《编译原理与设计》

语法分析程序的设计与实现

实验报告

班级	2014211304
学号	2014210336
姓名	杨炫越
日期	2016.11.14

1. 实验内容

编写语法分析程序，实现对算术表达式的语法分析。要求所分析的算术表达式由如下的文法产生：

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow (E) \mid \mathrm{num}$

2. 实验要求

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。编写语法分析程序实现自底向上的分析，要求如下：

(1) 构造识别该文法所有活前缀的 DFA；

(2) 构造该文法的 LR 分析表；

(3) 编程实现算法 4.3，构造 LR 分析程序。

3. 开发环境

操作系统：Microsoft Windows 10.0.14393 (x64)

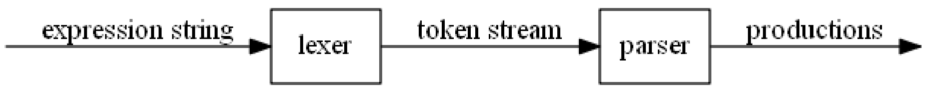
IDE：Microsoft Visual Studio Community 2015

编译器：MSVC++ 14.0

附加库：Boost 1.62.0

4. 设计思路

本语法分析程序首先**利用上次实验实现的词法分析程序**将输入串转化为 token 流，再在此基础上采用 SLR(1)分析方法对输入算术表达式进行分析，并输出分析过程采用的产生式。流程如下：



SLR(1)分析的关键在于构造预测分析表，然后使用分析表与一个分析栈进行联合控制，实现对输入符号串的自顶向上分析。构造预测分析表的前序工作依次如下（各部分算法细节详见 5. 程序实现）：

(1) 将文法拓广；

(2) 构造文法的 FIRST 集；

(3) 构造文法的 FOLLOW 集；

(4) 构造文法的 LR(0)项目集规范族及识别其所有活前缀的 DFA。

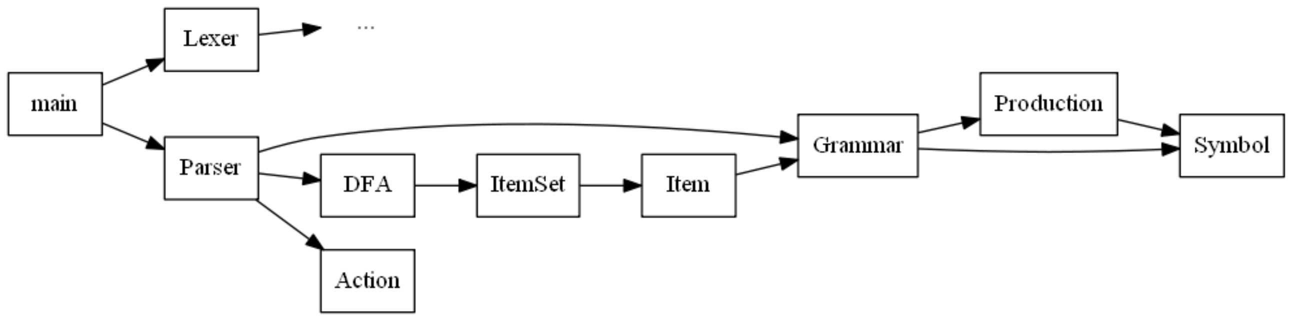
5. 程序实现

源码：
　　Local：/src_code
　　Online：https://github.com/YangXuanyue/Compiler

本语法分析程序采用了 C++来编写，程序中实现了一个`Parser`类作为语法分析器的对外接口，并为其重载了输入流操作符，可与上次实验实现的词法分析类`Lexer`连接如下：

```
Lexer lexer('\n'); //以换行符为输入串结尾
Parser parser;
cin >> lexer >> parser;
```

本程序的结构示意图如下所示：



其中 Lexer 分支为词法分析实验实现的词法分析器，Grammar、Production、Symbol 模块在 LL 语法分析实验报告中已有介绍，其余各模块实现简述如下：

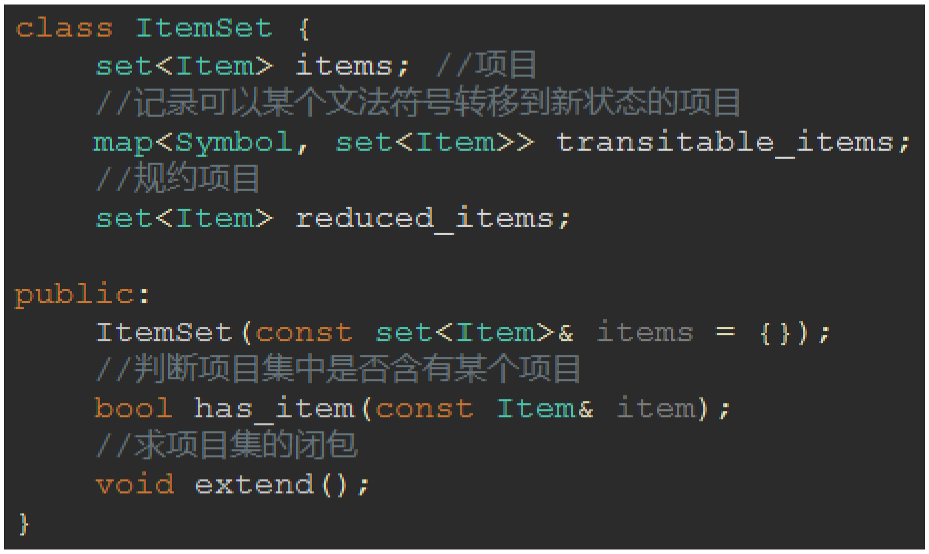
5.1. Item

该模块定义了 LR(0)项目的存储设计如下：

```
struct Item {
    int production_idx; //产生式编号
    int dot_pos; //圆点位置
}
```

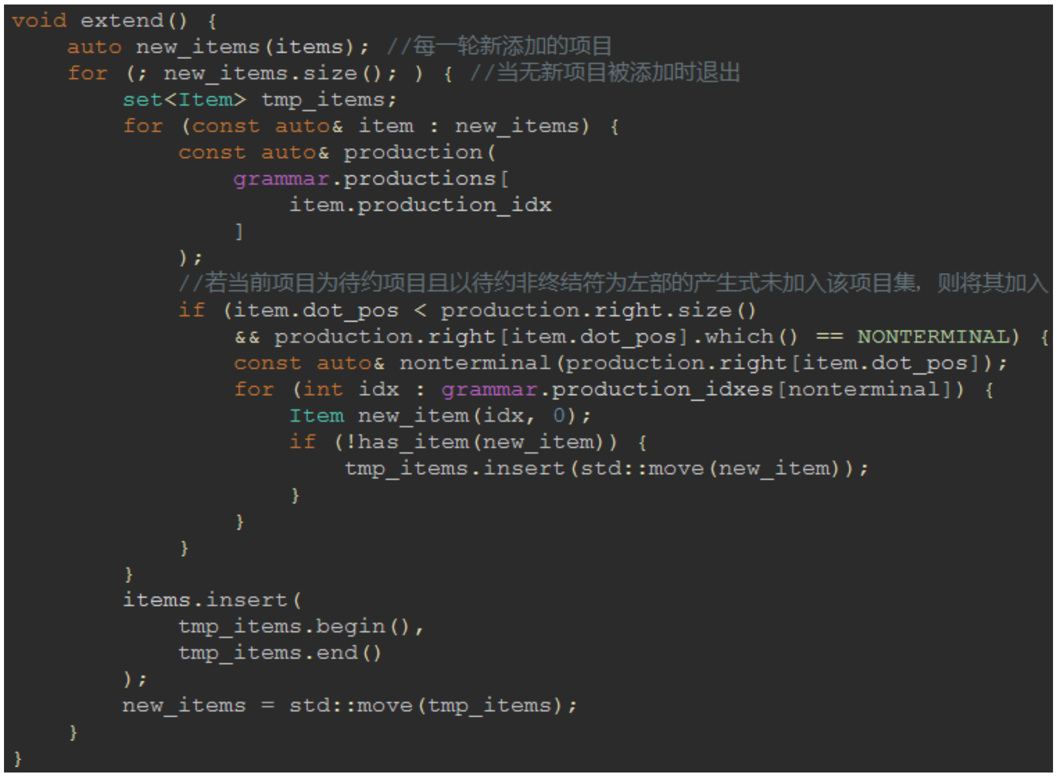
5.2. ItemSet

该模块定义了 LR(0)项目集类大致如下：



```
class ItemSet {
    set<Item> items; //项目
    //记录可以某个文法符号转移到新状态的项目
    map<Symbol, set<Item>> transitable_items;
    //规约项目
    set<Item> reduced_items;

public:
    ItemSet(const set<Item>& items = {});
    //判断项目集中是否含有某个项目
    bool has_item(const Item& item);
    //求项目集的闭包
    void extend();
}
```

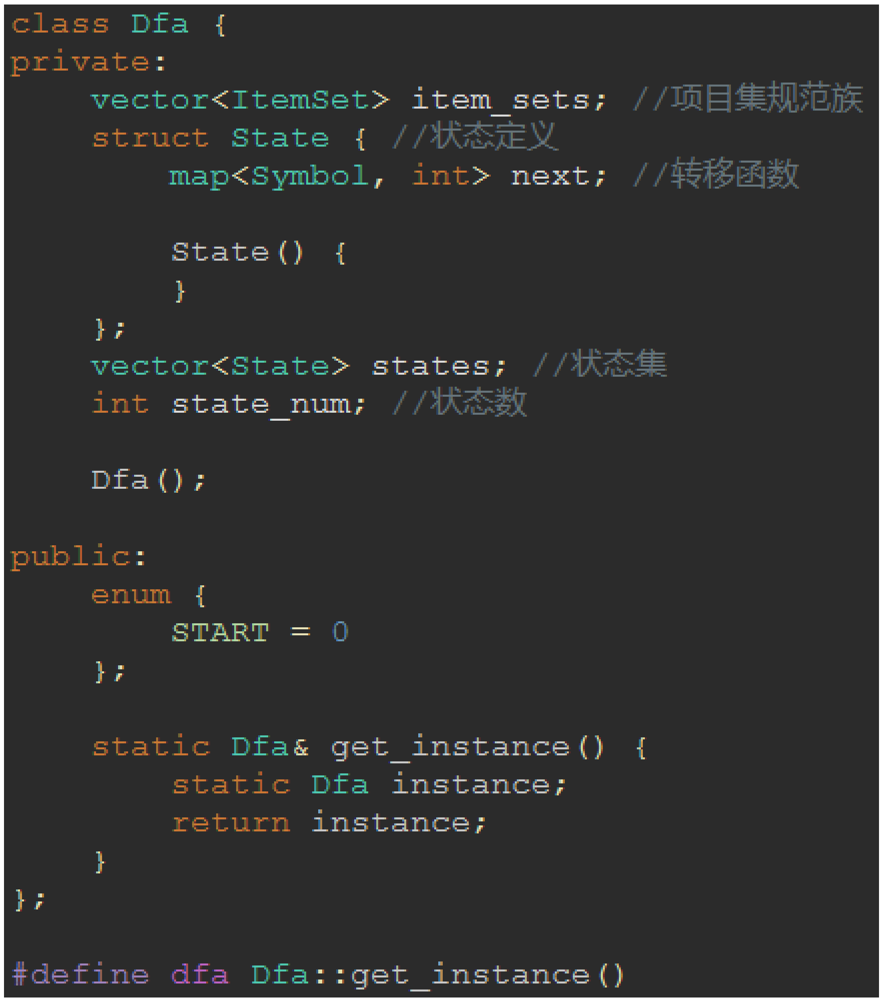
该类的构造函数接受初始项目集为参数，并将其拓展为其闭包。并预处理好一些信息如`transitable_items`、`reduced_items`以方便 DFA 的构造。



```
void extend() {
    auto new_items(items); //每一轮新添加的项目
    for (; new_items.size(); ) { //当无新项目被添加时退出
        set<Item> tmp_items;
        for (const auto& item : new_items) {
            const auto& production(
                grammar.productions[
                    item.production_idx
                ]
            );
            //若当前项目为待约项目且以待约非终结符为左部的产生式未加入该项目集，则将其加入
            if (item.dot_pos < production.right.size()
                && production.right[item.dot_pos].which() == NONTERMINAL) {
                const auto& nonterminal(production.right[item.dot_pos]);
                for (int idx : grammar.production_idxes[nonterminal]) {
                    Item new_item(idx, 0);
                    if (!has_item(new_item)) {
                        tmp_items.insert(std::move(new_item));
                    }
                }
            }
        }
        items.insert(
            tmp_items.begin(),
            tmp_items.end()
        );
        new_items = std::move(tmp_items);
    }
}
```

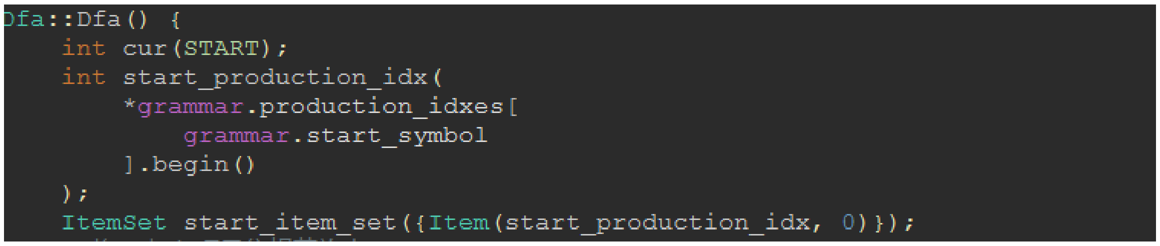
5.3. DFA

该模定义 DFA 类大致如下：

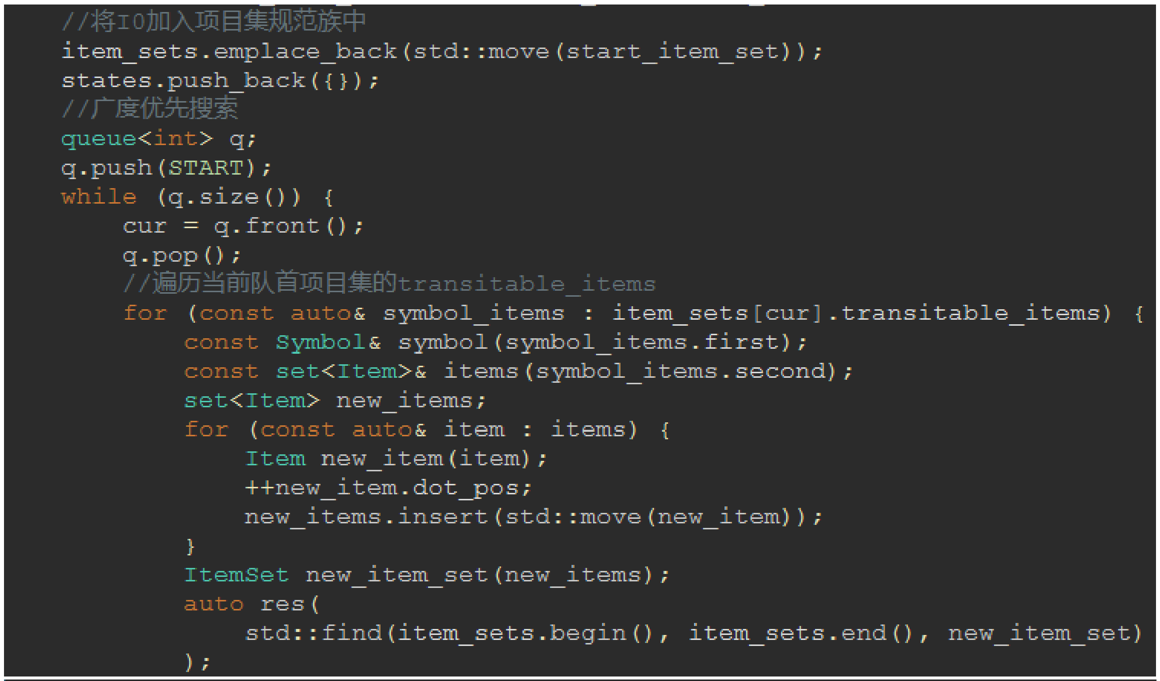


```
class Dfa {
private:
    vector<ItemSet> item_sets; //项目集规范族
    struct State { //状态定义
        map<Symbol, int> next; //转移函数

        State() {
        }
    };
    vector<State> states; //状态集
    int state_num; //状态数

    Dfa();

public:
    enum {
        START = 0
    };

    static Dfa& get_instance() {
        static Dfa instance;
        return instance;
    }
};

#define dfa Dfa::get_instance()
```

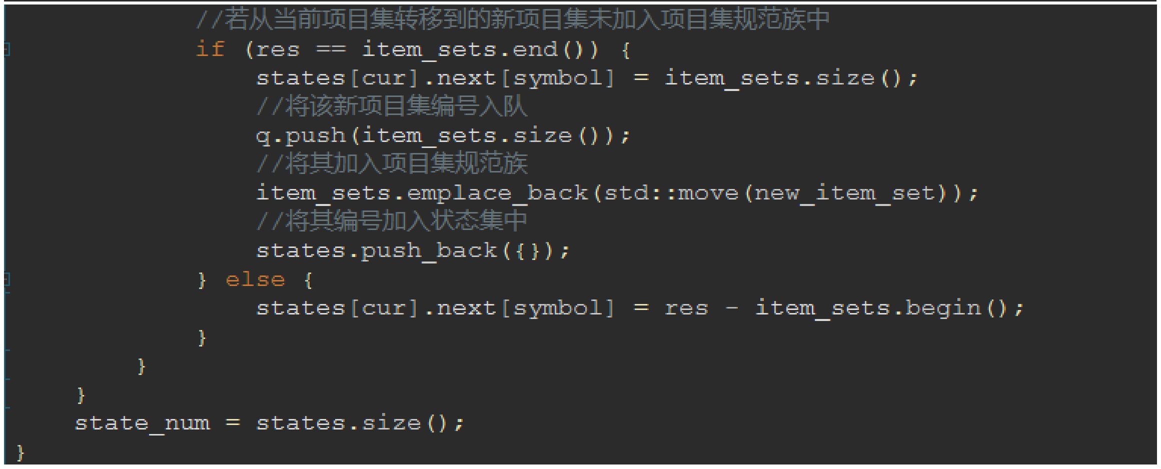
在其构造函数中实现了文法的 LR(0)项目集规范族及识别其所有活前缀的 DFA 的构造算法，主要思路是从起始项目集I_0开始进行**广度优先搜索**，将转移到的新项目集加入项目集规范组中，并将其编号加入到状态集中，实现如下：



```
Dfa::Dfa() {
    int cur(START);
    int start_production_idx(
        *grammar.production_idxes[
            grammar.start_symbol
        ].begin()
    );
    ItemSet start_item_set({Item(start_production_idx, 0)});
```



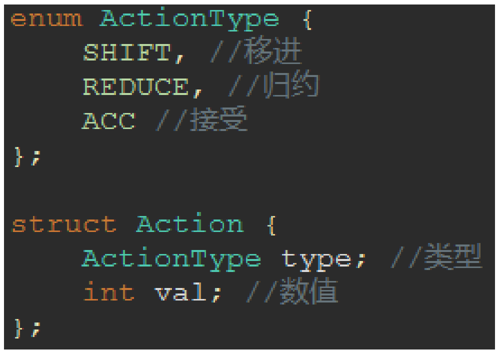
```
    //将I0加入项目集规范族中
    item_sets.emplace_back(std::move(start_item_set));
    states.push_back({});
    //广度优先搜索
    queue<int> q;
    q.push(START);
    while (q.size()) {
        cur = q.front();
        q.pop();
        //遍历当前队首项目集的transitable_items
        for (const auto& symbol_items : item_sets[cur].transitable_items) {
            const Symbol& symbol(symbol_items.first);
            const set<Item>& items(symbol_items.second);
            set<Item> new_items;
            for (const auto& item : items) {
                Item new_item(item);
                ++new_item.dot_pos;
                new_items.insert(std::move(new_item));
            }
            ItemSet new_item_set(new_items);
            auto res(
                std::find(item_sets.begin(), item_sets.end(), new_item_set)
            );
```



```
            //若从当前项目集转移到的新项目集未加入项目集规范族中
            if (res == item_sets.end()) {
                states[cur].next[symbol] = item_sets.size();
                //将该新项目集编号入队
                q.push(item_sets.size());
                //将其加入项目集规范族
                item_sets.emplace_back(std::move(new_item_set));
                //将其编号加入状态集中
                states.push_back({});
            } else {
                states[cur].next[symbol] = res - item_sets.begin();
            }
        }
    }
    state_num = states.size();
}
```

5.4. Action

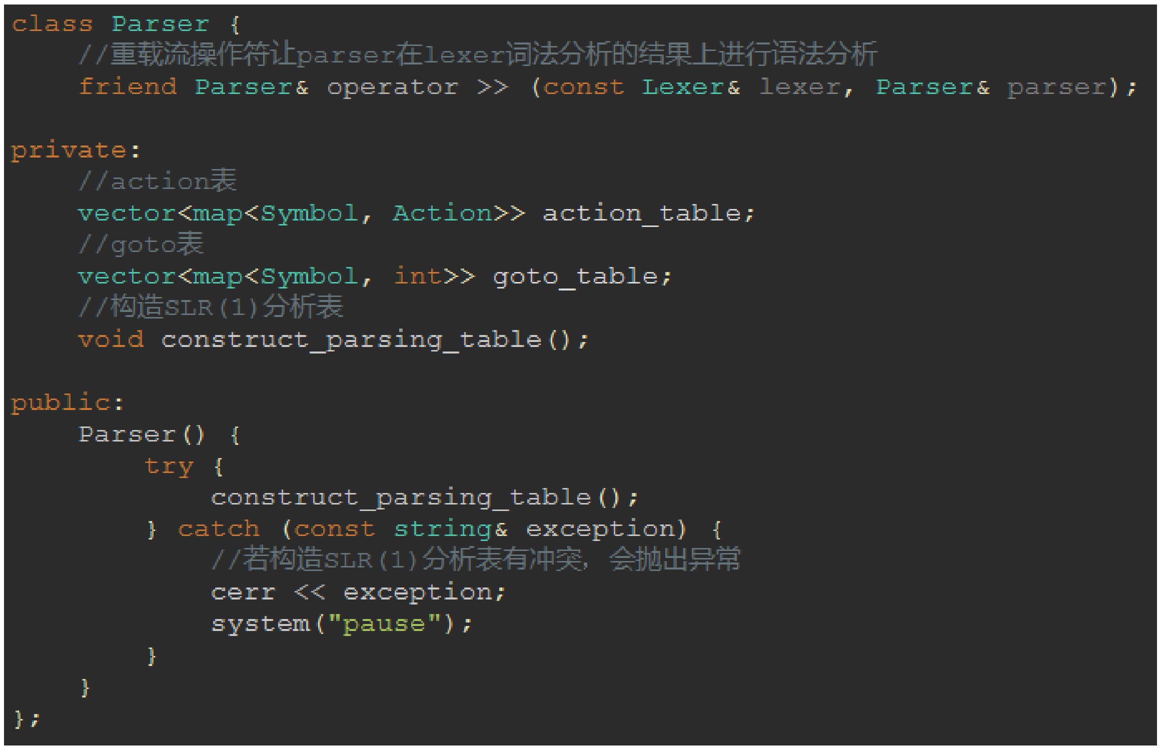
该模块定义了分析表中 action 表表项结构体如下：



```
enum ActionType {
    SHIFT, //移进
    REDUCE, //归约
    ACC //接受
};

struct Action {
    ActionType type; //类型
    int val; //数值
};
```

5.5. Parser

该模块定义了语法分析器类大致如下：



```
class Parser {
    //重载流操作符让parser在lexer词法分析的结果上进行语法分析
    friend Parser& operator >> (const Lexer& lexer, Parser& parser);

private:
    //action表
    vector<map<Symbol, Action>> action_table;
    //goto表
    vector<map<Symbol, int>> goto_table;
    //构造SLR(1)分析表
    void construct_parsing_table();

public:
    Parser() {
        try {
            construct_parsing_table();
        } catch (const string& exception) {
            //若构造SLR(1)分析表有冲突，会抛出异常
            cerr << exception;
            system("pause");
        }
    }
};
```

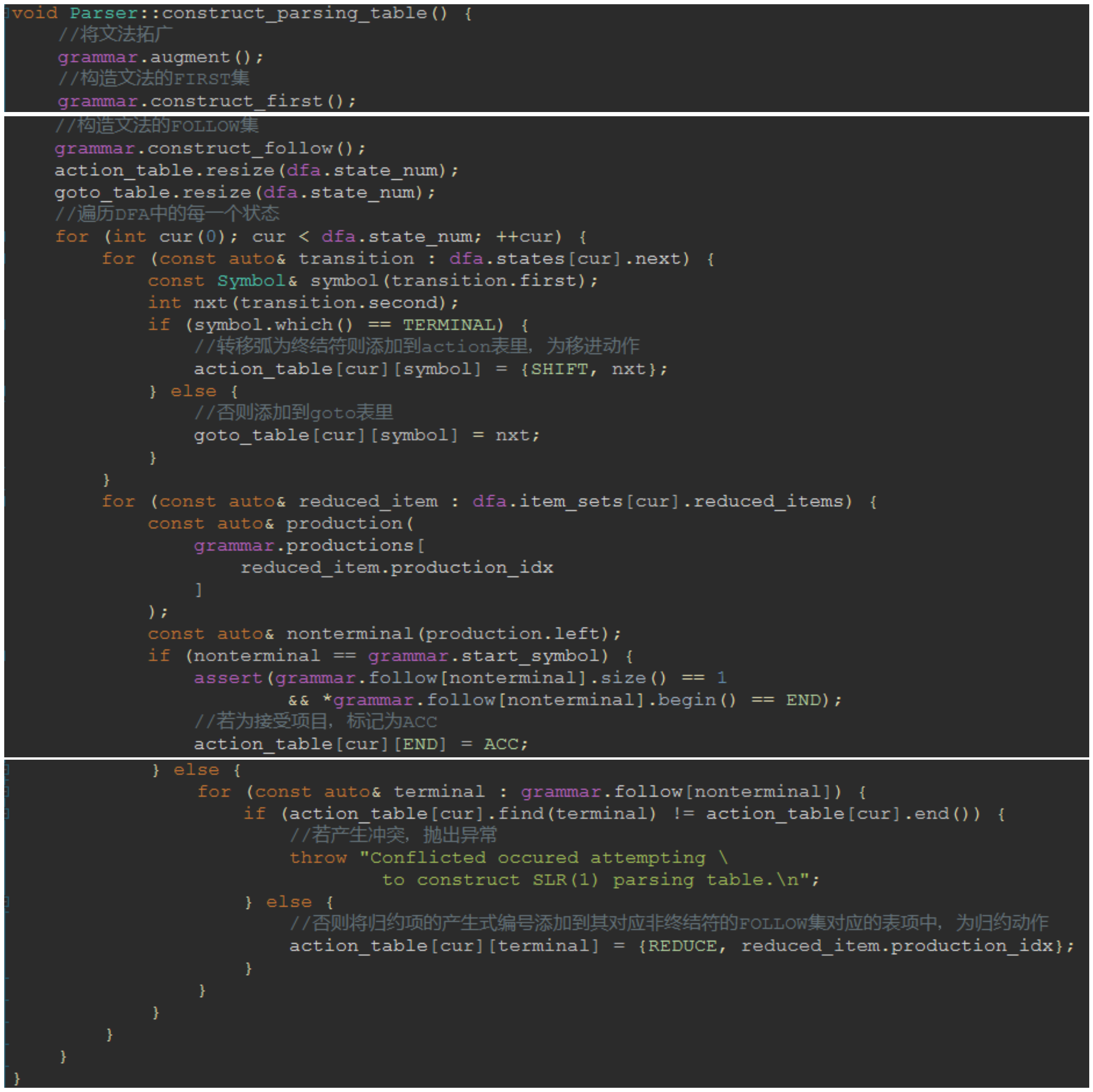
5.5.1. 构造预测分析表

本程序先以已构造得的文法的 LR(0)项目集规范族及识别其所有活前缀的 DFA 尝试构造 SLR(1)分析表。SLR(1)分析法对任一非终结符采用其 FOLLOW 集作为归约时的向前看符号集，该处理比较简单从而可能导致冲突表项，本程序在发现冲突表项时抛出异常。在执行过程中未发现异常，说明采用 SLR(1)分析法可分析该文法。

预测分析表分为 action 表与 goto 表：action 表指明某一状态下看到某一待输入 Token 时该移进该 Token 并转移到某个新状态还是该先按某个产生式归约；goto 表指明在某一状态相对某个非终结符的后继状态。

由于 LR 分析对错误处理没有确定解决方法，需人工为分析表添加特定错误信息与处理动作，而无法或难以通过程序自动完成（如程序无法判断该采用移进输入指针还是将一个虚拟符号压栈的方式较好），本程序不对输入表达式进行错误处理。

算法实现如下：

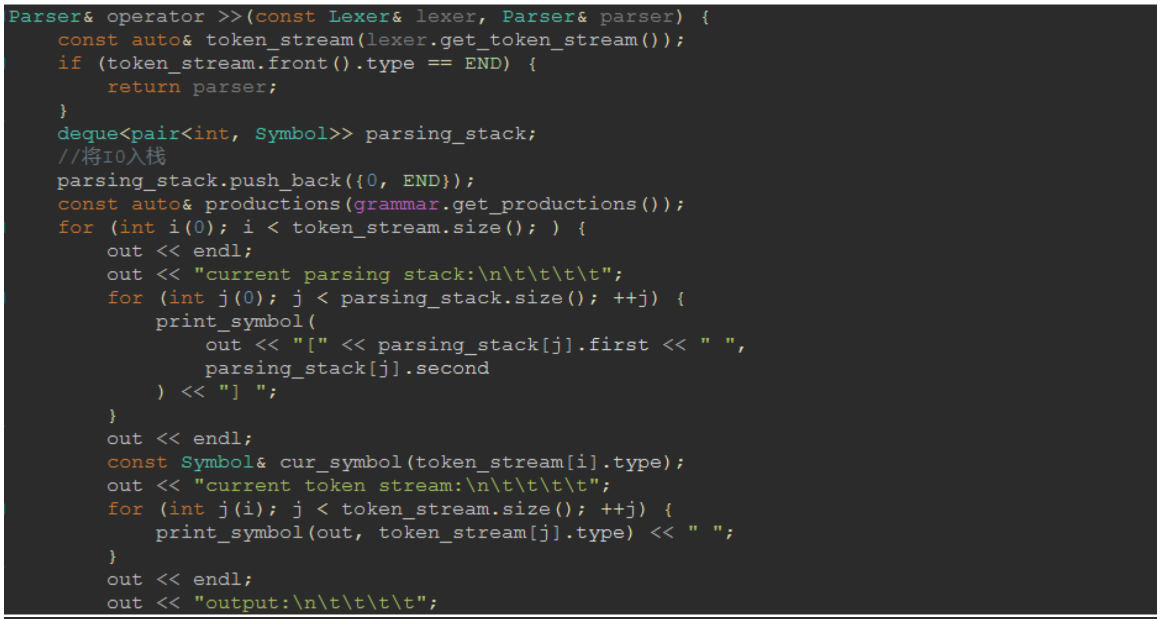


```
void Parser::construct_parsing_table() {
    //将文法拓广
    grammar.augment();
    //构造文法的FIRST集
    grammar.construct_first();
    //构造文法的FOLLOW集
    grammar.construct_follow();
    action_table.resize(dfa.state_num);
    goto_table.resize(dfa.state_num);
    //遍历DFA中的每一个状态
    for (int cur(0); cur < dfa.state_num; ++cur) {
        for (const auto& transition : dfa.states[cur].next) {
            const Symbol& symbol(transition.first);
            int nxt(transition.second);
            if (symbol.which() == TERMINAL) {
                //转移弧为终结符则添加到action表里，为移进动作
                action_table[cur][symbol] = {SHIFT, nxt};
            } else {
                //否则添加到goto表里
                goto_table[cur][symbol] = nxt;
            }
        }
        for (const auto& reduced_item : dfa.item_sets[cur].reduced_items) {
            const auto& production(
                grammar.productions[
                    reduced_item.production_idx
                ]
            );
            const auto& nonterminal(production.left);
            if (nonterminal == grammar.start_symbol) {
                assert(grammar.follow[nonterminal].size() == 1
                        && *grammar.follow[nonterminal].begin() == END);
                //若为接受项目，标记为ACC
                action_table[cur][END] = ACC;
            } else {
                for (const auto& terminal : grammar.follow[nonterminal]) {
                    if (action_table[cur].find(terminal) != action_table[cur].end()) {
                        //若产生冲突，抛出异常
                        throw "Conflicted occured attempting \
                            to construct SLR(1) parsing table.\n";
                    } else {
                        //否则将归约项的产生式编号添加到其对应非终结符的FOLLOW集对应的表项中，为归约动作
                        action_table[cur][terminal] = {REDUCE, reduced_item.production_idx};
                    }
                }
            }
        }
    }
}
```

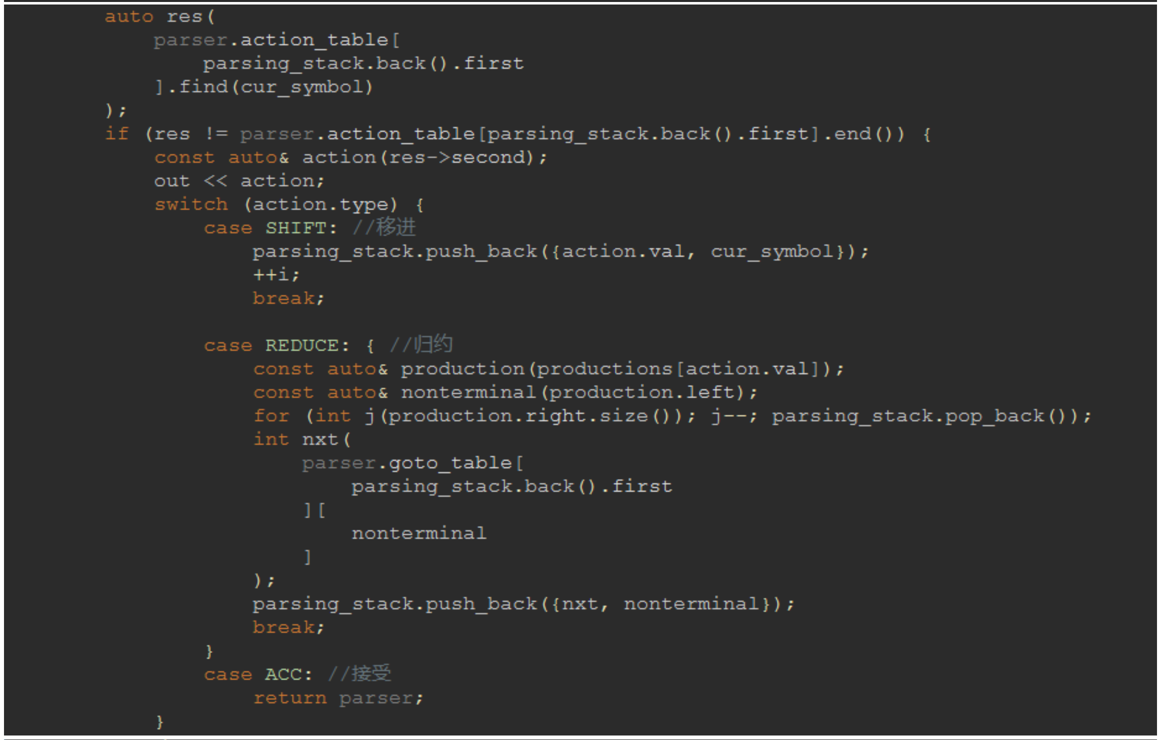
5.5.2. 分析控制程序

LR 分析控制程序是 LR 分析程序的核心。控制程序通过一个栈与分析表，在每一步根据栈顶符号与向前指针所指向的符号查找分析表，从中获得动作指示信息并采取相应的分析动作。其中分析栈分为状态栈与符号栈两部分，同步增减。

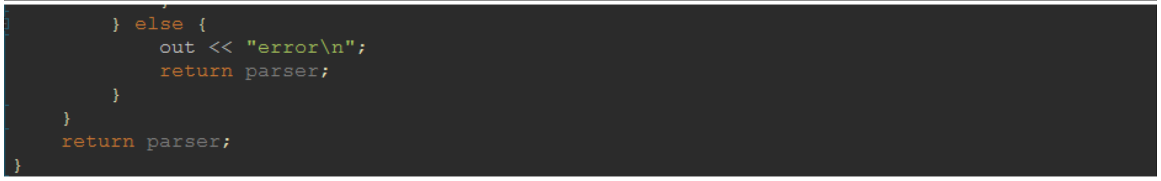
算法实现如下：



```
Parser& operator >>(const Lexer& lexer, Parser& parser) {
    const auto& token_stream(lexer.get_token_stream());
    if (token_stream.front().type == END) {
        return parser;
    }
    deque<pair<int, Symbol>> parsing_stack;
    //将I0入栈
    parsing_stack.push_back({0, END});
    const auto& productions(grammar.get_productions());
    for (int i(0); i < token_stream.size(); ) {
        out << endl;
        out << "current parsing stack:\n\t\t\t\t";
        for (int j(0); j < parsing_stack.size(); ++j) {
            print_symbol(
                out << "[" << parsing_stack[j].first << " ",
                parsing_stack[j].second
            ) << "] ";
        }
        out << endl;
        const Symbol& cur_symbol(token_stream[i].type);
        out << "current token stream:\n\t\t\t\t";
        for (int j(i); j < token_stream.size(); ++j) {
            print_symbol(out, token_stream[j].type) << " ";
        }
        out << endl;
        out << "output:\n\t\t\t\t";
```



```
        auto res(
            parser.action_table[
                parsing_stack.back().first
            ].find(cur_symbol)
        );
        if (res != parser.action_table[parsing_stack.back().first].end()) {
            const auto& action(res->second);
            out << action;
            switch (action.type) {
                case SHIFT: //移进
                    parsing_stack.push_back({action.val, cur_symbol});
                    ++i;
                    break;

                case REDUCE: { //归约
                    const auto& production(productions[action.val]);
                    const auto& nonterminal(production.left);
                    for (int j(production.right.size()); j--; parsing_stack.pop_back());
                    int nxt(
                        parser.goto_table[
                            parsing_stack.back().first
                        ][
                            nonterminal
                        ]
                    );
                    parsing_stack.push_back({nxt, nonterminal});
                    break;
                }
                case ACC: //接受
                    return parser;
            }
```



```
        } else {
            out << "error\n";
            return parser;
        }
    }
    return parser;
}
```

6. 程序测试

测试样例格式如下：

```
（输入算术表达式）
current parsing stack:
                          （当前分析栈）
current token stream:
                          （当前待扫描 token 流）注：已经词法分析将符号串转为 token 流
output:
                          （输出产生式）
```

以下为测试样例：

```
1

current parsing stack:
                                  [0 end]
current token stream:
                                  num end
output:
                                  shift 5

current parsing stack:
                                  [0 end] [5 num]
current token stream:
                                  end
output:
                                  reduce F -> num

current parsing stack:
                                  [0 end] [2 F]
current token stream:
                                  end
output:
                                  reduce T -> F

current parsing stack:
                                  [0 end] [3 T]
current token stream:
                                  end
output:
                                  reduce E -> T

current parsing stack:
                                  [0 end] [1 E]
current token stream:
                                  end
output:
                                  acc
```

```
1+1
current parsing stack:
                                  [0 end]
current token stream:
                                  num + num end
output:
                                  shift 5

current parsing stack:
                                  [0 end] [5 num]
current token stream:
                                  + num end
output:
                                  reduce F -> num

current parsing stack:
                                  [0 end] [2 F]
current token stream:
                                  + num end
output:
                                  reduce T -> F

current parsing stack:
                                  [0 end] [3 T]
current token stream:
                                  + num end
output:
                                  reduce E -> T

current parsing stack:
                                  [0 end] [1 E]
current token stream:
                                  + num end
output:
                                  shift 6

current parsing stack:
                                  [0 end] [1 E] [6 +]
current token stream:
                                  num end
output:
                                  shift 5

current parsing stack:
                                  [0 end] [1 E] [6 +] [5 num]
current token stream:
                                  end
output:
                                  reduce F -> num

current parsing stack:
                                  [0 end] [1 E] [6 +] [2 F]
current token stream:
                                  end
output:
```

```
                                reduce T -> F

current parsing stack:
                                [0 end] [1 E] [6 +] [11 T]
current token stream:
                                end
output:
                                reduce E -> E + T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
2.3+4.5e6
current parsing stack:
                                [0 end]
current token stream:
                                num + num end
output:
                                shift 5

current parsing stack:
                                [0 end] [5 num]
current token stream:
                                + num end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                + num end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                + num end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                + num end
output:
                                shift 6

current parsing stack:
                                [0 end] [1 E] [6 +]
current token stream:
```

```
                                num end
output:
                                shift 5

current parsing stack:
                                [0 end] [1 E] [6 +] [5 num]
current token stream:
                                end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [1 E] [6 +] [2 F]
current token stream:
                                end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [1 E] [6 +] [11 T]
current token stream:
                                end
output:
                                reduce E -> E + T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
(1 + 3) * (3 / 2 + 4)
```

```
current parsing stack:
                                [0 end]
current token stream:
                                ( num + num ) * ( num / num + num ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (]
current token stream:
                                num + num ) * ( num / num + num ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [5 num]
current token stream:
                                + num ) * ( num / num + num ) end
output:
                                reduce F -> num

current parsing stack:
```

```
                                [0 end] [4 (] [2 F]
current token stream:
                                + num ) * ( num / num + num ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T]
current token stream:
                                + num ) * ( num / num + num ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                + num ) * ( num / num + num ) end
output:
                                shift 6

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +]
current token stream:
                                num ) * ( num / num + num ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [5 num]
current token stream:
                                ) * ( num / num + num ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [2 F]
current token stream:
                                ) * ( num / num + num ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [11 T]
current token stream:
                                ) * ( num / num + num ) end
output:
                                reduce E -> E + T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                ) * ( num / num + num ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [10 E] [15 )]
```

```
current token stream:
                                * ( num / num + num ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                * ( num / num + num ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                * ( num / num + num ) end
output:
                                shift 9

current parsing stack:
                                [0 end] [3 T] [9 *]
current token stream:
                                ( num / num + num ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (]
current token stream:
                                num / num + num ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [5 num]
current token stream:
                                / num + num ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [2 F]
current token stream:
                                / num + num ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [3 T]
current token stream:
                                / num + num ) end
output:
                                shift 8

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [3 T] [8 /]
current token stream:
```

```
                              num + num ) end
output:
                              shift 5

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [3 T] [8 /] [5
num]
current token stream:
                              + num ) end
output:
                              reduce F -> num

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [3 T] [8 /]
[13 F]
current token stream:
                              + num ) end
output:
                              reduce T -> T / F

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [3 T]
current token stream:
                              + num ) end
output:
                              reduce E -> T

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [10 E]
current token stream:
                              + num ) end
output:
                              shift 6

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [10 E] [6 +]
current token stream:
                              num ) end
output:
                              shift 5

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [10 E] [6 +]
[5 num]
current token stream:
                              ) end
output:
                              reduce F -> num

current parsing stack:
                              [0 end] [3 T] [9 *] [4 (] [10 E] [6 +]
[2 F]
current token stream:
                              ) end
output:
                              reduce T -> F
```

```
current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [10 E] [6 +]
[11 T]
current token stream:
                                ) end
output:
                                reduce E -> E + T

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [10 E]
current token stream:
                                ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [3 T] [9 *] [4 (] [10 E] [15 )]
current token stream:
                                end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [3 T] [9 *] [14 F]
current token stream:
                                end
output:
                                reduce T -> T * F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
(3.2 + 6.9)
```

```
current parsing stack:
                                [0 end]
current token stream:
                                ( num + num ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (]
current token stream:
                                num + num ) end
```

```
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [5 num]
current token stream:
                                + num ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [2 F]
current token stream:
                                + num ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T]
current token stream:
                                + num ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                + num ) end
output:
                                shift 6

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +]
current token stream:
                                num ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [5 num]
current token stream:
                                ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [2 F]
current token stream:
                                ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [10 E] [6 +] [11 T]
current token stream:
                                ) end
output:
```

```
                                reduce E -> E + T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [10 E] [15 )]
current token stream:
                                end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
((((((4))))))
current parsing stack:
                                [0 end]
current token stream:
                                ( ( ( ( ( ( num ) ) ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (]
current token stream:
                                ( ( ( ( ( num ) ) ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [4 (]
current token stream:
```

```
                              ( ( ( ( num ) ) ) ) ) ) end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (]
current token stream:
                              ( ( ( num ) ) ) ) ) ) end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (]
current token stream:
                              ( ( num ) ) ) ) ) ) end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (] [4 (]
current token stream:
                              ( num ) ) ) ) ) ) end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(]
current token stream:
                              num ) ) ) ) ) ) end
output:
                              shift 5

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(] [5 num]
current token stream:
                              ) ) ) ) ) ) end
output:
                              reduce F -> num

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(] [2 F]
current token stream:
                              ) ) ) ) ) ) end
output:
                              reduce T -> F

current parsing stack:
                              [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(] [3 T]
current token stream:
                              ) ) ) ) ) ) end
output:
                              reduce E -> T
```

```
current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(] [10 E]
current token stream:
                                ) ) ) ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [4
(] [10 E] [15 )]
current token stream:
                                ) ) ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [2
F]
current token stream:
                                ) ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (] [3
T]
current token stream:
                                ) ) ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (]
[10 E]
current token stream:
                                ) ) ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [4 (]
[10 E] [15 )]
current token stream:
                                ) ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [2 F]
current token stream:
                                ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [3 T]
current token stream:
                                ) ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [10 E]
current token stream:
                                ) ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [4 (] [10 E]
[15 )]
current token stream:
                                ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [2 F]
current token stream:
                                ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [3 T]
current token stream:
                                ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [10 E]
current token stream:
                                ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [4 (] [10 E] [15 )]
current token stream:
                                ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [4 (] [2 F]
current token stream:
                                ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T]
current token stream:
                                ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E]
current token stream:
                                ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [15 )]
current token stream:
                                ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [2 F]
current token stream:
                                ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T]
current token stream:
                                ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [10 E] [15 )]
current token stream:
                                end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
```

```
                                end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
((0.2-9) *(5*9/9+(10)))
current parsing stack:
                                [0 end]
current token stream:
                                ( ( num - num ) * ( num * num / num +
( num ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (]
current token stream:
                                ( num - num ) * ( num * num / num +
( num ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [4 (]
current token stream:
                                num - num ) * ( num * num / num +
( num ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [4 (] [5 num]
current token stream:
                                - num ) * ( num * num / num +
( num ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [4 (] [2 F]
current token stream:
                                - num ) * ( num * num / num +
( num ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T]
current token stream:
```

```
                                - num ) * ( num * num / num +
( num ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E]
current token stream:
                                - num ) * ( num * num / num +
( num ) ) ) end
output:
                                shift 7

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [7 -]
current token stream:
                                num ) * ( num * num / num + ( num ) ) )
end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [7 -] [5 num]
current token stream:
                                ) * ( num * num / num + ( num ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [7 -] [2 F]
current token stream:
                                ) * ( num * num / num + ( num ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [7 -] [12 T]
current token stream:
                                ) * ( num * num / num + ( num ) ) ) end
output:
                                reduce E -> E - T

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E]
current token stream:
                                ) * ( num * num / num + ( num ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [10 E] [15 )]
current token stream:
                                * ( num * num / num + ( num ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
```

```
                                [0 end] [4 (] [2 F]
current token stream:
                                * ( num * num / num + ( num ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T]
current token stream:
                                * ( num * num / num + ( num ) ) ) end
output:
                                shift 9

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *]
current token stream:
                                ( num * num / num + ( num ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (]
current token stream:
                                num * num / num + ( num ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [5 num]
current token stream:
                                * num / num + ( num ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [2 F]
current token stream:
                                * num / num + ( num ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T]
current token stream:
                                * num / num + ( num ) ) ) end
output:
                                shift 9

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [9
*]
current token stream:
                                num / num + ( num ) ) ) end
output:
                                shift 5

current parsing stack:
```

```
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [9
*] [5 num]
current token stream:
                                / num + ( num ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [9
*] [14 F]
current token stream:
                                / num + ( num ) ) ) end
output:
                                reduce T -> T * F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T]
current token stream:
                                / num + ( num ) ) ) end
output:
                                shift 8

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [8
/]
current token stream:
                                num + ( num ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [8
/] [5 num]
current token stream:
                                + ( num ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T] [8
/] [13 F]
current token stream:
                                + ( num ) ) ) end
output:
                                reduce T -> T / F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3 T]
current token stream:
                                + ( num ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
current token stream:
                                + ( num ) ) ) end
```

```
output:
                                shift 6

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +]
current token stream:
                                ( num ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (]
current token stream:
                                num ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (] [5 num]
current token stream:
                                ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (] [2 F]
current token stream:
                                ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (] [3 T]
current token stream:
                                ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (] [10 E]
current token stream:
                                ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [4 (] [10 E] [15 )]
current token stream:
                                ) ) end
output:
```

```
                                    reduce F -> ( E )

current parsing stack:
                                    [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [2 F]
current token stream:
                                    ) ) end
output:
                                    reduce T -> F

current parsing stack:
                                    [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[6 +] [11 T]
current token stream:
                                    ) ) end
output:
                                    reduce E -> E + T

current parsing stack:
                                    [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
current token stream:
                                    ) ) end
output:
                                    shift 15

current parsing stack:
                                    [0 end] [4 (] [3 T] [9 *] [4 (] [10 E]
[15 )]
current token stream:
                                    ) end
output:
                                    reduce F -> ( E )

current parsing stack:
                                    [0 end] [4 (] [3 T] [9 *] [14 F]
current token stream:
                                    ) end
output:
                                    reduce T -> T * F

current parsing stack:
                                    [0 end] [4 (] [3 T]
current token stream:
                                    ) end
output:
                                    reduce E -> T

current parsing stack:
                                    [0 end] [4 (] [10 E]
current token stream:
                                    ) end
output:
                                    shift 15

current parsing stack:
                                    [0 end] [4 (] [10 E] [15 )]
current token stream:
```

```
                                end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

```
((2*(5 - 9)/8)*(9/(1/(9-9.36e5))))
current parsing stack:
                                [0 end]
current token stream:
                                ( ( num * ( num - num ) / num ) *
( num / ( num / ( num - num ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (]
current token stream:
                                ( num * ( num - num ) / num ) *
( num / ( num / ( num - num ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [4 (]
current token stream:
                                num * ( num - num ) / num ) * ( num
/ ( num / ( num - num ) ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [4 (] [5 num]
current token stream:
                                * ( num - num ) / num ) * ( num /
( num / ( num - num ) ) ) ) end
output:
```

```
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [4 (] [2 F]
current token stream:
                                * ( num - num ) / num ) * ( num /
( num / ( num - num ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T]
current token stream:
                                * ( num - num ) / num ) * ( num /
( num / ( num - num ) ) ) ) end
output:
                                shift 9

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *]
current token stream:
                                ( num - num ) / num ) * ( num /
( num / ( num - num ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [4
(]
current token stream:
                                num - num ) / num ) * ( num / ( num
/ ( num - num ) ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [5 num]
current token stream:
                                - num ) / num ) * ( num / ( num /
( num - num ) ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [2 F]
current token stream:
                                - num ) / num ) * ( num / ( num /
( num - num ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [3 T]
current token stream:
```

```
                                 - num ) / num ) * ( num / ( num /
( num - num ) ) ) ) end
output:
                                 reduce E -> T

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E]
current token stream:
                                 - num ) / num ) * ( num / ( num /
( num - num ) ) ) ) end
output:
                                 shift 7

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E] [7 -]
current token stream:
                                 num ) / num ) * ( num / ( num /
( num - num ) ) ) ) end
output:
                                 shift 5

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E] [7 -] [5 num]
current token stream:
                                 ) / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                 reduce F -> num

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E] [7 -] [2 F]
current token stream:
                                 ) / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                 reduce T -> F

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E] [7 -] [12 T]
current token stream:
                                 ) / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                 reduce E -> E - T

current parsing stack:
                                 [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E]
current token stream:
                                 ) / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
```

```
                                shift 15

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [4
(] [10 E] [15 )]
current token stream:
                                / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [9 *] [14
F]
current token stream:
                                / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                reduce T -> T * F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T]
current token stream:
                                / num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                shift 8

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [8 /]
current token stream:
                                num ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [8 /] [5
num]
current token stream:
                                ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T] [8 /] [13
F]
current token stream:
                                ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                                reduce T -> T / F

current parsing stack:
                                [0 end] [4 (] [4 (] [3 T]
current token stream:
```

```
                              ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                              reduce E -> T

current parsing stack:
                              [0 end] [4 (] [4 (] [10 E]
current token stream:
                              ) * ( num / ( num / ( num -
num ) ) ) ) end
output:
                              shift 15

current parsing stack:
                              [0 end] [4 (] [4 (] [10 E] [15 )]
current token stream:
                              * ( num / ( num / ( num -
num ) ) ) ) end
output:
                              reduce F -> ( E )

current parsing stack:
                              [0 end] [4 (] [2 F]
current token stream:
                              * ( num / ( num / ( num -
num ) ) ) ) end
output:
                              reduce T -> F

current parsing stack:
                              [0 end] [4 (] [3 T]
current token stream:
                              * ( num / ( num / ( num -
num ) ) ) ) end
output:
                              shift 9

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *]
current token stream:
                              ( num / ( num / ( num - num ) ) ) )
end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (]
current token stream:
                              num / ( num / ( num - num ) ) ) )
end
output:
                              shift 5

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [5
num]
current token stream:
```

```
                              / ( num / ( num - num ) ) ) ) end
output:
                              reduce F -> num

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [2
F]
current token stream:
                              / ( num / ( num - num ) ) ) ) end
output:
                              reduce T -> F

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T]
current token stream:
                              / ( num / ( num - num ) ) ) ) end
output:
                              shift 8

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /]
current token stream:
                              ( num / ( num - num ) ) ) ) end
output:
                              shift 4

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (]
current token stream:
                              num / ( num - num ) ) ) ) end
output:
                              shift 5

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [5 num]
current token stream:
                              / ( num - num ) ) ) ) end
output:
                              reduce F -> num

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [2 F]
current token stream:
                              / ( num - num ) ) ) ) end
output:
                              reduce T -> F

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T]
current token stream:
                              / ( num - num ) ) ) ) end
```

```
output:
                                shift 8

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /]
current token stream:
                                ( num - num ) ) ) ) end
output:
                                shift 4

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (]
current token stream:
                                num - num ) ) ) ) end
output:
                                shift 5

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [5 num]
current token stream:
                                - num ) ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [2 F]
current token stream:
                                - num ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [3 T]
current token stream:
                                - num ) ) ) ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E]
current token stream:
                                - num ) ) ) ) end
output:
                                shift 7

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E] [7 -]
current token stream:
                                num ) ) ) ) end
output:
```

```
                                shift 5

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E] [7 -] [5 num]
current token stream:
                                ) ) ) ) end
output:
                                reduce F -> num

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E] [7 -] [2 F]
current token stream:
                                ) ) ) ) end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E] [7 -] [12 T]
current token stream:
                                ) ) ) ) end
output:
                                reduce E -> E - T

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E]
current token stream:
                                ) ) ) ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [4 (] [10 E] [15 )]
current token stream:
                                ) ) ) end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T] [8 /] [13 F]
current token stream:
                                ) ) ) end
output:
                                reduce T -> T / F

current parsing stack:
                                [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [3 T]
current token stream:
                                ) ) ) end
output:
                                reduce E -> T
```

```
current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [10 E]
current token stream:
                              ) ) ) end
output:
                              shift 15

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [4 (] [10 E] [15 )]
current token stream:
                              ) ) end
output:
                              reduce F -> ( E )

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T] [8 /] [13 F]
current token stream:
                              ) ) end
output:
                              reduce T -> T / F

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [3
T]
current token stream:
                              ) ) end
output:
                              reduce E -> T

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [10
E]
current token stream:
                              ) ) end
output:
                              shift 15

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [4 (] [10
E] [15 )]
current token stream:
                              ) end
output:
                              reduce F -> ( E )

current parsing stack:
                              [0 end] [4 (] [3 T] [9 *] [14 F]
current token stream:
                              ) end
output:
                              reduce T -> T * F

current parsing stack:
```

```
                                [0 end] [4 (] [3 T]
current token stream:
                                ) end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [4 (] [10 E]
current token stream:
                                ) end
output:
                                shift 15

current parsing stack:
                                [0 end] [4 (] [10 E] [15 )]
current token stream:
                                end
output:
                                reduce F -> ( E )

current parsing stack:
                                [0 end] [2 F]
current token stream:
                                end
output:
                                reduce T -> F

current parsing stack:
                                [0 end] [3 T]
current token stream:
                                end
output:
                                reduce E -> T

current parsing stack:
                                [0 end] [1 E]
current token stream:
                                end
output:
                                acc
```

7. 分析总结

经分析，测试结果正确。

有了前两次实验已实现的架构基础，本次的 LR 语法分析程序实现较为轻松，主要难点在于识别文法所有活前缀的 DFA 的构造，在 ItemSet 类的构造中先预处理好一些信息为此提供了遍历，采用广度优先搜索的方法也不重不漏地实现了其构造。