CS323 Principle of Compiler - Project Phase 1

Simple Parser Implement

> 陈俊峰 *12011423*
>
> 张闻城 *12010324*
>
> 唐骞 *11911922*

Note: Since we want to implement pointer, we consider `&` as a valid lexeme, which represents addressing operation. And thus on test case `test_1_r07.spl`, we will get a different output than the reference output. Another self-written test case with the similar error type, `test_1_r07.spl` in `test/test-ex`, is provided. You can test correctness of our parser using that test file instead.

1. Features

1.1 Basic Features

1. Recognize all token defined in **SPL**
2. Construct **SPL** language's parser tree correctly
3. Basic error recognition and error recovery:
   - Error type A: undefined characters or tokens
   - Error type B: missing closing symbols (parenthesis or semicolon)
   - Recognition of illegal hexadecimal integer and hex-form character

1.2 Bonus Features

Note: output file's location of parser will be the same as parser input file's folder.

1. File inclusion: a preprocessor for file inclusion written in `lex` is implemented. Before formally starting the lexical analysis, this preprocessor should be executed to generate temp file with `#include` statement being replaced by corresponding file text. The maximum include depth defaults to be 8 and can be modified by the macro `MAX_INCLUDE_DEPTH` in `preprocess.l`.

```
cd test/test-ex/file-inclusion
make
./pre main.spl # output in main.out
```

2. Single and multi-line comment: comments will all be ignored

```
bin/splc test/test-ex/comment.spl # output in comment.out
```

3. Pointer type recognition (`void *`, `int *`, `char *`), also addressing `&` and de-addressing `*`

```
bin/splc test/test-ex/pointer.spl # output in pointer.out
```

4. String literal recognition

```
bin/splc test/test-ex/string.spl # output in string.out
```

5. Support simple `for` statement.

```
bin/splc test/test-ex/for.spl # output in for.out
```

6. Support `break` and `continue` operation (these two tokens are considered as part of `Exp` production). Now our parser can only recognize these two as valid tokens, further check of their position is required.

```
bin/splc test/test-ex/break-continue.spl # output in break-continue.out
```

7. Several

2. Design and Implementation

During lexical and syntax analysis, we mainly use a class `Node` to store necessary information and data. `Node` is defined as below

```
enum class DataType
{
    INT, FLOAT, CHAR, STRING, ID, ERR
    DTYPE,      // data type
    OTHER,      // keyword, operator and other symbols
    PROD        // non-terminal
};

class Node
{
public:

    string token_name{};
    string data{};
    DataType type{};
    int lineno{0};
    Node *parent = nullptr;
    vector<Node *> children{};
};
```

Once a token is identified by lexer, one `Node` object will be created to store the token value, and then be assigned to `yylval` for parser to use.

2.1 File inclusion

Use a stack, `YY_BUFFER_STATE include_stack[MAX_INCLUDE_DEPTH]` to store the files to process. And `int include_stack_ptr` is used to point to stack top, representing current file. When meet `#include` directive, store the current file pointer into the stack and switch the scanner's buffer input to the included file. When meet `<<EOF>>`, check if there is still file in the stack. If there is, then delete the current buffer using `yy_delete_buffer()`, and delete stack top file then switch to the current stack top using `yy_switch_to_buffer()`, otherwise terminate current program.

2.2 Single and multi-line comment

```
"//" { while((c = yyinput()) != '\n'); unput(c); }
"/*" {
    c = yyinput();
    while(1) {
        char tmp = yyinput();
        if (c == '*' && tmp == '/') break;
        c = tmp;
        if (c == '\n') lines++;
    }
}
```

The implementation to ignore comments is the same as what Pro. Liu has mentioned in lab courses.

2.3 String

```
{string-literal}            {
    if (yytext[yyleng - 2] =='\\' && yytext[yyleng - 1] != '"') {
        yyless(yyleng - 1);
        yymore();
    } else {
        yylval = new Node("STRING", yylloc.first_line, DataType::STRING, yytext);
        return STRING;
    }
}
```

The implementation to recognize string is the same as what Pro. Liu has mentioned in lab courses.

2.4 `for` statement

We implement this feature by simply add the production below into `syntax.y`

```
Stmt -> FOR LP Exp SEMI Exp SEMI Exp RP Stmt
```