程式語言期末專題

Linear Algebra Tool

GuoJTim

章節

前言

本篇期末專題的成品

Linear Algebra Tool

本專題靈感非常淺顯易懂，其成品內容也相當的好理解，本專題以解線性代數有關的工具為其實際功能，內容包含矩陣與其相關的運算功能。

詳細功能

上方提到此專題(程式)包含了矩陣等相關運算功能，在此介紹其詳細功能：

1. 高斯刪去法 和 其化簡的詳細過程
2. 高斯喬登刪去法 和 其化簡的詳細過程
3. 給定以多個不同變數的線性方程式求其解 和 其求解過程
4. 包含解集合(無限多種解案例) 和 其解法的詳細過程
5. 算逆矩陣

最後在以 LaTeX 方式輸出其算法。

程式語言相關技術

雖然這單單只是一個擁有多功能的程式，但每個功能下都包含許多的技術，以 LaTeX 舉例，需要用到類似編譯器的功能，需要讀輸入的程式在講解什麼最後重新組合一個語言(LaTeX)出來如果說每項功能都單一去寫，雖然會好寫很多，但是相對的檔案會很多，如果將此都集合在一個程式裡面，是可以方便許多，缺點就是需要一個很完整的系統架構去支撐，不然到最後會不知道功能，因此會利用到類別上的繼承與組合等關係，和其運算子多載、STL 等。

章節

實務專題

這專題需要一個完整的系統，故在此以實作報告展示，也就是每一個類別與其功能都會拿來做解釋，在此分以下類別(category)來講。

LongNum.h

在未來算消去法時，可能會遇到各種數字，單純的 Long 或 Long Long 想必是不夠的，因此會利用到大數運算等功能。

Num.h

因為在 LongNum 裡面定義的除法不帶有小數點，且 LongNum 為一「整數」大數，故在 Num 裡補充其的不足，將數字型態化為分數型式，而在分數型式下，也包含兩分數的加減乘除等功能。

Matrix.h

定義完分數型態的類別之後，就可以以此類別當作矩陣裡的每一個元素(element)，而在此矩陣以列運算(row operation)為基底，之後會有相對應的延伸，例如擴增矩陣(augmented matrix)、逆矩陣(InverseMatrix)等功能。
而在 Matrix.h 裡面有一個 Augment Matrix 類別繼承 Matrix.h，Augment Matrix 顧名思是就是擴增矩陣，同樣能使用在之後的高斯消去或高斯喬登消去法裡面。

Elimination.h

高斯消去法與高斯喬登刪去法皆在此進行，輸入一個矩陣，輸出其削減過程，取獲得其解

LinearEquation.h

能輸入多個線性方程式，並且解出其線性方程式的解，若答案為無解顯示無解、無限多組解顯示其解集合、只有一組解則顯示出該線性方程式之解。

InverseMatrix.h

以 Matrix 為基底並且合併一個單位矩陣，用高斯喬登刪去法求其逆矩陣之解。

Latex.h

將過程都轉為 LaTeX，能以代碼的方式來表示數學公式。
因此這個 LaTeX 類別基本上是貫穿整個系統，系統裡的每一個步驟都會由這個類別來記錄所做的一切，最後在以 LaTeX 輸出。

大數運算

需求功能

需要包含支援無限上的大數做加法、減法、除法、乘法、模數且可以比大小等運算

資料成員

有兩個資料成員，分別是布林代數的 minus 與 int 儲存的不定長陣列 vector 的 v。minus 用來代表此 LongNum 的正負，預設值為 false 即代表正數。V 代表此 LongNum 的數字(digit)陣列。

建構子

在 LongNum 的類別裡有三個建構子，可以支援字串輸入、長長整數輸入、vector 輸入。

```
LongNum(){
	vector<int> v(1,0);
}
LongNum(const LongNum & n){
	minus = n.minus;
	v = vector<int>(n.v);
}
LongNum(int num){
	if(num == 0) v.push_back(0);
	if(num < 0){
		minus = true;
		num = num * -1;
	}
	for(;num;num/=10)v.push_back(num%10);
	if(v[0] == 0 && v.size() == 1) minus = false;
}
LongNum(string s){
	for(int i = s.length()-1 ; i >= 0 ; i--) {
		if (s[i] == '-') minus = true;
		else v.push_back(s[i]-'0');
	}
	if(v[0] == 0 && v.size() == 1) minus = false;
}
```

```
LongNum(vector<int> s){
	v = vector<int>(s);
	for(int i = v.size()-1;i>=0;i--){
		if(v[i] == 0) v.pop_back();
		else break;
	}
	if(v.size() == 0){
		v.push_back(0);
	}
	if(v[0] == 0 && v.size() == 1) minus = false;
}
```

字串輸入與長長整數輸入是利用在之後可以直接以這兩種形式去定義一個 LongNum 的物件。

vector 輸入則是之後在做加減乘除時的運算子重載可以直接定義一個新的 vector 最後直接輸出，在此需要知道的是紀錄數字的 v 是顛倒過來的，也就是說數字 156 在 vector 裡面是 651。

大小運算子重載

先寫比 LongNum 大小的部分。在寫大小的運算子重載時輸入部分分為兩類，長長整數與 LongNum 型態的輸入。

在此以大於的多載(長長整數與 LongNum 輸入)來做展示：

```
bool operator> (const LongNum & L){
	if (minus && !L.minus)return false;
	else if (!minus && L.minus) return true;
	else{
		if(v.size() > L.v.size()){//length of *this > length of L
			return true ^ minus;
			// xor(result,minus)
			// -1000 < -1  =>  len(*this) > len(L) but minus = 1
		}else if(v.size() < L.v.size()){
			return false ^ minus;
		}else{
			//len(*this) == len(L)
			//compare each digit
			int c = v.size()-1;
			for(;c >= 0;c--){
				if(v[c] > L.v[c]) return true ^ minus;
				else if(v[c] < L.v[c]) return false ^ minus;
			}
			if(c == -1) return false; // *this == L
		}
	}
}
bool operator< (long long int n){
	return operator<(LongNum(n));
}
```

小於的寫法也相同只是 true 變成 false。

至於大於等於和等於的運算子也很簡單：

```
bool operator== (const LongNum & L){
	if(v.size() != L.v.size() || minus != L.minus) return false;
	int c = v.size()-1;
	for(;c>=0;c--){
		if(v[c] != L.v[c]) return false;
	}
	return true;
}
bool operator== (long long int n){
	return operator==(LongNum(n));
}
bool operator>= (const LongNum & L){
	return !operator<(L);
}
bool operator>= (long long int n){
	return !operator<(LongNum(n));
}
```

加減法運算子重載

接著是加法與減法的運算，這是之後除法和模數的關鍵。除法使用基本的長除法，而長除法則是需要用到減法的觀念，其求模數也相同概念。

而在寫加法與減法時應該會是息息相關的，可以看以下例子：

4+6=10 | 4+(-6) = -2 | 6+(-4) = 2 | -4+(-6) = -10

可以推出，如果兩個數的正負號都相同則執行加法(最後在看原本的加減法補上負數)，而看到中間兩個例子，有一個很明顯的正負區分的話則是執行減法(最後在依照兩數較大者給予正負號)

下方函式為執行兩整數 vector 的加法(不考慮正負號)

```
vector<int> addition(const vector<int> & augend,const vector<int> & addend){
	int c = 0;
	vector<int> newn;//new number
	int i;
	for(i = 0 ; i < min(augend.size(),addend.size());i++){
		if(augend[i]+addend[i]+c > 9){
			newn.push_back(augend[i]+addend[i]+c-10);c = 1;
		}else{
			newn.push_back(augend[i]+addend[i]+c);c = 0;
		}
	}
	for(;i < max(augend.size(),addend.size());i++){
		if(augend.size() > addend.size() && augend[i]+c > 9) {
			newn.push_back(augend[i]+c-10);c = 1;
		} else if(augend.size() < addend.size() && addend[i]+c > 9){
			newn.push_back(addend[i]+c-10);c = 1;
```

```
            }else if(augend.size() > addend.size()){
                newn.push_back(augend[i]+c);c = 0;
            }else {
                newn.push_back(addend[i]+c);c = 0;
            }
        }
        if (c) newn.push_back(1);
        return newn;
}
```

加法的原理也叫好講解，因為之前定義在 LongNum 裡面紀錄每個數字(digit)的陣列是顛倒的，故可以以一個 int c 代表其進位，如果有進位則加 1，依此類推去推結果。下方則是在此情況：假設兩數 1 與 999，指標指標在 1 的位置已經沒有了而 999 還有兩個 9 尚未被讀到，因此會繼續做下去。

而減法的程式碼如下：

```
vector<int> subtraction(const vector<int> & minuend,const vector<int> & subtrahend){
        // minuend - subtrahend
        // to note , minuend > subtrahend
        //
        int i = 0;
        int c = 0;
        bool iszero = true;
        vector<int> newn;//new number
        for(; i < subtrahend.size();i++){
            if (minuend[i] < subtrahend[i]+c){
                iszero = (10-c-subtrahend[i]+minuend[i] == 0 && iszero);
                newn.push_back(10-c-subtrahend[i]+minuend[i]);
                c=1;
            }else{
                iszero = (minuend[i]-subtrahend[i]-c == 0 && iszero);
                newn.push_back(minuend[i]-subtrahend[i]-c);
                c = 0;
            }
        }
        for(; i<minuend.size();i++){
            newn.push_back(minuend[i]-c);
            c = 0;
        }
        return newn;
}
```

而在執行減法時做一個定義，minuend 一定會比 subtrahend 大，這樣一來只需要從最小位元開始減，同樣的會有一個 int c 代表向前方索取進位。

而最後的運算子重載，程式碼如下：

```
LongNum operator+(const LongNum & L ){
	if (!(minus ^ L.minus)) {
		LongNum out = LongNum(addition(v,L.v));
		out.minus = minus;
		return out;
	}else{
		LongNum left = *this,right = L;
			left.minus = false;right.minus = false;
		if (left == right){
			return LongNum(0);
		}
		if (left > right && minus){
			// -x + y => -(x-y)
			LongNum out = LongNum(subtraction(v,L.v));
			out.minus = true;
			return out;
		}else if (left > right && L.minus){
			// x + (-y) => (x-y)
			return LongNum(subtraction(v,L.v));
		}else if(left < right && minus){
			// -x + y => (y-x)
			return LongNum(subtraction(L.v,v));
		}else{
			LongNum out = LongNum(subtraction(L.v,v));
			out.minus = true;
			return out;
		}
	}
}
LongNum operator+(long long int n){
	return operator+(LongNum(n));
}
LongNum operator-(const LongNum & L ){
	LongNum right = L;
	right.minus = !right.minus;
	return operator+(right);
}
LongNum operator-(long long int n){
	return operator+(n*-1);
}
```

只要加法做完整，減法部分只需要將減數*-1 當成加數即可。

乘法運算子重載

大數乘法運算子程式：

```
LongNum operator*(const LongNum & L ){
	vector<int> newn(v.size()+L.v.size(),0);
	for(int i = 0 ; i < v.size();i++){
		for(int j = 0 ; j < L.v.size();j++){
			newn[i+j]+=v[i]*L.v[j];
		}
	}
	int c = 0;
	for(int i = 0 ; i < newn.size();i++){
		newn[i]+=c;
		c = newn[i] / 10;
		newn[i] %= 10;
	}
	LongNum out = LongNum(newn);
	out.minus = minus ^ L.minus;
	return out;
}
```

簡單以一個例子，就可以看出大數乘法的簡易算法：

Index / Vector	7	6	5	4	3	2	1	0
v				1	2	3	4	5
L.v						1	2	3
				3	6	9	12	15
			2	4	6	8	10	
		1	2	3	4	5		
newn		1	4	10	16	22	24	18
carried		1	5	1	8	5	5	8

newn 有一個 dp 的演算

$$\forall i, j \in \mathbb{N}, 0 \leq i < M, 0 \leq j < N$$
$$\text{Where } dp[i+j] = dp[i+j] + A[i] \times B[j]$$

其中 M 與 N 分別代表 A 與 B 的 vector 容器大小(size)，且 dp 的初始化直接為 0。

除法與模數運算子重載

至於除法與求其模數則是以長除法來達到求其除法且其餘數。

```
vector<int> addzero(const vector<int> source,int num){
	vector<int> n(num,0);
	for(int i = 0 ; i < source.size();i++){
		n.push_back(source[i]);
	}
	return n;
}
pair<vector<int>,vector<int> > divide(vector<int> dividend,vector<int> divisor){
	pair<vector<int>,vector<int> > result;
	int zeros = (dividend.size() - divisor.size());
	zeros = (zeros > 0 ? zeros : 0);
	vector<int> out(zeros+1,0);
	for(int cnt = 0;zeros >= 0;zeros--,cnt++){
		vector<int> newdivisor = addzero(divisor,zeros);
		while(LongNum(dividend) >= LongNum(newdivisor)){
			LongNum db = LongNum(dividend);
			out[cnt]+= 1;
			dividend = subtraction(dividend,newdivisor);
		}
	}
	reverse(out.begin(),out.end());
	result.first = out;
	result.second = dividend;
	return result;
}
```

在上方的 divide 以 pair 型態輸出，first 位置代表其商，則 second 位置則是餘數，利用補 0 連減的方法求商。

```
operator bool(){
	return !(v.size() == 1 && v[0] == 0);
}
LongNum operator%=(const LongNum & L){
	LongNum o = operator%(L);v = vector<int>(o.v);
	return o;
}
LongNum operator*=(const LongNum & L){
	LongNum o = operator*(L);v = vector<int>(o.v);
	return o;
}
LongNum operator/=(const LongNum & L){
	LongNum o = operator/(L);v = vector<int>(o.v);
	return o;
}
```

operator bool()的案例發生在以轉成布林代數時運作的，也就是 if(LongNum 物件)時會呼叫此 opeator bool()，而我們一般的整數型態的轉乘布林代數只需要確定如果整數不是 0 則輸出 1。

分數運算

需求功能

能達到兩個分數之間的加減乘除，並且數字已 LongNum 為基底，因此需要 LongNum 的加減乘除求餘數(模數)等功能，來達到分數運算。

資料成員

三個資料成員，分別是 LongNum 的 TOP 與 BOT 和一個布林代數 minus，TOP 代表分子(top)，BOT 代表分母(bottom)，而 minus 代表此分數的正負號。

建構子

以下為 Num 分數類別的建構子：

```
Num(){
}
Num(const Num &n){
	TOP = n.TOP;
	BOT = n.BOT;
	reduction();
}
Num(string a,string b="1"){
	this->TOP = a;
	this->BOT = b;
	reduction();
}
Num(LongNum a , LongNum b = 1){
	TOP = LongNum(a);
	BOT = LongNum(b);
	reduction();
};
Num(int a ,int b=1){
	TOP = LongNum(a);
	BOT = LongNum(b);
	reduction();
};
```

Num 類別下的建構子包含

1. Num(const Num&)

2. Num(string,string="1")
3. Num(LongNum,LongNum=1)
4. Num(int,int=1)

這些都淺顯易懂，支援兩輸入，如果只有單一輸入時則分母為 1

下方是以符點數為輸入的一個例子：

```
Num(double num){
	//ex 0.366 -> 355/1000
	if(num < 0){
		this->minus = true;
		num *= -1;
	}
	else this->minus = false;
	ostringstream ss;
	ss << num;
	string s(ss.str());
	long long int l = 0;
	for(int i = 0 ; i < s.length();i++){
		if(s[i] == '.') {
			l = s.length() - i - 1;
			break;
		}
	}
	num *= (long long int)pow(10,l);
	this->TOP = (long long int)num;
	this->BOT = (long long int)pow(10,l);
};
```

在此使用 ostringstream，將 double 型態轉乘 ostringstream，再轉乘 string 型態，一一去找出其分子分母。

最大公因數，最小公倍數

在進行兩分數加減乘除時 u 一定會用到化簡，而此化簡會使用到上下同除最大公因數，而在做加法時也會需要將兩分數的分母做最小公倍數等通分，再做相加，故需要用到 GCD 與 LCM 等功能，而 GCD 與 LCM 算法皆需要用到求餘加減乘除等功能，而我們在 LongNum 時已經定義完，故我們可以直接使用：

```
LongNum GCD(LongNum a,LongNum b){
	a.minus = false;b.minus = false;
	if(b) while((a%=b) && (b%=a));
	return a+b;
}
LongNum LCM(LongNum a,LongNum b){
	a.minus = false;b.minus = false;
	return (a*b)/GCD(a,b);
}
```

在此只需要介紹 GCD 的原理即可，我們需要知道(a%=b)的輸出為 a = a%b 之後的 a，所以假設兩個數字：74 與 12 去做 GCD 則過程為：

a	12	(a%=b)12	(a%=b)0
b	74	(b%=a)2	return a+b

加法減法運算

```
Num operator+(const Num& b){
	LongNum topa = this->TOP;LongNum topb = b.TOP;
	LongNum bota = this->BOT;LongNum botb = b.BOT;
	LongNum _lcm = this->LCM(botb,bota);
	topa *= _lcm/bota;
	topb *= _lcm/botb;
	LongNum _top,_bot;
	_top = topa + topb;
	_bot = _lcm;
	Num out = Num(_top,_bot);
	out.reduction();
	return out;
}
Num operator-(const Num& b){
	LongNum topa = this->TOP;LongNum topb = b.TOP;
	LongNum bota = this->BOT;LongNum botb = b.BOT;
	LongNum _lcm = this->LCM(botb,bota);
	topa *= _lcm/bota;
	topb *= _lcm/botb;
	LongNum _top,_bot;
	_top = topa - topb;
	_bot = _lcm;
	Num out = Num(_top,_bot);
	out.reduction();
	return out;
}
```

加法減法運算較為好講，首先就是最終數字的分母為兩數的最小公倍數(lcm)，故可以看到_bot=_lcm 這行，而分子只需要將各的分子乘上分母與最小公倍數的商(quotient)再將其相加相減其可，而再相加相減部分可以確定此分數分子的正負號，最後如果分子小於 0 則此分數為負數。

乘法除法運算

```
Num operator*(const Num& b){
	LongNum topa = this->TOP;LongNum topb = b.TOP;
	LongNum bota = this->BOT;LongNum botb = b.BOT;
	if(topa == 0 || topb == 0) return Num(0);
	LongNum _top = topa*topb;
	LongNum _bot = bota*botb;
	LongNum _gcd = this->GCD(_top,_bot);
	_gcd.minus = false;
	Num num = Num(_top/_gcd,_bot/_gcd);
	num.reduction();
	return num;
}
Num operator/(const Num& b){
	LongNum topa = this->TOP;LongNum topb = b.TOP;
	LongNum bota = this->BOT;LongNum botb = b.BOT;
	LongNum _top = topa*botb;
	LongNum _bot = bota*topb;
	LongNum _gcd = this->GCD(_top,_bot);
	Num num = Num(_top/_gcd,_bot/_gcd);
	num.reduction();
	return num;
}
```

分數的乘法只需要將分子相乘，分母相乘最後再求其最大公因數消去即可，因為數字系統基於我們所寫的 LongNum，故支援不限長的大數相乘與相除，而能確保_gcd 一定能整除 _top 與 _bot，故在此不需要考慮到小數問題。

取得數值、倒數值、化簡

```
void reduction(){
	if (BOT.minus) TOP.minus = !TOP.minus;
	BOT.minus = false;
	minus = TOP.minus;
	if (TOP == 0) {
		TOP = 0;
		BOT = 1;
	}
	else{
		LongNum _gcd = GCD(this->TOP,this->BOT);
		TOP = TOP/_gcd;
		BOT = BOT/_gcd;
	}
}
Num reciprocal(){
	LongNum t=TOP,b=BOT;
	b.minus = t.minus;
	t.minus = false;
	Num out = Num(b,t);
```

```
        return out;
}
string getValue(){
        if(this->TOP == 0) return "0";
        if(this->BOT == 1) return (this->minus? "-":"")+ itos(this->TOP);
        if(this->BOT == 0) return "ERROR";
        string s("/");
        return (this->minus? "-":"")+ itos(this->TOP) + s + itos(this->BOT);
}
```

reduction 函數在此用於將分母分子化簡，並且以 TOP 代表正負號(雖然 BOT 也可以代表正負號，但是之後再做倒數或其他功能時可能會出問題)，getValue 函數的用意在於，最後輸出的數字，以字串型態輸出，而 reciprocal 則是求此數的倒數，用於之後化簡步驟。

比較運算子重載

```
bool operator> (const Num & b){
        LongNum topa = this->TOP;LongNum topb = b.TOP;
        LongNum bota = this->BOT;LongNum botb = b.BOT;
        LongNum _lcm = this->LCM(botb,bota);
        topa *= _lcm/bota;
        topb *= _lcm/botb;
        return topa > topb;
}
bool operator> (int n){
        return operator>(Num(n));
}
bool operator< (const Num & b){
        LongNum topa = this->TOP;LongNum topb = b.TOP;
        LongNum bota = this->BOT;LongNum botb = b.BOT;
        LongNum _lcm = this->LCM(botb,bota);
        _lcm.minus = false;
        topa *= _lcm/bota;
        topb *= _lcm/botb;
        return topa < topb;
}
bool operator< (int n){
        return operator<(Num(n));
}
bool operator== (const Num & b){
        LongNum topa = this->TOP;LongNum topb = b.TOP;
        LongNum bota = this->BOT;LongNum botb = b.BOT;
        LongNum _lcm = this->LCM(botb,bota);
        _lcm.minus = false;
        topa *= _lcm/bota;
        topb *= _lcm/botb;
        return topa == topb;
}
bool operator== (int n){
        return operator==(Num(n));
}
```

```
bool operator!= (const Num & b){
	return !operator==(b);
}
bool operator!= (int n){
	return !operator==(n);
}
```

分數的比較較為簡單，可以以加法的方式將分母變成一模一樣，最後在比較分子即可。

額外功能

```
bool isZero(){
	return TOP == 0;
}
bool isOne(){
	return TOP == BOT;
}
```

在此額外寫一個 isZero 與 isOne 在其物件使用時可以避免使用 ==0 功能來判斷(避免問題出在 int 轉 Num)。

矩陣容器

需求功能

在寫 Matrix.h 之前，我們需要先知道矩陣可以看作一個 2 維陣列，同樣的也可以看做他由一個 col vector 與對應數量的 row vector 來組成，故在此我們會定義一個 rowVector 作為基底。

rowVector 類別

rowVector 代表矩陣裡面的每一列，而之後再 Matrix 類別裡面可以以一個 vector<rowVector>來記錄每一行的列元素，而 rowVector 所能達到的功能不僅僅只是查詢與設定值，還要能支持列運算(row operation)，列運算有三種：

1. 交換任兩列
2. 一列乘上一個非 0 的數字
3. 將一列乘上一個非 0 的數字加到另一列

資料成員

在 rowVector 裡面有一個資料成員 vector<Num>用來儲存每個元素。

建構子

```
rowVector(int s = 0){
	v = vector<Num>(s,0);
}
rowVector(vector<Num> vec){
	v = vector<Num>(vec);
}
```

在此 rowVector 只用到兩個建構子，一個是帶有預設值的定長度，另一個則是給定一 rowVector 做複製輸入。

給值取值

相對的，在建構子只有單單設定一個初始化元素陣列，所以我們還需要再多兩個函數，有點類似特地開一個出入口，改值與取值都通過此通道。

```
void setValue(int p,Num val){
	v[p] = val;
}
Num getValue(int p){
	return v[p];
}
```

列基本運算

上述有提到，rowVector 需要能使用列運算等功能，故在此定義三個函數，來達到列運算：

```
void swap(rowVector & nv){
	rowVector temp = nv;
	nv = *this;
	*this = temp;
}
void multiply(Num value){
	for(int i = 0 ; i < v.size();i++) v[i] = v[i] * value;
}
void addto(rowVector & nv,Num value){
	for(int i = 0 ; i < v.size();i++){
		nv.v[i] = nv.v[i] + (v[i] * value);
	}
}
```

Matrix 類別

Matrix 類別與 rowVector 類關係以 has-a 組合，因為我們可以視為有一個 vector 來儲存每一列(rowVector)，因為是組合關係，故列的基本運算同樣要再定義一次。

資料成員

來到了 Matrix，也就是 2 維儲存空間，故會有兩個變數 R 與 C 分別記錄 row 與 column，另外還會有一個 colVector 來記錄每列的 rowVector，最後則是一個列舉 type 用來儲存目前的矩陣型態，其值有 MATRIX、AUGMENT、INVERSE。

建構子

```
Matrix(int r,int c){
	type = MATRIX;
	R=r;C=c;
	for(int i = 0 ; i < r;i++) colVector.push_back(rowVector(c));
}
Matrix(const Matrix&m){
	type = m.type;
	R=m.R;C=m.C;
	colVector = m.colVector;
}
```

我們會先給此矩陣一個全為 0 的空矩陣，之後再給予設定值。而因為 rowVector 的每一項元素都是 Num 型態，故給定的值可以是超長整數、分數等形式。

列基本運算

```
void swap(int i,int j){
	colVector[i].swap(colVector[j]);
}
void multiply(int i,Num value){
	colVector[i].multiply(value);
}
void addto(int i,int j,Num value=1){
	colVector[i].addto(colVector[j],value);
}
```

同樣有點類似開個出口，讓外界以這個函數進行列運算。

列基本運算

```
void setValue(int i,int j,Num value){
	colVector[i].setValue(j,value);
}
```

同樣也是開個通道給予設定值。

矩陣格式化輸出

為了讓此矩陣能以更好看的形式表現出來，所以利用到 setw 等功能，來隔開數字。

```
void formatted(){
	if (R == 1){
		printf("〔");
		for(int i = 0 ;i < C;i++){
			if(i) cout << " ";
			cout << colVector[0].getValue(0).getValue();
		}
		printf("〕");
	}else{
		for(int i = 0 ; i < R;i++){
			if(!i) cout << " ┌";
			else if(i == R-1)cout << " └";
			else cout << " │ ";
			for(int j = 0 ; j < C;j++){
				cout << setw(17) << colVector[i].getValue(j).getValue();
			}
			if(!i) cout << "┐ ";
			else if(i == R -1) cout << "┘ ";
			else cout << " │ ";
			cout << endl;
		}
	}
}
```

AugmentMatrix 類別

AugmentMatrix 擴增矩陣，很顧名思義就是擴增，且保有 Matrix 的所有功能，只是在每一列多了一個元素(常數矩陣)。

```
class AugmentMatrix:public Matrix{
	public:
		AugmentMatrix(Matrix x,vector<Num> constMatrix):Matrix(x){
			if(type != MATRIX) return;
			type=AUGMENT;
			for(int i = 0;i<R;i++)colVector[i].v.push_back(constMatrix[i]);
			C++; }
};
```

高斯(喬登)消去

需求功能

能支援不管是 Matrix 或 AugmentMatrix 的物件去做到消去法，並且將每個例子都顯示出來。
在此我們會需要多一個類別 Solution 用來記錄從上一個步驟到此步驟所進行的動作，最後以 vector<Solution>來記錄全部的消去步驟，Solution 獨立出來的好處是，之後在寫 LaTeX 時可以透過修改 Solution 裡面直接達到轉成數學代碼的效果。

Solution 類別

Solution 用於紀錄每個消去的步驟，也用於之後轉 LaTeX 出來，故每個 Solution 都會記錄一個 Matrix 與其來源的方法：

```
enum Stype{START,SWAP,MULTI,ADDTO};
class Solution{
	public:
		enum Stype prevType;
		Matrix matrix;
		Num t1=0,t2=0,t3=0;
		Solution(Matrix m,Stype type=START,Num t1 = 0,Num t2 = 0,Num t3 =
0):matrix(m),
		prevType(type),t1(t1),t2(t2),t3(t3){
		}
};
```

Stype 在此以列舉方式儲存，代表列基本運算的三種(swap,multi,addto)，而 start 代表此矩陣消去的開始(也就是頭)的部分。
因為是由 Matrix 紀錄，故也可以支援 AugmentMatrix 來做紀錄。

Elimination 類別

在 Elimination 裡面，會有一個 vector<Solution>來記錄所有消去過程。並且分成 GaussElimination 與 GaussJordanElimination 等功能。

GaussElimination 函數

```
GaussElimination(){
	int j = 0;
	for(int i = 0 ; i < m.R;i++){
		while(j < m.C){
			if(m.getValue(i,j).isZero()) {
				for(int k = i+1 ; k < m.R;k++){
					if(!m.getValue(k,j).isZero()) {
						m.swap(j,k);
						sols.push_back(Solution(m,SWAP,j,k));
						break;
					}
				}
			}
			if(m.getValue(i,j).isZero()){
				j++;
				continue;
			}else if(!m.getValue(i,j).isOne()){
				Num dis = m.getValue(i,j);
				dis = dis.reciprocal();
				m.multiply(i,dis);
				sols.push_back(Solution(m,MULTI,i,dis));
			}
			for(int k = i+1; k < m.R;k++){
				if(m.getValue(k,j).isZero()) continue;
				Num dis = m.getValue(k,j)*Num(-1);
				m.addto(i,k,dis);
				sols.push_back(Solution(m,ADDTO,i,k,dis));
			}
			break;
		}
	}
}
```

高斯消去的作法可以很明顯地看到，最外層有一個變動量 j，在此先假設他是紀錄目前所指引的行數，我們舉個例子：

今天有一矩陣長這樣

4	7	8	6
1	2	4	7
6	5	4	1

而左至右的索引值分別是 0~3，然後目前的 j 是 1 則指向的是數字 7、2、5 這行，也就是說如果再給予正確的指定列數(0~3)可以抓取指定數值。

為甚麼要特別定行而不是列呢？原因是列運算，讓行變成指定索引我們可以判斷第幾列的第幾行我們想要把她消成矩陣裡面的第一項非 0 元素皆為 1，我們稱這數為領導數 1(leading 1)，並且每一行最多就只有一個領導數 1。

GaussJordanElimination 函數

GaussJordanElimination 與 GaussElimantion 的差異只是將 GaussElimination 的結果再去做消去，GaussJordanElimination 使用到代入法(row echelon)，消去到最簡。

```
GaussJordanElimination(){
	GaussElimination();
	for(int i = m.R -1 ; i >= 0 ; i--){
		for(int j = 0 ; j < m.C;j++){
			if(m.getValue(i,j).isZero()) continue;
			else if(m.getValue(i,j).isOne()){
				//do row echelon
				for(int k = i-1;k >= 0 ; k--){
					Num dis = m.getValue(k,j)*Num(-1);
					m.addto(i,k,dis);
					sols.push_back(Solution(m,ADDTO,i,k,dis));
				}
			}else break;
		}
	}
}
```

此代入法使用兩個 for，最外層的 for 迴圈以 i 為變動量，代表目前執行的第 i 列，i 從最後一列到 0。

第二個 for 則是以 j 為變動量，用來尋找此列第一個非 0 且為 1 的數字，如果找到則執行消去，對上方的每一列做消去，將第 j 行的數字都消減只剩下一個 1 其餘都是 0。

線性方程式

需求功能

在此類別裡可以支援輸入多個以不同且的英文字元組成的一線性方程式。
而將字串轉成一個方程式，另外還要有一個解線性方程組的一個功能，支援多個線性方程式輸入，最後求其解。
而在此將這兩個功能劃分為兩個類別 Equation 與 LinearEquation 類別。

Equation 類別

Equation 類別主要功能是將字串轉成一個線性方程組，並且可以分配每個變數的係數。

```
class Equation{
	public:
		vector<Num> mem = vector<Num>(26,Num(0,1));
		vector<int> variable;
		Num constNum = Num(0,1);
		Equation(string s){
			int index = 0;
			string temp;
			for(index = 0; s[index] != '=' && index < s.length();index++){
				if(s[index] >= 'a' && s[index] <= 'z'){
					variable.push_back(s[index]-'a');
					mem[s[index]-'a'] = mem[s[index]-'a'] + Num(temp.empty() ?
"1":(temp == "-" ? "-1": temp)  );
					temp.clear();
				}else if(s[index] == '+') continue;
				else{
					temp = temp + s[index];
				}
			}
			temp.clear();
			for(index++;index < s.length();index++) temp = temp + s[index];
	constNum = Num(temp);
	}
};
```

在 Equation 類別裡面，用一個 26 長度的陣列 mem 來記住分別以 a~z 為變數的領導係數，而 variable 則是記錄目前有幾個已

出現的變數(0~25 表示 a~z)，而 constNum 則是這線性方程式的常數項。

LinearEquation 類別

在 LinearEquation 裡，必須記錄目前有幾個 Equation 並且提供一個功能能解出一線性方程式的解種類。

新增線性方程式

設定一個函數，可以新增新的線性方程式到此線性方程組裡，故需要紀錄目前有哪些變數，和其每個線性方程式的位置
在此使用 eq 代表一個不限長的 vector 陣列來儲存目前有的 Equation 物件，而 variable 則是一個 set 用來儲存已經出現的變數，因為變數可能會重複，故使用集合來減少之後的消去。而最後在利用一個 vector 陣列 constNum 來記錄所有線性方程式的常數。

```
addEquation(string s){
	Equation neq = Equation(s);
	for(int i = 0 ; i < neq.variable.size();i++){
		variable.insert(neq.variable[i]);
	}
	eq.push_back(neq);
	constNum.push_back(neq.constNum);
}
```

解線性方程組

以 eq、variable 和 constNum 變數來儲存一個線性方程組後接著就是解其線性方程組的解，在此我們使用擴增矩陣。

```
Matrix m = Matrix(eq.size(),variable.size());
for(int i = 0 ; i < eq.size();i++){
	int j = 0;
	for(set<int>::iterator it = variable.begin();it != variable.end();it++,j++){
		m.setValue(i,j,eq[i].mem[*it]);
	}
}
AugmentMatrix am = AugmentMatrix(m,constNum);
Elimination em = Elimination(am);
em.GaussJordanElimination();
```

同樣的先建立一個矩陣物件，其列數(r)為方程式數量、行數(c)

為變數總數，接著是設定矩陣裡的每一個值。
由上面的步驟可以不難看出 m 就是我們的係數矩陣，接著將此矩陣做擴增，也就是將每個線性方程式的常數(constNum)合併到此係數矩陣，再去做消去法。
而因為 Elimination 的輸入為 Matrix，且 AugmentMatrix 是繼承 Matrix 故可以直接將 AugmentMatrix 帶入到 Elimination 去做消去。

無解情況

```
bool noSolution(Matrix m){
	bool nosol = false;
	for(int i = 0 ; i < m.R && !nosol;i++){
		bool fullzero = true;
		int j = 0 ;
		for(; j < m.C-1 && !nosol;j++){
			fullzero = m.getValue(i,j).isZero() && fullzero;
		}
		nosol = fullzero && !m.getValue(i,j).isZero(); // fullzero and last not zero
	}
	return nosol;
}
```

在利用矩陣求線性方程組解時，高斯喬登消去完後，只要有一行的係數項全為 0 且常數不為 0，則代表此線性方程組無解(0x+0y=1)。

洽一組解情況

```
bool oneSolution(Matrix m){
	bool onesol = (m.R >= variable.size() );
	for(int i = 0 ; i < min(m.R,(int)variable.size());i++) onesol = onesol &&
m.getValue(i,i).isOne();
	return onesol;
}
```

在利用矩陣求線性方程組解時，高斯喬登消去完後，每一個變數皆對應到一個常數值，也就是說假設今天有 4 個變數，那若要此方程組為一組解則擴增矩陣高斯喬登消去完後的係數矩陣應為 4 列(也就是一個單位矩陣)。

無限多組解情況

如果不符合無解、洽一組解的情況，那只剩下無限多組解，則在給出無限多組解的過程中需要注意的是，在此要給出解集合，而不是直接給予無解這個答案。

```
int temp = 1;
map<int,int> var;
for(int i = em.m.R-1; i >= 0 ; i--){
	int leadingone = -1;
	for(int j = 0 ; j < em.m.C-1;j++) {
		if(!em.m.getValue(i,j).isZero()) {
			leadingone = j;
			break;
		}
	}
	if(leadingone == -1) continue; //full zero row
	for(int j = leadingone+1;j < em.m.C-1;j++){
		if(!em.m.getValue(i,j).isZero() && !var[j]){
			cout << "let " << char((*variable.begin()+j)+'a') << "= (temp" << temp <<
") in RealNumber." << endl;
			var[j] = temp++;
		}
	}
	cout << char((*variable.begin()+leadingone)+'a') << " = " <<
em.m.getValue(i,em.m.C-1).getValue();
	for(int k = leadingone+1 ; k < em.m.C-1;k++){
		if(var[k]){
			Num n = em.m.getValue(i,k) * Num(-1);
			if (!n.minus) cout << "+" << n.getValue() << "(temp" << var[k] << ")";
			else cout << n.getValue() << "(temp" << var[k] << ")";
		}
	}
	cout << endl;
}
```

而通常我們在給予一解集合時，會有一個動作「另 x = t, t∈R」在程式裡面，使用(temp1,temp2)來取代 t,s 這種可能會重複的變數，所以會看到一行「let x = (temp1) in RealNumber」其中 x 代表目前指定的變數名稱。

判斷的過程我們以下方擴增矩陣為例(已經高斯消去消到最簡)

a	b	c	d	
1	1	0	2	-6
0	0	1	-1	6

白底為其係數矩陣的解，從此擴增矩陣可以看出一定有無限多組解，而這方程式的解集合可以寫成：

令 d=t1，t1 屬於任意實數，c=6+t1

令 b=t2，t2 屬於任意實數，a=-6+2t1-t2

上方程式的寫法是這樣的，設定一個索引值 i 指到最後一列，美次執行第 i 列時先找此列的領導 1(leading 1)，也就是由左至右，因為是高斯喬登消去後的矩陣，故可以很放心的直接用這種方法。找到領導 1 之後到最後一個係數項，如果有尚未被 var 記錄到的值則重新定義這個變數(令 x=t，t 屬於 R)，而 var 這個 map 對應的 key 與 value 分別代表，key 代表目前的第幾個變數，value 此變數為 temp 幾(也就是重新定義的第幾個變數)。而最下方的 for 迴圈則是輸出目前這列領導係數的值，因為目前重新定義的變數都在左側故要移位(乘上-1)。上方的程式執行結果為：

```
let d= (temp1) in RealNumber.
c = 6+1(temp1)
let b= (temp2) in RealNumber.
a = -6-1(temp2)-2(temp1)
```

判斷最後的解

在進行高斯喬登消去之後，先確認是否無解，再來是恰一組解，最後才是無限多組解。

```
if (noSolution(em.m)){
	cout << "these linear equations has no solution" << endl;
}else if (oneSolution(em.m)){
	cout << "these linear equations has only 1 soluiton:" << endl;
	int j = 0;
	for(set<int>::iterator it = variable.begin();it != variable.end();it++,j++){
		cout << char(*it+'a') << "=" << em.m.getValue(j,am.C-1).getValue() << endl;
	}
}else{
	cout << "inf solution" << endl;
}
```

逆矩陣

需求功能

此 InverseMatrix 同樣是繼承 Matrix 類別，作法如同擴增矩陣只是合併了一個跟列數量依樣大小的單位矩陣。
而在 InverseMatrix 中還要能提供一個功能，求其逆矩陣的解。

程式解析

```
typedef struct Result{
	Matrix m;
	bool isInverse;
};
class InverseMatrix:public Matrix{
	public:
		InverseMatrix(Matrix x):Matrix(x){
			if(type != MATRIX) return;
			type = INVERSE;
			// to know if want to find a inverse matrix A^-1 , A must be square matrix
			//
			//
			for(int i = 0;i<R;i++){
				for(int j = 0 ; j <C;j++){
					if(i==j) colVector[i].v.push_back(1);
					else colVector[i].v.push_back(0);
				}
			}
			C*=2;
		}

		Result getInverse(){
			bool invm = (C == R*2);
			for(int i = 0 ; i < R ; i++){
				invm = colVector[i].v[i].isOne() && invm;
			}
			Matrix m = Matrix(R,R);
			for(int i = 0 ; i < R;i++){
				for(int j = 0 ; j < R;j++){
					m.setValue(i,j,colVector[i].v[j+R]);
				}
			}
			return Result{m,invm};
		}
};
```

因為大部分內容都與擴增矩陣差不多，故在此直接以程式來做

講解，在建構子裡面先判斷目前 Matrix 進入的型態如果是 Matrix 則執行合併單位矩陣的功能。
那位什麼要寫呢?因為在之後可以直接以結果(InverseMatrix)m 轉成 InverseMatrix 型態而不會出現問題，因為 m 原本就是 InverseMatrix 所以不會再被執行合併另一個單位矩陣。

逆矩陣解

在上方的 Result 可以看到，在 Result 結構裡有兩個變數，分別代表逆矩陣結果的 Matrix 與是否可逆的 isInverse。
判斷可逆的結果只需要看擴增後的高斯喬登消去結果是否左側為單位矩陣，如果不是則此矩陣不可逆。

程式例子

這邊給出一個例子：

```
InverseMatrix AI = InverseMatrix(m);
Elimination state = Elimination(AI);
state.GaussJordanElimination();
Result r = ((InverseMatrix)state.m).getInverse();
```

在 InverseMatrix 建構子裡面設定一個如果此型態來源是原本矩陣(非逆矩陣)的話則轉成逆矩陣，不是則不轉。因此在下方的

```
Result r = ((InverseMatrix)state.m).getInverse();
```

可以很人性化的江 state.m 也就是做完高斯喬登消去後的矩陣轉成 InverseMatrix，再求其解 getInverse()。

LaTeX 數學表示

需求功能

能將解此線性方程組、逆矩陣、高斯(喬登)消去的每一個步驟明確並以 LaTeX 方式表示出來。
從上方的說明可以很明顯的看到，這個 LaTeX 類別基本上是貫穿了整個系統，所以而在此以「傳送位址」等方式來確保這個 LaTeX 物件在每個系統上是統一的，由統一的 LaTeX 控制。
因此需要明確定義這個 LaTeX 類別的每項功能。

建構子

在建構子裡，我們要先重製輸出的檔案：

```
LaTeX(string filename):filename(filename){
	ofstream reset(filename);
	reset << "";
	reset.close();
}
```

如果在此無法使用這個 ofstream，請確保 compiler 是 c++11 以上的版本。

新增文字

```
void addText(string s){
	ofstream add(filename,std::ios::app);
	add << s;
	add.close();
}
```

接著我們要來新增文字，在 ofstream 後面加上 std::ios::app 表示為新增(append)而不是複寫(overwrite)。

統整系統

讓 Matrix.h、Elimination.h、LinearEquation.h、InverseMatrix.h 等標頭黨都能使用 LaTeX 且都以指標存取，更改的地方有：

```
//Matrix.h
LaTeX *LTX;

Matrix(int r,int c,LaTeX *LTX=NULL):LTX(LTX){
	type = MATRIX;
	R=r;C=c;
	for(int i = 0 ; i < r;i++) colVector.push_back(rowVector(c));
}
Matrix(const Matrix&m){
	LTX = (m.LTX);
	type = m.type;
	R=m.R;C=m.C;
	colVector = m.colVector;
}
```

```
//Elimination.h  |  Solution class
LaTeX *LTX;
Solution(Matrix m,LaTeX *LTX=NULL,Stype type=START,Num t1 = 0,Num t2 = 0,Num t3
= 0):matrix(m),
prevType(type),t1(t1),t2(t2),t3(t3){}
```

```
//Elimination.h |Elimination class
LaTeX *LTX;

Elimination(Matrix m,LaTeX *LTX=NULL):m(m),LTX(LTX){
	sols.push_back(Solution(m));
}
```

```
//LinearEquation.h
LaTeX *LTX;

LinearEquation(LaTeX *LTX = NULL):LTX(LTX){}
```

在每個建構子的部分可以看到，預設值都是 NULL，故可以不用指定要 LaTeX 輸出，因此在之後轉換過程都需要偵測是否 LTX 存在。

```
string getLaTeX(){
	if(this->TOP == 0) return "0";
	if(this->BOT == 1) return  (this->minus? "-":"")+ itos(this->TOP);
	return (this->minus? "-":"") + string("{ ")+itos(this->TOP) + string("\\over ")+
itos(this->BOT)+ string("}");
}
```

在 Num.h 裡面則是多一個 分數形式的 LaTeX。

列印矩陣

在 LaTeX 的矩陣表示法如下：

\begin{bmatrix}
1 & 2 & 3\\
4 & 5 & 6\\
\end{bmatrix}

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

故我們要列印一個矩陣出來時則將矩陣型態轉換成上方 LaTeX 形式的輸出。

```
void write(){
	if(LTX == NULL) return;
	LTX->addText("\\begin{bmatrix} ");
	for(int i = 0 ; i < R;i++){
		for(int j = 0 ; j < C;j++){
			if(j) LTX->addText(" & ");
			LTX->addText(colVector[i].getValue(j).getLaTeX());
		}
		if(i != R-1) LTX->addText("\\\\ \n");
	}
	LTX->addText("\\end{bmatrix} \\\\");
}
```

轉換過程(列運算)

接著是寫輸出列運算的部分，而在 LaTeX 裡文字已\text{}來表示而箭頭部分則是使用\xrightarrow{}括號中圍箭頭上的內容：

\xrightarrow{\text{add row 0 to row 2 by value -3}}

$$\xrightarrow{\text{add row 0 to row 2 by value -3}}$$

其修改地方為 Elimination.h

```
Solution(Matrix m,LaTeX *LTX=NULL,Stype type=START,Num t1 = 0,Num t2 = 0,Num t3
= 0):matrix(m),prevType(type),t1(t1),t2(t2),t3(t3){
	if(LTX != NULL){
		if(type == SWAP) LTX->addText("\\xrightarrow{\\text{swap " + t1.getValue() +
" <-> " + t2.getValue() + "}} ");
		if(type == MULTI) LTX->addText("\\xrightarrow{\\text{multi " +t1.getValue()+"
value: " + t2.getValue() + "}} ");
		if(type == ADDTO) LTX->addText("\\xrightarrow{\\text{add " + t1.getValue() + "
to " + t2.getValue() + " by value " +t3.getValue() + "}} ");
	}
	m.formatted();
}
```

列印出其解

而這部分需要講解的地方並不多，只是單純的列印結果出來。

各類別的成員參考表

LongNum

Member Types

member type	definition
minus	bool
v	vector<int>

Member Functions

(constructor)	建構 LongNum 類別
addzero	除法部分補 0 使用
divide	除法運算
addition	加法運算
subtraction	減法運算

Operator Overloads

operator+	加法運算子
operator-	減法運算子
operator*	乘法運算子
operator/	除法運算子
operator (compare)	比較運算子
bool	轉乘布林代數型態
operator%=	模後改變自身值
operator*=	乘後改變自身值
operator/=	除後改變自身值

Num

Member Types

member type	definition
TOP	LongNum
BOT	LongNum
minus	bool

Member Functions

(constructor)	建構 Num 類別
reduction	將此分數約分到最簡
reciprocal	回傳此分數的倒數
GCD	求兩 LongNum 的最大公因數
LCM	求兩 LongNum 的最小公倍數
getLaTeX	取得 LaTeX 形式字串輸出
getValue	取得一般形式字串輸出
isZero	是否為 0
isOne	是否為 1

Operator Overloads

operator+	加法運算子
operator-	減法運算子
operator*	乘法運算子
operator/	除法運算子
operator (compare)	比較運算子
bool	轉乘布林代數型態

rowVector

Member Types

member type	definition
v	vector<Num>

Member Functions

(constructor)	建構 rowVector 類別
setValue	設定元素
getValue	取出元素
swap	列互換
multiply	所有元素乘上一倍數
addto	乘上倍數加至另一 rowVector

Matrix

Member Types

member type	definition
type	MatrixType
colVector	vector<rowVector>
R,C	int
*LTX	LaTeX

Member Functions

(constructor)	建構 rowVector 類別
swap	互換矩陣內兩列
multiply	指定矩陣內的一列乘上倍數
addto	將矩陣內任一行乘上一倍數加至另一行
setValue	設定元素
getValue	取得元素
write	LaTeX 輸出(formatted 內建)
formatted	輸出矩陣文字

Solution

Member Types

member type	definition
prevType	Stype{START,SWAP,MULTI,ADDTO}
matrix	Matrix
t1,t2,t3	Num
*LTX	LaTeX

Member Functions

(constructor)	建構 Solution 類別

Elimination

Member Types

member type	definition
sols	vector<Solution>
m	Matrix
*LTX	LaTeX

Member Functions

(constructor)	建構 Elimination 類別
GaussElimination	高斯消去法
GaussJordanElimantion	高斯喬登消去法

Equation

Member Types

member type	definition
mem	vector<Num>
variable	vector<int>
constNum	Num

Member Functions

(constructor)	建構 Equation 類別
showDebug()	顯示線性方程式

LinearEquation

Member Types

member type	definition
eq	vector<Equation>
variable	set<int>
constNum	vector<Num>
*LTX	LaTeX

Member Functions

(constructor)	建構 LinearEquation 類別
addEquation	新增線性方程式
Solve	求線性方程組解
noSolution	無解判斷
oneSolution	洽有一組解

Matrix::InverseMatrix

Member Functions

(constructor)	建構 InverseMatrix 類別
getInverse	逆矩陣結果

LaTeX

Member Types

member type	definition
filename	string

Member Functions

(constructor)	建構 LaTeX 類別
addText	新增文字

結論

對於這次有系統性的規劃此系統，從無到有，歷經多次挫敗，也耗時將近 40 小時，雖然說這不是我第一次寫需要完整性規劃的系統。我個人認為要寫一個完整性非常高的系統需要注意很多，不僅是當下開發時所想的，也要考慮到之後可能會用到的功能，也就是說相容性要高，而這種規模不小的專題，每個類別每個功能的解釋都要清楚，會有助於之後來修正錯誤，假設今天找到一個錯誤也比較好回推源頭。

我個人認為在開發一個完整性高的系統時，為了不做到超級封閉，也就是有一定的相容性，(一)是要將功能做得非常廣，換句話說就是假設今天有一個判斷是否為 0 的值，雖然我們可以直接複寫運算子(==)但是我在此卻又多一個 isZero 的函數，不僅可以大大避免之後程式出錯，還可以增加程式的可讀性。(二)是要馬就死記所有類別下的資料成員功能，不然就是將類別裡的資料成員功能寫得非常清楚，同樣的是增加程式的可讀性。(三)

則是開發過程可以很慢沒關係，主要是自己知道目前在做甚麼，要非常的清楚，一但有一個地方錯之後可能都會錯，我是覺得再多加一個類別不難，難的是之後出現 Bug 要花更長時間去修復。(最後)則是在規劃一個完整性的系統時，要先列出需要的功能，在此空能去想那些子功能，藉由慢慢完成這些子功能來完成整個系統。

程式實例

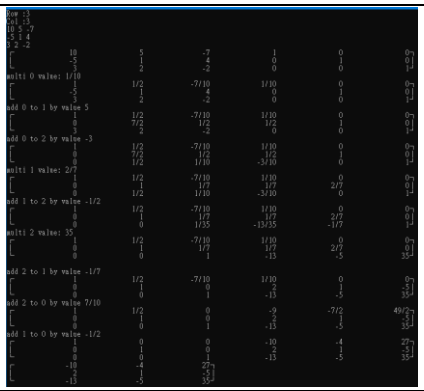
```
#include "LinearEquation.h"
#include "Elimination.h"
#include "Matrix.h"
#include "InverseMatrix.h"
#include "LaTeX.h"
#include <iostream>
using namespace std;
int main(){
	LaTeX out = LaTeX("testfile.txt");
	LinearEquation system = LinearEquation(&out);
	system.addEquation("a+b+c+d+e+f+g+h+4i=0");
	system.addEquation("4a+9i=1");
	system.Solve();
}
```

```
a+b+c+d+e+f+g+h+4i=0
4a+9i=1
┌   1    1    1    1    1    1    1    1    4     0┐
└   4    0    0    0    0    0    0    0    9     1┘
add 0 to 1 by value -4
┌   1    1    1    1    1    1    1    1    4     0┐
└   0   -4   -4   -4   -4   -4   -4   -4   -7     1┘
multi 1 value: -1/4
┌   1    1    1    1    1    1    1    1    4     0┐
└   0    1    1    1    1    1    1    1  7/4  -1/4┘

add 1 to 0 by value -1
┌   1    0    0    0    0    0    0    0  9/4   1/4┐
└   0    1    1    1    1    1    1    1  7/4  -1/4┘
inf solution
let c= (temp1) in RealNumber.
let d= (temp2) in RealNumber.
let e= (temp3) in RealNumber.
let f= (temp4) in RealNumber.
let g= (temp5) in RealNumber.
let h= (temp6) in RealNumber.
let i= (temp7) in RealNumber.
b = -1/4-1(temp1)-1(temp2)-1(temp3)-1(temp4)-1(temp5)-1(temp6)-7/4(temp7)
a = 1/4-9/4(temp7)
```

```
\text{a+b+c+d+e+f+g+h+4i=0}\\\text{4a+9i=1}\\\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 0\\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 1\end{bmatrix} \\\xrightarrow{\text{add 0 to 1 by value -4}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 0\\
0 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -7 & 1\end{bmatrix} \\\xrightarrow{\text{multi 1 value: -1/4}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 0\\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & { 7\over 4} & -{ 1\over 4}\end{bmatrix} \\\xrightarrow{\text{add 1 to 0 by value -1}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & { 9\over 4} & { 1\over 4}\\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & { 7\over 4} & -{ 1\over 4}\end{bmatrix} \\\text{systems of linear equations with inf solutions}\\let \space c= (temp1) \in R \\let \space d= (temp2) \in R \\let \space e= (temp3) \in R \\let \space f= (temp4) \in R \\let \space g= (temp5) \in R \\let \space h= (temp6) \in R \\let \space i= (temp7) \in R \\b=-{ 1\over 4}-1(temp1)-1(temp2)-1(temp3)-1(temp4)-1(temp5)-1(temp6)-{ 7\over 4}(temp7)\\a={ 1\over 4}-{ 9\over 4}(temp7)\\
```

```
#include "LinearEquation.h"
#include "Elimination.h"
#include "Matrix.h"
#include "InverseMatrix.h"
#include "LaTeX.h"
#include <iostream>
using namespace std;
int main(){
	int col,row;
	LaTeX out = LaTeX("testfile.txt");
	cout << "Row :"; cin >> row;
	cout << "Col :"; cin >> col;
	Matrix m = Matrix(row,col,&out);
	for(int i = 0 ; i < row;i++){
		for(int j = 0; j < col;j++){
			string s;
			cin >> s;
			m.setValue(i,j,Num(s));
		}
	}
	InverseMatrix AI = InverseMatrix(m);
	Elimination state = Elimination(AI,&out);
	state.GaussJordanElimination();
	Result r = ((InverseMatrix)state.m).getInverse();
	if(r.isInverse) {
		r.m.formatted();
	}else{
		cout << "this matrix dont have inverse matrix" << endl;
	}
}
```



```
\begin{bmatrix} 10 & 5 & -7 & 1 & 0 & 0\\
-5 & 1 & 4 & 0 & 1 & 0\\
3 & 2 & -2 & 0 & 0 & 1\end{bmatrix} \\\xrightarrow{\text{multi 0 value: 1/10}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
-5 & 1 & 4 & 0 & 1 & 0\\
3 & 2 & -2 & 0 & 0 & 1\end{bmatrix} \\\xrightarrow{\text{add 0 to 1 by value 5}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & { 7\over 2} & { 1\over 2} & { 1\over 2} & 1 & 0\\
3 & 2 & -2 & 0 & 0 & 1\end{bmatrix} \\\xrightarrow{\text{add 0 to 2 by value -3}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & { 7\over 2} & { 1\over 2} & { 1\over 2} & 1 & 0\\
0 & { 1\over 2} & { 1\over 10} & -{ 3\over 10} & 0 & 1\end{bmatrix} \\\xrightarrow{\text{multi 1 value: 2/7}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & 1 & { 1\over 7} & { 1\over 7} & { 2\over 7} & 0\\
0 & { 1\over 2} & { 1\over 10} & -{ 3\over 10} & 0 & 1\end{bmatrix} \\\xrightarrow{\text{add 1 to 2 by value -1/2}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & 1 & { 1\over 7} & { 1\over 7} & { 2\over 7} & 0\\
0 & 0 & { 1\over 35} & -{ 13\over 35} & -{ 1\over 7} & 1\end{bmatrix} \\\xrightarrow{\text{multi 2 value: 35}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & 1 & { 1\over 7} & { 1\over 7} & { 2\over 7} & 0\\
0 & 0 & 1 & -13 & -5 & 35\end{bmatrix} \\\xrightarrow{\text{add 2 to 1 by value -1/7}} \begin{bmatrix} 1 & { 1\over 2} & -{ 7\over 10} & { 1\over 10} & 0 & 0\\
0 & 1 & 0 & 2 & 1 & -5\\
0 & 0 & 1 & -13 & -5 & 35\end{bmatrix} \\\xrightarrow{\text{add 2 to 0 by value 7/10}} \begin{bmatrix} 1 & { 1\over 2} & 0 & -9 & -{ 7\over 2} & { 49\over 2}\\
0 & 1 & 0 & 2 & 1 & -5\\
0 & 0 & 1 & -13 & -5 & 35\end{bmatrix} \\\xrightarrow{\text{add 1 to 0 by value -1/2}} \begin{bmatrix} 1 & 0 & 0 & -10 & -4 & 27\\
0 & 1 & 0 & 2 & 1 & -5\\
0 & 0 & 1 & -13 & -5 & 35\end{bmatrix} \\
```