编译原理 语法分析 实验报告

姓名：陈俊卉　　班级：2020219111　　学号：2020212256

一、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。要求所分析算术表达式由如下的文法产生。
  - E→E+T|E-T|T
  - T→T*F|T/F|F
  - F→(E)|num
- 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。
  - 方法1：编写递归调用程序实现自顶向下的分析。
  - 方法2：编写LL(1)语法分析程序，要求如下。
    - 编程实现算法4.2，为给定文法自动构造预测分析表。
    - 编程实现算法4.1，构造LL(1)预测分析程序。
  - 方法3：编写语法分析程序实现自底向上的分析，要求如下。

        - 构造识别 该文法所有活前缀的DFA。
        - 构造该文法的LR分析表。
        - 编程实现算法4.3，构造LR分析程序。
    - 方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

- **本次实验笔者实现了前三个方法。**
- **注意一：事实上，在调用语法分析时，语句/代码会事先预处理好，整理好词法传入语法分析代码内。所以所有的数字（包括整数、小数等）都会转化为一个文法符号表示。本次实验我们在输入时使用单个的数字（0~9）代表这个文法符号，以笼统地代表所有处理完毕后的所有数字类型。**
    - **也就是说，我们输入的语句中，只需要输入单位数字，其代表的是在词法分析中解析出来的代表数字的文法符号。**
- **注意二：当前版本的C++中，当想要将string直接当作char *使用时会发生强制类型转换，这会引发编译警告**

```
warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
```

可以用c_str()或data()解决该问题。但为了代码的简洁起见，我们仍保留这样的方式。

二、实验环境

- windows10系统
- Visual Stdio Code
- C++ 17

三、方法一：递归

1、原理分析

为了通过递归调用程序实现自顶向下的分析，我们**需要将产生式进行变形，使之没有左递归，提取左公因子**，并且生成非终结符的first集合和follow集合。随后根据**最左推导**的方式完成语句的分析。

2、前置运算

① 消除左递归，提取左公因子

对于形如

$$A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_n|\beta_1|\beta_2|\ldots|\beta_m$$

使用如下公式：

$$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_m A'$$

$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_n A' | \varepsilon$$

可以得到消除左递归后的文法如下（e为空符号，n为num）：

$$\begin{gathered}
E \rightarrow TP \\
P \rightarrow +TP \\
P \rightarrow -TP \\
P \rightarrow e \\
T \rightarrow FQ \\
Q \rightarrow *FQ \\
Q \rightarrow /FQ \\
Q \rightarrow e \\
F \rightarrow (E) \\
F \rightarrow n
\end{gathered}$$

② 生成first集合与follow集合

生成的first集合与follow集合如下：

	E	P	T	Q	F
FIRST	(, num	+, -, e	(, num	*, /, e	(, num
FOLLOW	) , $	$,)	+, -, $,)	+, -, $,)	*, /, +, -, $,)

③ 实现简介

- 构造每一个产生式左方非终结符的函数，按照产生式右方的符号依次执行代码。利用判断区分同一个左方非终结符的不同产生式。

3、代码展示

```cpp
#include<iostream>
#include<string.h>
#include<string>
#include<stdlib.h>
#include<fstream>
#include <iomanip>
#include <unordered_map>
using namespace std;

// 方法一  本质：从左到右递归遍历求解

// 设置一个标记，如果有任何一个地方与产生式不符则置0
int flag = 1;

// 接收字符串
char str[100000];
// 指针
int p = 0;

void handle_F();
```

```
void handle_Q();
void handle_T();
void handle_P();
void handle_E();




void input_str(){
    cout << "plz input string for analysis:" << endl;
    int q = 0;
    char c;
    while (1) {
        if ((c = getchar()) == '\n'){
            str[q++] = '$';
            str[q++] = '\0';
            break;
        }
        str[q++] = c;
    }
}


// 产生式
char *production[] = {
    "E->TP",
    "P->+TP",
    "P->-TP",
    "P->e",
    "T->FQ",
    "Q->*FQ",
    "Q->/FQ",
    "Q->e",
    "F->(E)",
    "F->n",
};

void print_str(){
    cout << "left char: ";
    for(int i = p; ;){
        if(str[i] != '$'){
            cout << str[i];
            i++;
        }
        else
            break;
    }
    cout << endl;
    cout << endl;
}


void handle_F(){
    if (str[p] == '(') {
        cout << "F->(E)" << endl;
        print_str();
        p++;
        handle_E();
```

```
        if (str[p] == ')') {
            p++;
        }

        else {
            flag = 0;
        }
    }

    else if (str[p] >= '0' && str[p] <= '9') {
        cout << "F->num" << endl;
        print_str();
        p++;
    }

    else {
        flag = 0;
    }
}

void handle_Q(){
    if (str[p] == '*') {
        cout << "Q->*FQ" << endl;
        print_str();
        p++;
        handle_F();
        handle_Q();
    }

    else if (str[p] == '/') {
        cout << "Q->/FQ" << endl;
        print_str();
        p++;
        handle_F();
        handle_Q();
    }
    else {
        cout << "Q->e" << endl;
        print_str();
    }
}


void handle_T(){
    cout << "T->FQ" <<endl;
    print_str();
    handle_F();
    handle_Q();
}

void handle_P(){
    if (str[p] == '+') {
        cout << "P->+TP" << endl;
        print_str();
        p++;
        handle_T();
        handle_P();
    }
```

```
    else if (str[p] == '-') {
        cout << "P->-TP" << endl;
        print_str();
        p++;
        handle_T();
        handle_P();
    }
    else {
        cout << "P->e" << endl;
        print_str();
    }
}


void handle_E(){
    cout << "E->TP" << endl;
    print_str();
    handle_T();
    handle_P();
}

void is_correct(){
    if (flag == 1 && str[p] == '$')
        cout << "the sentence is correct for the grammar." << endl;
    else
        cout << "the sentence is wrong for the grammar." << endl;
}

int main(){
    input_str();
    cout << endl;
    cout << "start analysing:" << endl;
    handle_E();
    is_correct();
}
```

4、测试结果

- **约定：e为空（下文也同样）**

```
plz input string for analysis:
5-3*(1+2)

start analysing:
E->TP
left char: 5-3*(1+2)

T->FQ
left char: 5-3*(1+2)

F->num
left char: 5-3*(1+2)

Q->e
```

```
left char: -3*(1+2)

P->-TP
left char: -3*(1+2)

T->FQ
left char: 3*(1+2)

F->num
left char: 3*(1+2)

Q->*FQ
left char: *(1+2)

F->(E)
left char: (1+2)

E->TP
left char: 1+2)

T->FQ
left char: 1+2)

F->num
left char: 1+2)

Q->e
left char: +2)

P->+TP
left char: +2)

T->FQ
left char: 2)

F->num
left char: 2)

Q->e
left char: )

P->e
left char: )

Q->e
left char:

P->e
left char:

the sentence is correct for the grammar.
```

```
plz input string for analysis:
5*(7-1)/3

```

```
start analysing:
E->TP
left char: 5*(7-1)/3

T->FQ
left char: 5*(7-1)/3

F->num
left char: 5*(7-1)/3

Q->*FQ
left char: *(7-1)/3

F->(E)
left char: (7-1)/3

E->TP
left char: 7-1)/3

T->FQ
left char: 7-1)/3

F->num
left char: 7-1)/3

Q->e
left char: -1)/3

P->-TP
left char: -1)/3

T->FQ
left char: 1)/3

F->num
left char: 1)/3

Q->e
left char: )/3

P->e
left char: )/3

Q->/FQ
left char: /3

F->num
left char: 3

Q->e
left char:

P->e
left char:

the sentence is correct for the grammar.
```

经过检验，输出完全正确。

四、方法二：LL(1)分析

1、原理分析

由于我们需要编写LL(1)语法分析程序，所以我们首先需要将给出的文法变为LL(1)文法，再写出相应的first集合和follow集合，以使用算法4.2自动构造预测分析表；随后根据预测分析表构造LL(1)预测分析程序。**LL(1)同样是自上而下的分析方法，而且是有效的无回溯的。**无回溯的原因是构造并使用了分析表。

2、前置运算

① 改写文法（与方法一是一致的）

$$
\begin{gathered}
E \rightarrow TP \\
P \rightarrow +TP \\
P \rightarrow -TP \\
P \rightarrow e \\
T \rightarrow FQ \\
Q \rightarrow *FQ \\
Q \rightarrow /FQ \\
Q \rightarrow e \\
F \rightarrow (E) \\
F \rightarrow n
\end{gathered}
$$

② 生成first集合与follow集合

	E	P	T	Q	F
FIRST	(, num	+, -, e	(, num	*, /, e	(, num
FOLLOW	$,)	$,)	+, -, $,)	+, -, $,)	*, /, +, -, $,)

③ LL(1)预测分析表（目的是为了验证生成的分析表是否正确）

	+	-	*	/	(	)	num	$
E					E→TP		E→TP	
P	P→+TP	P→-TP				P→e		P→e
T					T→FQ		T→FQ	
Q	Q→e	Q→e	Q→*FQ	Q→/FQ		Q→e		Q→e
F					F→(E)		F→num	

④ 实现简介

- 使用table数组记录预测分析表。
  - ```
    // table
    string table[100][100][100];
    /*
            表格式:
                +    -    *    /    (    )    n    $
            E
            T
            F
            P
            Q
    */
    ```
- 使用p数组记录表中某处是否存在产生式。
  - ```
    // 三维数组最后一维的指针
    int p[6][9] = {
        {0,0,0,0,0,0,0,0,0},
        {0,-1,-1,-1,-1,-1,-1,-1,-1},
        {0,-1,-1,-1,-1,-1,-1,-1,-1},
        {0,-1,-1,-1,-1,-1,-1,-1,-1},
        {0,-1,-1,-1,-1,-1,-1,-1,-1},
        {0,-1,-1,-1,-1,-1,-1,-1,-1},
    };

    ```
  - 代码前半部分使用switch语句，提高可读性；后半部分输出使用unordered_map，不再重复case过程，缩减代码冗余，增加效率。

算法 4.2 预测分析表的构造方法

输入：文法 G。

输出：文法 G 的预测分析表 M。

方法：

```
for(文法 G 的每一个产生式 A→α) {
    for(每个终结符号 a ∈FIRST(α)) 把 A→α 放入表项 M[A,a]中;
    if( ε ∈FIRST(α))
        for (每个 b ∈FOLLOW(A)) 把 A→α 放入表项 M[A,b]中;
    };
for( 所有无定义的表项 M[A,a] )  标上错误标志。
```

算法 4.1 非递归预测分析方法。

输入：输入符号串 ω，文法 G 的一张预测分析表 M。

输出：若 $\omega \in L(G)$，则输出 ω 的最左推导，否则报告错误。

方法：

首先，初始化，即将 $ 压入栈底，将文法开始符号 S 压入栈顶；将 $\omega$$ 放入输入缓冲区中，并置向前指针 ip 指向 $\omega$$ 的第一个符号。

然后，预测分析控制程序根据分析表 M 对输入符号串 ω 作出自顶向下的分析，过程如下：

```
do{
      令 X 是栈顶文法符号,a 是 ip 所指向的输入符号;
      if( X 是终结符号或$)
            if( X==a ) {从栈顶弹出 X;ip 前移一个位置;};
            else  error();
      else                                          /* X 是非终结符号 */
          if( M[X,a]=X→Y1Y2…Yk){
            从栈顶弹出 X;
            依次把 Yk,Yk-1,…,Y2,Y1 压入栈;          /* Y1 在栈顶 */
            输出产生式 X→Y1Y2…Yk;
          };
          else  error();
} while (X!=$)                                      /* 栈非空,分析继续 */
```

3、代码展示

- **注：其实代码本身若都使用unordered_map，不需要这么多行。前面使用switch是因为这样可读性更高。可以使用映射表映射来减轻代码冗余，但可读性也相应地降低。**

```cpp
#include<iostream>
#include<string.h>
#include<string>
#include<stdlib.h>
#include<fstream>
#include <iomanip>
#include <unordered_map>
using namespace std;


// 产生式 LL(1)
char *production[] = {
    "E->TP",
    "P->+TP",
    "P->-TP",
```

```
    "P->e",
    "T->FQ",
    "Q->*FQ",
    "Q->/FQ",
    "Q->e",
    "F->(E)",
    "F->n",
};

// 终结符
char *terminator[] = {
    "+",
    "-",
    "*",
    "/",
    "(",
    ")",
    "n",
};

// F的first集合和follow集合
char *first_F[] = {
    "(",
    "n",
};
char *follow_F[] = {
    "*",
    "/",
    "+",
    "-",
    "$",
    ")",
};

// T的first集合和follow集合
char *first_T[] = {
    "(",
    "n",
};
char *follow_T[] = {
    "+",
    "-",
    "$",
    ")",
};

// E的first集合和follow集合
char *first_E[] = {
    "(",
    "n",
};
char *follow_E[] = {
    ")",
    "$",
};

// P的first集合和follow集合
char *first_P[] = {
```

```
    "+",
    "-",
    "e",
};
char *follow_P[] = {
    ")",
    "$",
};

// Q的first集合和follow集合
char *first_Q[] = {
    "*",
    "/",
    "e",
};
char *follow_Q[] = {
    "+",
    "-",
    ")",
    "$",
};


// 三维数组最后一维的指针
int p[6][9] = {
    {0,0,0,0,0,0,0,0,0},
    {0,-1,-1,-1,-1,-1,-1,-1,-1},
    {0,-1,-1,-1,-1,-1,-1,-1,-1},
    {0,-1,-1,-1,-1,-1,-1,-1,-1},
    {0,-1,-1,-1,-1,-1,-1,-1,-1},
    {0,-1,-1,-1,-1,-1,-1,-1,-1},
};



// table
string table[100][100][100];
/*
        表格式:
            +    -    *    /    (    )    n    $
        E
        T
        F
        P
        Q
*/

// 实现char *的substr
char *sub_str(char *s,int n,int len)
{
    static char p[20];//或者设置为静态变量
    int i, j = 0;
    while (n--) {
        s++;//确定字符串的首位置
    }
    for(i = n; i < n + len; i++){
         p[j++] = *s;
         s++;
```

```
    }
    return p;
}

void set_table(){
    table[0][1][0] = "+";
    table[0][2][0] = "-";
    table[0][3][0] = "*";
    table[0][4][0] = "/";
    table[0][5][0] = "(";
    table[0][6][0] = ")";
    table[0][7][0] = "n";
    table[0][8][0] = "$";

    table[1][0][0] = "E";
    table[2][0][0] = "T";
    table[3][0][0] = "F";
    table[4][0][0] = "P";
    table[5][0][0] = "Q";

}

void print_table(){
    for (int i = 1; i <= 5; i++){
        for (int j = 1; j <= 8; j++){

            char left;
            char right;
            switch (i)
            {
                case 1:
                {
                    left = 'E';
                    break;
                }
                case 2:
                {
                    left = 'T';
                    break;
                }
                case 3:
                {
                    left = 'F';
                    break;
                }
                case 4:
                {
                    left = 'P';
                    break;
                }
                case 5:
                {
                    left = 'Q';
                    break;
                }
            }
            switch (j)
            {
```

```
                case 1:
                {
                    right = '+';
                    break;
                }
                case 2:
                {
                    right = '-';
                    break;
                }
                case 3:
                {
                    right = '*';
                    break;
                }
                case 4:
                {
                    right = '/';
                    break;
                }

                case 5:
                {
                    right = '(';
                    break;
                }
                case 6:
                {
                    right = ')';
                    break;
                }
                case 7:
                {
                    right = 'n';
                    break;
                }
                case 8:
                {
                    right = '$';
                    break;
                }
            }
            cout << "(" << left << "," << right << "): ";
            // if (p[i][j] == -1) cout << "error";
            for (int k = 0; k <= p[i][j]; k++) {
                cout << table[i][j][k] << " ";
            }
            cout << endl;
        }
    }
}


void predict_analysis_table(){
    for (int i = 0; i <= 9; i++){
        char *pro = production[i];
        // first集合
        char *first_set[100];
```

```
248         int top_first_set = 0;
249 
250         // 由于产生式格式一样，直接将指针往右移动三位得到产生式右边的字符串
251         pro += 3;
252 
253         // 如果第一个为终结符，那么first(α)就是该终结符
254 
255         // 构造α的first集合
256         switch (pro[0])
257         {
258             // 如果是终结符
259             case '+':
260             case '-':
261             case '*':
262             case '/':
263             case '(':
264             case ')':
265             case 'n':
266             case 'e':
267             {
268                 first_set[top_first_set++] = sub_str(pro, 0, 1);
269                 break;
270             }
271 
272 
273             // 如果是非终结符
274             case 'F':
275             {
276                 for (int i = 0; i < 2; i++){
277                     first_set[top_first_set++] = first_F[i];
278                 }
279                 break;
280             }
281 
282             case 'T':
283             {
284                 for (int i = 0; i < 2; i++){
285                     first_set[top_first_set++] = first_T[i];
286                 }
287                 break;
288             }
289 
290             case 'E':
291             {
292                 for (int i = 0; i < 2; i++){
293                     first_set[top_first_set++] = first_E[i];
294                 }
295                 break;
296             }
297 
298             case 'P':
299             {
300                 for (int i = 0; i < 3; i++){
301                     first_set[top_first_set++] = first_P[i];
302                 }
303                 break;
304             }
305 
```

```
            case 'Q':
            {
                for (int i = 0; i < 3; i++){
                    first_set[top_first_set++] = first_Q[i];
                }
                break;
            }
        }
        
        
        
        
        
        // 指针倒回去 构造第一种情况
        pro -= 3;
        for (int i = 0; i < top_first_set; i++)
        {
            // cout << pro << endl;
            switch (pro[0])
            {
                case 'E':
                {
                    switch (first_set[i][0])
                    {
                        case '+':
                        {
                            table[1][1][++p[1][1]] = pro;
                            break;
                        }

                        case '-':
                        {
                            table[1][2][++p[1][2]] = pro;
                            break;
                        }

                        case '*':
                        {
                            table[1][3][++p[1][3]] = pro;
                            break;
                        }

                        case '/':
                        {
                            table[1][4][++p[1][4]] = pro;
                            break;
                        }

                        case '(':
                        {
                            table[1][5][++p[1][5]] = pro;
                            break;
                        }

                        case ')':
                        {
                            table[1][6][++p[1][6]] = pro;
                            break;
```

```
                        }

                        case 'n':
                        {
                            table[1][7][++p[1][7]] = pro;
                            break;
                        }

                        case '$':
                        {
                            table[1][8][++p[1][8]] = pro;
                            break;
                        }
                    }
                    break;
                }

                case 'T':
                {
                    switch (first_set[i][0])
                    {
                        case '+':
                        {
                            table[2][1][++p[2][1]] = pro;
                            break;
                        }

                        case '-':
                        {
                            table[2][2][++p[2][2]] = pro;
                            break;
                        }

                        case '*':
                        {
                            table[2][3][++p[2][3]] = pro;
                            break;
                        }

                        case '/':
                        {
                            table[2][4][++p[2][4]] = pro;
                            break;
                        }

                        case '(':
                        {
                            table[2][5][++p[2][5]] = pro;
                            break;
                        }

                        case ')':
                        {
                            table[2][6][++p[2][6]] = pro;
                            break;
                        }

                        case 'n':
```

```
                        {
                            table[2][7][++p[2][7]] = pro;
                            break;
                        }

                        case '$':
                        {
                            table[2][8][++p[2][8]] = pro;
                            break;
                        }
                    }
                    break;
                }

                case 'F':
                {
                    switch (first_set[i][0])
                    {
                        case '+':
                        {
                            table[3][1][++p[3][1]] = pro;
                            break;
                        }

                        case '-':
                        {
                            table[3][2][++p[3][2]] = pro;
                            break;
                        }

                        case '*':
                        {
                            table[3][3][++p[3][3]] = pro;
                            break;
                        }

                        case '/':
                        {
                            table[3][4][++p[3][4]] = pro;
                            break;
                        }

                        case '(':
                        {
                            table[3][5][++p[3][5]] = pro;
                            break;
                        }

                        case ')':
                        {
                            table[3][6][++p[3][6]] = pro;
                            break;
                        }

                        case 'n':
                        {
                            table[3][7][++p[3][7]] = pro;
                            break;
```

```
                        }

                        case '$':
                        {
                            table[3][8][++p[3][8]] = pro;
                            break;
                        }
                    }
                    break;
                }

                case 'P':
                {
                    switch (first_set[i][0])
                    {
                        case '+':
                        {
                            table[4][1][++p[4][1]] = pro;
                            break;
                        }

                        case '-':
                        {
                            table[4][2][++p[4][2]] = pro;
                            break;
                        }

                        case '*':
                        {
                            table[4][3][++p[4][3]] = pro;
                            break;
                        }

                        case '/':
                        {
                            table[4][4][++p[4][4]] = pro;
                            break;
                        }

                        case '(':
                        {
                            table[4][5][++p[4][5]] = pro;
                            break;
                        }

                        case ')':
                        {
                            table[4][6][++p[4][6]] = pro;
                            break;
                        }

                        case 'n':
                        {
                            table[4][7][++p[4][7]] = pro;
                            break;
                        }

                        case '$':
```

```
                        {
                            table[4][8][++p[4][8]] = pro;
                            break;
                        }
                    }
                    break;
                }

                case 'Q':
                {
                    switch (first_set[i][0])
                    {
                        case '+':
                        {
                            table[5][1][++p[5][1]] = pro;
                            break;
                        }

                        case '-':
                        {
                            table[5][2][++p[5][2]] = pro;
                            break;
                        }

                        case '*':
                        {
                            table[5][3][++p[5][3]] = pro;
                            break;
                        }

                        case '/':
                        {
                            table[5][4][++p[5][4]] = pro;
                            break;
                        }

                        case '(':
                        {
                            table[5][5][++p[5][5]] = pro;
                            break;
                        }

                        case ')':
                        {
                            table[5][6][++p[5][6]] = pro;
                            break;
                        }

                        case 'n':
                        {
                            table[5][7][++p[5][7]] = pro;
                            break;
                        }

                        case '$':
                        {
                            table[5][8][++p[5][8]] = pro;
                            break;
```

```
                        }
                    }
                    break;
                }

            }
        }


        // 构造第二种情况
        // 判断是否存在 e ∈ FIRST(α)
        bool empty_flag = 0;
        for (int i = 0; i < top_first_set; i++){
            // cout << first_set[i] << ' ';
            if ('e' == first_set[i][0])
            {
                empty_flag = 1;
                break;
            }
        }
        // cout << endl;

        // cout << "empty_flag:" << empty_flag << endl;

        if (empty_flag == 1){
            // 选择pro[0]
            // cout << pro << endl;;
            switch (pro[0])
            {
                case 'E':
                {
                    for (int i = 0; i <= 1; i++){
                        switch (follow_E[i][0])
                        {
                            case '+':
                            {
                                table[1][1][++p[1][1]] = pro;
                                break;
                            }
                            case '-':
                            {
                                table[1][2][++p[1][2]] = pro;
                                break;
                            }
                            case '*':
                            {
                                table[1][3][++p[1][3]] = pro;
                                break;
                            }
                            case '/':
                            {
                                table[1][4][++p[1][4]] = pro;
                                break;
                            }
                            case '(':
                            {
                                table[1][5][++p[1][5]] = pro;
                                break;
```

```
                            }
                            case ')':
                            {
                                table[1][6][++p[1][6]] = pro;
                                break;
                            }
                            case 'n':
                            {
                                table[1][7][++p[1][7]] = pro;
                                break;
                            }
                            case '$':
                            {
                                table[1][8][++p[1][8]] = pro;
                                break;
                            }
                            break;
                        }
                    }
                    break;
                }

                case 'T':
                {
                    for (int i = 0; i <= 3; i++){
                        switch (follow_T[i][0])
                        {
                            case '+':
                            {
                                table[2][1][++p[2][1]] = pro;
                                break;
                            }
                            case '-':
                            {
                                table[2][2][++p[2][2]] = pro;
                                break;
                            }
                            case '*':
                            {
                                table[2][3][++p[2][3]] = pro;
                                break;
                            }
                            case '/':
                            {
                                table[2][4][++p[2][4]] = pro;
                                break;
                            }
                            case '(':
                            {
                                table[2][5][++p[2][5]] = pro;
                                break;
                            }
                            case ')':
                            {
                                table[2][6][++p[2][6]] = pro;
                                break;
                            }
                            case 'n':
```

```
                                {
                                    table[2][7][++p[2][7]] = pro;
                                    break;
                                }
                                case '$':
                                {
                                    table[2][8][++p[2][8]] = pro;
                                    break;
                                }
                                break;
                            }
                        }
                        break;
                    }

                    case 'F':
                    {
                        for (int i = 0; i <= 5; i++){
                            switch (follow_F[i][0])
                            {
                                case '+':
                                {
                                    table[3][1][++p[3][1]] = pro;
                                    break;
                                }
                                case '-':
                                {
                                    table[3][2][++p[3][2]] = pro;
                                    break;
                                }
                                case '*':
                                {
                                    table[3][3][++p[3][3]] = pro;
                                    break;
                                }
                                case '/':
                                {
                                    table[3][4][++p[3][4]] = pro;
                                    break;
                                }
                                case '(':
                                {
                                    table[3][5][++p[3][5]] = pro;
                                    break;
                                }
                                case ')':
                                {
                                    table[3][6][++p[3][6]] = pro;
                                    break;
                                }
                                case 'n':
                                {
                                    table[3][7][++p[3][7]] = pro;
                                    break;
                                }
                                case '$':
                                {
                                    table[3][8][++p[3][8]] = pro;
```

```
                                break;
                            }
                            break;
                        }
                    }
                    break;
                }

                case 'P':
                {
                    for (int i = 0; i <= 1; i++){
                        switch (follow_P[i][0])
                        {
                            case '+':
                            {
                                table[4][1][++p[4][1]] = pro;
                                break;
                            }
                            case '-':
                            {
                                table[4][2][++p[4][2]] = pro;
                                break;
                            }
                            case '*':
                            {
                                table[4][3][++p[4][3]] = pro;
                                break;
                            }
                            case '/':
                            {
                                table[4][4][++p[4][4]] = pro;
                                break;
                            }
                            case '(':
                            {
                                table[4][5][++p[4][5]] = pro;
                                break;
                            }
                            case ')':
                            {
                                table[4][6][++p[4][6]] = pro;
                                break;
                            }
                            case 'n':
                            {
                                table[4][7][++p[4][7]] = pro;
                                break;
                            }
                            case '$':
                            {
                                table[4][8][++p[4][8]] = pro;
                                break;
                            }
                            break;
                        }
                    }
                    break;
                }
```

```
828
829                 case 'Q':
830                 {
831                     for (int i = 0; i <= 3; i++){
832                         // cout << follow_Q[i][0];
833                         switch (follow_Q[i][0])
834                         {
835                             case '+':
836                             {
837                                 table[5][1][++p[5][1]] = pro;
838                                 break;
839                             }
840                             case '-':
841                             {
842                                 table[5][2][++p[5][2]] = pro;
843                                 break;
844                             }
845                             case '*':
846                             {
847                                 table[5][3][++p[5][3]] = pro;
848                                 break;
849                             }
850                             case '/':
851                             {
852                                 table[5][4][++p[5][4]] = pro;
853                                 break;
854                             }
855                             case '(':
856                             {
857                                 table[5][5][++p[5][5]] = pro;
858                                 break;
859                             }
860                             case ')':
861                             {
862                                 table[5][6][++p[5][6]] = pro;
863                                 break;
864                             }
865                             case 'n':
866                             {
867                                 table[5][7][++p[5][7]] = pro;
868                                 break;
869                             }
870                             case '$':
871                             {
872                                 table[5][8][++p[5][8]] = pro;
873                                 break;
874                             }
875                             break;
876                         }
877                         // cout << i;
878                     }
879                     break;
880                 }
881
882
883
884                 default:
885                     break;
```

```
            }
        }



    }
}

// 输入数组
char w[10000];
char* input_str(){
    cout << "plz input string for analysis:" << endl;
    int p = 0;
    char c;
    while (1) {
        if ((c = getchar()) == '\n'){
            w[p++] = '\0';
            break;
        }
        w[p++] = c;
    }
    return w;
}
void LL1_prediction(){
    cout << endl;
    cout << "start analysis:" << endl;

    // 建立一个与下标对应的map
    unordered_map<char, int> map{
        {'E', 1},
        {'T', 2},
        {'F', 3},
        {'P', 4},
        {'Q', 5},

        {'+', 1},
        {'-', 2},
        {'*', 3},
        {'/', 4},
        {'(', 5},
        {')', 6},
        {'n', 7},
        {'$', 8},
    };

    // 初始化

    char stack[10000];
    // stack_p指向stack中的栈顶元素
    int stack_p = -1;
    // 将终止符压入栈
    stack[++stack_p] = '$';
    // 将文法开始符号E压入栈
    stack[++stack_p] = 'E';

    char buffer[10000];
    int buffer_p = 0;
    // 将w$放入输入缓冲区
```

```
944     while (1) {
945         if(w[buffer_p] != '\0'){
946             buffer[buffer_p] = w[buffer_p];
947             buffer_p++;
948         }
949         else break;
950     }
951     buffer[buffer_p++] = '$';
952 
953     // 设置向前指针指向w$的第一个符号
954     int forward = 0;
955 
956     // 计数
957     int count = 1;
958 
959     while (stack[stack_p] != '$') {
960         cout << "step" << count << ": " << endl;
961         count++;
962         // 判断是否为终止符的记号
963         int ter_flag = 0;
964         for (int i = 0; i <= 6; i++) {
965             if (stack[stack_p] == terminator[i][0] ||
(stack[stack_p]>='0' && stack[stack_p]<='9')){
966                 ter_flag = 1;
967                 break;
968             }
969         }
970 
971         // log
972         cout << "stack: ";
973         for (int i = 0; i <= stack_p; i++) cout << stack[i];
974         cout << endl;
975 
976         cout << "buffer: ";
977         for (int i = forward; i < buffer_p; i++) cout << buffer[i];
978         cout << endl;
979 
980 
981         if (stack[stack_p] == '$' || ter_flag == 1) {
982             if (stack[stack_p] == buffer[forward] || (stack[stack_p] ==
'n' && (buffer[forward] >= '0' && buffer[forward] <= '9'))) {
983                 // 弹出X，其实就是stack_p--即可
984                 stack_p--;
985                 // forward
986                 forward++;
987             }
988             else {
989                 cout << "error!" << endl;
990                 return ;
991             }
992         }
993 
994         else {
995             // cout << "else" << endl;
996             char buffer_forward = buffer[forward];
997             // 如果是数字需要将其变为n
998             if (buffer[forward] >= '0' && buffer[forward] <= '9') {
999                 buffer_forward = 'n';
```

```
            }

            // 如果有产生式
            if (p[ map[stack[stack_p]] ][ map[buffer_forward] ] != -1) {
                // 坐标
                int x = map[stack[stack_p]];
                int y = map[buffer_forward];
                int z = p[ map[stack[stack_p]] ][ map[buffer_forward] ];
                // 栈顶弹出
                stack_p--;
                // 依次将Y_k Y_k-1 ... Y_1压入栈
                string pro = table[x][y][z];
                for (int i = pro.length() - 1; i >= 0; i--){

                    if (pro[i] == '>') break;
                    else if (pro[i] == 'e') continue;
                    else stack[++stack_p] = pro[i];
                }

                // 输出产生式
                cout << "using production: " << pro << endl;

            }

            else {
                cout << "error!" << endl;
                return ;
            }
        }
        cout << endl;
    }
    if (buffer[forward] == '$')
        cout << "successful!" << endl;
    else
        cout << "error!" << endl;
}



int main() {
    set_table();
    cout << "analysis_table: " << endl;
    predict_analysis_table();
    print_table();
    cout << endl;
    input_str();
    LL1_prediction();
}
```

4、测试结果

```
analysis_table:
(E,+):
(E,-):
(E,*):
```

```
(E,/):
(E,(): E->TP
(E,)):
(E,n): E->TP
(E,$):
(T,+):
(T,-):
(T,*):
(T,/):
(T,(): T->FQ
(T,)):
(T,n): T->FQ
(T,$):
(F,+):
(F,-):
(F,*):
(F,/):
(F,(): F->(E)
(F,)):
(F,n): F->n
(F,$):
(P,+): P->+TP
(P,-): P->-TP
(P,*):
(P,/):
(P,():
(P,)): P->e
(P,n):
(P,$): P->e
(Q,+): Q->e
(Q,-): Q->e
(Q,*): Q->*FQ
(Q,/): Q->/FQ
(Q,():
(Q,)): Q->e
(Q,n):
(Q,$): Q->e

plz input string for analysis:
5*(1+7)/2

start analysis:
step1:
stack: $E
buffer: 5*(1+7)/2$
using production: E->TP

step2:
stack: $PT
buffer: 5*(1+7)/2$
using production: T->FQ

step3:
stack: $PQF
buffer: 5*(1+7)/2$
using production: F->n

step4:
```

```
stack: $PQn
buffer: 5*(1+7)/2$

step5:
stack: $PQ
buffer: *(1+7)/2$
using production: Q->*FQ

step6:
stack: $PQF*
buffer: *(1+7)/2$

step7:
stack: $PQF
buffer: (1+7)/2$
using production: F->(E)

step8:
stack: $PQ)E(
buffer: (1+7)/2$

step9:
stack: $PQ)E
buffer: 1+7)/2$
using production: E->TP

step10:
stack: $PQ)PT
buffer: 1+7)/2$
using production: T->FQ

step11:
stack: $PQ)PQF
buffer: 1+7)/2$
using production: F->n

step12:
stack: $PQ)PQn
buffer: 1+7)/2$

step13:
stack: $PQ)PQ
buffer: +7)/2$
using production: Q->e

step14:
stack: $PQ)P
buffer: +7)/2$
using production: P->+TP

step15:
stack: $PQ)PT+
buffer: +7)/2$

step16:
stack: $PQ)PT
buffer: 7)/2$
using production: T->FQ
```

```
step17:
stack: $PQ)PQF
buffer: 7)/2$
using production: F->n

step18:
stack: $PQ)PQn
buffer: 7)/2$

step19:
stack: $PQ)PQ
buffer: )/2$
using production: Q->e

step20:
stack: $PQ)P
buffer: )/2$
using production: P->e

step21:
stack: $PQ)
buffer: )/2$

step22:
stack: $PQ
buffer: /2$
using production: Q->/FQ

step23:
stack: $PQF/
buffer: /2$

step24:
stack: $PQF
buffer: 2$
using production: F->n

step25:
stack: $PQn
buffer: 2$

step26:
stack: $PQ
buffer: $
using production: Q->e

step27:
stack: $P
buffer: $
using production: P->e

successful!
```

经过对照，分析表生成正确，且分析过程也正确。

五、方法三：LR分析

1、原理分析

LR分析实际上是从下而上的分析，**是最右推导的逆过程，也就是最左规约。为了分析器只有一个接受状态（LR分析以归约到初始符号为接受），我们需要对文法进行增广。**然后求first集与follow集，构造识别所有活前缀的DFA，构造项目集规范族，并以此构造LR分析表，最后使用LR分析表完成算法的实现。

2、前置运算

① 增广文法

$$
\begin{gathered}
S \rightarrow E \\
E \rightarrow E+T \\
E \rightarrow E-T \\
E \rightarrow T \\
T \rightarrow T*F \\
T \rightarrow T/F \\
T \rightarrow F \\
F \rightarrow (E) \\
F \rightarrow n
\end{gathered}
$$

② 生成first集合与follow集合

	S	E	T	F
FIRST	(, num	(, num	(, num	(, num
FOLLOW	$	+, -, $,)	*, /, +, -, $,)	*, /, +, -, $,)

③ 识别所有活前缀的DFA，构造项目及规范族（略，可以在LR分析表中体现）

④ LR分析表

状		A	C	T	I	O	N		G	O	T	O
态	(	)	+	-	*	/	num	$	S	E	T	F
0	S4						S5			1	2	3
1			S6	S7				acc				
2		R4	R4	R4	S8	S9		R4				
3		R7	R7	R7	R7	R7		R7				
4	S4						S5			10	2	3
5		R9	R9	R9	R9	R9		R9				
6	S4						S5				11	3
7	S4						S5				12	3
8	S4						S5					13
9	S4						S5					14
10		S15	S6	S7								
11		R2	R2	R2	S8	S9		R2				
12		R3	R3	R3	S8	S9		R3				
13		R5	R5	R5	R5	R5		R5				
14		R6	R6	R6	R6	R6		R6				
15		R8	R8	R8	R8	R8		R8				

⑤ 实现简介

- 无需画出表格，所以全程使用unordered_map，目的是使用符号映射到下标。
  - ```
    // 文法分析表
    // 建立map
    unordered_map<char, int> map{
        {'(', 0},
        {')', 1},
        {'+', 2},
        {'-', 3},
        {'*', 4},
        {'/', 5},
        {'n', 6},
        {'$', 7},

        {'S', 0},
        {'E', 1},
        {'T', 2},
        {'F', 3},
    };
    ```
- 使用action数组、goto数组记录表格内容：
    ```

```
// S为正数 R为负数 acc为100 -100为err
int ACTION[16][8] = {
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,-100,6,7,-100,-100,-100,100},
    {-100,-4,-4,-4,8,9,-100,-4},
    {-100,-7,-7,-7,-7,-7,-100,-7},
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,-9,-9,-9,-9,-9,-100,-9},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,15,6,7,-100,-100,-100,-100},
    {-100,-2,-2,-2,8,9,-100,-2},
    {-100,-3,-3,-3,8,9,-100,-3},
    {-100,-5,-5,-5,-5,-5,-100,-5},
    {-100,-6,-6,-6,-6,-6,-100,-6},
    {-100,-8,-8,-8,-8,-8,-100,-8},
};
// -100为err
int GOTO[16][4] = {
    {-100,1,2,3},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,10,2,3},
    {-100,-100,-100,-100},
    {-100,-100,11,3},
    {-100,-100,12,3},
    {-100,-100,-100,13},
    {-100,-100,-100,14},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
};
```

算法 4.3 LR 分析程序。

输入：文法 G 的一张分析表和一个输入符号串 ω。

输出：若 $\omega \in L(G)$，得到 ω 的自底向上的分析，否则报错。

方法：

首先初始化，将初始状态 S_0 入栈，将 $\omega\$$ 存入输入缓冲区中；并置 ip 指向 $\omega\$$ 的第一个符号

```
do {
    令 S 是栈顶状态，a 是 ip 所指向的符号；
    if(action[S,a]=shift S′) {
        把 a 和 S′分别压入符号栈和状态栈的栈顶；
        推进 ip,使它指向下一个输入符号；
    };
    else if (action[S,a]=reduce by A→β) {
            从栈顶弹出|β|个符号；
            令 S′是当前栈顶状态，把 A 和 goto[S′,A]分别压入符号栈和状态栈的栈顶；
            输出产生式 A→β;
        };
        else if (action[S,a]=accept) return;
            else error();
} while(1);
```

3、代码展示

- **注：分析表使用两个有mask的数组表示，正数表示S，负数表示R，-100为error，100为acc.**

```cpp
#include<iostream>
#include<string.h>
#include<string>
#include<stdlib.h>
#include<fstream>
#include <iomanip>
#include <unordered_map>
#include<typeinfo>
#include <stdio.h>
using namespace std;


// 接收字符串
char str[100000];
int ip = 0;

// 状态栈
int state_stack[10000];
int state_stack_p = 0;

// 符号栈
char sign_stack[10000];
int sign_stack_p = 0;


// 文法分析表
// 建立map
unordered_map<char, int> map{
    {'(', 0},
```

```
    {')', 1},
    {'+', 2},
    {'-', 3},
    {'*', 4},
    {'/', 5},
    {'n', 6},
    {'$', 7},

    {'S', 0},
    {'E', 1},
    {'T', 2},
    {'F', 3},
};




// S为正数 R为负数 acc为100 -100为err
int ACTION[16][8] = {
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,-100,6,7,-100,-100,-100,100},
    {-100,-4,-4,-4,8,9,-100,-4},
    {-100,-7,-7,-7,-7,-7,-100,-7},
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,-9,-9,-9,-9,-9,-100,-9},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {4,-100,-100,-100,-100,-100,5,-100},
    {-100,15,6,7,-100,-100,-100,-100},
    {-100,-2,-2,-2,8,9,-100,-2},
    {-100,-3,-3,-3,8,9,-100,-3},
    {-100,-5,-5,-5,-5,-5,-100,-5},
    {-100,-6,-6,-6,-6,-6,-100,-6},
    {-100,-8,-8,-8,-8,-8,-100,-8},
};

// -100为err
int GOTO[16][4] = {
    {-100,1,2,3},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,10,2,3},
    {-100,-100,-100,-100},
    {-100,-100,11,3},
    {-100,-100,12,3},
    {-100,-100,-100,13},
    {-100,-100,-100,14},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
    {-100,-100,-100,-100},
};


```

```
88
89
90
91
92  // 产生式 LR
93  // 因为E在多个产生式的左部出现，接受项目不唯一，所以需要增加S->E
94  char *production[] = {
95      "S->E",
96      "E->E+T",
97      "E->E-T",
98      "E->T",
99      "T->T*F",
100     "T->T/F",
101     "T->F",
102     "F->(E)",
103     "F->n",
104 };
105
106
107
108 void set_up_stack(){
109     state_stack[0] = 0;
110     sign_stack[0] = '$';
111 }
112
113
114
115
116 void input_str(){
117     cout << "plz input string for analysis:" << endl;
118     int q = 0;
119     char c;
120     while (1) {
121         if ((c = getchar()) == '\n'){
122             str[q++] = '$';
123             str[q++] = '\0';
124             break;
125         }
126         str[q++] = c;
127     }
128 }
129
130
131
132 void LR_analisis(){
133     while (1) {
134         char str_ip = str[ip];
135         if (str_ip >= '0' && str_ip <= '9') str_ip = 'n';
136
137         // log
138         // cout << "ACTION:";
139         // cout <<ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] <<
    endl;
140
141         // cout << "state_p:";
142         // cout << state_stack_p << endl;
143
144         // cout << "ip:";
```

```
        // cout << ip << endl;

        cout << "state stack:";
        for (int i = 0;i <= state_stack_p; i++) cout << state_stack[i] << '
';
        cout << endl;

        cout << "sign stack:";
        for (int i = 0;i <= sign_stack_p; i++) cout << sign_stack[i];
        cout << endl;

        cout << "the rest:";
        int temp_ip = ip;
        while (str[temp_ip] != '$') {
            cout << str[temp_ip];
            temp_ip++;
        }
        cout << endl;









        // 如果是S
        if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] > 0 &&
ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] != 100) {
            cout << "S" << ACTION[ state_stack[state_stack_p] ][
map[str_ip] ] << endl;


            // 将状态加入

            state_stack[state_stack_p + 1] = ACTION[
state_stack[state_stack_p] ][ map[str_ip] ];

            state_stack_p++;

            // 将符号加入
            sign_stack[++sign_stack_p] = str[ip];
            // 推进ip
            ip++;

            cout << endl;
            cout << endl;
        }

        // 如果是R
        else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] < 0 &&
ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] != -100) {
            cout << "R" << abs(ACTION[ state_stack[state_stack_p] ][
map[str_ip] ]) << '\t';
            // 绝对值的下标从1开始，所以我们需要先取绝对值再减一得到对应production数
组的下标
            int pro_p = abs(ACTION[ state_stack[state_stack_p] ][
map[str_ip] ]) - 1;

            // 标记到箭头右边
```

```
            int right_flag = 0;
            // 计数
            int count = 0;
            for (int i = 0; ; i++) {
                if (production[pro_p][i] == '>') {
                    right_flag = 1;
                    continue;
                }
                else if (production[pro_p][i] == '\0') {
                    break;
                }
                else if (right_flag == 1) {
                    count++;
                }
            }
            // 两个栈分别弹出count个符号
            state_stack_p -= count;
            sign_stack_p -= count;

            // 将当前状态的goto压入状态栈栈顶， 将产生式左边的非终结符压入符号栈栈顶
            // 将状态加入

            state_stack[state_stack_p + 1] = GOTO[
state_stack[state_stack_p] ][ map[production[pro_p][0]] ];
            state_stack_p++;
            sign_stack[++sign_stack_p] = production[pro_p][0];

            // 输出
            cout << production[pro_p] << endl;

            cout << endl;
        }
        // 如果是acc
        else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] ==
100) {
            cout << "accept!" << endl;
            return ;
        }

        else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] ==
-100) {
            cout << "error!" << endl;
            return ;
        }
    }
}

int main(){
    set_up_stack();
    input_str();
    LR_analisis();
}
```

4、测试结果

```
plz input string for analysis:
5*(1+7)/2
state stack:0
sign stack:$
the rest:5*(1+7)/2
S5


state stack:0 5
sign stack:$5
the rest:*(1+7)/2
R9      F->n

state stack:0 3
sign stack:$F
the rest:*(1+7)/2
R7      T->F

state stack:0 2
sign stack:$T
the rest:*(1+7)/2
S8


state stack:0 2 8
sign stack:$T*
the rest:(1+7)/2
S4


state stack:0 2 8 4
sign stack:$T*(
the rest:1+7)/2
S5


state stack:0 2 8 4 5
sign stack:$T*(1
the rest:+7)/2
R9      F->n

state stack:0 2 8 4 3
sign stack:$T*(F
the rest:+7)/2
R7      T->F

state stack:0 2 8 4 2
sign stack:$T*(T
the rest:+7)/2
R4      E->T

state stack:0 2 8 4 10
sign stack:$T*(E
the rest:+7)/2
S6
```

```
state stack:0 2 8 4 10 6
sign stack:$T*(E+
the rest:7)/2
S5


state stack:0 2 8 4 10 6 5
sign stack:$T*(E+7
the rest:)/2
R9      F->n

state stack:0 2 8 4 10 6 3
sign stack:$T*(E+F
the rest:)/2
R7      T->F

state stack:0 2 8 4 10 6 11
sign stack:$T*(E+T
the rest:)/2
R2      E->E+T

state stack:0 2 8 4 10
sign stack:$T*(E
the rest:)/2
S15


state stack:0 2 8 4 10 15
sign stack:$T*(E)
the rest:/2
R8      F->(E)

state stack:0 2 8 13
sign stack:$T*F
the rest:/2
R5      T->T*F

state stack:0 2
sign stack:$T
the rest:/2
S9


state stack:0 2 9
sign stack:$T/
the rest:2
S5


state stack:0 2 9 5
sign stack:$T/2
the rest:
R9      F->n

state stack:0 2 9 14
sign stack:$T/F
```

```
the rest:
R6      T->T/F

state stack:0 2
sign stack:$T
the rest:
R4      E->T

state stack:0 1
sign stack:$E
the rest:
accept!
```

结果完全正确。

六、总结与展望

本次实验，我分别使用递归分析法，LL(1)分析法与LR分析法对输入的算术表达式进行语法分析。其中，递归分析法是自上而下的分析，且存在回溯；LL(1)分析法是自上而下的分析，由于使用了分析表所以没有回溯发生；LR分析是下而上的分析。

在文法改写方面，自上而下的分析**需要去除文法左递归并提取左公因子**。而自下而上的分析**需要增广文法，确保接受状态只有一个。**

本次实验，我不仅对三种不同语法分析的方法有了更加深刻的理解，而且也很好地锻炼了我的C++代码编写能力与整体功能架构的设计能力。我还明白了事先构思好数据处理的方式可能能够大幅度减少代码的冗余（如方法二中的代码，经过unordered_map的替换之后可以压缩到300行左右）。

美中不足的是，由于时间紧张，本次实验我采用的都是面向过程的编程方法。希望我在之后能够对语法分析有更加清晰和深刻的认识，也希望我的C++代码能力能够进一步的提高。