**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Основы метапрограммирования

Студент: Ефимов Александр

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата: 04.11.2019

Оценка:

Москва, 2019

1. Постановка задачи

Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.

Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат фигур необходимо использовать шаблон std::pair.

Создать набор шаблонов, создающих функции, реализующие:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Параметром шаблона должен являться тип класса фигуры ( например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple. Например, std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> должен интерпретироваться как треугольник. std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> - как квадрат. Каждый std::pair<int,int> - соответствует координатам вершины фигуры вращения.

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания (как в виде класса, так и в виде std::tuple).

• Вызывать для нее шаблонные функции (1-3).

При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable\_if, std::tuple\_size, std::is\_same.

Вариант 1: Треугольник, Квадрат, Прямоугольник.

1. Описание программы

Библиотека шаблонов написана в *Shape.h*, вместе шаблонными функциями *Area*, *Centre* и *Print*, которые ищут площадь, центр фигуры и печатают её соответственно.

В *main.cpp* содержатся *Triangle, Rectangle* и *Square* типа **double**, которые принимают ввод из терминала. Если фигура после ввода не соответствует классу фигуры, в которую она введена (например, ввод прямой в класс треугольника), то она обозначается как Abstract (абстрактная), но все еще высчитывает центр и площадь и выводит точки.

Если

1. Набор testcases

В файлах *test\_01.txt*, *test\_02.txt* рассмотрены примеры правильного и неправильного ввода соответственно.

В файлах *test\_03*, *test\_04* содержатся переменные шаблонов типа **Tuple**, которые используют значения прошлых двух тестов

1. Результаты выполнения тестов

Все тесты возвращают ожидаемые результаты

**test\_01.txt**

|  |
| --- |
| Creating triangle...  Input point coordinates  A: 4.25 9.25  B: -1.75 4.25  C: -0.75 -1.75  Creating rectangle...  Input point coordinates clockwise or counter clockwise  A: -0.5 -2.5  B: 4.5 2.5  C: -7.5 14.5  D: -12.5 9.5  Creating square...  Input point coordinates clockwise or counter clockwise  A: -5.5 10.5  B: -4.5 6.5  C: -8.5 5.5  D: -9.5 9.5  Triangle: (4.25 , 9.25); (-1.75 , 4.25); (-0.75 , -1.75); Centre = (0.583333 , 3.91667); Area = 20.5  Rectangle: (-0.5 , -2.5); (4.5 , 2.5); (-7.5 , 14.5); (-12.5 , 9.5); Centre = (-4 , 6); Area = 120  Square: (-5.5 , 10.5); (-4.5 , 6.5); (-8.5 , 5.5); (-9.5 , 9.5); Centre = (-7 , 8); Area = 17 |

**test\_02.txt**

|  |
| --- |
| Creating triangle...  Input point coordinates  A: 0 1  B: 0 2  C: 0 12  Creating rectangle...  Input point coordinates clockwise or counter clockwise  A: 0 0  B: 0 0  C: 0 0  D: 0 0  Creating square...  Input point coordinates clockwise or counter clockwise  A: 0 -2  B: 5 3  C: -7 15  D: -12 10  Abstract: (0 , 1); (0 , 2); (0 , 12); Centre = (0 , 5); Area = 0  Abstract: (0 , 0); (0 , 0); (0 , 0); (0 , 0); Centre = (0 , 0); Area = 0  Abstract: (0 , -2); (5 , 3); (-7 , 15); (-12 , 10); Centre = (-3.5 , 6.5); Area = 120 |

**test\_03**

|  |
| --- |
| Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  Rectangle: (0 , -2); (5 , 3); (-7 , 15); (-12 , 10); Centre = (-3.5 , 6.5); Area = 120  Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  Triangle: (4.25 , 9.25); (-1.75 , 4.25); (-0.75 , -1.75); Centre = (0.583333 , 3.91667); Area = 20.5  Rectangle: (-0.5 , -2.5); (4.5 , 2.5); (-7.5 , 14.5); (-12.5 , 9.5); Centre = (-4 , 6); Area = 120  Square: (-5.5 , 10.5); (-4.5 , 6.5); (-8.5 , 5.5); (-9.5 , 9.5); Centre = (-7 , 8); Area = 17 |

**test\_04**

|  |
| --- |
| Abstract: (0 , 0); (0 , 1); (0 , 2); Centre = (0 , 1); Area = 0  Abstract: (0 , 0); (0 , 0); (0 , 0); (0 , 0); Centre = (0 , 0); Area = 0 |

1. Листинг программы

**Shape.h**

|  |
| --- |
| #ifndef SHAPE\_TEMPLATE\_NEW\_H  #define SHAPE\_TEMPLATE\_NEW\_H  #include <iostream>  #include <utility>  #include <tuple>  #include <cmath>  typedef std::pair<double, double> DoublePoint;  template <typename T>  class Triangle {  public:  typedef std::pair<T, T> Point;  Triangle(std::istream&);  bool abstract = false;  std::pair<double, double> centre;  double area{ 0 };  Point p[3];  };  template <typename T>  class Rectangle {  public:  typedef std::pair<T, T> Point;  Rectangle(std::istream&);  bool abstract = false;  std::pair<double, double> centre;  double area{ 0 };  Point p[4];  };  template <typename T>  class Square {  public:  typedef std::pair<T, T> Point;  Square(std::istream&);  bool abstract = false;  std::pair<double, double> centre;  double area{ 0 };  Point p[4];  };  //-----------------------------------------------------------------------------------------------------------  bool DoubleEqual(double lhs, double rhs)  {  const double EPS = 0.00001;  if (lhs > rhs)  return (lhs - rhs) < EPS ? true : false;  else  return (rhs - lhs) < EPS ? true : false;  }  bool IsRight(std::pair<double, double> a, std::pair<double, double> b, std::pair<double, double> c)  {  std::pair<double, double> vec1 = { b.first - a.first, b.second - a.second };  std::pair<double, double> vec2 = { c.first - a.first, c.second - a.second };  double result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (DoubleEqual(result, 0)) return true;  return false;  }  bool IsRight(std::pair<int, int> a, std::pair<int, int> b, std::pair<int, int> c)  {  std::pair<int, int> vec1 = { b.first - a.first, b.second - a.second };  std::pair<int, int> vec2 = { c.first - a.first, c.second - a.second };  int result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (result == 0) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  template <class T>  bool IsTriangle(std::pair<T, T> p[3])  {  return !DoubleEqual(((p[1].first - p[0].first) \* (p[2].second - p[0].second) - (p[1].second - p[0].second) \* (p[2].first - p[0].first)), 0);  }  //-----------------------------------------------------------------------------------------------------------  template <class T>  bool IsRectangle(std::pair<T, T> p[4])  {  std::pair<T, T> null(0, 0);  if (p[0] == null && p[1] == null && p[2] == null && p[3] == null) return false;  if (  IsRight(p[2], p[1], p[3]) &&  IsRight(p[3], p[2], p[0]) &&  IsRight(p[1], p[0], p[2]) &&  IsRight(p[0], p[3], p[1])  ) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  template <class T>  inline double distance(std::pair<T, T> a, std::pair<T, T> b)  {  return (b.first - a.first) \* (b.first - a.first) + (b.second - a.second) \* (b.second - a.second);  }  template <class T>  bool IsSquare(std::pair<T, T> p[4])  {  if (  IsRectangle(p) &&  DoubleEqual(distance(p[0], p[1]), distance(p[1], p[2])) &&  DoubleEqual(distance(p[1], p[2]), distance(p[2], p[3])) &&  DoubleEqual(distance(p[2], p[3]), distance(p[3], p[0])) &&  DoubleEqual(distance(p[3], p[0]), distance(p[0], p[1]))  ) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  template <class T>  DoublePoint Centre(Triangle<T>& tri)  {  DoublePoint res(0, 0);  for (int i = 0; i < 3; ++i)  {  res.first += tri.p[i].first;  res.second += tri.p[i].second;  }  res.first /= 3;  res.second /= 3;  return res;  }  template <class T>  DoublePoint Centre(Rectangle<T>& rec)  {  DoublePoint res(0, 0);  for (int i = 0; i < 4; ++i)  {  res.first += rec.p[i].first;  res.second += rec.p[i].second;  }  res.first /= 4;  res.second /= 4;  return res;  }  template <class T>  DoublePoint Centre(Square<T>& sqr)  {  DoublePoint res(0, 0);  for (int i = 0; i < 4; ++i)  {  res.first += sqr.p[i].first;  res.second += sqr.p[i].second;  }  res.first /= 4;  res.second /= 4;  return res;  }  template <class T>  DoublePoint Centre(std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& tri)  {  DoublePoint res(0, 0);  res.first += std::get<0>(tri).first;  res.second += std::get<0>(tri).second;  res.first += std::get<1>(tri).first;  res.second += std::get<1>(tri).second;  res.first += std::get<2>(tri).first;  res.second += std::get<2>(tri).second;  res.first /= 3;  res.second /= 3;  return res;  }  template <class T>  DoublePoint Centre(std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& rec)  {  DoublePoint res(0, 0);  res.first += std::get<0>(rec).first;  res.second += std::get<0>(rec).second;  res.first += std::get<1>(rec).first;  res.second += std::get<1>(rec).second;  res.first += std::get<2>(rec).first;  res.second += std::get<2>(rec).second;  res.first += std::get<3>(rec).first;  res.second += std::get<3>(rec).second;  res.first /= 4;  res.second /= 4;  return res;  }  template <class T>  double Area(Triangle<T>& tri)  {  double area = (double)(tri.p[0].first \* (tri.p[1].second - tri.p[2].second) + tri.p[1].first \* (tri.p[2].second - tri.p[0].second) + tri.p[2].first \* (tri.p[0].second - tri.p[1].second)) / 2;  return area < 0 ? -area : area;  }  template <class T>  double Area(Rectangle<T>& rec)  {  return sqrt(distance(rec.p[0], rec.p[1]) \* distance(rec.p[1], rec.p[2]));  }  template <class T>  double Area(Square<T>& sqr)  {  return sqrt(distance(sqr.p[0], sqr.p[1]) \* distance(sqr.p[1], sqr.p[2]));  }  template <class T>  double Area(std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& tri)  {  double area = (double)(get<0>(tri).first \* (get<1>(tri).second - get<2>(tri).second) + get<1>(tri).first \* (get<2>(tri).second - get<0>(tri).second) + get<2>(tri).first \* (get<0>(tri).second - get<1>(tri).second)) / 2;  return area < 0 ? -area : area;  }  template <class T>  double Area(std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& rec)  {  return sqrt(distance(get<0>(rec), get<1>(rec)) \* distance(get<1>(rec), get<2>(rec)));  }  template <class T>  std::ostream& operator << (std::ostream& os, const std::pair<T, T>& p)  {  os << '(' << p.first << " , " << p.second << ')';  return os;  }  template <class T>  void print(std::ostream& os, Triangle<T>& tri)  {  if (tri.abstract)  os << "Abstract: ";  else  os << "Triangle: ";  os << tri.p[0] << "; " << tri.p[1] << "; " << tri.p[2] << "; Centre = " << tri.centre << "; Area = " << tri.area << std::endl;  }  template <class T>  void print(std::ostream& os, Rectangle<T>& rec)  {  if (rec.abstract)  os << "Abstract: ";  else  os << "Rectangle: ";  os << rec.p[0] << "; " << rec.p[1] << "; " << rec.p[2] << "; " << rec.p[3] << "; Centre = " << rec.centre << "; Area = " << rec.area << std::endl;  }  template <class T>  void print(std::ostream& os, Square<T>& sqr)  {  if (sqr.abstract)  os << "Abstract: ";  else  os << "Square: ";  os << sqr.p[0] << "; " << sqr.p[1] << "; " << sqr.p[2] << "; " << sqr.p[3] << "; Centre = " << sqr.centre << "; Area = " << sqr.area << std::endl;  }  template <class T>  void print(std::ostream& os, std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& tri)  {  bool abstract;  std::pair<T, T> p[4];  p[0] = std::get<0>(tri);  p[1] = std::get<1>(tri);  p[2] = std::get<2>(tri);  if (IsTriangle(p)) abstract = false;  else abstract = true;  DoublePoint centre = Centre(tri);  double area = Area(tri);  if (abstract)  os << "Abstract: ";  else  os << "Triangle: ";  os << p[0] << "; " << p[1] << "; " << p[2] << "; Centre = " << centre << "; Area = " << area << std::endl;  }  template <class T>  void print(std::ostream& os, std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> >& rec)  {  int switcher = 0;  std::pair<T, T> p[4];  p[0] = std::get<0>(rec);  p[1] = std::get<1>(rec);  p[2] = std::get<2>(rec);  p[3] = std::get<3>(rec);  if (IsSquare(p)) switcher = 2;  else if (IsRectangle(p)) switcher = 1;  else switcher = 0;  DoublePoint centre = Centre(rec);  double area = Area(rec);  switch (switcher)  {  case 2:  os << "Square: ";  break;  case 1:  os << "Rectangle: ";  break;  case 0:  os << "Abstract: ";  break;  }  os << p[0] << "; " << p[1] << "; " << p[2] << "; " << p[3] << "; Centre = " << centre << "; Area = " << area << std::endl;  }  //-----------------------------------------------------------------------------------------------------------  bool operator == (std::pair<int, int> lhs, std::pair<int, int> rhs)  {  return lhs.first == rhs.first && lhs.second == rhs.second;  }  bool operator == (std::pair<double, double> lhs, std::pair<double, double> rhs)  {  return DoubleEqual(lhs.first, rhs.first) && DoubleEqual(lhs.second, rhs.second);  }  template <class T>  Triangle<T>::Triangle(std::istream& is)  {  std::cout << "Input point coordinates\nA: ";  is >> p[0].first >> p[0].second;  std::cout << "B: ";  is >> p[1].first >> p[1].second;  std::cout << "C: ";  is >> p[2].first >> p[2].second;  if (IsTriangle(p)) abstract = false;  else abstract = true;  centre = Centre(\*this);  area = Area(\*this);  }  template <class T>  Rectangle<T>::Rectangle(std::istream& is)  {  std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";  is >> p[0].first >> p[0].second;  std::cout << "B: ";  is >> p[1].first >> p[1].second;  std::cout << "C: ";  is >> p[2].first >> p[2].second;  std::cout << "D: ";  is >> p[3].first >> p[3].second;  if (IsRectangle(p)) abstract = false;  else abstract = true;  centre = Centre(\*this);  area = Area(\*this);  }  template <class T>  Square<T>::Square(std::istream& is)  {  std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";  is >> p[0].first >> p[0].second;  std::cout << "B: ";  is >> p[1].first >> p[1].second;  std::cout << "C: ";  is >> p[2].first >> p[2].second;  std::cout << "D: ";  is >> p[3].first >> p[3].second;  if (IsSquare(p)) abstract = false;  else abstract = true;  centre = Centre(\*this);  area = Area(\*this);  }  #endif |

**main.cpp**

|  |
| --- |
| /\*  Разработать шаблоны классов согласно варианту задания.  Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.  Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат фигур необходимо использовать шаблон std::pair.  Создать набор шаблонов, создающих функции, реализующие:  1. Вычисление геометрического центра фигуры;  2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;  3. Вычисление площади фигуры;  Параметром шаблона должен являться тип класса фигуры ( например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple. Например, std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> должен интерпретироваться как треугольник. std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> - как квадрат. Каждый std::pair<int,int> - соответствует координатам вершины фигуры вращения.  Создать программу, которая позволяет:  • Вводить из стандартного ввода std::cin фигуры, согласно варианту задания (как в виде класса, так и в виде std::tuple).  • Вызывать для нее шаблонные функции (1-3).  При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable\_if, std::tuple\_size, std::is\_same.  Вариант 1: Треугольник, Квадрат, Прямоугольник.  Ефимов Александр Владимирович  M8O-201Б  \*/  #include "Shape.h"  using namespace std;  int main()  {  cout << "Creating triangle..." << endl;  Triangle<double> tri(cin);  cout << "Creating rectangle..." << endl;  Rectangle<double> rec(cin);  cout << "Creating square..." << endl;  Square<double> sqr(cin);  print(cout, tri);  print(cout, rec);  print(cout, sqr);  } |

**test\_03.cpp**

|  |
| --- |
| #include "Shape.h"  using namespace std;  int main()  {  typedef pair<int, int> IntPair;  typedef pair<double, double> DoublePair;  IntPair a, b, c, d;  a = make\_pair(4, 9);  b = make\_pair(-2, 4);  c = make\_pair(-1, -2);  tuple<IntPair, IntPair, IntPair> TriArgs (a,b,c);  print(cout, TriArgs);  a = make\_pair(0, -2);  b = make\_pair(5, 3);  c = make\_pair(-7, 15);  d = make\_pair(-12, 10);  tuple<IntPair, IntPair, IntPair, IntPair> RecArgs (a,b,c,d);  print(cout, RecArgs);  a = make\_pair(-6, 10);  b = make\_pair(-5, 6);  c = make\_pair(-9, 5);  d = make\_pair(-10, 9);  tuple<IntPair, IntPair, IntPair, IntPair> SqrArgs(a,b,c,d);  print(cout, SqrArgs);  DoublePair e, f, g, h;    e = make\_pair(4.25, 9.25);  f = make\_pair(-1.75, 4.25);  g = make\_pair(-0.75, -1.75);    tuple<DoublePair, DoublePair, DoublePair> TriArgsDouble(e, f, g);  print(cout, TriArgsDouble);  e = make\_pair(-0.5, -2.5);  f = make\_pair(4.5, 2.5);  g = make\_pair(-7.5, 14.5);  h = make\_pair(-12.5, 9.5);  tuple<DoublePair, DoublePair, DoublePair, DoublePair> RecArgsDouble(e, f, g, h);  print(cout, RecArgsDouble);  e = make\_pair(-5.5, 10.5);  f = make\_pair(-4.5, 6.5);  g = make\_pair(-8.5, 5.5);  h = make\_pair(-9.5, 9.5);  tuple<DoublePair, DoublePair, DoublePair, DoublePair> SqrArgsDouble(e, f, g, h);  print(cout, SqrArgsDouble);  } |

**test\_04.cpp**

|  |
| --- |
| #include "Shape.h"  using namespace std;  int main()  {  typedef pair<int, int> IntPair;  typedef pair<double, double> DoublePair;  IntPair a, b, c, d;  a = make\_pair(0, 0);  b = make\_pair(0, 1);  c = make\_pair(0, 2);  tuple<IntPair, IntPair, IntPair> TriArgs (a,b,c);  print(cout, TriArgs);  a = make\_pair(0, 0);  b = make\_pair(0, 0);  c = make\_pair(0, 0);  d = make\_pair(0, 0);  tuple<IntPair, IntPair, IntPair, IntPair> RecArgs (a,b,c,d);  print(cout, RecArgs);  } |

1. Выводы

Template Programming является громоздким семантически громоздким и трудным в использовании. Но взамен он позволяет найти ошибки на этапе компиляции. Несмотря на это, как было подмечено разными источниками, его следует использовать лишь в крайних случаях, особенно с появлением **constexpr**.

1. Список литературы

* Статьи по компьютерным наукам [Электронный ресурс]. URL: https://www.geeksforgeeks.org/template-metaprogramming-in-c/ (дата обращения: 19.09.2019).