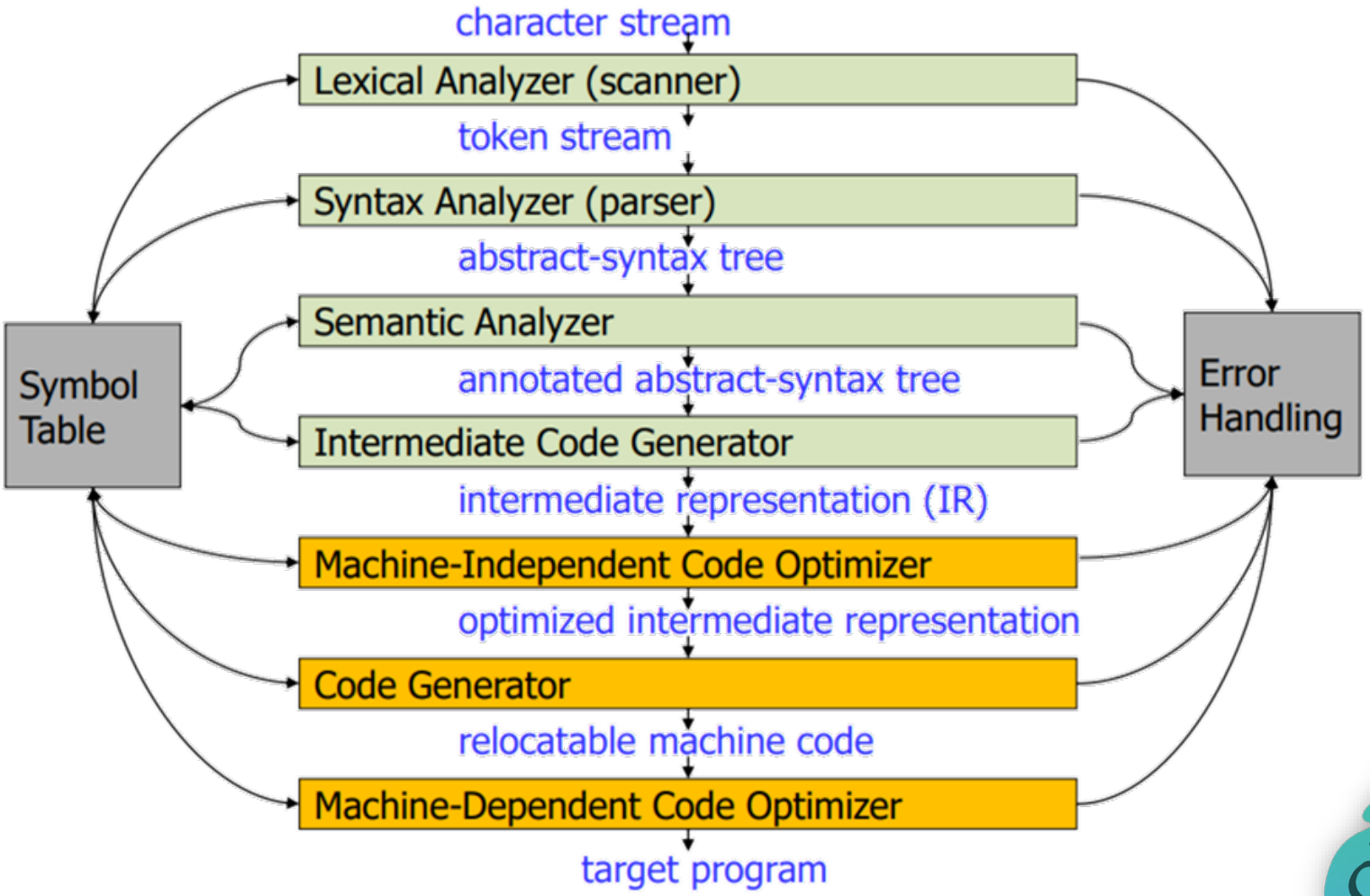
編譯器製作

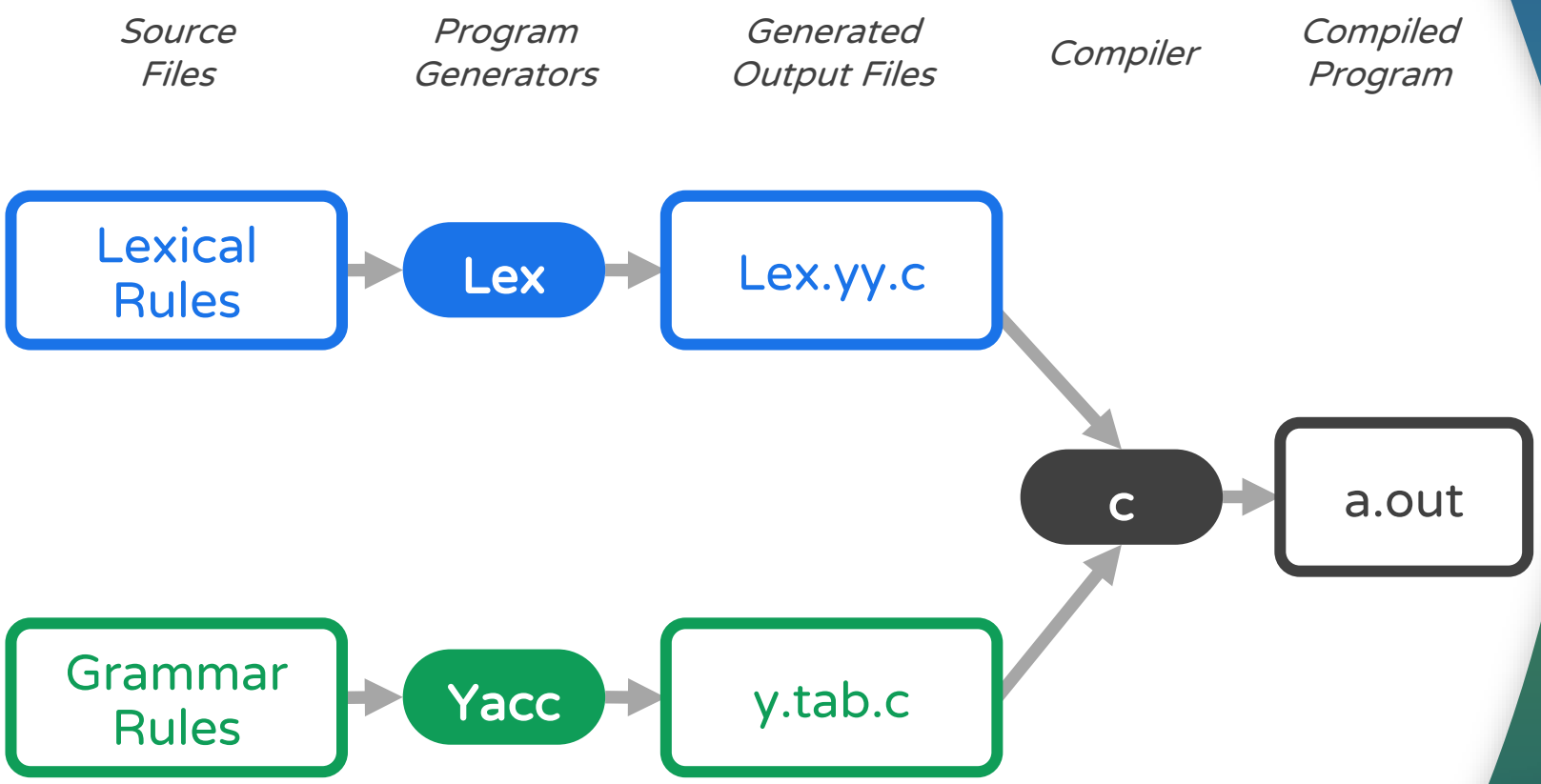
Yacc Parser

助教| 李冠廷 CansCurtis.com

The Structure of a Compiler



Lex & Yacc


Source Files
Program Generators
Generated Output Files
Compiler
Compiled Program
Lexical Rules
Lex
Lex.yy.c
Grammar Rules
Yacc
y.tab.c
c
a.out

Lex & Yacc

Lex	Yacc
Lexical Analyzer	Yet Another Compiler Compiler
比對正規語言 所描述的字串	判斷句子 是否符合語法
將輸入資料 切為小單位 Token	將 Token 以邏輯方式加以組織

Yacc 的工作

目的

- Yacc 的目的是「檢查語法是否合法」

運作

- Yacc 會把 Input 當作 A Sequence of Tokens
  - 一個以上連續的 Tokens 會形成 **Grammar**
- Lex 只是 Yacc 的一個 Routine
  - 負責回傳 Token 給 Yacc

Yacc 語法

- 假設現在要做一個簡單的計算機的 Parser
- 設計語法，其中 NUMBER 為 Lex 抓到的 Token

```
expression -> NUMBER
expression -> expression + NUMBER
expression -> expression - NUMBER
```

- 同一個 LHS 可以合併在一起，再用 | 隔開每個 RHS
- 以上語法，在 Yacc 會被表示成

```
expression : NUMBER
           | expression + NUMBER
           | expression - NUMBER
```

Yacc 無法處理的狀況

- Yacc 無法處理 Ambiguous 的語法
  Yacc 無法處理 需要參考一個以上 Token 的語法
- 請改寫語法，或確保優先順序 (%left)

```
Pharse -> cart_animal AND CART
        | work_animal AND PLOW

cart_animal -> HORSE | GOAT
work_animal -> HORSE | OX
```

```
Pharse -> cart_animal CART
        | work_animal PLOW
```

Yacc 程式

Yacc 格式

分成三部分，每個部分以 %% 區隔

```
Definition
%%
Grammars
%%
User Code
```

Definition

calc.y

```
%{
#include <stdio.h>
int yylex();
double ans = 0;
void yyerror(const char* message) {
    printf("Invaild format\n");
};
%}

%union {
    float   floatVal;
    int     intVal;
}
%type <floatVal> NUMBER
%type <floatVal> expression term factor group
%token PLUS MINUS MUL DIV
%token LP RP
%token NUMBER NEWLINE

%%
```

Definition

calc.y

```
%{
#include <stdio.h>
int yylex();
double ans = 0;
void yyerror(const char* message) {
    printf("Invaild format\n");
};
%}
```

Definition

calc.y

```
%union {
    float   floatVal;
    int     intVal;
}

%type <floatVal>   NUMBER
%type <floatVal>   expression term
                   factor group

%token PLUS MINUS MUL DIV
%token LP RP
%token NUMBER NEWLINE

%%
```

%type 通常宣告
Non-Terminals

%token 通常宣告
Terminals

Definition

NUMBER 雖然是 Terminal
但是我們一樣需要知道他的實際數值！

```
%type <floatVal>   NUMBER
%type <floatVal>   expression term
                   factor group
```

%type 目的

除了把 Token 的 “類別” 傳入 Yacc，
我們也許需要知道他的 “實際數值” (通常是 Non-Terminals)，
所以需要在這邊定義 Token 的數值型態，
讓 Lex 可以透過存入 yylval，來將實際數值傳入 Yacc

Grammars

calc.y

```
%%
lines :/* empty */
    | lines expression NEWLINE {printf("%lf\n", $2);}
    ;
expression : term { $$ = $1; }
    | expression PLUS term { $$ = $1 + $3; }
    | expression MINUS term { $$ = $1 - $3; }
    ;
term : factor {  $$ = $1; }
    | term MUL factor { $$ = $1 * $3; }
    | term DIV factor { $$ = $1 / $3; }
    ;
factor : NUMBER { $$ = $1;}
    | group {$$ = $1; }
    ;
group : LP expression RP { $$ = $2; }
    ;
%%
```

Grammars

calc.y

```
expression : term { $$ = $1; }
```

$$ $1

```
    | expression PLUS term { $$ = $1 + $3; }
```

$1 $2 $3

```
    | expression MINUS term { $$ = $1 - $3; }
```

$1 $2 $3

```
    ;
```

User Code

calc.y

```
%%

int main() {
    yyparse();
    return 0;
}
```

修改 Lex 程式

Definition

calc.l

```
%{
#include "y.tab.h"
#include <stdio.h>
%}

Digit [0-9]+

%%
```

Grammars

calc.l

```
%%

{Digit}       { sscanf(yytext, "%f",
                &yylval.floatVal); return NUMBER;}
\+            {return PLUS;}
\-            {return MINUS;}
\*            {return MUL;}
\/            {return DIV;}
\(            {return LP;}
\)            {return RP;}
\n            {return NEWLINE;}
.             {return yytext[0];}

%%
```

使用 Yacc

如何使用 Lex File

- 首先必須安裝flex這個程式來編譯我們的lex file

```
sudo apt-get install bison
```

<- 以ubuntu為例

- 編譯 cau.y　　(產生 y.tab.c 及 y.tab.h)

```
bison -y -d cau.y
```

- 編譯 cau.lex　　(產生 lex.yy.c)

```
flex cau.l
```

- 透過gcc產生可執行檔 (產生calc這個執行檔)

```
gcc lex.yy.c y.tab.c -ly -lfl -o calc
```

- 執行方式

```
./calc < testfile
```

編譯流程

- 在Example中，有幫大家寫好 makefile 可以參考

```
all:   clean y.tab.c lex.yy.c
       gcc lex.yy.c y.tab.c -ly -lfl -o calc

y.tab.c:
     bison -y -d cau.y

lex.yy.c:
     flex cau.l

clean:
     rm -f calc lex.yy.c y.tab.c y.tab.h
```

- 執行「make all」即可編譯產生 「calc」

Error Handling

test1.java

INPUT

```
3++9
5/2
9*3
3+5
4**6
5+***6+*6
4+3*9-10*8
```

OUTPUT

```
** Syntax Error at Line 1 **
Line 2: 5 / 2
Line 3: 9 * 3
Line 4: 3 + 5
** Syntax Error at Line 5 **
** Syntax Error at Line 6 **
Line 7: 4 + 3 * 9 - 10 * 8
```

執行結果 – Test1

test1.java

INPUT

```
/* Test file: Perfect test file
 * Compute sum = 1 + 2 + ... + n
 */
class sigma {
  // "final" should have
const_expr
  final int n = 10;
  int sum, index;

  main()
  {
    index = 0;
    sum = 0;
    while (index <= n)
    {
      sum = sum + index;
      index = index + 1;
    }
    print(sum);
  }
}
```

OUTPUT

```
line  1: /* Test file: Perfect test file
line  2:  * Compute sum = 1 + 2 + ... + n
line  3:  */
line  4: class sigma {
line  5: // "final" should have const_expr
line  6: final int n = 10 ;
line  7: int sum , index ;
line  8:
line  9: main ( )
line 10: {
line 11: index = 0 ;
line 12: sum = 0 ;
line 13: while ( index <= n )
line 14: {
line 15: sum = sum + index ;
line 16: index = index + 1 ;
line 17: }
line 18: print ( sum ) ;
line 19: }
line 20: }
```

執行結果 – Test2

test2.java

INPUT

```
/* Test file: ... */
class Point
{
    static int counter ;
    int x, y ;
    /*Duplicate declare x*/
    int x ;
    void clear()
    {
        x = 0 ;
        y = 0 ;
    }
}
```

OUTPUT

```
line  1: /* Test file:  ... */
line  2: class Point
line  3: {
line  4: static int counter ;
line  5: int x , y ;
line  6: /*Duplicate declare x*/
line  7: int x ;
> 'x' is a duplicate identifier.
line  8: void clear ( )
line  9: {
line 10: x = 0 ;
line 11: y = 0 ;
line 12: }
line 13: }
```

執行結果 – Test3

test3.java

INPUT

```
/* Test file of ... */
class Point {
    int z;
    int x y ;
    /*Need  ',' before y*/
    float w;
}
class Test {
    int d;
    Point p = new Point()
    /*Need ';' at EOL*/
    int w,q;
}
```

OUTPUT

```
line  1: /* Test file of ... */
line  2: class Point {
line  3: int z ;
Line  4, char: 12, a syntax error at "y"
line  4: int x y ;
line  5: /*Need  ',' before y*/
line  6: float w ;
line  7: }
line  8: class Test {
line  9: int d ;
line 10: Point p = new Point ( )
Line 10, char: 17, statement without semicolon
line 11: /*Need ';' at EOL*/
line 12: int w , q ;
line 13: }
```

關於作業II

推薦的環境

- Cygwin
  - 一個類 Linux 環境
  - 直接在 Windows 上操作，不須虛擬機
  - 記得安裝 `bison`, `flex`, `m4` 這三個 Packages
- 在虛擬機上裝 Ubuntu
  - Ubuntu 22

大家應該都有修這學期希家老師的 UNIX
也乖乖做好第一個作業了… 對吧XDD

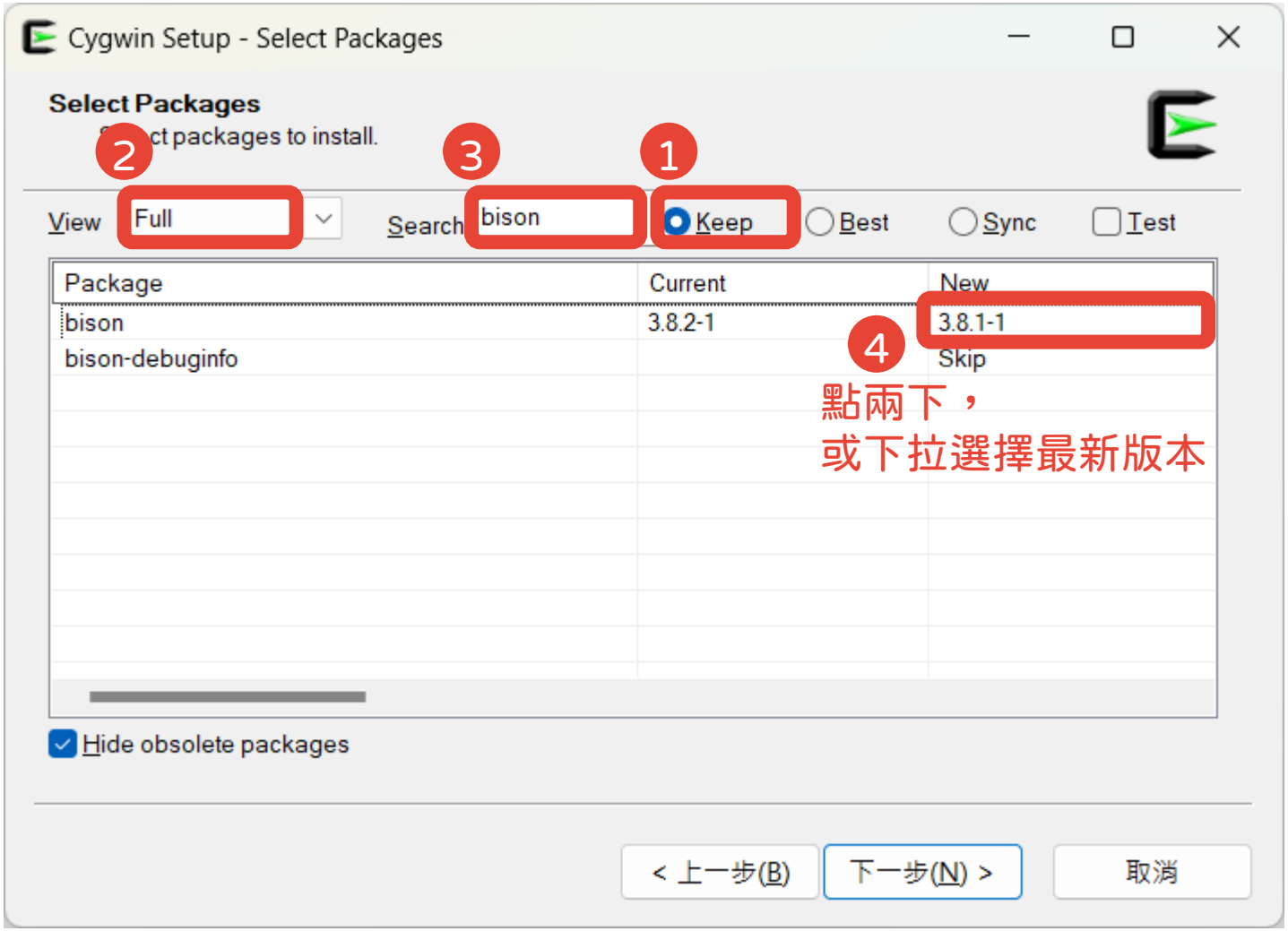
關於 Cygwin 的提示

- 請記得在安裝 Cygwin 時，一併安裝以下這些 Packages
  - tcsh
  - ncurses
  - dos2unix
  - Emacs
  - nano
  - gcc-core
  - gcc-g++
  - make
  - bison
  - flex
  - m4

這些不在 UNIX 課程的安裝教學裡
請特別留意

關於 Cygwin 的提示

- 如果你已經安裝好 Cygwin ，
  但是有 Packages 沒有安裝到，
  別擔心！不用重新安裝！
- 請到 Cygwin 官網下載安裝檔 (setup-x86_64.exe)
  - https://www.cygwin.com/install.html
- 執行安裝檔，下一步直到 “Select Packages”
- View 選 “Full”，右邊選擇 “Keep”
  接下來透過 Search，搜尋並選取所有需要的內容
  - 可參考下頁


Cygwin Setup - Select Packages
Select Packages
ct packages to install.
2
3
1
View
Full
Search
bison
Keep
Best
Sync
Test
Package
Current
New
bison
3.8.2-1
3.8.1-1
bison-debuginfo
Skip
4
點兩下，
或下拉選擇最新版本
Hide obsolete packages
< 上一步(B)
下一步(N) >
取消

作業繳交注意事項

- **DUE DATE: 2023年05月21日 23:59**
- Yacc 的設計要比 Lex 要複雜很多，因此請馬上開始撰寫。
- 程式 Demo 環境是 Cygwin / Ubuntu 22.04.2 LTS，
  因此請保證你們的程式碼能夠在至少其中一個編譯執行
- 請參考課程網頁中的測試檔案來驗證你的程式
- 請準時繳交作業，作業遲交分數打七折
- 請把作業包成一個壓縮包，上傳至**網路大學**，
  檔名命為「**學號_hw2.zip**」。
- 在繳交截止後，會安排時間 Demo **(5/22~5/26)**，
  請準時到 EC5023 資料庫系統實驗室 找助教 Demo。

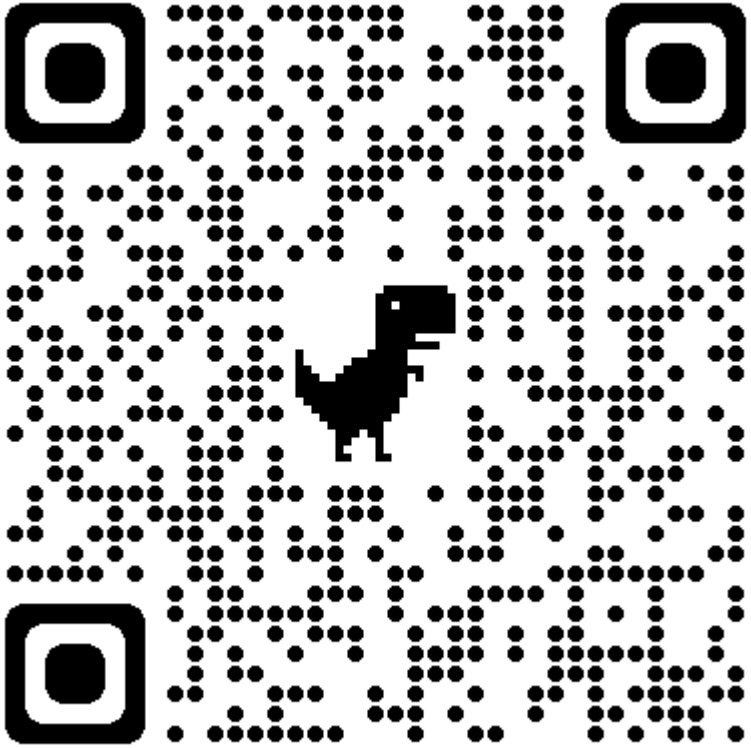
必須考慮下列這些問題

- 你的 parser 在遇到 error 時，要能產生出好的 error messages。
  - 例如：發生 error 的行號、字元的位置和解釋 error 發生的原因。
- 當 parser 遇到 error 時，要盡可能的處理完輸入。
  - 也就是說 parser 要遇到 error 要做 recovery。

評分方式

每測資 20%	6個公開測資中隨機挑3個， 必須與題目的錯誤訊息相同 (錯誤訊息可用不同表示方式)
每測資 10%	2個隱藏測資， 從公開測資中隨機排列組合
5%	註解 ： 解釋如何處理各個 Statements
5%	Readme.pdf (內容請參考作業說明第一頁)
5% + 5%	口頭問答*2
~%	Bonus

DEMO 時段登記



- 帳號：學號
  密碼：sesame
- 登入後請盡速變更密碼
- 程式 Demo 及 口頭問答

https://booking.canscurtis.com/event/compiler2

如果有何問題

沒事的，歡迎詢問助教：

李冠廷
Curtis@CansCurtis.com
EC5023 資料庫系統實驗室

- 請附上你的 學號 + 姓名
  以及你的「學號.l、學號.y」程式檔案
- 直接私訊我，也是可以的啦~