编译原理 实验一报告

181860155 朱晓晴

181860155@smail.nju.edu.cn

1 编译方式

在/Lab/Code 目录下执行以下指令进行编译，得到可执行文件 parser：

`make parser`

执行以下指令进行测试，path 为测试文件路径：

`./parser <path>`

2 程序功能

2.1 语法树

语法树结点和相关的函数定义在 SyntaxTree.h 和 SyntaxTree.c 中，语法树结点 TreeNode 的定义如下：

```
typedef struct Node {
    NodeType nodeType;        //结点类型
    NodeValue nodeValue;      //结点值（数值、字符串）
    int lineno;               //结点词素所在行号
    int childrenNum;          //子结点个数
    struct Node* parent;      //父结点指针
    struct Node** children;   //子结点指针数组
} TreeNode;
```

相关的函数如下：

```
//创建结点，返回指针
TreeNode* createNode(NodeType type, int lineno);
//添加子结点
void addChildNode(TreeNode* parent, int count, ...);
//打印语法树信息
void printSyntaxTree(TreeNode* root, int depth);
```

2.2 词法分析

在词法分析部分，程序根据附录 A 的定义识别词法单元，创建语法树结点，并将相应的类型返回给语法分析模块。在这一部分，完成了识别浮点数的选做任务（任务号：14）。

除未定义字符错误以外，该部分额外定义了以下 3 个错误：

```
INT_ERROR       //识别八进制数和误以 0 开头的十进制数
FLOAT_ERROR     //识别错误的非指数型浮点数，.25 和 2.等
HEX_ERROR       //识别十六进制数
```

匹配到以上 3 个词法错误后，程序会报错（type A），但将 INT、FLOAT 和 INT 作为返回值传递给语法分析模块。定义以上错误的目的是使得程序在分析诸如“int a=0xFF22;”的语句时，无需将 0xff22 识别为 INT、ID 和 INT，继而进行非必要的错误恢复。同时，如此定义可以使得错误信息更加准确。

2.3 语法分析

在语法分析部分，程序在 C--文法的基础上自定义了若干错误恢复，例如对变量声明错误、表达式错误、分号缺失、括号缺失等错误的恢复，不在此一一罗列。匹配到相应的错误恢复规则后，会输出错误代码的具体内容和修改建议。

由于 C--文法中存在可以生成空串的非终结符，并且主观定义错误恢复方式存在一定的局限和偏好，因此必然存在一些程序无法准确恢复的错误。对于这些错误，以 yyerror 被调用的次数为标准，如果在 yyerror 被调用时，上一次的错误仍未输出信息，那么程序会输出“undefined error”的字样。

此外，程序支持一次编译多个文件，当文件数大于 1 时，程序会同时输出**文件名**和编译信息。

3 心得体会

1. 在错误恢复部分，理论上向文法产生式中加入越多 error 越容易恢复代码错误，并使得代码能够被顺利编译完毕。但实际实现时，要考虑到文法的特性和移入/归约冲突的存在，有选择地添加 error。同时，过多的 error 会导致一行错误语句最后被识别为两到三个错误语句，尽管在有限的测试用例中程序能够完成编译。但是，在调试模式中追踪 bison 行为就会发现程序并没有以理想的方式的恢复错误。
2. 编写 syntax.y 时，由于不清楚后续实验要对当前代码框架做出何种程度的改动，并未大量使用宏、内联函数等提高代码简洁性和可读性。同时，错误信息也分散在语义动作中，暂未实现批量管理和生成。在后续实验中，会对代码进行优化。