四则运算题目生成程序编程设计(3h-10/23, 1h-10/24)

//guarating mode is right there rather than in CMode because of CMode’s data struction. So you can not create a CMode except ReadEquation function.

1. ReadEquation(string s)

if s.empty:

return false

//deal with char of s in order

For ch in s:

//ch is one of {0~9, ’ , |}

If isFraction(ch) or ch==’(‘:

If prvsIsFraction:

Return error

If ch==’(‘:

CMode.takeInOperator(ch)

//do not change prvs state because a fraction is needed after (

continue

I=0

While(isFraction)

//a fraction contain no more than 5 int

If i>=3 : Error

If isInteger(ch):

If prvsIsInteger:

rgNum[i] = rgNum[i]\*10 + int(ch)

Else:

rgNum[i++]=int(ch)

else:

assert ch==’ ’ ’ && i==1

assert ch==’|’ && i==2

ch++;

if i==1:

setRestOfArray()

//serve as a flag show that accept a correct fraction

i=3

if i==3:

CMode.takeInFraction( CFraction(\*rgNum) )

setPrvsAsFraction()

Else： Error

Else if isOperator(ch) or ch==’)’:

Assert prvsIsFraction

If ch==’)’:

CMode.takeInOperator(ch)

//do not change prvs state because a operator is wanted after )

Else:

CMode.takeInOperator(ch)

setPrvsAsOperator()

Else

Return error.

1. Class Fraction

Private:

//guaranting that fraction is in lowest terms.

m\_int, m\_num, m\_deno, m\_type;

public:

Fraction(m\_int, m\_num, m\_deno)

//

setValue(m\_int, m\_num, m\_deno)

,

setValue(m\_int, m\_num, m\_deno)

//determining m\_type according to inputs.

//return error if m\_deno is 0.

//Reduction of this fraction.

Self.reduction()

isInt()

//reduction that guarating it is in lowest terms.

// TODO : whether we can omit this part

Self.reduction()

//judging whether it is a integer.

If m\_num==0:

Return true

Reduction()

//what if deno is minus

If m\_deno<0:

m\_deno = -m\_deno

m\_num = m\_num

GCD = getGCD(m\_num,m\_deno)

//reduction

m\_num /=GCD

m\_deno/=GCD

m\_int += m\_num/m\_deno

m\_num -= m\_num/m\_deno \* m\_deno

//what if m\_num is minus

If m\_num < 0:

// m\_num must be great than minus m\_deno

m\_num += m\_deno

m\_int -=1

//set denominator to 1 if m\_num is 0 to reduce computation.

equalToFraction(temp)

//assume temp is in lowest terms

If (m\_int, m\_num, m\_deno) == temp

Return true

// TODO : there are two possible solution. We can deal with this according two fractions are integer or not. And we can also transform it into improper fraction. Here we implement the latter one.

operator +

getImproperFraction(m\_int, m\_num,m\_deno,&num1,&deno1)

fetImproperFraction(temp.m\_int, temp.m\_num, temp.m\_deno,&num2,&deno2)

//outDeno is two denos’ LCM. Of it could be zero. But it is be prevented when init a new fraction to return .

outDeno= getLCM(deno1,deno2)

outNum = num1\*deno/deno1 + num2\*deno/deno2

// reduction of this can be down by self.reduction function

CFraction(0,outNum,outDeno)

operator –

//like what did in operator +

getImproperFraction(m\_int, m\_num,m\_deno,&num1,&deno1)

fetImproperFraction(temp.m\_int, temp.m\_num, temp.m\_deno,&num2,&deno2)

outDeno= getLCM(deno1,deno2)

outNum = num1\*deno/deno1 - num2\*deno/deno2

// reduction of this can be down by self.reduction function

CFraction(0,outNum,outDeno)

operator \*

//transform the fraction into improper fraction

getImproperFraction(m\_int, m\_num,m\_deno,&num1,&deno1)

fetImproperFraction(temp.m\_int, temp.m\_num, temp.m\_deno,&num2,&deno2)

//TODO : what if temporary result overflow?

outNum = num1 \* num2

outDeno = deno1 \* deno2

CFraction(0, outNum, outDeno)

operator /

getImproperFraction(m\_int, m\_num,m\_deno,&num1,&deno1)

fetImproperFraction(temp.m\_int, temp.m\_num, temp.m\_deno,&num2,&deno2)

outNum = num1 \* deno2

outdeno = num2 \* deno1

CFraction(0, outNum, outDeno)

operator ==

self.equalToFraction()

1. Class CToken

Private:

m\_op;

m\_frac;

CToken(ch)

M\_op = ch;

CToken(Fraction)

M\_op = number;

M\_frac = Fraction

getOp()

return m\_op;

getFrac()

return m\_frac;

1. CMode:

Private:

List <CToken> expression.

Public:

takeInFraction(Fraction)

assert (not readFinished)

expression.push\_back ( CToken(Fraction) )

takeInOperator(ch)

assert (not readFinished)

expression.push\_back( CToken(ch))

clear()

expression.clear()

//set a begin indication for later convenience.

expression.push\_back(CToken(expressionBegin))

//once we set readFinished true ,we cannot try to add fraction or operator //into CMode.

//TODO : what if we want to add expression again. resetReadFinished ?

setReadFinished()

readFinished = True

next = expression.begin

prvs = NULL

getToken()

assert readFinished

//the end signal of this expression

if next==expression.end()

return CToken(expresEnd)

else

prvs = next

next ++

return \*prvs

pushToken()

assert prvs!=NULL

//we do not push anything if it is the end of the expression

if next==expression.end()

continue

//push a token bake means point next go back a site.

next—

//expresstion.begin can not use -- operate.

// this will not be use when calculating a expression

if prvs==expression.begin:

prvs = NULL

else prvs--

1. Expression() term() primary()

Expression(mode)

Left = term()

While(1)

Token = mode.getToken()

If(Token.getOp() == ‘+’) left = left + term()

Else if(Token.getOp() == ‘-‘) left = left – term()

Else mode.pushToken();return left. //here mainly for (primary)

Term(mode)

//The same as expresstion except term() which is replaced by primary()

Primary(mode)

Token = mode.getToken()

If Token.getOp() == ‘(‘:

Temp = expression()

Token = mode.getToken()

If Token.getOp() == ‘)’

Retur temp;

Else

Erroe

Else if token.getOp()==number

Return Token.getFrac

Else

error