Implementing a VSOP Compiler

Cyril SOLDANI, Pr. Pierre GEURTS

February 22, 2016

Part I

Lexical Analysis

1 Introduction

In this assignment, you will use the lexical analyser generator of your choice to produce a scanner for the VSOP language, according to the lexical rules described in the VSOP manual.

To allow easy testing of your generated lexical analyser, your program will dump the tokens corresponding to input VSOP source code on the standard output, in the format described in section 2.

As VSOP source code files could contain lexical errors, you will need to detect those and print error messages when they occur. The way lexical errors should be handled is not precisely described in the VSOP manual. This documents gives some more information about the way your compiler should handle those errors in section 3.

Finally, the way you should submit your work for evaluation is described in section 4.

This assignment is due at the latest for Wednesday the **2nd of March** 2016 (23:59 CET).

2 Output Format

You will convert the sequence of characters in the source code input file (of which the path will be given to your program as argument) into a stream of tokens.

Each generated token will then be printed on its own line on standard output, according to the following format:

```
token-output-line = LINE, ",", COLUMN, ",", TOKEN-CLASS, [ "," TOKEN-VALUE ]
```

where

LINE is the line number of the first character of the token in the source code (counting from 1).

COLUMN is the column number of the first character of the token in the source code (counting from 1).

TOKEN-CLASS is the kind of token read. You will use the same names as in the VSOP manual, *e.g.* `integer-literal` if the token is an integer literal, `object-identifier` if the token is

an object identifier, *etc.* If the token is a keyword, use the keyword itself as name for the token class.

TOKEN-VALUE is the value of the token.

- For integer literals, output the decimal value (whatever the input format was).
- For type and object identifiers, it is the identifier itself.
- For strings, it will be the character string escaped as follows. The string will be enclosed in double-quotes and each non-printable character (in the ASCII sense, *i.e.* with a byte value lower than 32 or greater than 126) will be replaced by a `\xhh` escape sequence, as described in the VSOP manual (don't bother with `\n`, `\r` and the like). Additionally, the double-quote itself must be escaped as `\"`.
- Other token types have no associated value.

For example, the input source code

```
class MyClass {
    s : string <- "One\n\x1b[33;mTwo and \
                   three\r\n"
    i : int32 <- 0x1b
}
```

should give as output

```
1,1,class
1,7,type-identifier,MyClass
1,15,lbrace
2,5,object-identifier,s
2,7,colon
2,9,string
2,16,assign
2,19,string-literal,"One\x0a\x1b[33;mTwo and three\x0d\x0a"
4,5,object-identifier,i
4,7,colon
4,9,int32
4,15,assign
4,18,integer-literal,27
5,1,rbrace
```

3 Error Handling

All lexical errors will be printed on the standard error stream. They will be prefixed with `FILE_NAME:LINE:COLUMN: lexical error` where `FILE_NAME` is the input file name, as given on the command-line of your program, and `LINE` and `COLUMN` are the line and column of the position of the error as defined below.

It is an error when a character is read that is not a valid VSOP character. Report the error at the position of the faulty character.

It is an error if a multi-line comment is not terminated when end-of-file is reached. Report the error at the position of the opening `(*` tag for that comment.

It is an error if you see an integer literal such as `0xgZ` or `0b0101147`. Report the error at the start of the literal. You can restrict yourself to alphanumeric characters (`[a-zA-Z0-9]`). *I.e.* `0x8g9Z` should report that `0x8g9Z` is not a valid integer literal (rather than returning `integer-literal`

`0x8` followed by `object-identifier` `g9Z`), but `0x2a+` should return a token for `integer-literal` 42, followed by token `plus`.

It is an error if a string literal contains an invalid string character such as a raw line feed. Report the error at the position of the faulty character.

It is an error if a string literal is not terminated when end-of-file is reached. Report the error at the position of the opening double-quote.

Is is an error if a string contains an invalid escape sequence. Report the error at the beginning of the escape sequence (*i.e.* the position of the backslash). Treat `\xhh` escape sequences similarly to integer literals.

4 How to Submit your Work?

Your work should be submitted through the submission platform.

You should submit an archive named `vsopcompiler.tar.xz`. That archive should contain a `vsopcompiler` folder with your code, and a `Makefile` to build it. A skeleton `vsopcompiler` folder and `Makefile` is provided on the assignments web page. You can actually build your compiler with whatever tool you want (if you need building at all), but still provide a `Makefile` with the targets described here.

Your code will be built as follows. From inside your `vsopcompiler` folder, `make install-tools` will be run. This should install any software needed to build your compiler (`make` and `llvm-dev`) are already installed. You can use `sudo` if you need administrator privileges. Then, `make vsopc` will be issued to build your compiler, which should, obviously, be named `vsopc`. Appending the letter `c` (for **C**ompiler) to the name of the language is a common convention for compilers.

Your code will be executed as follows. Your built `vsopc` executable will be called with the arguments `-lex <SOURCE-FILE>` where `<SOURCE-FILES>` is the path to the input VSOP source code. Your program should then output the read tokens on standard output, and eventual error messages on standard error, as described above.

The reference platform is an amd64 machine, powered by Debian Jessie (a GNU\Linux distribution). You can use the virtual machine provided on the assignments web page. Alternatively, you can use `debian/jessie64` with vagrant or `debian:jessie` with docker, or your own Debian-based machine.

We will try to add test scripts to the submission platform that check that your submission is in the right format, and test your lexer. If you want to be able to use that feedback, don't wait until the last minute to submit your lexer (you can submit as many times as you want until the deadline, only the last submission will be taken into account).