Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

*Predmet:* Programski prevodioci 1
*Nastavnik:* doc. dr Dragan Bojić
*Asistent:* Nemanja Kojić
*Školska:* 2014/2015.
*Ispitni rok:* Januarski ispitni rok
*Datum:* 06.10.2014.

# 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, može se koristiti diskusiona lista za razjašnjavanje nejasnoća u zahtevima, van onoga sto se može samostalno rešiti uvođenjem razumnih pretpostavki. **Srećan rad!**

# 1. Uvod

Cilj projektnog zadatka je realizacija kompajlera za programski jezik Mikrojavu. Kompajler omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvršava na virtuelnoj mašini za Mikrojavu. Sintaksno i semantički ispravni Mikrojava programi su definisani specifikacijom.

Programski prevodilac za Mikrojavu ima četiri osnovne funkcionalnosti: leksičku analizu, sintaksnu analizu, semantičku analizu i generisanje koda.

*Leksički analizator* treba da prepoznaje jezičke lekseme i vrati skup tokena izdvojenih iz izvornog koda, koji se dalje razmatraju u okviru sintaksne analize. Ukoliko se tokom leksičke analize detektuje leksička greška, potrebno je ispisati odgovarajuću poruku na izlaz.

*Sintaksni analizator* ima zadatak da utvrdi da li izdvojeni tokeni iz izvornog koda programa mogu formiraju gramatički ispravne sentence. Tokom parsiranja Mikrojava programa potrebno je na odgovarajući način omogućiti i praćenje samog procesa parsiranja na način koji će biti u nastavku dokumenta detaljno opisan. Nakon parsiranja sintaksno ispravnih Mikrojava programa potrebno je obavestiti korisnika o uspešnosti parsiranja. Ukoliko izvorni kod ima sintaksne greške, potrebno je izdati adekvatno objašnjenje o detektovanoj sintaksnoj grešci, izvršiti oporavak i nastaviti parsiranje.

*Semantički analizator* se dobija proširenjem funkcionalnosti sintaksnog analizatora. Semantička analiza se vrši sprovodi kroz sintaksno-upravljano prevođenje. Osnovnoj gramatici, kojom je specificiran sintaksni analizator, dodaju se atributi i akcije (atributivno-translaciona gramatika).

*Generator koda* prevodi sintaksno i semantički ispravne programe u izvršni oblik za odabrano izvršno okruženje Mikrojava VM. Generisanje koda se implementira na sličan način kao i semantička analiza, kroz nadogradnju sintaksnog analizatora (sintaksno upravljano prevođenje).

Svi relevantni pomoćni materijali za izradu projekta se mogu pronaći na sajtu predmeta ili u okviru sekcije Prilog ovog dokumenta.

# 2. Reference

[MJ] Specifikacija jezika Mikrojava prilagođena postavci zadatka, http://ir4pp1.etf.rs/Domaci/mikrojava_2014_2015_v1.1.pdf

# 3. Specifikacija projektnih zadataka

## I Leksička analiza

U nastavku teksta su navedeni i opisani projektni zahtevi za implementaciju leksičkog analizatora.

*Leksička analiza nosi maksimalno 5 poena.*

[R1] Potrebno je realizovati leksički analizator (*skener*) izvornih programa napisanih na jeziku Mikrojava.

[R2] Leksički analizator se mora implementirati na programskom jeziku Java.

[R3] Interfejs leksičkog analizatora prema sintaksnom analizatoru mora biti standardni CUP interfejs. Za više informacija, pogledati primer mini domaćeg u vežbama na sajtu predmeta.

[R4] Skener prihvata fajl za izvornim kodom na jeziku Mikrojava i deli ga na tokene.

[R5] Token se vraća eksplicitnim pozivom leksičkog analizatora (operacija `next_token()`).

[R6] Potrebno je detektovati i obraditi sledeće leksičke strukture:

[R7] – identifikatore,

[R8] – konstante,

[R9] – ključne reči,

[R10] – operatore,

[R11] – komentare.

[R12] Leksičke strukture implementirati prema specifikaciji jezika [MJ§A.2p3].

[R13] Leksički analizator treba da preskače komentare i "beline" u tekstu programa.

[R14] Pod "belinama" se smatraju: tabulatori (\t), prelazak u novi red (\r\n), razmak (' '), bekspejs (\b), prelazak na novu stranu (\f, form feed).

[R15] U slučaju leksičke greške analizator, ispisuje se greška i nastavlja se obrada teksta programa.

[R16] Poruka o grešci treba da sadrži sledeće informacije:

[R17] – niz znakova koji nije prepoznat,

[R18] – broj linije teksta programa u kojoj se desila greška, i

[R19] – kolonu (poziciju prvog znaka) u kojoj je detektovana greška.

[R20] Potrebno je napisati test program (*main*) koji testira rad leksičkog analizatora. Program poziva leksički analizator sve dok se ne stigne do kraja ulaznog fajla i ispisuje redom vraćene tokene na izlaz.

[R21] Test program bi trebalo da ima opciju da rezultat ispiše ili na standardni izlaz ili u zadati fajl.

[R22] Obavezno je korišćenje alata **JFlex**.

[R23] Moraju se koristiti jdk 1.7 i aktuelna verzije JFlex alata kao što je opisano u primerima na vežbama.

[R24] Korišćenje integrisanih okruženja (Eclipse ili Netbeans) se preporučuje, 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.

# II Sintaksna analiza

Potrebno je napisati LALR(1) gramatiku na osnovu specifikacije jezika i implementirati sintaksni analizator (*parser*) za programe napisane na jeziku Mikrojava.

*Sintaksna analiza nosi maksimalno 10 poena.*

*Oporavak od grešaka nosi maksimalno 5 od 10 poena.*

**Opšti tehnički zahtevi**

[R25] Gramatika jezika Mikrojava mora biti napisana u skladu sa specifikacijom jezika definisanom u [MJ].

[R26] Za implementaciju parsera mora se koristiti generator sintaksnih analizatora CUP.

[R27] Specifikacija parsera mora biti napisana u CUP fajlu, u formatu koji CUP generator prepoznaje.

[R28] Sintaksni analizator mora biti integrisan sa CUP kompatibilnim leksičkim analizatorom za jezik Mikrojava.

[R29] 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 2).

[R30] U slučaju nailaska na sintaksnu grešku parser prijavljuje poruku na standardni izlaz greške (System.err), vrši oporavak od greške i nastavlja sa parsiranjem ostatka fajla.

[R31] Parser treba da omogući oporavak od sintaksnih grešaka za zadate jezičke elemente.

[R32] Opis sintaksne greške TREBA da sadrži:

[R33] – broj linije ulaznog programa u kojoj je greška detektovana (videti realizaciju u primeru mini domaćeg sa sajta predmeta),

[R34] – opis greške.

**Implementacija parsera**

[R35] Opciju *precedence* u .cup fajlu je zabranjeno koristiti za definisanje prioriteta operatora.

[R36] Gramatika mora da sadrži neterminale koji su nazvani isto kao u odabranoj specifikaciji uz eventualne dodatne neterminale ukoliko se za tim ukaže potreba.

[R37] Generisana klasa parsera mora da ima metodu `main` koja pokreće parsiranje nad MJ fajlom.

[R38] Ime MJ fajla se zadaje isključivo preko komandne linije.

[R39] U parser se mora ugraditi prebrojavanje jezičkih elemenata. Potrebno je implementirati prebrojavanje sledećih elemenata:

[R40] – deklaracije globalnih promenljivih tipa char,

[R41] – deklaracije globalnih nizova,

[R42] – definicije funkcija u glavnom programu,

[R43] – definicije unutrašnjih klasa,

[R44] – blokovi naredbi,

[R45] – pozive funkcija u telu metode *main*,

[R46] – naredbe stvaranja objekata,

[R47] – definicije metoda unutrašnjih klasa,

[R48] – deklaracije polja unutrašnjih klasa,

[R49] – broj izvođenja klasa.

[R50] U gramatiku TREBA dodati smene i akcije za oporavak od grešaka. Implementirati oporavak od grešaka za sledeće jezičke elemente:

[R51] – definicija globalne promenljive – ignorisati karaktere do prvog znaka ";" ili sledećeg ","

[R52] – deklaracija lokalnih promenljivih – ignorisati karaktere do prvog ";" ili "{"

[R53] – konstrukcija iskaza dodele – ignorisati karaktere do ";"

[R54] – poziv funkcije - ignorisati karaktere do prvog ";"

[R55] – logičke izraze unutar if ili while konstrukcije - ignorisati karaktere do prvog znaka ")".

[R56] – liste parametara u pozivu funkcije – ignorisati karaktere do prvog znaka ")"

[R57] – izraza za indeksiranje niza – ignorisati karaktere do prvog znaka "]".
[R58] – deklaracija formalnog parametra funkcije – ignorisati znakove do znaka "," ili ")".
[R59] – deklaracija polja unutrašnje klase – ignorisati karaktere do prvog ";" ili "}"
[R60] – deklaracija proširenja natklase – ignorisati znakove do prvog znaka "{".

**Testiranja rada implementiranog parsera:**

[R61] Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve sintaksno ispravne kombinacije elemenata MJ gramatike (npr. funkcija bez formalnih parametara, sa jednim formalnim parametrom, sa više formalnih parametara).

[R62] Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve kombinacije grešaka koje su navedene u zahtevima [R51-R60].

# III Semantička analiza

Potrebno je proširiti sintaksni analizator iz prethodnog poglavlja da vrši semantičku analizu MJ programa i ažurira tabelu simbola.

*Semantička analiza se boduje sa maksimalno 10 poena. Funkcionalnosti su raspoređene po grupama zbog efikasnijeg bodovanja. Grupe su definisane u odeljku za generisanje koda.*

*Raspored bodova po grupama je: A (2), B(5), C(10).*

**Opšti zahtevi**

[R63] Semantička analiza se implementira proširenjem specifikacije sintaksnog analizatora (uvođenjem atributa neterminala i terminala, kao i akcija za implementaciju semantičkih provera).

[R64] Sintaksno-semantički analizator je potrebno integrisati sa tabelom simbola. Tabela simbola čuva informacije o simbolima pronađenim u MJ programu koji se prevodi.

[R65] Mora se koristiti implementacija tabele simbola dostupna na sajtu predmeta: http://ir4pp1.etf.rs/Domaci/symboltable.jar.

[R66] Tabela simbola se uvezuje sa ostatkom programa kao Java biblioteka (.jar) i dozovljeno je koristit sve njene javne klase, metode i polja. Uz biblioteku se dostavlja i izvorni kod koji je predviđen samo za čitanje i informisanje o detaljima implementacije.

[R67] NIJE DOZVOLJENO raspakivati biblioteku za tabelu simbola, menjati njenu implementaciju i ponovo je prevoditi i uvezivati sa projektom.

[R68] Ukoliko postojeća implementacija tabele simbola ne zadovoljava sve zahteve iz date specifikacije, može se nadograditi ISKLJUČIVO pomoću izvođenja klasa i redefinisanja postojećih metoda. Tabela simbola ima nekoliko tačaka za proširenja.

[R69] Ukoliko ni redefinisanje klasa ne doprinosi željenom ponašanju tabele simbola, a takav zaključak se MORA izvesti samo posle adekvatne diskusije na listi predmeta ili na časovima vežbi, upućuje se zahtev na listu predmeta i kontrolu preuzima predmetni asistent da u što kraćem roku prilagodi implementaciju tabele simbola potrebama realizacije domaćeg zadatka.

**Implementacioni zahtevi.**

[R70] Implementirati unos svih simbola u tabelu simbola dodavanjem semantičkih akcija u postojeći sintaksni analizator. Obrađuju se jezički elementi predviđeni u specifikaciji [MJ].

[R71] Implementirati detektovanje korišćenja simbola u programu dodavanjem semantičkih akcija u postojeći sintaksni analizator, i to za sledeće jezičke elemente:

[R72] – konstante

[R73] – globalne promenljive

[R74] – lokalne promenljive

[R75] – globalne funkcije (pozivi)

[R76] – unutrašnje klase

[R77] – polja unutrašnjih klasa

[R78] – metode unutrašnjih klasa (pozivi).

[R79] Za svaki detektovani simbol potrebno je proveriti sledeće:

[R80] – da li ime postoji u tabeli simbola,

[R81] – da li je ispravnog tipa.

[R82] Implementirati proveru svih kontekstnih uslova navedenih u specifikaciji [MJ§A.4p5] predviđenih za odabranu grupu funkcionalnosti.

[R83] Za svaku akciju semantičke analize potrebno je ispisati odgovarajuću poruku na standardni izlaz. Videti Prilog 3 za format poruke.

[R84] Poruka o detektovanom simbolu MORA da sadrži sledeće podatke:

[R85] – linija izvornog koda u kojoj je pronađen simbol,

[R86] – naziv pronađenog simbola,

[R87] – ispis objektnog čvora iz tabele simbola koji odgovara pronađenom simbolu.

[R88] Parser mora da ima javno dostupnu metodu `void dump()` za ispis sadržaja tabele simbola. Videti prilog 3.

[R89] U glavnom programu parsera metoda `dump` se poziva nakon završetka parsiranja.

**Testiranja rada implementiranog semantičkog analizatora:**

[R90] Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve sintaksno i semantički ispravne MJ programe uz pokrivanje svih smena iz gramatike.

[R91] Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve kombinacije semantičkih grešaka.

# IV Generisanje koda

Potrebno je proširiti sintaksno-semantički analizator iz prethodnog poglavlja da vrši generisanje bajtkoda za izvršno okruženje MJVM.

*Generisanje koda nosi maksimalno 15 poena. Zahtevi su po težini podeljeni u grupe.*

[R92] Generator koda mora da generiše ispravan bajtkod za MJVM:

[R93] Za implementaciju generatora koda moraju se koristiti alati `Code`, `disasm` i `Run`.
http://ir4pp1.etf.rs/Domaci/mj-runtime.jar

[R94] Izlaz generatora koda mora da bude izvršivi .obj fajl za MJVM.

[R95] **Za 4 poena** potrebno je implementirati generisanje koda za SVE gramatičke smene u nastavku (osnovni iskazi, aritmetički izrazi i rad sa nizovima prostih tipova):
Statement := Designator = Expt, ;
Statement := Designator++;
Statement := Designator--;
Statement := "read" "(" Designator ")" ";"
Statement := "print" "(" Expr [“,” number] ")" ";"
Expr := ["-"] Term {Addop Term}
Term := Factor {Mulop Factor}.
Factor := number | charConst | "(" Expr ")" | boolConst | "new" Type [ "[" Expr "]" ]
Designator := ident { "[" Expr "]" | "." ident }.
Addop := "+" | "-".
Mulop := "*" | "/" | "%".
Od nizova, treba podržati samo nizove ugrađenih tipova podataka, izuzev referenci na stringove. Program treba da ima samo jednu funkciju (`main`), globalne/lokalne promenljive (proste ili nizovne), globalne konstante. Ne treba obrađivati unutrašnje klase i globalne funkcije.

[R96] **Za 8 poena** potrebno je implmenentirati SVE zahteve u [R99] i još dodatno SVE gramatičke smene u nastavku (kontrolne strukture, uslovni izrazi, pozivi globalnih metoda, stringovi):
Statement := Designator "(" [ActPars] ")" ) ";"
Statement := "if" "(" Condition ")" Statement ["else" Statement ]
Statement := "while" "(" Condition ")" Statement
Statement := "break" “;”
Statement := "return" [Expr] “;”
Statement := "{" {Statement} "}".
ActPars := Expr {"," Expr }.
Condition := CondTerm { "||" CondTerm }.
CondTerm := CondFact { "&&" CondFact }.
CondFact = Expr [ Relop Expr ]. // samo jedan izraz (Expr) u slučaju bool promenljive
Factor := strConst
Factor := Designator [ "(" [ActPars] ")" ].

[R97] **Za svih 15 poena** potrebno je implementirati SVE zahteve navedene u [R101] i još dodatno zahteve u nastavku (unutrašnje klase, supstitucija i polimorfizam):
- implementirati nasleđivanje klasa;
- implementirati supstituciju (na mestu gde se očekuje referenca na osnovnu klasu može se predati referenca na objekat izvedene klase);
- implementirati pravljenje objekata unutrašnjih klasa;
- implementirati pravljenje tabela virtuelnih funkcija;
- implementirati polimorfno pozivanje metoda unutrašnjih klasa.

# V Ocenjivanje domaćih zadataka

**Opšti zahtevi**

[R98] Generisanje koda, koje proizvodi ispravne izvršne bajtkod fajlove, je POTREBAN uslov za izlazak na odbranu domaćeg zadatka.

[R99] Važeća specifikacija jezika se nalazi na sajtu predmeta, u odeljku za Domaći zadatak.

[R100] Ispunjenost zahteva se ocenjuje po principu "sve ili ništa".

[R101] Kada se odabere jedna od tri grupe funkcionalnosti (koje su jasno naznačene u sekciji za Generisanje koda), dobija se broj poena iz one grupe iz koje su ispunjeni SVI zahtevi. Polovično ispunjeni zahtevi se ne razmatraju.

[R102] Ukoliko student ne implementira SVE funkcionalnosti ni iz osnovne grupe,projekat neće biti razmatran na odbrani.

Primer programa:

```
program Program

	class A{
		int x[],y[];
	}

	const int pi = 3, e = 2;

	class B extends A{
		int i; int x[];
		int getValue(int a) int b; bool c;
		{ return this.i + this.x[0] + a; }
	}

	class C extends B{
		A theA;
		string a;
	}
{
	void main() A a; C c; int i; int x[];
	{
		a = new A;
		a.x = new int[5];
		a.y = new int[5];
		c = new C;
		c.theA = a;
		c.x = new int[5];
		x = new int[3];
		i=0;
		read(c.i);
		while(i<5) {
			read(c.x[i]); read(c.theA.x[i]);
			i++;
		}
		print(c.getValue(c.theA.x[0]));
	}
}
```

# 4. Napomene u vezi sa izradom i odbranom rešenja

Elementi rešenja su sledeći:

a) Propratna dokumentacija u obliku Word dokumenta MJProjekat.doc koji treba da se nalazi u korenom direktorijumu rešenja i da sadrži:
b) naslovnu stranu,
c) kratak opis postavke zadatka od nekoliko rečenica,
d) opis komandi za generisanje java koda alatima, prevođenje koda kompajlerom, pokretanje i testiranje rešenja,
e) kratak opis (UML model) 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).

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.

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 zadaci se rade individualno i brane usmeno pre ispita u prvom ispitnom roku. Uspešna odbrana projekta je uslov za izlazak na ispit. Ukoliko se na odbrani utvrdi **nedozvoljena** saradnja između studenata prilikom izrade domaćeg, u moguće posledice osim gubitka poena spada i trajno dobijanje negativnih poena koji će biti uključeni u zbir za konačnu ocenu.

Odbrana se organizuje posle roka predaje domaćeg, prema naknadnom obaveštenju. Radovi se predaju preko specijalne veb forme. Zadaci ne mogu da se brane na sopstvenom računaru.

Po potrebi će ulazni test fajlovi biti pokretani na odbrani domaćeg.

2. Na odbrani će, pored samog rešenja, biti proveravano i poznavanje rada sa alatima jflex, CUP.

**VAŽNO**

Studenti su dužni da svoje zadatke testiraju na računarima u laboratoriji pre dana odbrane, ukoliko nisu potpuno sigurni da im je rešenje prenosivo na drugi računar. Na odbrani se zadatak isključivo brani. Nije dozvoljena nikakva dorada, osim ako to ne bude zahtevala osoba zadužena za ispitivanje na odbrani. Student ima maksimalno 20 minuta za odbranu zadatka.

# 5. Prilog

## Prilog 1 – Transformisanje gramatike

2. 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 */).

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 → 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.

# Prilog 2 - Primeri izlaza

Ulazni program:

```
class P
	const 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 standarni izlaz.

```
==================SEMANTICKA OBRADA====================
Greska na 7: x vec deklarisano
Pretraga na 9(x), nadjeno Var x: int, 0, 1
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 ]]
```

# 6. Zapisnik revizija

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

## Verzija 1.1

| Strana | Izmena |
|---|---|
| | |
| | |