****

编译原理

语法分析器实验报告

Compilers Principles

Syntax Analyzer Programming

**Laboratory Report**

姓名Name ： 许润

学号Number： 71119103

东南大学软件学院

School of Software Engineering

Southeast University

2021年12月

**一、实验目的**

（1）理解并掌握词法，语法分析的原理与方法。

（2）能够使用某种语言实现词法，语法分析程序。

（3）对编译的基本概念，原理和方法有完整和清楚的理解，并能正确而熟练的运用。

**二、实验内容**

（1）选择某个语言的一个子集（与词法分析相同），称其为mini语言，将其语法用BNF范式表示（从语言的标准剪裁）

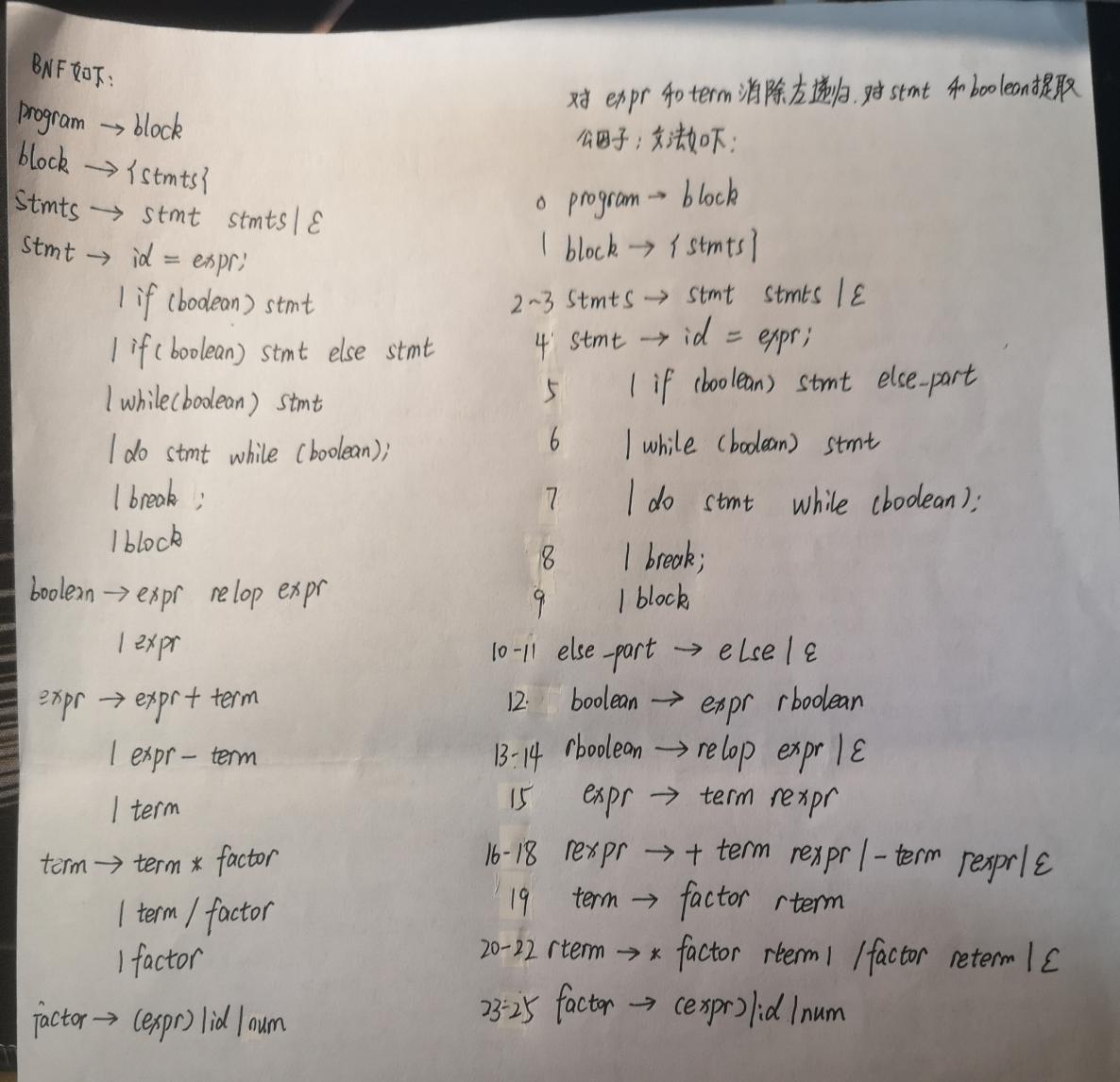
（2）用LL(1)方法进行语法分析

（3）用前述mini语言写出若干程序作为测试用例。

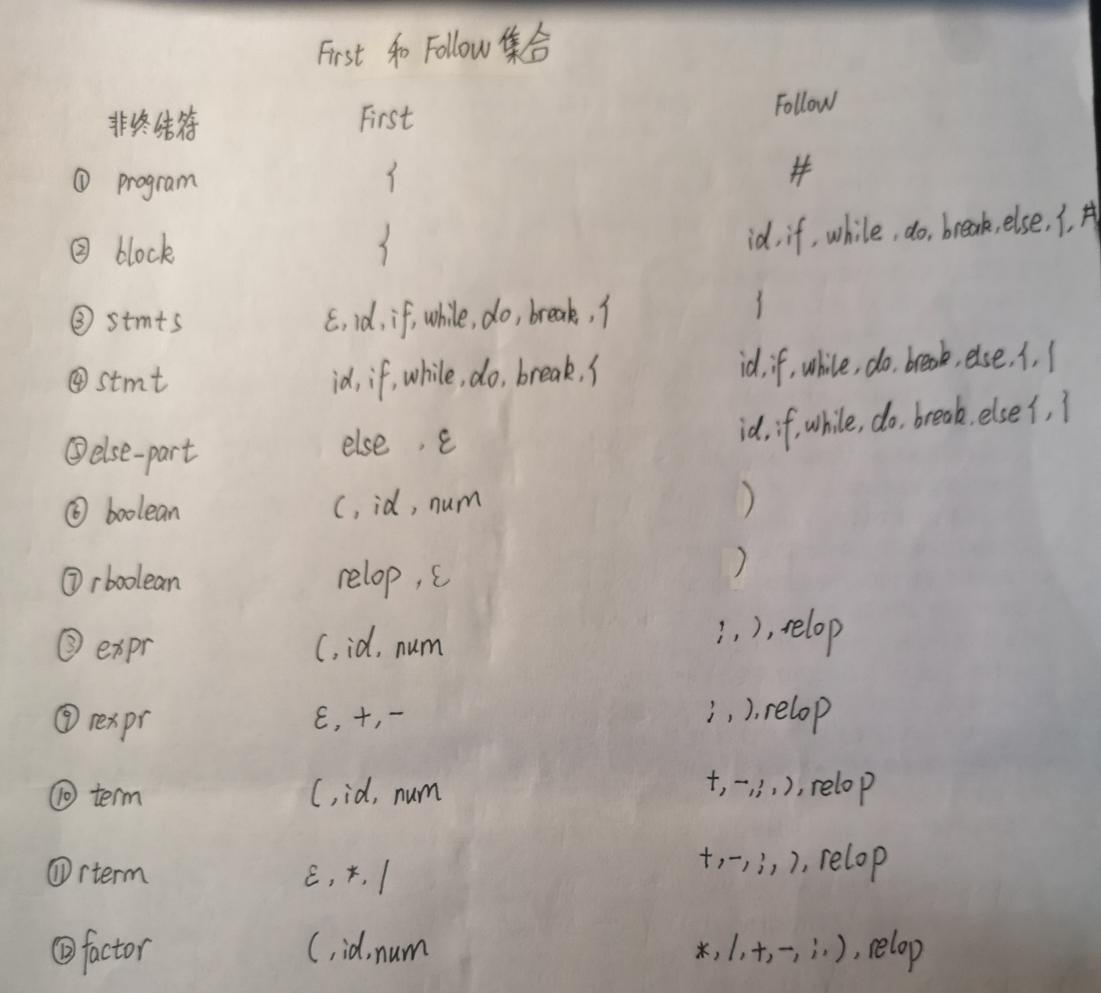
（4）用前述LL(1)分析程序对这些测试用例进行语法分析，生成语法树。检查相应的语法树是否正确。

**三、实验过程**

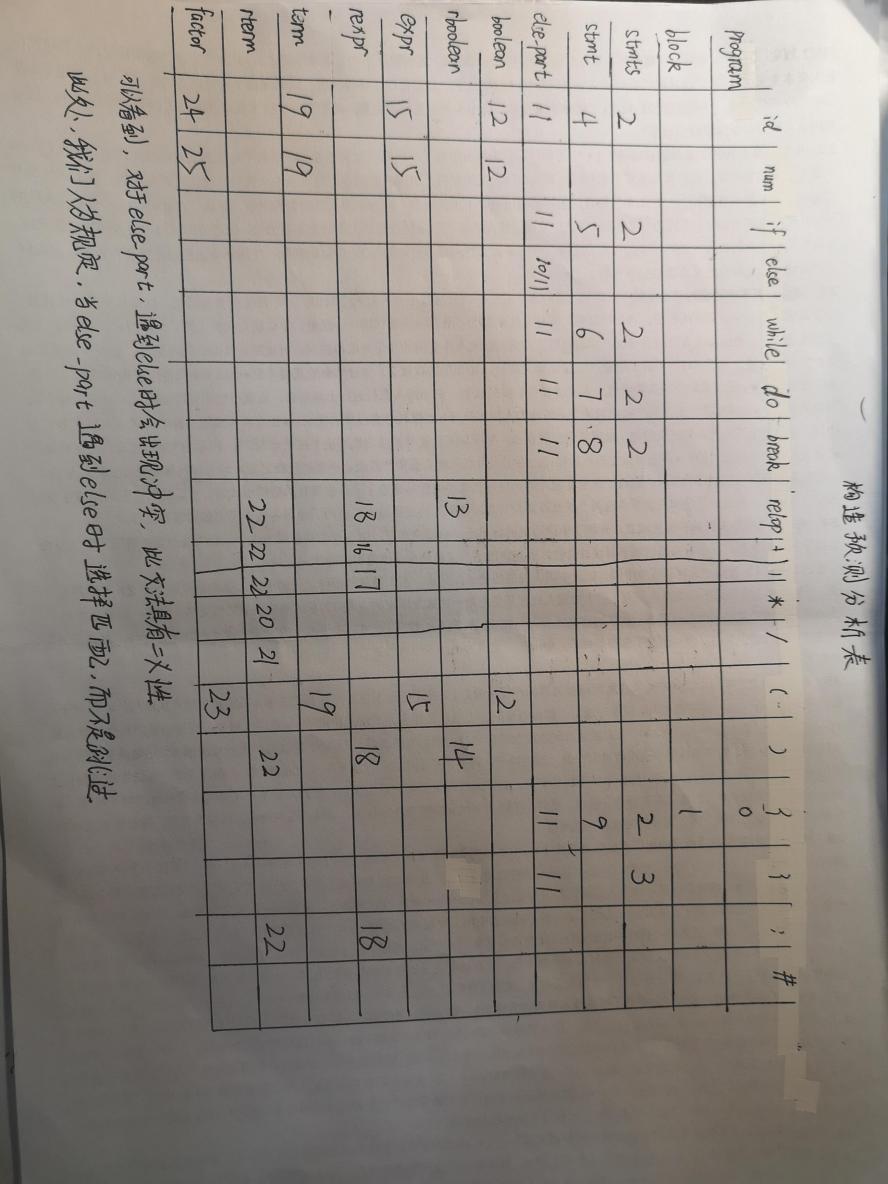
（1）写出BNF式并提取公共左因子消除、消除左递归：



1. 分析First集，Follow集：



1. 构造LL(1)预测分析表，并消除二义性：



可以看到，上表中对于else\_part，遇到else时会有两种可能的选择，因此该文法具有二义性。消除二义性的一种方法是

stmt → matched\_ stmt

|open\_stmt

matched stmt →if ( boolean ) matched stmt else matched\_stmt

|other

open\_stmt →if ( boolean ) stmt

|if ( boolean ) matched\_stmt else open\_stmt

以此来确保else 会跟随紧挨着的上一个if。但是这样做又引入了if ( boolean )公因子的问题，当处理stmt时的下一个终结符为if时，无法决定是走matched\_stmt还是open \_stmt，而且这个公因子消除比较困难，也可能无法消除。如果无法消除，那么说明这个文法不是LL(1)文法。

另一种方法是，人为规定else\_part 遇到else时选择匹配，而不是跳过，这样也能确保else会跟随紧挨着的上一个if，而且解决了else\_part，遇到else时会有两种可能的选择的问题。这里采用第二种方法。

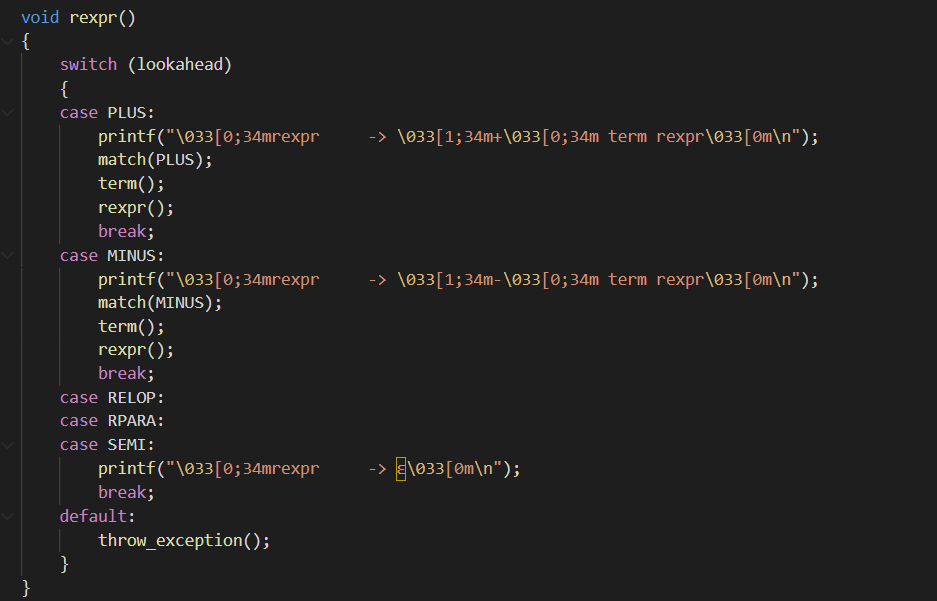
1. **词法分析器的实现**
2. **通过Lex生成词法分析器**

预先定义了每种类型的词法单元的枚举常量：

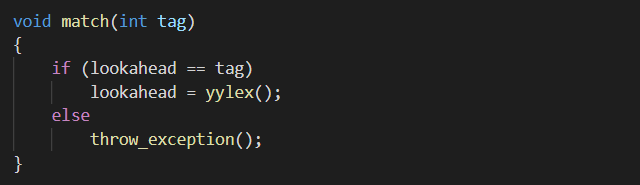
enum tag{ IF, ELSE, DO, WHILE, BREAK, ID, NUM, ASSIGN, RELOP, PLUS, MINUS, TIMES, DIVIDE, SEMI, LPARA, RPARA, LBRACE, RBRACE };

1. **递归下降分析**

根据当前的非终结符在预测分析表中每一个接下来可能出现的终结符lookahead进行分支，如果进入default，则进入了上表的空白区域，因此需要报错。以rexpr为例：



match函数将当前遇到的终结符lookahead和所传入的标记进行比较，如果没有问题，则继续通过yylex取下一个词法单元，否则报错。



1. **实验结果**
2. **测试用例test1.c**

{

while (i <= 100)

{

if (i == 2)

sum = sum + i;

else if (i > 5)

do

{

sum = sum \* (2 - sum) / 2 + 1;

if (sum < 0)

break;

} while (i < 0);

if (i <= 2 \* (i + 1))

sum = 1;

else

sum = 0;

i = i + 2;

}

}

1. **运行**

依次输入以下命令编译运行：先通过flex编译lexer.l，之后将生成的lex.yy.c和parser.c一起编译

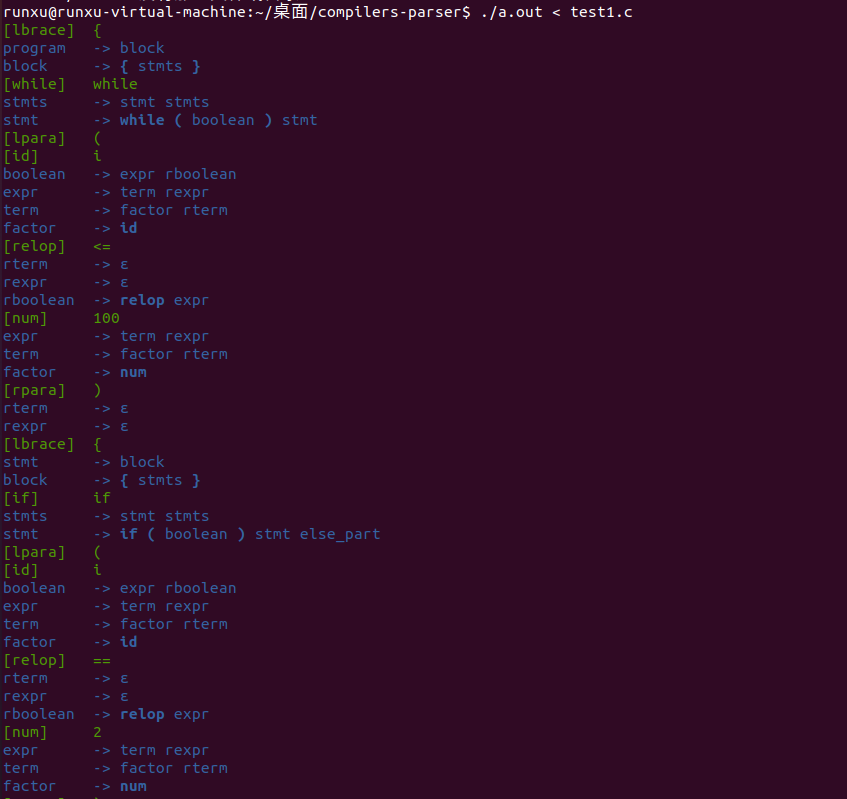
flex lexer.l

cc lex.yy.c parser.c -ll

./a.out < test1.c

1. **结果**

**部分结果如图所示：**

****

**完整结果如下所示：**

[lbrace] {

program -> block

block -> { stmts }

[while] while

stmts -> stmt stmts

stmt -> while ( boolean ) stmt

[lpara] (

[id] i

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] <=

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 100

expr -> term rexpr

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[lbrace] {

stmt -> block

block -> { stmts }

[if] if

stmts -> stmt stmts

stmt -> if ( boolean ) stmt else\_part

[lpara] (

[id] i

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] ==

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 2

expr -> term rexpr

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[id] sum

stmt -> id = expr

[assign] =

[id] sum

expr -> term rexpr

term -> factor rterm

factor -> id

[plus] +

rterm -> ε

rexpr -> + term rexpr

[id] i

term -> factor rterm

factor -> id

[semi] ;

rterm -> ε

rexpr -> ε

[else] else

else\_part -> else stmt

[if] if

stmt -> if ( boolean ) stmt else\_part

[lpara] (

[id] i

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] >

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 5

expr -> term rexpr

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[do] do

stmt -> do stmt while ( boolean )

[lbrace] {

stmt -> block

block -> { stmts }

[id] sum

stmts -> stmt stmts

stmt -> id = expr

[assign] =

[id] sum

expr -> term rexpr

term -> factor rterm

factor -> id

[times] \*

rterm -> \* factor rterm

[lpara] (

factor -> ( expr )

[num] 2

expr -> term rexpr

term -> factor rterm

factor -> num

[minus] -

rterm -> ε

rexpr -> - term rexpr

[id] sum

term -> factor rterm

factor -> id

[rpara] )

rterm -> ε

rexpr -> ε

[divide] /

rterm -> / factor rterm

[num] 2

factor -> num

[plus] +

rterm -> ε

rexpr -> + term rexpr

[num] 1

term -> factor rterm

factor -> num

[semi] ;

rterm -> ε

rexpr -> ε

[if] if

stmts -> stmt stmts

stmt -> if ( boolean ) stmt else\_part

[lpara] (

[id] sum

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] <

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 0

expr -> term rexpr

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[break] break

stmt -> break

[semi] ;

[rbrace] }

else\_part -> ε

stmts -> ε

[while] while

[lpara] (

[id] i

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] <

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 0

expr -> term rexpr

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[semi] ;

[if] if

else\_part -> ε

stmts -> stmt stmts

stmt -> if ( boolean ) stmt else\_part

[lpara] (

[id] i

boolean -> expr rboolean

expr -> term rexpr

term -> factor rterm

factor -> id

[relop] <=

rterm -> ε

rexpr -> ε

rboolean -> relop expr

[num] 2

expr -> term rexpr

term -> factor rterm

factor -> num

[times] \*

rterm -> \* factor rterm

[lpara] (

factor -> ( expr )

[id] i

expr -> term rexpr

term -> factor rterm

factor -> id

[plus] +

rterm -> ε

rexpr -> + term rexpr

[num] 1

term -> factor rterm

factor -> num

[rpara] )

rterm -> ε

rexpr -> ε

[rpara] )

rterm -> ε

rexpr -> ε

[id] sum

stmt -> id = expr

[assign] =

[num] 1

expr -> term rexpr

term -> factor rterm

factor -> num

[semi] ;

rterm -> ε

rexpr -> ε

[else] else

else\_part -> else stmt

[id] sum

stmt -> id = expr

[assign] =

[num] 0

expr -> term rexpr

term -> factor rterm

factor -> num

[semi] ;

rterm -> ε

rexpr -> ε

[id] i

stmts -> stmt stmts

stmt -> id = expr

[assign] =

[id] i

expr -> term rexpr

term -> factor rterm

factor -> id

[plus] +

rterm -> ε

rexpr -> + term rexpr

[num] 2

term -> factor rterm

factor -> num

[semi] ;

rterm -> ε

rexpr -> ε

[rbrace] }

stmts -> ε

[rbrace] }

stmts -> ε

1. **实验总结**

通过本次实验，我收获了很多东西。首先对编译原理这门课有了进一步的深刻理解，同时对LL(1）文法分析的原理和过程有了进一步的巩固，也锻炼了我编程的能力，巩固了平时所学的知识，真正做到了学以致用。

在做实验的过程中，发现自己在编写程序过程中，总是会忽略各种细节，从而导致经常修改一些很小的低级错误才能使程序正常运行，不仅浪费时间，还影响对其他地方的修改，并且在很多步骤处理上，方法不正确。使结果不能符合要求，深刻体会到了自己在编程方面与别人的差距，在今后的学习中，我会注意改正自己在这方面的缺点，促使自己的编程水平不断进步。

编译原理是一门专业学科，对于现阶段的我来说，只能掌握它的一些基本原理和概念，对于一些更深层的知识还是有很多难以理解的地方。但在这次实验过程中，锻炼了自己的思考能力，也锻炼了自己的动手编程能力，对于将知识的转化有了很大的帮助。