

哈尔滨工业大学

编译系统 2023 春

实验二

学院:	未来技术学院
姓名:	刘天瑞
学号:	7203610121
指导教师:	朱庆福

一、实验目的

1. 巩固对语义分析的基本功能和原理的认识;
2. 能够基于语法指导翻译的知识进行语义分析;
3. 理解并处理语义分析中的异常和错误。

二、实验内容

(一) 实验的运行环境

GCC Linux Release: 虚拟机 Ubuntu 20.04.1 kernel version 5.15.0-67-generic; GCC version 9.4.0; GNU Flex version 2.6.4-6.2; GUN Bison version 2:3.5.1

(二) 实验过程

在实验一词法与语法分析基础上加入语义分析部分的函数声明文法:

ExtDef->Specifier FunDec SEMI()

其中包括符号表与函数表如下图所示:

```
struct TABLE* table[16384]; //Storage structure or variable
struct FUNCTION* function[16384]; //Store function definitions or declarations

struct TABLE
{
    //0 is Yes, 1 is No
    //The name of the structure definition will be placed in the symbol table, but it is not a variable that can be used
    int is_def_struct;
    FieldList field;
    struct TABLE* next_for_openshash;
    int linenummer;
};

struct FUNCTION
{
    char* name; //function name
    FieldList field; //parameter list
    Type return_type; //return value
    int declaration; //Number of declarations
    int definition; //Define the number
    int linenummer; //For error printing
    struct FUNCTION* next_for_openshash;
};
```

其中也需要对每个语法单元即产生式中的非终结符设置相应的函数, 再根据产生式进行调用。

各部分新增程序功能如下所示:

根据子节点是否为空来递归调用函数 ExtDef():

Specifier():

进行区分: 若类型为 TYPE, 则根据具体情况设置为 int 或 float, 若类型为 StructSpecifier, 则再调用函数 StructSpecifier()。

StructSpecifier():

1. STRUCT OptTag LC DefList RC

通过读取 OptTag 中的 ID 信息, 生成结构体名称, 再查找该命名是否已经被定义过。调用 DefList()生

成域，若无错误，则将该结构体加入符号表中。

2. STRUCT Tag

通过读取 Tag 中的 ID 信息，在符号表中查找，找到则返回对应类型；否则报错。

DefList(), DecList(), VarList(), ExtDecList(), StmtList()

上述函数根据产生式进行递归调用，再根据具体情况将产生内容保存到 FieldList()中。

Dec():

1. VarDec

先调用函数 VarDec()。

2. VarDec ASSIGNOP Exp

根据产生式传递过来的 judge 值来进行判断，若为 0，则用结构体定义，对应错误类型 15；否则调用函数 Exp()；若返回类型 kind 与 type 不同，则对应错误类型 5。

VarDec():

1. VarDec LB INT RB

说明类型 kind 为数组 ARRAY，选择向上传递调用 VarDec()。

2. ID

首先创建一个域名列表 FieldList()，再读取变量 ID 值。函数声明的参数：返回 type，注意不加入符号表；函数定义的参数或定义的变量：需要先在符号表中查询，若重复定义则报出错误类型 3，否则加入符号表；结构体的定义：需要先在符号表中查询，若重复定义则报出错误类型 15，否则加入符号表。

FunDec():

创建一个函数 FUNCTION，设置对应的类型等参数值。

声明：判断是否已经加入 FUNCTION 表。

定义：判断是否已经被定义过，若已被定义则报出错误类型 4，否则加入函数表。

Stmt():

1. Exp SEMI

调用函数 Exp()。

2. CompSt

调用函数 CompSt()。

3. RETURN Exp SEMI

调用函数 Exp()，再将返回值和 type 值进行比较，若不同则报出错误类型 8，return 的返回类型和函数定义类型不匹配。

4. IF 或 WHILE

调用函数 Exp()，若不是 int 类型，则报出错误类型 7，即操作数类型不匹配；否则调用函数 Stmt()。

Exp():

1. INT 或 FLOAT

可以直接生成对应类型 type。

2. ID

首先在符号表中查找是否已存在，若不存在则报出错误类型 1，即变量未经定义就被使用；否则返回 FieldList()中的 type。

3. MINUS Exp 或 LP Exp RP

调用函数 Exp()。

4. NOT Exp 或 AND 或 OR

调用函数 Exp()，若不为 int 类型，则对应报出错误类型 7，即操作数类型不匹配。

5. ASSIGNOP

判断左部是否满足左值的产生式，若不满足则对应报出错误类型 6，即赋值号左边出现一个只有右值的

表达式；若左右 Exp 类型不同则对应报出错误类型 5，即赋值号两边表达式类型不匹配。

6. RELOP or PLUS or MINUS or STAR or DIV

若左右表达式的类型不同，则对应报出错误类型 7。

7. Exp DOT ID

首先判断左边表达式是否为结构体 STRUCTURE 类型，若不是则对应报出错误类型 13，即对非结构体类型变量使用“.”操作符；其次再判断右边的 ID 是否被定义过，若未经定义则对应报出错误类型 14，即访问了结构体中未被定义的域。

8. Exp LB Exp RB

首先判断左边表达式是否为数组 ARRAY 类型，若不是则对应报出错误类型 10，即对非数组型变量使用了“[...]”；其次再判断右边表达式是否为 int 类型，若不满足则对应报出错误类型 12，即数组访问操作符“[...]”中出现了非整数。

9. ID LP Args RP 或 ID LP RP

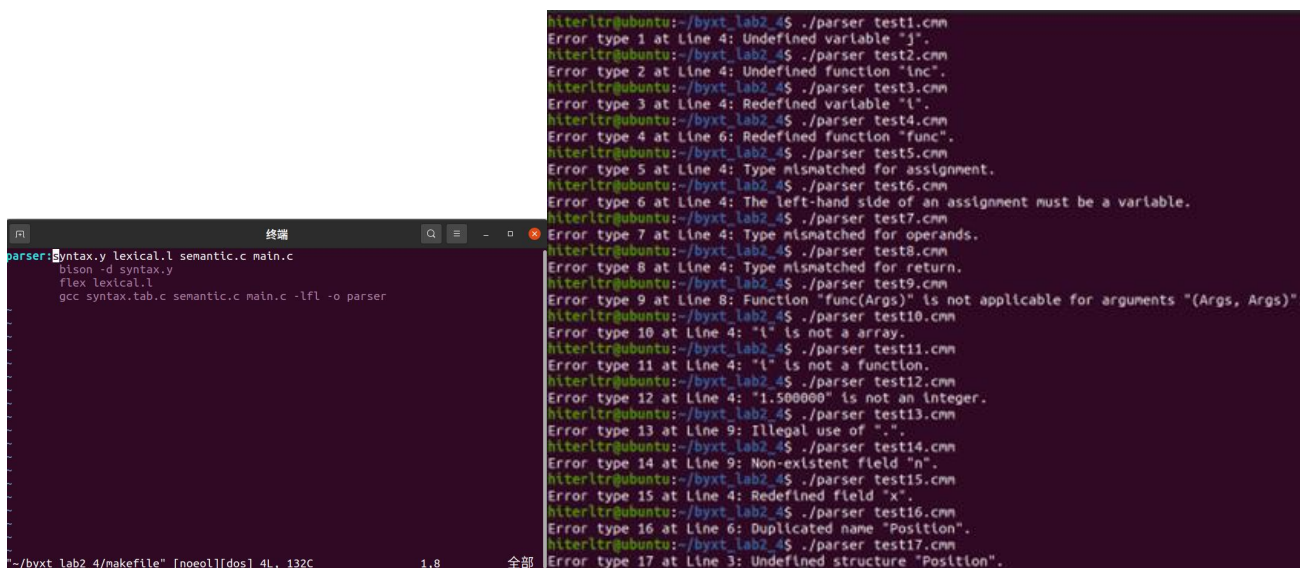
首先判断左边的 ID 是否在符号表中，若已存在则对应报出错误类型 11，即对普通变量使用了“(...)”或“()”操作符；其次再判断左边的 ID 是否在 FUNCTION 表中，若不在则对应报出错误类型 2，即函数在调用时未经定义；最后调用函数 Args()来判断函数参数是否符合规范，若不满足则对应报出错误类型 9，即函数调用时实参与形参的数目或类型不匹配。

Args:

调用 FieldList->type = Exp(), 并将相连的 Args 产生的 FieldList 通过链表结构进行连接。

(三)编译过程

本次实验采用了 Makefile 文件进行编译操作，其所包含的具体命令行内容如左下图所示：



```
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test1.cmn
Error type 1 at Line 4: Undefined variable "j".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test2.cmn
Error type 2 at Line 4: Undefined function "inc".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test3.cmn
Error type 3 at Line 4: Redefined variable "l".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test4.cmn
Error type 4 at Line 6: Redefined function "func".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test5.cmn
Error type 5 at Line 4: Type mismatched for assignment.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test6.cmn
Error type 6 at Line 4: The left-hand side of an assignment must be a variable.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test7.cmn
Error type 7 at Line 4: Type mismatched for operands.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test8.cmn
Error type 8 at Line 4: Type mismatched for return.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test9.cmn
Error type 9 at Line 8: Function "func(Args)" is not applicable for arguments "(Args, Args)".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test10.cmn
Error type 10 at Line 4: "l" is not an array.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test11.cmn
Error type 11 at Line 4: "l" is not a function.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test12.cmn
Error type 12 at Line 4: "1.500000" is not an integer.
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test13.cmn
Error type 13 at Line 9: Illegal use of ".".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test14.cmn
Error type 14 at Line 9: Non-existent field "n".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test15.cmn
Error type 15 at Line 4: Redefined field "x".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test16.cmn
Error type 16 at Line 6: Duplicated name "Position".
hiterl@ubuntu:~/byxt_lab2_4$ ./parser test17.cmn
Error type 17 at Line 3: Undefined structure "Position".
```

(四)实验结果

这里我就仅仅展示一下必做内容的 17 个以样例为模板的标准分析结果，如右上图所示：

三、实验总结与收获

答：通过本次实验，我主要学会了如何实现简单的语义分析任务。首先需要明确的是本实验在实验一所构建的语法树的基础上完成，并且实验二仍然需要对语法树进行遍历从而进行符号表的相关操作以及类型的构造与检查。但是与实验一所不同的是本次实验并不再借助已有现成的工具，所有的任务都必须手写代码来完成。除此之外虽然语义分析在整个编译器的实现中并不是难度最大的任务，但却是最细致、琐碎的任务。因此需要我用心地设计诸如符号表、变量类型等数据结构的实现细节，从而正确、高效地实现语义分析的各种功能。最后需要注意的是，由于在后面的实验中还会用到本次实验已经写好的代码，因此保持一个良好的代码风格、系统地设计代码结构和各模块之间的接口对于整个实验来讲相当重要。