

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。

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 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

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]Parser of 你/妳的學號

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）：

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

例如：

```
✧ int[] a = new int[10];
```

常數的宣告格式：

```
final type identifier_list;
```

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

例如：

```
final float pi = 3.14;
```

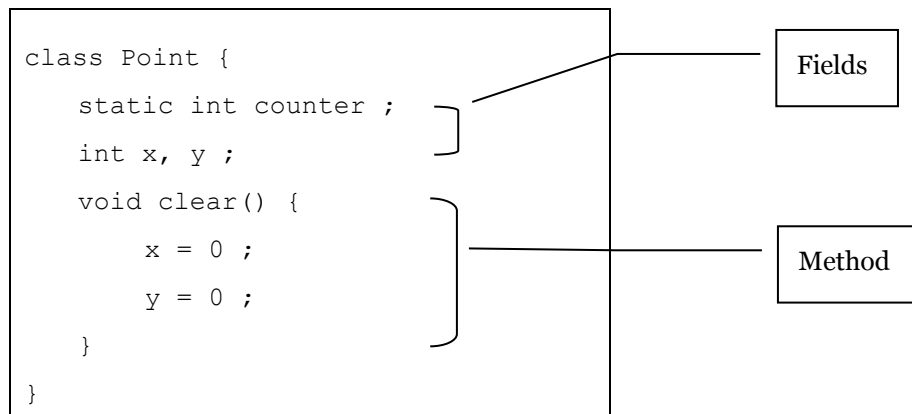
注意事項

- ✧ [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 compound 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 statement** 之後，這些 **variables** 和 **constants** 就失效了。

一個 **compound statement** 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：

- (c) 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。
- (d) 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 Example Simple Java Program

Input 1

```
/*
 * Compute sum = 1 + 2 + ... + n
 */
class sigma {
    final int n = 10;
    int sum, index;

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

Output 1

```
Line 1: class sigma {  
Line 2:   final int n = 10;  
Line 3:   int sum, index;  
Line 4:  
Line 5:   main()  
Line 6:   {  
Line 7:     index = 0;  
Line 8:     sum = 0;  
Line 9:     while (index <= n)  
Line 10:    {  
Line 11:      sum = sum + index;  
Line 12:      index = index + 1;  
Line 13:    }  
Line 14:    print(sum);  
Line 15:  }  
Line 16: }
```

Input 2

```
class Point {  
    static int counter ;  
    int x, y ;  
    int x ;  
    void clear() {  
        x = 0 ;  
        y = 0 ;  
    }  
}
```

Output 2

```
Line 1 : class Point {  
Line 2 :     static int counter ;  
Line 3 :     int x, y ;  
Line 4 :     int x ;  
> 'x' is a duplicate identifier.  
Line 5 :     void clear() {  
Line 6 :         x = 0 ;  
Line 7 :         y = 0 ;  
Line 8 :     }  
Line 9 : }
```

Input 3

```
class Point {  
    int x y ;  
}  
class Test {  
    Point p = new Point()  
}
```

Output 3

```
Line 1 : class Point {  
Line 2 :     int x  
Line 2, 1st char : 11, a syntax error at "y".  
Line 3 : }  
Line 4 : class Test {  
Line 5 :     Point p = new Point() ;  
Line 6 : }
```