对于这种结对的工作，由于有过电子设计实践的基础，大概知道建一个工程需要做的事，有点经验还是有帮助的。

一、问题要求：

1·主要功能是随机产生有效的运算式。

2·可以学霸或者老师选择运算式的要求，比如：运算式的个数，运算式的长度，运算符的种类，运算数的大小范围，运算数的种类（整数，小数，分数）。

3·当自己产生的式子得自己算,然后给出运算式和结果。

4.和UI对接，这也是这次作业的关键，我们要接受来自UI的设置参数，并将表达式和结果以API形式传给UI。

二、分析问题与处理思想

分析

一、 如何产生合法的表达式：

1. 产生的东西只有数与符
2. 数字的运算都是加减乘除乘方
3. 采用什么方式产生表达式才能合法？逆波兰式？波兰式？表达式二叉树？

二、 如何分类以及如何设计API

1. 这个项目的主要模块是：设置参数模块、表达式二叉树结点模块、表达式模块
2. 这些模块之间的耦合和模块类型？类？结构体？
3. API的设计的关键是我们要把数据以什么形式传给UI，文件？xml？指针

某鑫的思考：

在最初思考建立表达式中间有一环特别烦，那就是不能产生重复的表达式，所谓的重复，呵呵呵，关键一开始还有一种不重复的情况我都没有理解清楚，直到想到了中缀表达式所对应的二叉树的时候才反应过来啥是重复，啥是不重复。直到这个时候，心中才有了眉目，应该怎么去实现随机算式的产生以及判断重复。

因为要将符号与数生在一棵树上，所以将二者统一为一种类，数字和字符随机选择，选择完毕在随机选择更加细致的内容：数字的大小，运算符的种类。

三种数的产生与储存和运算处理

首先分数怎么储存？2/3怎么放？13/37怎么放？没办法，就这么放了，分子放一边分母放一边。那么检查检查整数和小数可不可以融进分数这样的结构中？不能兼容也得兼容呀，不然我的树在建立的时候还得分是分数树还是非分数树？那么整数和小数区别于分数在于没有分母（分母为1）这是关键，至关重要哟，小数区别于整数的地方就是...就是...哎呀，你选择保留几位小数之后就不去别了嘛，就只有在除法的时候麻烦一点，判断整除约分处理的时候注意一下下就好了。

数字的储存统一格式之后就是一定要重载各种运算符了。

计算和产生表达式都是后序遍历二叉树的结果，所以就写在一起。

某林的思考：

如何区分子模块，将子模块写成类还是结构体，模块与模块之间的联系是什么是我拿到这个题目后最先想的问题。首先第一次真正的写C++，要将什么定为一个类是一个难题。鉴于我们已经讨论好用表达式二叉树储存表达式，那么对象也渐渐清晰。

首先树肯定是一个类，也就是表达式，但是我们在表达式这个层面，更多的是基于树与树之间的操作，而不是单棵树的操作，比如说检查表达式是否重复，于是就将这个类定为产生一系列的树generate

那么树的结点也是一个类，这个结点可以是运算符，也可以是运算数，里面存放着关于数和运算符的所有信息，之后的+-\*/^这些运算都是这个类的方法，包括约分等等，这个类是计算的重点，node

还有一个设置参数模块，这个模块应该用结构体还是类？鉴于这个结构体会被generate这个类和node类调用，我们就暂时把它设为结构体。但事实上，我们后来需要基于这个类的信息处理一些函数，并且这些函数与node和generate没有直接关系，于是又把它建成了类。类setting。

这三个类的关系很简单，generate类调用node类的方法和setting类的方法，node类调用setting类的方法。node 并不会调用generate，setting 则不会调用其他的类。

事实证明，这个结构划分给我们的程序已比较简洁明了的层次感，也降低了出现致命错误的概率。

三、PSP表格：



四、源码展示：

数据结构：

Setting类：

主要用于参数设置和修改，以及基于参数做一些纯数学处理。

class Setting:exception

{

private:

int expnumber; //生成表达式的个数

int operator\_account; //操作符数量

int operation; //操作符对应函数值

bool is\_proper\_fraction; //是否支持真分数运算

bool is\_decimal; //是否支持小数运算

int accuracy; //精度

int range[2]; //范围

struct Operate { //操作符选择//这个好烦啊

bool add;

bool sub;

bool mul;

bool div;

bool pow;

};

Operate operate;

public:

int funcOperate();

int numbertype();

std::string load(double number); //将int 型转换成char\*数组

Setting(int ExpNumber, int operator\_number,int Accuracy, bool fraction, bool decimal, int min, int max, bool Add, bool Sub, bool Mul, bool Div, bool Pow);

int getRange\_min();

int getRange\_max();

int getAccuracy();

int getOperator\_account();

bool getIs\_proper\_fraction();

bool getIs\_decimal();

int getExpnumber();

void init(int ExpNumber, int operator\_number, int Accuracy, bool fraction, bool decimal,int min, int max, int operator\_mode);

Setting();

void setOperate\_pow(bool value);

void setRange(int min, int max);

};

node类：

二叉树表达式结点，存放运算符或操作数，方法主要是结点运算等。

class node

{

private:

int operate; //运算符的类型 0表示不是运算符 1表示+ 2表示- 4表示\* 8表示/ 16表示^

bool type; //1是操作符 0是操作数

//int numtype; //设置操作数类型,整数为0，小数为1， 真分数为2

float up; //分子

long long down; //分母

public:

node(bool Type, int operate, double up, long long down);

node();

~node();

node\* leftptr; //左指针

node\* rightptr; //右指针

void setNode(bool type, Setting setting); //设置结点类型,并生成结点数据

void setNode(bool Type, double Up, long long Down, int Operate);

void setNodePow(); //设置乘方结点

void geneNode(Setting setting); //生成结点类型，并生成结点数据

bool getType(); //获取结点类型

double getup(); //获取分子

long long getdown(); //获取分母

int getOperate(); //获取操作符

bool operator\_is\_div(); //判断是否为除法

bool operator\_is\_pow(); //判断是否为乘方

bool operator==(node &num2); //判断对象是否相等

node operator+(node &num2);

node operator-(node &num2);

node operator\*(node &num2);

node operator/(node &num2);

node power(node &right);

bool judge\_node(); //判断结点是否非法 //也就是分母

std::string num2str(Setting setting); //将真分数转化为char型数组

int judge\_priority(node node2); //判断优先级

std::string transform\_Operate(); //操作符转换

void simplify(); //约分

};

generate类：

产生一系列表达式，检查是否重复……

typedef struct Expression {

string expression;

node consequence;

}Expression;

class generate

{

private:

Setting setting;

vector<node\*> TreeList; //用vector存放表达式指针的数组//或者用什么其它的链表之类的

node\* creExpression(node\* Nptr, int &onum); //递归生成一棵表达式二叉树

bool checkrepeat(node\* Nptr, Expression &exp); //检查是否重复

bool checkRepeatT(node\* ptr, node\* rootptr); //检查树是否重复

void addTree(node\* rootptr); //把树放到vector中

void addExpression(Expression expression); //把表达式放到vector

Expression getExpression(node\* rootptr); //中序遍历得到表达式和值

node getValue(node left, node right, int Operate); //计算

public:

generate();

~generate();

bool set(int ExpNumber, int operator\_number, int Accuracy, bool fraction, bool decimal,int min, int max, int operator\_mode); //设置参数结构体

vector<Expression> ExpressionList; //表达式数组

void setRootnode(node\* rootptr, Setting setting); //设置根结点参数

void expression(); //创建一系列表达式（一堆二叉树，并用vector存放）

void show(); //显示表达式

void consequence(); //最终结果

void str\_cat(Expression &dis, Expression src1, Expression src2); //表达式的链接

};

接下来是主要代码：

随机产生结点：

void node::setNode(bool type1, Setting setting) {

srand((unsigned)time(NULL));

if (type1 == OPERATOR) {

type = OPERATOR;

int op = setting.funcOperate();

int op1;

do {

op1 = (int)pow(2, rand() % 5);

} while (!(op1 & op));

operate = op1;

up = UP;

down = DOWN;

}

else if (type1 == NUMBER) {

type = NUMBER;

operate = OPERATE;

int num = setting.numbertype();

switch (num) {

case 0:

{

up = setting.getRange\_min() + rand() % (setting.getRange\_max() - setting.getRange\_min() + 1);

down = 1;

}

//真分数

case 1:

{

up = setting.getRange\_min() + rand() % ((setting.getRange\_max() - setting.getRange\_min() + 1));

down = setting.getRange\_min() + rand() % ((setting.getRange\_max() - setting.getRange\_min() + 1));

}

case 2:

{

up = setting.getRange\_min() + (rand() % ((setting.getRange\_max() - setting.getRange\_min())\*setting.getAccuracy() + 1) / setting.getAccuracy());

down = 1;

}

}

}

}

void node::geneNode(Setting setting) {

srand((unsigned)time(NULL));

type = rand() % 2;

if (type == OPERATOR) {

int op = setting.funcOperate();

int op1;

do {

op1 = (int)pow(2, rand() % 5);

} while (!(op & op1));

operate = op1;

up = UP;

down = DOWN;

}

else if (type == NUMBER) {

operate = OPERATE;

int num = setting.numbertype();

switch (num) {

case 0:

{

up = setting.getRange\_min() + rand() % (setting.getRange\_max() - setting.getRange\_min() + 1);

down = 1;

break;

}

//真分数

case 1:

{

up = setting.getRange\_min() + rand() % ((setting.getRange\_max() - setting.getRange\_min() + 1));

down = setting.getRange\_min() + rand() % (setting.getRange\_max() - setting.getRange\_min() + 1);

break;

}

case 2:

{

up = setting.getRange\_min() + (double)(rand() % ((setting.getRange\_max() - setting.getRange\_min())\*setting.getAccuracy() + 1)) / (double)setting.getAccuracy();

down = 1;

break;

}

}

}

}

驾驶员：林静雯　　领航员：李鑫

主要功能就是随机产生结点，或者在一定要求下随机产生结点

建树：

node\* generate::creExpression(node\* Nptr, int & onum) {

int min = 0;

int max = 0;

if (!Nptr) return NULL;

//左结点

Nptr->leftptr = new node;

if ((\*Nptr).operator\_is\_pow()) {

setting.setOperate\_pow(false);

min = setting.getRange\_min();

max = setting.getRange\_max();

setting.setRange(setting.getRange\_min(), setting.getRange\_min()+10);

}

//操作符数目还未到达上限

if (onum <= setting.getOperator\_account()) {

Nptr->leftptr->geneNode(setting); //生成左结点类型

//如果结点类型为操作符

if (Nptr->leftptr->getType() == 1) {

onum++; //操作符数目加一

creExpression(Nptr->leftptr, onum); //递归生成左子树

}

}

//操作符已达上限

else {

Nptr->leftptr->setNode(NUMBER, setting);

}

//右结点

Nptr->rightptr = new node();

//如果操作符是乘方

if ((\*Nptr).operator\_is\_pow()) {

Nptr->rightptr->setNodePow();

}

//操作符数目未达上限

else if (onum <= setting.getOperator\_account()) {

Nptr->rightptr->geneNode(setting); //生成右结点类型

//如果结点类型为操作符

if (Nptr->rightptr->getType() == 1) {

onum++; //操作符数目加一

creExpression(Nptr->rightptr, onum); //递归生成右子树

}

}

//操作符数目达到上限

else {

Nptr->rightptr->setNode(NUMBER, setting);

}

if ((\*Nptr).operator\_is\_pow()) {

setting.setOperate\_pow(true);

setting.setRange(min, max);

}

return Nptr;

}

驾驶员：林静雯　　领航人：李鑫

主要就是产生表达式==

遍历树来产生表达式和计算数值：

Expression generate::getExpression(node\* rootptr) {

if (!rootptr) {

node num3;

Expression Express;

Express.consequence = num3;

Express.expression = {};

return Express;

}//虽然可能这种情况不存在的

if (!rootptr->getType()) {

Expression Express;

Express.consequence = \*rootptr;

Express.expression = Express.consequence.num2str(setting);

return Express;

}

Expression Express;

Express.consequence = (\*rootptr);

Expression left = getExpression(rootptr->leftptr);

if (left.consequence.getup() > MAX\_NUM || left.consequence.getdown() > MAX\_NUM) {

left.consequence.setNode(NUMBER, UP, 0, OPERATE);

}

Expression right = getExpression(rootptr->rightptr);

if (right.consequence.getup() > MAX\_NUM || right.consequence.getdown() > MAX\_NUM) {

right.consequence.setNode(NUMBER, UP, 0, OPERATE);

}

//子树结果出现非法

if (!left.consequence.judge\_node()) {

return left;

}

if (!right.consequence.judge\_node()) {

return right;

}

if ((\*rootptr).operator\_is\_div() && right.consequence.getup()) {//除法

if (!setting.getIs\_decimal() && !setting.getIs\_proper\_fraction()) {//仅支持整数

if (left.consequence.getup() / right.consequence.getup() != (long long)(left.consequence.getup() / right.consequence.getup())) {//不整除

long long makeup = ((long long)left.consequence.getup()) % ((long long)right.consequence.getup());

makeup = (long long)right.consequence.getup() - makeup;

node\* new\_operate = new node;

node\* new\_number = new node;

(\*new\_operate).setNode(OPERATOR, setting);

(\*new\_number).setNode(NUMBER, setting);

(\*new\_operate).setNode(OPERATOR, UP, DOWN, ADD);

(\*new\_number).setNode(NUMBER, makeup, DOWN, NUMBER);

(\*new\_operate).rightptr = new\_number;

(\*new\_operate).leftptr = rootptr->leftptr;

rootptr->leftptr = new\_operate;

string c\_num = setting.load(makeup);

if (c\_num == "wrong") {

Express.consequence.setNode(NUMBER, UP, 0, OPERATE);

return Express;

}

left.expression = left.expression + new\_operate->transform\_Operate() + c\_num;

left.consequence.setNode(NUMBER, left.consequence.getup() + makeup, DOWN, ADD);

}

}

}

Express.consequence = getValue(left.consequence, right.consequence, rootptr->getOperate());

if (Express.consequence.getup() < 0) {

Express.consequence.setNode(NUMBER, -Express.consequence.getup(), Express.consequence.getdown(), Express.consequence.getOperate());

node\* tem;

tem = rootptr->leftptr;

rootptr->leftptr = rootptr->rightptr;

rootptr->rightptr = tem;

str\_cat(Express, right, left);

}

else {

str\_cat(Express, left, right);

}

if (setting.numbertype() == 2) {

double tem = Express.consequence.getup();

tem /= Express.consequence.getdown();

Express.consequence.setNode(NUMBER, tem, DOWN, Express.consequence.getOperate());

}

else {

Express.consequence.simplify();

}

return Express;

}

驾驶员：李鑫　　领航人：林静雯

整除的处理由于害怕大概率出现的减法和除法混合的话，很容易出现无法小规模修改的情况让出现a/0这样小学生无法处理的东西，所以无法整除的时候由原来的减去余数改成加上余数的补数，出现负数的时候直接交换左右子树，变负为正。

如果真的出现非法的式子：分数次幂，除数为0，数字太大超过10位数，则强行将这棵树废掉（将这棵树最后的返回结果的分母置为0，即无意义）

关键的判断重复：

bool generate::checkrepeat(node\* rootptr, Expression &exp) {

for (Expression exp1 : ExpressionList) {

if (exp1.consequence == exp.consequence)

return true;

}

for (node\* ptr : TreeList) {

if (checkRepeatT(ptr, rootptr)) return true;

}

return false;

}

bool generate::checkRepeatT(node\* ptr, node\* rootptr) {

//如果都是运算符

if (ptr == NULL && rootptr == NULL) return true;

if (ptr == NULL || rootptr == NULL) return false;

if (\*ptr == \*rootptr) {

return checkRepeatT(ptr->leftptr, rootptr->leftptr) && checkRepeatT(ptr->rightptr, rootptr->rightptr) || checkRepeatT(ptr->leftptr, rootptr->rightptr) && checkRepeatT(ptr->rightptr, rootptr->leftptr);

}

else return false;

}

驾驶员：林静雯　　领航员：李鑫

重载了==运算，外加利用string类的==运算，很方便的判断出了重复

对外唯一接口的设定和若干入口检验：

bool generate::set(int ExpNumber, int operator\_number, int Accuracy, bool fraction, bool decimal, int min, int max, int operator\_mode) {

fstream setfile;

setfile.open("setting", ios::out);

//表达式个数

setfile << "ExpNumber:" << endl;

if (ExpNumber > MAX || ExpNumber < 0) {

ExpNumber = EXPNUMBER;

setfile << "false" << endl;

}

else setfile << "true" << endl;

setfile << ExpNumber << endl;

//操作符个数

setfile << "operator\_number:" << endl;

if (operator\_number > OPERATOR\_NUMBER || operator\_number < 0) {

operator\_number = OPERATOR\_NUMBER;

setfile << "false" << endl;

}

else setfile << "true" << endl;

setfile << operator\_number << endl;

//小数位数

setfile << "Accuracy:" << endl;

if (Accuracy < 0 || Accuracy >3) {

setfile << "false" << endl;

Accuracy = ACCURACY;

setfile << Accuracy << endl;

}

else {

setfile << "true" << endl;

setfile << Accuracy << endl;

Accuracy = (int)pow(10, Accuracy);

}

//分数和小数的设置

if (fraction && decimal) {

setfile << "fration:" << endl;

setfile << "false" << endl;

fraction = false;

setfile << "no" << endl;

setfile << "decimal:" << endl;

setfile << "false" << endl;

decimal = false;

setfile << "no" << endl;

}

//分数

else {

setfile << "fraction:" << endl;

setfile << "true" << endl;

if (fraction) setfile << "yes" << endl;

else setfile << "no" << endl;

//小数

setfile << "decimal:" << endl;

setfile << "true" << endl;

if (decimal) setfile << "yes" << endl;

else setfile << "no" << endl;

}

//最小范围数

setfile << "minnum:" << endl;

if (min > max || min>MAX || min<0) {

min = MIN;

setfile << "false" << endl;

}

else setfile << "true" << endl;

setfile << min << endl;

//范围最大数

setfile << "maxnum:" << endl;

if (max > MAX || max <= 0) {

max = MAX;

setfile << "false" << endl;

}

else setfile << "true" << endl;

setfile << max << endl;

setfile << "operator\_mode:" << endl;

//运算符格式

if (operator\_mode < 0 || operator\_mode >4) {

operator\_mode = OPERATOR\_MODE;

setfile << "flase" << endl;

setfile << "+-\*" << endl;

}

else {

setfile << "true" << endl;

switch (operator\_mode) {

case 0: setfile << "+" << endl;

break;

case 1: setfile << "+-" << endl;

break;

case 2:setfile << "+-\*" << endl;

break;

case 3:setfile << "+-\*/" << endl;

break;

case 4:setfile << "+-\*/^" << endl;

break;

default:

break;

}

}

setfile << operator\_mode << endl;

setfile.close();

setting.init(ExpNumber, operator\_number, Accuracy, fraction, decimal, min, max, operator\_mode);

return true;

}

　驾驶员：林静雯　　领航员：李鑫

将数字方便的转化成我们想要的样子：

string node::num2str(Setting setting)

{

string upc;

upc = setting.load(up);

if (upc == "wrong") {

down = 0;

return "wrong";

}

if (down == 1) {

return upc;

}

else {

string downc;

downc = setting.load(down);

if (downc == "wrong") {

down = 0;

return "wrong";

}

string express = upc + "//" + downc;

return express;

}

}

string Setting::load(double number)

{

string c;

number \*= accuracy;

bool x = !(accuracy == 1);

number = (number - (long long)number > 0.5) ? ceil(number) : floor(number);

number /= accuracy;

c = to\_string(number);

int a = (int)log10(accuracy);

if ((long long)number) {

try {

long b = (long)log10((long long)number);

b += 1 + x + a;

c.erase(b, c.size());

}

catch (exception &e) {

return"wrong";

}

return c;

}

else c.erase(1 + x + a, c.size());

return c;

}

　　驾驶员：李鑫　　领航员：林静雯

 五、对接部分：

两种对接方式：

第一种是输出文件：

接口就是只有一个设置函数：

extern "C++" \_declspec(dllexport) bool set(int ExpNumber, int operator\_number, int Accuracy, bool fraction, bool decimal, int min, int max, int operator\_mode);

第二种是输出内存：

需要结构体和一个函数：

struct Answer {

string\* consequnce;

string\* express;

};

extern "C++" \_declspec(dllexport) Answer set(int ExpNumber, int operator\_number, int Accuracy, bool fraction, bool decimal, int min, int max, int operator\_mode);

两种方法均测试成功。

写在最后：

某林：

emmmm终于结束了。不过收获还是超级大。

第一是第一次写C++/真正意义上的那种，虽然没有用到继承/重载/泛型等比较复杂的技巧，但终于是有点理解对象的设定和相关信息的封装，比如说，要怎么才能让generate不必知道node里到底发生了什么，也就是所谓的避免类与类之间的语义耦合把，虽然还是不能做到完完全全的避免。但我们至少做到了类之间的调用关系相对简单这个点。不足的是我觉得类的划分还可以再细一点，因为有的类的方法太多了。

第二是如何比较快速的Debug，这也是这次的收获，而且这要归功于我的队友李鑫2333之前虽然也知道条件断点之类的，但都没有实际用过，出了问题也不知道如何迅速排查，没有跟进程序的思维和思路，一遇到程序终止就手足无措，之前的Debug虽然自己也用，但没有实际效率。看了队友的调试过程经过了一周的学习，终于也能比较快地发现了程序的错误。

第三是更好理解了封装和对接的含义。

封装有利于修改程序，有利于代码维护、有利于鼓起勇气面对比较长的代码，而API是合作的关键，是子系统与子系统之间对接的关键。如何不把自己的数据结构暴露给UI是我们需要思考的问题。

总之是很爆炸的一周也是收获很多的一周，感谢队友!!!!!!