Simple Java -- Scanner

這份作業是使用 lex 寫一個 Simple Java 的 scanner。 你 (妳) 的 scanner 要能處理所有的保留字 (reserved words) 、符號 (symbols) 和註解 (comments)。如果處理到不合法的 token，你 (妳) 的 scanner 要產生 error message (error detection)。你 (妳) 的 scanner 要盡可能地處理完所有的輸入 (input)。如果發生錯誤，產生 error message，之後能繼續處理其它的輸入 (error recovery)。

Scanner 的輸出 (output) 會列出每個 token 以及此 token 的型態 (integer, float, ID, reserved word, string, operator ,symbol, comment) 、此 token 所在的 line no.、第一個字元所在的位置。你要自己建立一個 symbol table，此 symbol table 會記錄所有找到的 Identifier (ID)（詳見 6）

1 What to Sumbit

你必須繳交下列檔案：

- Scanner，檔名為 – 你/妳的學號.l
- 你/妳的測試檔
- makefile 檔
- 一個 Readme 檔，裡面包含
  - Lex 版本
  - 作業平台
  - 執行方式
  - 你/妳如何處理這份規格書上的問題
  - 你/妳寫這個作業所遇到的問題
  - 所有測試檔執行出來的結果，存成圖片或文字檔

請用壓縮軟體將上述這些檔案壓縮成一個檔案，檔名為 – 你/妳的學號，寄給助教

信件主旨請寫：[Compiler]Scanner of 你/妳的學號

以下所述是關於這個 Simple Java 的 lexical definition。

2 Character Set

Simple Java 是由 ASCII 的字元所構成。Simple Java 的語言定義不使用控制字元 (Control characters)。

3 Lexical Definitions

Tokens 分成兩類：(1) 會傳給 parser 的 tokens; (2) 會被 scanner 直接丟棄的 tokens (會被辨識出來，但不會傳給 parser)。

3.1 會傳給 parser 的 tokens

Symbols

每一個 symbol 會被當成一個 token 傳回給 parser。

Comma	,
Colon	:
Semicolon	;
Parentheses	( )
square brackets	[ ]
Brackets	{ }

Arithmetic, Relational, and Logical Operators

每一個 operator 會被當成一個 token 傳回給 parser。

addition	+ ++
subtraction	- --
multiplication	*
division	/ %
assignment	=
relational	< <= >= > == !=
logical	&& || !

Keywords

下列這些 keywords 是 Simple Java 的 reversed words (大小寫有區別)。

boolean, break, byte, case, char, catch, class, const, continue, default, do, double, else, extends, false, final, finally, float, for, if, implements, int, long, main, new, print, private, protected, public, return, short, static, string, switch, this, true, try, void, while …

上列的每一個 keyword 都會被當作 token 傳回給 parser。

Identifiers

一個識別字 (identifier) 是一個字元和數字組合而成的字串，而且開頭必須是一個字元。字元有區分大小寫，例如：`peter` 、 `Peter` 和 `PETER`，這是三個不同的識別字。Keywords 不能拿來當作識別字。你可以把 function name 當作一個 Identifier。

Integer Constants

一個或多個數字組合而成，且 integer 有正負之分。例如：`100, -200`。

Float Constants

有正負之分，且有小數點 (decimal point) 表示法和科學符號表示法兩種。例如：

- `1.0, 3.14`
- `12.25e+6, -2E-2`

String Constants

一個 string constant 是由一對雙引號 (") 括起來的零個或多個 ASCII 字元所組合而成。如果雙引號是 string constant 的一部分的話，在 string constant 必須使用加上反斜線來表示一個雙引號。例如：`"aa\"bb"` 表示 string constant – `aa"bb`。

3.2 會被 scanner 直接丟棄的 tokens

下列的 tokens 會被 scanner 辨識出來，但是會直接被丟棄，而不會傳給 parser。

Whitespace

空白、tabs、換行。

Comments

Comments 有兩種表示方法：

- C-style：由 /* 和 */ 所包含的文字，可以跨行。
- C++-style：由 // 和跟在其後的文字所組成，不能跨行。

例如：

```
// this is a comment // line */ /* with /* delimiters */ before the end
```

和

```
/* this is a comment // line with some /* and

// delimiters */
```

都是合法的 comments。

4 Recovery

你/妳的 scanner 必須儘可能處理完 input。在發生錯誤時，要能 recover，能夠繼續處理其它的輸入。

5 Symbol Tables

你必須實作 symbol tables 來儲存所有的 identifiers。Symbol tables 應該被設計成容易新增和擷取資料，所以通常會使用 hash tables 來實作。為了能夠建立和管理 symbol tables，至少要提供下列這些 functions：

`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。

6 你的 scanner 要能做些什麼事？

Scanner 的輸出 (output) 會列出每一個 token 以及此 token 的型態 (integer, float, ID, reserved word, string, operator ,symbol, comment) 、此 token 所在的 line no.、第一個字元所在的位置。最後，你必須要印出 symbol table 中的所有 Identifier，舉個例子來講，假設有下列這些程式碼：

```
// print hello world
{
  print("hello world");
  int a = 5 + 5.5;
}
```

你的 scanner 將會輸出下列的結果： (以下 reserved word 可視為 keyword)

Line: **1**, 1st char: **1**, "**// print hello world"** is a "**comment".**
Line: **2**, 1st char: **1**, **"{"** is a "**symbol"**.
Line: **3**, 1st char: **3**, **"print"** is a "**keyword"**.
Line: **3**, 1st char: **8**, **"("** is a **"symbol"**.
Line: **3**, 1st char: **10**, **"hello world"** is a "**string"**.
Line: **3**, 1st char: **21**, **")"** is a **"symbol"**.
Line: **3**, 1st char: **22**, **";"** is a "**symbol"**.
Line: **4**, 1st char: **3**, **"int"** is a "**reserved word"**.
Line: **4**, 1st char: **7**, **"a"** is an "**ID"**.
Line: **4**, 1st char: **9**, **"="** is a "**operator"**.
Line: **4**, 1st char: **11**, **"5"** is a "**Integer"**.
Line: **4**, 1st char: **13**, **"+"** is a "**operator"**.
Line: **4**, 1st char: **15**, **"5.5"** is a "**real"**.
Line: **4**, 1st char: **16**, **";"** is a "**symbol"**.
Line: **5**, 1st char: **1**, **"}"** is a "**symbol"**.
The symbol table contains:
a

***lex* sample code**

```
%{
#define MAX_LINE_LENG 256
#define LIST strcat(buf,yytext)
#define token(t) {LIST; printf("<%s>\n",t);}
#define tokenInteger(t,i) {LIST; printf("<%s:%d>\n",t,i);}
#define tokenString(t,s) {LIST; printf("<%s:%s>\n",t,s);}
int linenum = 0;
char buf[MAX_LINE_LENG];
%}
%%
"(" {token('(');}
[0-9]+ {tokenInteger(yytext, atoi(yytext));}
\n {
    LIST;
    printf("%d: %s", linenum++, buf);
    buf[0] = '\0';
    }
[ \t]* {LIST;}
. {
    LIST;
    printf("%d:%s\n", linenum+1, buf);
    printf("bad character:'%s'\n",yytext);
    exit(-1);
    }
%%
main(){
yylex();
return 0;
}
```