void m3(int b) int a[], b[], c, d[]; {}
void m4(int c) int a[], b[], c[], d[]; {}
```

U slučaju metode m3 to je promenljiva c a u slučaju metode m4 to su promenljive c i d.

Broj klasa koje imaju jednak broj polja nizovskog i ne niz tipa:

```
class klasa1 {
  int a, b, c, d, e, f;
}

class klasa2 {
  int a, b, c, d[], e[], f[];
}
```

Klasa klasa1 ne zadovoljava postavljeni uslov dok klasa klasa2 isti zadovoljava.

Izraza dodele nakon read metode:

```
void m3(int b) int a[], b[], c, d[]; {
  read(a);
  b = a;  // ovaj izraz se broji
  c = e;  // ovaj izraz se ne broji

  read(l);
  a++
  s = l;  // ovaj izraz se ne broji
}
```