Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Programski prevodioci 1
Nastavnik: dr Dragan Bojić, vanr. prof.
Asistenti: mast. inž. Maja Vukasović, mast. inž. Kristijan Žiža
Školska: 2019/2020.
Ispitni rok: Januarsko-februarski ispitni rok
Datum: 24.11.2019.

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 [MJ].

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 formira na osnovu apstraktnog sintaksnog stabla koje je nastalo kao rezultat sintaksne analize. Semantička analiza se sprovodi implementacijom metoda za posećivanje čvorova apstraktnog sintaksnog stabla. Stablo je formirano na osnovu gramatike implementirane u prethodnoj fazi. Ukoliko izvorni kod ima semantičke greške, potrebno je prikazati adekvatnu poruku o detektovanoj semantičkoj grešci.

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, implementacijom metoda koje posećuju čvorove.

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

Projektni zahtevi su razvstani po težini i obimu na 3 nivoa: Nivo A (maks. 20 poena), Nivo B (maks 30 poena), Nivo C (maks. 40) poena. U sekciji V funkcionalnosti su detaljno podeljene po nivoima.

Odbranjen projekat je uslov za izlazak na ispit. Projekat je odbranjen ukoliko student na odbrani osvoji najmanje 20 poena, odnosno ukoliko implementira sve funkcionalnosti predviđene bar za Nivo A. U sekciji VI je detaljno opisano bodovanje projekata.

2. Reference

[MJ] Specifikacija jezika Mikrojava prilagođena postavci zadatka, http://ir4pp1.etf.rs/Domaci/2019-2020/mikrojava_2019_2020_jan.pdf

[PT] Šablon projekta podešen za integrisano razvojno okruženje Eclipse, http://ir4pp1.etf.rs/Domaci/2017-2018/pp1lab.templateAST.zip

3. Specifikacija zahteva

I Leksička analiza (3 poena)

U nastavku teksta su navedeni i opisani projektni zahtevi za implementaciju leksičkog analizatora.
Leksička analiza nosi maksimalno 3 poena.

- Potrebno je realizovati leksički analizator (*skener*) izvornih programa napisanih na jeziku Mikrojava.
- Leksički analizator se implementira pisanjem .flex specifikacije, čiji format je detljano opisan u prezentacijama primera domaćih zadataka sa sajta predmeta.
- Specifikacija leksičkog analizatora mora da se smesti u fajl [PT]/src/spec/mjlexer.flex.
- Specifikacija .flex se transformiše u implementaciju leksera na programskom jeziku Java korišćenjem alata **JFlex sa sajta predmeta.**
- Generisana klasa leksičkog analizatora mora da pripada paketu rs.ac.bg.etf.pp1 u okviru direktorijuma [PT]/src.
- Interfejs leksičkog analizatora prema sintaksnom analizatoru mora biti standardni CUP interfejs. Za više informacija, pogledati primer mini domaćeg u vežbama na sajtu predmeta.
- Skener prihvata fajl za izvornim kodom na jeziku Mikrojava i deli ga na tokene.
- Token se vraća eksplicitnim pozivom leksičkog analizatora (operacija `next_token()`).
  Potrebno je detektovati i obraditi sledeće leksičke strukture:
  – identifikatore,
  – konstante,
  – ključne reči,
  – operatore,
  – komentare.
- Leksičke strukture implementirati prema specifikaciji jezika [MJ§A.2p3].
- Leksički analizator treba da preskače komentare i "beline" u tekstu programa.
- Pod "belinama" se smatraju: tabulatori (\t), prelazak u novi red (\r \n), razmak (' '), backspace (\b), prelazak na novu stranu (\f, form feed).
- U slučaju leksičke greške, ispisuje se greška i nastavlja se obrada teksta programa.
- Poruka o grešci treba da sadrži sledeće informacije:
  – niz znakova koji nije prepoznat,
  – broj linije teksta programa u kojoj se desila greška, i
  – kolonu (poziciju prvog znaka) u kojoj je detektovana greška.
- Obavezno je korišćenje **jdk 1.8** kao što je opisano u primerima na vežbama.

II Sintaksna analiza (maks. 7 poena)

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 7 poena.
Raspodela poena po nivoima je: A(5), B(6), C(7).

Opšti tehnički zahtevi (maks. 4 poena)

- Gramatika jezika Mikrojava mora biti napisana u skladu sa specifikacijom jezika definisanom u [MJ].
- Za implementaciju parsera mora se koristiti generator sintaksnih analizatora AST-CUP (u nastavku teksta: AST-CUP generator). AST-CUP generator je lokalno razvijeno proširenje alata CUP za rad sa sintaksnim stablima.
- Mora se koristiti instalacija alata isključivo sa sajta predmeta (biblioteka cup_v10k.jar).

- Gramatička specifikacija parsera mora biti napisana u CUP fajlu, u formatu koji AST-CUP generator prepoznaje (u nastavku teksta: AST-CUP specifikacija).
- AST-CUP specifikacija mora da se smestu u fajl [PT]/src/spec/mjparser.cup.
- Sintaksni analizator mora biti integrisan sa CUP kompatibilnim leksičkim analizatorom za jezik Mikrojava.
- U slučaju uspešnog parsiranja ulaznog fajla parser na kraju rada na standardnom izlazu prikazuje apstraktno sintaksno stablo pozivom funkcije toString() nad korenom stabla (videti primer mini domaćeg).
- Parser treba da omogući oporavak od sintaksnih grešaka za zadate jezičke elemente.
- U slučaju nailaska na sintaksnu grešku parser:
  – ispisuje poruku greške u log fajl,
  – vrši oporavak od greške i
  – nastavlja sa parsiranjem ostatka fajla.
- Opis sintaksne greške TREBA da sadrži:
  – broj linije ulaznog programa u kojoj je greška detektovana (videti realizaciju u primeru mini domaćeg sa sajta predmeta),
  – nedvosmislen opis greške.

Implementacija parsera

- Nije dozvoljeno koristiti opciju *precedence* u .cup fajlu za definisanje prioriteta operatora. Izuzetak je korišćenje direktive precedence left ELSE; koja se može koristiti da se razreši konflikt između smena za if iskaz (statement) sa else delom i if iskaz bez else dela.
- Neterminali u AST-CUP specifikaciji moraju biti imenovani na način kako je to propisano zadatom specifikacijom [MJ] uz eventualno dodavanje sopstvenih neterminale, ukoliko se za tim ukaže potreba.
- Svakoj produkciji mora se zadati jedinstveni naziv na osnovu kojeg AST-CUP generator generiše Java klasu koja reprezentuje deo podstabla koji odgovara toj produkciji.
- Na osnovu AST-CUP specifikacije AST-CUP generator proizvodi standardnu CUP specifikaciju i geneiše klase elemenata sintaksnog stabla.
- Dobijena CUP specifikacija mora biti smeštena u fajl [PT]/src/spec/mjparser_astbuild.cup.
  Generisane klase apstr. sint. stabla moraju biti smeštene u paket rs.ac.bg.etf.pp1.ast u okviru direktorijuma [PT]/src.
- Uz izuzetak prijave sintaksnih grešaka, nije dozvoljeno ubacivati nikakve druge akcije u AST-CUP specifikaciju parsera {: :}.
  Dozvoljeno je dodavati uslužne metode ili polja u code {: :} sekciju AST-CUP specifikacije parsera isključivo za prijavljivanje i/ili oporavak od sinktasnih grešaka.
- Napisati klasu rs.ac.bg.etf.pp1.Compiler na programskom jeziku Java sa funkcijom glavnog programa *main* koja pokreće parsiranje Mikrojava programa. U slučaju uspešnog parsiranja, ispisuje strukturu sintaksnog stabla kako je opisano u zahtevima.
- Putanja do ulaznog fajla sa Mikrojava izvornim kodom prosleđuje se glavnom programu klase Compiler kao prvi argument komandne linije.

Oporavak od grešaka (maks. 3 poena)

- U AST-CUP specifikaciju gramatike TREBA dodati smene i akcije za oporavak od grešaka. Implementirati oporavak od grešaka za sledeće jezičke elemente:
- NIVO A (1 poen):
  – definicija lokalne promenljive – ignorisati karaktere do prvog znaka ";" ili sledećeg ","
  – konstrukcija iskaza dodele – ignorisati karaktere do ";"
- NIVO B (2 poena, uključujući i A):
  – deklaracija formalnog parametra funkcije – ignorisati znakove do znaka "," ili ")"
  – logički izraz unutar for petlje - ignorisati karaktere do znaka ";"
- NIVO C (3 poena, uključujući i sve za B):

– deklaracija metoda apstraktne unutrašnje klase – ignorisati karaktere do ";"
– deklaracija proširenja natklase – ignorisati znakove do prvog znaka "{" ili “extends”.

Testiranje rada implementiranog parsera:

- Napisati reprezentativni skup testova sintaksno ispravnih i neispravnih programa i testirati oporavak od grešaka.

III Semantička analiza

U sklopu semantičke analize vrši se ažuriranje tabele simbola i provera kontekstnih uslova opisanih u [MJ].

Semantička analiza se boduje sa maksimalno 10 poena. Funkcionalnosti su raspoređene po nivoima. Raspored bodova po nivoima je: A (4), B(7), C(10).

Opšti zahtevi (1 poen)

- Semantička analiza se vrši obilaskom apstraktnog sintaksnog stabla koje je nastalo kao rezultat sintaksne analize.
- Potrebno je implementirati klasu SemanticAnalyzer koja proširuje automatski generisanu klasu rs.ac.bg.etf.pp1.ast.VisitorAdapter i u njoj redefinisati metode za obilazak onih čvorova stabla koji su relevatni za semantičku analizu.
- Klasa SemanticAnalyzer mora biti smeštena u paket rs.ac.bg.etf.pp1.
- Nije dozvoljeno dodavati nikakve semantičke akcije u AST-CUP sepecifikaciju parsera {: :}, niti klasi parsera dodavati metode niti polja koja obavljaju ili neposredno ili posredno utiču na semantičku analizu.
- Semantički obilazak stabla se pokreće u funkciji glavnog programa klase Compiler nakon završetka sintaksne analize, tako što se objekat klase SemanticAnalyzer prosleđuje korenu sintaksnog stabla.
- Semantički analizator je potrebno integrisati sa tabelom simbola.
- Mora se koristiti implementacija tabele simbola dostupna na sajtu predmeta: http://ir4pp1.etf.rs/Domaci/symboltable-1-1.jar.
- Tabela simbola se uvezuje sa ostatkom programa kao Java biblioteka (.jar) i dozvoljeno je koristiti sve njene javne klase, metode i polja. Uz biblioteku se dostavlja i izvorni kod koji je predviđen samo za informisanje o detaljima implementacije i njihovo razumevanje.
- NIJE DOZVOLJENO raspakivati biblioteku tabele simbola, menjati njenu implementaciju i ponovo je prevoditi i uvezivati sa projektom.
- 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.
- Implementirati javno dostupnu metodu void tsdump() u klasi Compiler za ispis sadržaja tabele simbola. Metoda mora da se pozove glavnom programu klase Compiler po završetku semantičkog prolaza..

Detektovanje korišćenja simbola (Bodovanje: NivoA - 1, NivoB - 2, NivoC - 3)

- U klasi SemanticAnalyzer implementirati detektovanje upotrebe simbola za sledeće jezičke elemente:
- NIVO A (1 poen)
  – simboličke konstante,
  – globalne promenljive,
  – pristup elementu niza.
- NIVO B (2 poena, uključuje i sve zahteve iz A)
  – globalne funkcije (pozivi)
  – for petlje

– korišćenje formalnog argumenta funkcije
- NIVO C (3 poena, uključuje i sve zahteve iz B)
  – apstraktne unutrašnje klase
  – polja unutrašnjih klasa (pristup polju)
  – metode unutrašnjih klasa (pozivi).
- Za svaki detektovani simbol potrebno je proveriti sledeće:
  – da li ime postoji u tabeli simbola,
  – da li je ispravnog tipa.

Format poruke
- Poruka o detektovanom simbolu MORA da sadrži sledeće podatke (videti Prilog 2):
  – linija izvornog koda u kojoj je pronađen simbol,
  – naziv pronađenog simbola,
  – ispis objektnog čvora iz tabele simbola koji odgovara pronađenom simbolu.

Provera kontekstnih uslova (Bodovanje: Nivo A - 2, NivoB - 4, NivoC - 6)
- U klasi SemanticAnalyzer implementirati proveru svih kontekstnih uslova navedenih u specifikaciji [MJ§A.4p5], a predviđenih za odabrani nivo težine.

Testiranja rada implementiranog semantičkog analizatora:
- Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve sintaksno i semantički ispravne MJ programe uz pokrivanje svih smena iz gramatike.
- Napisati ulazne fajlove na programskom jeziku Mikrojava koji sadrže sve kombinacije semantičkih grešaka.

IV Generisanje koda

Generisanje koda podrazumeva transformaciju sintaksno i semantički ispravnog sintaksnog stabla u bajtkod za izvršno okruženje za MJ virtuelnu mašinu (MJVM).

Generisanje koda nosi maksimalno 20 poena. Zahtevi su po težini podeljeni u grupe. Raspodela poena po nivoima je: A(8), B(14), C(20).

Opšti zahtevi
- Generisanje koda vrši se obilaskom apstraktnog sintaksnog stabla koje je nastalo kao rezultat sintaksne analize i zadovoljilo uslove semantičke provere.
- Potrebno je implementirati klasu rs.ac.bg.etf.pp1.CodeGenerator, koja proširuje automatski generisanu klasu rs.ac.bg.etf.pp1.ast.VisitorAdapter, i u njoj redefinisati medote za obilazak elemenata sintaksnog stabla koji su relevanti za generisanje koda.
- Nije dozvoljeno implementirati generisanje koda u akcijama {: :} AST-CUP specifikacije parsera, niti klasi parsera dodavati metode niti polja koja obavljaju ili posredno ili neposredno utiču na generisanje koda.
- Generator koda mora da generiše ispravan bajtkod za MJVM.
- Za implementaciju generatora koda moraju se koristiti alati `Code`, `disasm` i `Run`. dostupni i biblioteci mj-runtime.jar: http://ir4pp1.etf.rs/Domaci/mj-runtime-1.1.jar
- Generisanje koda se pokreće u glavnom programu klase Compiler po završetku semantičke analize i ispisa sadržaja tabele simbola. Implementira se prosleđivanjem objekta klase CodeGenerator korenu sintaksnog stabla.
- Izlaz generatora koda mora da bude izvršivi .obj fajl za MJVM.
- Putanja do izlaznog .obj fajla prosleđuje se glavnom programu klase Compiler kao drugi argument komandne linije.

V Podela funkcionalnosti po nivoima

- **NIVO A –** potrebno je implementirati generisanje koda za SVE gramatičke smene u nastavku (osnovni iskazi, aritmetički izrazi, rad sa nizovima prostih tipova i rad sa funkcijama bez parametara):

  Statement := DesignatorStatement ";".
  DesignatorStatement := Designator "=" Expr.
  DesignatorStatement := Designator "++".
  DesignatorStatement := Designator "--".
  Statement := DesignatorStatement ";".
  Statement := "read" "(" Designator ")" ";".
  Statement := "print" "(" Expr [“,” number] ")" ";".
  Expr := ["-"] Term {Addop Term}.
  Term := Factor {Mulop Factor}.
  Factor := numConst | charConst | "(" Expr ")" | boolConst | "new" Type ["[" Expr "]"].
  Factor := Designator ["(" ")"].
  Designator := ident ["[" Expr "]"].
  Addop := "+" | "-" .
  Mulop := "*" | "/" | "%".

- Od nizova, treba podržati samo nizove ugrađenih tipova podataka. Program mora da sadrži funkciju main, globalne funkcije, globalne/lokalne promenljive (proste ili nizovne), globalne konstante. Ne treba obrađivati unutrašnje klase.

- **NIVO B –** potrebno je implementirati SVE zahteve za Nivo A i SVE gramatičke smene u nastavku (kontrolne strukture, uslovni izrazi, pozivi globalnih metoda):

  DesignatorStatement := Designator "(" [ActPars]")" .
  Statement := "if" "(" Condition ")" Statement ["else" Statement].
  Statement := "for" " (" [DesignatorStatement] ";" [Condition] ";" [DesignatorStatement] ")"
  Statement.
  Statement := "break" ";".
  Statement := "continue" ";".
  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 := Designator ["(" [ActPars] ")"].

  Potrebno je implementirati optimalno izračunavanje složenih kondicionala. Ukoliko je na osnovu parcijalnog dela složenog kondicionala poznato da će ceo uslov biti prihvaćen ili odbijen, ne izračunava se ostatak kondicionala. Optimizovano izračunavanje je detaljnije opisano u materijalima za vežbe (http://ir4pp1.etf.rs/Vezbe/MikrojavaVirtuelnaMasina.pdf - Prevođenje kontrolnih struktura).

- **Nivo C –** potrebno je implementirati SVE zahteve za Nivo B i još dodatno zahteve u nastavku (unutrašnje klase, supstitucija i polimorfizam):

- implementirati nasleđivanje klasa (apstraktnih i ne-apstraktnih);
- implementirati supstituciju (na mestu gde se očekuje referenca na osnovnu klasu može se predati referenca na objekat izvedene klase, odnosi se i na slučaj kada je osnovna klasa apstraktna, a izvedena ne);
- implementirati pravljenje objekata unutrašnjih klasa i nizova objekata unutrašnjih klasa;
- implementirati pravljenje tabela virtuelnih funkcija;
- implementirati polimorfno pozivanje metoda unutrašnjih klasa.

VI Ocenjivanje domaćih zadataka

Opšti zahtevi

- Generisanje koda, koje proizvodi ispravne izvršne bajtkod (predmetne) fajlove u skladu sa opisanim zahtevima, je POTREBAN i DOVOLJAN uslov za izlazak na odbranu domaćeg zadatka.
- Na odbrani se ocenjuju FUNKCIONALNOSTI kompajlera, koje su opisane u specifikaciji jezika [MJ].
- FUNKCIONALNOST je izvršiva jedinica kompajlera koja odgovarajuću naredbu, izraz ili definiciju prevodi u izvršne bajtkod instrukcije ili podatke zaglavlja predmetnog (obj) fajla. Mora biti implementirana na sva četiri nivoa (leksika, sintaksa, semantika i generisanje koda ili podataka zaglavlja predmetnog fajla).

Bodovanje projekta

- Projekat se boduje prema cenovniku za Nivo A, akko su ispunjeni SVI zahtevi predviđeni za Nivo A. Maksimalan broj poena predviđen za Nivo A je 20.
- Projekat se boduje prema cenovniku za Nivo B, akko ispunjava SVE zahteve za Nivo A i, ako ispunjava DEO, a najviše SVE zahteve propisane za Nivo B. Maksimalan broj poena predviđen za Nivo B je 30.
- Projekat se boduje prema cenovniku za Nivo C, akko ispunjava SVE zahteve predviđene za Nivo B, i ako ispunjava DEO ili SVE funkcionalnosti predviđene za Nivo C. Maksimalan broj poena predviđen za Nivo C je 40.
- Ukoliko student ne implementira SVE funkcionalnosti za Nivo A, projekat neće biti razmatran na odbrani.

VII Primer programa:

```
program Program
      abstract class A{
         int x[],y[];
         abstract int getValue(int a);
      }
      const int pi = 3, e = 2;
      class B extends A {
         int i;
         {
            int getValue(int a) int b; bool c;{ return this.i + this.x[0] + a; }
            void m(){}
         }
      }
      class C extends B{B theA;int a;}
{
      void main() B b; C c; int i; int x[];{
            b = new B;
            b.x = new int[5];
            b.y = new int[5];
            c = new C;
```

```
            c.theA = b;
            c.x = new int[5];
            x = new int[3];
            read(c.i);
            for(i=0; i<5; i++){
                  read(c.x[i]); read(c.theA.x[i]);
            }
            print(c.getValue(c.theA.x[0]));
            c.m();
      }
}
```

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 priloženih test primera (ne uključivati ulaze niti izlaze testiranja u izveštaj).
f) kratak opis novouvedenih klasa.

Izvorni i prevedeni programski kod mora da sledi direktorijumsku strukturu koja je određena u šablonu projekta [PT]. Dakle moraju se poslati .flex i .cup fajlovi, svi izgenerisani i rukom pisani .java fajlovi koji čine rešenje i odgovarajući prevedeni .class fajlovi. Rešenje treba da sadrži i .jar arhive alata AST-CUP i Flex.

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, AST-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.

Napomena: Primer je ilustrativnog karaktera namenjen prikazivanju akcija pretrage i obrade grešaka kao što je navedeno u postavci projekta i ne mora odgovarati sintaksnim zahtevima postavke.

```
==================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