Implementing a Lexical-Syntax Analyzer

By Duowen Liu,
student ID:5130309723

Introduction

In this project, I implement a simple compiler front-end, including a lexical analyzer and a syntax analyzer, for Small-C , which is a C-like language containing a subset of the C programming language.

> **Tools and environment:**
>
> - Lex,a lexical analyzer which help to divide code into tokens
> - Yacc,a syntax analyzer help to generate parse tree
> - compiler g++ under ubuntu

Implement

step 1: Lexical analyzer

In this step, we will write a lexical analyser. The lexical analyser reads
in the Small-C source code, and recognize tokens according to regular definitions

In **lex.l**,we define several tokens:

```
SEMI        [;]
COMMA       [,]
DOT         [.]
UNDERSCORE  [_]
```

LP [()]

After that,we use **flex** to compile **lex.l** for future use

step 2: Syntax analyzer

In this step,we use **Yacc** to build our analyzer.Yacc is an LALR parser generator, which stands for yet another compiler-compiler . In this project, we can use yacc/bison (bison is another version of yacc) to generate a parser.Here are syntax example:

```
EXTDEFS:
    EXTDEF EXTDEFS  {
        cout<<"deal with extdefs -> extdef extdefs\n";
        addNode("EXTDEFS");
        arr[counter-1].son[1] = stk.top();stk.pop();
        arr[counter-1].son[0] = stk.top();stk.pop();
        stk.push(&arr[counter-1]);
    }

    |/*EMPTY*/{
        cout<<"deal with extdefs -> empty\n";
        addNode("EXTDEFS");addNode("empty");arr[counter-2].son[0]
=&arr[counter-1];
    stk.push(&arr[counter-2]);
    }

    ;
```

step 3: Tree building

```
-PROGRAM
-----EXTDEFS
---------EXTDEF
-------------SPEC
-----------------TYPE
-------------FUNC
-----------------main()
-----------------LP
-----------------PARAS
--------------------empty
-----------------RP
-------------STMTBLOCK
```

Error handling

When your code exists an error,the parser will deliver an error message:

YACC: syntax error
YACC: line 2
YACC: at }

To implement that we need to write yyerror function like this:

```
int yyerror (const char *msg) {
    printFlag = false;
    fprintf (stderr, "YACC: %s\n", msg);
    fprintf(stderr, "YACC: line %d\n", yylineno);
    fprintf(stderr, "YACC: at %s\n", yytext);
    return -1;
}
```

Test cases

To test my project,please run the **test.sh** script and run **./a.out**:

```
xxx@ubuntu: ./test.sh
xxx@ubuntu: ./a.out
```