Exercise 3: Processing a Phone Book

Assume that persons and their phone numbers are stored in a text file with the following format:

```
Doe, John Maxwell
    home +44 (20) 24684567,
    work (020) 234567
Miller, Ann
    23456789
...
```

Each entry begins with a surname, a comma, and one or more first names. This is followed by a comma-separated list of phone numbers, which can take one of the following forms:

+44 (20) 24684567	Country code, area code in brackets without leading 0, number
(020) 234567	Area code in brackets with leading 0, number
23456789	Number without country code and area code

Each phone number can be preceded by the keyword "home" or "work". If it is missing, "home" is assumed. If the country code is missing, "+44" is used, if the area code is missing, "020" is assumed.

a) Describe the structure of this text file by a context-free grammar.

b) Turn this into an attributed grammar so that the names and phone numbers are read and stored in a suitable data structure with separate fields for the country code, the area code, the number and the identification home/work. Output this data structure for control. Use Coco/R to create a program that reads the input file, builds the data structure and outputs it.

Solution

a) *Context-free grammar*

```
PhoneBook   = Entry {Entry}.
Entry       = Name Number {"," Number}.          // a person can have multiple phone numbers
Name        = ident "," ident {ident}.           // last name, first name(s)
Number      = [Kind] [CountryCity | City] number.
Kind        = "home" | "work".
CountryCity = "+" number "(" number ")".         // country code and city code without leading 0
City        = "(" "0" number ")".                // city code with leading 0
```

b) *Compiler specification*

```
import java.util.ArrayList;

COMPILER PhoneBook

  // ------------------ global declarations ---------------------

  class Person {
    String firstName;
    String lastName;
    ArrayList<Phone> numbers;
    Person(String first, String last) {
      firstName = first; lastName = last;
      numbers = new ArrayList<Phone>();
    }
  }
```

```
   class Phone {
      String kind;                  // "home", "work"
      String countryCode;           // e.g., "+44"
      String cityCode;              // e.g., "020"
      String number;                // e.g., 123456
      Phone (String k, String co, String ci, String n) { kind = k; countryCode = co; cityCode = ci; number = n; }
   }

   void printPhoneBook() {
      for (Person p: phoneBook) {
         System.out.println(p.lastName + ", " + p.firstName);
         for (Phone phone: p.numbers) {
            System.out.println("  " + phone.kind + " " + phone.countryCode + " " + phone.cityCode + " " + phone.number);
         }
      }
   }

   ArrayList<Person> phoneBook = new ArrayList<Person>();

//---------- scanner specification ----------
CHARACTERS
   letter    = 'A'..'Z' + 'a'..'z'.
   digit     = '0' ..'9'.
   nonzero   = digit - '0'.

TOKENS
   ident = letter {letter}.
   number = nonzero {digit}.

IGNORE '\r' + '\n' + '\t'

//---------- parser specification ----------
PRODUCTIONS

PhoneBook = Entry { Entry }    (. printPhoneBook(); .) .

Entry                          (. Person person; Phone phone; .)
= Name <out person>
  Number <out phone>           (. person.numbers.add(phone); .)
  {    ","
       Number <out phone>      (. person.numbers.add(phone); .)
  }                            (. phoneBook.add(person); .) .

Name <out Person person>       (. String firstName, lastName; .)
=  ident                       (. lastName = t.val; .)
   "," ident                   (. firstName = t.val; .)
   {  ident                    (. firstName += " " + t.val; .)
   }                           (. person = new Person(firstName, lastName); .) .

Number <out Phone phone>       (.  String kind = "home";       // default
                                   String countryCode = "+44"; // default
                                   String cityCode = "020";    // default
                                   .)
=  [ "home"                    (. kind = "home"; .)
   | "work"                    (. kind = "work"; .)
   ]
   [   "+" number              (. countryCode = "+" + t.val; .)
       "(" number              (. cityCode = "0" + t.val; .)
       ")"
   |   "(" "0" number          (. cityCode = "0" + t.val; .)
       ")"
   ]
   number                      (. phone = new Phone(kind, countryCode, cityCode, t.val); .) .

END PhoneBook.
```

The only terminal symbols we have (apart from the literals) are names (ident) and numbers that do not start with 0 (number). In addition to blanks, line endings and tabs should be ignored.

As the data structure for the phone book, we use a list phoneBook, which is declared in the global semantic declarations. The list contains persons (class Person with first name, last name and phone numbers). The phone numbers are a list of Phone objects with their kind (home or work), the country code, the area code and the actual phone number.

An entry in the input file consists of a name (Name) and a sequence of phone numbers (Number). Name returns a Person object as an output attribute, Number returns a Phone object. These objects are then entered into the data structure phoneBook. The method printPhoneBook() outputs this data structure for checking its correctness.

We still need a main program that gets the name of the input file as a command line argument, generates a scanner and a parser and starts the syntax analysis:

```
class PhoneBook {
   public static void main (String[] arg) {
      Scanner scanner = new Scanner(arg[0]);
      Parser parser = new Parser(scanner);
      parser.Parse();
      System.out.println(parser.errors.count + " errors detected");
   }
}
```

We process the compiler specification with Coco/R:

```
java -jar Coco.jar PhoneBook.atg
```

thus generating a scanner and a parser. Then we compile all Java files

```
javac Scanner.java Parser.java PhoneBook.java
```

and invoke the program as follows:

```
java PhoneBook input.txt
```

The file input.txt contains the names and phone numbers as described above. The output for the above example is:

```
Doe, John Maxwell
   home +44 020 24684567
   work +44 020 234567
Miller, Ann
   home +44 020 23456789
...
```