Computer Science III
2024-III
WorkShop No. 2 — Compilers

Eng. Carlos Andrés Sierra, M.Sc.
Computer Engineering
Universidad Distrital Francisco José de Caldas

1. For each one of next cases definr a regular expression as used in a compiler based on the Python `re` library:

   (i) **Identifier:** A regular expression to match valid identifiers (variable names, function names, etc.).

   (ii) **Integer Literal:** A regular expression to match integer literals.

   (iii) **Floating Point Literal:** A regular expression to match floating-point literals.

   (iv) **String Literal:** A regular expression to match string literals enclosed in double quotes.

   (v) **Single-line Comment:** A regular expression to match single-line comments starting with '//'.

   (vi) **Multi-line Comment:** A regular expression to match multi-line comments enclosed in '/* */'.

   (vii) **Whitespace:** A regular expression to match whitespace characters (spaces, tabs, newlines).

   (viii) **Operators:** A regular expression to match common operators (e.g., '+', '-', '*', '/', '==', '!=').

   (ix) **Keywords:** A regular expression to match reserved keywords (e.g., 'if', 'else', 'while', 'return').

   (x) **Hexadecimal Literal:** A regular expression to match hexadecimal literals.

2. Be G a `context-free grammar` with the following productions:

Carlos Andrés Sierra, Computer Engineer, M.Sc. on Computer Engineering, Titular Professor at Universidad Distrital Francisco José de Caldas.

Any comment or concern related to this document could be send to Carlos A. Sierra at e-mail: *cavirguezs@udistrital.edu.co*

```
S -> Program
Program -> StatementList
StatementList -> Statement StatementList | <lambda>
Statement -> Assignment | IfStatement | WhileStatement | ReturnStatement
Assignment -> Identifier "=" Expression ";"
IfStatement -> "if" "(" Expression ")" "{" StatementList "}" ElsePart
ElsePart -> "else" "{" StatementList "}" | <lambda>
WhileStatement -> "while" "(" Expression ")" "{" StatementList "}"
ReturnStatement -> "return" Expression ";"
Expression -> Term Expression'
Expression' -> "+" Term Expression' | "-" Term Expression' |
<lambda>
Term -> Factor Term'
Term' -> "*" Factor Term' | "/" Factor Term' | <lambda>
Factor -> "(" Expression ")" | Identifier | Number
Identifier -> [a-zA-Z_][a-zA-Z0-9_]*
Number -> [0-9]+
```

Explanation:

- **S** is the start symbol.
- **Program** consists of a list of statements.
- **StatementList** is a sequence of statements or an empty sequence ($< lambda >$).
- **Statement** can be an assignment, an if statement, a while statement, or a return statement.
- **Assignment** assigns an expression to an identifier.
- **IfStatement** includes an optional else part.
- **WhileStatement** represents a while loop.
- **ReturnStatement** returns an expression.
- **Expression** consists of terms combined with addition or subtraction.
- **Term** consists of factors combined with multiplication or division.
- **Factor** can be an expression in parentheses, an identifier, or a number.
- **Identifier** matches typical variable names.
- **Number** matches sequences of digits.

Based on the provided context-free grammar, create derivation trees for the following statements:

(a) **Exercise 1:**

```
x = 5 + 3 * 2;
```

(b) **Exercise 2:**

```
if (x > 0) {
    y = x − 1;
} else {
    y = 0;
}
```

(c) **Exercise 3:**

```
while (x < 10) {
    x = x + 1;
}
```

(d) **Exercise 4:**

```
return (a + b) ∗ c;
```

Deadline: Saturday, 8th of February, 2025, 14:00 (local time).