Simple Java — Parser

This assignment is to write a Simple Java Parser. You must write the grammar that conforms to the syntactic definitions in the following sections. Once you have defined these grammars, you can substitute them into Yacc to generate a C file named "y.tab.c" (this C file contains yyparse()). yyparse() will call yylex() to obtain a token, so you need to modify your first assignment – Scanner – to allow yyparse() to obtain the token.

Full Java grammar structure:
http://db.cse.nsysu.edu.tw/%7Echangyi/slides/compiler/lab/Java.doc

You must consider the following issues:

(a) Your parser must be able to generate error messages when it encounters an error. Those error messages include the following cases:

1. The line number where the error occurred.
2. The position of the character, Moreover an explanation of the cause of the error.

(b) When the parser encounters an error, it should process the input as completely as possible. That is, the parser should perform recovery, when it encounters an error.

1. What to Submit

You must submit the following files:

- ☐ **The revised Scanner should be named – your student ID.l**
- ☐ **Your Parser should be named – your student ID.y**

(It should include comments to explain how to process statements.)

- ☐ **Your test files**
- ☐ **All .c and .h files**
- ☐ **Makefile**
- ☐ **A Readme.pdf containing:**
  - ☐ **Lex, Yacc version**
  - ☐ **Operating platform**
  - ☐ **Execution method**
  - ☐ **How you handle the issues in this specification**
  - ☐ **Problems you encountered while writing this assignment**
  - ☐ **The results of executing all test files, saved as an image.**

Please compress all the above files into one file, named – your student ID_hw2

2 Syntactic Definitions

The following syntactic definitions are just snippets. You must come up with your own grammar that conforms to these syntactic definitions to complete your assignment.

2.1 Data Types and Declarations

The basic data types are boolean, char, int, float, and String. A variable is declared in the following format:

[static] type identifier_list;

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

For example:

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

The array declaration is in the following format. (In this assignment, we only consider one-dimensional arrays and do not consider the actions of the Assignment):

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

For example:

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

The declaration format of a constant (final):

```
final type identifier_list;
```

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

For example:

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

Things to note

- [x] means that x will appear 0 or 1 times.
- {x} represents that x will appear 0 or more times.
- x|y means either x or y.

Classes and Objects

Every object has a type, which is the object's class. Each class type has two members:

- Fields are data variables associated with a class and its objects.
- Methods contain the executable code of a class.

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

Fields

Method

There can be multiple classes in the same file.

Creating objects

Use the `new` keyword to create objects.

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

Fields

There are two types of fields:

- class fields (static fields), such as static int counter;
- instance fields (non-static fields), such as int x, y;

2.2 Methods

A method declaration should have the following format:

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

method_modifier →
public | protected | private

Even if arguments are not declared, parentheses are still required. Within a method, no other methods can be declared. The format of a formal argument is as follows:

type identifier

If there are multiple formal arguments, separate them with commas.

Methods may return a value or not. If a method does not return a value, its type will be void. For example, the following examples are valid method declarations:

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

Each method has a unique name.

2.3 Statements

There are six different types of 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}
}
```

Variables and constants declared within a compound statement are domain-dependent. They become invalid, once the statement is removed.

An example of a compound statement:

```
{
    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 →

++ |

--

For example:

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

method invocation

A method call has the following format:

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

The format of a loop statement is as follows:

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

or

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

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

ForUpdateOpt →
identifier ++ |
identifier --

For example:

```
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

The format of a return statement is as follows:

```
return expression ;
```

2.3.6Method Invocation

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

3 Semantic Definition

Your parser must be able to perform a simple Semantic Definition check.

For example, two identical variables cannot be declared within the same scope.

For example:

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

Declaring two variables of 'a' within this scope is illegal, and your parser must be able to detect it.

4. Error and Recovery

You must consider the following cases:

(a) Your parser must be able to generate error messages when it encounters an error. Those error messages include the following cases:

1. The line number where the error occurred.
2. The position of the character, Moreover an explanation of the cause of the error.

(b) When the parser encounters an error, it should process the input as completely as possible; that is, the parser must perform recovery, when encountering an error.

5. Scoring Methods

(a) 6 publicly available test cases. I will select 3 that are exactly the same, and only those that are all correct will receive points (20% each).

(b) 2 hidden test cases, randomly arranged from the publicly available test cases (10% each).

(c) Annotations: Explanation of how to process statements (5%)

(d) Readme.pdf (5%)

(e) Verbal Q&A (5% * 2)

(f) Bonus (including statements that are 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

(Note: Line 10 should be missing Semicolon, but it's acceptable for the error to appear in the int of Line 12, meaning at least one error occurred.)

- 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

```java
/*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: }
```