编译原理 实验 1 实验报告

201220062 黄子睿

一、程序功能：

a) 程序实现了 c—语言要求的所有基础词法和语法分析，并通过了 OJ 平台上的所有测试用例。

b) 选做部分完成如下：

i. 识别八进制、十六进制整数

ii. 识别指数浮点数

iii. 实现注释识别与相关报错

iv. 实现字符串与 char 型变量的识别

c) 下面介绍实现语法树的数据结构：

i. 在 node.h 中定义语法树节点 Node：

```
typedef struct node{
    int lineno;
    char* name;
    NodeType type;
    char* val;
    struct node* children;
    struct node* next; /* next sibling */
} Node;
typedef Node* pNode;
```

语法树的构建使用 children-sibling 数据结构。

将节点的种类记录为枚举变量 NodeType，主要可以分为 TOKEN，即叶子节点，与 NOT-A-TOKEN，即非叶节点。节点建立基于两个函数，分别为建立非叶节点的变长参数函数 newNode，与建立叶子节点的函数 newTokenNode。其中，对 newNode 函数中变长参数的访问使用了 C 库函数中的 va_start，va_arg，va_end 宏，具体代码如下：

```
if(argc > 0){
    va_list ap;
    va_start(ap, argc);
    curr->children = (pNode)va_arg(ap, pNode);
    pNode prev = curr->children;
    for(int i = 1; i < argc; i++){
        assert(prev);
        prev->next = (pNode)va_arg(ap, pNode);
        if(prev->next) prev = prev->next;
    }
    va_end(ap);
}
```

文件 node.h 中的函数在词法与语法分析中将会被反复调用。因此，使用关键字 inline 以降低反复调用的时空开销，优化程序运行效率。

ii. 识别八进制、十六进制整数与指数浮点数：

1. 这一功能使用正则表达式实现。除了正确的词法形式外，还设计了错误词法形式的识别，使得词法分析可以具体识别分析各种整数、浮点数的书写形式，同时指明可能出现的词法错误。举例如下：

```
DEC (0)|([1-9]{digit}*)
OCT 0[0-7]+
HEX (0x|0X)[0-9A-Fa-f]+
INT {DEC}|{OCT}|{HEX}
/*Lexical Error -- INT*/
ERRI_OCT 0[0-7]*[89]{digit}*
ERRI_HEX (0x|0X)[0-9A-Fa-f]*[G-Zg-z][0-9A-Fa-f]*
```

iii. 实现注释识别与相关报错：

1. 以“//”开头的注释行：由正则表达式识别。

```
COMMAND_LINE \/\/[^\n]*
```

2. 以“/**/”识别的注释块：
   a) 正则表达式识别到“/*”后，调用函数 int commandHandler()处理，正确则返回 1，若没有“*/”作为注释的终结，则返回 0 指出词法错误。
   b) 函数 commandHandler()利用 flex 中自带的 input 函数逐字从 yytext 中读取字符内容。用变量 state 识别“*/”，通过自动机的方式实现对注释块的词法分析。具体见下：

```
int commandHandler(){
    int state = 0;
    while(state != 2){
        char c = input();
        if(c == 0) return 0;
        else if(c == '*') state = 1;
        else if(state == 1 && c == '/') state = 2;
        else state = 0;
    }
    return 1;
}
```

iv. 实现字符串与 char 型变量的识别：
1. 字符 char 变量的识别：
   a) 在 flex 词法分析中修改 TYPE：

```
TYPE int|float|char
```

   b) 用正则表达式识别单引号表示的字符变量，同时辨别引号中出现多个字符的词法错误，具体见下：

```
CHAR '[^']'
/*Lexical error -- CHAR*/
ERRC '[^']{2,}'
```

2. 字符串的识别：
   a) 利用正则表达式识别，字符串识别的正则表达式如下：

```
STRING \"((\\.)|([^\"]))*\"
```

   括号中识别字符串引号内部的内容，分为两种，一是转义符与之后的任意一个字符（包括引号），二是除引号外的任意字符。
3. 在语法分析中实现对指针的分析，具体为增加文法 VarDec -> STAR ID，使得语法分析器可以识别指针变量的定义。

```
VarDec:         ID
        |       STAR ID
        |       VarDec LB INT RB
        |       error RB
        ;
```

二、程序运行：
a) 进入 Code 文件夹，命令行输入 make 将完成 parser 的编译，make run 指令将遍历 Test 中的所有文件并将结果打印至 Result 文件中的.output 文件中。make clean 将删除 Result 文件夹与 Code 文件夹中的所有输出文件。注意 Result 与 Code 必须在同一个文件夹下，Makefile 与脚本 autorun.py 在 Code 中。