程设实验 Project04

Linux-Polynomial calculator

191220090 沈天杰

目录

CONTENTS

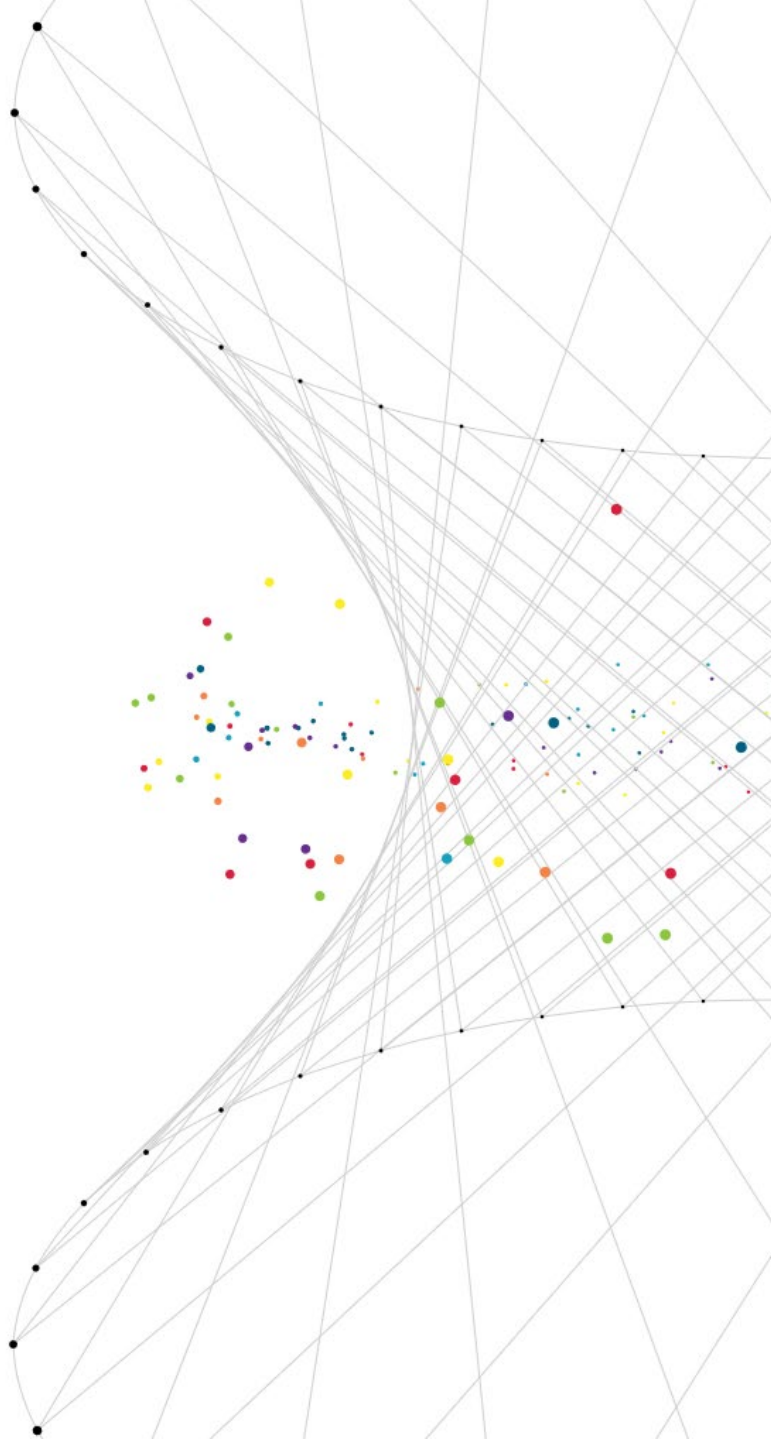
需求分析

PART ONE

PART ONE 需求分析

多项式计算器具有以下功能:

1. 多项式输入输出
2. 多项式混合计算(求导/定积分、乘法、加法)
3. 多项式求逆
4. 多项式除法/取模
5. 多项式求根
6. 继续进行下一轮计算
7. …

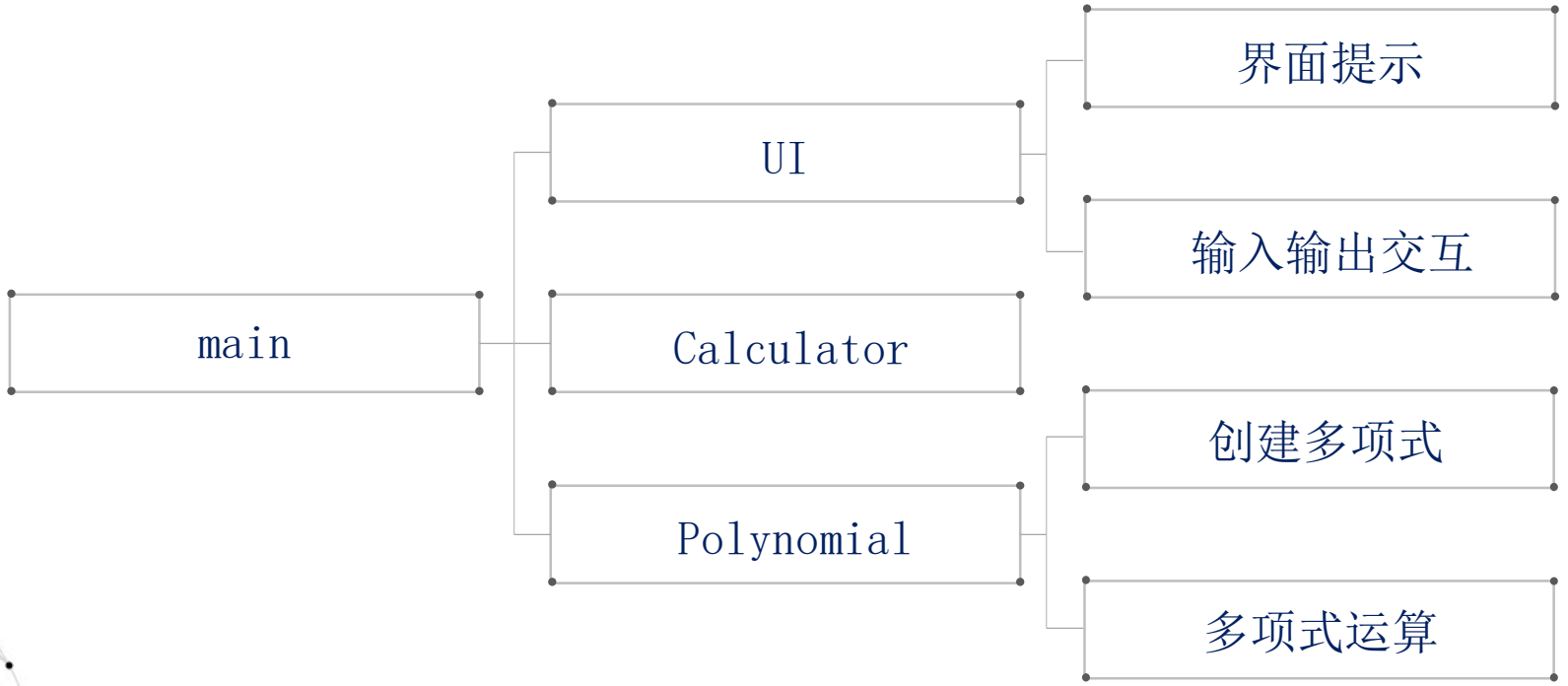


模块设计

PART TWO

模块划分

用户界面　　输入输出　　多项式运算



PART TWO 模块设计

makefile

```
main:Polynomial.o Calculator.o UI.o main.o
	g++ Polynomial.o Calculator.o UI.o main.o -o main
main.o:main.cpp
	g++ -c main.cpp -o main.o
Polynomial.o:Polynomial.cpp
	g++ -c Polynomial.cpp -o Polynomial.o
Calculator.o:Calculator.cpp
	g++ -c Calculator.cpp -o Calculator.o
UI.o:UI.cpp
	g++ -c UI.cpp -o UI.o
clean:
	rm *.o
	rm main
```

```
void UI::show()
{
	cout<<"\033[35mwelcome to polynomial calculator\033[0m\n";
	char ch;
	do{
		cout<<"\033[37m1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
\033[0m\n请选择: ";
		int choice; cin>>choice;
		switch(choice){
			case 1:this->cal->createPoly();break;
			case 2:this->cal->MIXoperator();break;
			case 3:this->cal->GetInverse();break;
			case 4:this->cal->GetMod();break;
			case 5:this->cal->GetRoot();break;
			case 6:this->cal->PrintPoly();break;
			default:cout<<"无该选项\n"; break;
		}
		cout<<"是否继续输入(y/n):"; cin>>ch;
	}while(ch!='n');
}
```

循环输入命令模板

实现功能1-6
输入提示
结束退出

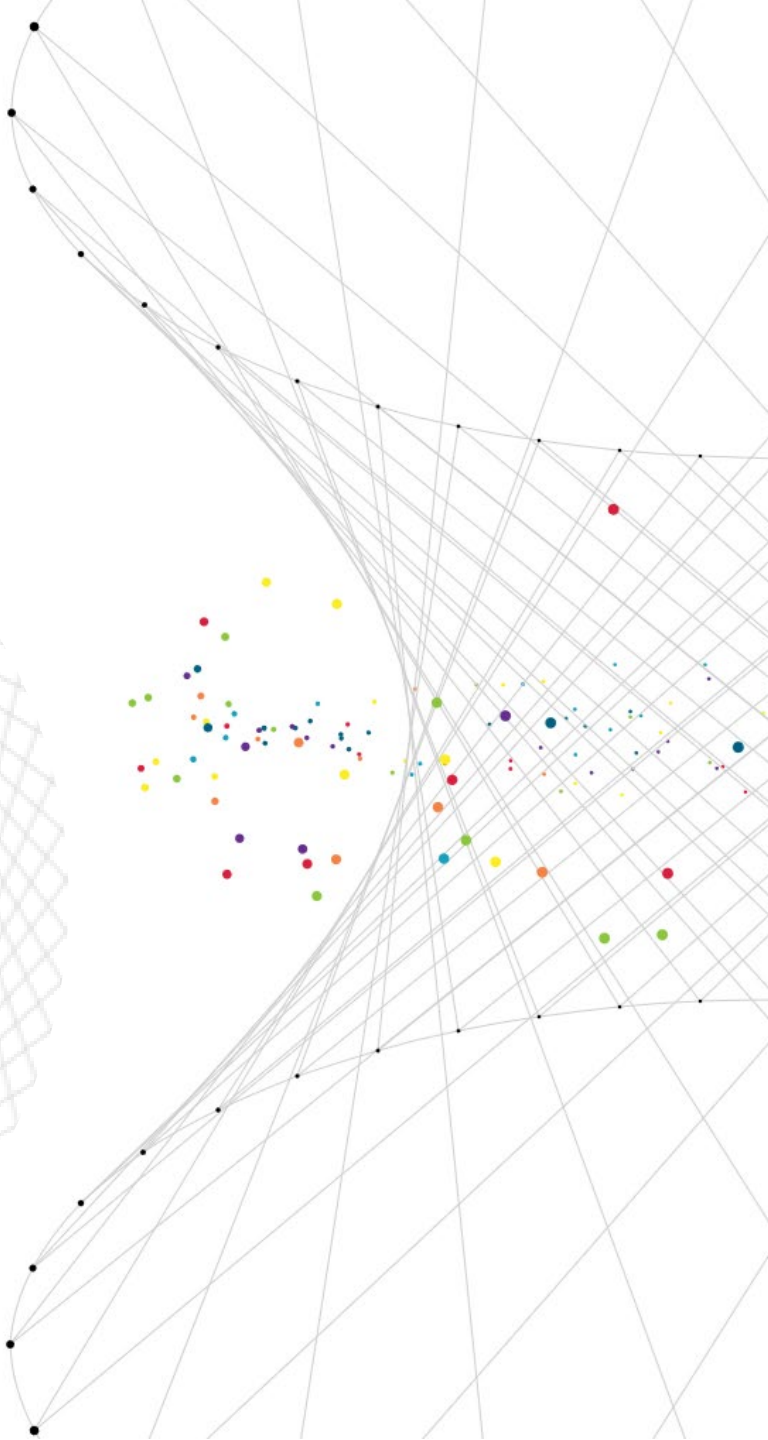
数据结构

PART THREE

PART THREE 数据结构

class Calculator

class Polynomial



PART THREE 数据结构

计算器类

用unordered_map<string,Polynomial> mapPoly捆绑多项式名称

计算器功能

表达式读取

```
class Calculator
{
        public:
                Calculator();
                ~Calculator();
                void createPoly();
                void addPoly(string name,Polynomial Poly);
                void MIXoperator();
                void GetInverse();
                void GetMod();
                void GetRoot();
                void PrintPoly();
        private:
                unordered_map<string,Polynomial> mapPoly;
                int postion=0;
                struct Myoperator{
                        char op;
                        double a,b;
                        int pos;
                };
                struct Myname{
                        string pname;
                        int pos;
                };
                stack<Myoperator> stkop;
                stack<Myname> stkname;
                int Priority(const char op);
                int inPriority(const char op);
                bool read(string exp);
};
```

计算器类

用unordered_map<string,Polynomial> mapPoly捆绑多项式名称

操作符结构

```
struct Myoperator
{
        char op;
        double a,b;
        int pos;
};
```

操作数结构

```
struct Myname{
        string pname;
        int pos;
};
```

用a,b 存放积分上下限

Pos便于比较先后次序 对单目操作符合法性判断

PART THREE 数据结构

多项式类

用vector<double> factors 存储高次开始的多项式系数

多项式运算

辅助函数

```
11 class Polynomial{
12 public:
13         Polynomial();
14         ~Polynomial();
15         Polynomial(double* cofficients,int len);
16         friend ostream &operator << (ostream &out, const Polynomial &Poly);
17         Polynomial operator + (const Polynomial& other)const;
18         Polynomial operator - (const Polynomial& other)const;
19         Polynomial operator * (const Polynomial& other)const;
20         Polynomial operator / (Polynomial& other);
21         Polynomial operator % (Polynomial& other);
22         Polynomial derivate();                              //求导
23         Polynomial integrate(double a, double b);           //积分
24         Polynomial inv(int len);                            //求逆
25         double value(double x);                             //求值
26         int leng();                                         //最高次
27
28 private:
29         vector<double> factors;                             //系数
30         Polynomial preverse();                              //翻转
31         Polynomial mod(int len);                            //模x^n
32         struct complex{
33                 double real,imag;
34                 complex() { real = 0;imag = 0; }
35                 complex(double x,double y){real=x; imag=y;}
36                 inline complex operator + (const complex b)const{return complex
   (real+b.real,imag+b.imag);}
37                 inline complex operator - (const complex b)const{return complex
   (real-b.real,imag-b.imag);}
38                 inline complex operator * (const complex b)const{return complex
   (real*b.real-imag*b.imag,real*b.imag+imag*b.real);}
39         };
40         void change(complex*y,int len)const;            //蝴蝶变换
41         void FFT(complex*y,int len,const int fla)const; //快速傅里叶变换
42 };
```

核心函数

PART FOUR

PART FOUR 核心函数

All
1 构造函数与析构函数
对象创建初始化数据成员；对象消亡回收内存空间

Polynomial
2 多项式操作符重载
多项式加法、乘法、除法、取模以及输出重载

Calculator
3 表达式合法性读取
针对混合运算的输入表达式 拆分表达式内容 并 判断其合法性

P / C
4 功能直接实现函数
每个功能相应的函数

UI
5 主界面函数
输出提示性信息，do-while中反复计算器操作

All
6 功能辅助函数
一些帮助功能更好实现的函数 如多项式乘法的FFT函数

PART FOUR 核心函数

Polynomial

2 多项式操作符重载

输出

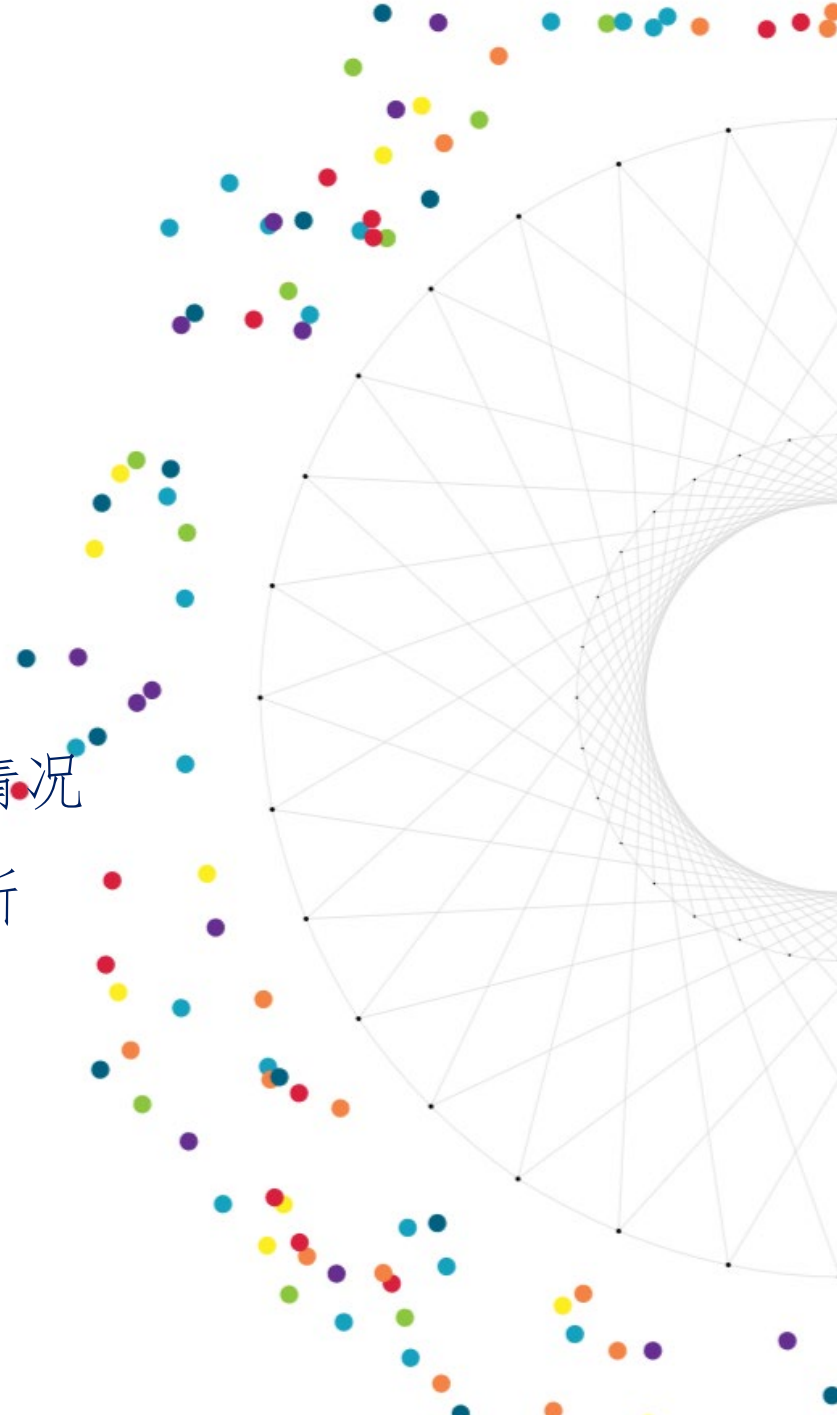
多项式加法、乘法、除法、取模以及输出重载

1

```
ostream& operator << (ostream &out,const Polynomial &Poly)
{
        int len=Poly.factors.size();
        bool flag=0;
        for(int index=len-1;index>=0;--index)
        {
                if(Poly.factors[index]!=0)
                {
                        if(flag&&Poly.factors[index]>0)
                                out<<"+";
                        if(Poly.factors[index] == -1&&index!=0)
                                out<<"-";
                        if(fabs(Poly.factors[index])!=1||index==0)
                                out<<Poly.factors[index];
                        if(index > 1)
                                out<<"x^"<<index;
                        else if(index==1)
                                out<<"x";
                        flag=1;
                }
        }
        return out;
}
```

输出情况的判断

从高次到低次，输出不显示系数为0的项，
系数1不显示，且不输出小数点后多余的0



PART FOUR 核心函数

Polynomial

2 多项式操作符重载

乘法

多项式加法、乘法、除法、取模以及输出重载

2 不运用暴力算法而是运用FFT算法使多项式乘法做到O(nlogn)

```
Polynomial Polynomial::operator *(const Polynomial& other)const{
        int len1=this->factors.size(),len2=other.factors.size(),len=1;
        while(len<len1*2 || len<len2*2) len<<=1;//分治DFT处理多项式长度只能为2^n
        complex *x1 = new complex[len],*x2 = new complex[len];
        for(int i=0;i<len1;++i) x1[i]=complex(this->factors[i],0);
        for(int i=len1;i<len;++i) x1[i]=complex(0,0);
        for(int i=0;i<len2;++i) x2[i]=complex(other.factors[i],0);
        for(int i=len2;i<len;++i) x2[i]=complex(0,0);
        FFT(x1,len,1);  FFT(x2,len,1);
        for(int i=0;i<len;++i)  x1[i]=x1[i]*x2[i];
        FFT(x1,len,-1);
        double* xx=new double[len];
        for(int i=0;i<len;++i)  xx[i]=x1[len-i-1].real;
        return Polynomial(xx,len);
}
```

PART FOUR 核心函数

Calculator

3 表达式合法性读取

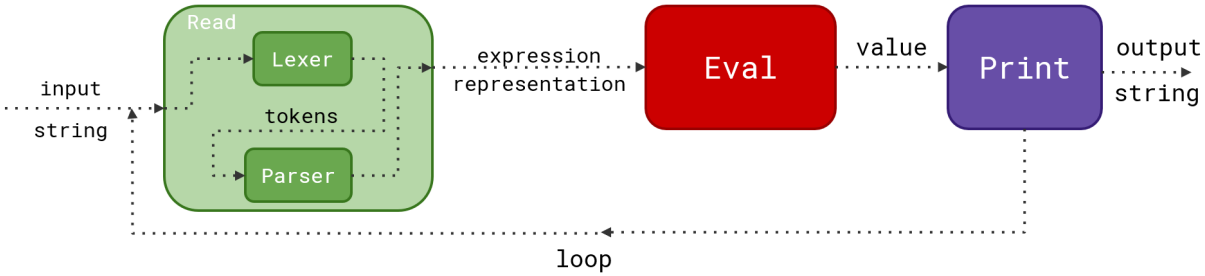
针对混合运算的输入表达式 拆分表达式内容 并 判断其合法性

混合运算过程：

具体方法：

用两个栈分别存多项式和运算符{+,*,$,!,()}，按优先级运算。

具体：



注：将$[a,b]经过结构体处理成 $

记$stack < Myoperator > stkop$和$stack < string > stkname$存放表达式

例

假如要对下面的表达式求值：

```
$[0,6](F+G)*H!
```

↓

```
$(F+G)*H!
```

PART FOUR 核心函数

```
1 $(F+G)*H!
```

我们从左往右开始扫描。首先遇到操作符'$'，入栈

stkop: **栈顶**: $

stkname: NULL

继续往后扫描，遇到'('直接入栈，'F'入栈，栈顶是左括号，'+'入栈，'G'入栈

stkop: **栈顶**: $(+

stkname: **栈顶**: FG

继续往后扫描，遇到右括号，它与栈顶操作符'+'相比，优先级要高，因此，将'+'出栈，弹出两个操作数'F','G'，计算结果得到'A'(设此处A=F+G)，并入栈：

stkop: **栈顶**: $(

stkname: **栈顶**: A

继续出栈，直到遇到左括号

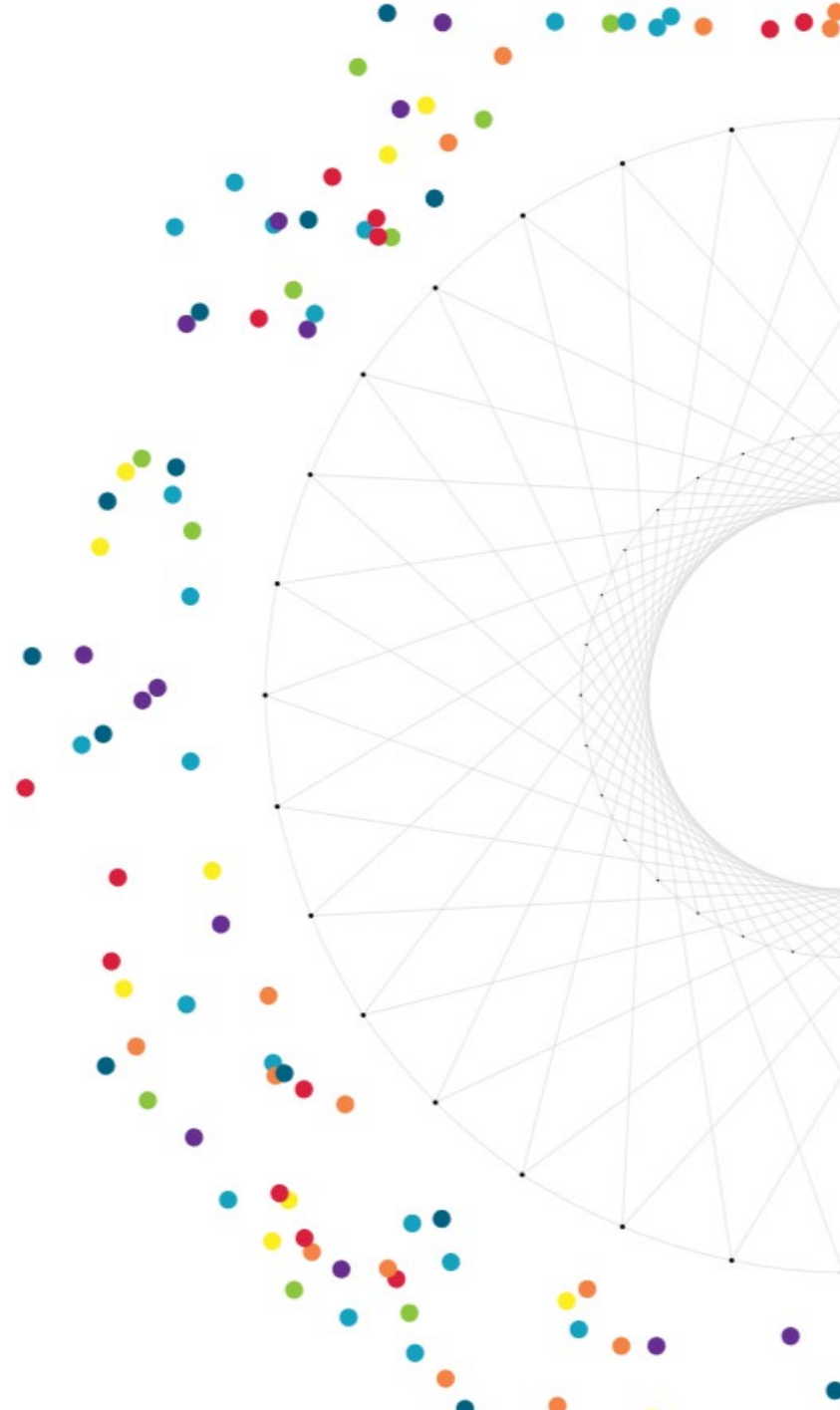
stkop: **栈顶**: $

stkname: **栈顶**: A

往后扫描一位不是！继续出栈，遇到$，将'$'出栈，弹出操作数'A'，计算结果得到'B'(设此处B=$A)，并入栈：

stkop: NULL

stkname: **栈顶**: B



PART FOUR 核心函数

```
1 $(F+G)*H!
```

继续往后扫描，遇到操作符'*',入栈:

stkop: **栈顶:** *

stkname: **栈顶:** B

继续往后扫描，遇到操作数'H'和操作符'!',入栈:

stkop: **栈顶:** *!

stkname: **栈顶:** BH

扫描结束，发现操作符栈不为空，弹出操作符'！'和操作数'H'，并进行计算，得到'C'(设此处C=B!)，入栈:

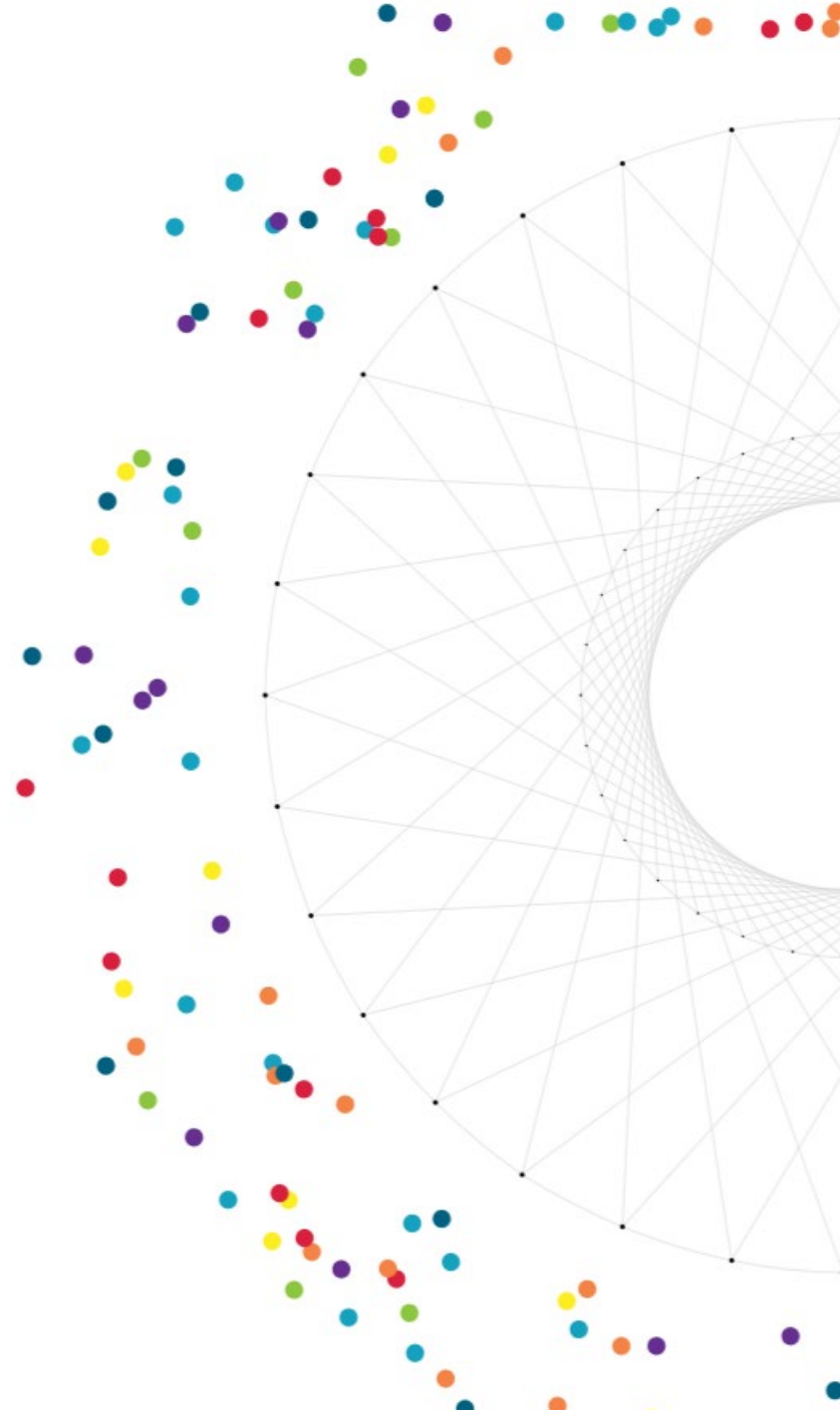
stkop: **栈顶:** *

stkname: **栈顶:** BC

发现操作符栈仍不为空，弹出操作符'*'和两个操作数，并进行计算，得到'D'(设此处D=B*C)，入栈,得到最终结果。

stkop: NULL

stkname: **栈顶:** D



PART FOUR 核心函数

Calculator

3 表达式合法性读取

针对混合运算的输入表达式 拆分表达式内容 并 判断其合法性

具体实现的伪代码:

```
int i = 0, len = exp.length();//exp[i]遍历exp
while(i<len){
    if(stkop.empty()&&stkname.size()==1&&i==len)//计算完成
        return result;
    if(exp is name)
        stkname.push(name);
    else if(exp is op){
        if(!stkop.empty() && priority(op)<in(stkop.top()) )
            op; pop;//栈顶优先级不低于该运算符的运算符运算
        stkop.push(op);
    }
    while (i == len && !stkop.empty())
        op; pop;
}
```

一致

PART FOUR 核心函数

P / C

4 功能直接实现函数

求逆

每个功能相应的函数

做法:倍增

从最简单的情况开始考虑，易得$P^{-1}(x) \equiv \frac{1}{a_0}(mod\ x)$

假设已知$P(x)P_0^{-1}(x) \equiv 1(mod\ x^{\lceil \frac{n}{2} \rceil})$，求$P(x)$在模$x^n$下的逆元$P^{-1}(x)$，有

$$P(x)P^{-1}(x) \equiv 1(mod\ x^{\lceil \frac{n}{2} \rceil})$$

$$P^{-1}(x) - P_0^{-1}(x) \equiv 0(mod\ x^{\lceil \frac{n}{2} \rceil})$$

两边平方，得到

$$P^{-1}(x) \equiv P_0^{-1}(x)\left(2 - P(x)P_0^{-1}(x)\right)\ (mod\ x^n)$$

参考资料：https://www.cnblogs.com/yoyoball/p/8724115.html

PART FOUR 核心函数

P / C

4 功能直接实现函数

除法

每个功能相应的函数

设$F(x)$的次数为n，$G(x)$的次数为m

构造变换$F^R(x) = x^n F\left(\frac{1}{x}\right)$，即 $F^R(x)$为$F(x)$的翻转

eg. $4x^3 + 3x^2 - x + 1$变化为 $x^3 - x^2 + 3x + 4$

对于等式$F(x) = Q(x)G(x) + R(x)$，运用上述变换，得到

$$F^R(x) = Q^R(x)G^R(x) + x^{n-m+1}R^R(x)$$

因为$Q^R(x)$的次数为$n - m < n - m + 1$，所以可以两边模x^{n-m+1}消除$R(x)$的影响，得到

$$F^R(x)\left(G^R(x)\right)^{-1} \equiv Q^R(x)\ (mod\ x^{n-m+1})$$

即可计算出$Q(x)$，再将其翻转得到$R(x)$

参考资料：https://www.cnblogs.com/yoyoball/p/8724115.html

PART FOUR 核心函数

P / C

4 功能直接实现函数

求根

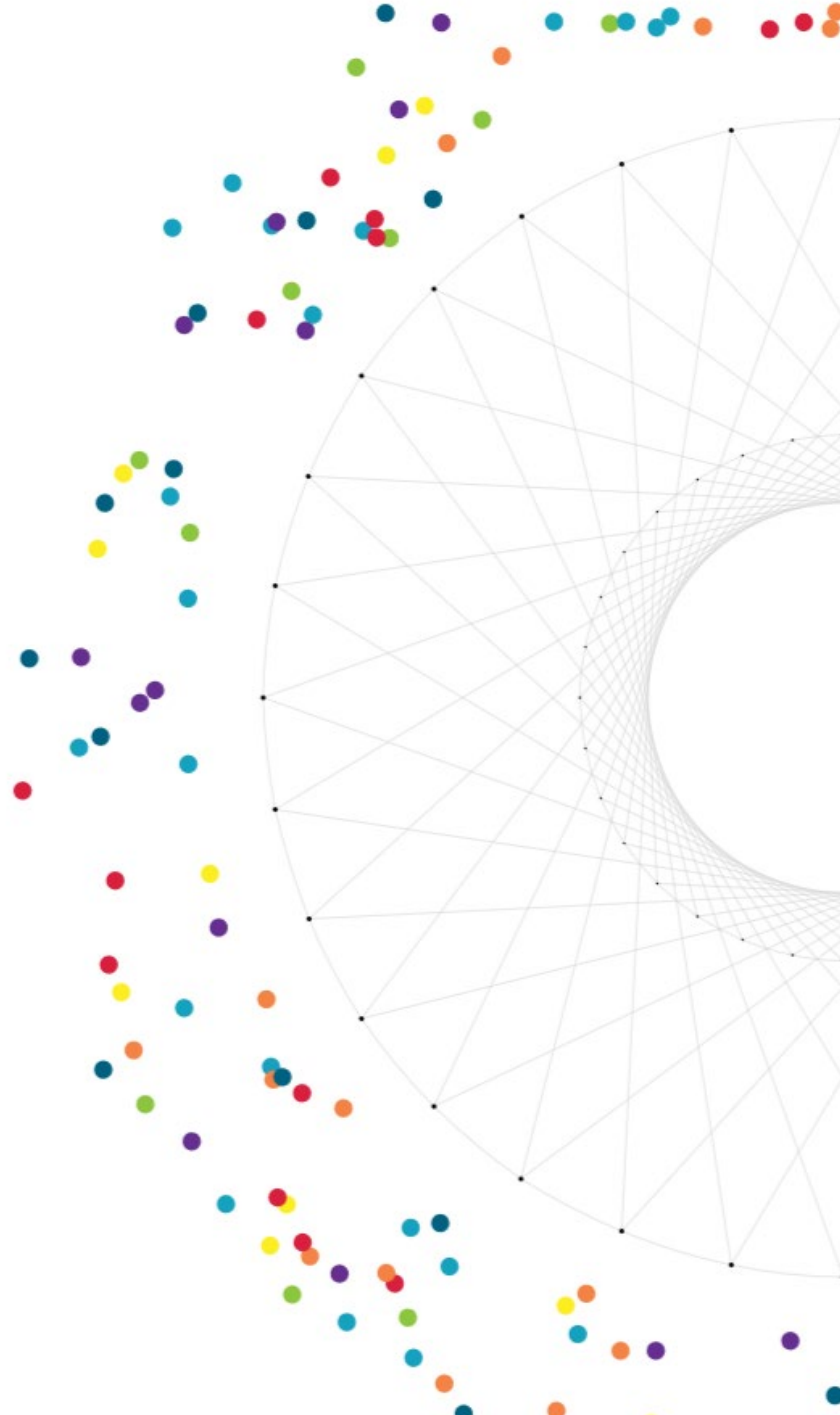
每个功能相应的函数

尽管通过因式分解和利用求根公式可以很方便的得出多项式方程的根，但大多数时候这个多项式的次数都很高，计算将变得非常复杂，因此，选择使用牛顿迭代法。

首先，要选择一个初始值x=x0，使得该初始值接近实根的值。然后，迭代计算如下的公式：

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}$$

不断迭代直到$|x_{i+1} - x_i| \leq 10^{-5}$



PART FOUR 核心函数

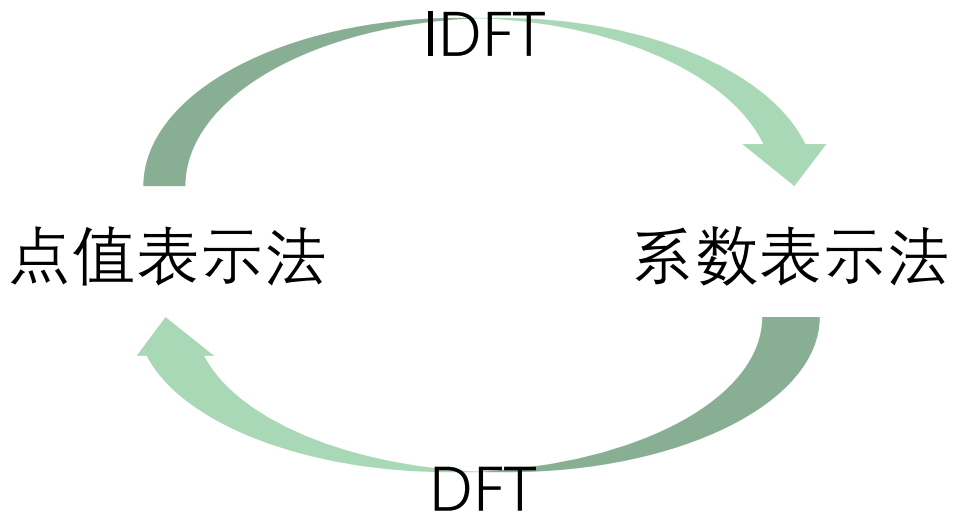
All

6 功能辅助函数

FFT

一些帮助功能更好实现的函数 如多项式乘法的FFT函数

多项式两种表示方法:系数表示法，点值表示法。转换关系如下:



FFT: 通过取某些特殊的x的点值来加速 DFT 和 IDFT 的过程。

参考资料：https://oi-wiki.org/math/poly/fft/#_1

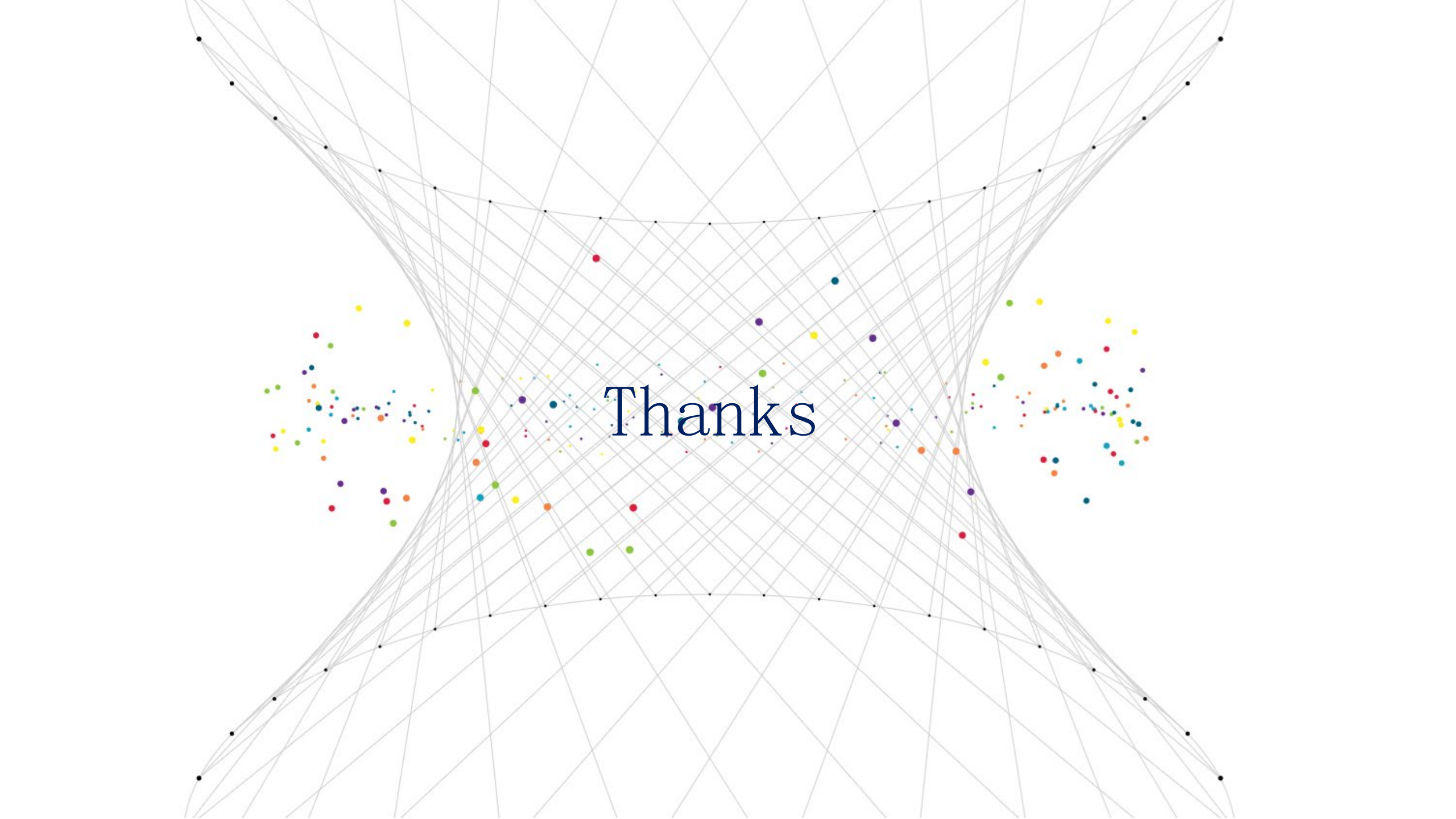
功能展示

PART FIVE

```
welcome to polynomial calculator
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 1
多项式长度: 5
请输入
1 6 -1 1 2
多项式名为:e
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 6
多项式名为:e
e = x^4+6x^3-x^2+x+2
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 5
多项式名为:e
e的存在实数根为:-0.583494
计算成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 1
多项式长度: 3
请输入
2 0 -1
多项式名为:abcd
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 2
请输入表达式:($[0.3]abcd+e)!
输入错误,是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 2
请输入表达式:($[0,3]abcd+e)!
($[0,3]abcd+e)!=4x^3+18x^2-2x+1
输入成功，是否继续输入(y/n):
```

样例

```
welcome to polynomial calculator
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 2
请输入表达式:$[0,6](f+g)*h!
$[0,6](f+g)*h!=432
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 2
请输入表达式:$[0,6]f+g*h!
$[0,6]f+g*h!=x^3+x+70
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 3
多项式名为:f
逆元为:f^-1=-0.125x^2+0.25x+0.5
求逆成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 4
请输入表达式:g/f
商为:x+1余数为:2
计算成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 5
多项式名为:g
g的存在实数根为:-1.3788
计算成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: 6
多项式名为:f
f = x^2-x+2
输入成功，是否继续输入(y/n):y
1.输入 2.混合运算 3.求逆元 4.除法/取模运算 5.求根 6.查看
请选择: g
选择失败，请重试(退出请输入-1):
```


Thanks