哈尔滨工业大学

编译原理 2024 春

实验二

学院：	计算学部
姓名：	徐柯炎
学号：	2021110683
指导教师：	单丽莉

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、实验环境

- GNU Linux Release: Ubuntu 12.04, kernel version 3.2.0 – 29。
- GCC version 4.6.3。
- GNU Flex version 2.5.35。
- GNU Bison version 2.5。

三、实验内容

（一）实现功能

本程序实现了实验指导书上所有错误类型的检查，包括：

1) 变量（包括数组、指针、结构体）或过程未经声明就使用；
2) 变量（包括数组、指针、结构体）或过程名重复声明；
3) 运算分量类型不匹配；
4) 操作符与操作数之间的类型不匹配。

（二）数据结构

1. 语法树节点的数据结构：

实验二的语法树节点的数据结构沿用了实验一的结构体，包括以下几个变量：

1) 行号 int lineNo；
2) 节点类型 NodeType type；
3) 节点名称 char* name；
4) 节点值 char* val；
5) 下一个孩子节点 struct node* child；
6) 下一个兄弟节点 struct node* next；

2. 语义分析时符号表的数据结构：

实验二的语义分析符号表采用一张哈希表来实现，表中的每一项都是如右图所示的结构体，下面解释一下每一

```
typedef struct tableItem {
    int symbolDepth;     // 符号层次
    pFieldList field;    // 域
    pItem nextSymbol;  // 下一个符号表项
    pItem nextHash;     // 下一个哈希表项
} TableItem;     // 符号表项
```

项的作用：

1) int symbolDepth：符号层次，用来指明符号的作用域；
2) pFieldList field：域，用来存放符号的名字以及类型；
3) pItem nextSymbol：下一个符号表项，层次相同时下一个符号项；
4) pItem nextHash：下一个哈希表项，当哈希值一样的时候下一个符号项。

而在本实验中，每个域存放了符号的名字以及类型，而类型分为 4 种，也就是基础类型（int，float），结构体，函数，数组，分别用一个结构体表示并存放其类型和数值。

（三）语义分析

本实验语义分析的大体过程如下：

1) 首先进行语法分析和词法分析，构建语法分析树；
2) 有了语法分析树后，从根节点开始向下逐级分析，在必要的地方对错误类型进行检查分析。

以错误类型 1：变量在使用时未经定义为例：

```
// Exp -> ID
else if (!strcmp(t->name, "ID")) {
    pItem tp = searchTableItem(table, t->val);
    if (tp == NULL || isStructDef(tp)) {
        char msg[100] = {0};
        sprintf(msg, "Undefined variable \"%s\".", t->val);
        pError(UNDEF_VAR, t->lineNo, msg);
        return NULL;
    } else {
        return copyType(tp->field->type);
    }
```

当语义分析到 exp 节点时，如果子节点为 ID，也就是应当定义过的变量，这时去符号表中查找是否存在该变量，如果在符号表中没有查找到该符号，说明变量没有经过定义就使用了，于是报错，在 command 上打印”undefined variable...”，分析结束。

最后附上所有错误类型需要检查的文法：

```
UNDEF_VAR = 1,          // Undefined Variable, Exp -> ID
UNDEF_FUNC,             // Undefined Function, Exp -> ID LP Args RP | ID LP RP
REDEF_VAR,              // Redefined Variable, ExtDecList -> VarDec, ParamDec -> Specifier VarDec, Dec -> VarDe
REDEF_FUNC,             // Redefined Function, FunDec -> ID LP VarList RP
TYPE_MISMATCH_ASSIGN,   // Type mismatched for assignment, Dec -> VarDec ASSIGNOP Exp, Exp -> Exp ASSIGNOP Exp
LEFT_VAR_ASSIGN,  // The left-hand side of an assignment must be a variable, Exp -> Exp ASSIGNOP Exp
TYPE_MISMATCH_OP,       // Type mismatched for operands, Exp -> Exp AND Exp
TYPE_MISMATCH_RETURN,   // Type mismatched for return, Stmt -> RETURN Exp SEMI
FUNC_AGRC_MISMATCH,     // Function is not applicable for arguments, Exp -> ID LP RP, Args -> Exp COMMA Args
NOT_A_ARRAY,            // Variable is not a Array, Exp -> Exp LB Exp RB
NOT_A_FUNC,             // Variable is not a Function, Exp -> ID LP Args RP
NOT_A_INT,              // Variable is not a Integer, Exp -> Exp LB Exp RB
ILLEGAL_USE_DOT,        // Illegal use of ".", Exp -> Exp DOT ID
NONEXISTFIELD,          // Non-existentfield, Exp -> Exp DOT ID
REDEF_FEILD,            // Redefined field, Dec -> VarDec | VarDec ASSIGNOP Exp
DUPLICATED_NAME,        // Duplicated name, StructSpecifier -> STRUCT OptTag LC DefList RC
UNDEF_STRUCT            // Undefined structure, StructSpecifier->STRUCT Tag
```

（四）编译过程和测试过程

本程序采用 makefile 编译，并采用 bash 脚本进行测试样例的测试，部分截图如下：

Makefile：

```
parser: syntax $(filter-out $(LFO),$(OBJS))
    $(CC) -o parser $(filter-out $(LFO),$(OBJS)) -lfl

syntax: lexical syntax-c
    $(CC) -c $(YFC) -o $(YFO)

lexical: $(LFILE)
    $(FLEX) -o $(LFC) $(LFILE)

syntax-c: $(YFILE)
    $(BISON) -o $(YFC) -d -v $(YFILE)
```

test.sh

```
testpath=./test
parserpath=./parser

testfiles=$(find $testpath -name "*.cmm")
find $testpath -name "*.c"

for file in $testfiles
do
    echo ========================= Testing $file =========================
    $parserpath $file
    echo
done
```

四、实验结果

部分测试结果如下：

```
vm@vm-virtual-machine:/mnt/hgfs/Compile/lab2/pre$ bash test.sh
========================= Testing ./test/test01.cmm =========================
Error type 1 at Line 4: Undefined variable "j".

========================= Testing ./test/test02.cmm =========================
Error type 2 at Line 4: Undefined function "inc".

========================= Testing ./test/test03.cmm =========================
Error type 3 at Line 4: Redefined variable "i".

========================= Testing ./test/test04.cmm =========================
Error type 4 at Line 6: Redefined function "func".

========================= Testing ./test/test05.cmm =========================
Error type 5 at Line 4: Type mismatched for assignment.

========================= Testing ./test/test06.cmm =========================
Error type 6 at Line 4: The left-hand side of an assignment must be avariable.
```

五、实验总结

1) 学习了如何在词法分析和语法分析的基础上更进一步进行语义分析；
2) 增强了编程能力、分析问题的能力和动手能力；
3) 对语义分析有了更深层次的认识。