

113-2 國立成功大學編譯系統課程

NCKU Compiler Construction - 2024 Spring

Homework 1. Lexical Analysis (Scanner)

本學期的作業配分

- 這學期總共有 3 份作業，作業成績佔學期成績 40%
 - Homework 1: 10%
 - Homework 2: 15%
 - Homework 3: 15%



作業目標

- 實作出一個簡易的 Rust Compiler
- 三份作業的內容合起來就是一個可以動的 Compiler
- 備註：第二份作業與第三份作業有依賴關係，
作業三需使用到作業二的 code
- 作業模板

： <https://github.com/ColtenOuO/113-2-NCKU-CompilerH>

[W1](#)

實作的目標語言：輕量版 Rust

- 作業會將 Rust 的語法與特定簡化，有些語法與原始不同
 - `println!()` -> `println()`
- 詳細可以參考作業的測試資料
- 輕量版 Rust: 靜態強型別語言
 - 靜態 (Static Typing): 編譯時確認型態，檢驗型態匹配
 - 強型別 (Strong Typing): 不會自動做型別轉換

Homework 1: Lexical Analysis (Scanner)

- 這是實作 Compiler 的第一個步驟!
- 我們要先把程式碼切成一小片一小片的東西 (Token)
 - `println("Hello World\n");`
 - `println`
 - `(`
 - `"`
 - 以此類推

Homework 1: Lexical Analysis (Scanner)

- 切下來的 Token 會被分為兩種類型
 - 有用的 Token (對程式來說有功能的東西)
 - 沒用的 Token (需忽略掉, 像是註解裡面的東西)

Homework 1: Lexical Analysis (Scanner)

- 在這個作業你必須實作一個 Scanner 掃描出所有 Token
- 並且輸出掃描時的相關資訊
- 輸入：一段程式碼
- 輸出：掃描結果

```
fn main() { // Your first rust program
    println("Hello World!"); // equivalent to println!("Hello World!"); in rust
    /* Hello
    World */ /*
    */
}
```

```
fn          FUNC
main        IDENT
(           LPAREN
)           RPAREN
{           LBRACE
// Your first rust program      COMMENT
NEWLINE
println     PRINTLN
(           LPAREN
"           QUOTA
Hello World!  STRING_LIT
"           QUOTA
)           RPAREN
;           SEMICOLON
// equivalent to println!("Hello World!"); in rust  COMMENT
NEWLINE
/* Hello
   World */      MUTI_LINE_COMMENT
/*
   */           MUTI_LINE_COMMENT
NEWLINE
}              RBRACE

Finish scanning,
total line: 6
comment line: 5
```

Homework 1: Lexical Analysis (Scanner)

- 實作使用 C 語言
- Scanner 工具：Flex
 - 安裝指令：apt install flex
(出現權限問題請記得 sudo)
 - 此工具能幫忙匹配我們想找的 Token

Homework 1: Lexical Analysis (Scanner)

- 基礎模板 (scanner.l) 已經幫大家寫好,

你需要寫的在 Rule Section 區塊

- 此區塊會照你寫的規則:

(大括號內為匹配到時要做的事情)

```
/* Rules section */  
%%  
"+"      { print_token("ADD"); }  
"-"      { print_token("SUB"); }  
<<EOF>>  { line_num++; yyterminate(); }  
[ \t]+    {;}  
.  
%%
```

Homework 1: Lexical Analysis (Scanner)

- 掃描完成後會輸出以下資訊

(請在掃描的過程中一邊更新相關變數, 使其正確)

- 該程式碼的行數 (line_num)
- 註解數量 (comment_num)

```
printf("\nFinish scanning,\n");  
printf("total line: %d\n", line_num);  
printf("comment line: %d\n", comment_num);
```

Homework 1: Lexical Analysis (Scanner)

- 以下列出所有你需要解析的 Token

Delimiters

Delimiters	Symbols
Parentheses	<code>() [] { }</code>
Semicolon	<code>;</code>
Comma	<code>,</code>
Quotation	<code>" "</code>
Newline	<code>\n</code>

Operators

Operators	Symbols
Arithmetic	<code>+ - * /</code>
Relational	<code>< > <= >= == !=</code>
Assignment	<code>= += -= *= /= %=</code>
Logical	<code>&& !</code>
Bitwise	<code>& ^ >> <<</code>

Types	Keywords
Data type	<code>i32 f32 bool char</code>
Conditional	<code>if else for while loop</code>
Variable declaration	<code>let</code>
Functional	<code>fn return</code>

Homework 1: Lexical Analysis (Scanner)

- IDENT:
以字母與數字或底線組成，不使用數字開頭且非關鍵字
(宣告變數時的變數名稱就是 IDENT 的一種)

Homework 1: Lexical Analysis (Scanner)

- INT_LIT: 整數
- FLOAT_LIT: 浮點數
- STR_LIT: 字串

let

x

:

i32

=

3

;

NEWLINE

LET

IDENT

COLON

INT

ASSIGN

INT_LIT

SEMICOLON

Homework 1: Lexical Analysis (Scanner)

- 除此之外，註解也是必須的

```
fn          FUNC
main        IDENT
(           LPAREN
)           RPAREN
{           LBRACE
// Your first rust program      COMMENT
      NEWLINE
println     PRINTLN
(           LPAREN
"           QUOTA
Hello World!  STRING_LIT
"           QUOTA
)           RPAREN
;           SEMICOLON
// equivalent to println!("Hello World!"); in rust  COMMENT
      NEWLINE
/* Hello
   World */      MUTI_LINE_COMMENT
/*
   */           MUTI_LINE_COMMENT
      NEWLINE
}             RBRACE

Finish scanning,
total line: 6
comment line: 5
```

Homework 1: Lexical Analysis (Scanner)

- 所有 Token 的輸出內容可參考測試資料的答案
- 模板內有寫好一兩個 Token 給大家參考

```
/* Rules section */  
%%  
"+"      { print_token("ADD"); }  
"-"      { print_token("SUB"); }  
<<EOF>>  { line_num++; yyterminate(); }  
[ \t]+    {;}  
.  
%%
```

Symbol	Token		Symbol	Token		Symbol	Token
+	ADD	-	&&	LAND	-	print	PRINT
-	SUB	-		LOR	-	println	PRINTLN
*	MUL	-	!	NOT	-	if	IF
/	QUO	-	(LPAREN	-	else	ELSE
%	REM	-)	RPAREN	-	for	FOR
>	GTR	-	[LBRACK	-	i32	INT
<	LSS	-]	RBRACK	-	f32	FLOAT
>=	GEQ	-	{	LBRACE	-	..	DOTDOT
<=	LEQ	-	}	RBRACE	-	bool	BOOL
==	EQL	-	;	SEMICOLON	-	true	TRUE
!=	NEQ	-	,	COMMA	-	false	FALSE
=	ASSIGN	-	"	QUOTA	-	let	LET
+=	ADD_ASSIGN	-	\n	NEWLINE	-	mut	MUT
-=	SUB_ASSIGN	-	:	COLON	-	fn	FUNC
*=	MUL_ASSIGN	-	Int Number	INT_LIT	-	return	RETURN
/=	QUO_ASSIGN	-	Float Number	FLOAT_LIT	-	break	BREAK
%=	REM_ASSIGN	-	String Literal	STRING_LIT	-	as	AS
&	BAND	-	Identifier	IDENT	-	in	IN
	BOR	-	Comment	COMMENT / MULTI_LINE_COMMENT	-	while	WHILE
~	BNOT	-	->	ARROW	-	loop	LOOP
>>	RSHIFT	-	<<	LSHIFT			

Homework 1: Lexical Analysis (Scanner)

● 作業提供 local judge 給大家自行測試

- pip3 install local-judge

```
student1@sivslab-System-Product-Name:~/rust_compiler/lab01$ judge
local-judge: v2.7.2
```

Sample	Accept
a00_helloworld_comment	✓
a01_arithmetic	✓
a02_precedence	✓
a03_scope	✓
a04_assignment	✓
a05_casting	✓
a06_if	✓
a07_while	✓
a08_function	✓
a09_array	✓
a10_error	✓
b01_auto_type_detection	✓
b02_loop	✓
b03_foreach	✓
b04_2d_array	✓
b05_slice	✓

```
Correct/Total problems: 16/16
Obtained/Total scores: 112/112
```

Homework 1: Lexical Analysis (Scanner)

- 模板的 makefile 可以自行更改
- 直接輸出測資答案將不予給分
- 輸出 Token 時請直接使用我們 define 好的 print_token 函式輸出，避免遇到格式不一致的問題

作業繳交

- 將 makefile 與 scanner.l 壓縮成同一個 zip
- 並將名稱命名為 學號.zip 上傳 moodle

Homework 1: 作業提示 1

- Flex 支援正則表達式，只要一開始先定義好 (類似變數宣告)
接下來在切 Token 的時候就可以直接使用
- 某些 Token 你會需要好好善用這一個東西
- 定義方式：[名稱] [正則表達式]
- 使用時：將名稱用大括號包起來即可使用
- Example:
 - 定義：digit [0-9]
 - 使用：{digit}

Homework 1: 作業提示 2

- Flex 有以下兩種規則
- 最長匹配：不用擔心 \leq 會被拆解成 $<$ 跟 $=$ ，因為 Flex 會盡可能的找到 Token 長度最長的去做匹配
- 先定義先匹配：如果有一段字串同時被 2 個 Token 規則匹配，而且這兩個 Token 長度一樣，那麼 Flex 會選先被定義的那一個 Token

Homework 1: 作業提示 3

- 這份作業最困難的地方是多行註解的處理
- 善用 `yymore()`，他的功能是把當前讀到的東西先丟到緩衝區去，之後要拿出來時會一次全部丟給 `yytext`
- 在註解的部分善用 `yymore()` 會輕鬆很多！

Homework 1: 作業提示 4

- Flex 有 state 的功能
- 你可以設立一個 state, 的起始條件跟終止條件, 這些條件可以放入指定的 Token, 協助你判斷更多東西
- 像是多行註解你就可以把起始 Token 設定成 /* 終止 Token 設定成 */
- 最後再搭配 yymore() 就好寫很多了

Homework 1: 作業提示 5

- 簡單來說可以想像，如果遇到起始 Token，就會開始進入那一個 state 裡面做事情
- 使用 state 之前，記得先將你的 state 做宣告 (%x state名稱)

```
起始Token                { BEGIN(state 名稱); }  
<state 名稱>終止 Token    { BEGIN(INITIAL); }  
<state 名稱>state 內判斷的 Token { 要做的事情 }  
<state 名稱>state 內判斷的 Token { 要做的事情 }  
<state 名稱>state 內判斷的 Token { 要做的事情 }
```


Homework 1: 作業提示 6

- Flex 語法資源：
 - 找 Stackoverflow
 - <https://westes.github.io/flex/manual/>