编译原理实验报告

实验三 语义分析

学号：202011998088　　姓名：臧祝利　　日期：2022年11月18日

实验要求

基于前面的实验，编写一个程序对使用 `C--` 语言书写的源代码进行语义分析，输出语义分析中发现的错误（涉及 17 种错误类型）并完成实验报告，实验中主要使用 `C` 语言。

- 基本要求

  1. 对程序进行语法分析，输出语法分析结果；✅
  2. 能够识别多个位置的语法错误。✅

- 附加要求

  1. 修改假设3在文法中加入函数声明（需要修改文法）并能够检查函数声明涉及的两种错误：函数声明了但没定义、函数多次声明冲突。✅
  2. 修改假设 4 在假设“变量的定义可受嵌套作用域的影响，外层语句块中定义的变量可以在内层语句块中重复定义，内层语句块中定义的变量到了外层语句块中就会消亡，不同函数体内定义的局部变量可以相互重名”。✅
  3. 修改前面的C--语言假设5，将结构体间的类型等价机制由名等价改为结构等价。在新的假设5下，完成对错误类型1-17的检查。✅

实验分工

实验分工如下：

> 臧祝利：基本程序框架+解决错误3，7，9，18
>
> 陈金利：解决错误4，15，17+附加要求2，3
>
> 孙泽林：解决错误5①，6，8，16，19+附加1+Debug
>
> 赵建锟：解决错误1，2，5②，10，11，12，13，14

实验环境

- Linux：Ubuntu 20.04 LTS
- Flex： V2.6.4
- GCC： V9.3.0
- Bison：V3.5.1

在终端输入如下命令：

```
sh cmd.sh
```

即可进行编译，若不行，可以将里面的内容复制，再进行编译；

实验设计

文件

限于篇幅，只展示部分，具体细节内容请查看文件；

SymbolTable.h

符号表使用`imperative style`形式实现；

定义了作用域和类型、符号表的数据结构以及相关函数的实现；

作用域

```
struct FieldList_
{
    char name[32];
    Type type;
    FieldList tail; //域包括域名，域类型，整体是作为一个链表；
};
```

类型：

```
struct Type_
{
    enum
    {
        BASIC,     //基本类型
        ARRAY,     //数组类型
        STRUCTURE, //结构体
        FUNCTION   //函数
    } kind;
    union
    {
        int basic; // 0->int , 1->float
        struct
        {
            Type elem;
            int size;
        } array_; //数组:包括元素的类型和数组的大小；

        struct
        {
            FieldList structure;
            char *name;
        } structure_; //结构体:包括结构体名和结构体的域

        struct
        {
            Type ReturnParameter;
            int num;
            FieldList parameters;
        } fuction; //返回参数的类型、参数个数、参数列表
    } u;
};
```

符号表头

```
struct SymbolBucket
{
    struct SymbolNode *head;
    struct SymbolBucket *next; //符号表，整体是一个链表
};
```

SymbolTable.c

在其中定义两个符号表：全局符号表和结构体符号表；

定义了作用域链表，每次遇到一个`Compst`就会进入一个新的作用域，整体是采用十字链表的形式，即横向为不同作用域，纵向为作用域中的符号表结点；每一次进入局部作用域就会新建一个结点然后进行插入；退出时会遍历找到尾部的作用域，把作用域中的符号表结点进行删除；

实现了`SymbolTable.h`里的函数，主要函数作用为插入符号表结点以及查询操作；

semantic.h

函数声明，主要函数就是对产生式进行分析，再加上错误输出函数，错误输出函数包括三类：错误9、错误12和其他错误的输出；

semantic.c

根据产生式，基于生成的语法树进行深度优先搜索；

在搜索的过程中实现符号表的填入与查找，由于产生式含有递归语法，因此程序也使用递归函数进行分析；

错误解决

error 3

错误3认为出现在以下推导中：

```
ExtDecList → VarDec
           | VarDec COMMA ExtDecList

Dec → VarDec
    | VarDec ASSIGNOP Exp
```

对于`ExtDecList`的产生式，检查`Vardec`里的参数是否存在，存在的话即为重定义；

对于`Dec`，分成了两小类，一种是单纯定义，另一个是定义时赋值，检查变量名是否在符号表中出现，出现的话即为重定义；

error 7

错误7认为出现在以下推导中：

```
Stmt → IF LP Exp RP Stmt
     | IF LP Exp RP Stmt ELSE Stmt
     | WHILE LP Exp RP Stmt

Exp → Exp ** Exp //**: AND OR RELOP PLUS MINUS STAR DIV
```

对于`Stmt`的分析式，检查`Exp`是否为整型，如果不是，则进行错误输出；

```
Error type 7 at Line 5: Type mismatched for operands.
```

对于`Exp`的分析式，则要用`CheckType(A,B)`函数分析一下符号两遍的`Exp`的类型，如果不一致则进行错误输出；

注意和`error5`区分

error 9

错误9认为出现在以下推导中：

```
Exp → ID LP Args RP
     | ID LP RP

Args → Exp COMMA Args
      | Exp
```

分成了如下几类：

(1) `Exp -> ID LP Args RP`

①函数定义的参数列表为空，但是`Args`不为空，进行错误输出，输出结果如下：

```
Error type 9 at Line 5: Function "func()" is not applicable for arguments "(float)"
```

②函数定义的参数列表不为空，但是和调用时的参数个数不同，会进行错误输出，输出结果如下：

```
Error type 9 at Line 5: Function "func(int,int)" is not applicable for arguments "(float,int,int)"
```

(2) `Exp -> ID LP RP`

③调用时无参数，但是函数定义时有参数，进行错误输出，输出结果如下：

```
Error type 9 at Line 5: Function "func(int,int)" is not applicable for arguments "()"
```

(3)对`Args`进行分析；**前提**：函数定义的参数个数和调用时的参数个数相同；

④第一个参数类型不对，报错；

```
Error type 9 at Line 6: Function "func(int,int,int)" is not applicable for arguments "(float,float,int)"
```

⑤第二个之后的参数类型不对，报错：

```
Error type 9 at Line 6: Function "func(int,int,int)" is not applicable for arguments "(int,int,float)"
```

错误9需要输出的报错信息比较复杂，因此以一个样例进行讲解：

以`函数定义的参数列表不为空，但是和调用时的参数个数不同`为例：

首先要分析`Exp -> ID LP Args RP`中Args是否为空，如果不为空，进行分析：

然后不断去获得Args的第二个孩子`COMMA`，得到调用语句的参数的个数，与函数参数域里的个数进行比较，如果不一致，则需要输出错误，在输出错误时，采用以下方式获得完整信息：

首先定义一个链表，用来存函数名称和函数名称+类型名称，此部分的插入见`FunDecAnalyse()`函数；

```
struct error9
{
    char *name;
    char *nameandtype;
    struct error9 *next; //对错误9的数据结构，包括函数名称，函数和参数类型构成的字符串，整体作为一个链表
};
```

```
                        char *defss = (char *)malloc(1024);  //申请空间
                        struct error9 *tmp9head = error9head;  //头结点
                        while (tmp9head != NULL)
                        {
                            if (strcmp(tmp9head->name, fuctionname) == 0)
                            {
                                strcat(defss, tmp9head->nameandtype);  //找到函数的对应名
称，"func(int,int,int)"
                                break;
                            }
                            tmp9head = tmp9head->next;
                        }
                        char *usess = (char *)malloc(1024);  //调用的参数类型
                        strcat(usess, "(");
```

```
Node *tmptmp3 = tmp3;
Node *ttmp2 = GetChild(tmp3, 1);
while (GetChild(tmptmp3, 1) ≠ NULL)
{
    Node *ttmp1 = GetChild(tmptmp3, 0);

    Node *tttmp = GetChild(ttmp1, 0);

    if (strcmp(tttmp→name, "INT") == 0)
    {
        strcat(usess, "int");
    }
    else if (strcmp(tttmp→name, "FLOAT") == 0)
    {
        strcat(usess, "float");
    }

    tmptmp3 = GetChild(tmptmp3, 2);

    strcat(usess, ",");
}
// last one
Node *ttmp1 = GetChild(tmptmp3, 0);
Node *tttmp = GetChild(ttmp1, 0);
//printf("name = %d\n", tttmp→intvalue);
if (strcmp(tttmp→name, "INT") == 0)
{
    strcat(usess, "int");
}
else if (strcmp(tttmp→name, "FLOAT") == 0)
{
    strcat(usess, "float");
}
strcat(usess, ")");
errorprint9(cur→linenum, defss, usess);
```

error 18

函数进行了声明，但是未定义；

解决方法：

定义一个链表，存储定义过的函数的名称；

```
struct Dec_Fuc //存函数的名称，用来检验是否被定义
{
    char name[32];          //函数名称
    int linenum;            //行号
    struct Dec_Fuc *next; //链表
};
```

定义函数(具体实现请看`SymbolTable.c`)

`void PushDecFuc(char *name, int column);`

作用： 向链表里插入出现过的函数的名称和行号；

`void CheckFucDef();`

作用： 遍历函数名称链表，然后在全局作用域中根据函数名称查询函数的声明或定义情况；

做法分析：由于出现过的函数名称都被放入了函数链表中，说明函数起码已经声明过；一旦出现根据函数名称查询的符号结点的`isdef≠1`，说明没有被定义，即报错为函数未被定义；

实验结果

（部分文件进行了更改，因此与课本的可能不一致）

`test3`

```
zangzhuli@ubuntu:~/Desktop/byyltest/Our$ ./parser Testfile/test3.cmm
Error type 3 at Line 7: Redefined variable "i".
```

`test7`

```
zangzhuli@ubuntu:~/Desktop/byyltest/Our$ ./parser Testfile/test7.cmm
Error type 7 at Line 6: Type mismatched for operands.
```

`test9`

```
zangzhuli@ubuntu:~/Desktop/byyltest/Our$ ./parser Testfile/test9.cmm
Error type 9 at Line 6: Function "func(int,int,int)" is not applicable for argum
ents "(int,int,float)"
```

`test18`

不会报错，但是如果只声明未定义会报错为：

```
zangzhuli@ubuntu:~/Desktop/byyltest/Our$ ./parser Testfile/test18.cmm
Error type 18 at Line 1: Undefined function "func".
```

实验反思

通过了所有的课本上的测试，大致完成效果不错，但还是可能存在错误类型没有考虑周全或者是重复考虑的情况；

对符号表的结构还应当进一步明晰，争取实现代码的简洁易懂。

这个实验好难啊QAQ