Programski prevodioci 1

Domaći zadatak

Izrada kompajlera za “Microjava” programski jezik

Profesor: Dragan Bojić

Asistenti: Maja Vukasović, Kristijan Žiža

Student: Mandić Filip 2015/0308

Datum: 6.2.2019.

Mesto: U Beogradu

Opis projekta

Cilj ovog projekta jeste izrada kompajlera za programski jezik Microjava. Ovom prilikom realizovan je B novo postavke zadatka, koji obuhvata elemente proceduralnih programskih jezika.

Pregled postavke zadatka za B nivo

Najvažniji elementi jezika koje je potrebno realizovati prilikom izrade B nivoa obuhvataju:

* + Definiciju programa
  + Definiciju globalnih i lokalnih promenljivih koje se čuvaju u statičkoj zoni memorije tipova int, char, bool
  + Definiciju globalnih konstanti
  + Definiciju nabrojanih konstanti (enum-a)
  + Definiciju nizova podataka određenog tipa koji će se čuvati u dinamičkoj zoni memorije
  + Definiciju globalnih metoda(funkcija) programa, među kojima je i metoda main od koje počinje izvršavanje
  + Naredbe dodele vrednosti
  + Naredbe grananja (if-else)
  + Naredbe ciklusa (for(;;))
  + Naredbe break i continue za kontrolu toka u petljama
  + Naredbe čitanja/pisanja na standardni ulaz/izlaz
  + Predefinisane funkcije za konverziju između brojeva i karaktera (ord i chr)
  + Predefinisanu funkciju za dohvatanje duzine niza (len)
  + Predefinisanu konstantu za ispis novog reda (eol)

Korišćeni alati prilikom realizacije

Kompajler je realizovan korišćenjem pomoćnih alata pomoću kojih su realizovane leksička analiza, sintaksna analiza i tabela simbola i generisanje koda.

Za realizaciju leksičkog analizatora korišćen je alat JFlex. Ovaj alat na osnovu flex specifikacije generiše Yylex klasu koja vrši leksičku analizu.

Za realizaciju sintaksnog analizatora korišćena je CUP biblioteka. Ovaj alat na osnovu cup specifikacije generiše klasu MJParser koja vrši sintaksnu analizu, klasu sym u kojoj kodira sve dobijene lekseme od leksičkog analizatora i generiše folder ast u kome generiše klase pomoću kojih se kreira apstraktno sintaksno stablo. Takođe, u folderu ast generišu se interfejs Visitor i klasa VisitorAdaptor, pomoću kojih je moguće vršiti obilazak stabla, pri čemu se visit metode izvršavaju u postorder redosledu.

Za realizaciju semantičke analize i generisanja koda korišćena je implementacija biblioteke symboltable. Ova biblioteka sadrži realizaciju tabele simbola i definiše brojne konstante i tipove koji olakšavaju proveru semantičke ispravnosti programa.

Takođe, korišćena je i biblioteka mj-runtime pomoću koje je vršeno generisanje binarnog koda i pokretanje dobijenog binarnog koda.

U cilju lakšeg testiranja programa korišćena je biblioteka log4j.

Klase korišćene za realizaciju

Pored generisanih i bibliotečkih klasa, uvedene su dodatne klase za realizaciju semantičke analize i generisanja koda.

Klasa SemanticAnalyzer nasledjuje klasu VisitorAdaptor i implementira visit metode za čvorove za koje je to neophodno, kako bi se proverila semantička ispravnost ulaznog programa.

Klasa ActualParametersBuffer je pomoćna klasa za realizaciju semantičke analize. U njoj se skladište stvarni argumenti prilikom poziva funkcije kako bi se na kraju proverilo da li je broj argumenata ispravan i njihovi tipovi adekvatni.

Klasa CodeGenerator takođe vrši obilazak apstraktnog sintaksnog stabla realizacijom klase VisitorAdaptor. U njenim visit metodama vrši se generisanje binarnog koda.

Klasa JumpAddressStack je pomoćna klasa prilikom generisanja koda. Ona sadrži stekove adresa koje treba prepraviti prilikom backpatching-a za vreme generisanja koda.

U folderu test nalazi se klasa MJParserTest pomoću koje je moguće pokrenuti kompajler. Ova klasa očekuje dva argumenta komandne linije, putanje do ulaznog i izlaznog fajla.

Pokretanje

generisanje parsera:

* iz ant-a pokrenuti target repackage

pokretanje kompajliranja:

* pokrenuti klasu Compiler uz parametre komandne linije
* Parametri:
  + putanja do ulaznog fajla
  + putanja do izlaznog fajla

pokretanje izvrsavanja programa:

* java -cp lib\mj-runtime.jar rs.etf.pp1.mj.runtime.disasm <ime\_programa>
* java -cp lib\mj-runtime.jar rs.etf.pp1.mj.runtime.Run <ime\_programa> [-debug]

Prvu komandu koristiti za normalno pokretanje a drugu ukoliko je potrebno ispisivati informacije o steku.

Program je moguće pokrenuti i iz build skripte. Target disasm omogućava da se vidi dobijeni kod u tekstualnom obliku. Target runObj omogućava pokretanje, međutim zbog verzije ant-a koju podržava okruženje Intellij u kome je projekat razvijan, ukoliko se pokreće program ovako, neće biti moguće vršiti unos podataka( instrukcije read i bread neće raditi).

Testiranje

U folderu test nalaze se i paketi sa test primerima za svaki deo kompajlera(leksička analiza, sintaksna analiza, semantička analiza i generisanje koda). Testovi obuhvataju veliki broj ispravnih i neispravnih test primera za svaki deo kompajlera.

Testovi 1 i 2 su namenjeni za testiranje leksičke analize.

Testovi od 10 do 16 testiraju sintaksnu analizu, pri čemu su ulazni fajlovi ispravni. Testovi 21 – 24 testiraju programe sa sintaksnim greškama koje kompajler treba da prepozna i da se oporavi.

Testovi 100 – 108 testiraju semantičku analizu, pri čemu razmatraju razne ispravne i neispravne situacije.

Testovi 1000-1014 testiraju generisanje koda. Testovi 301 i 302 su javni testovi za nivo A i B i oni su bili dati na sajtu predmeta.