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: 2012/2013.
Datum: 03.12.2012.

Laboratorijska vežba
- 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 date laboratorijske vežbe potrebno je pokriti ključne faze i detalje za konstrukciju kompajlera za Mikrojavu koji omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji može da se izvršava na Mikrojava virtuelnoj mašini.

Ciljevi vežbe su:

Trening za razvoj (kompleksnih) softerskih sistema pisanih na programskom jeziku Java

Trening za rad sa potrebnim alatima za konstrukciju Mikrojava kompajlera

Konstrukcija uskog skupa funkcionalnosti kompajlera za Mikrojavu.

Zadaci na laboratorijskoj vežbi baziraju se na specifikaciji gramatike i leksičke strukture Mikrojave koja je osmišljena isključivo za potrebe laboratorijskih vežbi. Specifikacija gramatike Mikrojave napisana je u EBNF notaciji i sadrži samo podskup izabranih produkcija originalne gramatike.

Specifikacija leksičke strukture

Ključne reči:	program, print, return
Identifikatori:	ident = letter {letter | digit | "_"}.
Celobrojne konstante:	number = digit {digit}.
Operatori:	+ = , ; ( ) { }
Komentari:	// ... do kraja linije
Tipovi:	int

Specifikacija gramatike programskog jezika Mikrojava

```
Program     = "program" ident {VarDecl} "{" {MethodDecl} "}".
VarDecl     = Type ident ";".
MethodDecl  = Type ident "(" [FormPars] ")" {VarDecl} "{" {Statement} "}".
FormPars    = Type ident { "," Type ident }.
Type        = ident.
Statement   = Designator "=" Expr ";"
                  |
                  "print" "(" Expr ")" ";"
                  |
                  "return" [Expr] ";".
Expr        = Term {Addop Term}.
Term        = Factor.
Factor      = Designator [“(” [ActPars] “)” ] | number .
ActPars     = Expr {, Expr }.
Designator  = ident.
Addop       = "+".
```

2. Opis razvojnog okruženja i alata

Za konstrukciju kompajler za Mikrojavu koriste se standardni generatori kompajlera. Kompajler se sastoji od dve komponente: leksičkog analizatora i sintaksnog analizatora, koji se obogati funkcionalnostima za semantičku analizu i generisanje koda.

Za razvoj kompajlera mogu se koristiti integrisana razvojna okruženja, ali je potrebno da krajnji proizvod (kompajler) može da se pokrene iz komandne linije, bez ikakve interakcije sa razvojnim okruženjem.
Klase koje čine rešenje se generišu, kompjliraju i pokreću iz ComandPrompt-a koristeći alate `Jflex` i `java-cup-11a` koji se nalaze u istoimenim java bibliotekama. Pretpostavljamo da će rešenje (klase) biti smešteno u paket `resenje`.

Leksički analizator tj. skener se generiše na osnovu .flex fajla (na primer, skener.flex). Jflex.jar fajl treba smestiti u koreni direktorijum koji treba da bude podešen kao koren i u Comand Promptu. Komandom **java -jar JFlex.jar resenje\skener.flex** dobićemo klasu `Yylex.java`, koji predstavlja anlizator, u folderu `resenje`.

Parser se generiše na osnovu .cup fajla (na primer, MJParser.cup) pomoću CUP generatora koji se nalazi u biblioteci `java-cup-11a.jar`. CUP fajl treba smestiti u direktorijum `resenje`. Biblioteka `java-cup-11a.jar` mora biti u korenom direktorijumu kao i u prethodnom slučaju. Komandom **java -jar java-cup-11a.jar -destdir resenje resenje\MJParser.cup** dobijaju se klase parser.java i sym.java u folderu `resenje`.

Prevodjenje .java fajlova i generisanje .class fajlova se vrši sledećim komandama:
javac -cp .;java-cup-11a.jar resenje\parser.java .
Ovom komandom se ne generiše samo fajl parser.class već i .class fajlovi svih drugih klasa korišćenih u parseru.
Pokretanje parsera se vrši komandom:
java -cp .;java-cup-11a.jar resenje.parser test\prog.mj >test\log.out 2>test\log.err Pri tom se kao agrument prosledjuje fajl koji analiziramo, `prog.mj` iz foldera test. Standardni izlaz preusmeravamo u fajl `log.out`, a izlaz za greške u fajl `log.err`, koji se takodje nalaze u folderu test.

Za pokretanje parsera, ukoliko želimo generisanje koda koristimo sledeće komande:
java -cp .;java-cup-11a.jar resenje\parser test\primer.mj test\primer.obj

Ukoliko je potrebno proveriti generisani bajtkod i prikazati ga u čitljivoj formi, postoji komanda za disasembliranje Mikrojava bajtkoda:
java disasm test\primer.obj

Za pokretanje Mikrojava virtuelne masine i izvršavanje bajtkoda prevedenog programa koristi se sledeća komanda:
java Run test\primer.obj

Prilikom pokretanja Mikrojava virtuelne masine i izvršavanje bajtkoda prevedenog programa moguće je interaktivno ispisivanje bajtkoda koji se upravo izvršava. Za to je u prethodnoj komandi potrebno uključiti opciju debug.
java Run -debug test\primer.obj

3. Specifikacija zadataka za lab. vežbu

1) Konstrukcija leksičkog analizatora (na osnovu specifikacije leksičke strukture)
    i) Napisati .jflex fajl (specifikaciju leksičkog analizatora) za CUP kompatibilan leksički analizator.
    ii) Izgenerisati dati leksički analizator pomoću alata jflex.
    iii) Napisati Junit testove koji testiraju funkcionalnosti dobijenog leksičkog analizatora.

2) Konstrukcija sintaksnog analizatora/parsera (na osnovu specifikacije gramatike programskog jezika Mikrojava)
    i) Napisati .cup fajl (specifikacija sintaksnog analizatora).
    ii) Izgenerisati sintaksni analizator korišćenjem CUP generatora parsera alata.
    iii) U dobijenom parseru implementirati brojanje definicija funkcija unutar glavnog programa.
    iv) U dobijenom parseru implementirati brojanje iskaza dodele u MJ programima.

3) Oporavak parsera od sintaksnih grešaka
    i) Implementirati oporavak od greške pri deklarisanju globalne promenljive – ignorisati znakove do znaka ";".
    ii) Implementirati oporavak od greške u izrazu dodele – ignorisati znakove do znaka ";".

4) Semantička analiza
    i) Integrisati tabelu postojeću implementaciju tabele simbola u vidu heš tabele sa dobijenim parserom.
    ii) Implementirati metodu init u tabeli simbola za inicijalizaciju Universe opsega.
    iii) Obraditi (stavljanjem imena u tabelu simbola):
        (1) Deklaracije globalnih promenljivih
        (2) Deklaracije lokalnih promenljivih
        (3) Deklaracija funkcija glavnog programa
    iv) Detektovati svako naknadno korišćenje imena koja su stavljena u tabelu simbola.
    v) Implementirati semantičku analizu i proveru kontekstnih uslova za:
        (1) Iskaz dodele vrednosti.
        (2) Za izraz u naredbi return (mora da se slaže sa tipom povratne vrednosti).

5) Implementirati generisanje Mikrojava bajtkoda za sve sintaksno i semantički ispravne programe koji se mogu napisati na osnovu gramatike čija je specifikacija data na početku. Na raspolaganju su implementacija Mikrojava virtuelne mašine, kao i generator koda.

4. Prilog

Prilog 1 - 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.

5. Zapisnik revizija

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

Verzija 1.1

Strana	Izmena