上海大学

SHANGHAI UNIVERSITY

<编译原理>实验报告

学　　院　计算机工程与科学学院

组　　号　8

实 验 题 号　五

日　　期　2022 年 5 月 27 日

学号	姓名	主要工作	贡献因子
16121803	许睿	代码调试	20%
19120188	孙瑶	代码优化	20%
19120191	汪雨卿	代码测试	20%
19121442	曹卓文	代码编写	20%
19122408	严邹莹	报告撰写	20%

实验五 中间代码生成

一、实验目的与要求

1、通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。

2、 掌握目前普遍采用的语义分析方法——语法制导翻译技术。

3、给出 PL/0 文法规范，要求在语法分析程序中添加语义处理，对于语法正确的算术表达式，输出其中间代码。

二、实验环境

c/c++、visual studio

三、实验内容

已给 PL/0 语言文法，在实验三的表达式语法分析程序里，添加语义处理部分输出表达式的中间代码，用四元式序列表示。

四、实验内容的设计与实现

4.1 实验设计

列出本次实验 TAC 的翻译模式如下：

* SLR(1):

S -> E　　{print(E.val)}

E -> E'PT　　{E.val:=P.val(E'.val,T.val)}

E -> PT　　{E.val:=P.val(T.val)}

E -> T　　{E.val:=T.val}

T -> T'MF　　{T.val:=M.val(T'.val,F.val)}

T -> F　　{T.val:=F.val}

F -> (S)　　{F.val:=S.val}

F -> n {F.val:=n.lexval}

F -> i {F.val:=i.lexval}

P -> + {P.val:=+.lexval}

P -> - {P.val:=-.lexval}

M -> * {M.val:=*.lexval}

M -> / {M.val:=/.lexval}

TAC 在数字规约时产生，使用 buffer 暂存分析栈的栈顶计算值，并记录 buffer 中的元素个数 sz。由于在 TAC 中，运算符可以表示任意二元以下运算或操作，因此对 buffer 内元素个数 sz 进行分情况讨论（基于之前的实验与文法）：

1. buffer 仅有单个元素且为数字，输出中间代码

（1）该元素为数字，则输出中间代码

（2）非数字，则直接返回该元素

2. buffer 有 2 个元素，输出中间代码。

3. buffer 有 3 个元素，分类讨论

（1）buffer 内元素含有括号，则直接返回中间值

（2）buffer 内为 3 元计算式，输出中间代码。

4.2 主要代码实现

1、buffer 分类讨论

```
static std::pair<TResult, std::string> buffer[3];     // 存放中间状态
static int sz;
sz = 0;
for (auto&& r : reduce)
{
    assert(r == std::get<1>(s.top())); // 判断弹出的 element 是否和规约式一致
```

```
            buffer[sz++] = std::get<2>(s.top());       // 栈顶的计算值压入 buffer 暂存
            s.pop();       // 出栈
        }
        std::pair<TResult, std::string> res;
        if (sz == 1) {
        auto temp = buffer[0];
        res = buffer[0];       // buffer 仅有单个元素，分类讨论。
        if(reduce[0]==number){
        buffer[0].second="t"+std::to_string(++counter);
        res = buffer[0];       //为数字时，输出中间代码
        std::cout << "(:=" << ',';
        __printitem(temp);
        std::cout << ", ";
        std::cout << ",t"+std::to_string(counter);
        std::cout << ')' << std::endl;}}
        else if (sz == 2)
        {     // buffer 有 2 个元素。
           res = std::make_pair(calc(buffer[1].first, buffer[0].first), "t" + std::to_string(++counter));
           print(static_cast<Element>(buffer[1].first), buffer[0], res);}
        else if (sz == 3)
        {     // buffer 有 3 个元素，分类讨论。
        if (buffer[2].second == "(" && buffer[0].second == ")") //  含括号情况，直接返回中间值。
        res = buffer[1];
        else
        {     // 3 元计算式，输出中间代码。
                              res = std::make_pair(calc(buffer[2].first, buffer[1].first, buffer[0].first),
"t" + std::to_string(++counter));
            print(static_cast<Element>(buffer[1].first), buffer[2], buffer[0], res);       // 输出四元组
        }
      }
```

2、TAC 的输出

```
void ExpressionCalculator::print(Element op,
                                 std::pair<TResult, std::string> lhs,
                                 std::pair<TResult, std::string> rhs,
                                 std::pair<TResult, std::string> res)
{
    std::cout << '(' << op << ',';
    __printitem(lhs);
    std::cout << ',';
    __printitem(rhs);
    std::cout << ',';
    __printitem(res);
    std::cout << ')' << std::endl;
}

void ExpressionCalculator::print(Element op,
                                 std::pair<TResult, std::string> rhs,
                                 std::pair<TResult, std::string> res)
{
    std::cout << '(' << op << ',';
    __printitem(rhs);
    std::cout<<", ";
    std::cout << ',';
    __printitem(res);
    std::cout << ')' << std::endl;
}
```

4.3 实验结果

左图为测试数据，右图为输出结果。

测试用例 1-3：

```
(1+2)/3;
1+1;
2+3*4*4/5;
(2+3)*4/5;
(2+3);
((2)+3);
-((-2)+3);
-4+(5*(-2)/1);
-(4+5*(-2)/1)*(10/2);
-(4+5*(-2)/1)+3/5;
```

```
++++++ intermediate code and results ++++
==== Line1 ====
OK
(:=,1, ,t1)
(:=,2, ,t2)
(+,t1,t2,t3)
(:=,3, ,t4)
(/,t3,t4,t5)
1
==== Line2 ====
OK
(:=,1, ,t1)
(:=,1, ,t2)
(+,t1,t2,t3)
2
==== Line3 ====
OK
(:=,2, ,t1)
(:=,3, ,t2)
(:=,4, ,t3)
(*,t2,t3,t4)
(:=,4, ,t5)
(*,t4,t5,t6)
(:=,5, ,t7)
(/,t6,t7,t8)
(+,t1,t8,t9)
11
```

测试用例 4-7：

```
(1+2)/3;
1+1;
2+3*4*4/5;
(2+3)*4/5;
(2+3);
((2)+3);
-((-2)+3);
-4+(5*(-2)/1);
-(4+5*(-2)/1)*(10/2);
-(4+5*(-2)/1)+3/5;
```

```
==== Line4 ====
OK
(:=,2, ,t1)
(:=,3, ,t2)
(+,t1,t2,t3)
(:=,4, ,t4)
(*,t3,t4,t5)
(:=,5, ,t6)
(/,t5,t6,t7)
4
==== Line5 ====
OK
(:=,2, ,t1)
(:=,3, ,t2)
(+,t1,t2,t3)
5
==== Line6 ====
OK
(:=,2, ,t1)
(:=,3, ,t2)
(+,t1,t2,t3)
5
==== Line7 ====
OK
(:=,2, ,t1)
(-,t1, ,t2)
(:=,3, ,t3)
(+,t2,t3,t4)
(-,t4, ,t5)
-1
```

测试用例 8-9：

```
(1+2)/3;
1+1;
2+3*4*4/5;
(2+3)*4/5;
(2+3);
((2)+3);
-((-2)+3);
-4+(5*(-2)/1);
-(4+5*(-2)/1)*(10/2);
-(4+5*(-2)/1)+3/5;
```

```
==== Line8 ====
OK
(:=,4, ,t1)
(-,t1, ,t2)
(:=,5, ,t3)
(:=,2, ,t4)
(-,t4, ,t5)
(*,t3,t5,t6)
(:=,1, ,t7)
(/,t6,t7,t8)
(+,t2,t8,t9)
-14
==== Line9 ====
OK
(:=,4, ,t1)
(:=,5, ,t2)
(:=,2, ,t3)
(-,t3, ,t4)
(*,t2,t4,t5)
(:=,1, ,t6)
(/,t5,t6,t7)
(+,t1,t7,t8)
(:=,10, ,t9)
(:=,2, ,t10)
(/,t9,t10,t11)
(*,t8,t11,t12)
(-,t12, ,t13)
30
```

测试用例 10：

```
(1+2)/3;
1+1;
2+3*4*4/5;
(2+3)*4/5;
(2+3);
((2)+3);
-((-2)+3);
-4+(5*(-2)/1);
-(4+5*(-2)/1)*(10/2);
-(4+5*(-2)/1)+3/5;
```

```
==== Line10 ====
OK
(:=,4, ,t1)
(:=,5, ,t2)
(:=,2, ,t3)
(-,t3, ,t4)
(*,t2,t4,t5)
(:=,1, ,t6)
(/,t5,t6,t7)
(+,t1,t7,t8)
(-,t8, ,t9)
(:=,3, ,t10)
(:=,5, ,t11)
(/,t10,t11,t12)
(+,t9,t12,t13)
6
```

对包含 10 个测试用例的数据集进行了测试，达到实验要求。

五、收获与体会

通过中间代码生成的实验，小组成员们更深刻得体会了语义分析的意义，中间代码比源语言更加接近于目标语言，关系到整个程序的正确编译，更是进行下一步编译的重要条件。

通过本学期的实验，对词法分析到语法分析到语义分析的知识点有了比较完整的回顾以及串联，对于每个阶段输入什么，输出什么，信息怎么储存，用什么算法计算有了更直观的了解，对这些概念形成了理解记忆。在优化与调试代码的过程中，对自己先前的专业知识同样进行了巩固与提高，明白了要更加注重对于基础科目的掌握，感谢腾中梅老师的耐心验收与悉心指导。