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　四

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%

实验四 语义分析

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，输出其计算值。

二、实验环境

c/c++、visual studio

三、实验内容

已给 PL/0 语言文法，在表达式的语法分析程序里，添加语义处理部分。

四、实验内容的设计与实现

4.1 实验设计

在语义分析中，使用 BNF 如下：

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’

<加法运算符> ::= +|-

<乘法运算符> ::= *|/

每个方法按照 BNF 定义进行匹配

使用的 SLR(1)文法构成 SLR 自动机，文法如下：

S -> E

E -> E'PT

E -> PT

E -> T

T -> T'MF

T -> F

F -> (S)

F -> n

F -> i

P -> +

P -> -

M -> *

M -> /

基于此文法生成的 SLR(1)分析表为：

	Action									**GoTo**				
状态	**(**	**)**	**+**	**-**	*****	**/**	**i**	**n**	**#**	**E**	**T**	**F**	**P**	**M**
0	**S1**		**S2**	**S3**			**S8**	**S9**		**4**	**7**	**5**	**6**	
1	**S1**		**S2**	**S3**			**S8**	**S9**		**10**	**7**	**5**	**6**	
2	**R9**						**R9**	**R9**						
3	**R10**						**R10**	**R10**						
4			**S2**	**S3**					**acc**				**11**	
5		**R5**	**R5**	**R5**	**R5**	**R5**			**R5**					
6	**S1**						**S8**	**S9**			**12**	**5**		
7		**R3**	**R3**	**R3**	**S13**	**S14**			**R3**					**15**
8		**R7**	**R7**	**R7**	**R7**	**R7**			**R7**					
9		**R8**	**R8**	**R8**	**R8**	**R8**			**R8**					
10		**S16**	**S2**	**S3**									**11**	
11	**S1**						**S8**	**S9**			**17**	**5**		
12		**R2**	**R2**	**R2**	**S13**	**S14**			**R2**					**15**
13	**R11**						**R11**	**R11**						
14	**R12**						**R12**	**R12**						
15	**S1**						**S8**	**S9**				**18**		
16		**R6**	**R6**	**R6**	**R6**	**R6**			**R6**					
17		**R1**	**R1**	**R1**	**S13**	**S14**			**R1**					**15**
18		**R4**	**R4**	**R4**	**R4**	**R4**			**R4**					

4.2 主要代码实现

1、SLR(1)分析表（仅截取状态 0 1 作展示）

```
const std::vector<std::map<Element, std::pair<Action, int>>> ExpressionCalculator::f = {
        // 0
        {
                {lparen, {Shift, 1}},
                {plus_, {Shift, 2}},
                {minus_, {Shift, 3}},
                {ident, {Shift, 8}},
                {number, {Shift, 9}},
                {Expr, {Goto, 4}},
                {Term, {Goto, 7}},
                {Factor, {Goto, 5}},
                {Plus, {Goto, 6}}},
        // 1
        {
                {lparen, {Shift, 1}},
                {plus_, {Shift, 2}},
                {minus_, {Shift, 3}},
                {ident, {Shift, 8}},
                {number, {Shift, 9}},
                {Expr, {Goto, 10}},
                {Term, {Goto, 7}},
                {Factor, {Goto, 5}},
                {Plus, {Goto, 6}}},
```

2、分析栈的实现

```
std::pair<Action, int> ExpressionCalculator::getNext(int current, Element type)
{    // 通过栈顶数据和读入字符，根据 SLR(1)分析表，判断下一步 Action
    auto it = f[current].find(type);
    if (it == f[current].end()) throw ExpressionException(type);
    return it->second;
}
TResult ExpressionCalculator::getLexval(Element obj, std::string token)
{
    if (obj == number) return std::stoi(token);
    if (obj < number) return static_cast<TResult>(obj);
```

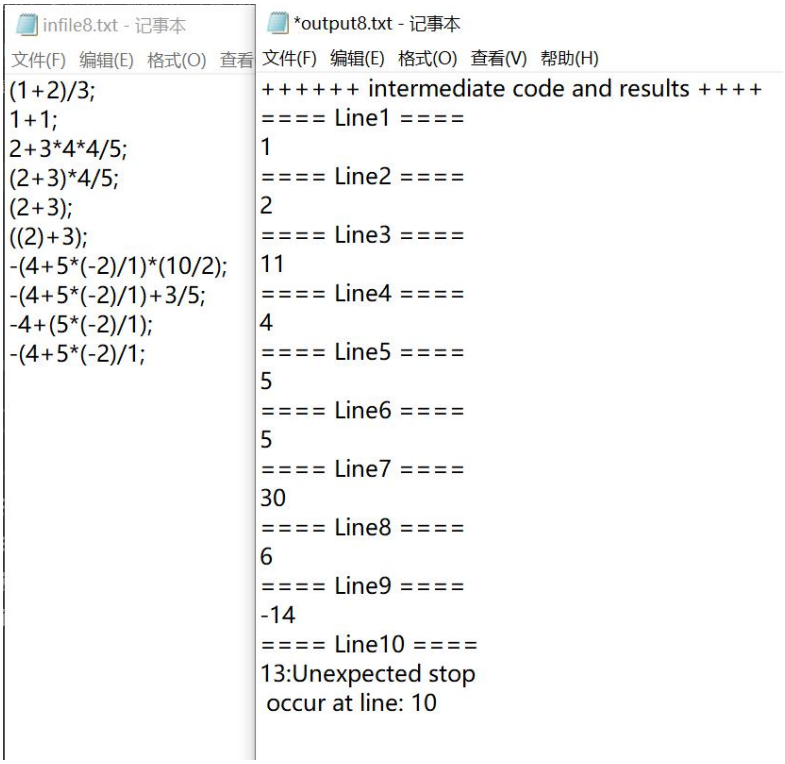
```
    return 0;
}
TResult ExpressionCalculator::calculate(const std::vector<std::pair<Element, std::string>>& elements) const
{
截取分析计算部分
for (auto&& [element, token] : elements)       // 对读入行进行分析计算
    {
        if(element == error){
            std::cout << "illegal operator, please rewrite your testing files" << std::endl;
        }
        for (;;)
        {
            auto action = getNext(std::get<0>(s.top()), element);
            // 根据获取的 Action 类型，进行下一步分析。
            if (action.first == Shift)
            {    // 移进,(状态，符号，(计算值，token))入栈
                s.push(std::make_tuple(action.second, element,
                    std::make_pair(getLexval(element, token), token)));
                break;
            }
            else if (action.first == Reduce)
            {    // 规约
                auto it = getReduce(action.second);     //获取规约文法
                auto reduce = it.second;       // 从右往左，出栈
                std::reverse(reduce.begin(), reduce.end());
                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，直接返回该元素。
                    if(reduce[0]==number){
                        buffer[0].second="t"+std::to_string(++counter);
                        res = buffer[0];
                    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，计算结果，并输出中间代码。
            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，计算结果，并输出中间代码。
                    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);
```

```
                }
            }
            else //:(
                assert(false);
            auto go = getNext(std::get<0>(s.top()), it.first);   // 获取下一步 Action
            assert(go.first == Goto);     // Action 仅能为 Goto!
            s.push(std::make_tuple(go.second, it.first, res));   // 状态入栈，继续分析。
        }
        else if (action.first == Accept)
            return std::get<2>(s.top()).first;
        else // :(
            assert(false);
    }
}
```

4.3 实验结果

左图为测试数据，右图为输出结果。



对包含 10 个测试用例的数据集进行了测试，其中样例 10 表达式缺少“)”，抛出异常，当除数为 0 时，同样抛出异常。表达式语义值计算正确，达到实验要求。

五、收获与体会

在进行语义分析的实验过程中，通过语法制导翻译技术，使用了 SLR(1)方式对于表达式进行了语义处理。在实验的过程中，小组成员们依从分析过程，根据每个产生式添加的语义动作进行翻译，当语义分析完成时，组员们对编译程序工作的基本过程及其各阶段的基本任务，编译程序的生成过程有了更深的理解。在进行测试的过程中，发现问题—解决问题的循环切实提高了大家的能力，对课堂上学习的知识有了更深入的了解。