Міністерство освіти і науки України

Національний Технічний Університет

«Харківський Політехнічний Інститут»

Кафедра «Стратегічного Керування»

Лабораторна робота № 1

«Створення та використання класів C++»

Перевірила: ас. кафедри СК

Вільхівская О. В.

Виконав:

Харків, 2018

Лабораторна робота №

Тема:

Завдання

1. Клас для представлення простого дробу

Створити клас для представлення простого дробу. Реалізувати конструктори, функцію скорочення дробу, а також перевантажити операції +, -, \*, /, введення та виведення. Здійснити демонстрацію можливостей класу в функції main().

1. Клас для представлення двовимірного масиву

Розробити клас для представлення двовимірного масиву (матриці) цілих чисел довільних розмірів. Створити конструктори та деструктор, перевантажити операції додавання, віднімання і множення (згідно з правилами роботи з матрицями), звертання за індексом, введення з потоку та виведення в потік. Створити власні класи винятків та генерувати відповідні об'єкти-винятки, якщо неможливо виконати ту чи іншу операцію.

Створити окрему функцію, яка отримує посилання на матрицю і виконує над масивом дії, вказані в таблиці. Функція не повинна бути методом класу або дружньою функцією.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер варіанту  (номер студента у списку)** | | **Правило перетворення елементів масиву** | **Кількість рядків m** | **Кількість стовпців n** |
| 2 |  | Усі елементи з парними значеннями повинні бути замінені їх квадратами | 3 | 5 |

1. Підрахунок суми введених значень

Створити клас з одним закритим елементом даних цілого типу, геттером і конструктором з одним параметром. В цьому ж класі створити закрите статичне поле, яке зберігає суму цілих елементів даних всіх раніше створених об'єктів. Під час кожного виклику конструктора до статичного поля повинно додаватися нове значення. Статична публічна функція цього ж класу повинна повертати цю суму.

У функції main() створити декілька об'єктів і вивести отриману суму.

Хід роботи

1. Клас для представлення простого дробу

|  |
| --- |
| #include <iostream>  #include <cmath>  using namespace std;  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);  }  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);  }  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();  }  private:  int num, den;  public:  fraction(int num, int den) {  this->num = num;  this->den = den;  }  fraction& reduction() {  int gcd = GCD(num, den);  num /= gcd;  den /= gcd;  return \*this;  }  void setNum(int value) {num = value;}  void setDen(int value) {den = value;}  int getNum() { return num; }  int getDen() { return den; }  };  int main() {  fraction a(-10, 2);  fraction b(4, 2);  fraction c = a + b;  fraction d = c - a;  fraction e = c\*d;  cout << "a: " << a << endl;  cout << "a reducted: " << a.reduction() << endl;  cout << "b: " << b << endl;  cout << "c = a+b: " << c << endl;  cout << "d = c-a: " << d << endl;  cout << "e = c\*d: " << e << endl;  fraction f(0, 0);  cout << endl << "Enter num, den space-separated: ";  cin >> f;  fraction g = e / f;  //g.reduction();  cout << "f: " << f << endl;  cout << "e/f = " << e << " : " << f << " = " << g << endl;  system("pause");  } |

Приклад виконання програми:

|  |
| --- |
| a: -10/2  a reducted: -5/1  b: 4/2  c = a+b: -6/2  d = c-a: 4/2  e = c\*d: -6/1  Enter num, den space-separated: 3 9  f: 3/9  e/f = -6/1 : 3/9 = -18/1  Press any key to continue . . . |

1. Клас для представлення двовимірного масиву

|  |
| --- |
| #include <iostream>  using namespace std;  int\*\* arrCreation(int row, int col);  class Matrix {  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);  }  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 = new int\*[a.row];  for (int i = 0; i < a.row; i++) {  a.arr[i] = new int[a.col];  for (int j = 0; j < a.col; j++) {  a.arr[i][j] = 0;  }  }  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;  }  private:  int row, col;  int\*\* arr;  int size;  int count = 0;  void addElem(int value) {  arr[count / col][count%col] = value;  count++;  }  public:  int\*\* getArr() { return arr; }  int getRow() { return row; }  int getCol() { return col; }  int getSize() { return size; }  class NotMatch {  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; }  };  Matrix() { arr = 0; size = 0; }  Matrix(int row, int col) {  this->arr=arrCreation(row, col);  this->row = row;  this->col = col;  size = row\*col;  }  class Proxy {  private:  int\* \_arr;  public:  Proxy(int\* \_arr) {  this->\_arr = \_arr;  }  int& operator[](int index) {  return \_arr[index];  }  ~Proxy() {}  };  Proxy operator[](int index) {  return Proxy(arr[index]);  }  ~Matrix() {  }  };  void roots(Matrix& a);  int\*\* arrCreation(int row, int col) {  int\*\* arr = new int\*[row];  for (int i = 0; i < row; i++) {  arr[i] = new int[col];  for (int j = 0; j < col; j++) {  arr[i][j] = 0;  }  }  return arr;  }  void roots(Matrix& a) {  for (int i = 0; i < a.getRow(); i++) {  for (int j = 0; j < a.getCol(); j++) {  if ((i + j) % 2 == 0) a.getArr()[i][j] \*= a.getArr()[i][j];  }  }  }  int main() {  Matrix a(3, 2);  a[2][0] = 5;  cout << "Matrix a: " << endl << a << endl;  roots(a);  cout << "Matrix a (changed): " << endl << a << endl;  Matrix b;  cin >> b;  cout << "Matrix b: " << endl << b;  try  {  Matrix c = a + b;  cout << "Matrix c = a+b: " << endl << c << endl;  }  catch (Matrix::NotMatch ex)  {  cout << "Not Match " << ex.getRow1() << "x" << ex.getCol1() << " != " << ex.getRow2() << "x" << ex.getCol2() << endl;  }  Matrix e(2, 4);  e[0][0] = 1;  e[0][1] = 2;  e[0][2] = 3;  e[0][3] = 7;  cout << "Matrix e: " << endl << e << endl;  Matrix f(3, 3);  f[0][0] = 1;  f[1][0] = 2;  f[0][1] = 5;  f[2][1] = 6;  f[2][0] = 3;  cout << "Matrix f: " << endl << f << endl;  system("pause");  try  {  Matrix d = e \* f;  cout << "Matrix d: " << endl << d << endl;  }  catch (Matrix::NotMatch ex)  {  cout << "Not Match " << ex.getCol1() << "!=" << ex.getRow2() << endl;  }  system("pause");  } |

Приклад виконання програми:

|  |
| --- |
| Matrix a:  0 0  0 0  5 0  Matrix a (changed):  0 0  0 0  25 0  Enter parameters (rows, columns): 2 2  Enter 4 elements in a row  1 2 3 4  Matrix b:  1 2  3 4  Not Match 3x2 != 2x2  Matrix e:  1 2 3 7  0 0 0 0  Matrix f:  1 5 0  2 0 0  3 6 0  Press any key to continue . . .  Not Match 4!=3  Press any key to continue . . . |

1. Підрахунок суми введених значень

|  |
| --- |
| #include <iostream>  using namespace std;  class counter {  private:  static int sum;  int a;  public:  counter(int a) : a(a){ sum += a; }  int getSum() { return sum; }  ~counter() { sum -= a;}  };  int counter::sum = 0;  int main() {  counter a(15);  counter \*p = new counter(4);  cout << a.getSum() << endl; // 19  delete p;  cout << a.getSum() << endl; // 15  system("pause");  } |

Приклад виконання програми:

|  |
| --- |
| 19  15  Press any key to continue . . . |

**Висновок**

Виконуючи цю лабораторну роботу я познайомився з класами – структурованими типами даних, навчився використовувати модифікатори доступа. Також довелося створити клас виняток, до якого ми звертаємося у разі виникнення певних помилок. Корисною функцією є перевантаження операторів, що дозволяє більш повне налаштування свого типу (класу).