# Homework 1
# Readme of
# Simple Java – Scanner

Chih-Hsuan Yang

National Sun-Yet-San University, Taiwan

Instructor: Ye-In Chang

April 17, 2021

# Contents

# 1 Lex version

```
$ lex –version
flex 2.6.4
```

# 2 Operating System

```
$ cat /etc/os-release | sed -En 's/PRET.*"(.*)"/\1/p'

Debian GNU/Linux bullseye/sid
```

# 3 Execution manual

## 3.1 Build

**Symbol table** (Already built, but if you want to compile it by yourself)

```
$ cd lib && mkdir build && cd build
$ cmake ..
$ make
$ mv libsymbol_table.a ..
```

**Lexer**

```
$ flex -o lex.c B073040047.l
$ gcc -c lex.c
$ g++ lex.o lib/libsymbol_table.a -lfl -o my_lex.elf
```

**LaTeX**

```
$ cd Readme
$ make
```

## 3.2 Run

```
$ make # Build it first
$ ./my_lex.elf < input.java
```

**Shared symbol table** Static linking to your program

```
$ gcc lex.c -L./lib -lsymbol_table -lfl -lstdc++ -static -Ofast -o my_lex.elf
$ ./my_lex.elf < input.java
```

# 4 How to implement

Regex in Lex and std::unordered_set in C++

# 5 Problems

1. About the symbol table: the requirement has said:

   > create() 建立一個symbol table。
   > lookup(s) 傳回字串s的index；假如s沒找到的話，就傳回-1。
   > insert(s) 新增s到symbol table中，並傳回存放位置的index。
   > dump() 將symbol table中所有的資料印出。

   (a) About the lookup(s) function:
   Why you need indexing? And how you do the index?
   The symbol table is a set mathematically. A set does not have any argument is called index.
   However, the requirement asked:

   > lookup(s) 傳回字串s的index；假如s沒找到的話，就傳回-1。

   Why?
   How can I return an index of a set?
   What is the index of an element in a set?

   (b) About the create() function:
   This is an implementation issue.
   According to the RAII pattern: In RAII, holding a resource is a class invariant, and is tied to object lifetime: resource allocation (or acquisition) is done during object creation (specifically initialization), by the constructor, while resource deallocation (release) is done during object destruction (specifically finalization), by the destructor.
   Therefore, request a function to construct an object, but not destruct is unreasonable. It would cause the memory leaking issue.

   (c) About the dump() function:
   Well, this is a naming issue.
   We use the word "dump" in programming, that means some error(s) happened, and we would get some information about the "crash" event. However, there is no "crash" happened while we were printing the symbol table, right?
   Hence, we shouldn't take the "dump" to be the function name.
   For clean code, we should named it as "export" or "view". Which give users a decision to handle the exported table, no matter they want to "print", "store" etc.
   Again, we shouldn't use "dump" here.

2. The GNU Flex does not fully support POSIX:
   I tried [:alpha:] for [A-Za-z] in regex. However, the flex does not support.
   And I tried the dollar sign '$' for the "End-of-line". However, the flex does not support too.

3. The test file is DOS format:
   Well, this is not a mistake. I just do not want to precessing these formatting issues. Every UNIX(-like) system are using '\n' as newline. I don't want to add a special for DOS format.

4. The invalid ID is not reasonable and not completeness:
   The regex is followed by "longest match rule", that is, suppose there is a string $S$, which

is concatenated by 2 substring $A$, $B$.

$$S = A \parallel B$$

And, two patterns $P_1$ and $P_2$, which can match $S$ and $A$. Given a string $S$ for these 2 patterns, we will get $S$ in $P_1$ rather than $A$.

Claim: The negation of a regex is almost-surely match all string.

*Proof.* Suppose there are 2 patterns $P$ and $Q$. $\mathbb{P}$ and $\mathbb{Q}$ are the set of regex $P$ and $Q$. The $\mathbb{P} \not\subset \mathbb{Q}$, $\mathbb{Q} \not\subset \mathbb{P}$. And we define a symbol $\neg$ is the inversion set, that is, $\neg\mathbb{P} = \{\mathbb{U} \setminus \mathbb{P}\}$. $\mathbb{U}$ is the universal set of any regex set.

$$\neg\mathbb{P} = \{\mathbb{U} \setminus \mathbb{P}\}$$
$$\Rightarrow \mathbb{Q} \subseteq \neg\mathbb{P}$$

$\forall S = A \parallel B = \neg\mathbb{P}^*$. There are two possibilities: $B = \mathbb{U}^+$ or $B = \epsilon$.
Consider $B = \mathbb{U}^+$:
Obviously, call the regex matching recursively until $B = \epsilon$ go to the next possibility, otherwise:

$$S \notin \mathbb{Q} \wedge S \in \neg\mathbb{P} \tag{1}$$

Consider $B = \epsilon$ :
There are another two possibilities:

(a) The state of $Q$ is terminated ($\perp$)

(b) The state of $Q$ is not terminated ($\not\perp$).

If $Q$ is $\perp$: Both $S \in \neg\mathbb{P}$ and $S \in \mathbb{Q}$ are correct.

$$S \in \neg\mathbb{P} \wedge S \in \mathbb{Q} \tag{2}$$

If $Q$ is $\not\perp$: Only if $S \in \neg\mathbb{P}$

$$S \notin \mathbb{Q} \wedge S \in \neg\mathbb{P} \tag{3}$$

And if we want to get $S$ from regex $Q$, only if $\|S\| = \|Q\|$. However,

$$P(\|S\| = \|Q\|) = 0$$

$$\Rightarrow P(S = \{\exists\omega \in \neg\mathbb{P} \mid \lim_{n\to\infty} \neg\mathbb{P}_n(\omega)\}) = 1$$

By definition, that is almost-surely convergence to $S \in \neg\mathbb{P}$. ■

To simplify in one table:

| | $\mathbb{Q}$ | $\mathbb{Q}^\perp$ |
|---|---|---|
| $B$ | Recursion | $\neg\mathbb{P}$ |
| $B^\perp$ | $\neg\mathbb{P}$ | $\neg\mathbb{P}$ or $\mathbb{Q}$ |

# 6 Outcome

```
rm -rf lib/build lex.o my_lex.elf lex.c
flex -o lex.c B073040047.l
gcc -c lex.c
g++ lex.o lib/libsymbol_table.a -lfl -o my_lex.elf
./my_lex.elf < TestFile_Lab1_2021/Test1.java
Line: 2, 1st char: 1, "public" is a "reserved word".
Line: 2, 1st char: 8, "class" is a "reserved word".
Line: 2, 1st char: 14, "Test1" is a "ID".
Line: 2, 1st char: 20, "{" is a "symbol".
Line: 3, 1st char: 5, "public" is a "reserved word".
Line: 3, 1st char: 12, "static" is a "reserved word".
Line: 3, 1st char: 19, "int" is a "reserved word".
Line: 3, 1st char: 23, "add" is a "ID".
Line: 3, 1st char: 26, "(" is a "symbol".
Line: 3, 1st char: 27, "int" is a "reserved word".
Line: 3, 1st char: 31, "a" is a "ID".
Line: 3, 1st char: 32, "," is a "symbol".
Line: 3, 1st char: 34, "int" is a "reserved word".
Line: 3, 1st char: 38, "b" is a "ID".
Line: 3, 1st char: 39, ")" is a "symbol".
Line: 3, 1st char: 41, "{" is a "symbol".
Line: 4, 1st char: 9, "return" is a "reserved word".
Line: 4, 1st char: 16, "a" is a "ID".
Line: 4, 1st char: 18, "+" is a "operator".
Line: 4, 1st char: 20, "b" is a "ID".
Line: 4, 1st char: 21, ";" is a "symbol".
Line: 5, 1st char: 5, "}" is a "symbol".
Line: 7, 1st char: 5, "public" is a "reserved word".
Line: 7, 1st char: 12, "static" is a "reserved word".
Line: 7, 1st char: 19, "void" is a "reserved word".
Line: 7, 1st char: 24, "main" is a "reserved word".
Line: 7, 1st char: 28, "(" is a "symbol".
Line: 7, 1st char: 29, ")" is a "symbol".
Line: 7, 1st char: 31, "{" is a "symbol".
Line: 9, 1st char: 9, "int" is a "reserved word".
Line: 9, 1st char: 13, "c" is a "ID".
Line: 9, 1st char: 14, ";" is a "symbol".
Line: 10, 1st char: 9, "int" is a "reserved word".
Line: 10, 1st char: 13, "a" is a "ID".
Line: 10, 1st char: 15, "=" is a "operator".
Line: 10, 1st char: 17, "5" is a "integer".
Line: 10, 1st char: 18, ";" is a "symbol".
Line: 11, 1st char: 9, "c" is a "ID".
Line: 11, 1st char: 11, "=" is a "operator".
Line: 11, 1st char: 13, "add" is a "ID".
Line: 11, 1st char: 16, "(" is a "symbol".
Line: 11, 1st char: 17, "a" is a "ID".
Line: 11, 1st char: 18, "," is a "symbol".
Line: 11, 1st char: 20, "10" is a "integer".
Line: 11, 1st char: 22, ")" is a "symbol".
Line: 11, 1st char: 23, ";" is a "symbol".
Line: 12, 1st char: 9, "if" is a "reserved word".
Line: 12, 1st char: 12, "(" is a "symbol".
Line: 12, 1st char: 13, "c" is a "ID".
Line: 12, 1st char: 15, ">" is a "operator".
Line: 12, 1st char: 17, "10" is a "integer".
Line: 12, 1st char: 19, ")" is a "symbol".
Line: 13, 1st char: 13, "print" is a "reserved word".
Line: 13, 1st char: 18, "(" is a "symbol".
Line: 13, 1st char: 19, ""c = "" is a "string".
Line: 13, 1st char: 26, "+" is a "operator".
Line: 13, 1st char: 28, "-" is a "operator".
Line: 13, 1st char: 29, "c" is a "ID".
Line: 13, 1st char: 30, ")" is a "symbol".
Line: 13, 1st char: 31, ";" is a "symbol".
Line: 14, 1st char: 9, "else" is a "reserved word".
Line: 15, 1st char: 13, "print" is a "reserved word".
Line: 15, 1st char: 18, "(" is a "symbol".
Line: 15, 1st char: 19, "c" is a "ID".
Line: 15, 1st char: 20, ")" is a "symbol".
Line: 15, 1st char: 21, ";" is a "symbol".
```

```
Line: 16, 1st char: 9, "print" is a "reserved word".
Line: 16, 1st char: 14, "(" is a "symbol".
Line: 16, 1st char: 15, ""Hello World"" is a "string".
Line: 16, 1st char: 28, ")" is a "symbol".
Line: 16, 1st char: 29, ";" is a "symbol".
Line: 18, 1st char: 5, "}" is a "symbol".
Line: 20, 1st char: 1, "}" is a "symbol".
The symbol table contains:
b
a
c
add
Test1
./my_lex.elf < TestFile_Lab1_2021/Test2.java
Line: 1, 1st char: 1, "// this is a comment // line */ /* with /* delimiters */ before the end
" is a "comment".
Line: 3, 1st char: 1, "public" is a "reserved word".
Line: 3, 1st char: 8, "class" is a "reserved word".
Line: 3, 1st char: 14, "Test2" is a "ID".
Line: 3, 1st char: 20, "{" is a "symbol".
Line: 4, 1st char: 5, "int" is a "reserved word".
Line: 4, 1st char: 9, "i" is a "ID".
Line: 4, 1st char: 11, "=" is a "operator".
Line: 4, 1st char: 13, "-100" is a "integer".
Line: 4, 1st char: 17, ";" is a "symbol".
Line: 5, 1st char: 5, "double" is a "reserved word".
Line: 5, 1st char: 12, "d" is a "ID".
Line: 5, 1st char: 14, "=" is a "operator".
Line: 5, 1st char: 16, "12.25e+6" is a "float".
Line: 5, 1st char: 24, ";" is a "symbol".
Line: 7, 1st char: 5, "public" is a "reserved word".
Line: 7, 1st char: 12, "static" is a "reserved word".
Line: 7, 1st char: 19, "void" is a "reserved word".
Line: 7, 1st char: 24, "main" is a "reserved word".
Line: 7, 1st char: 28, "(" is a "symbol".
Line: 7, 1st char: 29, ")" is a "symbol".
Line: 7, 1st char: 31, "{" is a "symbol".
Line: 8, 1st char: 1, "/* this is a comment // line with some /* and
// delimiters */" is a "comment".
Line: 10, 1st char: 5, "}" is a "symbol".
Line: 11, 1st char: 1, "}" is a "symbol".
The symbol table contains:
d
i
Test2
./my_lex.elf < TestFile_Lab1_2021/Test3.java
Line: 2, 1st char: 1, "public" is a "reserved word".
Line: 2, 1st char: 8, "class" is a "reserved word".
Line: 2, 1st char: 14, "Test3" is a "ID".
Line: 2, 1st char: 20, "{" is a "symbol".
Line: 3, 1st char: 5, "int" is a "reserved word".
Line: 3, 1st char: 9, "A" is a "ID".
Line: 3, 1st char: 10, ";" is a "symbol".
Line: 4, 1st char: 5, "int" is a "reserved word".
Line: 4, 1st char: 9, "a" is a "ID".
Line: 5, 1st char: 5, "double" is a "reserved word".
Line: 5, 1st char: 12, "b" is a "ID".
Line: 5, 1st char: 13, ";" is a "symbol".
Line: 6, 1st char: 5, "double" is a "reserved word".
Line: 6, 1st char: 12, "A" is a "ID".
Line: 6, 1st char: 13, ";" is a "symbol".
Line: 8, 1st char: 5, "public" is a "reserved word".
Line: 8, 1st char: 12, "Test3" is a "ID".
Line: 8, 1st char: 17, "(" is a "symbol".
Line: 8, 1st char: 18, ")" is a "symbol".
Line: 8, 1st char: 20, "{" is a "symbol".
Line: 9, 1st char: 9, "a" is a "ID".
Line: 9, 1st char: 11, "=" is a "operator".
Line: 9, 1st char: 13, "1" is a "integer".
Line: 9, 1st char: 14, ";" is a "symbol".
Line: 10, 1st char: 9, "A" is a "ID".
Line: 10, 1st char: 11, "=" is a "operator".
Line: 10, 1st char: 13, "2" is a "integer".
```

```
Line: 10, 1st char: 14, ";" is a "symbol".
Line: 11, 1st char: 9, "b" is a "ID".
Line: 11, 1st char: 11, "=" is a "operator".
Line: 11, 1st char: 13, "-1.2" is a "float".
Line: 11, 1st char: 17, ";" is a "symbol".
Line: 12, 1st char: 5, "}" is a "symbol".
Line: 13, 1st char: 1, "}" is a "symbol".
The symbol table contains:
b
a
A
Test3
```