Homework 5:

Implement FFT(DIT & DIF) & IFFT by C++.

一、 Fourier 序列类的定义

```
class fourierSequence {
    Complex *data;
    Complex *wn;
    int length;
    int bitLength;

    int reverseNum(const int xx,const int bitLength)const;
    void reverseList();
    void nDIT_FFT(Complex *x,const int n);
    void nDIF_FFT(Complex *x,const int n);
    void conjugate();

public:
    fourierSequence();
    ~fourierSequence(){delete []data;delete []wn;}

    void DIT_FFT();
    void DIF_FFT();
    void IFFT();

    void print()const;
};
```

私有成员中，序列 data 用于存储数据序列 $x(n)$ 或 $X(k)$，序列 wn 表示 W_N^n。length 表示序列长 N，bitLength 表示序列位长 $\log_2 N$。reverseNum()用于计算变址地址，reverseList()则对整个序列进行倒序，conjugate()用于计算共轭序列 $x^*(n)$ 或 $X^*(K)$，nDIT_FFT()、nDIF_FFT()则用于 DIT_FFT()、DIF_FFT()函数过程中。

公有函数中，DIT_FFT()、DIF_FFT()函数分别用基 2 时选和基 2 频选对序列进行 FFT，IFFT()利用共轭性质对序列进行 IFFT。

二、 倒序和变址的实现

分析知，倒序的实质是对地址的二进制码取倒序，即 $x(n_2n_1n_0)_2 \rightarrow x(n_0n_1n_2)_2$。因而计算变址函数通过对原地址做%2 运算得到最低位然后重构得到新地址。计算过程如下：

```
int fourierSequence::reverseNum(const int xx, const int bitLength)const{
    int x=xx;
    int result=0;
    for (int i=0; i<bitLength-1; i++) {
        result=result*2+x%2;
        x/=2;
    }
    return result*2+x%2;
}
```

由对称性，倒序过程只需对一半序列进行互换即可。因而整个序列的倒序函数实现如下：

```
void fourierSequence::reverseList(){
    int newAddress;
    Complex temp;
    for (int i=1; i<length/2; i++) {
        newAddress=reverseNum(i, bitLength);
        if (i!=newAddress) {
            temp=data[newAddress];
            data[newAddress]=data[i];
            data[i]=temp;
        }
    }
}
```

三、 基 2 时选 FFT 的实现

分析如图 1 所示，N 位序列的 DIT-FFT 是对两个 N/2 位的序列进行蝶形运算，而 N/2 位的序列对两个 N/4 位的序列进行蝶形运算……

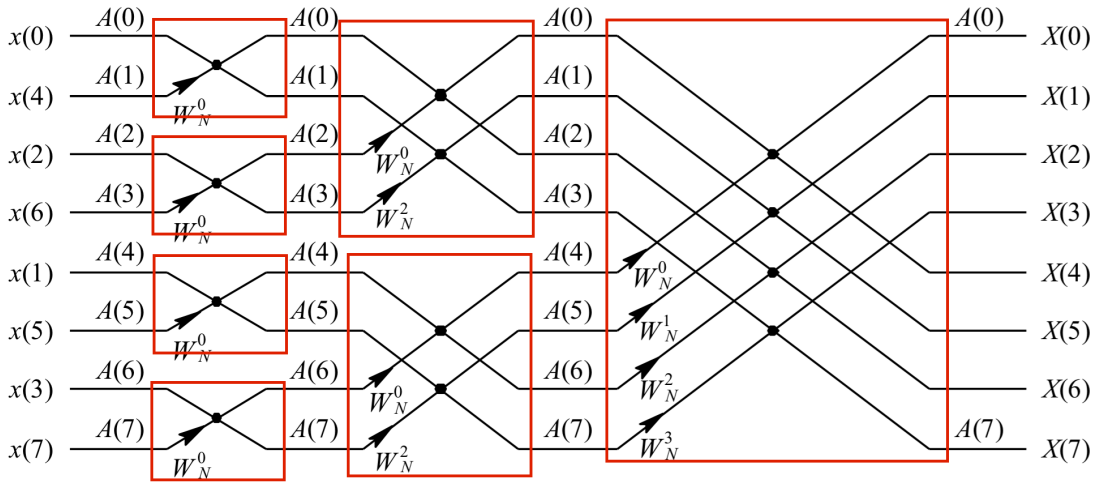

图 1 N点DIT-FFT运算流图(N=8)

因而 DIT_FFT()函数的基本环节为 N 点蝶形运算，定义为 nDIF_FFT(Complex *x, const int n)。而其起始地址和位数控制由函数 DIT_FFT()实现。具体代码

如下：

```
void fourierSequence::DIT_FFT(){
    reverseList();
    for (int i=1; i<=bitLength; i++) {
        int time=pow(2, bitLength-i);
        for (int j=0; j<time; j++) {
            nDIT_FFT(data+int(j*pow(2, i)), pow(2, i));
        }
    }
}
void fourierSequence::nDIT_FFT(Complex *x,const int n){
    Complex temp1,temp2;
    int k=length/n;
    for (int i=0; i<n/2; i++) {
        temp1=*(x+i)+(*(x+i+n/2))*(wn[i*k]);
        temp2=*(x+i)-(*(x+i+n/2))*(wn[i*k]);
        *(x+i)=temp1;
        *(x+i+n/2)=temp2;
    }
}
```

四、 基 2 频选 FFT 的实现

分析如图 2 所示，N 位序列的 DIF-FFT 的实现方法与 DIT-FFT 及其类似。只是在循环控制和蝶形运算上稍有不同。在此不再赘述，具体代码详见附录。

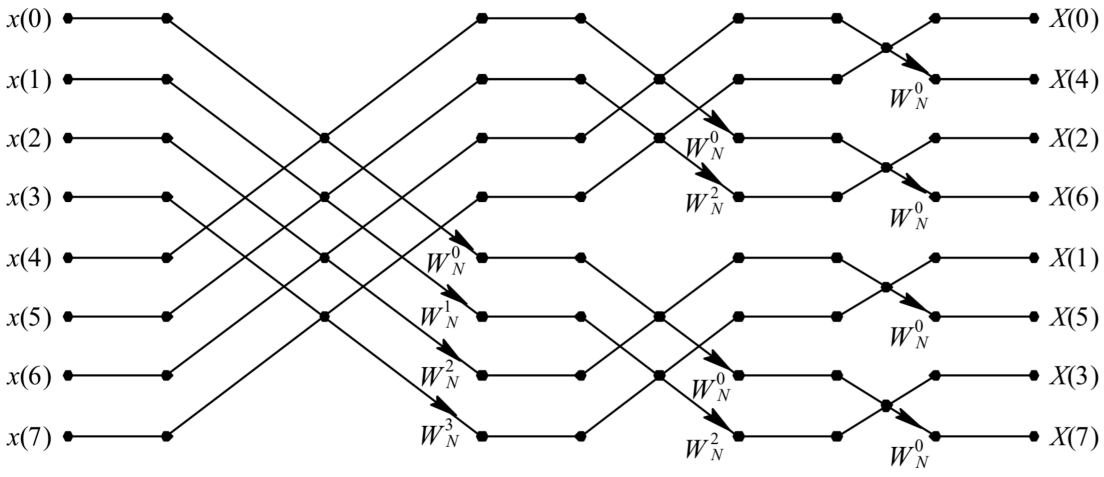

图 2 DIF－FFT 运算流图(N=8)

五、 IFFT 的实现

根据 FFT 运算性质：

$$x(n)=[x^*(n)]^*=\{[\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk}]^*\}^*=[\frac{1}{N}\sum_{k=0}^{N-1}X^*(k)W_N^{nk}]^*=\frac{1}{N}\{DFT[X^*(k)]\}^*$$

因而 IFFT 运算可分为四步：

1. 对序列 $X(k)$ 取共轭 $X^*(k)$

2. 调用 FFT 函数计算 $DFT[X^*(k)]$
3. 对计算结果再取共轭
4. 对计算结果乘 $\frac{1}{N}$

实现代码如下：

```
void fourierSequence::IFFT(){
    conjugate();
    DIF_FFT();
    conjugate();
    for (int i=0; i<length; i++) {
        data[i]=data[i]/length;
    }
}
```

六、 运行结果

```
The original sequence:
1+5i        9+2i        9        4+i        3i        4+i        1+4i        1

After DIT_FFT:
29+16i        -0.171573-11.6569i        -7-4i        5+4.34315i        -7+8i        -5.82843-0.343146i        -11+12i        5+15.6569i

After DIF_FFT:
29+16i        -0.171573-11.6569i        -7-4i        5+4.34315i        -7+8i        -5.82843-0.343146i        -11+12i        5+15.6569i

After IFFT:
1+5i        9+2i        9        4+i        3i        4+i        1+4i        1
Program ended with exit code: 0
```

七、 算法的时间优化

在计算过程中，W_N^k 使用较多，因而在 fourierSequence 类构造函数中直接计算 W_N^k 并缓存，之后直接读取，以减少计算量。

实际计算中，并未根据 W_N^k 的周期性和对称性进行化简。考虑到该运算时间复杂度为 $O(n)$，而 FFT 等主函数时间复杂度为 $O(n\log n)$，且该计算只在类构造初期运算一次，对整体计算效率影响不大。其次也由于时间有限。

计算 IFFT 时，最后的取共轭和乘 $\frac{1}{N}$ 可以放在一个循环里以减少运算，但考虑到代码的整体性，并未进行优化。

八、 其他分析

MatLab 给我们提供了一个很好的应用工具，而过好的应用工具则会限制我们的学习，尤其是对底层技术和算法的掌握和了解，而 C++这一语言不仅让我对 FFT 有了更深刻的认识，还极大地提高了计算效率（MatLab 的计算效率极低，尤其是进行大型运算）。

该类定义的最大不足在 fourierSequence 类构造函数。因本程序重点在于两种 FFT 和 IFFT 的实现，故构造函数中仅生成一个固定序列用于测试，距离作为封装类投入其他计算和科学研究还有一定改进空间。

关于代码，一定要在充分了解算法的数学模型基础上先构建出函数原型，设计好接口和程序结构，再进行实现和细化。好的结构能极大程度地提升编写效率。在我第一次编写此程序时，由于程序架构设计不够完善，不停地改改修修，最后还是没能实现。经历失败后，脱离电脑对算法重新进行了细致的分析，决定采用面向对象的设计方法，对整个程序架构和函数原型重新设计，最后逐步实现完善和优化。最让我开心的，这次程序一次运行成功，没有 debug 等环节，这在以前的复杂程序设计中是前所未有的。

该程序系 OS X Yosemite 下 XCode 6.1 编辑编译完成。

九、 附代码

```
//
//  Complex.h
//  Complex
//
//  Created by F on 13-5-10.
//  Copyright (c) 2013年 F. All rights reserved.
//

#ifndef __Complex__Complex__
#define __Complex__Complex__

#include <iostream>
#include <cmath>
using namespace std;
class Complex{
    friend Complex operator+(const Complex &c1,const Complex &c2);
    friend Complex operator-(const Complex &c1,const Complex &c2);
    friend Complex operator*(const Complex &c1,const Complex &c2);
    friend Complex operator/(const Complex &c1,const Complex &c2);
    friend ostream& operator<<(ostream & os,const Complex &c);
private:
    double a,b;
public:
    Complex(double A=0,double B=0):a(A),b(B){
        if (abs(a)<10e-6) a=0;
        if (abs(b)<10e-6) b=0;
```

```
    };
    void value(double A=0,double B=0);
    void conjugate();

};
Complex expj(double x);
#endif /* defined(__Complex__Complex__) */
//
//  Complex.cpp
//  Complex
//
//  Created by F on 13-5-10.
//  Copyright (c) 2013年 F. All rights reserved.
//

#include "Complex.h"

Complex operator+(const Complex &c1,const Complex &c2)
{
    return Complex(c1.a+c2.a,c1.b+c2.b);
}
Complex operator-(const Complex &c1,const Complex &c2)
{
    return Complex(c1.a-c2.a,c1.b-c2.b);
}
Complex operator*(const Complex &c1,const Complex &c2)
{
    return Complex(c1.a*c2.a-c1.b*c2.b,c1.a*c2.b+c1.b*c2.a);
}
Complex operator/(const Complex &c1,const Complex &c2)
{
    return
Complex((c1.a*c2.a+c1.b*c2.b)/(c2.a*c2.a+c2.b*c2.b),(c1.b*c2.a-c1.a*
c2.b)/(c2.a*c2.a+c2.b*c2.b));
}
Complex expj(double x){
    return Complex(cos(x),sin(x));
}
void Complex::conjugate(){
    b=-b;
}
void Complex::value(double A,double B){
    a=A;b=B;
    if (abs(a)<10e-6) {
```

```
        a=0;
    }
    if (abs(b)<10e-6) {
        b=0;
    }
};

ostream& operator<<(ostream & os,const Complex &c)
{
    if(c.a!=0)
    {
        os<<c.a;
        if(c.b==-1)
        {
            os<<"-i";
        }
        else if (c.b==1)
        {
            os<<"+i";
        }
        else if (c.b<0)
        {
            os<<c.b<<'i';
        }
        else if (c.b>0)
        {
            os<<'+'<<c.b<<'i';
        }
        return os;
    }
    else
    {
        if (c.b==0) {
            os<<0;
        }
        else if(c.b==1)
            os<<'i';
        else if(c.b==-1)
            os<<"-i";
        else
        {
            os<<c.b<<'i';
        }
        return os;
```

```
    }

}
//
//  fourierSequence.h
//  FFT
//
//  Created by F on 14/11/2.
//  Copyright (c) 2014年 F. All rights reserved.
//

#ifndef __FFT__fourierSequence__
#define __FFT__fourierSequence__
#include "Complex.h"
class fourierSequence {
    Complex *data;
    Complex *wn;
    int length;
    int bitLength;

    int reverseNum(const int xx,const int bitLength)const;
    void reverseList();
    void nDIT_FFT(Complex *x,const int n);
    void nDIF_FFT(Complex *x,const int n);
    void conjugate();

public:
    fourierSequence();
    ~fourierSequence(){delete []data;delete []wn;}

    void DIT_FFT();
    void DIF_FFT();
    void IFFT();

    void print()const;

};

#endif /* defined(__FFT__fourierSequence__) */
//
//  fourierSequence.cpp
//  FFT
//
//  Created by F on 14/11/2.
```

```
//  Copyright (c) 2014年 F. All rights reserved.
//

#include "fourierSequence.h"
#include <iostream>
#include <cmath>
const double pi=M_PI;
using namespace std;

fourierSequence::fourierSequence(){
    length=8;
    data=new Complex[length];
    data[0].value(1,5);
    data[1].value(9,2);
    data[2].value(9,0);
    data[3].value(4,1);
    data[4].value(0,3);
    data[5].value(4,1);
    data[6].value(1,4);
    data[7].value(1,0);

    bitLength=log(length)/log(2);

    wn=new Complex[length];
    for (int i=0; i<length; i++) {
        wn[i]=expj(-2*pi/length*i);
    }
}


int fourierSequence::reverseNum(const int xx, const int bitLength)const{
    int x=xx;
    int result=0;
    for (int i=0; i<bitLength-1; i++) {
        result=result*2+x%2;
        x/=2;
    }
    return result*2+x%2;
}
void fourierSequence::reverseList(){
    int newAddress;
    Complex temp;
    for (int i=1; i<length/2; i++) {
        newAddress=reverseNum(i, bitLength);
```

```
        if (i!=newAddress) {
            temp=data[newAddress];
            data[newAddress]=data[i];
            data[i]=temp;
        }
    }
}

void fourierSequence::print()const{
    for (int i=0; i<length; i++) {
        cout<<data[i]<<"\t\t";
    }
    cout<<endl;
}


void fourierSequence::nDIT_FFT(Complex *x,const int n){
    Complex temp1,temp2;
    int k=length/n;
    for (int i=0; i<n/2; i++) {
        temp1=*(x+i)+(*(x+i+n/2))*(wn[i*k]);
        temp2=*(x+i)-(*(x+i+n/2))*(wn[i*k]);
        *(x+i)=temp1;
        *(x+i+n/2)=temp2;
    }
}

void fourierSequence::nDIF_FFT(Complex *x, const int n){
    Complex temp1,temp2;
    int k=length/n;
    for (int i=0; i<n/2; i++) {
        temp1=*(x+i)+(*(x+i+n/2));
        temp2=(*(x+i)-(*(x+i+n/2)))*(wn[i*k]);
        *(x+i)=temp1;
        *(x+i+n/2)=temp2;
    }
}

void fourierSequence::DIT_FFT(){
    reverseList();
    for (int i=1; i<=bitLength; i++) {
        int time=pow(2, bitLength-i);
        for (int j=0; j<time; j++) {
            nDIT_FFT(data+int(j*pow(2, i)), pow(2, i));
```

```
        }
    }
}

void fourierSequence::DIF_FFT(){
    for (int i=bitLength; i>0; i--) {
        int time=pow(2, bitLength-i);
        for (int j=0; j<time; j++) {
            nDIF_FFT(data+int(j*pow(2, i)), pow(2, i));
        }
    }
    reverseList();
}
void fourierSequence::conjugate(){
    for (int i=0; i<length; i++) {
        data[i].conjugate();
    }
}

void fourierSequence::IFFT(){
    conjugate();
    DIF_FFT();
    conjugate();
    for (int i=0; i<length; i++) {
        data[i]=data[i]/length;
    }
}
//
//  main.cpp
//  FFT
//
//  Created by F on 14/11/2.
//  Copyright (c) 2014年 F. All rights reserved.
//
#include <iostream>
#include "fourierSequence.h"
using namespace std;

int main(){
    fourierSequence test,test2;

    cout<<"The original sequence: "<<endl;
    test.print();
    cout<<endl;
```

```
    cout<<"After DIT_FFT: "<<endl;
    test.DIT_FFT();
    test.print();
    cout<<endl;

    cout<<"After DIF_FFT: "<<endl;
    test2.DIF_FFT();
    test2.print();
    cout<<endl;

    cout<<"After IFFT: "<<endl;
    test.IFFT();
    test.print();

    return 0;
}
```