编译原理实验二报告

学号	1190202401
姓名	陈豪

一、实验环境

Ubuntu20.04TLS

gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

flex 2.6.4

bison (GNU Bison) 3.5.1

二、程序实现的功能

本程序实现了对C--文法的语义分析，完成了指导书指明的选做一和选做二

必做内容

采用的数据结构是hash表加十字链表的方式，维护风格是imperative Style。hash函数使用的是指导书上提供的P.J.Weinberger提出的hash函数。

在实验指导书提供的数据结构基础上修改了type的表示

```
struct Type_
{
    Kind kind;
    union
    {
        BasicType basic;
        //  数组类型信息包括元素类型大小构成
        struct
        {
            pType elem;
            int size;
        } array;
        //  结构体类型信息是一个链表
        struct{
            char *name;
            pFieldList structureField;
        } structure;
        // 函数
        struct {
            int argc;           // 函数参数个数
            pFieldList argv;    // 函数的参数，使用fieldlist串联
            pType returnType;   // 返回类型
        } function;
    } u;
};
```

对于结构体为了区分struct{}a和struct a{}这样的两种形式所以在type的结构体中加入了name属性。如果是前者那么a存储在fieldlist中，同时在name中使用数字作为id。如果是后者那么a存储在fieldlist中，但type中的name为NULL。

除了指导书给出的数据机构外还使用的数据结构如下

```
/*符号表中的一个项*/
struct TableItem_
{
    int symbolDepth;  //  在符号表中的深度，可以方便查看后续有多少个同hash值的item
    pFieldList field;
    //使用十字链表所以需要两个指针，一个指栈中同一层的item，一个指相同hash的item
    pTableItem nextSymbol;  //  相同嵌套深度的符号，竖指针
    pTableItem nextHash;  //  同hash值的item，横指针
};
/*符号表中的hash表*/
struct HashTable_
{
    pTableItem *hashArray;
};

/* @brief 符号栈，主要处理在多重作用域下的情况*/
struct Stack_
{
    int stackDepth;
    pTableItem *stackArray;
};

/*@brief 符号表通过hash表和一个栈构成*/
struct SymbolTable_
{
    pHashTable hashTable;//符号表
    pStack stack;//符号栈，为了选做二而采用的结构
    unsigned unamedStructNum;//未被命名的结构体
};

/*函数声明链表。*/
struct FuncDeclarationStack_ {
    int stackDepth;//可以用来记录栈的大小，其实没用到
    pTableItem item;//需要注意的是这里面的item的symbolLength给我用来记录行号了，目的见选做分析
};
```

这里为上述的结构体定义都提供了相关的建立，插入，删除等操作。部分简单的操作使用了宏定义，可在.h文件中见。

在语义分析中，程序采用指导书的建议，在获取了语法分析树后，自顶向下的进行语义分析。从根结点开始对于每一个非终结符都建立同名的函数，并根据分析需要传递语法节点作为函数参数，如果需要父亲或者左兄弟的类型信息就额外传递类型信息。当父亲节点中需要儿子的类型信息时就让对应的儿子节点返回type，否则返回void。相关的函数基本都利用了语法分析树中的节点名字进行判断以选择合适的分析函数。

选做内容

选做一

首先在语法分析文件中添加了对应的产生式，然后在处理函数头定义的时候检查它的儿子的右兄弟名字是否是SEMI，如果是的话就不会进行函数体分析，同时将相关的声明加入函数声明链表中去。接下来检验函数声明和定义是否一致的情况通过检查链表中的信息进行检查。

在分析结束后将遍历链表查看是否有函数没有被定义，有没被定义的就打印出来。

选做二

当进入下一层CompSt结构时，将会将栈增长一层，并且相关符号的symbolDepth同时跟进栈的增长，以此避免不同作用域下相同变量名的冲突。当分析完后清除相关变量并且将栈减一层回到原来的状态。

三、程序编译方式

本次程序使用了makefile进行编译。编译命令如下

```
~/compilerLab/Lab2/code$ make
```

编译后，测试必做样例的命令如下

```
~/compilerLab/Lab2/code$ make havetodotest
```

编译后，测试选做样例的命令如下

```
~/compilerLab/Lab2/code$ make nothavetodotest
```

四、参考资料

- [1] 编译原理实践与指导教程
- [2] github