编译原理

实验一

1 程序功能

程序实现了对 C—源代码的词法分析与语法分析，并对其中可能出现的词法错误和语法错误进行提示。

词法分析功能主要使用 GNU Flex 这一词法分析工具实现，语法分析功能主要使用 GNU Bison 这一语法分析工具实现。

按照指导书上的说明即可快速完成对词法分析和语法分析功能的实现，本次实验的一个难点在于当 C--程序无任何语法和词法错误时，需要按先序遍历输出程序对应的语法树。

首先要完成的就是对语法树的数据结构设计。考虑用左儿子右兄弟的方式存储程序对应的语法树，将每个终结符或非终结符的行号、类型、名字、值、左儿子节点和右兄弟节点等信息放入一个结构体中，作为代表这一终结符或非终结符的一个节点。其中，节点类型根据输出时打印信息的不同要求分为 ID_TOKEN、INT_TOKEN、FLOAT_TOKEN、TYPE_TOKEN、OTHER_TOKEN 和 NOT_A_TOKEN。具体代码如下：

```
typedef enum nodeType
{
    INT_TOKEN,
    FLOAT_TOKEN,
    ID_TOKEN,
    TYPE_TOKEN,
    OTHER_TOKEN,
    NOT_A_TOKEN
} NodeType;

typedef struct node
{
    int line;
    NodeType type;
    char* name;
    char* val;
    struct node* child;
    struct node* sibling;
} Node;

typedef Node* pNode;
```

然后通过定义一系列的函数来实现对树的构建、删除及打印信息的操作。其中，构建叶节点（与终结符对应）需使用 newTokenNode 函数；构建非叶节点（与非终结符对应）需使用 newNode 函数；删除语法树（释放内存）需使用 delNode 函数；打印语法树信息需使用 printTree 函数。值得一提的是，因为 lexical.l 和 syntax.y 中都有大量的对上述函数的调用，通过将它们设置为 static inline 函数的方式能减少调用开销。

构建非叶节点时，由于语法产生式右侧的符号数量是不定的，所以需要传入该产生式右侧的符号数量并用到变长参数，然后在 newNode 中借助 va_arg 函数遍历参数列表设置当前节点的左儿子节点，再设置左儿子节点的右兄弟节点，具体代码如下：

```
va_list arg_ptr;
va_start(arg_ptr, argc);
pNode tempNode = va_arg(arg_ptr, pNode);

curNode->child = tempNode;
for (int i = 1; i < argc; i++)
{
    tempNode->sibling = va_arg(arg_ptr, pNode);
    if (tempNode->sibling != NULL)
    {
        tempNode = tempNode->sibling;
    }
}

va_end(arg_ptr);
```

除了完成基本要求外，程序还完成了选做1.2的要求，实现了对指数形式浮点数的识别，并对不符合词法定义的指数形式浮点数给出词法错误的提示。这一功能通过分别对符合词法定义和不符合词法定义的指数形式浮点数书写正则表达式即可实现。

修改后的浮点数正则表达式如下：

```
FLOAT {digit}+"."{digit}+|{digit}*"."{digit}+[eE][+-]?{digit}+|
{digit}+"."{digit}*[eE][+-]?{digit}+
```

不符合词法定义的浮点数正则表达式及响应函数如下：

```
"."{digit}+|{digit}+"."|{digit}*"."{digit}+[eE]|{digit}+"."{digit}-
*[eE]|{digit}+[eE][+-]?{digit}*|"."[eE][+-]?{digit}+ {lexError = 1;
printf("Error type A at Line %d: Illegal floating point number
\"%s\".\n", yylineno, yytext);}
```

同时，程序也通过编写正则表达式的方式，实现了对不符合词法定义的标识符的错误提示。这一功能只需识别出以数字开头的标识符即可简单实现。

2 程序运行

在源程序所在文件夹下进入终端，依次输入以下指令，即可得到编译结果 parser:

```
zr@ubuntu:~/compilerlab$ bison -d syntax.y
zr@ubuntu:~/compilerlab$ flex lexical.l
zr@ubuntu:~/compilerlab$ gcc main.c syntax.tab.c -lfl -o parser
```

将测试用例也置于该文件夹下，测试时，在终端输入“./parser ”加上待测试文件名即可得到结果。例如，若要对 test1.cmm 文件进行测试，输入指令及结果如下：

```
zr@ubuntu:~/compilerlab$ ./parser test1.cmm
Error type A at Line 4: Mysterious character '~'.
```