MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

“KHARKOV POLYTECHNICAL INSTITUTE”

LABORATORY WORK № 2

# “Using Polymorphism and Templates in C++”

Created by student of 1.КН.201.8г

Chukwu Irele Omike

Checked by

KHARKIV 2019

## 1 Training Assignment

### 1.1 The Hierarchy of Classes

Implement classes "Human," "Citizen", "Student", "Employee". Each class must implement virtual function that shows related data on the screen. Create an array of pointers to different objects of the class hierarchy. Create a cycle and display data that represents objects of different types.

### 1.2 Using Polymorphism

Create a class for solving problem set in task 1.2 of the [sixth laboratory training](http://iwanoff.inf.ua/programming_1/LabTraining06.html) from previous semester. The class should contain at least two member functions: a function that returns some value according to individual task, and pure virtual function that is called from the previous one and defines the left side of the equation or researched function (according to the task).

Class should be allocated in a separate header file. The relevant implementation file should contain the definition of non-abstract member functions.

Another translation unit should contain a class derived from the previous one. This class should contain the definition of the virtual function, which is a subject of investigation. Create an object of the derived class and implement the individual task in main() function.

Note: You should add some base class member functions that will calculate the first (second) derivative.

### 1.3 Class Template for Representation of Two-Dimensional Array

Convert class created in the task 1.2 of the previous laboratory training into class template. Implement global template function that returns minimum array item. Create arrays of integers, real numbers, and simple fractions (previously created class) in main() function. For these three arrays you should test the function of finding the minimum value, as well as other class features with catching possible exceptions. You should also solve the problem from the individual task.

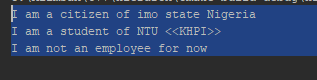
Note: in order to be able to find the minimum value in the array of fractions you should overload comparison operation for objects of the class "Simple fraction".

Task one :

The code :

#include <iostream>  
  
using std::cout;  
  
class Human {  
public:  
 virtual void showStatus() {  
 cout << "Hello, I am a human\n";  
 }  
};  
  
class Student : public Human {  
public:  
 void showStatus() {  
 cout << "I am a student of NTU <<KHPI>>\n";  
 }  
};  
  
class Citizen : public Human  
{  
public:  
 void showStatus() {  
 cout << "I am a citizen of imo state Nigeria\n";  
 }  
};  
  
class Employee : public Human {  
public:  
 void showStatus() {  
 cout << "I am not an employee for now\n";  
 }  
};  
  
int main() {  
 Human \*human;  
  
 Citizen \*citizenAndrew = new Citizen();  
 Student \*studentNTU = new Student();  
 Employee \*employeeNone = new Employee();  
  
 Human \*arr[3];  
 arr[0] = citizenAndrew;  
 arr[1] = studentNTU;  
 arr[2] = employeeNone;  
  
 for (int i = 0; i < 3; i++)  
 arr[i]->showStatus();  
  
 std::cin.get();  
 return 0;  
}

Execution :



Task two :

The code :

Main.cpp

#include <iostream>  
#include "Virtual.h"  
  
using std::cout;  
  
class MyDichotomy : public FuncVir  
{  
 virtual double f(double x) {  
 return x \* x\*x + x \* x\*x + x \* x;  
 }  
};  
  
int main() {  
  
 MyDichotomy d;  
 cout << d.getMinValSecDer(-1, 1);  
 system("pause");  
 return 0;  
}

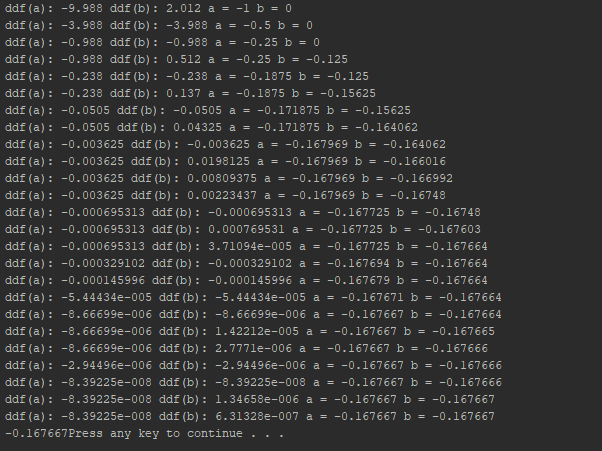
Virtual.h

//  
// Created by alimsah on 5/16/19.  
//  
  
#ifndef POLYMORPH\_VIRTUAL\_H  
#define POLYMORPH\_VIRTUAL\_H  
class FuncVir  
 {  
 public:  
  
 double eps = 0.001;  
 double firstDer(double a);  
 double secondDer(double a);  
 double getMinValSecDer(double a, double b, double eps = 0.0000001);  
 virtual double f(double x) = 0;  
  
 };  
#endif //POLYMORPH\_VIRTUAL\_H

FuncVir.cpp

#include "Virtual.h"  
#include <iostream>  
#include <cmath>  
  
using std::cout;  
  
double FuncVir::firstDer(double a)  
{  
 return (f(a + eps) - f(a)) / eps;  
}  
  
double FuncVir::secondDer(double a)  
{  
 double f1 = firstDer(a);  
 double f2 = firstDer(a + eps);  
 double b = (f2 - f1) / eps;  
 //cout << "Second der: " << b << "\n";  
 return b;  
}  
  
double FuncVir::getMinValSecDer(double a, double b, double eps)  
{  
 double x;  
 do {  
 x = (a + b) / 2;  
 if (secondDer(a) \* secondDer(x) <= 0) {  
 b = x;  
 }  
 else {  
 a = x;  
 }  
 cout << "ddf(a): " << secondDer(a) << " ddf(b): " << secondDer(x) <<  
 " a = " << a << " b = " << b << "\n";  
 } while (fabs(b - a) > eps);  
  
 //(f2 - f1) / eps  
  
  
 return x;  
}

Execution:



Task 3:

The code :

Main.cpp

#include "Fraction.h"  
#include <iostream>  
#include <iomanip>  
#include <cmath>  
  
using std::cout;  
using std::cin;  
using std::istream;  
using std::endl;  
using std::ostream;  
  
  
template <typename T> class Matrix2D {  
  
 friend ostream& operator <<(ostream& out, const Matrix2D<T>& m) {  
 for (int i = 0; i < m.rows; i++) {  
 for (int j = 0; j < m.cols; j++) {  
 out << std::setw(2) << m.matrix[i][j] << ' ';  
 }  
 out << endl;  
 }  
 return out;  
 }  
  
 friend istream& operator >>(istream& in, Matrix2D<T>& m) {  
 for (int i = 0; i < m.rows; i++) {  
 for (int j = 0; j < m.rows; j++) {  
 in >> m.matrix[i][j];  
 }  
 }  
 return in;  
 }  
  
private:  
 T \* \*matrix;  
 int rows;  
 int cols;  
 int next\_indexR;  
 int next\_indexC;  
public:  
 class OutOfBounds  
 {  
 int outOfBounIndexRow;  
 int outOfBounIndexCol;  
 public:  
 OutOfBounds(int x, int y) : outOfBounIndexRow(x), outOfBounIndexCol(y) {}  
 int getOutOfBounIndexRow() const { return outOfBounIndexRow; }  
 int getOutOfBounIndexCol() const { return outOfBounIndexCol; }  
 };  
  
 Matrix2D() {  
 matrix = alloc(0, 0);  
 next\_indexR = 0;  
 next\_indexC = 0;  
 }  
  
 ~Matrix2D() {  
 dealloc();  
 }  
  
 T& operator() (int i, int j) {  
 if (i >= rows || j >= cols)  
 throw OutOfBounds(i, j);  
 return matrix[i][j];  
 }  
 const T& operator() (int i, int j) const {  
 if (i >= rows || j >= cols)  
 throw OutOfBounds(i, j);  
 return matrix[i][j];  
 }  
  
 int getRows() { return rows; }  
 int getCols() { return cols; }  
 int getNextIndexR() { return next\_indexR; }  
 int getNextIndexC() { return next\_indexC; }  
  
 //void addElement(T element);  
  
  
  
 Matrix2D<T> operator+(const Matrix2D& m) const {  
 // first, make sure matrices can be added. if not, return original matrix  
 if (rows != m.rows || cols != m.cols) {  
 cout << "Matrix sizes do not match.";  
 return (\*this);  
 }  
 else {  
 //matrix new\_mat(row,col);  
 Matrix2D<T> new\_mat;  
 new\_mat.alloc(rows, cols);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 new\_mat(i, j) = matrix[i][j] + m(i, j);  
 //cout << "Sum " << new\_mat(i, j) << " ";  
 }  
 //cout << "\n";  
 }  
 cout << "Sum:\n" << new\_mat;  
 return new\_mat;  
 }  
 }  
  
 Matrix2D<T> operator-(const Matrix2D& m) {  
 Matrix2D<T> resultMatrix;  
  
 // first, make sure matrices can be added. if not, return original matrix  
 if (rows != m.rows || cols != m.cols) {  
 cout << "Matrix sizes do not match.";  
 return (\*this);  
 }  
  
 //matrix new\_mat(row,col);  
 Matrix2D<T> new\_mat;  
 new\_mat.alloc(rows, cols);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 new\_mat(i, j) = matrix[i][j] - m(i, j);  
 }  
 }  
 return new\_mat;  
 }  
  
 Matrix2D<T> operator\*(const Matrix2D<T>& m) const {  
 if (cols == m.rows) {  
  
 Matrix2D<T> new\_mat;  
 new\_mat.alloc(rows, m.cols);  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < m.cols; j++) {  
 new\_mat.matrix[i][j] = 0;  
 for (int k = 0; k < cols; k++) {  
 new\_mat.matrix[i][j] += matrix[i][k] \* m.matrix[k][j];  
 }  
 }  
 cout << endl;  
 }  
 cout << "Multipl:\n" << new\_mat;  
 return new\_mat;  
 }  
 else {  
 return m;  
 }  
  
 }  
  
 T \*\*alloc(int rows, int cols) {  
 this->rows = rows;  
 this->cols = cols;  
 next\_indexC = 0;  
 next\_indexR = 0;  
  
 matrix = new T \*[rows];  
 for (int row = 0; row < rows; row++) {  
 matrix[row] = new T[cols];  
 }  
 return matrix;  
  
 }  
  
 void dealloc() {  
 for (int row = 0; row < rows; row++)  
 delete[] matrix[row];  
  
 delete[] matrix;  
 }  
};  
  
  
  
/\*void outputInfo(int a, Matrix2D<T> m);  
void Triple(Matrix2D<T>& m);\*/  
  
template <typename T> void getMin(Matrix2D<T>& m) {  
 T min = T();  
  
 min = m(0, 0);  
 for (int i = 0; i < m.getRows(); i++) {  
 for (int j = 0; j < m.getCols(); j++) {  
 if (m(i, j) < min)  
 min = m(i, j);  
 }  
 }  
 cout << min;  
}  
  
template <typename T> void getMax(Matrix2D<T> &m)  
{  
 T max = T();  
 max = m(0, 0);  
 for (int i = 0; i < m.getRows(); i++) {  
 for (int j = 0; j < m.getCols(); j++) {  
 if (m(i, j) > max)  
 max = m(i, j);  
 }  
 }  
 cout << max;  
  
}  
  
template <typename T>  
void Triple(Matrix2D<T>& m) {  
 for (int i = 0; i < m.getRows(); i++) {  
 for (int j = 0; j < m.getCols(); j++) {  
  
 if (m(i, j) < 2) {  
 m(i, j) = m(i, j) \* m(i, j) \* m(i, j);  
 }  
 }  
 }  
}  
  
int main() {  
 cout << "Choose type of array: 1-int 2-double 3-fraction\n";  
 int answer;  
 cin >> answer;  
 if (answer == 1) {  
 Matrix2D<int> A1, A2;  
  
 A1.alloc(2, 2);  
 A2.alloc(2, 2);  
 try {  
 cout << "Enter A1 matrix: " << endl;  
 cin >> A1;  
 cout << "Enter A2 matrix: " << endl;  
 cin >> A2;  
 }  
 catch (Matrix2D<int>::OutOfBounds e) {  
 cout << "Bad row index: " << e.getOutOfBounIndexRow() << endl;  
 cout << "Bad col index: " << e.getOutOfBounIndexCol() << endl;  
  
 }  
  
 cout << "Entered matrices\n";  
 cout << A1 << endl;  
 cout << A2 << endl;  
  
 A1 + A2;  
 A1 \* A2;  
 //cout << "Sum\n" << A1 + A2 << endl;  
 //cout << "Multipl\n" << A1 \* A2 << endl;  
  
 cout << "Min element of the first matrix: ";  
 getMin(A1);  
  
 cout << "Max element of the first matrix: ";  
 getMax(A1);  
  
 cout << "\nMin elements tripled: \n";  
  
  
 Triple(A1);  
 cout << A1;  
 }  
 else if (answer == 2)  
 {  
 Matrix2D<double> B1, B2;  
 B1.alloc(2, 2);  
 B2.alloc(2, 2);  
 }  
 else if (answer == 3) {  
 Matrix2D<Fraction> F1, F2;  
 F1.alloc(2, 2);  
 F2.alloc(2, 2);  
  
  
 try {  
 cout << "Enter F1 matrix: " << endl;  
 cin >> F1;  
 cout << "Enter F2 matrix: " << endl;  
 cin >> F2;  
 }  
 catch (Matrix2D<int>::OutOfBounds e) {  
 cout << "Bad row index: " << e.getOutOfBounIndexRow() << endl;  
 cout << "Bad col index: " << e.getOutOfBounIndexCol() << endl;  
  
 }  
  
 cout << "Entered matrices\n";  
 cout << F1 << endl;  
 cout << F2 << endl;  
  
 //F1 + F2;  
 //F1 \* F2;  
 //cout << "Sum\n" << A1 + A2 << endl;  
 //cout << "Multipl\n" << A1 \* A2 << endl;  
  
  
 cout << "Min element of the first matrix: ";  
 getMin(F1);  
  
 cout << "\nMin element tripled: \n";  
 Triple(F1);  
 cout << F1;  
 }  
  
 system("pause");  
 return 0;  
}

Fraction.cpp

//  
// Created by alimsah on 5/16/19.  
//  
  
  
#include "Fraction.h"  
using std::cout;  
using std::cin;  
using std::endl;  
  
using std::ostream;  
using std::istream;  
  
Fraction::Fraction() {  
 numerator = 0;  
 denominator = 0;  
}  
  
Fraction::Fraction(int numerator, int denominator)  
{  
 this->numerator = numerator;  
 this->denominator = denominator;  
}  
  
int Fraction::GetNumerator()  
{  
 return numerator;  
}  
  
int Fraction::GetDenominator()  
{  
 return denominator;  
}  
  
void Fraction::SetNumerator(int value)  
{  
 numerator = value;  
}  
  
void Fraction::SetDenominator(int value)  
{  
 denominator = value;  
}  
  
ostream& operator<<(ostream& out, Fraction& fraction)  
{  
 // output the frACTION N/D  
 out << fraction.GetNumerator();  
 out << "/";  
 out << fraction.GetDenominator();  
 return out;  
}  
  
istream& operator>>(istream & in, Fraction & fraction)  
{  
 int numerator, denominator;  
 //out << "Please enter numerator and denominator: \n";  
 in >> numerator >> denominator;  
 fraction.SetNumerator(numerator);  
 fraction.SetDenominator(denominator);  
 return in;  
}  
  
Fraction Fraction::operator+(Fraction fraction)  
{  
 Fraction resultFraction;  
  
 if (this->denominator == fraction.GetDenominator()) {  
 resultFraction.SetNumerator(this->numerator + fraction.GetNumerator());  
 resultFraction.SetDenominator(this->denominator);  
 }  
 else {  
 resultFraction.SetNumerator((this->numerator \* fraction.GetDenominator()) +  
 (fraction.GetNumerator() \* this->denominator));  
 resultFraction.SetDenominator(this->denominator \* fraction.GetDenominator());  
 }  
  
 return resultFraction;  
 return Fraction();  
}  
  
Fraction Fraction::operator-(Fraction fraction) {  
 Fraction resultFraction;  
  
 if (this->denominator == fraction.GetDenominator()) {  
 resultFraction.SetNumerator(this->numerator - fraction.GetNumerator());  
 resultFraction.SetDenominator(this->denominator);  
 }  
 else {  
 resultFraction.SetNumerator((this->numerator \* fraction.GetDenominator()) - (fraction.GetNumerator() \* this->denominator));  
 resultFraction.SetDenominator(this->denominator \* fraction.GetDenominator());  
 }  
  
 return resultFraction;  
}  
  
Fraction Fraction::operator\*(Fraction fraction) {  
 Fraction resultFraction;  
  
 resultFraction.SetNumerator(this->numerator \* fraction.GetNumerator());  
 resultFraction.SetDenominator(this->denominator \* fraction.GetDenominator());  
  
 return resultFraction;  
}

Fraction header:

//  
// Created by alimsah on 5/16/19.  
//  
  
#ifndef TEMPLATE\_FRACTION\_H  
#define TEMPLATE\_FRACTION\_H  
  
  
#include <iostream>  
#include <string>  
  
using std::cout;  
using std::cin;  
using std::endl;  
  
using std::ostream;  
using std::istream;  
  
class Fraction {  
private:  
 long numerator;  
 long denominator;  
public:  
 Fraction();  
 Fraction(int numerator, int denominator);  
  
 friend ostream& operator<<(ostream& out, Fraction& fraction);  
 friend istream& operator>>(istream& in, Fraction& fraction);  
  
 Fraction operator+(Fraction fraction);  
 Fraction operator-(Fraction fraction);  
 Fraction operator\*(Fraction fraction);  
  
 friend bool operator<(Fraction& a, int b) {  
 return a.GetNumerator() < b;  
 }  
  
 bool operator<(Fraction a) {  
 return this->numerator \* a.GetDenominator() < a.GetNumerator() \* this->denominator;  
 }  
  
  
 int GetNumerator();  
 int GetDenominator();  
 void SetNumerator(int value);  
 void SetDenominator(int value);  
};  
  
  
#endif //TEMPLATE\_FRACTION\_H

Execution:

