# LL(1)语法分析设计

## 程序功能描述

输入词法分析的输出二元组序列，输入文法构造LL(1)分析表同时构造first集和follow集，给出输入串是否为该文法定义的算术表达式的判断结果。同时能发现简单的语法错误。

## 数据结构描述

本实验包含如下数据结构：first集、follow集、文法集合、LL1分析表、分析栈、非终结符号集和终结符号集。

首先说明first集与follow集的类型是Set。在LL(1).h中定义此类型如下，

typedef map<char, set<char>> Set;

这里的map是C++的映射类型，采用key-value结构存储数据，这里的含义是以非终结符号作为key去寻找此非终结符号的first集或follow集，而集合用set类型描述。

文法集合grammar的类型是NODES，定义如下，

typedef unordered\_map<char, set<string>> NODES;

这里使用了C++的无序映射类型，通过非终结符号找到其所有产生式右部，这些产生式用set<string>来存储。set<string>是一个集合类型，里面每一个元素类型均为string，即产生式的右部。

LL1分析表table的类型为LL1\_table，定义如下，

typedef map<Query, string, compareQuery> LL1\_table;

这里的含义是通过Query类型的变量去查询用以规约的产生式右部。Query类型定义如下，

typedef struct query {

char non\_end;//非终结符号

char end;//终结符号

}Query;

含义是通过非终结符号与终结符号连接的查询相应产生式。compareQuery是一个仿函数，用于map类型排序的方式，属于C++的语言规范，这里不展开讨论。

分析栈LL1\_stack类型是stack<char>，采用C++的栈类型，每一个元素类型为char。

非终结符号集nonTerminate与终结符号集Terminate的类型是set<char>，分别用于存储符号。

此外还有一个类型为NODE，定义如下，

//产生式

typedef struct node {

char left;//产生式左部

string right;//产生式右部

}NODE;

这里数据结构的定义充分体现了在利用分析表进行规约的特点。

## 程序结构描述

## 3.1设计方法

程序设计遵循LL1分析方法，分为三步走。1、求出各非终结符号的first集和follow集；2、利用上一步的两个集合构造LL1分析表；3、利用LL1分析表和LL1分析栈进行LL1分析。具体过程见函数说明。

为了方便程序输入与处理，将原文法中的E’与T’改为X与Y，同时把ε用’$’表示，定义一个宏为EPSILON。文法如下所示，

规定文法的开始符号是S

G[S]: S→V = E

E→TX

X→ATX | ε

T→FY

Y→MFY | ε

F→(E) | i

A→ + | -

M→ \* |/

V→i

## 3.2程序模块

程序包含LL(1).cpp、LL(1).h、main.cpp、utility.cpp、lex.h。其中LL(1).cpp负责LL(1)语法分析，LL(1).h定义了相关数据结构，main.cpp负责调用LL(1).cpp的相关函数以及实施LL1分析。utility.cpp包含辅助函数。lex.h中定义了各种符号的宏。

## 3.3函数说明

LL(1).cpp中包含getCurrent、splitGeneration、setGrammar、genFirst、genFollow、genLL1Table、showFirst、showFollow、showGrammar、showLL1Table、LL\_1这些函数。下面一一介绍。

getCurrent函数在第二次实验中用到，作用是从词法分析输出的中间文件提取出当前的单词符号，这里不再赘述。

splitGeneration函数的作用是分解产生式，传入参数是string类型的产生式generation，格式为E->AX，函数的返回值是NODE类型，将E->AX结构分解为E和AX，装入一个NODE变量返回。分解的方法是取generation的首字符作为左部，’->’之后的以首个非空字符作为首字符的子串为右部。

setGrammar函数的作用是设置文法NODES，同时收集非终结符号与终结符号。当用户在终端上输入一个’E->AX’结构后，setGrammar函数会先调用splitGeneration函数分解产生式(注：这里为了简单起见，输入产生式之间不允许有空格)。接下来，以产生式左部作为key，将相应的右部插入grammar中，写法如下，

grammar[gram.left].insert(gram.right);

然后将左部放入非终极符号集nonTerminate中，右部中的终结符号放入终结符号集Terminate中，写法如下，

nonTerminate.insert(gram.left);

genFirst函数的作用是生成符号的first集。分为对终结符号和非终结符号求解。首先说终结符号，其first集就是自身，直接遍历Terminate将每个元素都加入其first集即可。

非终结符号的求解较为复杂。首先设立一个标志位change，作为first集是否还增大的标志，若change为true则循环进行求解，否则退出。在循环中做的事情是遍历nonTerminate，找出每个非终结符号的产生式集合，然后遍历产生式集合，得到每个产生式(右部)，针对右部的每个字符判断是非终结还是终结符号，若是终结符号，则将其加入当前非终结符号的first集，同时置change为true。若当前符号为非终结符号，则将其first集中的元素(除了$)加入当前非终结符号的first集，若当前非终结符号的first集不包含$，则对右部的遍历到此结束，否则继续向右遍历，这是通过一个标志位forward实现的。若整个右部能够遍历完，则将$加入当前非终结符号的first集。源码见附录。

genFollow函数的作用是生成非终结符号的follow集。方便起见，将文法的开始符号设置为S，follow集中包含$。与求解first集类似，首先遍历非终结符号集，找出所有产生式，继而找出所有右部，对右部的每个符号判断其当前位置，若当前位置不是最后一个字符，则判断当前符号是否为非终结符号，若是非终结符号，则将其后一个字符的first集元素(除了$)放进当前符号的follow集里。若后一个字符的first集包含$，则将左部的first集元素放入当前元素的follow集里。若当前元素已为末尾元素，并且为非终结符号，则将左部的first集放入其follow集中。源码见附录。

genLL1Table函数的作用是生成LL1分析表。首先遍历文法，继而遍历某一个非终结符号的所有右部，将右部的首个字符中的first集的元素与非终结符号组合成Query变量，若first集的元素不为$，则将其插入进table。否则遍历所有终结符号，若不为$并且在非终结符号的follow集里，则组合成Query插入table中。这里要对#单独进行判断，即#是否在非终结符号的follow集中，若在则插入table。

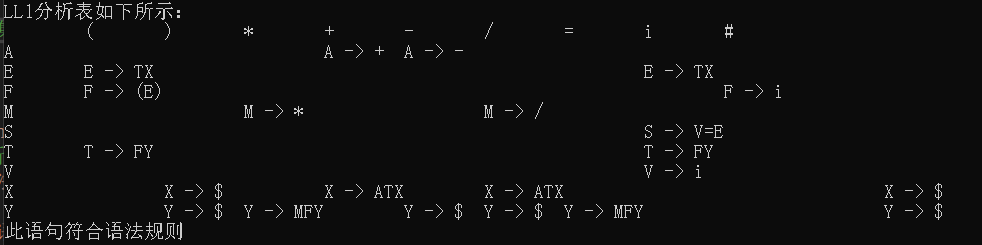
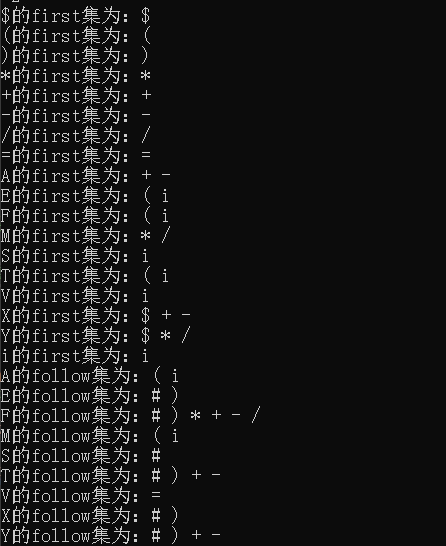
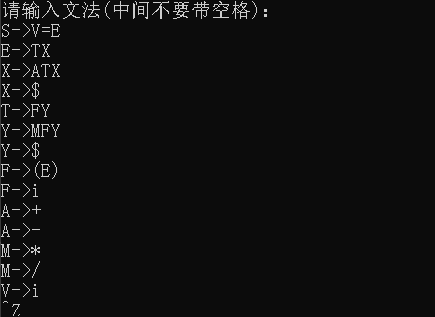
showFirst、showFollow、showLL1Table函数分别用于显示first集、follow集和分析表。

LL\_1函数用于LL\_1分析。首先向分析栈LL1\_stack推入#、S，然后循环读取中间文件，提取出当前符号current，判断栈顶符号是终结符号或#时，若输入符号是标识符，则判断是否与栈顶符号相同，若相同则将当前栈顶符号出栈，同时调用getCurrent函数提取下一个输入符号，否则报错。若输入符号不是标识符，则判断其是否与栈顶符号相同，若相同则只将栈顶符号出栈，同时输入串推进。否则报错。若栈顶符号不是终结符号，则用非终结符号与当前输入符号组成Query去查询table，若没找到则说明出现语法错误，否则如果不是$则使用查询到的右部逆序推入分析栈。

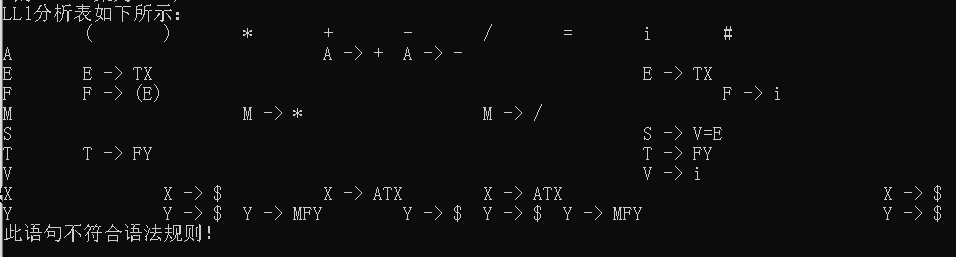
在main.cpp中调用上述函数，最终给出语法分析的结果。

## 程序测试

第一个例子i = i\*(i+i)\*((i+i\*i))#符合语法规则，结果如下所示，



第二个例子i = i\*(i+i)\*((i+i\*a))#因为含有a不符合语法规则，如下所示，



i = i\*(i+i)\*((i+i\*i)))#因为最右边多了一个括号，同样不符合语法规则。

# 源代码附录

这里给出LL(1).cpp的内容，

#include "lex.h"

#include "LL(1).h"

Set first, follow;//first集、follow集

NODES grammar;//文法集合

LL1\_table table;//LL1分析表

stack<char> LL1\_stack;//分析栈

set<char> nonTerminate,Terminate;//非终结符号集, 终结符号集

WORD current = { -1,"" };

WORD getCurrent(FILE\* fp) {

    if (feof(fp))

        return current;

    current.word.clear();

    fseek(fp, 1, 1);

    fscanf(fp, "%d", &current.id);

    fseek(fp, 1, 1);

    char ch = fgetc(fp);

    while (ch != ')') {

        current.word += ch;

        ch = fgetc(fp);

    }

    fseek(fp, 2, 1);//windows下回车占2个字节

    return current;

}

//分解产生式，以->为界

NODE splitGeneration(string generation) {

    string right;

    int len = generation.size();

    int pos = 0;

    for(; pos<len;pos++)

        if (generation[pos] == '>') {

            pos++;

            break;//此时找到->后的首个非空字符

        }

    right = generation.substr(pos, len - pos);

    return { generation[0],right };

}

void setGrammar() {//设置文法NODES, 同时收集非终结符号与终结符号

    cout << "请输入文法(中间不要带空格)：" << endl;

    string generation;

    NODE gram;

    while (cin >> generation) {

        gram = splitGeneration(generation);//首先分解产生式

        grammar[gram.left].insert(gram.right);//再将此产生式加入产生规则

        nonTerminate.insert(gram.left);//收集非终结符号

        int rightLen = gram.right.size();//产生式右部长度

        for (int i = 0; i < rightLen; i++)

            if (!((gram.right)[i] >= 'A' && (gram.right)[i] <= 'Z'))//不是大写字母就是终结符号

                Terminate.insert(gram.right[i]);

    }

}

void genFirst() {//生成first集

    bool change = true;//作为first集是否还增大的标志

    int i = 0;

    //非终结符号求first集

    while (change) {//只要first集还增大，就循环

        change = false;//初始化为false

        for (set<char>::iterator it = nonTerminate.begin(); it != nonTerminate.end(); ++it) {//遍历非终结符号集，找出产生式

            set<string> generations = grammar[\*it];//通过非终结符号找到其所有产生式右部

            for (set<string>::iterator vit = generations.begin(); vit != generations.end(); ++vit) {//遍历这个非终结符号的所有产生式

                string right = \*vit;//产生式右部

                int rightlen = right.size();

                char left = \*it;//产生式左部

                if (Terminate.find(right[0]) != Terminate.end()) {// x -> aa, 首字符属于终结符号集

                    if (first[left].find(right[0]) == first[left].end()) {//如果不在first集才加入

                        first[left].insert(right[0]);//将首字母加入follow(left)

                        change = true;

                    }

                }

                else if (nonTerminate.find(right[0]) != nonTerminate.end()) {//x -> y1y2...yk, 首字符属于非终结符号集

                    i = 0;

                    for (; i < rightlen; ++i) {//遍历右部的每个字符

                        if (Terminate.find(right[i]) != Terminate.end()) {//属于终结符号

                            if (first[left].find(right[i]) == first[left].end()) {//如果不在first集才加入

                                first[left].insert(right[i]);//将首字母加入first(left)

                                change = true;

                            }

                            break;//遇到终结符号就退出

                        }

                        else {

                            bool forward = false;//若当前非终结符号的first集没有epsilon，则停止向后扫描

                            if (first[right[i]].find(EPSILON) != first[right[i]].end())

                                forward = true;

                            for (set<char>::iterator iter = first[right[i]].begin(); iter != first[right[i]].end(); ++iter) {

                                if (first[left].find(\*iter) == first[left].end() && \*iter != EPSILON) {//如果不在first集并且不为epsilon才加入

                                    first[left].insert(\*iter);//将首字母加入first(left)

                                    change = true;

                                }

                            }

                            if (!forward)

                                break;//停止向后扫描

                        }

                    }

                    if (i == rightlen)//此时y1y2...yk的first集均含有epsilon

                        first[left].insert(EPSILON);//向left的first集加入epsilon

                }

            }

        }

    }

    //终结符号求first集

    for (set<char>::iterator it = Terminate.begin(); it != Terminate.end(); ++it)

            first[\*it].insert(\*it);

}

void genFollow() {//生成follow集

    bool change = true;//作为first集是否还增大的标志

    follow['S'].insert('#');//规定文法的开始符号是S

    while (change) {

        change = false;//初始化为false

        for (set<char>::iterator it = nonTerminate.begin(); it != nonTerminate.end(); ++it) {//遍历非终结符号集，找出产生式

            set<string> generations = grammar[\*it];//通过非终结符号找到其所有产生式右部

            for (set<string>::iterator vit = generations.begin(); vit != generations.end(); ++vit) {//遍历这个非终结符号的所有产生式

                string right = \*vit;//产生式右部

                int rightlen = right.size();

                char left = \*it;//产生式左部

                for (int i = 0; i < rightlen; i++) {

                    if (i < rightlen - 1 && nonTerminate.find(right[i]) != nonTerminate.end()) {//A -> aB...

                        for (set<char>::iterator iter = first[right[i + 1]].begin(); iter != first[right[i + 1]].end(); ++iter) {

                            if (follow[right[i]].find(\*iter) == follow[right[i]].end() && \*iter != EPSILON) {//如果不在follow集并且不为epsilon才加入

                                follow[right[i]].insert(\*iter);

                                change = true;

                            }

                        }

                        if (first[right[i + 1]].find('$') != first[right[i + 1]].end()) {//$在right[i+1]的first集

                            for (set<char>::iterator iter = follow[left].begin(); iter != follow[left].end(); ++iter) {

                                if (follow[right[i]].find(\*iter) == follow[right[i]].end()) {//如果不在follow集才加入

                                    follow[right[i]].insert(\*iter);

                                    change = true;

                                }

                            }

                        }

                    }

                    else if (i == rightlen - 1 && nonTerminate.find(right[i]) != nonTerminate.end()) {//A -> aB

                        for (set<char>::iterator iter = follow[left].begin(); iter != follow[left].end(); ++iter) {

                            if (follow[right[i]].find(\*iter) == follow[right[i]].end()) {//如果不在follow集才加入

                                follow[right[i]].insert(\*iter);

                                change = true;

                            }

                        }

                    }

                }

            }

        }

    }

}

void genLL1Table() {//生成LL1分析表

    for (NODES::iterator it = grammar.begin(); it != grammar.end(); ++it) {//遍历文法

        for (set<string>::iterator vit = it->second.begin(); vit != it->second.end(); ++vit) {//遍历某一个非终结符号的所有右部

            char right = (\*vit)[0];//右部的第一个字符

            for (char c : first[right]) {

                Query q;

                q.non\_end = it->first;

                q.end = c;

                if (c != '$') {

                    if (table.find(q) == table.end())

                        //table[q] = \*vit;

                        table.insert(make\_pair(q, \*vit));

                }

                else {

                    for (set<char>::iterator iter = Terminate.begin(); iter != Terminate.end(); ++iter) {

                        if (\*iter != '$' && follow[it->first].find(\*iter) != follow[it->first].end()) {

                            q.end = \*iter;

                            if (table.find(q) == table.end())

                                table.insert(make\_pair(q, \*vit));

                        }

                    }

                    //单独判断#号

                    if (follow[it->first].find('#') != follow[it->first].end()) {

                        q.end = '#';

                        if (table.find(q) == table.end())

                            table.insert(make\_pair(q, \*vit));

                    }

                }

            }

        }

    }

}

void showFirst() {//显示非终结符号的first集

    for (Set::iterator it = first.begin(); it != first.end(); ++it) {

            cout << it->first << "的first集为：";

            for (set<char>::iterator vit = it->second.begin(); vit != it->second.end(); ++vit)

                cout << \*vit << " ";

            cout << endl;

    }

}

void showFollow() {//显示follow集

    for (Set::iterator it = follow.begin(); it != follow.end(); ++it) {

        if (nonTerminate.find(it->first) != nonTerminate.end()) {

            cout << it->first << "的follow集为：";

            for (set<char>::iterator vit = it->second.begin(); vit != it->second.end(); ++vit)

                cout << \*vit << " ";

            cout << endl;

        }

    }

}

void showGrammar() {

    for (NODES::iterator it = grammar.begin(); it != grammar.end(); ++it) {

        for (set<string>::iterator vit = it->second.begin(); vit != it->second.end(); ++vit) {

            cout << it->first << " -> " << \*vit << endl;

        }

    }

}

void showLL1Table() {//显示LL1分析表

    cout << "LL1分析表如下所示：" << endl;

    cout << "\t";

    for (set<char>::iterator it = Terminate.begin(); it != Terminate.end(); ++it) {

        if(\*it!='$')

            cout << \*it << "\t";

    }

    cout << '#' << endl;

    for (set<char>::iterator it = nonTerminate.begin(); it != nonTerminate.end(); ++it) {

        cout << \*it << "\t";

        Query q;

        for (set<char>::iterator vit = Terminate.begin(); vit != Terminate.end(); ++vit) {

            q = { \*it,\*vit };

            if (\*vit != '$' && table.find(q) != table.end())

                cout << \*it << " -> " << table[q] << "\t";

            else if(\*vit != '$')

                cout << "\t";

        }

        q = { \*it,'#' };

        if (table.find(q) != table.end())

            cout << \*it << " -> " << table[q];

        cout << endl;

    }

}

//LL(1)分析

bool LL\_1(FILE\* fp) {

    char c;

    LL1\_stack.push('#');

    LL1\_stack.push('S');

    current = getCurrent(fp);

    while (!feof(fp)) {

        if (current.id == -1)

            break;

        char non\_end = LL1\_stack.top();

        if (Terminate.find(non\_end) != Terminate.end() || non\_end == '#') {//栈顶符号是终结符号或#

            if (current.id == ID) {//标识符

                if (current.word[0] == non\_end) {

                    LL1\_stack.pop();

                    current = getCurrent(fp);//输入串向前推进

                }

                else {

                    cout << "出现未定义的变量" << current.word[0] << endl;

                    return false;

                }

            }

            else if (non\_end == convertion[current.id][0]) {

                if (non\_end == '#')

                    return true;

                LL1\_stack.pop();

                current = getCurrent(fp);//输入串向前推进

            }

            else return false;

        }

        else {

            Query q;

            q.non\_end = non\_end;

            if (current.id == ID)

                q.end = current.word[0];

            else q.end = convertion[current.id][0];

            if (table.find(q) == table.end())

                return false;

            else {

                LL1\_stack.pop();

                string right = table[q];

                if (right != "$") {

                    for (int i = right.size() - 1; i >= 0; i--)

                        LL1\_stack.push(right[i]);

                }

            }

        }

    }

    return true;

}