1.程序功能

完成了 C--语言的词法分析和语法分析，选做完成了浮点数的识别。

使用 flex 进行词法分析，bison 进行语法分析；其中 node.h 是对语法树的数据结构设计，通过将相关函数定义为 static inline 函数，减少调用开销，因为 lexical.l 和 syntax.y 中有大量构建节点函数的重复调用，通过内联省去函数调用额外开销。

语法树节点的数据结构如下：每个节点包括行号，名称，值以及最左孩子节点和兄弟节点。

```
typedef struct node {
    int lineNo;
    NodeType type;
    char* name;
    char* val;

    struct node* child;
    struct node* next;

} Node;
```

然后在 newNode 中会遍历参数列表设置当前节点的 child 和子节点的 next；将产生式传入的参数连接，构建语法树：

```
pNode tempNode = va_arg(vaList, pNode);

curNode->child = tempNode;

for (int i = 1; i < argc; i++) {
    tempNode->next = va_arg(vaList, pNode);
    if (tempNode->next != NULL) {
        tempNode = tempNode->next;
    }
}
```

enum.h 是对 enum 类型变量的声明，本实验中只有 enum nodeType 这一种枚举变量，主要用于定义节点的类型（整数，浮点，id，类型声明，其他 Token 和非终结符），声明类型的主要作用是为了在打印节点时候分别处理。

```
#ifndef ENUM_H
#define ENUM_H

// define node type
typedef enum nodeType {
    TOKEN_INT,
    TOKEN_FLOAT,
    TOKEN_ID,
    TOKEN_TYPE,
    TOKEN_OTHER,
    NOT_A_TOKEN

} NodeType;
#endif
```

lexical.l 和 syntax.y 的编写主要参照了实验手册的说明，因为主要实现了必做，其他地方没有太个性化的内容，因此这里不详细讲。

2.编译及运行

编写了 makefile 文件，只需在 Code 目录下 make 即可编译编写好的.l 文件与.y 文件为 parser 输出：

```
parser: syntax $(filter-out $(LFO),$(OBJS))
    $(CC) -o parser $(filter-out $(LFO),$(OBJS)) -lfl -ly
```

在测试方面编写了自动测试脚本，只需在根目录下运行 auto-test.sh 即可测试所有样例并保存输出为文件：

```
$ parse.sh
1   mkdir -p out2
2   file_name=$(basename $1)
3   ./Code/parser $1 > ./out2/${file_name}_out.txt 2>&1
```

```
$ auto-test.sh
1   for file in ./test2/*
2       do
3           echo Testing with $file
4           ./parse.sh $file
5       done
```