编译原理第二次实验

> 水兵 191098191

一、文件结构

```
-map //开源库，c实现的unordered_map
sdt.c,sdt.h//错误检查程序
其余不变
```

亮点：1.宏简化代码 2.灵活使用指针辅助类型推导

二、SDT及错误判断

2.0 框架

对每个语法规则使用一个函数，其中函数的`形参`是**继承属性**，函数的`返回值`是**综合属性**。当一个语法规则有多个推导时候，根据该规则孩子节点属性调用相应推导。

在推导的过程中检查是否有错误，并报错。

2.1 变量类

2.1.1符号表

建立了局部变量符号表`localVariableTable`和全局变量符号表`globalVariableTable`。符号表使用**hash_map**，`key`为变量名称，`value`为指向符号表该项的指针

```
struct varItem_{char* name; Type type; int isInit;}
```

如上图所示，表项为该变量名称，它的类型，以及**是否初始化**

2.1.2 变量类型推导

变量定义可以在三处，全局变量，局部变量，和函数形参。它们的原理基本相同。现在举局部变量的例子。

> Def → Specifier DecList SEMI

`Specifier`会返回变量的`type`，这个综合属性最终会被传入`VarDec`中。`VarDec`是变量的具体定义，通过继承属性进行SDT。

```
VarDec->ID {错误检查;addTable(varItem(ID, VarDec.type, 0));}//添加到符号表中
VarDec1->{VarDec2.type = Array(VarDec1.type, INT.val);} VarDec2 LB INT RB //由于是左递归，所以属性的推导是从右向左，type作为继承属性逐级传递，直到遇到ID
```

变量的使用在`Exp`中，为定义的逆过程。通过综合属性**进行类型链表的遍历。**

```
Exp->ID{Exp.Type = lookUpTable(ID);错误检查;}
Exp1->Exp2 LB Exp RB {错误检查;Exp1.Type = Exp2.Type.tail;}//逆过程，消除一级数组
```

2.2 函数类

2.2.1 函数表

```
struct funcItem_{char* name; FieldList para; Type retType; int isDef; int
lineSum; int lineNo[256];};
```

函数表型记录的信息比变量要多，`para`记录函数的所有形参，`retType`记录函数的返回类型。`isDef,lineSum,lineNo`是为拓展2.1准备，会记录该函数是否定义，以及所有声明的所在行数。

2.2.2 支持函数声明（附加2.1）

在语法规则中新增 $ExtDef->Specifier\quad FunDec\quad SEMI$

这一项，并且对它做相应的制导。在这个制导中，仍然会新建一个函数表项`funcItem`，在`funcItem`的`lineNo`记录该声明的行数，并且置`isDe`f为0，表示并**未定义**。直到碰到定义，才置`isDef`为1。

在程序全部制导完成后，扫描函数表，如果发现`isDef`为0的函数项，说明该函数没有定义，则报错。报错行数记录在`lineNo`中。

2.3 记录类

仍然是新建了一张记录表，用于登记所有有名字的记录。制导关心的并不是记录的名字，而是记录具体的**记录结构**——手册中定义的数据结构，因此每次处理一个匿名记录类就会新生成一个**记录结构**，而有名字的记录则是从记录表中获取该名字对应的**记录结构**的指针，也就是说，程序在处理过程中始终传递的是**记录结构**的指针,而非记录的名字。

2.4 表达式

```
struct expVal_{Type type;union {...}val;
    enum {l_func, l_var} lType;
    enum {L_VAL, R_VAL} lr;
}
```

表达式的内涵非常丰富，它的推导是使用**综合属性**来完成的。表达式的**综合属性**要能够准确表达出该表达式的**值**以及**值的属性**。具体来说，要区分是左值还是右值，左值要区分是变量还是函数，变量或函数的类型，右值要区分整形还是浮点型。

通过一些辅助函数，可以准确获得这个表达式的类型。例如判断这个表达式是不是变量。

```
int isVar(expVal tmp){
    return (tmp != NULL) && (tmp->lr == L_VAL) && (tmp->lType == l_var) && (tmp-
>type != NULL);
}
```

2.5 将属性变为全局变量

例如判断函数返回值是否正确，当前`return`语句所在的函数返回值类型需要通过继承属性从函数定义处获得。但是从函数定义到`return`语句之间有很长的推导过程。因此可以用一个全局变量`current_func`记录下当前位于哪个函数之中。当函数定义的时候，将`current_func`置为该函数表对应的项，当函数定义完成后，将`current_func`置为`NULL`。这样就避免了继承属性的不断传递。