编译原理实验报告

实验一：词法分析与语法分析

学院：	计算学部	指导老师：	单丽莉
学号：	2021112845	姓名：	张智雄

一、实验目的

1. 巩固对词法分析与语法分析的基本功能和原理的认识
2. 能够应用自动机的知识进行词法与语法分析
3. 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

编写一个程序对使用 C--语言书写的源代码进行词法和语法分析并打印分析结果。程序要能够查出 C--源代码中可能包含的下述几类错误：

a) 词法错误(错误类型 A)：出现 C--词法中未定义或不符合 C--词法单元定义的字符；

b) 语法错误(错误类型 B)。

2.1 实验环境

Ubuntu 22.04；GCC version 11.4.0；GNU Flex version 2.6.4；GUN Bison version 3.8.2

2.2 实验过程

2.2.1 语法分析树的定义

首先在 node.h 中定义了语法分析树结点的结构体如下：

```
typedef struct node {
    int lineNo;              // 结点的行号
    NodeType type;           // 结点类型
    char* name;              // 结点名称，存储语法规则或语法单元的字符串
    char* val;               // 结点单元，存储词法分析阶段识别出的文本信息
    struct node* child;      // 非终端节点第一个子节点
    struct node* next;       // 非终端节点下一兄弟节点
} Node;
typedef Node* pNode;
```

而后通过 newNode，newTokenNode 函数创建语法树的新节点(newTokenNode 用于创建叶子节点，没有子节点，只有名称和值)。printTreeInfo 函数用于打印树形结构中节点的信息，如果当前节点为空，则函数直接返回，不进行任何操作；否则先打印当前节点的名称，并根据节点类型打印不同的信息，而后递归调用对当前节点的子节点和兄弟节点进行打印。

打印时根据层次结构缩进打印树形结构中节点的信息，每个节点都会打印其名称以及根据节点类型不同而不同的附加信息，同时保持树形结构的层次感。同时这里通过 strtol()函数将可能的八进制/十六进制转化为十进制 int 类型整数。

2.2.2 词法分析

词法分析程序的主要任务是将输入文件中的字符流组织成为词法单元流，在某些字符不

符合程序设计语言词法规范时它也要有能力报告相应的错误，并为后续的语法分析提供输入。这一过程主要通过编写正则表达式和有限状态自动机完成。

根据附录 A 中 C--文法编写正则表达式如下，能够识别 1) 十进制数、八进制数、十六进制，2)包含指数形式的浮点数，3)各种标点符号及关键字，4)注释及空白符，5)错误八进制数、十六进制数、浮点数、ID 等词法单元。

```
%option yylineno
digit [0-9]
digits {digit}*
/*Decimal Octal Hexadecimal -> Integer*/
DEC ([1-9]{digit}{0,9}|0)
OCT (0[1-7]([0-7]){0,10}|0)
HEX (0x|0X)([1-9a-fA-F]([0-9a-fA-F]{0,3})|0)
INT {DEC}|{OCT}|{HEX}
/* Float */
FLOAT ((([1-9]{digits}|0)"."{digits})|((({digits}"."{digit}+)|({digit}+"."{digits}))[Ee][+-]?{digit}+)
/* Punctuation */
SEMI ;
COMMA ,
ASSIGNOP =
RELOP >|<|>=|<=|==|!=
PLUS \+
MINUS -
STAR \*
DIV \/
AND &&
OR \|\|
DOT \.
NOT !
TYPE int|float
LP \(
RP \)
LB \[
RB \]
LC \{
RC \}
BLANK " "|\n|\r|\t
/* Keywords */
STRUCT struct
RETURN return
IF if
ELSE else
WHILE while
/* Comment */
LINE_COMMENT ("//"[.]*)
BLOCK_COMMENT "/\*"[.]*"\*/"
COMMENT {LINE_COMMENT}|{BLOCK_COMMENT}
/* Error */
INT8_ERROR 0[0-7]*[8-9]+[0-7]*
INT16_ERROR 0[Xx][a-fA-F0-9]*[g-zG-Z]+[a-fA-F0-9]*
FLOAT_ERROR [0]+{0}[1-9][0-9]*\.[0-9]+|\.[0-9]+|[0-9]+\.|[0-9]+\.[0-9]+[Ee][+-]?[0-9]*|([+-]?([0-9]*\.[0-9]+|[0-9]+\.[0-9]*)[Ee][+-]?[0-9]+\.[0-9]+)|[0-9]+\.[0-9]*[Ee]
NUM_ERROR {INT8_ERROR}|{INT16_ERROR}|{FLOAT_ERROR}
ID_ERROR [0-9]+[a-zA-Z_]+
/* ID at last to avoid conflict */
ID [_a-zA-Z][_0-9a-zA-Z]*
%%
```

图 1 正则表达式

这里关于浮点数的正确与错误判断识别考虑了多种情况，能够识别如正确的 3.14, 0.314, 3.1e4 类型和错误的 03.14, .314, 314., 3e14, 0.3e1.4, 3.14e 类型。同时为了避免与前面的表达式产生冲突导致错误识别为 ID，这里将 ID 的识别放到最后。

对于识别的正则表达式，除删除注释、未定义或不符合 C--词法单元定义的字符输出错误类型 A 提示信息等操作外，其余响应函数均为通过 newTokenNode 创建叶子结点并返回属性值给语法分析。而具体处理单行注释 // 和多行注释 /* ... */ 的操作为：当识别到 // 时，会不断使用 input()函数读入字符以丢弃当前行的内容；而当识别到多行注释开头 /* 时，会同样调用 input()以跳过多行注释部分直到 */ 出现。

2.2.3 语法分析

语法分析程序的主要任务是读入词法单元流、判断输入程序是否匹配程序设计语言的语法规范，并在匹配规范的情况下构建起输入程序的静态结构。这里的语法分析主要通过编写上下文无关文法的产生式完成。

这一过程主要根据附录 A 中 C--文法编写产生式并定义语义动作(通过 newNode 函数**自底向上地**建立非叶结点)，同时对可能出现的错误恢复编写尽可能完备的产生式。这里将词法单元使用%union{pNode node}定义其类型，而对于 2.2.2 中定义的非终结符则使用%type<node>定义，并对终结符的结合性进行定义以消除二义性。

本实验并未将 2.2.1 节中识别的错误八进制、十六进制、浮点数、ID 补充到 Exp 产生式中，因此在测试用例 6 和 8 中会同时给出 A 和 B 类型错误（加上此处识别即只报 A 类型错误）。

由于错误恢复时，会有再同步的过程，因此需要较为全面地书写包含 error 的产生式才能够检查出输入文件中存在的各种语法错误，否则会持续反复地移入 error，然后丢弃输入的词法单元，直至程序结尾。

以下是部分包含 error 的产生式，能够识别 1)运算表达式，2)全局/局部变量定义，3) 语句逻辑等当中的一些可能包含 error 的场景。

```
Exp:    …
    |       Exp ASSIGNOP error          { synError = TRUE; }
    |       Exp AND error                { synError = TRUE; }
Stmt:   …
    |       IF LP error RP Stmt ELSE Stmt    { synError = TRUE; }
    |       error LP Exp RP Stmt        { synError = TRUE; }
ExtDef: …
    |       error SEMI                   { synError = TRUE; }
    |       Specifier error              { synError = TRUE; }
```

2.3 实验结果

在 linux 环境下，在文件夹目录下中运行 make 指令编译所有的文件，而后输入 make test 指令便可以测试所有的样例。

测试结果如下（部分），能够正确分析所有测试用例（必做与选做）：

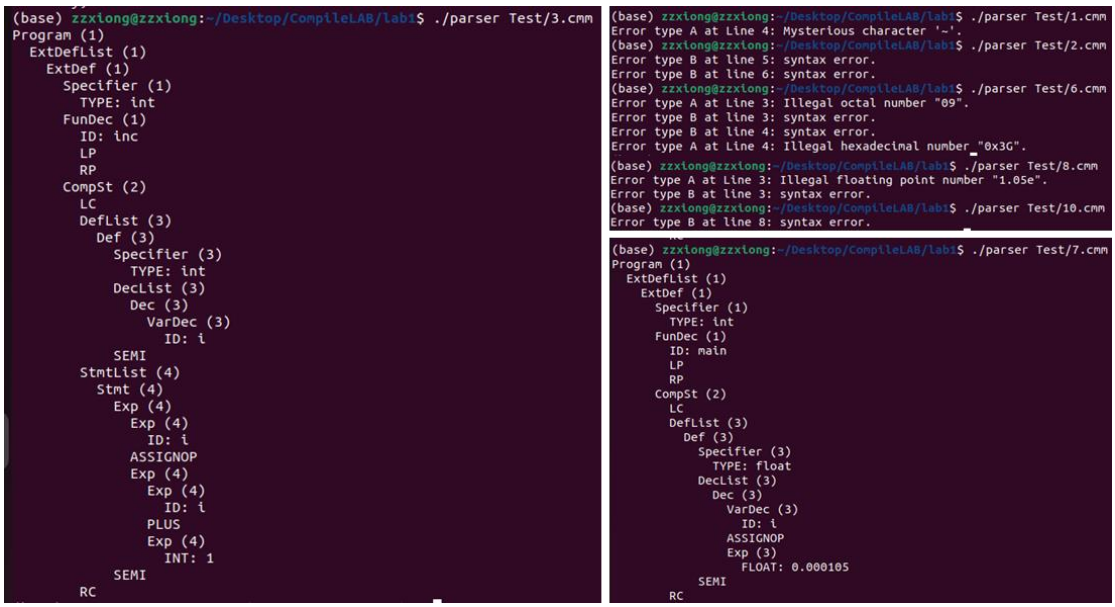

图 2 测试结果

三、实验收获

1. 熟悉了词法分析工具 GNU Flex 和语法分析工具 GNU Bison，在实践中进一步加深了对词法分析与语法分析的有关知识，如正则表达式、移入规约等的理解认识。

2. 通过检查语法分析的状态转换和错误恢复的过程，查看每一个状态所对应的产生式、该状态何时进行移入何时进行归约，进一步加深了对自底向上的语法分析过程的理解，并能够简单处理词法分析与语法分析中的异常和错误。