Principles of Programming Languages and Compilers

Computer Engineering and Informatics Department

University of Patras

Spring Semester 2024

Professors: J. Garofalakis, S. Sioutas, P. Hadjidoukas

Lab Exercise

The purpose of the lab exercise is to familiarize yourself with the description of a language in BNF format, basic concepts of compilers and finally to implement two of the parts of a compiler, a syntax and a lexical analyzer using the Bison and Flex tools.

PROCEEDINGS

Deliverables

❖ **Written Report** including:
- The details of the team members (Fullname, Student Number, e-mail)
- The description of the grammar of the language in BNF.
- The **FINAL** language description files, given as **input** to Flex and Bison
- Screenshots of example applications (at least one example for each question, for successful and unsuccessful analysis)
- Comments - assumptions made for the development of the project

❖ **Compressed into one file (zip)**:
- The above written report.
- The **FINAL** input files for flex and bison.
- The **FINAL** C code (and the .h file) generated by the two programs and the parser executable code.
- The **test files** provided as input to the parser to check for proper functionality.

The zip file must be named with the registration numbers of the team members separated by _ and from the lowest to the highest (e.g. **1000_1543_2788_3972.zip**), and **must be submitted (MANDATORY) to the eclass of the course and the message section must include the full names, the year and the corresponding registration numbers of the team members, as well as the e-mail addresses of all team members.**

Clarifications

- You can find information about the operation of the Flex and Bison tools in the course eclass.
- The exercise will be done in groups of 1 to 4 students.
- The contribution to the final grade is 30%.
- The exercise is compulsory and the grade must be at least 5.
- The final delivery dates will be determined according to the written examination dates of June and September periods respectively. The date of the oral examination of the assignment will be determined accordingly.
- Any questions or suggestions should be posted in the "Discussions" area on the course eclass page

  (https://eclass.upatras.gr/modules/forum/?course=CEID1091)

EXERCISE DESCRIPTION

The language to be analyzed is an imaginary object-oriented programming language. The programs of the language are organized into classes, as shown in the specifications below.

Language specifications

- Every program in the language consists of one or more classes. A class can be nested within another class.

- Language identifiers consist of letters (**A...Z**, **a...z**), digits (**0...9**) and the special underscore character **_**. An identifier cannot start with a digit. The language is case sensitive.

- Each class follows the following format:

  public class <class_name > **{**

  optional variable declaration

  optional method declaration

  }

  The name of each class starts with a capital letter.

  For example:

  ```
  public class A {
      public void method1() {
          out.println("Hello world!");
      }
  }
  ```

- The following language identifiers are reserved and cannot be used as common identifiers: int, char, double, boolean, String, class, new, return, void, if, else, while, do, for, switch, case, default, break, true, false, public, private.

- The basic (primitive) data types supported by the language are **int**, **char**, **double** and **boolean**.
  - The **int** data type is used to declare/create variables of integer type.
  - The **char** data type is used to declare/create variables consisting of a single or special character (e.g. a newline character) enclosed in single quotes. For example, 'a', 'b', '9', '9', '\n'.
  - The **double** data type is used to declare/create floating-point variables. At the end of each value of this type should be the character d, e.g. 5.99d, 6.0d, 123545.23436d.
  - The **boolean** data type is used to declare/create variables of logical type and can only take the values true or false.

- The language additionally supports the non-primitive data type String, which is used to declare a sequence of characters enclosed by double quotes.

For example:

“Hello123$”, “This is a valid str1ng”, “this is \‘valid\’ too”

- The variable declaration follows the following format:

  [<modifier>] <data_type> <variable_name>**;**

  while the allowed data types are those presented previously. Optionally, a variable may be declared as public or private (modifier term).
  For example:

  int var1; private double var2;

- The creation of a class object follows the following format:

  <class_name> <object_name> **= new** <class_name>**();**

  Accessing members (variable, method) of a class follows the following format:

  <object_name>**.**<class_member_name>**;**

- The definition of methods follows the Java definition style. In the body of any method there is first an optional variable declaration, followed by the commands (the allowed commands of the language are shown below). A method can be public or private. A method may or may not return a value. A method may take none, one or more arguments..

  Present and fully explain the method definition in your grammar.

- A variable or method that is public is accessible by objects of all classes. A variable or method that is private is accessible only by objects of the class they belong to.

- The rest of the program commands of the language are divided into:

  - *Assignment statements*

    They have the form **<variable> = <expression>;**, where the expression can be a literal or a compound expression of a combination of literals, variables and parentheses, a method call or a class object creation. The method call follows the Java logic. In addition, the compound expression can be any numeric expression that includes the operations **+**, **-**, *****, **/**.

  - *Loop statements*

    There are 2 types of loop statements. The first type has the following format:

    do {

    //program commands

    }

    while (condition);

    Where the condition is any logical expression containing the comparative operators **>**, **<**, **==**, **!=** and the logical operators **&&** and **||**.

The second type has the following form:

for (expression1**;** expression2**;** expression3**) {**

//program commands

}

where: expression1 is the initialization of the loop variable (e.g. int i = 0), expression2 is a logical expression that defines the loop execution condition (e.g. i < 10), and expression3 is an assignment instruction that modifies the value of the loop variable (e.g. i = i + 15).

- *Control statements*

There are two types of control statements. The first type has the following format:

if (condition1) **{**

// program commands

} else if (condition2) **{**

// program commands

} else if (condition3) **{**

// program commands

}

…

else {

// program commands

}

The else if and else blocks are optional.

The second type has the following form:

```
switch (expression) {
  case expression:
    // program commands
  case expression:
    // program commands
  …
  default:
    // program commands
}
```

The default option is optional.

- *Print statement*:

  Has the form **out.print(**“text”[,var]**);**
  (i.e. text within “” followed immediately by optional variable(s))

- *Return statement for a method*:

  The **return** command is used to return from a method. When it is a method that returns a type, return is followed by an expression and **;**.

- *Loop termination statement*:

  Has the form **break;**

- Language comments are supported at any point in the code. These can be of two forms:
  - Single-line comments: Single-line comments start with the **//** sequence. Anything following // until the line break is a comment and is ignored.
  - Multi-line comments: Multi-line comments are defined with the sequence **/* ... */**. Anything contained within /* ... */ (including line changes) is a comment and is ignored.

- Unless explicitly stated otherwise, spaces and line break characters are not part of the language and play no role in syntax.

Questions

1. (60%)

   a. Present the BNF definition of grammar of the language.

   b. Using the Flex and Bison programs, implement a lexical and syntax analyzer that takes as input a file written in the pseudolanguage described above and checks in one pass whether the program is syntactically correct. Your program will be called from the command line as follows:

   ***prompt*> myParser.exe file_name**

   and will return the program itself on the screen and a diagnostic message as to whether it was correctly written, or an appropriate error message (showing the line where the error is and the analysis should stop there).

2. (10%)

   Modify your analyser to

   a. recognize value assignment when declaring a variable. This is of the form [<modifier>]**<data_type> <variable> = <expression>;**
   For example:

   int var1 = 12;

   public String my_str = “H3ll0”;

   b. recognize the declaration of multiple variables of the same type on a single line. This is of the form

   <data_type> <variable> = <expression> , <variable> = <expression>, <variable> = <expression>, …;

   ή

   <data_type> <variable> , <variable> , <variable>, …;

   For example:

   int x = 15, y = 16, z = 17;

   double x, y, z;

   Present the updated BNF.

3. (20%)

   a. Make the necessary changes to your code to check that the variables used anywhere in the program are declared correctly. In addition, when methods are called, check that they have been defined.
   Otherwise, the analysis should be terminated by displaying a relevant per-case error.

   b. Make the necessary changes to your code to perform scope checking when using variables and methods.

   c. Extend your code so that, if a numeric expression is assigned to a variable, the result of the expression is calculated - following the precedence rules - and an additional message with the name of the variable and the value assigned to it is displayed on the screen.

The numerical expression may include numbers, parentheses, and variables that must have been assigned values in previous commands - otherwise the analysis process is interrupted by the display of an appropriate error.
Note: if this question is implemented, include in your deliverables test files that confirm this functionality.

4. (10%)
   Make the necessary changes to your program so that if an error occurs, the program does not stop running, but continues and all possible errors are identified until the end of the input file.

   Note: if this question is implemented, include in your deliverables both the files related to the case of stopping the analysis when an error is detected and the files related to finding all errors.