实验一 词法分析和语法分析 实验报告

匡亚明学院 陈劭源 161240004

文件夹结构

注意：Makefile在根目录下（而不是在Code文件夹内）

```
.
├── Code                    // 源代码文件
│   ├── error.c
│   ├── lexical.l           // flex词法文件
│   ├── main.c
│   └── syntax.y            // bison语法文件
├── Include                 // 头文件
│   ├── cmm.h
│   ├── cst.h
│   └── error.h
├── Makefile                // Makefile文件
├── parser                  // 语法分析器可执行文件
├── README.txt              // README文件
├── report.md               // 本实验报告的源代码
├── report.pdf              // 本实验报告
├── Test
│   ├── sample              // 提供的测试用例
│   │   └── ...
│   └── secret              // 自行构造的测试用例
│       └── ...
└── testrun.sh              // 测试用例运行脚本
```

编译和运行方法

编译环境

- OS: Ubuntu 18.04.2 LTS
- gcc: gcc 7.3.0
- flex: flex 2.6.4
- bison: GNU Bison 3.0.4
- make: GNU Make 4.1
- shell: GNU bash 4.4.19

编译方法

切换到根目录，输入

```
make
```

即可从源代码生成可执行文件parser（位于根目录）。

运行方法

输入

```
./parser
```

或

```
make run
```

即可运行语法分析器。语法分析器默认从标准输入读入c--源代码，可以通过参数指定从文件读入：

```
./parser source_file
```

运行

```
make clean
```

可以清除所有中间文件和目标文件。

为便于测试，根目录包含了一个测试脚本，可按如下方式运行：

```
./testrun.sh Test/sample/*
```

脚本会将源代码和分析器的输出用`less`打印在屏幕上。

完成的功能点

1. 识别词法、语法错误
2. 输出没有词法、语法错误代码的语法树
3. （选做）识别八进制、十六进制整数
4. （选做）识别指数形式的浮点数
5. （选做）识别两种注释风格

实现方法

词法分析

词法分析部分使用flex工具实现。

为避免形如

```
int x = 090;
```

被分为

```
{int} {x} {=} {0}{90}{;}
```

导致语法错误，这里参考了C语言的处理方法，即将形如

```
\.?[0-9]([.0-9a-zA-Z_]|(e+)|(e-)|(E+)|(E-))*
```

的字符串序列识别为一个token，随后再判断是整数还是浮点数，以及是何种表示方式。这样可以使上述错误在词法分析阶段即可被检查出来。

两种注释风格是在flex中定义注释状态实现的。一旦检测到`//`或`/*`，即将flex置为对应的注释状态，直到遇到`*/`或换行时，退出注释状态。

语法分析

语法分析部分使用GNU Bison工具实现。

错误恢复部分使用较为保守的错误恢复策略，只定义了以下两条规则：

```
ExtDef -> error
Stmt -> error ';'
```

数据结构表示

由于本实验中语法树儿子的数量在创建时已经确定，这里没有采用左儿子、右兄弟的实现方法，而是直接采用数组存储所有的儿子节点：

```
typedef struct cst_node {
    char *buf;
    int nr_child;
    cmm_loc_t loc;
    struct cst_node **child;
} cst_node_t;
```

同时，定义了节点的构造函数、析构函数和递归打印节点信息的函数：

```
static inline cst_node_t
*cst_node_ctor(cmm_loc_t loc, int nr_child, const char *fmt, ...);

static inline void cst_node_print(cst_node_t *cst, int indent);

static inline void cst_node_dtor(cst_node_t *cst);
```

实验总结

本次实验使用flex和GNU Bison工具完成词法分析和语法分析部分，生成的分析器能够识别出c--源代码中的词法错误和语法错误，并能打印无词法、语法错误的c--源代码的语法树。