哈尔滨工业大学　编译系统 2023 春

实验一 词法分析与语法分析

学院 计算学部 | 姓名 王炳轩 | 学号 120L022115 | 指导教师 陈鄞

一、 实验目的

a) 巩固对词法分析与语法分析的基本功能和原理的认识
b) 能够应用自动机的知识进行词法与语法分析
c) 理解并处理词法分析与语法分析中的异常和错误

二、 实验内容

(一) 实验环境

Ubuntu 20.04, GNU GCC，GNU Flex version 2.5.35；GUN Bison version 2.5

(二) 程序功能

1. 基本数据类型和数据结构

本次实验用到的数据结构是一个树——一个结构体数组，维护了结点的名称（语法单元，词法单元的名称）、行号（报错用）、子结点的数量、子结点列表和一个联合体（TYPE：ID，INT，FLOAT，用于存放对应的数据）。

```
struct TREE{
    int line; //行号
    char* name; //文法符号名称
    int n; // 子结点数量
    struct TREE *child[maxNum];
    union{
        char* ID; //标识符名称
        int INT;
        float FLOAT;
    };
};
```

```
void yyerror(char *msg){
    hasFault = 1;
    fprintf(stderr, "Error type B at line %d, coloum %d: %s : %s.\n", yylineno, yycolumn, yytext, msg);
}
//打印整个树
```

2. 词法分析

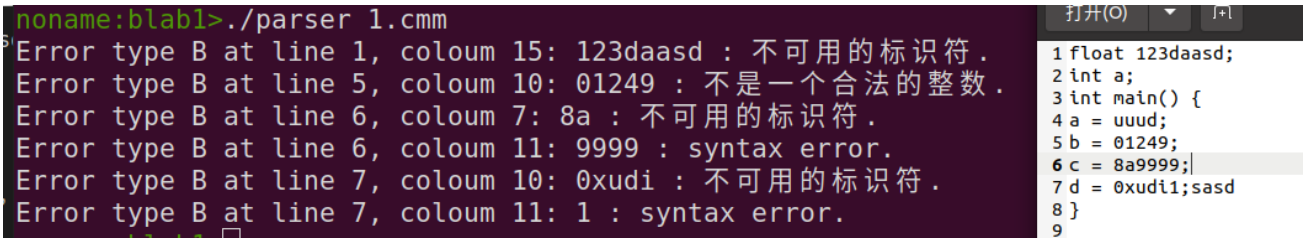
根据 C—的文法定义，我按照指导书上的要求，做到了以下功能：

1、能够正确识别十进制数、八进制数、十六进制数并能进行**进制转换**。
2、可以判断出错误的八进制数、十六进制数、浮点数、指数形式数。
3、能够正确识别浮点数、指数形式的浮点数。
4、能够**识别所有的注释符号，并判断出错误的注释符号。**
5、能够识别所有的标识符词法单元的 ID。
6、能够识别出标识符词法单元的类型（int 和 float）
7、能够对于错误给出列号。
8、能够区分错误的标识符，给出具体的错误类型。

```
*/
ID : ID_CORRECT
    | ID_ERROR
    ;
INT :INT_CORRECT
    |INT_ERROR
    ;
FLOAT: FLOAT_CORRECT
    |FLOAT_ERROR
    ;
```

本实验做的比较特别的内容在于实现了判断出词法分析中的错误类型——使用对于错误的八进制数、十六进制数、浮点数、ID 再进行定义的方法。首先，先将 ID\FLOAT\INT 修改为非终结符，然后添加以下产生式：

然后添加对于 ID_ERROR、INT_ERROR、FLOAT_ERROR 的正则表达式，修改原有的词法单元 ID\FLOAT\INT 为 ID_CORRECT、FLOAT_CORRECT、INT_CORRECT。



比如，八进制数首位为 0 但是后面的数中不能出现 9，所以对首位为 0 且后面出现 9 的数定义为 INT8_ERROR。同理，对于 INT16_ERROR，我们认定在 0x 后面出现[g-zG-Z]都为错误。对于形如 10.e 和.e 和 10.和.05 都是错误的浮点数。同样对于 ID_ERROR，定义为数字在前字母在后。然后我们对于这些错误，匹配 pattern 和 action，打印出错误信息。

3. 语法分析

在语法分析部分，主要在于语法树的建立和错误恢复。程序主要实现的功能包括：

- 构造语法树并按照先序遍历的方式打印每一个结点信息。
- 语法单元：打印输入文件中的行号。若产生ε，不打印。
- 词法单元：打印词法单元的名称，不打印该词法单元的行号。
- ID：打印词素；TYPE：打印 int 或 float；NUMBER：打印该十进制数。
- 若程序错误，打印行号，并判断错误类型。并能**进行错误恢复**，让程序继续检测。

首先是语法树的构建，每一个语法单元或者词法单元都是一个树的结点。建立的结点的过程就是将 pattern 中匹配到的语法单元和词法单元作为结点内容创建，然后加入到数组中。

```
| Exp DOT ID                  {treeNode childList[3] = {$1, $2, $3}; $$= newNode("Exp", 3, yylineno, childList);nodeL
| ID                        {treeNode childList[1] = {$1}; $$= newNode("Exp", 1, yylineno, childList);nodeList[nodeNum]
| INT                       {treeNode childList[1] = {$1}; $$= newNode("Exp", 1, yylineno, childList);nodeList[nodeNum]
| FLOAT                       {treeNode childList[1] = {$1}; $$= newNode("Exp", 1, yylineno, childList);nodeList[node
```

特别的，我们需要判断是否为终结符，若不是则要将子结点数组传入这个结点，若是终结符又要判断是否为 ID、TYPE 或是 INT、FLOAT，以便在打印树的时候对其值打印。

在打印结点的过程中，首先判断是否出错，如果出错则不打印，如果没出错则使用深度优先搜索对整个语法树进行先序遍历。对整个树进行打印的过程中，还将记录所在层数，每一层都将额外在前面打印两个空格，然后再打印结点内容。

(三) 编译过程

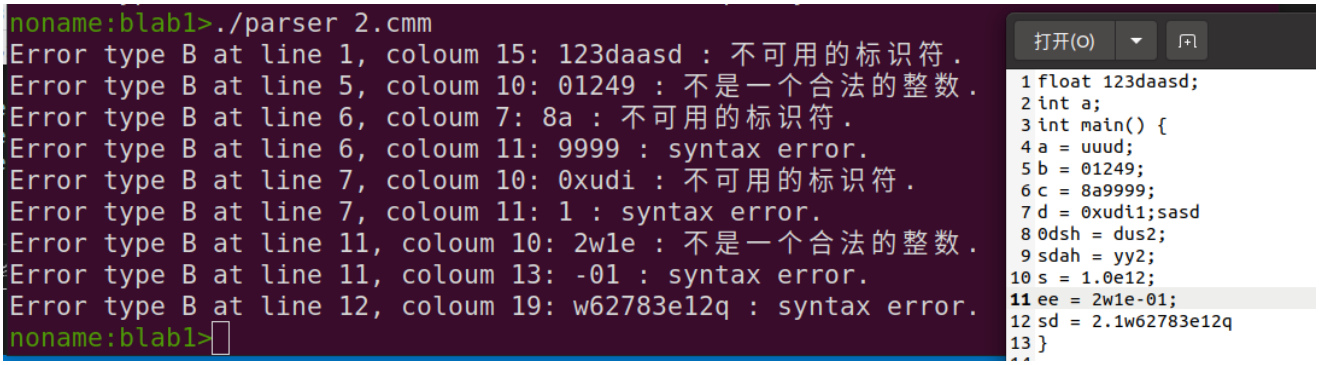
在编译语法分析源代码的时候，利用 flex、bison、gcc 进行编译。

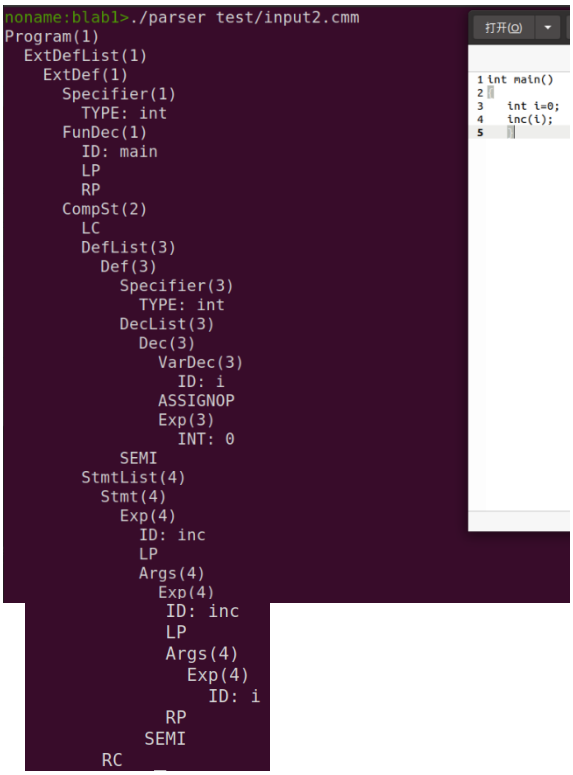
项目使用 Visual Studio Code IDE 进行设计，因此设计了 tasks，可以直接使用 Ctrl+Shift+B 进行编译。

具体的 tasks.json 配置如下：

（四） 实验结果

如图所示，在第一行中使用了数字开头的标识符，编译器报错并提示，标识符不能以数字开头。在第 5 行中，因为数字以 0 开头，因此是一个八进制数字，而其中却包含 9，显然不是一个合法的整数。其他的错误以此类推。





左图展示了一个完整的无错误的 C—代码 parser 后得到的语法树。

三、实验总结

1. 学会了 Flex，Bison 等工具进行词法和语法的分析。会编写 Flex 和 Bison 的源代码，并完成指定的语法分析。
2. 更深入地掌握了编译器的工作原理，对程序的错误恢复和错误判断有进一步的了解。
3. 更透彻地理解了八进制、十六进制数、浮点数的识别，和 C 语言为何规定标识符不能以数字开头。