编译原理第二次实验报告

161220115 汤佳铭 1612201179 周科

实验任务

在词法分析和语法分析的基础上编写一个程序，对 C--源程序进行语义分析和类型检查，并打印分析结果

实验思路

数据结构

非函数符号表

变量及结构体符号表的数据结构，我们采用了教材上提供的数据结构，但在此基础上做出了一点修改。教材上的数据结构我们细究之下发现了一丝小问题，就是在表示结构体变量和结构体嵌套时会发生冲突。

比如下面的例子，分别是一个名为 u 的结构体变量，以及一个名为 A 的嵌套结构体。

```
struct A {
      int a;
      int b;
} u;
```

```
struct u {
     struct  A{
            int a;
            int b;

       }x;

 };
```

我们发现左边的结构体变量和右边的结构体在表示时，开头的几个 FieldList 和 Type 两者是完全一样的，直到右边的出现了名为“x”的 FieldList 才出现不同。这意味着在我们查表时，我们必须对这两个变量每次都遍历到最后的域，才能分清楚他到底是一个结构体，还是一个结构体变量。二层嵌套可能还好，一旦更复杂些（虽然本次实验可能遇不到），这无疑会增加程序的负担。所以我们做出了修改。

以左边的结构体变量 u 为例，教程上的数据结构只是针对于变量来设计的，对于 A 这种结构体类型符号，我们需要对数据结构进行如下修改：

1.在 kind 里面添加 STRUCTURE_TYPE 作为 A 的 type->kind。作为对比，此时 u 的 type->kind 为 STRUCTUER.

2.在 union 里面添加对应的 FieldList structure*type.*此时 A 的 type->structure*type 指向一个新的 type 节点 t1,t1 的 type 为 structure,t1->structure 指向第一个域，不*

*同域之间用 FieldList 的 st*type 节点进行连接。作为对比，此时 u 的 type_kind 为 structure,u->structrue 指向符号表中 A 对应的节点。

符号表类型为链表形式，以 Head，和 Tail 节点进行链表增删改查操作。

函数符号表

```
struct func { //函数
      char *name;
      Type retype;
      int pnum;
      struct parameter * para;
      struct func *next;
}*funcHead, *funcTail;
```

函数的符号表数据结构如上，parameter 结构体包含了一个 FieldList 指针指向函数形参，以及一个 parameter 指针指向存有下一个形参的 parameter 结构体。

选做部分：作用域问题

我们在分析到 ExtDef → Specifier FunDec CompSt 时，会逐个对函数的形参、函数的主体分别进行分析，在此之前我们用一个指针 now 指向变量符号表的尾部 Tail，用来记录将这个函数的形参以及局部变量填表之前，符号表应该是什么样。待我们处理完函数的形参和主体以后，我们再让符号表的 Tail 再次回到 now 位置，随后释放该函数部分添加到表里的变量的节点内存。

语义分析

我们采用的语义分析方法是在自底向上构造语法树之后，再通过函数递归自顶向下重新遍历语法树来完成语义分析。

函数列表

```
struct node *Get_Child(struct node *root,int i); //获得root 第i 个子节点

unsigned int Count_Child(struct node *root); //计算子节点的数量

void Init_Hash(); // 初始化符号表
bool Insert_Symbol_Var(FieldList f); //将节点插入符号表
FieldList Search_Symbol_name(char *name);//根据名字在符号表里面查找节点。
bool TypeEqual(Type p1,Type p2); //判断两个type 是否相等
Type Handle_Exp(struct node *ast); //处理Exp 语法单元
FieldList Handle_VarDec(struct node *root,Type t, int i);
//处理VarDec 节点
Type Handle_Specifier(struct node *root); //处理Specifier 节点
```

```
Type Handle_StructSpecifier(struct node *root);//处理StructSpecifier 节点
FieldList Handle_DefList(struct node *root,int i );//处理DefList 节点
FieldList Handle_DecList(struct node *root,Type t,int i);
//处理DecList 节点
int Handle_Compt(struct node *compst,Type t);//处理Compt 节点
void Handle_ExtDefList(struct node *root);//处理ExtDefList 节点
FieldList Handle_ExtDecList(struct node *root,Type t);//处理ExtDecList 节点
void Realse(FieldList func_local_val);//函数结束之后释放局部变量内存
void GoProgram(struct node *root); //处理Program 节点
```

其他

对匿名结构体的处理，我们设置了一个全局变量 int anonymous_count 来帮助处理他在符号表里面的位置，将其转为字符串之后给结构体命名，每命名一个匿名结构体数值加 1.

对函数的形参的处理，除了将它们填入函数节点以外，还得将他们也作为函数定义的局部变量填入变量符号表。