PA2 -- Syntax Analysis

---

1. Description

For this assignment, you will implement the second phase of the compiler -- syntax analyzer (parser). The parser you build will take the token stream from the lexer as the input and check whether there is any syntax error or not.

You will build the parser using **bison**, which is a parser generator that works seamlessly with flex. You just need to write the Context Free Grammar(CFG) for the programming language, and bison will generate a parser for the language implemented in C. Please go through the document ***Introducing_bison*** before you start working on the assignment.

2. Starter Code

Makefile	used to build/compile the parser
utility.h / .cc	interface / implementation of various utility functions
errors.h / .cc	class that defines error reporting methods
location.h	class that defines type for token location
lexer.h	type definitions and prototype declarations for lexer
lexer.l	flex input file (use your own lexer.l from PA1, we will not provide lexer.l in PA2's starter code)
parser.h	declare parser functions and types
parser.y	**skeleton of a bison parser**
main.cc	the entry point where the whole program will be started
samples	directory contains test input files

When you are going through the starter code, please read the comment as well. The comments are really helpful in explaining the structure of the code.

In this assignment your implementation will all go into parser.y. You need to write the correct CFG in the bison input file in order to generate the parser for the language.

Note: In PA1, the tokens are declared in lexer.h, which is a temporary work-around because we do not have **parser.y** file. For this assignment, if you take a look at the starter files, all the tokens are declared in **parser.y** using the **%token** keyword.(You don't need to modify the token declaration.)

We will provide the new lexer.h, but you still need to **make the following changes** in your **lexer.l**: (Note: you only need to copy your **lexer.l** to PA2 directory, you should **not** copy **lexer.h**, we will provide the new **lexer.h**)

(1) Copy your **lexer.l** from PA1 folder to PA2 folder, and do the following changes to lexer.l

(2) Add the following 2 extra inclusions **after other inclusions** at the top of the file:

```
#include <iostream>
#include "lexer.h"
#include "location.h"
#include "errors.h"
#include <vector>
#include "utility.h"
#include "parser.h"
```

(3) Delete the `PrintOneToken()` and `main()` methods at the bottom

(4) Delete the following 2 declarations

```
   struct yyltype yylloc; // manually dclared for pp1, later
Yacc provides
   YYSTYPE yylval;  // manually declared for pp1, later Yacc
provides
```

(5) Add the following global variable declaration

You can just put it after the line that defines TAB_SIZE, so the result would be the following:

```
#define TAB_SIZE 8
vector<const char*> savedLines;
```

(6) Add the following method at the bottom of the file

```
/* Function: GetLineNumbered()
 * ---------------------------
 * Returns string with contents of line numbered n or NULL if the
 * contents of that line are not available.  Our scanner copies
 * each line scanned and appends each to a list so we can later
 * retrieve them to report the context for errors.
 */
const char *GetLineNumbered(int num) {
   if (num <= 0 || num > savedLines.size()) return NULL;
   return savedLines[num-1];
}
```

(7) Add a new start condition declaration before **%%,** and add the rules in that start condition after **%%,** and add ***yy_push_state(COPY)*** to **void InitLexer()**:

```
%x COPY
%option stack
%%
<COPY>.*
{savedLines.push_back(strdup(yytext));
                        curColNum = 1;
                       yy_pop_state();
```

```
                        yyless(0); }

<COPY><<EOF>>           { yy_pop_state(); }

---------------------------------------------------------------------------

void InitLexer()
{
    yy_flex_debug = false;
    curLineNum = 1;
    curColNum = 1;
    yy_push_state(COPY);
}
```

Please make sure you did the above changes correctly, otherwise you will get errors when you try to **make.**

3. Example

If you take a look at parser.y, you can see that a small portion of the grammar has been implemented already. If you build and run the parser, it should be able to successfully parse some simple variable declaration statements like *int var1* or *bool var2*. You can give it a try by doing the following:

```
%  make
% ./parser < samples/test1.java
```

(You will see some the *type clash on default action:* warning during the make, you can just ignore it for this assignment)

Then you should see it parsed the source code successfully. But if you try to parse another given test file, *test2.java*, you will get the syntax error:

```
%
%*** Error line 1.
%int var1 = a + b;
%         ^
%*** syntax error
%
```

Even though *test2.java* has no syntax error, it still raised the error because you have not written the corresponding CFG to parse it.

4. Testing your work

We will provide some simple test cases and the corresponding output which you can find in the **samples** folder. You should definitely come up with your own test cases to test your parser to make sure that:

(1) Your parser will throw syntax error for all the syntax-invalid source code.
(2) Your parser will not throw any error for syntax-valid source code.

5. Grading

You can go to any teaching staff's lab hours and ask for a check-off when you are done with your assignment. You will be asked to run your assignment against the given test cases, and we will check the output of your assignment. You also need to generally describe how did you approach the assignment to the teaching staff.

6. Context Free Grammar

This is the CFG that you should implement in the bison input file for this assignment.

Symbols starts with upper-case "T" are terminals, which correspond to the Token declared in parser.y.

All the other symbols are non-terminals; for this assignment, when you are declaring your non-terminals, you can just declare them all with type *integerConstant*, and ignore those type clash warnings during the make.

Lines with blue color indicates that the two lines should be on the same line, but it is out of space so I just color them with blue.

program:
    declaration_list

declaration_list:
    declaration_list declaration
    declaration

declaration:
    single_declaration
    function_definition

single_declaration:

```
        type_specifier    T_Identifier    T_Semicolon
        type_specifier    T_Identifier    assignment_operator    expression T_Semicolon

type_specifier:
        T_Void
        T_Int
        T_Bool

function_definition:
        function_prototype    compound_statement_with_scope
        function_prototype    T_Semicolon

function_prototype:
        function_prototype_header    T_RightParen

function_prototype_header:
        type_specifier    T_Identifier    T_LeftParen
        type_specifier    T_Identifier    T_LeftParen    parameter_declaration_list

parameter_declaration_list:
        parameter_declaration_list    T_Comma    parameter_declaration
        parameter_declaration

parameter_declaration:
        type_specifier T_Identifier

compound_statement_with_scope:
        T_LeftBrace    statement_list    T_RightBrace
        T_LeftBrace    T_RightBrace



statement_list:
         statement_list    statement
         statement

statement:
        compound_statement_with_scope
        simple_statement

simple_statement:
        expression_statement
        selection_statement
```

iteration_statement
return_statement
decl_statement
break_statement

expression_statement:
T_Semicolon
expression T_Semicolon

selection_statement:
T_If T_LeftParen expression T_RightParen compound_statement_with_scope
T_If T_LeftParen expression T_RightParen compound_statement_with_scope T_Else compound_statement_with_scope

iteration_statement:
while_statement
for_statement

while_statement:
T_While T_LeftParen condition T_RightParen statement

for_statement:
T_For T_LeftParen expression_statement expression_statement expression T_RightParen statement

condition:
expression

return_statement:
T_Return expression_statement

decl_statement:
single_declaration

break_statement:
T_Break T_Semicolon

expression:
assignment_expression
arithmetic_expression
relational_expression
equality_expression

```
        logical_expression
        unary_expression

assignment_expression:
        unary_expression    assignment_operator    expression

assignment_operator:
        T_Equal
        T_MulAssign
        T_DivAssign
        T_AddAssign
        T_SubAssign

arithmetic_expression:
        expression    T_Plus    expression
        expression    T_Dash    expression
        expression    T_Star    expression
        expression    T_Slash    expression

relational_expression:
        expression    T_LeftAngle    expression
        expression    T_RightAngle    expression
        expression    T_LessEqual    expression
        expression    T_GreaterEqual    expression

equality_expression:
        expression    T_EQ    expression
        expression    T_NE    expression

logical_expression:
        expression    T_And    expression
        expression    T_Or    expression
postfix_expression:
        primary_expression
        postfix_expression    T_Inc
        postfix_expression    T_Dec
        func_call_expression

func_call_expression:
        function_call_header_with_parameters    T_RightParen
        function_call_header_no_parameters    T_RightParen

function_call_header_no_parameters:
```

```
        function_identifier    T_LeftParen    T_Void
        function_identifier    T_LeftParen

function_call_header_with_parameters:
        function_identifier    T_LeftParen    arg_list

arg_list:
        assignment_expression
        arg_list    T_Comma    assignment_expression
        primary_expression
        arg_list    T_Comma    primary_expression

function_identifier:
        T_Identifier

primary_expression:
        T_Identifier
        constant
        T_LeftParen expression    T_RightParen

unary_expression:
        postfix_expression
        T_Inc    unary_expression
        T_Dec    unary_expression
        T_Plus    unary_expression
        T_Dash    unary_expression

constant:
        T_IntConstant
        T_BoolConstant
```