CS323 Project1 Report

Haotian Liu@SUSTech 11613015

Project contents: Lexical Analyzer & Syntax analyzer for SUSTech programming language (SPL)

1. Write Regular Expression using flex to recognize tokens
2. Write Context-Free Grammar (CFG) using bison to specify the SPL's grammar

Data Structure: The tree-structured AST with siblings linked together.

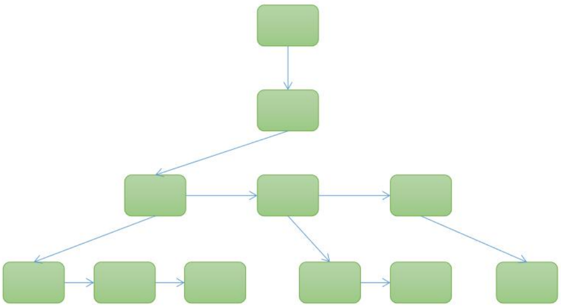
Each node in the tree is defined as (see /ast.c)

```
struct ast {
    int lineno;
    char *name;
    char *value;
    struct ast *sub_ast;
    struct ast *next_sibling;
};
```

The lineno corresponds to the line number, the name to the token name, all of the token's values are stored as the string directly in value. The sub_ast points to the child node lower one level while next_sibling points to the node at the same level.

In this project, all of the tokens and grammar units are defined as ast.

The AST nodes are linked as below:



Function Implementations (see /ast.c)

`struct ast *new_node (char *name, char*value, int lineno);`

When a token is recognized by the regular expression defined in flex file (see /lex.l), the function new_node will be invoked to generate a new ast with the token's name, value and the line number of it.

`struct ast *new_ast (char *name, int num, ...);`

When a grammar in SPL is matched the CFG defined in bison file (see /syntax.y), the function new_ast will be invoked to connect the nodes and generate a new AST. The name represents the name grammar unit, the num is the number of children, and also the variable number of the variable argument list, where variables will be parsed as ast's.

`void preorder (struct ast *ast, int level);`

Traverse the AST in pre-order way with function preorder, print the expected result.

Optional Features

✓ Support single-line comments.

Realization: when recognizes the “//” pattern, using input() function to read characters from the input buffer without handling. That is, discard all characters behind “//” in the line by the following code:

```
"//" { char c; while((c=input()) != '\n'); }
```

✓ Support multi-line comments and detect nested multi-line comments.

Realization: when recognizes the “/*” pattern, read and discard the characters until meet “*/”. The code is as below:

```
"/*" {
    char c;
    while (1) {
        if ((c=input()) == '*') {
            if ((c=input()) == '/') {
                break;
            }
            else {unput(c);}
        }
    }
    if ((c=input()) == '/') {
        if ((c=input()) == '*') {
            Raise an syntax error;
        }
        else {unput(c);}
    }
}
"*/" {
    Raise an syntax error;
}
```

I directly raise the syntax error in lex.l file for such implementation is simple and the additional syntax unit can be omitted.

If “/*” or “*/” are contained in pair of double-quotes, they will be recognized as string, whose matching length is longer.

✓ Support hexadecimal representation and detect illegal form of hex-int.

Define the regular expression below

```
HEX_INT [-+]?0[xX]([0-9a-f]{0,8})
HEX_INT_WRONG [-+]?0[xX][0-9a-zA-Z]*
```

And processing HEX_INT before HEX_INT_WRONG, when a correct hexadecimal is recognized, although both of the pattern can match it, the former one will be returned. When a wrong hex-int or hex-int more than 32bits detected, the latter one will be returned.

✓ Support lex-form characters, and detect illegal form like ‘\xt0’

```
CHAR ('[a-zA-Z0-9]')|('\\x[0-9a-fA-F]{2}')
CHAR_WRONG ('\\x[0-9a-zA-Z]*')
```

Similar as hexadecimal representation detection written above.