1.

코드

Complex.h

#pragma once

template <typename T>

class Complex

{

public:

Complex(T = 0.0, T = 0.0);

Complex add(const Complex&)const;

Complex subtract(const Complex&);

void printComplex();

void setComplexNumber(T, T);

Complex operator-(const Complex&);

private:

T realPart;

T imaginaryPart;

};

template <typename T>

Complex<T>::Complex(T real, T imaginary)

: realPart(real), imaginaryPart(imaginary) {}

template <typename T>

Complex<T> Complex<T>::add(const Complex& comp) const

{

Complex<T> temp(realPart + comp.realPart, imaginaryPart+comp.imaginaryPart);

return temp;

}

template <typename T>

Complex<T> Complex<T>::subtract(const Complex& comp)

{

Complex<T> temp(realPart - comp.realPart, imaginaryPart - comp.imaginaryPart);

return temp;

}

template <typename T>

void Complex<T>::printComplex()

{

cout << "(" << realPart;

if (imaginaryPart >= 0)

{

cout << "+";

}

cout<< imaginaryPart << "i)";

}

template <typename T>

void Complex<T>::setComplexNumber(T r, T i)

{

realPart = r;

imaginaryPart = i;

}

template <typename T>

Complex<T> operator+(const Complex<T>& comp1, const Complex<T>& comp2)

{

return comp1.add(comp2);

}

template <typename T>

Complex<T> Complex<T>::operator-(const Complex& comp)

{

return this->subtract(comp);

}

main\_Complex.h

#include <iostream>

using namespace std;

#include "Complex.h"

int main()

{

Complex<double> a(15.0, 20.3);

Complex<double> b(5.5, 7.6);

Complex<int> c(50, 10);

Complex<int> d(40, 55);

Complex<double> e = a + b;

Complex<int> f = c - d;

a.printComplex();

cout << "+";

b.printComplex();

cout << " = ";

e.printComplex();

cout << "\n";

c.printComplex();

cout << "+";

d.printComplex();

cout << " = ";

f.printComplex();

return 0;

}



2.

코드

Book.h

#ifndef BOOK\_H

#define BOOK\_H

#include<iostream>

#include<string>

using namespace std;

class Book

{

private:

string book;

public:

Book(string);

void setbook(string);

string getbook();

virtual void print() const;

virtual ~Book();

};

#endif

Book.cpp

#include "Book.h"

#include <iostream>

#include <iomanip>

using namespace std;

Book::Book(string b)

{

setbook(b);

cout << "Base Book - Constructor for " << getbook() << endl;

}

void Book::setbook(string b)

{

book = b;

}

string Book::getbook()

{

return book;

}

void Book::print() const

{

cout << "Book" << endl;

}

Book::~Book()

{

cout << "Base Book - Destructor for " << getbook() << endl;

}

Novel.h

#ifndef NOVEL\_H

#define NOVEL\_H

#include<iostream>

using namespace std;

#include "Book.h"

class Novel : public Book

{

public:

Novel(string);

void print() const;

~Novel();

};

#endif

Novel.cpp

#include <iostream>

#include <iomanip>

using namespace std;

#include "Novel.h"

Novel::Novel(string novel)

: Book(novel)

{

setbook(novel);

cout << "Derived Novel - Constructor for " << getbook() << endl;

}

void Novel::print() const

{

cout << "Novel" << endl;

}

Novel::~Novel()

{

cout << "derived Novel - Destructor " << getbook() << endl;

}

Biography.h

#ifndef BIOGRAPHY\_H

#define BIOGRAPHY\_H

#include<iostream>

using namespace std;

#include "Book.h"

class Biography : public Book

{

public:

Biography(string);

void print() const;

~Biography();

};

#endif

Biography.cpp

#include <iostream>

#include <iomanip>

using namespace std;

#include "Biography.h"

Biography::Biography(string Bio)

: Book(Bio)

{

cout << "Derived Biography - Constructor for " << getbook() << endl;

}

void Biography::print() const

{

cout << "Biography" << endl;

}

Biography::~Biography()

{

cout << "derived Biography - Destructor" << getbook() << endl;

}

main\_Book.cpp

#include <iostream>

using namespace std;

#include "Novel.h"

#include "Biography.h"

int main()

{

Book\* NovelPtr = new Novel("The Alchemist");

Book\* BiographyPtr = new Biography("Paulo Coelho");

NovelPtr->print();

BiographyPtr->print();

delete NovelPtr;

delete BiographyPtr;

return 0;

}





