Programski prevodioci 1 Rok za predaju rešenja: biće objavljen na listi predmeta
Domaći zadatak 4 v1.1

Kompajler za Mikrojavu

Uvod

Potrebno je realizovati kompajler za mikrojavu koji omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajt kod koji može da se izvršava na Mikrojava virtuelnoj mašini. Uvodni deo sadrži informacije o izmenama u standardnoj specifikaciji Mikrojave, kao i opis mogućih varijanti izrade domaćeg zadatka. Gramatika za Mikrojavu sa je navedena u prilogu datog projektnog zadatka i student je može koristiti prilikom realizacije kompajlera.

Izmene u specifikaciji Mikrojave

U specifikaciji Mikrojave načinjene su sledeće izmene:

1) Mikrojava program počinje sa: solution „ime_programa“ umesto „class“ ime_programa kao u standardnoj specifikaciji Mikrojave.
   Primer: `solution P1 {...}`
2) Za primitivni tip int se koristi ključna reč *integer*.
3) Za deklarisanje konstanti se umesto ključne reči *final* koristi ključna reč *constant*.
   Primer: `constant integer x=0,z=4,y=6;`
4) Definicija unutrašnje klase počinje ključnom rečju *struct* umesto ključne reči *class*. U nastavku ovog dokumenta, za unutrašnje klase biće korišćen i termin strukture.
5) Uvodi se nove kontrolne strukture i naredbe, koje su opisane u nastavku:
   a. **Foreach petlja.**
      Foreach petlja kontrolna struktura koja predstavlja kompaktni zapis za iteriranje kroz nizove. Sintaksa foreach petlje je prikazana/opisana na sledećem primeru:

```
foreach (a in array) {
   /* doSomethingWithElement(a); */
   /* i/ili ostali iskazi unutar tela petlje*/
}
```

      Semantika: telo petlje (koje nema break naredbu) izvršava se sve dok se obrade svi elementi niza array.
      Kontekstni uslovi: *array* mora biti niz. Vitičaste zagrade ne moraju da se navode ako se u telu petlje nalazi samo jedan iskaz. Niz sadrži elemente primitivnih tipova (integer, double, char).
   b. **Naredba continue.**
      Sintaksa: *continue;*
      Semantika: Posledica izvršavanja naredbe *continue* se ogleda u prekidanju izvršavanja naredbi iz tekuće iteracije petlje i započinjanje nove iteracije.
      Konktekstni uslovi: Greska je ako se *continue* naredba koristi izvan tela petlje.

Varijante izrade domaćeg zadatka

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.

I Leksička analiza (4 poena)

Potrebno je realizovati CUP kompatibilan leksički analizator (skener) Mikrojava fajlova. Skener prihvata fajl u kome se nalazi izvorni kod koji odgovara specifikaciji modifikovane Mikrojave (u nastavku MJ*) i deli ga na tokene. Tip tokena se vraća pri eksplicitnom pozivu leksičkog analizatora (poziv next_token() na Javi, ili yylex() na c++u).

II 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 nastavku zadatka) i na kraju poruku o uspešnom parsiranju (videti Prilog 3). 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 sintaksnoj analizi.

Sintaksa MJ* fajla je data u specifikaciji modifikovane Mikrojave, a jedan predlog LALR(1) gramatike MJ*, koja se može ali i ne mora koristiti u Prilogu 2. Predlog gramatike ne mora biti kompletan, pa se stoga od studenata zahteva njegovo kompletiranje. 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 MJ*:
        i. Deklaracije struktura (unutrašnjih klasa)
        ii. Metode glavnog programa
        iii. Deklaracije globalnih promenljivih prostog tipa
        iv. Deklaracije globalnih konstanti
        v. Deklaracije globanih nizova
        vi. Deklaracije lokalnih promenljivih u okviru tela main metoda
        vii. Iskazi u okviru tela main metoda (osim poziva funkcija)
        viii. Poziva funkcija u okviru tela main metoda.
        ix. Deklaracije polja struktura

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. Izraza dodele => ignorisati karaktere do ;
        ii. Iteracionog izraza u foreach petlji – ignorisati karakt. do prve)
        iii. Logičkog izraza unutar if konstrukcije => ignorisati karaktere do)
        iv. Parametara u pozivu funkcije => ignorisati karaktere do)
        v. Izraza u indeksu => ignorisati karaktere do]
        vi. Uslova u while petlji - ignorisati karakt. do prve)
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.

III 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 treba da bude organizovana kao jedinstvena tabela, a ne u vidu posebnih tabela za opsege. Na sajtu predmeta studenti mogu pronaći postojeće implementacije tabele simbola, ali one nisu jedinstvene tabele simbola. Medjutim, date implementacije treba iskoristiti samo u tom smislu da interfejs tabele simbola ostane nepromenjen u odnosu na onaj koji je pokazan na vežbama. Prilikom realizacije projektnog zadatka od studenta se očekuje implementacija jedinstvene tabele simbola.
    i. Studenti koji koriste Javu i 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 glavog programa i nazive predefinisanih 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. Dodavanjem semantičkih akcija u parser potrebno je omogućiti unos simbola u tabelu simbola i detektovati korišćenje simbola po sledećim pravilima:

i. Potrebno je obraditi deklaracije programa, konstanti, globalnih i lokalnih promenljivih (prostog tipa i nizova) i globalnih metoda (članova programa) i unutrašnjih klasa (struktura).
ii. Potrebno je detektovati svako naknadno korišćenje pomenutih objekata, proveriti da li objekat postoji u tabeli simbola i da li je ispravnog tipa. Za metode treba proveravati formalne parametre. **Ispisati odgovarajuću poruku na standarni izlaz (vidi Prilog 3 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] ";"
| "print" "(" Expr ["," number] ")" ";"
| "{" {Statement} "}"
| "foreach" "(" ident "in" ident ")" "{" ... "}"
| "continue"
| "while" "(" Condition ")" "{" Statement "}"
```

1. Uslovni izrazi (Condition) mogu biti samo oblika CondFact.
2. Obraditi stvarne parametre u pozivu funkcija.
3. Sintaksni element pod nazivom Designator može se odnositi na promenljivu prostog tipa, na niz ili na polje unutrašnje klase.
4. Izrazi (expr) treba da se obrade kompletno (izostaviti samo pravljenje instanci klase sa operatorom new).

iv. Prilikom zatvaranja opsega, treba ispisati sve simbole koji pripadaju datom opsegu. Primeri izlaza navedeni su u Prilogu 3 više kao ilustracija, ali za jedinstvenu tabelu simbola nisu relevantni.

IV 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).

Napomeni u vezi izrade, predaje i odbrane rešenja

Rok za predaju rešenja će biti naknadno objavljen na mejling listi predmeta. Rešenje se predaje u elektronskoj formi, zapakovano u .rar ili .zip arhivu, emailom na adresu compilers.etf@gmail.com, **subject**: solution_nov2010_Ime_Prezime_GGGG_BBBB. Potrebno je strogo ispoštovati zadati format naslova (subject) mejla, jer se mejlovi primaju i arhiviraju automatski. Ukoliko se ne ispoštuje zadati format naslova, mejl se ignoriše! Zabranjeno je slati više verzija istog rešenja. Ako vam se email vrati jer gmail odbija arhivu (pod sumnjom na viruse), zapakujte sve pod šifrom ABCD i ponovo pošaljite.
Eventualna pitanja i komentare u vezi domaćeg ne upućujte na navedenu adresu, već isključivo na mailing listu predmeta ir4pp1@rti.etf.rs odnosno si4pp1@rti.etf.rs (prethodno treba da budete prijavljeni na mailing listu).

Elementi rešenja su sledeći:
1. Propratna 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**.

Napomene u vezi izrade i odbrane 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.

Prilog 1

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 modifikovane 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 deklarisati 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:

Parameter_list → Parameter_list Parameter | 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 objektat pojavljuje jednom ili više puta može se koristiti sledeći oblik smene:

Parameter_list → Parameter_list Parameter_part | Parameter_part
Parameter_part → Parameter1 | Parameter2 | Parameter3 | ...

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:

Funkcija → ImeFunkcije (Parameter_list) | 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.

Funkcija → Ime (Parameter_list) | Ime ( )
Parameter_list → Parameter_list Parameter | Parameter

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

Funkcija → Ime (Parameter_list)
Parameter_list → Parameter_list Parameter | /* epsilon */

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

Primer LALR(1) gramatike za MikroJavu

Primer gramatike je dat u listingu 1. Jedini specijalni simboli su → (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).
Data gramatika je standardna uJava gramatika i ne sadrži zahtevane izmene, koje su navedene na početku ovog dokumenta. Studenti zato treba pažljivo da koriste datu gramatiku tokom izrade domaćeg.

```
Program →       CLASS IDENTIFIKATOR declaration_list
                {
                        method_declaration_list
                }
                |
                CLASS IDENTIFIKATOR
                {
                        method_declaration_list
```

```
                    }
                    CLASS IDENTIFIKATOR declaration_list
                            {                       }
                    |
                    CLASS IDENTIFIKATOR
                            {                       }


declaration_list →  declaration_list declaration_part
                    |
                    declaration_part

declaration_part →  ConstDecl ;
                    |
                    VarDecl ;
                    |
                    ClassDecl

Type        →       IDENTIFIKATOR

rhs         →       NUMBER      |  CHAR_CONST

ConstDecl   →       FINAL Type IDENTIFIKATOR = rhs

// VarDecl predstavlja deklaraciju grupe promenljivih
// Sastoji se od tipa promenljivih i liste imena promenljivih odvojenih zarezima
VarDecl     →       Type var_list

var_list    →       var_list , var_part     |       var_part

var_part →  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
| FOR (IDENTIFIKATOR=StartVal TO EndVal) Statement
| FOR (IDENTIFIKATOR=StartVal DOWNTO EndVal) Statement

StartVal → Expr

EndVal → Expr

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 ORCondTerm_list
|
CondTerm

ORCondTerm_list → ORCondTerm_list OR CondTerm
|
OR CondTerm

CondTerm → CondFact ANDCondFact_list
|
CondFact

ANDCondFact_list → ANDCondFact_list && CondFact
|
&& CondFact

CondFact → Expr Relop Expr

Napomena: produkcija za switch kontrolnu strukturu je sa namerom izostavljena. Studenti sami treba da osmisle datu produkciju.

Listing 1. Primer gramatike za modifikovanu MikroJavu.

Prilog 3. Primeri izlaza

Ulazni program:

```
program P
	const int size = 10;
	int g[];
	{
		void main()
			int x, i;
			char x;
		{ //---------- Initialize val
			x.i=1;
			g = new int[size];
			while(i<size) {
				pos[i++] = 0;
			}

			//---------- Read values
			x=5; i=1;
			while(i!=x) {
				g[i]=i*g[i-1];
				i++;
				print(g[i]); print(‘,’);
			}

		}
	}
```

Skica izlaza kompajlera za navedeni program je data u nastavku (dobijeni izlaz je isključivo ilustracija mogućeg izlaza kompajlera i nije dobijena propuštanjem datog listinga kroz kompajler). Prijave grešaka (prikazano podebljano) treba da idu na standardni izlaz greške, ostalo na standarni izlaz.

```
==================SEMANTICKA OBRADA====================
Greska na 7: x vec deklarisano
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 15(x), nadjeno Var x: int, 0, 1
Pretraga na 16(x), nadjeno Var x: int, 0, 1
Pretraga na 17(x), nadjeno Var x: int, 0, 1
Pretraga na 17(size), nadjeno Con size: int, 10, 1
Pretraga na 18(pos), nadjeno Var pos: Arr of int, 0, 1
Pretraga na 18(x), nadjeno Var x: int, 0, 1
Pretraga na 20(x), nadjeno Var x: int, 0, 1
==================SINTAKSNA ANALIZA====================
0     classes
1     methods in the program
0     global variables
1     global constants
1     global arrays
2     local variables in main
10    statements in main
2     function calls in main
==============SADRZAJ TABELE SIMBOLA==================
```

Tabela simbola treba da bude odstampana isključivo prema zahtevima u postavci domaćeg zadatka. Studentima se daje sloboda da osmisle relevantni ispis tabele simbola prema uputstvima. Tabela simbola prikazana u nastavku je opet data kao ilustracija na osnovu koje se može osmisliti format ispisa 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 ]]
```