Compilers (CSC426), Fall 2013
Preliminary Language Definition for YASL

Over the course of the semester you will be writing a compiler for the YASL programming language. YASL stands for **Y**et **A**nother **S**imple **L**anguage. As the semester continues you will see that this name has some historical significance. Although a complete definition of YASL will be delayed until later in the semester, this document provides enough of a definition to allow you to complete project 2 (the lexical analyzer or scanner). Many of the assignments this semester will refer to this document, and you will need to refer to it frequently.

YASL is primarily a subset of the Pascal language (which is itself similar to a simplified version of C++) although some of the Pascal constructs have been replaced by constructs which are modeled after C++. If you know C++ or Pascal, you should find it easy to learn YASL.

I. Lexical Information

1. The following strings have special meaning in YASL:

program function begin end if then else while do cout
cin endl or and div mod int boolean true false

Note that YASL, unlike C++ but like Pascal, is **case insensitive**. The strings listed above have special meaning whether they are written in upper case, lower case, or mixed case.

2. The following input symbols have special meaning in YASL:

== < <= >= > <> << >> + - * { }
( ) = ' ; , . $ & / _ ~

Each of these is a token except for { } ' $ / _

3. Unsigned integer constants are defined as a sequence of at least one and at most four digits. No leading + or - sign is permitted. Note that although negative constants and the unary minus sign are not supported, they can be mimicked with the subtraction operator. For example, instead of writing "X = -7" you can write "X = 0 - 7".

4. Identifiers (ids) are used to name variables, functions, and programs. They consist of an initial letter followed by a sequence of zero or more letters and/or digits and/or underscores. Identifiers may not exceed 12 characters in length. Remember that YASL is case insensitive. Thus "Count" and "COUNT" are the same identifier. Identifiers may not be any of the special strings defined in point "1" above. For example, "div" can't be

an identifier, nor can "DiV".

5. String constants are enclosed within a pair of single quote marks. They have a maximum length of 50 characters and cannot be split across lines. String constants also may not contain an embedded single quote mark. Therefore, the sequence:
'Three blind mice' run'
would be interpreted as the string constant **Three blind mice**, the identifier **run** and the start of another string constant. Note that although string *constants* are permitted, YASL does not support *variables* of type string.

In YASL there is no way to embed a single quote mark inside of a string constant. Unlike C++, the \ has no special meaning inside of a string constant. For example the sequence \n does not mean to go to a new line – it means nothing special at all.

6. The first type of comment consists of zero or more characters which are enclosed between { and }. They may span several lines. The second type of comment starts with a // and extends until a new line (or end of file) is reached, whichever is first.

7. Compiler directives are **exactly** of the form

open-brace dollar-sign single-character-specifier switch closed-brace

The specifier is a single character and its value specifies the type of action to be modified. For example **p** might mean "print." The switch can be either + or - and indicates whether the specifier should be turned on (+) or off (-). For example, the directive {$p+} tells the compiler to turn on the printing option, while {$p-} tells the compiler to turn printing off (this option will be explained later.) Since neither {$p+b} nor {$p +} is an exact match for a valid compiler option it is taken to be a comment. Compiler options are useful for a variety of purposes including debugging your compiler. I may specify additional compiler options in the future, and you should feel free to add your own if they will aid in your debugging efforts.

8. The following situations signify a lexical error. When a lexical *error* is encountered an appropriate message is printed and compilation *terminates*.

a. Any non-white space (blank space, tab, carriage return) symbol **other than those** listed in points (1) - (7) above **unless** the symbol is part of a string constant or comment as defined in points (5) - (6) above. An example of such a symbol is **#**. Reading a } $ or _ out of state 0 is also a lexical error.

b. An integer constant with more than four digits (see point (3) above.)

c. An identifier with length greater than 12 (see point (4) above.)

d. A string constant containing more than 50 characters (not including the quote marks), or one that contains an embedded carriage return or EOF (see point (5) above.)

e. The appearance of the end of the file after reading the opening { of a comment but before reading the corresponding closed }. Note that this only applies to the first style of comments.

f. The appearance of / that is not followed immediately by another / unless the first / appears inside of a comment or a string constant.

g. The appearance of a } other than inside a string constant or a comment that starts with // or as the last symbol of a comment that starts with a {

9. The following situation signifies a lexical warning. When a lexical *warning* is encountered an appropriate message is printed and compilation *continues*.

a. The appearance of a compiler option which is of the correct form except that it uses a character specifier that is not defined. Since you will start off with lower case p being the only valid specifier, both of the following inputs would cause a WARNING to be generated: **{$r+}** **{$P+}** (note the upper case P).

II. Syntactic Information

The syntax of a YASL program will be fully described in a few weeks. There are a few things that you will need to know now however.

The following are Relational Operators (**relops**) < <= == >= > <>

The following are Additive Operators (**addops**) **or** + -

The following are Multiplicative Operators (**mulops**) **and** * **div mod**