编译原理lab02 实验报告

161240056 谈婧

本次实验完成了对于输入代码词法分析/语法分析得到的语法树进行语义分析和类型检查的任务。

完成功能点

- 对于输入语法树进行语义分析，建立符号表。
  - 支持分析c—所支持的int,float两种basic变量、数组变量和结构体变量的定义部分。
  - 支持数组和结构体变量的嵌套定义（包括但不限于数组元素为结构体变量的多维数组，包含多维数组的结构体变量）
  - 支持n维数组变量的k(k < n)维解析，例如：二维数组a的a[]赋值分析。
- 对于输入语法树和符号表进行类型检查
  - 支持必做部分给出的17种类型错误的检查和选做2.3部分的改进内容。

实现方法与数据结构表示

- 数据结构：符号表的实现
  - 本次实验中的符号表采用链表表示，分为两个链表。
  - 包含基础变量、数组变量和结构体的链表中每个结点都为一个SymTable结构体（构成如图）所定义。

```
struct SymTable_
{
    char * name;
    Type type;
    SymTable next;
}SymTableNode;
```

  - 另一个链表用于存储定义的函数，每个结点为FuncTable所定义。如此设计的原因是函数需要记录的属性和另三个变量不太一致，因此比较方便分开存储。

```
struct FuncTable_
{
    char * name;
    Type returnType;
    FieldList VarList;
    FuncTable next;
    int isDefined;
}FuncNode;
```

  - FuncTable的结构体中包含字符串name，存储该函数的名字；包含Type类型的returnType，用于记录该函数的返回值类型；有FieldList类型的VarList，FieldList主要用于记录结构体参数或者函数参数，这里存在VarList中；指向了链表的下一个结点；isDefined是一个用于记录这个结点是否被定义过，这是因为如果扫描到未定义函数id，如果直接exit就会影响之后错误的判断，因此只能给未定义做标记，但还是将他们当作普通FuncTable结点。
  - Type和FieldList类型：
    - Type是用于表示语法树结点类型的结构体，kind中可以选择是基础变量或数组变量或结构体，其中结构体u中存储了具体相关参数。UNDEFINED是给检查到了未定义变量预留的类型，如此仍然可以返回一个TypeNode，保证程序继续进行查错。

```
struct Type_
{
    enum {BASIC, ARRAY, STRUCTURE, UNDEFINED} kind;
    union {
        int basic;
        struct {Type elem; int size; int dimension;} array;
        FieldList structure;
    }u;
    char * StructName;
}TypeNode;
```

- FieldList用于表示结构体或者函数的域。

```
struct FieldList_ {
    char* name;
    Type type;
    FieldList tail;
}Field;
```

- 实验方法：
  - 概述：
    - 首先对于语法树进行语义分析，建立语法树（symtable.c）。这里主要是分析了以ExtDef和Def为根的树上定义的变量，并在符号表中新建条目。在分析定义的同时开始检查有没有重复定义的错误。
      - 在语义分析数组时，采用函数递归检查VarDec结点的数量得到数组维度和大小，见函数TraverseDec, symtable.c。
    - 重新扫描了语法树之后再进行类型检查(check.c)。为了减少递归访问次数，为每一个语法树结点加入了一个新属性 ExpType，专门记录该Exp结点的类型属性，方便类型检查。对于Exp进行递归，不断分情况讨论，确定每一个Exp对应的表达式的类型。
      - 对于函数参数的比较依赖于存储函数定义和解析函数调用语句存储参数的顺序一致，所以直接用循环一对一比较每个参数链表中对应条目是否一致，此法还可以得知两个参数链表的长度。
      - isEqual函数中负责比较两个Type是否一致。其中结构体的相等判定按照选做2.3的要求实现了。为了保证如果遇到未定义变量仍能够继续类型检查，isEqual不会在两个输入比较参数中存在至少一个参数type为UNDEFINED时报错，而会用另一参数的type赋值给undefined参数。
  - 主要代码symtable.c和check.c函数说明：

函数定义	作用
symtable.c：语义分析	
void insertField(FieldList *Head, FieldList *node)	insert函数用于建立链表：结构体成员链表和函数参数链表。
static void checkPreSymTable(char * id, int line)	checkPreSymTable函数用于定义变量前检查已有符号表中是否存在重复定义。
static void checkStructMember(char * id, Type target, int line)	checkStructMember函数用于在定义结构体成员前检查结构体中是否存在重复定义。

函数定义	作用
symtable.c：语义分析	
static void checkFuncTable(char * id, int line)	checkFuncTable函数用于在定义函数前检查函数表中是否存在重复定义。
int getsize(Type type)	getSize函数可以返回给定类型type的大小。
void AddSymTable(char * id, Type type, int line)	AddSymTable函数利用输入id和type生成符号表新条目，并调用checkSymTable检查是否存在重复定义，但即使存在也继续定义。
char * TraverseDec(TreeNode root, int *num, int *ans)	TraverseDec对于root为根的子树进行遍历，用于判断变量是否为多维数组。函数中记录VarDec结点出现的次数num（num = dimension+1）和数组大小(ans = int * int * …)。
Type getType(TreeNode specifier)	getType用于分析specifier结点以及其子树，返回specifier所定义的类型。
void AnalyzeExtDecList(Type type, TreeNode ExtDecList)	AnalyzeExtDecList函数用于分析ExtDecList结点，ExtDecList为一种可能规约到ExtDef的情况。
void AnalyzeFuncDec(Type type, TreeNode FunDec)	AnalyzeFuncDec函数用于分析FuncDec结点，多为函数定义分析，FuncDec为一种可能规约到ExtDef的情况。
void AnalyzeExtDef(TreeNode root)	AnalyzeExtDef函数用于分析ExtDef结点，穷举了所有可能的规约情况，调用了AnalyzeExtDecList函数和AnalyzeFuncDec函数。
void AnalyzeDef(TreeNode root)	AnalyzeDef函数用于分析Def结点及其子树。
void SemanticAnalysis(TreeNode root)	SemanticAnalysis函数递归的遍历语法树，在syntax.y中调用。函数中检查所有ExtDef和Def结点，并调用了相应函数。
void Debugger()	Debugger函数打印符号表和函数符号表，作为debug工具存在。

函数定义	作用
check.c：类型检查	
void error(int num, int line, char * msg)	error函数是负责输出语义分析错误的函数，num为错误类型，line表示错误发生行数，msg为错误原因。
int isEqual(Type a, Type b)	isEqual函数负责比较a类型和b类型是否相同，相同返回0，不同返回1.
Type checkSymTable(char * id, int line)	checkSymTable函数检查id的变量是否在符号表中存在，如果存在则返回该类型，如果未定义则报错。类型检查不报重复定义错，因为这个错误会在定义变量/函数时被全面检查。

函数定义	作用
check.c：类型检查	
static Type checkStructMember(char * id, Type target, int line)	checkStruckMember函数检查在target类型（一定是结构体）中是否存在变量id，如果存在则返回该结构体成员的类型，不存在报错，重复定义不报错，原因同上。
static FuncTable checkFuncTable(char * id, int line)	checkFuncTable函数检查在FuncTable中是否存在名字为id的函数，存在返回该条目指针，不存在报错，重复定义不报错，原因同上。
void checkBigExp(TreeNode Exp)	checkBigExp函数用于递归的寻找并给Exp结点赋值ExpType. 在本函数中穷举了可以规约到Exp的所有情况，并给出了相应的动作。
TreeNode searchReturn(TreeNode root)	searchReturn函数在root结点为根的子树中寻找Return结点，并返回。
void checkReturnType(FuncTable Func, TreeNode CompSt)	checkReturnType中利用给定func结点，对其sibling结点调用了searchReturn函数锁定return位置，找到该函数的returnType，与FuncTable中记录的定义returnType进行比较，不同则报错。
void checkExtDef(TreeNode root)	checkExtDef则是完全为了每个检查每个函数的return变量而设计，这个函数如果搜索到func结点，就调用checkReturnType. 检查函数的return变量必须要在建立符号表之后进行，因为如果return的变量在函数体中定义，则在建立符号表的过程中可能因为该变量没有进入符号表而找不到。
void checkIniAssignment(TreeNode temp, Type defType)	checkIniAssignment检查初始化是否正确赋值。该函数递归寻找初始化时赋值的对应Exp类型，并和定义类型比较。
void check(TreeNode root)	check函数搜索整个语法树，检查所有Exp和ExtDef，并负责不重复检查Exp结点。

编译运行方法

- 可以使用make指令，或者使用shell脚本（在存在parser可执行文件后 ./test.sh）可以测试讲义所给样例，也可以使用命令行：
  - `bison -d syntax.y`
  - `flex lexical.l`
  - `gcc main.c syntax.tab.c -ll -ly -o parser [for mac os]`
  - `gcc main.c syntax.tab.c -lfl -ly -o parser [for linux]`
  - 如此会生成parser可执行文件，可以调用 ./parser [test url] 进行测试。
- 由于本实验是在mac os上完成，所以给出的parser可执行文件可能运行存在问题，建议重新编译后测试。
- 如果运行时出现segmentation fault，有可能是不知道啥原因被卡住了，可以重新运行几次。如果运行多次还是segmentation fault，那就是代码的问题了qaq。

实验总结

- C语言指针操作和链表实现不熟练导致了疯狂segmentation fault，debug时间过长。
  - 指针赋值最好先malloc再memcpy，除非该指针（cache或者temp）只起到为循环承上启下的作用。
- 代码模块化并且复用可以减轻debug负担，并且可以使程序更加简练。