CS323 Project1 - Report


```
Yifei Li 11811905
Zhuochen Xiong 11811806
```


Introduction

In this project, we use lexical & syntax analyzer are implemented to parser SPL source code. Our parser can accept SPL file and output the syntax tree of a valid SPL program or figure out the error in the code. There are two types of mistakes, lexical error and syntax error.

Design & Implementation

File structure

```
.
├── ast.c
├── ast.h
├── bin
│   └── splc
├── report
│   └── 11811806-project1.pdf
├── lex.l
├── lex.yy.c
├── Makefile
├── README.md
├── syntax.y
├── test
│   ├── test_11811806_1.out
│   ├── test_11811806_1.spl
│   ├── test_11811806_2.out
│   ├── test_11811806_2.spl
│   ├── test_11811806_3.out
│   ├── test_11811806_3.spl
│   ├── test_11811806_4.out
│   ├── test_11811806_4.spl
│   ├── test_11811806_5.out
│   └── test_11811806_5.spl
└── test-ex
    ├── test1.out
    └── test1.spl
```

Syntax Tree structure

```
typedef struct ast_node{
    char* name;
    token_type type;
    char* value;
    int line_num;
    int children_num;
    struct ast_node **children;
}ast_node;
```

Function

- pass all test case provided by SA & Teacher
- pass all test case provided by ourself
- Some basic feature:
    - print standard syntax tree
    - transfer hexadecimal number to decimal representation.
    - point out mistakes, lexical error and syntax error.
    - etc.

Optional Feature（Bonus)

- Support For statements

```
// test case
int test1() {
  int i = 0;
  for(i = 0; i < 10; i = i+1) {
    count = count + 1;
  }
  return count;
}
```

```
        Stmt (3)
            FOR
            LP
            ForVarList (3)
              DecList (3)
                Dec (3)
                  VarDec (3)
                    ID: i
                  ASSIGN
                  Exp (3)
                    INT: 0
              SEMI
              Exp (3)
                Exp (3)
                  ID: i
                LT
                Exp (3)
                  INT: 10
              SEMI
              Args (3)
                Exp (3)
                  Exp (3)
```

```
                ID: i
              ASSIGN
              Exp (3)
                Exp (3)
                  ID: i
                PLUS
                Exp (3)
                  INT: 1
          RP
          Stmt (3)
            CompSt (3)
              LC
              StmtList (4)
                Stmt (4)
                  Exp (4)
                    Exp (4)
                      ID: count
                    ASSIGN
                    Exp (4)
                      Exp (4)
                        ID: count
                      PLUS
                      Exp (4)
                        INT: 1
                  SEMI
              RC
```

syntax rules

```
Stmt: FOR LP ForVarList RP Stmt
ForVarList: DecList SEMI Exp SEMI Args
```