编译原理 实验二 语义分析

缪源清 161220093 myqlily@126.com

1 数据结构

1 概览

语义分析阶段维护两张表:符号表和类型表。

符号表(symbol_entry_table)采用 imperative style 的维护方式。表项指向的结构如下:

```
struct Symbol_Entry_{
    char name[32];              //变量名
    Type type;                  //变量类型(指针)
    Symbol_Entry hash_next;     //所在哈希表项中与之相连的后一个表项
    Symbol_Entry field_next;    //所在作用域中与之相连的后一个表项
    int field;                  //所在作用域标号
};
```

类型表(type_entry_table)的维护方式与符号表相同。表项指向的结构如下:

```
struct Type_Entry_{
    char name[32];              //类型名
    Type type;                  //类型(指针)
    Type_Entry hash_next;       //所在哈希表项中与之相连的后一个表项
    Type_Entry field_next;      //所在作用域中与之相连的后一个表项
    int field;                  //所在作用域标号
};
```

注意事项:

1. name 域的值。考虑到在一些情况下名字不能确定:变量/函数/结构重名、无名结构体定义、数组类型等。引入一个计数器，将以上情况中的表项名字定为计数器的数值，同时计数器自增。由于 C 语言中变量不以数字开头，所以方法不会引入与后续分析的冲突。

2. type_entry_table 存放的内容。结构、数组类型、函数类型。type_entry_table 存在的意义:1)存放类型 2)管理 Type_类型。任意 Symbol_Entry_中 type 域指向的类型一定和某个 Type_Entry_中的 type 域指向的类型是同一个(除整数类型 TYPE_INT 和浮点类型 TYPE_FLOAT)。Symbol_Entry_的 type 是某个 Type_Entry 中 type 的拷贝。

2 Type_的结构

```
struct Type_{
    enum {BASIC, ARRAY, STRUCTURE, FUNCTION} kind;
    union{
        int basic;                                          //基本类型
        struct {Type elem; int size;}array;                 //数组
        struct {ParaList function;FuncType ftp;}declaration; //函数
        FieldList structure;                                //结构体
    }u;
};
```

1)基本类型:

即 int 和 float 类型，此二类型直接定义在全局数据区。

2)数组:

结构参考了实验指导手册。需要创建一个 Type_Entry_，type 域指向此数组结构。

3)函数:

```
struct ParaList_{                    struct FuncType_{
    Type type;                           int tag;
    ParaList tail;                   Declaration pointer;
};                                   };
```

paralist 与 fieldlist 类似，function 域指向的第一个 paralist_中的 Type 是函数的返回值类型，其后的 ParaList 是函数的参数类型。

FuncType 是为了实现函数声明而设计的。tag=1 标志此函数被定义过。pointer 指向的是函数声明表。函数声明表是一个双向链表，表项如下:

```
struct Declaration_{
    int pos;                //声明所在的行数
    Symbol_Entry sy_en;     //声明的函数的符号表项的指针
    Declaration before;         //声明表前一个表项的指针
    Declaration next;       //声明表后一个表项的指针
};
```

函数被定义时，若之前声明过此函数，则将 tag 改为 1，并删除函数声明表中的对应表项。结束语法树的分析时，查看函数声明表是否为空，若不空，则输出相应的信息。

4)结构:

参考了实验手册的内容。在结构体的定义过程中维护一个局部的链表，此链表存放的是当前已定义的域的名字，在定义时，首先遍历此表，查看是否与之前定义的域重名，若重名则报错，若不重名，则在表的末尾加上此域的名字的节点。定义结束后，删除此表。

3 删除操作

删除符号表时，不考虑 type 域;删除类型表时，考虑 type 域。因为前者和某个 type 可能存在多对一的关系，而后者是一对一的。

4 作用域

每层大括号代表一层作用域。需要注意的是带参函数定义的作用域，函数的参数列表和它的定义是在同一层作用域的，参数列表算作定义的变量的一部分。

2 算法

前序遍历语法树，根据节点的类型进行处理。若有兴趣，可我的参考设计框图 design_pics.pdf。

3 实现功能

1.可检查出实验指导手册的 19 种错误，支持函数声明、嵌套作用域、结构等价。

2.结构体内定义结构体：

在 C 语言中，结构体内定义的结构体与外层结构体具有同样的作用域。实现了此功能。

测试方法见 README
test1~17 对应必做样例 1~17
test1b~6b 对应选做样例 1~6
test7b 是对结构体中定义结构体的测试。

4 对实验一的更正

修复 bug:
【1】int a[2][4];

```
VarDec:          ID W {stack[stackCount-1]->value = (YYSTYPE)$1;}    {reduce(1,"VarDec",*(int*)&@1);}
                 |   VarDec  LB{insert_flx("LB");}   INT{insert((YYSTYPE)$4,"INT");}   RB{insert_flx("RB");}{reduce(4,"VarDec",*(int*)&@1);}
                 ;
```

之前漏加标记部分的代码。使得数组定义的解析过程中发生了栈溢出。

【2】
int func(int a){

}

```
VarList:         ParamDec COMMA{insert_flx("COMMA");} VarList  {reduce(3,"VarList",*(int*)&@1);}
                 |   ParamDec {reduce(1,"VarList",*(int*)&@1);}
                 ;
```

之前参数设错了，修改为 1 后可以正确分析出带参函数。

【3】
return

```
Stmt:            Exp SEMI {insert_flx("SEMI");reduce(2,"Stmt",*(int*)&@1);}
                 |   CompSt {reduce(1,"Stmt",*(int*)&@1);}
                 |   RETURN{insert_flx("RETURN");} Exp SEMI {insert_flx("SEMI");}{reduce(3,"Stmt",*(int*)&@1);}
                 |   IF  S LP U  Exp RP V Stmt  {reduce(5,"Stmt",*(int*)&@1);}  %prec LOWER_THAN_ELSE
```

之前漏掉了“RETURN”的插入，修改后可以识别 return 语句

【4】添加了对函数声明的支持

5 代码风格说明

结构体名以下划线结尾；结构体的指针类型名是相应的结构体名去掉下划线。