B093040024
郭晏涵

編譯器製作 Homework 1

Readme of Simple Java - Scanner

Lex 版本：

```
$ lex --version
flex 2.6.4
```

作業平台：

Ubuntu 22.04.2 LTS

執行方式：

```
$ make
flex B093040024.l
gcc lex.yy.c -o demo -lfl
$ ./demo < Test2.java
```

makefile

```
FILE_lex = B093040024.l
PROG_lex = lex.yy.c
all:    $(PROG_lex)
    gcc $(PROG_lex) -o demo -lfl

$(PROG_lex):    $(FILE_lex)
    flex $(FILE_lex)

clean:
    rm demo $(PROG_lex)
```

如何處理這份規格書上的問題：

1. Symbols

```
symbol [,:;\(\)\{\}\[\]]
```

若辨識到[...]中任一字元，則將此 symbol 傳回給 parser。其中「(」、「)」、「[」、「]」、「{」和「}」為特殊字元，前面須加上跳脫字元\。

2. Operators

```
operator \+\+|--|==|>=|<=|!=|&&|\|\||<<|>>|>>>|[+\-*\/%=<>!&|^~]
```

前半部分是將需要兩個以上字元表示的 operator 以|符號連接，代表只要 match 到任一個即可。接著以|符號連接[...]，[...]中放的是單一字元的 operator。其中「+」、「-」、「|」、「/」等特殊字元前面須加上跳脫字元\。

3. Reserved Words

```
reserve_word abstract|boolean|break|byte|case|catch|char|class|
const|continue|default|do|double|else|extends|false|final|
finally|float|for|goto|if|implements|import|instanceof|int|
interface|long|main|native|new|package|print|private|protected|
public|return|short|static|string|super|switch|synchronized|
this|throw|throws|transient|true|try|void|volatile|while
```

Java 的 reserved words 都是由兩個字元以上字元組成，將這些 reserved words 以|符號連接，代表只要 match 到任一個即可。

4. Identifiers

```
id [A-Za-z_\$][A-Za-z0-9_\$]*
```

Java 的 identifier 第一個字元可以是一個大寫或小寫的英文字母，也可以是「_」或「$」符號，因此將這些符號放入[...]中。後面的字元可以是英數字或「_」、「$」符號，且字數不限，因此將這些符號放入[...]中，並在後面加上*，表示重複 0-∞次。

```
invalid_id [0-9^#]+[a-zA-Z_\$]+
```

不合法的 identifier 可能為數字開頭，或是以「^」、「#」開頭，因此將這些符號放入[...]中，並在後面加上+，表示至少出現一次。

5. Integer Constants

```
integer ([+-][ ]*)?(((0[Xx])[A-Fa-f0-9]+)|([0-9]+))
```

整數的符號和數值之間可能存在多個空格，以[ ]*表示，若是正數可不加「+」，因此將辨識正負號與空格的部分以(...)括起來，並在後面加上?，代表括號內的內容可出現 1 次或 0 次。接著可以加上「0x」或「0X」，表示十六進位。數字則出現至少一個，因此在[0-9]後面加上+。

6. Float Constants

```
float ([+-][ ]*)?([0-9]*\.[0-9]+)|([0-9]+\.[0-9]*)|[0-9]+([Ee][+-]?[0-9]+)?[Ff]?
```

浮點數判斷符號和空格的方式與整數相同。數值的部分有「.2」、「2.」、「2.0」三種呈現方式，實作上區分為兩種判斷方法，([0-9]*\.[0-9]+)可判斷「.2」和「2.0」這兩種寫法，([0-9]+\.[0-9]*)則符合「2.」、「2.0」這兩種寫法。此外，若後面加上科學符號表示法 E 或 e，甚至可以不需要小數點，因此加上第三種判斷方法[0-9]+。

雖然純數字的 token（例如：「87」）也符合上述的正規語言定義，但 integer 的 Lex Rule 擺放位置在 float 之上，因此兩者 match 長度相同的情況下，純數字的 token 仍會被辨識為整數。

最後若加上 F 或 f，代表此浮點數是 float 而不是 double。

7. String Constants

```
str \"([^\"\\\n\r]|\\['\"\\btnfr0-7])*\"
```

開頭和結尾的\"為字串前後的雙引號。字串中可包含多個字元，也可以是空字串，因此在(...)後面加上*。中間不能出現換行符號、回車符號、單一個「"」或是「\」，這部分以[^\"\\\n\r]表示，[^...]代表只 match [...]內以外的字元。而「\t」、「\n」、和「\0」等為特殊字元，且若要在字串中呈現「'」、「"」或「\」必須加上跳脫字元，這些規則以\\['\"\\btnfr0-7]表示。

以上的正規語言定義會將前後的雙引號包含進去，但字串常數其實不包含前後雙引號，因此 scanner 輸出時會省略前後各一個字元。

```
invalid_str \"([^\"\r\n])*
```

不合法的字串只有一個雙引號（例如："ab）。根據以上規則，「"""」會被判定成一個合法的空字串「""」和一個不合法字串「"」。

8. Characters Constants

```
character '([^'\\\n\r]|\\['\"\\btnfr0-7])'
```

由於規格書上沒有提到字元（例如：'a'）屬於哪一類，因此我將其獨立出來判斷。單引號內基本上只能有一個字元，且此字元不能是換行符號、回車符號、「'」或是「\」，這部分以[^'\\\n\r]表示。而「\t」、「\n」、和「\0」等為特殊字元，且若要在字串中呈現「'」、「"」或「\」必須加上跳脫字元，這些能夠在單引號內放入兩個字元的規則，以\\['\"\\btnfr0-7]表示。

與字串常數相同的是，字元常數不包含前後的單引號，因此 scanner 輸出時會省略前後各一個字元。

```
invalid_char '([^\'\r\n])*
```

不合法的字元只有一個單引號(例如：'a 或'ab)。根據以上規則，「'ab'」會被分別判定成兩個不合法的字元表示法「'ab」和「'」。

9. Whitespace

```
space [ \t]
eol \r?\n
```

我將 whitespace 區分為 space 和 end of line 來判別，因為兩者的 Lex Rules 不同。space 包含空白和 tabs，遇到只需要將計算字元數的變數 charCount 加一；若是遇到換行符號（前面可能接著一個回車符號），則須將計算行數的變數 lineCount 加一，並將 charCount 設回 1。

10. Comments

```
comment (\/\/[^\n\r]*)|(\/\*([^*]|\*+[^*\/])*\*+\/)
```

註解分為可跨行與不可跨行。\/\/[^\n\r]*辨識不可跨行的註解，「//」的兩個字元前面各需加上一個跳脫字元，後面可出現數個換行符號和回車符號以外的字元。

雖然投影片中提供的可跨行註解的解法為\/*(.|\n)**\/，但此寫法在某些情況下無法辨識出正確的註解結尾，因此我將寫法改成\/*([^*]|*+[^*\/])**+\/，思考過程如下：

原寫法：\/*(.|\n)**\/

Problem：若「*/」後面存在另一個「*/」，則第一個「*/」會被.|\n 辨識，造成註解被錯誤地繼續往後找。

Solution：讓.|\n 辨識出「*」的後面不能接「/」。

改良寫法 1：\/*([^*]|*[^\/])**\/

Problem：stars 會"成對"被*[^\/]辨識，若結尾有偶數個 stars(例如：「****/」)，會導致正確的註解結尾「*/」的「*」必定跟著前面的「*」一起被*[^\/]辨識 (也就是「**|**|/」)，而剩下的「/」又被[^*]辨識，造成註解被錯誤地繼續往後找。

Solution：讓[^\/]不能辨識到結尾「*/」的「*」。

改良寫法 2：\/*([^*]|*[^*\/])**\/

Problem：中間若有連續兩個以上的 stars 就會辨識失敗，因為「**」既不符合*[^*\/]，也不符合*\/。

Solution：讓*辨識連續兩個以上的 stars。

改良寫法 3：\/*([^*]|*+[^*\/])**\/

Problem：結尾若有連續兩個以上的 stars 還是會辨識失敗，因為「**/」既不符合*+[^*\/]，也不符合*\/。

Solution：讓*\/也可以辨識連續兩個以上的 stars。

最終寫法：\/*([^*]|*+[^*\/])**+\/

Conclusion：[^*\/]防止「*/」的「*」被辨識成註解內容，*+防止中間連續的 stars 辨識失敗，*+\/防止結尾連續的 stars 辨識失敗。

過程中遇到的問題：

1. 如何辨識加減號還是正負號

因為 scanner 會 scan 出長度最長的 token 去進行匹配，因此接在 0-9 前面的+/-可能會被誤判為正負號。在 Java 裡 integer、float、identifier 和 string 是可以作為運算元，接在這四種型態的 tokens 後面的+/-應代表加減號，因此建立一個變數 sub 紀錄前一個傳回給 parser 的 token 是否屬於上述四種類別，若 scan 出一個有帶符號的數字，且 sub 為 True，則將符號作為 operator，並與數字分開印出。

值得注意的是，symbol 中的小括號「(...)」可作為運算式集合運算子，因此當 scan 出一個「)」時並不能將 sub 的值改為 false。

2. 單行註解辨識問題

我在測試 Test.java 時曾經產生下圖結果：

```
cse132@cse132-virtual-machine:~/Documents/Compiler/lex$ ./demo < Test2.java
" is a "comments".1, "// this is a comment // line */ /* with /* delimiters */ before the end
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".
```

可以看到第一行輸出到註解的結尾時，游標會移到此行的開頭繼續輸出，這是因為原本判斷單行註解的正規語言寫法\/\/.*只排除換行符號，回車符號會被判斷進去。因此我將寫法改成\/\/[^\n\r]*來避免此狀況。

3. Symbol Tables 實作

我以 linked list 實作 symbol tables，每一個 identifier 的名稱與 index 儲存在一個節點中。程式開始執行時，symbol tables 並沒有任何資料，因此 create()只回傳一個空指標。每當偵測到一個 identifier，會呼叫 insert()檢查此 identifier 是否已存在於 symbol tables 中，若無則會動態配置一個新節點來儲存。

insert()並不會不直接呼叫 lookup()來確認 identifier 是否存在，因為 lookup()對於不存在的輸入只會回傳-1，但 insert()需要 symbol tables 中最後一個 identifier 的 index，並將此 index 加一來獲得新節點的 index，也需要將最後一個節點的 next 指標指向新節點，此過程會遍歷過一次 symbol tables，同時檢查 identifier 是否存在。

測試檔執行結果：

Test1.java

```
cse132@cse132-virtual-machine:~/Documents/Compiler/lex$ ./demo < 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: 16, "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:
Test1
add
a
b
c
```

Test2.java

```
cse132@cse132-virtual-machine:~/Documents/Compiler/lex$ ./demo < 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 an "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 an "ID".
Line: 4, 1st char: 11, "=" is an "operator".
Line: 4, 1st char: 13, "-100" is an "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 an "ID".
Line: 5, 1st char: 14, "=" is an "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:
Test2
i
d
```

Test3.java

```
cse132@cse132-virtual-machine:~/Documents/Compiler/lex$ ./demo < 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 an "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 an "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 an "ID".
Line: 5, 1st char: 5, "double" is a "reserved word".
Line: 5, 1st char: 12, "b" is an "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 an "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 an "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 an "ID".
Line: 9, 1st char: 11, "=" is an "operator".
Line: 9, 1st char: 13, "1" is an "integer".
Line: 9, 1st char: 14, ";" is a "symbol".
Line: 10, 1st char: 9, "A" is an "ID".
Line: 10, 1st char: 11, "=" is an "operator".
Line: 10, 1st char: 13, "2" is an "integer".
Line: 10, 1st char: 14, ";" is a "symbol".
Line: 11, 1st char: 9, "b" is an "ID".
Line: 11, 1st char: 11, "=" is an "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:
Test3
A
a
b
```