Simple Java – Parser

這份作業是寫一個 Simple Java 的 Parser。你/妳必須寫出符合下列各節的句法定義 (Syntactic definitions) 的 grammar。一旦你/妳定義好這些 grammar，就可以將這些 grammar 代入 **yacc**，使用 **yacc** 來產生一個"**y.tab.c**"的 c 檔案 (這個 c 檔案裡頭包含 **yyparse()**)。**yyparse()** 會呼叫 **yylex()** 來取得 token，所以你/妳需要修正你/妳的第一個作業 –Scanner – 來讓 **yyparse()** 取得 token。

完整版的 JAVA 文法結構：

(http://db.cse.nsysu.edu.tw/~changyi/slides/compiler/lab/Java.doc)

你 (妳) 必須考慮下列這些問題：

(a) 你 (妳) 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。

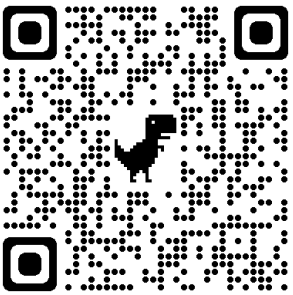
(b) 當 parser 遇到 error 時，要盡可能的處理完輸入，也就是說 parser 要遇到 error 要做 recovery。

1 What to Submit

你必須繳交下列檔案：

- ✧ 修改後的 Scanner，檔名為 – **你/妳的學號.l**
- ✧ 你/妳的 Parser，檔名為 – **你/妳的學號.y** (裡面需有註解，解釋如何處理 statements。)
- ✧ 你/妳的測試檔
- ✧ 所有的 .c 和 .h 檔 makefile 檔
- ✧ 一個 Readme.pdf 檔，裡面包含
  - ▫ Lex, Yacc 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

Demo 時段登記



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_hw2**

2 Syntactic Definitions

下列這些 syntactic definitions 只是片段，你/妳必須自己想出能符合下列 syntactic definitions 的 grammar，來完成你/妳的作業。

2.1 Data Types and Declarations

基本的資料型態有 **boolean, char, int, float,** 和 **String**。一個變數的宣告如下列的格式：

[*static*] *type identifier_list;*

identifier_list →
identifier [= *const_expr*] {, *identifier* [= *const_expr*]}

例如：

- int a, b, c = 10;
- int a = 10;
- int b, c = 2;
- int d = 1 + 2;
- static boolean b;

陣列的宣告如下列格式（在這個作業中，我們只考慮一維陣列，並不考慮 Assignment 的動作）：

type[] *identifier* = new *type*[*integer_constant*];

例如：

- int[] a = new int[10];

常數的宣告格式（final）：

final *type identifier_list*;

identifier_list →
identifier = *const_expr* {, *identifier* = *const_expr*}

例如： final float pi = 3.14;

注意事項

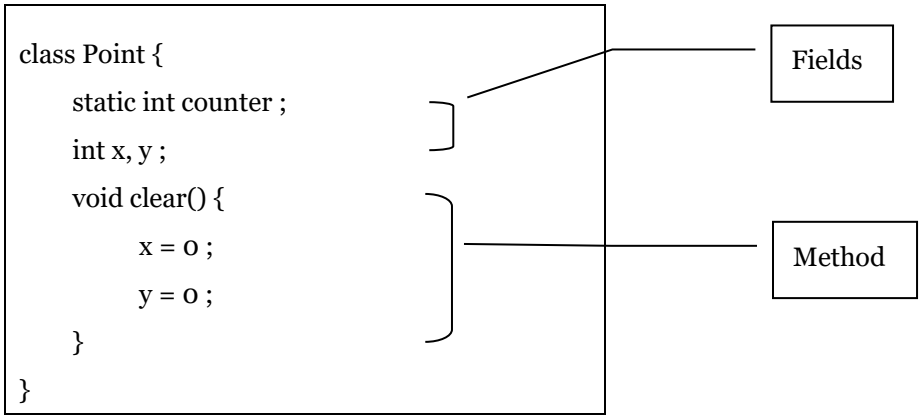
- [x]代表 x 會出現 0 或 1 次。
- {x} 代表 x 會出現 0 次或以上。
- x | y 表示 x 或 y 其中一個。

Classes and Objects

每個 object 有一個 type，這個 type 就是 object 的 class。每個 class type 有兩種成員：

✧ Fields are data variables associated with a class and its objects.

✧ Methods contain the executable code of a class.



同一個檔案裡可以有多個 classes。

建立 objects

使用 **new** 這個 keyword 來建立 objects。

```
Point lowerLeft = new Point() ;
Point upperRight = new Point() ;
```

Fields

有兩種 fields：

✧ class fields (static fields)，如 `static int counter ;`

✧ instance fields (non-static fields)，如 `int x, y ;`

2.2 Methods

一個 method declaration 如下列的格式：

method_modifier type identifier({zero or more formal arguments})
one compound statement

method_modifier →
public | **protected** | **private**

即使沒有宣告 arguments，括號還是要有。在一個 method 之內，不能再宣告 method。一個 formal argument 的格式如下：

type identifier

如果有多個 formal arguments 的話，要以 ',' (comma) 加以區隔。

Methods 可能會回傳一個值或不回傳值。假如一個 method 不回傳值的話，那這個 method 的 type 會是 **void**。例如，下列這些都是合法的 method declaration：

```
boolean func1(int x, int y, String z) {}
String func2(boolean a) {}
void func3() {}
```

每個 method 的名字都是獨一無二的。

2.3 Statements

有六種不同種類的 statements：compound, simple, conditional, loop, return, and method call。

2.3.1 Compound

A compund statement consists of a block of statements delimited by the **{** and **}**, and an optional variable and constant declaration section :

```
{
    {zero or more variable and constant declaration}
    {zero or more statements}
}
```

在 compound statement 內宣告的 variables 和 constants 有區域性，離開這個 compound statment 之後，這些 variables 和 constants 就失效了。

一個 compound statment 的例子：

```
{
    int a ;
    read(a) ;
    print(a) ;
}
```

2.3.2 Simple

simple →

name = *expression* **;** |

print(expression**) ;** |

read(*name***) ;** |

name++ **;** |

name-- **;** |

expression ; |

;

name →

identifier |

identifier.*identifier*

expressions

expression →

term |

expression + *term* |

expression – *term*

term →

factor {* *factor* | / *factor*}

factor →

identifier |

const_expr |

(*expression*) |

PrefixOp identifier |

identifier PostfixOp |

MethodInvocation

PrefixOp →

++ |

-- |

+ |

-

PostfixOp →

++ |

--

例如：

- a + -b
- (1+2)*3
- b + add(c, d) ;

method invocation

一個 method 呼叫的格式如下：

name({expressions separated by zero or more comma})

2.3.3 Conditional

if (*boolean_expr***)** one simple or compound statement

{**else** one simple or compound statement}

boolean_expr →

expression Infixop expression

Infixop →

== |

!= |

< |

> |

<= |

>=

2.3.4 Loop

Loop statement 的格式如下：

while (*boolean_expr***)**
one simple or compound statement

或

for (*ForInitOpt* ;*boolean_expr* ; *ForUpdateOpt*)
one simple or compound statement

ForInitOpt →
[**int**] *identifier* = *expression* {, *identifier* = *expression*}

ForUpdateOpt →
identifier ++ |
identifier --

例如：

```
int sum = 0, i =1 ;
while ( i <= 10) {
      sum = sum + i ;
       i = i + 1 ;
}
```

```
for (int index = 0; index < 10; index++) {
      if (list[index] > max) {
            max = list[index];
      }
}
```

2.3.5 return

return statement 的格式如下：

return *expression* ;

2.3.6 Method Invocation

name({expressions separated by zero or more comma}) ;

3 Semantic Definition

你/妳的 Parser 必須能做簡單的 Semantic Definition 的檢查 – 同一個 scope 內，不能宣告兩個相同的變數。

例如：

```
{
    int a ;
    float a ;
}
```

在這個 scope 內，宣告了兩個 a 的變數，這是不合法的，你/妳的 Parser 要能偵測的出來。

4 Error and Recovery

你（妳）必須考慮下列這些問題：

(a) 你（妳）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。

(b) 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。

5 配分方式

(a) 6 個公開測資，我會挑出 3 個一模一樣，全對才給分（各 20%）

(b) 2 個隱藏測資，從公開測資中隨機排列組合（各 10%）

(c) 註解：解釋如何處理 statements (5%)

(d) Readme.pdf (5%)

(e) 口頭問答 (5% * 2)

(f) Bonus (有做出 variable is never used) (5%)

6 Example Simple Java Program

- 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: }
```

- test2.java
  - ✧ input

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

  - ✧ output

```
line  1: /*Test file: Duplicate declare
     variable in the same scope*/
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: }
```

- test3.java

  (註：Line 10 應該是少 Semicolon，但把錯誤出現在 Line 12 的 int 也可以接受，也就是至少有一個 error 發生)

  ✧ intput

```
/*Test file of Syntax errer: Out of symbol.  But it can go through*/
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 Syntax errer:
           Out of symbol.
           But it can go through*/
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: }
```

test4.java

- input

```
/*Test file: Duplicate declaration in different scope and same scope*/
class Point
{
    int x, y ;
    int p;
    boolean test()
    {
        /*Another x, but in different scopes*/
        int x;
        /*Another x in the same scope*/
        char x;
        {
            boolean w;
        }
        /*Another w, but in different scopes*/
        int w;
    }
}
class Test
{
    /*Another p, but in different scopes*/
    Point p = new Point();
}
```

- output

```
   line  1: /*Test file: Duplicate declaration in different scope and
same scope*/
   line  2: class Point
   line  3: {
   line  4: int x , y ;
   line  5: int p ;
   line  6: boolean test ( )
   line  7: {
   line  8: /*Another x, but in different scopes*/
   line  9: int x ;
   line 10: /*Another x in the same scope*/
   ******'x' in the next line is a duplicated identifier in the
current scope.******
   line 11: char x ;
   line 12: {
   line 13: boolean w ;
   line 14: }
   line 15: /*Another w, but in different scopes*/
   line 16: int w ;
   line 17: }
   line 18: }
   line 19: class Test
   line 20: {
   line 21: /*Another p, but in different scopes*/
   line 22: Point p = new Point ( ) ;
   line 23: }
```

- test5.java

  - input

```
class test5{
    int add(int a1, int a2){
        return (a1 + a2);
    }
    void main() {
        int x, y, z;
        for(int i=0;i<2;i++){
            if(i==0){
//--------------ELSE WITHOUT IF
                else
                    i = 1;
            }
            for(x = 0; x<5;x++){
                y++;
//--------------FUNCTION CALL
                x = add(x,y);
                x = z(x,y);
            }
        }
        print("x:"+x+"y:"+y);
        z = ( x + y ) * 5 / 2-- -y;
    }
}

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

  - output

```
line  1: class test5 {
line  2: int add ( int a1 , int a2 ) {
line  3: retrun ( a1 + a2 ) ;
line  4: }
line  5: void main ( ) {
line  6: int x , y , z ;
line  7: for ( int i = 0 ; i < 2 ; i ++ ) {
line  8: if ( i == 0 ) {
line  9: //--------------ELSE WITHOUT IF
******Else Without If at line 10, char  4******
line 10: else
line 11: i = 1 ;
line 12: }
line 13: for ( x = 0 ; x < 5 ; x ++ ) {
line 14: y ++ ;
line 15: //--------------FUNCTION CALL
line 16: x = add ( x , y ) ;
line 17: x = z ( x , y ) ;
line 18: }
line 19: }
line 20: print ( "x:" + x + "y:" + y ) ;
line 21: z = ( x + y ) * 5 / 2 -- - y ;
line 22: }
line 23: }
line 24:
line 25: /* this is a comment // line// with some /* /*and
line 26: // delimiters */
```

test6.java

- input

```
class test6{
    void sum(){
//------NEVER USED
        int sumxyz = x + y + z ;
    }
    void main() {
//------ARRAY
        int [] i= new int [1];
        for(i[0] = 0; i[0]<5; i[0]++)
            i[0]++;

//------NEW CLASS
        Point lowerLeft = new Point() ;

//------ERROR CONDITION
        while(**/a++){
            print("error!!");
        }
//------CLASS DECLARE
        class Point {
            int x, y, z;
        }
    }

}
```

- output

```
line  1: class test6 {
line  2: void sum ( ) {
line  3: //------NEVER USED
line  4: int sumxyz = x + y + z ;
line  5: }
line  6: void main ( ) {
line  7: //------ARRAY
line  8: int [ ] i = new int [ 1 ] ;
line  9: for ( i [ 0 ] = 0 ; i [ 0 ] < 5 ; i [ 0 ] ++ )
line 10: i [ 0 ] ++ ;
line 11:
line 12: //------NEW CLASS
line 13: Point lowerLeft = new Point ( ) ;
line 14:
line 15: //------ERROR CONDITION
******Invalid Boolean Expression at line 16, char  9******
line 16: while ( * * / a ++ ) {
line 17: print ( "error!!" ) ;
line 18: }
line 19: //------CLASS DECLARE
line 20: class Point {
line 21: int x , y , z ;
line 22: }
line 23: }
line 24:
line 25: }
```