编译原理第二次实验报告

161240005 陈勇虎

1. 完成的功能:
   (1) 实验一已经完成的功能
   (2) 在词法和语法分析的基础上，满足规定的代码可以进行语义分析和类型检查，并且打印分析结果
   (3) 同样，为了方便测试，lab2 中输出的内容颜色统一设定为 Cyan 颜色,部分打印情况如下，颜色设定情况可以见 Code/grammertree.h 文件

```
Output of test3.cmm
Error type 3 at Line 4: Redefined variable "i".
Output of test4.cmm
Error type 4 at Line 5: Redefined function "func" .
Output of test5.cmm
Error type 5 at Line 3: Type mismatched for assignment.
Output of test6.cmm
Error type 6 at Line 4: The left-hand side of an assignment must be a variable.
Output of test7.cmm
Error type 7 at Line 4: Type mismatched for operands.
Output of test8.cmm
Error type 8 at Line 4: Type mismatched for return.
```

   (4) 目前已完成了所有的样例，包括必做和选做的内容。
2. 实现方法:
   (1) 实验中已经使用 flex 和 bison 实现了词法分析和词法分析
   (2) 对于语义分析的实现，首先是使用 lab1 的功能建立语法分析树，然后遍历结点依次进行语义分析，设计到类型的构造与检查，以及符号表的操作，关于类型变量我参考了讲义中的内容
   (3) 符号表的实现，我采用了哈希表的方式，哈希表的存储，查询相关函数可以参考 Code/semantic.h 文件
3. 数据结构
   1. 实验中使用的数据结构不再赘述
   2. lab2 中，需要构造如下的类型变量，原型见讲义

```
typedef struct Type_ {
  Kind kind;
  union {
    int basic;
    struct {
      Type elem;
      int size;
    }array;
    FieldList structure;
    struct {
      FieldList parameters;
      Type funcType;
      int paranum;
    }function;
  }u;
}Type_;
```

3. 此外，对于域的数据结构如下,简单说明和讲义不同的几个参数

```
typedef struct FieldList_ {
  char* name;
  int lineno;
  Type type;
  Nbool defined;
  FieldList tail;
  char parent[32];
  Hash hashflag;
}FieldList_;
```

1) int 型变量 lineno:域开始位置的行号，主要是为了在后面检测是否出现的函数都已经定义，如果未定义，将会输出其行号.
2) Nbool 型变量 defined:Nbool 为自定义类型，用于区分该域对应的是否是未定义的函数，定义的函数，亦或是函数参数等等.
3) char 型数组 parent:在作用域中，用于表示该域对应的变量的作用域，在后续处理作用域嵌套的问题
4) Hash 型变量 hashflag:Hash 为自定义类型，后续用于处理 hash 函数过程中的一些特殊情况

具体实现方式可以参考 Code/semantic.h 以及 Code/semanticTraverse.h 文件中的具体函数

4. 编译运行方法

待分析文本均位于 Test 文件夹下面，讲义中所有样例，包括选做样例也在其中

1 若需要分析名为 testaa.cmm 的文件，将 Code/Makefile 文件内容 test 伪目标内容修改为的 "./parser ../Test/testaa.cmm"，并在 Code 目录下 make && make test 即可

2 为了方便测试,添加了以下几个伪目标执行方法.

```
.PHONY: clean test CMM C all
test:
	./parser ../Test/test4.cmm

CMM:
	for n in $(INDEX_CMM);\
	do\
		echo "Output of test$$n.cmm";\
		./parser ../Test/test$$n.cmm;\
	done

C:
	for n in $(INDEX_C);\
	do	./parser ../Test/test$$n.c;\
	done

all:
	for case in $(shell find ../Test/*);\
		do	./parser $$case;\
		done
```

具体为；

1) make && make test 分析特定文件

2） make && make CMM 分析所有 CMM 后缀文件

3） make && make C 分析所有 C 后缀文件

4） make && make all 分析 Test 目录下所有文件

5. 实验总结

借助 flex 和 bison 可以很快完成对一段代码的分析，并通过自定义的数据结构完成对代码分析树的建立。得到我们需要的语法分析树后，我们可以对语法分析树进一步进行语义的分析，对输入的代码进行第二步的分析。