实验二 语义分析

161220033 刁含顺 161220055 金亦凡

一、程序实现的功能

1、在实验一词法分析和语法分析程序的基础上编写一个程序，对测试文件进行语义分析和类型检查，并打印分析结果，本实验将语义分析相关的代码都放入了一个源码文件中，为sym_table.c与sym_table.h

2、语义分析之前先设计了符号表等数据结构，其中定义如下所示

(1)符号表

```
50
51   SymNode sym_table[HASH_LENGTH];
52   FuncNode func_table[HASH_LENGTH];
53
```

定义了两个符号表，sym_table[HASH_LENGET]与func_table[HASH_LENGTH]，分别是变量表与函数表

（为便于编写，定义结构体名称为XXX_，将对应的指针重命名，如：typedef struct Type_* Type;）

(2)其中变量表节点定义为

```
19  struct Type_ {
20      enum { BASIC, ARRAY, STRUCT } kind;
21      union {
22          int basic;
23          struct {
24              SymNode arrlist;
25              Type element;
26              int size;
27          } array;
28          SymNode structure;
29      } u;
30  };
31
32  struct SymNode_ {
33      Type type;
34      int lineno;
35      char* name;
36      bool CanBeAss;
37      SymNode hash_next;
38      SymNode param_next;
39      SymNode struct_next;
40  };
```

其中Type表示变量类型，共有三种基本的结构类型：BASIC(int, float), ARRAY, STRUCT。其中ARRAY类型用size表示大小并递归使用Type表示数组的类型，STRUCT类型使用变量节点表示结构体内部的成员变量。变量节点SymNode由类型、行号、变量名、可否赋值判定等成员组成，并且维护三种链表：哈希表中用于处理冲突的hash_next、用于维护函数输入参数表的param_next和用于维护上述STRUCT中内部成员变量表的struct_next。

函数表节点定义为

```
42  struct FuncNode_ {
43      Type returnType;
44      SymNode params;
45      char* name;
46      bool ifdefine;
47      int lineno;
48      FuncNode hash_next;
49  };
```

函数表节点由返回类型、输入参数表、名称、行号、是否已被定义以及哈希表处理成员hash_next组成，基本同SymNode，相比而言更加简单。

(3)针对变量表以及函数表实现了插表以及查表的功能，函数原型如下所示

```
64  void insert_sym_table(SymNode n);
65  void insert_func_table(FuncNode n);
66  SymNode check_sym_table(char* name);
67  FuncNode check_func_table(char* name);
```

3、具体实现为根据附录提示，在实验一实现的语法分析的基础上，对语法树进行遍历以进行符号表的相关操作以及类型的构造与检查。实现了每个语法单元的执行函数，在遍历语法树遇到的每个语法单元都调用对应的函数进行操作，用来发现出现的语义错误。在生成新的变量或函数的语法单元的执行函数中，每当生成了新的变量或函数，便加入变量表或函数表（当然，要经过查表操作）。不同的语义错误有不同的位置和标准，在此不再赘述。

4、在调试bug时，我们通过在每个语法单元中加入指示输出判断函数具体执行的地方，配合语法树进行分析，很快的就解决了bug，并且通过了给出的17个例子

5、由于水平有限，选做2.2暂时没有实现，非常抱歉。

二、程序如何被编译

1、提交文件中除了实验一中的所包含的，新增了 sym_table.h 与 sym_table.c 文件
2、程序编译可以修改 Makefile 中 test 部分的代码，分别测试 1.cmm 至 17.cmm

```
.PHONY: clean test
test:
	./parser ../Test/1.cmm
	./parser ../Test/2.cmm
	./parser ../Test/3.cmm
	./parser ../Test/4.cmm
	./parser ../Test/5.cmm
	./parser ../Test/6.cmm
	./parser ../Test/7.cmm
	./parser ../Test/8.cmm
	./parser ../Test/9.cmm
	./parser ../Test/10.cmm
	./parser ../Test/11.cmm
	./parser ../Test/12.cmm
	./parser ../Test/13.cmm
	./parser ../Test/14.cmm
	./parser ../Test/15.cmm
	./parser ../Test/16.cmm
	./parser ../Test/17.cmm
clean:
```

目前我们的 Makefile 文件如上图所示

(1)make clean
(2)make
(3)make test 或者./parser ../Test/*.cmm
进行测试