Міністерство освіти і науки України

Національний Технічний Університет

«Харківський Політехнічний Інститут»

Кафедра «Стратегічного Керування»

Лабораторна робота № 2

«Використання поліморфізму та шаблонів у C++»

Перевірила: ас. кафедри СК

Вільхівская О. В.

Виконав:

Харків, 2018

Лабораторна робота № 2

Тема: Використання поліморфізму та шаблонів у C++

Завдання

1. Ієрархія класів

Реалізувати класи "Людина", "Громадянин", "Студент", "Співробітник". В кожному класі визначити віртуальну функцію виведення даних про об'єкт на екран. Створити масив указівників на різні об'єкти ієрархії. В циклі для кожного об'єкта вивести на екран рядок даних про нього.

1. Використання поліморфізму

Створити клас для розв'язання завдання 1.2 [шостої лабораторної роботи](http://iwanoff.inf.ua/algorithmization/LabTraining06.html) попереднього семестру.

Написати програму, яка реалізує перебір значень з метою пошуку деякого значення відповідно до індивідуального завдання, наведеного в таблиці. Необхідне значення може бути знайдено шляхом перевірки проміжних значень функції (або першої / другої похідної). Слід використати вказівник на функцію, для якого визначити **typedef**.

Сирцевий код повинен бути розділений на дві одиниці трансляції. Перша одиниця трансляції буде представлена заголовним файлом і файлом реалізації. Визначення **typedef**, а також прототип функції пошуку потрібного значення, повинні бути розташовані в заголовному файлі. Визначення цієї функції слід здійснити у файлі реалізації. Функція для перевірки працездатності програми, а також функція main(), повинні бути розташовані в іншій одиниці трансляції.

Клас повинен містити принаймні дві функції-члена - функцію, яка повертає значення відповідно до індивідуального завдання, а також суто віртуальну функцію, яка викликається з попередньої та визначає ліву частину рівняння або функцію для дослідження (відповідно до завдання).

Клас розташувати в окремому заголовному файлі. У відповідному файлі реалізації здійснити визначення однієї з двох функцій класу.

В іншій одиниці трансляції створити похідний клас із визначенням конкретної функції, яка підлягає дослідженню. У функції main() створити об'єкт похідного класу та здійснити виконання індивідуального завдання.

Правило пошуку (2 варіант): Мінімальне значення першої похідної

1. Узагальнений клас для представлення двовимірного масиву

Переробити клас, створений у завданні 1.3 попередньої лабораторної роботи, на шаблон класу. Реалізувати зовнішню узагальнену функцію знаходження мінімального елемента масиву. В функції main() створити масиви цілих, дійсних і простих дробів (раніше створений клас). Для цих трьох масивів здійснити перевірку роботи функції знаходження мінімального значення серед елементів масиву, здійснити тестування всіх можливостей класу з перехопленням можливих винятків, а також розв'язати індивідуальну задачу.

Примітка: для того, щоб можна було знаходити мінімальне значення у масиві дробів, у класі "Простий дріб" необхідно перевантажити операції порівняння.

Хід роботи

1. Ієрархія класів

|  |
| --- |
| #include <iostream>  #include <cstring>  using namespace std;  class Human {  private:  char name[30];  int age;  public:  Human(char\* name, int age) {  strcpy(this->name, name);  this->age=age;  }  virtual void info() {  cout << endl << "Name: " << name << " age: " << age;  }  virtual ~Human() {}  };  class Citizen : public Human {  private:  char location[40];  public:  Citizen(char\* name, int age, char\* location) : Human(name, age) {  strcpy(this->location, location);  }  virtual void info() {  Human::info();  cout << "\tLocation: " << location;  }  };  class Student : public Citizen {  public:  int year;  public:  Student(char\* name, int age, int year, char\* location) : Citizen(name,age,location), year(year){}  virtual void info() {  Citizen::info();  cout << "\tYear of study: " << year;  }  };  class Coworker : public Human {  private:  int total;  public:  Coworker(char\* name, int age, int total) : Human(name,age), total(total){}  virtual void info() {  Human::info();  cout << "\tWork togeher: " << total;  }  };  int main() {  const int N = 4;  Human\* people[N] = {  new Human("Mike", 21),  new Citizen("Michael", 40, "San Andreas"),  new Student("Nicola", 30, 2, "Nebraska"),  new Coworker("Steve", 25, 3)  };  for (int i = 0; i < N; i++) {  people[i]->info();  }  for (int i = 0; i < N; i++) {  delete people[i];  }  cout << endl;  system("pause");  } |

Приклад виконання програми:

|  |
| --- |
| Name: Mike age: 21  Name: Michael age: 40 Location: San Andreas  Name: Nicola age: 30 Location: Nebraska Year of study: 2  Name: Steve age: 25 Work togeher: 3  Press any key to continue . . . |

1. Використання поліморфізму

Вміст Derivative.cpp

|  |
| --- |
| #include "Derivative.h"  #include <iostream>  using std::cout;  using std::endl;  double Derivative::min\_der(double left, double right, double step, int n, double eps=0.0000001){  this->eps = eps;  this->n = n;  double y;  double temp;  bool mustFind = false;  double derivative;  for (double x = left; x <= right; x += step) {  y = function(x);  derivative = fst\_der(x);  if (!mustFind) {  temp = derivative;  mustFind = true;  }  if (derivative < temp) {  temp = derivative;  }  cout << "X = " << x << "\t Y = " << y << "\t\t Derivative = " << derivative << endl;  }  cout << endl;  return temp;  }  double Derivative::fst\_der(double x) {  return (function(x + eps) - function(x)) / eps;  }  class MyFunction : public Derivative{  public:  double function(double x) {  double y = 0;  if (x<0) {  for (int i = 1; i <= (n - 1); i++) {  for (int j = 1; j <= n; j++) y += x - i + j;  }  }  else  {  for (int i = 0; i <= (n - 1); i++) if (i) y += x / i;  }  return y;  }  };  class Parabola : public Derivative {  public:  double function(double x) {  return x\*x;  }  };  int main() {  MyFunction d;  cout << d.min\_der(-5, 5, 1, 5) << endl;  Parabola c;  cout << c.min\_der(-5, 5, 1, 5, 0.01) << endl;  system("pause");  } |

Вміст Derivative.h

|  |
| --- |
| #ifndef Derivative\_h  #define Derivative\_h  class Derivative {  public:  int n;  double eps;  double min\_der(double left, double right, double step, int n, double eps);  double fst\_der(double x);  virtual double function(double x)=0;  };  #endif |

Приклад виконання програми:

|  |
| --- |
| X = -5 Y = -90 Derivative = 20  X = -4 Y = -70 Derivative = 20  X = -3 Y = -50 Derivative = 20  X = -2 Y = -30 Derivative = 20  X = -1 Y = -10 Derivative = 20  X = 0 Y = 0 Derivative = 2.08333  X = 1 Y = 2.08333 Derivative = 2.08333  X = 2 Y = 4.16667 Derivative = 2.08333  X = 3 Y = 6.25 Derivative = 2.08333  X = 4 Y = 8.33333 Derivative = 2.08333  X = 5 Y = 10.4167 Derivative = 2.08333  2.08333  X = -5 Y = 25 Derivative = -9.99  X = -4 Y = 16 Derivative = -7.99  X = -3 Y = 9 Derivative = -5.99  X = -2 Y = 4 Derivative = -3.99  X = -1 Y = 1 Derivative = -1.99  X = 0 Y = 0 Derivative = 0.01  X = 1 Y = 1 Derivative = 2.01  X = 2 Y = 4 Derivative = 4.01  X = 3 Y = 9 Derivative = 6.01  X = 4 Y = 16 Derivative = 8.01  X = 5 Y = 25 Derivative = 10.01  -9.99  Press any key to continue . . . |

1. Узагальнений клас для представлення двовимірного масиву

Вміст main.cpp

|  |
| --- |
| #include "stdafx.h"  #include <iostream>  #include "Matrix.h"  #include "Fraction.h"  using namespace std;  void main() {  fraction t;  system("pause");  Matrix<fraction> f(3,4);  try {  f[0][0] = fraction(5, 8);  f[0][1] = fraction(10, 3);  f[0][2] = fraction(-4, 3);  f[1][0] = fraction(12, 19);  f[2][1] = fraction(5, 8);  f[16][1] = fraction(5, 8);  }  catch (Matrix<fraction>::OutOfBounds ex) {  cout << "Wrong index: " << "[" << ex.getIndex1() << "]" << "[" << ex.getIndex2() << "]" << endl;  }  cout << "Fraction Matrix: " << endl << f << endl << "Min: " << min(f.getArr(),f.getRow(),f.getCol()) << endl << endl;  Matrix<double> d(2, 4);  try {  d[0][0] = 4.1;  d[0][1] = 3.2;  d[1][1] = 17.2;  d[16][1] = 17.2;  }  catch (Matrix<double>::OutOfBounds ex) {  cout << "Wrong index: " << "[" << ex.getIndex1() << "]" << "[" << ex.getIndex2() << "]" << endl;  }  cout << "Double Matrix: " << endl << d << endl << "Min: " << min(d.getArr(), d.getRow(), d.getCol()) << endl << endl;  Matrix<int> i(3, 4);  try {  i[0][0] = 17;  i[0][1] = -5;  i[0][2] = 16;  i[1][0] = 5;  i[1][2] = 60;  i[-5][-2] = 60;  }  catch (Matrix<int>::OutOfBounds ex) {  cout << "Wrong index: " << "[" << ex.getIndex1() << "]" << "[" << ex.getIndex2() << "]" << endl;  }  cout << "Int Matrix: " << endl << i << endl << "Min: " << min(i.getArr(), i.getRow(), i.getCol()) << endl << endl;  system("pause");  } |

Вміст Fraction.h

|  |
| --- |
| #ifndef Fraction\_h  #define Fraction\_h  #include "stdafx.h"  #include <iostream>  #include <string>  using namespace std;  const char\* separator = "/" ;  class fraction {  static int GCD(int a, int b) {  a = abs(a);  b = abs(b);  while (a&&b) { (a > b) ? a %= b : b %= a; }  return a + b;  }  static int LCM(int a, int b) {  return a\*b / GCD(a, b);  }  friend istream& operator>> (istream& in, fraction& fr) {  return in >> fr.num >> fr.den;  }  friend ostream& operator<< (ostream& out, fraction& fr) {  return out << fr.num << "/" << fr.den;  }  friend fraction operator+(fraction& a, fraction& b) {  int lcm = LCM(a.den, b.den);  return fraction(a.num\*lcm / a.den + b.num\*lcm / b.den, lcm).reduction();  }  friend fraction operator-(fraction& a, fraction& b) {  int lcm = LCM(a.den, b.den);  return fraction(a.num\*lcm / a.den - b.num\*lcm / b.den, lcm).reduction();  }  friend fraction operator\*(fraction& a, fraction& b) {  return fraction(a.num\*b.num, a.den\*b.den).reduction();  }  friend fraction operator/(fraction& a, fraction& b) {  return fraction(a.num\*b.den, a.den\*b.num).reduction();  }  friend bool operator<(fraction& a, fraction &b) {  if ((a.den!=0)&&(b.den!=0)&&(a.num / a.den < b.num / b.den)) return true;  return false;  }  private:  int num, den;  public:  void operator=(int b) {  setNum(b);  setDen(b);  }  fraction() {};  fraction& reduction() {  int gcd = GCD(num, den);  num /= gcd;  den /= gcd;  return \*this;  }  fraction(int num, int den) : num(num), den(den) {}  void setNum(int value) { num = value; }  void setDen(int value) { den = value; }  int getNum() { return num; }  int getDen() { return den; }  };  #endif |

Вміст Matrix.h

|  |
| --- |
| #ifndef Array\_h  #define Array\_h  #include <iostream>  using namespace std;  template <typename T, typename T1> T\*\* arrCreation(T1 row, T1 col) {  T\*\* arr = new T\*[row];  for (int i = 0; i < row; i++) {  arr[i] = new T[col];  for (int j = 0; j < col; j++) {  arr[i][j] = 0;  }  }  return arr;  }  template <typename T>T min(T\*\* arr, int row, int col) {  T min = arr[0][0];  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  if (arr[i][j] < min) min = arr[i][j];  }  }  return min;  }  template <typename T> class Matrix {  friend ostream& operator<<(ostream& out, Matrix& a) {  for (int i = 0; i < a.row; i++) {  for (int j = 0; j < a.col; j++) out << a.arr[i][j] << "\t";  out << endl;  }  return out;  }  friend Matrix& operator>>(istream &in, Matrix& a) {  cout << "Enter parameters (rows, columns): ";  in >> a.row >> a.col;  a.arr = arrCreation<T>(a.row, a.col);  a.size = a.row\*a.col;  cout << "Enter " << a.size << " elements in a row" << endl;  int x;  for (int i = 0; i < a.size; i++) {  cin >> x;  a.addElem(x);  }  return a;  }  friend Matrix operator\*(Matrix& a, int x) {  for (int i = 0; i < a.row; i++) {  for (int j = 0; j < a.col; j++) {  a[i][j] \*= x;  }  }  return a;  }  friend Matrix operator\*(Matrix& a, Matrix& b) {  if (a.col != b.row) {  throw NotMatch(a, b);  }  Matrix c(a.row, b.col);  for (int i = 0; i < a.row; i++) {  for (int k = 0; k < b.col; k++) {  int sum = 0;  for (int j = 0; j < a.col; j++) {  sum += a[i][j] \* b[j][k];  }  c.addElem(sum);  }  }  return c;  }  friend Matrix operator+(Matrix& a, Matrix& b) {  if ((a.row == b.row) && (a.col == b.col)) {  Matrix c(a.row, a.col);  for (int i = 0; i < a.row; i++) {  for (int j = 0; j < a.col; j++) {  c[i][j] = a[i][j] + b[i][j];  }  }  return c;  }  throw NotMatch(a, b);  }  friend Matrix operator-(Matrix& a, Matrix& b) {  if ((a.row == b.row) && (a.col == b.col)) {  Matrix c(a.row, a.col);  for (int i = 0; i < a.row; i++) {  for (int j = 0; j < a.col; j++) {  c[i][j] = a[i][j] - b[i][j];  }  }  return c;  }  throw NotMatch(a, b);  }  private:  int row, col;  T\*\* arr;  int size;  int count = 0;  void addElem(int value){  arr[count / col][count%col] = value;  count++;  }  public:  T\*\* getArr() { return arr; }  int getRow() { return row; }  int getCol() { return col; }  int getSize() { return size; }  class NotMatch { // Класс исключение  int row1, row2, col1, col2;  public:  NotMatch(Matrix& a, Matrix& b) : row1(a.row), row2(b.row), col1(a.col), col2(b.col) {}    int getRow1() { return row1; }  int getRow2() { return row2; }  int getCol1() { return col1; }  int getCol2() { return col2; }  };  class OutOfBounds {  int index1;  int index2;  public:  OutOfBounds(int index1, int index2) : index1(index1), index2(index2){}  int getIndex1() { return index1; }  int getIndex2() { return index2; }  };  Matrix() { arr = 0; size = 0; }  Matrix(int row, int col) : row(row), col(col), arr(arrCreation<T>(row, col)), size(row\*col){}  class Proxy {  private:  int index1;  T\* \_arr;  int col;  int row;  public:  Proxy(int index1, T\* \_arr, int row, int col) : index1(index1), \_arr(\_arr), row(row), col(col) {}  T& operator[](int index2) {  if ((index2 < 0) || (index2 >= col) || (index1<0) || (index1>=row)) {  throw Matrix<T>::OutOfBounds(index1, index2);  }  return \_arr[index2];  }  };  Proxy operator[](int index1) {  return Proxy(index1, arr[index1], row, col);  }  ~Matrix() {  }  };  #endif |

Приклад виконання програми:

|  |
| --- |
| Для продолжения нажмите любую клавишу . . .  Wrong index: [16][1]  Fraction Matrix:  5/8 10/3 -4/3 0/0  12/19 0/0 0/0 0/0  0/0 5/8 0/0 0/0  Min: -4/3  Wrong index: [16][1]  Double Matrix:  4.1 3.2 0 0  0 17.2 0 0  Min: 0  Wrong index: [-5][-2]  Int Matrix:  17 -5 16 0  5 0 60 0  0 0 0 0  Min: -5  Для продолжения нажмите любую клавишу . . . |

Висновок

Ієрархія класів – класифікація об’єктних типів, що застосовується для зв’язку класів за допомогою реалізації *успадкування* – створення похідного класу від базового, *абстрагування* – створення абстрактного класу за допомогою включення суто-віртуальних функцій. Абстрактні класи реалізують один із принципів ООП – поліморфізм, а точніше поліморфізм часу виконання.

Шаблони реалізують інший різновид поліморфізму – поліморфізм часу компіляції. Шаблонні функції и шаблони типів можуть працювати з різними типами даних.