IR4PP1

Rok za predaju(jun): **16.06.2008.**
Rok za predaju(april): **07.05.2008.**

Domaći zadatak/drugi deo
Junski rok 2007/2008

Semantička analiza i prevođenje u Java izvorni kod

Potrebno je projektovati analizator za jezik MikroJava (MJ) koji, pored leksičke i sintaksne analize, puni tabelu simbola, vrši semantičku analizu MJ programa i prevođenje MJ izvornog koda u Java izvorni kod. Iskoristiti leksički analizator napravljen u prvom delu domaćeg zadatka. Takođe, kao polaznu osnovu za pisanje nove CUP specifikacije, iskoristiti CUP specifikaciju urađenu u prvom domaćem zadatku iz koje je potrebno ukloniti akcije koje su prebrojavale ili ispisivale pronađene elemente jezika. Oporavak od greške se može ili ukloniti ili ostaviti u gramatici. To nije bitno, jer će rad analizatora biti proveravan samo na sintaksno ispravnim MJ programima. Konkretno, potrebno je obezbediti sledeće:

1. (12 poena)

a) (8 poena)
Dodavanjem odgovarajućih akcija u parser, odnosno CUP specifikaciju, omogućiti unos svih simbola MikroJava jezika u tabelu simbola. Tabela simbola koju je potrebno implementirati ima sličnu strukturu i logiku popunjavanja kao tabela izložena na vežbama i predavanjima (pogledati materijale sa sajta), znači i dalje postoje Object, Struct i Scope čvorovi. Razlika je u tome što svuda gde je ranije korišćeno umetanje Object čvorova u jednostruko ulančanu listu, sada je potrebno koristiti stablo M-arnog pretraživanja. Interfejs se može proizvoljno usvojiti. Od pomoći pri implementaciji može biti ranija implementacija sa ulančanim listama, data na sajtu predmeta (http://ir4pp1.etf.bg.ac.yu/Domaci/TabSimImpl.rar). Pored unosa u tabelu simbola, potrebno je detektovati svako korišćenje simbola (pod korišćenjem se misli na upotrebu u naredbama u telu metoda), proveriti da li odgovarajući objekat postoji u tabeli simbola i ispisati poruku (koja uključuje i broj linije u kodu u kojoj je razmotreni simbol) o pronađenom simbolu ili poruku o grešci (npr. simbol nije pronađen; simbol nije tipa niza, a koristi se uz operator indeksiranja; simbol nije polje klase i slično). Poruka mora biti takva da se iz nje može nedvosmisleno zakjučiti koji simbol je pronađen (ime simbola, vrsta, tip).
Na kraju izvršavanja programa (po povratku iz parsera), pozivom metode `dump()`, na standardni izlaz ispisati sadržaj tabele simbola.
Kod objekata u tabeli simbola koji odgovaraju konstantama, potrebno je postavljati vrednost polja adr, jer je to vrednost konstante. Kod ostalih objekata u tabeli simbola NIJE POTREBNO dodeljivati vrednost polju adr.

b) (4 poena)
Obezbediti proveru sledećih kontekstnih (semantičkih) uslova MJ jezika:

01) (1.5 poena) U pozivu metode stvarni argumenti moraju po broju i tipu odgovarati formalnim argumentima u definiciji te metode. Ovo se odnosi i na predefinisane i na korisničke metode.

02) (1 poen) Posmatra se sledeća smena MikroJava gramatike (inkrement i dekrement).

```
Factor = “new” Type.
```

Za prethodno navedenu smenu mora važiti:

`Type` mora biti klasa.

03) (1 poen) Posmatra se sledeća smena MikroJava gramatike

```
Designator = Designator “.” ident.
```

`Designator` mora biti klasnog tipa. **`ident`** mora biti polje od **`Designator`**.

04) (0,5 poena) Posmatra se sledeća smena MikroJava gramatike (iskaz dodele):

```
Factor = Designator "(" [ActPars] ")".
```

Za nju mora važiti:

`Designator` mora biti metoda.

2. (8 poena)

Obezbediti da MJ analizator, na osnovu ulaznog MJ fajla, pravi novi fajl sa izvornim Java kodom koji odgovara ulaznom MJ programu. Pri tome važi sledeće:

01) Ime klase koja predstavlja analizator ne treba menjati, već to treba da ostane `parser`. Očekuje se da se napravljeni analizator pokreće na sledeći način:

```
java ime_paketa.parser imeMJfajla.mj
```

`ime_paketa` predstavlja ime paketa u kojem se nalaze klase napravljenog analizatora. Forma tog imena će biti definisana u delu Napomene. `imeMJfajla.mj` je izvorni MikroJava fajl koji predajemo analizatoru na analizu.

02) Kada se analizator pokrene za neki MJ program, on će u tekućem direktorijumu u kojem se nalazi napraviti fajl `imeGlavneMJKlase.java`, u koji će upisati odgovarajući Java kod.

03) Pošto se MikroJava i Java razlikuju, ne može se izvršiti 1-1 preslikavanje MJ koda u Java kod. Da bi se realizovalo korektno preslikavanje treba primeniti sledeća pravila preslikavanja:

- Na početak svakog Java fajla koji se dobija iz nekog MJ fajla dodajemo iskaz `import java.io.*;`
- Ispred ključne reči `class` na početku definicije glavne MJ klase se dodaje specifikator `public`.
- U telo glavne klase uvek dodati sledeće definicije pomoćnih statičkih promenljivih `in i scanner : private static InputStreamReader = new InputStreamReader(System.in); private static BufferedReader scanner = new BufferedReader(in);`
- Ispred definicija konstanti, deklaracija svih globalnih promenljivih se dodaju specifikatori: `private static`.(polja glavne klase u .java fajlu).
- Unutrašnje klase MJ programa treba staviti ispred glavne klase u java fajlu.

- Raspored velikih zagrada se prilagođava njihovom rasporedu u Javi.
- Definicija metode `main` mora izgledati kao u Javi, tj. `public static void main (String args[])`.
- Ispred definicija ostalih metoda se dodaju specifikatori `private static`.(metode glavne klase u .java fajlu)
- Na početku svake metode (uključujući i metodu main) , nakon lokalnih promenljivih, potrebno je dodati `try{`
- Na kraj svake metode (uključujući i metodu main), ispred return iskaza ukoliko postoji, potrebno je dodati `} catch(Exception e) { System.out.println(e); }`
- Lokalne promenljve složenog tipa treba inicijalizovati na null.
- Lokalne promenljive prostog tipa inicijalizovati na 0.
- Kreiranje novog objekta u MJ jeziku izgleda npr. `val = new Table;`. Da bi se dobio Java kod, dodaju se zagrade `val = new Table();`
- Predeklarisanu MJ metodu `print(s)` ili njenu varijantu `print(s, n)` menjamo sa `System.out.println(s). print(eol) treba zanemarivati`.
- Ako imamo poziv predeklarisane MJ metode `read(a)`, gde je `a` tipa int, onda se to menja sa `a = Integer.parseInt((scanner.readLine());` Ako imamo poziv predeklarisane MJ metode `read(b)`, gde je `b` tipa `char`, onda se to menja sa `b = scanner.readLine().charAt(0);`
- Predeklarisanu MJ metodu `len(a)` koja se nalazi u izrazu menjamo sa `a.length`.
- Što se tiče formatiranja izlaznog Java koda, potrebno je da kod bude «lepo» formatiran. Ne treba posvetiti previše vremenena samom izgledu ali svakako treba izbeći rešenja gde bi se svi elementi programa stavili u jedan red.
- Pri preslikavanju u Java kod potrebno je pridržavati se prethodnih pravila. Pritom treba voditi računa da se dobijeni Java fajl može kompajlirati i pokrenuti.

Prethodna pravila su ilustrovana primerom datim u Dodatku A ovog dokumenta.

Napomene

Slanje domaćeg

Ime paketa u kojem treba da se nalaze leksički analizator i parser je oblika xbbbgg, gde x predstavlja p za studente sa parnim brojem indeksa, a n za studente sa neparnim brojem indeksa; bbb je broj indeksa; gg je godina upisa.

Folder xbbbgg u kojem se nalaze .lex fajl, .cup fajl, izvorni Java fajlovi tabele simbola i izvorni Java falovi neophondi za funkcionisanje leksičkog analizatora i parsera treba zapakovati u zip arhivu i poslati 15.06 ili 16.06 (do ponoći) na adresu compilers.etf@gmail.com.

Ako neke delove domaćeg niste radili, to treba navesti u telu mejla.

U subject-u navesti xbbbgg po ranije opisanom formatu, zatim ime i prezime tim redom. Treba slati samo jedan mejl. Višestruko slanje se neće priznati.

Prijavljivanje za odbranu domaćeg će se vršiti na fakultetu od 17.06.2008. na način koji će biti naknadno objavljen preko mejling liste.

Osnovne napomene

Svi izvorni fajlovi, osim onih koje automatski generišu alati MORAJU BITI DETALJNO KOMENTARISANI.
Ako napravljeni analizator koristi i neke dodatne Java klase, potrebno je poslati i njihov izvorni kod i u komentarima navesti svrhu tih klasa.

Broj poena

Broj poena koji nosi drugi domaći zadatak je 20 (ispred svake stavke naveden je broj poena, koji nosi, unutar zagrada “()”). 20 je ujedno i maksimalni broj poena koji se može osvojiti na drugom domaćem.

Dodatak A – Preslikavanje MJ programa u Java program

Neka se analizatoru na obradu daje MJ fajl `pr1.mj`:

```
class Example

final int adult = 18;
final char name = 'n';
final char year = 'y';

class Person {
    char name[];
    int year;
}

final int num = 3;

{

    // metoda koja cita podatke sa ulaza o osobi kreira objekat Person
    // i popunjava odgoravajuca polja
    Person readPerson()
          int d, i;
          Person p;
          char c;
    {
          print('d', 5);    // drugi argument se ignorise
          print(eol);
          read(d);
          p = new Person;
          p.name = new char[d];
          i = 0;

          while(i < d) {
                read(c);
                p.name[i] = c;
                i++;
          }

          print('y');
          print(eol);
          read(p.year);

          return p;
    }

    // metoda koja sortira niz osoba na osnovu godina
    void sort(Person people[])
          int i;
          int j;
          Person p;

    {
          i = 0;
          j = 0;
```

```
        while(i < len(people)-1) {
              j = i+1;
              while(j < len(people)) {
                    if(people[i].year > people[j].year) {
                          p = people[i];
                          people[i] = people[j];
                          people[j] = p;
                    }
                    j++;
              }
              i++;
        }
    }

    void main ()
        int i;
        Person people[];
        Person p;
    {
        people = new Person[num];
        i = 0;

        while(i < num) {
              p = readPerson();
              people[i] = p;
              i++;
        }

        sort(people);

        i = 0;
        while(i < len(people)) {
              print(people[i].year);
              print(eol);
              i++;
        }
    }

}
```

Analizator će napraviti fajl `Example.java` sa sledećim sadržajem:

```
    import java.io.*;


    class Person {
      char name[];
      int year;
    }

    public class Example {
      private static final int adult = 18;
      private static final char name = 'n';
      private static final char year = 'y';
      private static final int num = 3;
```

```
  private static InputStreamReader in = new
InputStreamReader(System.in);
  private static BufferedReader scanner = new BufferedReader(in);

  public static void main(String[] args) {
    int i = 0;
    Person people[] = null;
    Person p = null;
    try {
      people = new Person[num];
      i = 0;
      while (i < num) {
        p = readPerson();
        people[i] = p;
        i++;
      }

      sort(people);

      i = 0;
      while (i < people.length) {
        System.out.println(people[i].year);
        i++;
      }
    } catch (Exception e) {
      System.out.println(e);
    }
  }

  private static Person readPerson() {
    int d = 0; // duzina imena
    int i = 0; // brojac
    Person p = null; // osoba koja se kreira
    char c = 0; // karakter sa ulaza
    try {
      System.out.println('d');
      d = Integer.parseInt(scanner.readLine());

      p = new Person();
      p.name = new char[d];
      i = 0;
      while (i < d) { // citanje imena sa ulaza
        c = scanner.readLine().charAt(0);
        p.name[i] = c;
        i++;
      }

      System.out.println('y');
      p.year = Integer.parseInt(scanner.readLine()); // citanje godina
sa ulaza
    } catch (Exception e) {
      System.out.println(e);
    }
    return p;
  }
```

```
    private static void sort(Person people[]) {
      int i = 0;
      int j = 0;
      Person p = null;
      try {
        i = 0;
        j = 0;
        while (i < people.length - 1) {
          j = i + 1;
          while (j < people.length) {
            if (people[i].year > people[j].year) {
              p = people[i];
              people[i] = people[j];
              people[j] = p;
            }
            j++;
          }
          i++;
        }
      } catch (Exception e) {
        System.out.println(e);
      }
    }

}
```

Napomena: Dobijeni Java program se može prevesti i pokrenuti.