编译原理 Project2 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. 在实验 1 生成的语法树基础上便利了整个语法树，完成了对程序的语义分析。本次是在语法树的基础上实现的，所以实现时候与书中的做法略有不同
2. 实现了一个用链表维护全局的符号表，符号表的结构会在后文阐述，实现了一个全局 structlist 用来维护定义好的 struct 结构
3. 对实现了讲义中要求的所有必须要求，但是在自己测试的时候仍然在一些样例中存在一些小瑕疵，所以还是会在后续继续完善这次的程序.
   r

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(*在工程文件下的 lexical 子目录下)

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 ../semantic/symbolTable.c ../semantic/covert.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. 符号表的维护

按照讲义上的指导定义了 FieldList 与 Type，
自己在此基础上定义了符号表表项，结构如下：

```
struct SymbolTableEntry_ {
    char* id;
    int lineno;
    enum{FUNC=0,VAR} kind;
    union{
        struct{Type VariableType;} Variable;
        struct{Type RetType; FieldList InputType;} Function;
    }u;
    SymbolTableEntry next;
};
```

以链表的形式维护，并且通过里面的属性区分函数和变量，并保存了函数或者变量相应的信息

2. 语义分析案例:
   一个简单清晰的函数:

```
FieldList CovertDef2FieldList(ParserTreeNode* x){
    ParserTreeNode* temp=x->m_firstchild;
    Type inittype=CovertSpecifier2Type(temp);
    temp=temp->m_nextsibiling;
    FieldList ret=CovertDecList2FieldList(temp,inittype);
    return ret;
}
```

将 Def(Def → Specifier DecList SEMI)转化成一个 FieldList

首先从第一个子节点 Specifier 获取了一个 Type 信息，传递到他下一个兄弟出，下一个兄弟根据这个继承属性，产生新的 FieldList
*在此给出一个简单的例子，具体的函数实现都在 semantic 文件夹下的 symbolTable.c 以及 covert.c 中

3. 错误处理
   对于不同的产生式处理不同的错误信息。特别在处理信息的时候需要用到一个 Exp 的类型属性，这个可以按照 SDT 的方法来产生，最后通过判断当前生成文法中各个 Exp 的类型以及综合其他信息，就可以完成对所有错误的判断

四．实验思考

实际使用中和书上提到的理论还是有很大的差距，很难在整个程序的语法树上构建一个完美的 SDT,所以实现里面程序变得很啰嗦，代码量也很大，完全没有书上的表达式简洁.
但同时，书上的理论还是可以在实践中给与很多的指导，在实践中我采用了大量的递归函数，函数设计的形式几乎是按照书上的 SDT 来设计的。通过这次实验，深入

的理解了理论和实践结合的道理.