

# Computer Engineering 175

## Phase I: Lexical Analysis

*Polonius*: “What do you read, my lord?”

*Hamlet*: “Words, words, words.”

Shakespeare, *Hamlet*, Act II

### 1 Overview

In this assignment, you will write a lexical analyzer for the Simple C language. This assignment is worth 10% of your project grade. Your program is due at 11:59 pm, Sunday, January 9th.

### 2 Lexical Structure

The following points summarize the lexical rules for Simple C:

- spaces, tabs, newlines, and other whitespace are ignored; it is formally defined by the regular expression  $(\_|\backslash t|\backslash n|\backslash f|\backslash v|\backslash r)^+$
- an integer consists of one or more digits; it is formally defined by the regular expression  $[0-9]^+$
- a string consists of zero or more characters enclosed in double quotes ("...") and may not contain a newline or a double quote unless they are preceded by a backslash
- a character consists of one or more characters enclosed in single quotes ('...') and may not contain a newline or a single quote unless they are preceded by a backslash
- an identifier consists of a letter or underscore followed by optional letters, underscores, and digits; it is defined by the regular expression  $[_a-zA-Z][_a-zA-Z0-9]^*$
- keywords are auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, and while; keywords may not be used as identifiers
- operators are =, |, ||, &&, ==, !=, <, >, <=, >=, +, -, \*, /, %, &, !, ++, --, ., ->, (, ), [, ], {, }, ;, :, and ,
- comments are surrounded by /\* and \*/ and may not include a \*/
- any other character is illegal

### 3 Assignment

You will write a simple lexical analyzer for Simple C by reading the **standard input** (std: : cin). Upon recognizing a lexical construct, your program will indicate what it has recognized to the **standard output** (std: : cout) as follows:

- whitespace, comments, and illegal constructs produce no output
- integers are indicated as `number: literal`
- strings are indicated as `string: literal`
- characters are indicated as `character: literal`
- identifiers are indicated as `identifier: identifier`

- keywords are indicated as `keyword: keyword`
- operators are indicated as `operator: text`

For example, the input string `int x;` consists of three tokens: a keyword, an identifier, and an operator. Therefore the output would consist of three lines: `keyword:int`, `identifier:x`, and `operator:;`. Your program will only be given **lexically correct** programs as input. However, it is strongly advised that you test your program against lexically incorrect programs as a way of finding errors in your implementation.

## 4 Hints

Most of the constructs are easily recognized by their first character. For example, a number starts with a digit, and keywords and identifiers start with a letter or underscore. You will find it easiest to use `cin.get()`, and if you wish, `cin.peek()` or `cin.putback()`. Also, you may find the functions such as `isdigit()` defined in `<cctype>` very helpful.

Although the constructs are specified as regular expressions, you will not find it very helpful to use a regular expression library as part of your implementation. Such libraries are designed to match strings held in buffers against regular expressions, not to read an input file and tokenize it.

A string or character literal cannot contain a newline character unless it is preceded by a backslash. They can however contain `\n` (two characters consisting of a backslash followed by the lowercase letter `n`), which is an escape sequence used to represent a newline character, just as `\t` is an escape sequence used to represent a tab character. Since your program will only be given lexically correct input, it is always safe to read and buffer the next character after a backslash.