实验二　语义分析

131220128 杨帆

一、实验环境

操作系统：　GNU Linux Release: Ubuntu 12.04
软件版本：　GNU Flex version 2.5.35；　GNU Bison version 2.5

二、实验内容

本次实验，任务是在词法分析和语法分析的基础上对C--语言的源代码进行语义分析和类型检查，并打印分析结果。

虽然语义分析在整个编译器的实现中并不是难度最大的任务，但却是最细致、琐碎的任务。因此需要用心地设计诸如符号表、变量类型等数据结构的实现细节，从而正确、高效地实现语义分析的各种功能。

三、实验过程

整体思路：

在实验一中我们已经实现了语法树的生成和输出，而实验二就需要在我们构造出的语法树上自底向上对每个节点进行语法分析。首先需要对生成的语法树进行深度优先的遍历。当遍历到叶子节点时（如ID、INT、FLOAT这样类型的节点）遍需要将该节点的信息记录（即将对应信息插入对应位置的符号表中）并传递。传递信息的过程就涉及到了课堂上所说的有关继承属性和综合属性的内容。在深度优先遍历的过程中设置一些判断条件，来实现对语义错误的辨识。

完成的功能点：

必做内容：错误类型1~错误类型12
选做内容：错误类型13~错误类型17
　　讲义上选做部分2.1、2.3

程序结构与编译过程：

在实验一的基础上新加入semantic.c　semantic.h　node.h几个文件
指令：bison –d syntax.y　　flex lexical.l
gcc main.c semantic.c syntax.tab.c –lfl –ly –o parser
或实验中使用的大班提供的makefile文件，直接执行make语句即可。

重要数据结构的表示和主要内容的实现方式：

整个程序对变量、数组、函数返回值、结构体的类型用枚举类型实现：

```
enum KType{
    INT, FLOAT, ARRAY, STRUCT, ERROR
};
```

本次实验我共维护了符号表和函数表两张信息表：

```
struct SymbolMsg{
    char name[30];
    //********
    char strtype[30];  // var in which struct
    int visitedTag;
    int lineNumber;
    int isArray;    // if it is int array[], then isArray = 1, symType = 0;
    int dimension;  // array dimension
    int isStr;      // when time period in struct deflist
    int inStr;      // = 1 if the var is in a struct
    enum KType symType;
    struct SymbolMsg *nextSym;
    struct ArrMsg *arr;             // not null if isArray = 1
    struct FieldMsg *fieldList;     // not null if is Str = 1
    int err;    // = 1 if the struct def is wrong
};
```

符号表记录了普通变量、结构体的一些信息。包括类型、名字、是否是数组类型、是否是结构体、同一位置下一项指针等信息

```
struct FuncMsg{
    char name[30];
    char strtype[30];
    int visitedTag;
    int argnum;
    int isArray;  // ===0
    int dimension;
    int isDef;    //  = 1 if it is defined
    int lineNumber;
    enum KType rtType;
    struct ArgList *Args;
    struct FuncMsg *nextFunc;
};
```

函数表同上，多记录了一些函数相关的：是否被定义过、参数列表、维度等信息

其次对更深层次信息的记录：例如对函数的参数列表、结构体的域列表、数组维度和大小的记录都维护了相应的数据结构。

```
struct FieldMsg
{
    char name[30];
    enum KType fieldType;
    struct FieldMsg *nextField;
};
```

对结构体域信息的记录，在符号表中有对应的信息

整个程序从 program 节点开始，为所有节点定义函数，这个函数的参数是这个节点信息和一些传入参数，返回值是它需要返回的类型。例如：

```
void semantic();
void init();
void Program_t(struct Node *node);
void ExtDefList_t(struct Node * node);
void ExtDef_t(struct Node *node, struct SymbolMsg *newstr);
enum KType Specifier_t(struct Node *node, struct SymbolMsg *newstr);
void ExtDecList_t(struct Node *node, enum KType sType, struct SymbolMsg *
newstr);
```

当遇到叶子节点时，不仅需要处理节点，还需要将处理的变量（包括结构体、函数等）插入对应的符号表中。而在 Exp 节点函数的调用时，需要从符号表中读取它的参数、域或返回值等信息，从而进行相应的类型判断。

由于很多信息需要在树的节点之间进行信息传递的判断，例如参数类型、变量是否为数组等信息。故实验中的函数的参数中很多传入了指针参数进行信息的传递。也可以将函数的类型改为需要返回信息的类型。总之，本次实验需要传递的参数和中间信息是很多的。

```
void Exp_t(struct Node *node, struct SymbolMsg *SMes, struct FuncMsg *FMes,
struct SymbolMsg *newstr);
```

图中函数中的 FMes、SMes、newstr 等都是为了传递信息设置的参数

新建节点（即新建符号表节点）有两个对应函数，分别为 newSymbol()：新建变量、数组、结构体的函数；newFunc()：新建函数的函数。使用讲义中给出的哈希函数建立哈希表，将对应信息的名称哈希到指定位置。后进行信息的记录。

```
void newSymbol(struct Node *, enum KType sType, struct ArrMsg *vArrayi,
struct SymbolMsg *newstr);
void newFunc(struct Node *node, struct ArgList *args, enum KType fType, int
defed);
```

四、未能实现的部分和存在的 bug 以及程序的一些缺陷:

1. 如有多个没有类型名的 struct 类型定义，程序只会将它们的域都插入符号表中，从而无法判断某个域属于哪个 struct。从而不能很好地进行结构体的错误分析；
2. 未能完成选做 2.2 对作用域的实现；
3. 程序封装过程实现得不好，注释加入得过多，对数据结构的定义未能参考讲义中的实现方式，导致整个程序代码量过多，结构有些混乱；
4. 错误 5 错误 7 其实可以统一实现。

五、实验体会

本次实验让我对语法分析有了更深层次的体会，尤其处理了很多课堂上提到的继承属性综合属性的信息传递。了解了语法制导的具体过程。对参数传递的方式、指针的使用、信息链表的建立和维护都是很好的学习和练习的机会。本次实验耗时一周，代码结构实现得还不够满意，也存在着一些发现和尚未发现的 bug，我也会在接下来的实验中对其进行完善。总体来说我的收获很大！