编译原理 Project1 实验报告

姓名:万子文

学号:151220102

邮箱:540304594@qq.com

如果测试过程中出现问题，希望助教可以通过邮箱联系我，避免因为gcc 版本或者其他原因产生问题。

一．实验完成情况

1. 利用 flex 工具实现了对程序的词法分析。具体实现在 lexical.l 文件中。包含了对浮点数，int(十进制，八进制，十六进制)的识别，通过了选择测试样例中浮点数和八/十六进制的部分
2. 利用 Bison 工具实现了语法分析，主要代码再 syntax.y 文件中。实现了附录中所有语法规则。并给出了结合性的声明代码。
3. 在(1)(2)的基础上构建了语法书，主要代码在 parsertree.c 文件里。构建过程大致如下:
   (1) 在词法分析的时候，在每个词法对应的操作函数中调用语法书中的函数，生成一个相应 token 的页结点
   (2) 在语法分析的时候，每当进行一次规约，调用相应的语义，生成这个规约对用的结点，并且将各个子节点关联起来，形成子树
   (3) 最后进行 program 规约的时候,将产生结点的地址存储为全局变量 root
   (4) 分析完毕之后一旦没有产生错误并且 root 不为空，就表示成功生成了一个根节点为 program 的语法书，则调用相应的函数对这棵语法树进行中序遍历，并输出最终的结果
4. 对于词法分析的错误，全部归为 Error A 并输出
5. 语法分析重构了 yyerror 函数，并且对于样例中的错误进行了特定的错误处理。但是由于错误处理种类过多，本次实验的错误处理并不完整。
   虽然构造了许多含有 error 的文法，但是输出统一输出 syntax error

二．实验说明

由于本次实验产生的代码文件并不多，所以采用手动逐条输入命令的方式。
编译步骤

1. flex lexical.l(产生词法分析代码)

2. bison –d syntax.y (产生语法分析代码)
3. gcc main.c syntax.tab.c parsertree.c -lfl -ly -o parser
   将上述所有代码编译，链接，最终产生的目标文件为 parser
4. 执行: ./parser example.c

三. 实验细节

1. 语法书的结构，产生以及遍历。语法书的结点定义如下

```
struct ParserNode
{
    NodeType m_NodeType;
    SyntaxType m_SyntaxType;
    int m_lineno;
    int m_depth;
    int m_childrennum;
    struct ParsingNode* m_firstchild;
    struct ParsingNode* m_parent;
    struct ParsingNode* m_nextsibiling;
        union
        {
                char* IDname;
                TypeType type_value;
                int int_value;
                float float_value;
                RelopType relop_value;
        };
};
typedef struct ParserNode ParserNode;
```

   :
   其中 NodeType 类型指明了改节点的性质(是否根节点，是否为空)
   SyntaxType 类型指明了语法成分
   剩下的 几个变量用来维护树形结构
   最后的 union 维护对应不同类型时的值
   维护了如下函数，实现各个节点的构造以及相互连接:

```
ParserNode* GenerateVariableNode(SyntaxType typex,int childrenNum,...);
ParserNode* GenerateTypeNode(int lineno,char* text);
ParserNode* GenerateIntNode(int lineno,char* text);
ParserNode* GenerateFloatNode(int lineno,char* text);
ParserNode* GenerateDummyNode(SyntaxType typex);
ParserNode* GenerateRelopNode(int lineno,char* text);
ParserNode* GenerateNormalTerminalNode(int lineno,SyntaxType typex);
ParserNode* GenerateIDNode(int lineno,char* text);
```

2. 实现了部分错误文法，例如下图:

```
ExtDef:Specifier ExtDecList SEMI{$$=GenerateVariableNode(AExtDef,3,$1,$2,$3);}
       |Specifier SEMI{$$=GenerateVariableNode(AExtDef,2,$1,$2);}
       |Specifier FunDec CompSt{$$=GenerateVariableNode(AExtDef,3,$1,$2,$3);}
       |Specifier ExtDecList error SEMI
       |Specifier error SEMI
```

   但是缺点是没有针对不同的错误类型给出更具体的语法错误原因。

3. 修改词法分析以及语法分析中 yylval 以及 token 的类型，使得每次结果变为语法分析中的一个结点。词法分析的结果是返回一个根节点，语义操作就是将

多个子树合并成一个以当前 syntax 为跟的子树。最终规约形成一个以 program 为根节点的语法书。

四．测试样例试验结果

在实际测试中发现,在所有必做的测试样例表现很好，可以正确识别出所有的问题。在我完成的选作部分的测试样例中，通过了绝大部分的选座，但是有一个样例少识别了一个错误。

因为时间关系没有来得及调试这个 bug,并且多进行更多的测试。我将在下次实验之前修复这个 bug,并加入更多的错误文法。