Introduction to exercise 2a: Tokenization

CS-460: Programming Language

Robert Bruce

What is a token?

- A token is one or more adjacent bytes which represent fundamental elements of the structure of a programming language.
- A token is a building block for a programming language and is programming language specific.

Why should I care about tokens?

- Before an interpreter can run an input program, the interpreter has to evaluate the programming statements, scope of variables, names of all functions and procedures, etc. This process involves parsing and lexical analysis. Consequently, the process of parsing and lexical analysis ***is much easier*** when the input program is converted into a series of tokens!
- Converting text into a series of tokens is a process called tokenization.

Tokenization: the process

- The first step is removal of comment in the program. One technique for removing comments is to convert each byte of a comment into whitespace. (This includes the characters that define a comment). ***Comments are NEVER tokenized***.
- After comment removal, the input program is then divided up into a series of tokens. The tokens, when read in linear order, represent the input program sans comments.

How are tokens defined?

- As noted previously, tokens are building blocks.
- Statements with a programming language are often comprised of multiple tokens.
- Backus-Naur Form (BNF) is a method for defining the grammar for a programming language.
- BNF can also be used to define the tokens within a programming language.

Examples of Backus-Naur Form (BNF)

```
<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<PLUS> ::= +

<MINUS> ::= -

<WHOLE_NUMBER> ::= <DIGIT> | <DIGIT> <WHOLE_NUMBER>

<INTEGER> ::= <WHOLE_NUMBER> | <PLUS> <WHOLE_NUMBER> | <MINUS>
  <WHOLE_NUMBER>
```

Examples of Backus-Naur Form (BNF)

```
<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<PLUS> ::= +

<MINUS> ::= -

<WHOLE_NUMBER> ::= <DIGIT> | <DIGIT> <WHOLE_NUMBER>

<INTEGER> ::= <WHOLE_NUMBER> | <PLUS> <WHOLE_NUMBER> | <MINUS>
  <WHOLE_NUMBER>
```

In this example, a token is a **DIGIT**, a **PLUS**, a **MINUS**, a **WHOLE NUMBER** or an **INTEGER**.

Example:

```
procedure main (void)
{
  int sum;
  sum = 1 + 2 * 3;
}
```

- String: “procedure”
- String: “main”
- L-paren: “(”
- String: “void”
- R-paren: “)”
- L-brace: “{”
- String: “int”
- String: “sum”
- Semicolon: “;”
- String: “sum”
- Assignment operator: “=”
- Integer (or Whole number): “1”
- Plus: “+”
- Integer (or Whole number): “2”
- Asterisk: “*”
- Integer (or Whole number): “3”
- Semicolon: “;”
- R-Brace: “}”

How will we tokenize our assignments?

There are multiple ways this can be done:

- One method would be to print out the input program on physical paper. Then draw bubbles on the paper to identify the byte or group of bytes which comprise a token. Lastly, one could also write a brief note about the type of token your team identified (e.g. identifier, left parenthesis, equals assignment, etc.). Take a photo of the annotated paper and upload to Canvas.
- Another option would be to import the text as a graphic then digitally highlight or annotate the text with superimposed graphics. Lastly, export the final output as a PDF file or an image.
- A third option would be syntax highlighting (but this method doesn't make it easy to denote the type of token encountered.
- A fourth method would be to physically denote the types of tokens encountered on a separate document. Unfortunately, this can be quite tedious.

Please note: don't forget to write the names of your team members on the document who contributed to this assignment before assignment submission!