Compilers Project1 - Scanner & Parser

Pre-requisite

Since I'm using c++ to compile flex and bison files, **g++** supporting **c++17** is required.

Scanner

Self defined class:

<scanner.hpp>:

1. `SPL_Scanner : public yyFlexLexer` Handle scan process and scanner error report process using `yylex` and `add_error` `error_reported`.
2. `Error` & `Lexical_Error : public Error` & `Syntax_Error : public Error`:
   - The encountered error would be encapsulated into Lexical or Syntax Error and added to the vector `lexical_errors` in `SPL_Driver` .
   - The `Lexical_Error` and `Syntax_Error` have different behavior (Type A and B) in `to_string` function, which is used in the error report process.
3. `Scan_Info`: The scanned lexeme and line number would be encapsulated into `Scan_Info` in flex file and used in bison file as information obtained from scanner.

Error report block

```
{ILLEGAL_HEX_INT} {
    add_error(new Scan_Info(std::string(yytext), yylineno));
    yylval->build<Scan_Info*>(new Scan_Info(std::string("0"),
yylineno));
    return token::INT;
}
```

This block would first call `add_error` to add a lexical error into `lexical_errors`, and build a dummy `Scan_Info`, which would be sent into bison file.

Parser

Self defined class:

<spl_driver.hpp>:

1. `SPL_Driver`:
   - Handle parse process by calling `SPL_Parser` class, which is generated by bison compiler.
   - The error report of parser is also handled there. The `Error` vector: `syntax_errors` would record all the syntax error instead of lexical error.
   - The `Program_Node` root is defined there. The root would be set using `set_root()` called in bison file.

Error report block

```
  | Specifier %prec ERROR {
    driver.add_syntax_error(";", $1);
  }
```

Since I don't find a proper way to get location in the bison-defined `error` function, I choose to call the driver to do syntax report at some production.

Abstract Syntax Tree (AST)

Self defined class & functions:

<ast.hpp>:

1. `AST_Node`: The base class of all the AST node class.
   - Contains a `line_no` member to record the earliest occurred line number of node.
   - Contains a `propagate_line_no` function to retrieve the line number of children recursively.
2. Other sub-class of `AST_Node`: They represent different kinds of nodes.
3. `void print_ast(AST_Node *node, int indent_level = 0);` Using back-tracking algorithm to print the AST using pre-defined format.

Features

Required

1. Error report: Done by `errors` which consists of all the errors reported by scanner and parser. In the main function, if the `errors` is not empty, it will output error reports instead of AST.
2. Valid program Done by constructing a `AST` class in bison file. The main function would call the `print_ast()` to recursively print the AST.

Optional

1. Single-line and Multi-line comments: Done in flex files by recognizing these comments format:

```
LINE_COMMENT            \/\/[^\n\r]*(\n|\r\n|\r)?
MULTI_LINE_COMMENT      \/\*[^*]*\*+(?:[^/*][^*]*\*+)*\/
```

2. Hexadecimal representation of integers: Done in flex files:

```
HEX_INT                 0[xX]([1-9A-Fa-f][0-9A-Fa-f]*|0)
ILLEGAL_HEX_INT         0[xX]([0-9A-Za-z]*)
```

3. Hex-form characters: Done in flex files:

```
HEX_CHAR                '\\x[0-9A-Fa-f][0-9A-Fa-f]'
ILLEGAL_HEX_CHAR        '\\x[0-9A-Za-z]*'
```

4. Nested multi-line comments: Not implemented