**Language Description**

Craig C. Douglas

August 25, 2023

**Introduction**

The F23 language (Fall 2023) will be similar to standard algorithmic languages, but will not follow any one in particular. You will not have an extensive library for input, output, system calls, or memory allocation.

The lexigraphical structure of the language is defined as follows:

* Ignore most white space (blanks, tabs, new lines, and form feeds).
* An identifier is a string made up of letters, digits, dollar signs, and underscores and does not begin with a digit. The maximum number of characters you should pay attention to are the first 32. Ignore everything after that. (Token name is **IDENTIFIER**.)
* A string constant is enclosed in double quotes. (Token name is **SCONSTANT**.)
* Special characters should be treated as C does using a backslash.
* An integer is a sequence of digits 0, ..., 9. (Token name is **ICONSTANT**.)
* A floating point number has 64 bits (i.e., a C or C++ double) and the following forms:
  + Mantissa only:  123., 123.3, 0.3, .3
  + Mantissa and exponent:  0.123d42, 1.23d-3, 0.001d+10, 123.456d000

Note that 1.00d-0 is the same as 1.00d+0 or 1.00d0 or 1.00.  (Token name is **DCONSTANT**.)

* Key words should processed specially. Unless noted, capitalize the key word as its K\_token (e.g., token name **K\_DO** for do).
  + do
  + double
  + else
  + exit
  + function
  + if
  + integer
  + print\_double
  + print\_integer
  + print\_string
  + procedure
  + program
  + read\_double
  + read\_integer
  + read\_string
  + return
  + string
  + then
  + while
* Operators (and punctuation) that need to be recognized are the following:

|  |  |
| --- | --- |
| **Operator** | **Token name** |
| := | **ASSIGN** |
| += | **ASSIGN\_PLUS** |
| -= | **ASSIGN\_MINUS** |
| \*= | **ASSIGN\_MULTIPLY** |
| /= | **ASSIGN\_DIVIDE** |
| %= | **ASSIGN\_MOD** |
| , | **COMMA** |
| // | **COMMENT** |
| && | **DAND** |
| / | **DIVIDE** |
| || | **DOR** |
| == | **DEQ** |
| >= | **GEQ** |
| > | **GT** |
| [ | **LBRACKET** |
| <= | **LEQ** |
| { | **LCURLY** |
| ( | **LPAREN** |
| < | **LT** |
| - | **MINUS** |
| -- | **DECREMENT** |
| % | **MOD** |
| \* | **MULTIPLY** |
| != | **NE** |
| ! | **NOT** |
| . | **PERIOD** |
| + | **PLUS** |
| ++ | **INCREMENT** |
| ] | **RBRACKET** |
| } | **RCURLY** |
| ) | **RPAREN** |
| ; | **SEMI** |

Tokens do not cross new lines. If lex cannot produce a string or number, you do not either. An error message is appropriate if you truncate something, however. An underflow (a floating point number that rounds off to 0 instead of whatever it really is in infinite precision arithmetic), however, is not an error, and should be treated as 0.

There are several special functions for reading and printing data types (double, integer, and string). These will be provided as C code for use by your compiled programs. However, you will have to pass information to these functions correctly. This will be much clearer when you get to the code generation part of the project.

**Defining It by Example**

The easiest way to see what the language really is, is to study one or more examples. On WyoCourses you will find Files/Language/mg.f23. Your compiler will eventually compile this example.

Comments aside, find any bugs or anomalies and report them to me.

**Notes:**

1. Declarations can occur anywhere in a function or procedure **{...}** block. The declarations are only valid within the function or procedure block. After the block ends, all of the variables declared in that block become undefined and memory should be freed automatically.
2. Not all of the key words have been used, but *you should implement them all anyway.*
3. Reading and printing data are special cases that will be discussed in class.
4. A comment is the text following the // to the end of the line, as in C++. The text can be thrown away by the lexer unless you want to save it for some reason (e.g., complete program reconstruction from your parse tree). C style comments (/\* … \*/) should not be implemented.

The class will create examples through the compiler assignments. I will post all of them on WyoCourses. You should compile your classmates’ examples as a check to see if your definition of the F23 language agrees with others. You may not agree as a whole on what the language really is.

**The Machine**

In Files/Language/Makefile are rules for creating an executable program from the output from your compiler, which should be renamed yourmain.h. It assumes that your output produces a callable C function (with an int return value) called yourname() that takes no arguments. Assuming your computer has the make utility installed, you can easily create (called f23) and run an executable program using the make command.

In Files/Language/f23.c is a program are a C main program and the support library for doing simple input, output, math functions, and dynamic memory management. The declarations for the registers and memory are here. Do not modify F23.c without talking to me first. The entire class is expected to use the version of F23.c that is on WyoCourses at all times. When your homework is graded it will be using either the GA’s or my computer and we will use the version on WyoCourses.

**Finally**

I will revise this document during the semester, so you should check to see if you have read the latest version from time to time.