**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: Баранников Степан Алексеевич

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата: 15.11.2019

Оценка:

Москва, 2019

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 задания

|  |  |  |  |
| --- | --- | --- | --- |
| 18. | Квадрат | Прямоугольник | Трапеция |

1. Описание программы

Программа позволяет:

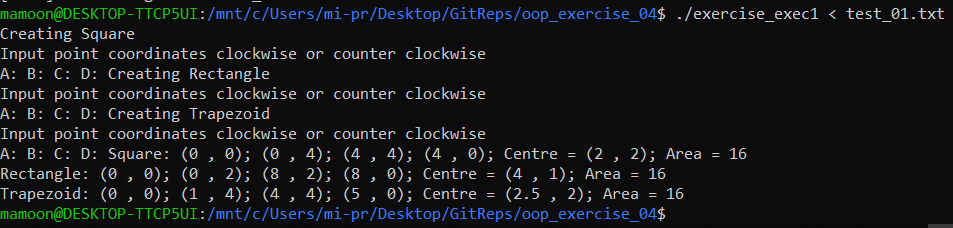
• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

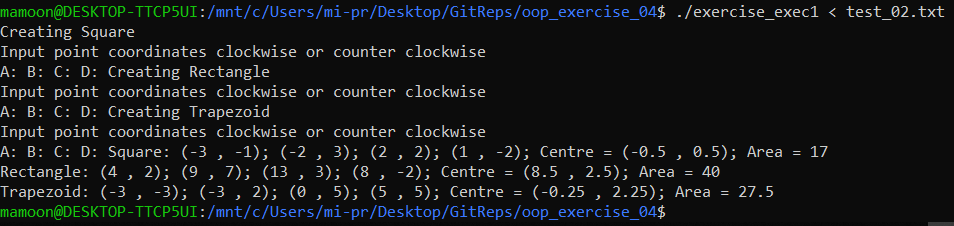
• Вводить кортежи std::tuple

1. Набор testcases  
    Запуск вместе с **exercise\_exec1**test\_01.txt  
   0 0 0 4 4 4 4 0  
   0 0 0 2 8 2 8 0  
   0 0 1 4 4 4 5 0  
   test\_02.txt  
   -3 -1 -2 3 2 2 1 -2  
   4 2 9 7 13 3 8 -2  
   -3 -3 -3 2 0 5 5 5

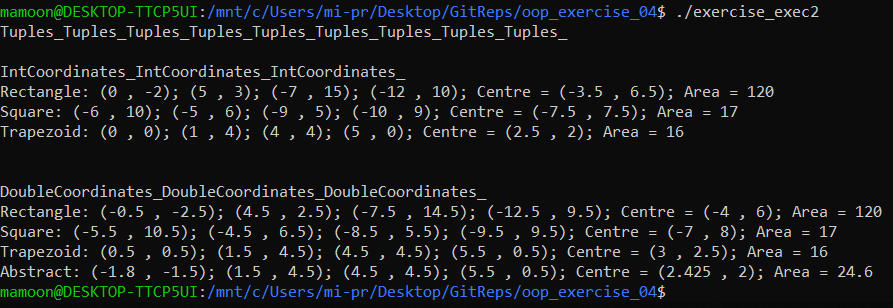
Запуск **exercise\_exec2**

1. Результаты выполнения тестов.

./exercise\_exec1 < test\_01.txt  


./exercise\_exec1 < test\_02.txt  


./exercise\_exec2



1. Листинг программы

Shapes.h

|  |
| --- |
| #include <iostream> using namespace std; #include <tuple> #include <vector> #include <cmath>  typedef pair<double, double> DoublePoint;  template <typename T> class Figure { public:  typedef pair<T, T> Point;   Figure(istream&);  bool Square = false, Rectangle = false, Trapezoid = false;  bool Abstract = false;   DoublePoint 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(pair<double, double> a, pair<double, double> b, pair<double, double> c){  pair<double, double> vec1 = { b.first - a.first, b.second - a.second };  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(pair<int, int> a, pair<int, int> b, pair<int, int> c){  pair<int, int> vec1 = { b.first - a.first, b.second - a.second };  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> inline double distance(pair<T, T> a, 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 IsRectangle(pair<T, T> p[4]){  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> bool IsSquare(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> bool IsTrapezoid(pair<T, T> p[4]){  pair<T, T> null(0, 0);  if (p[0] == null && p[1] == null && p[2] == null && p[3] == null) return false;   if (  !DoubleEqual(distance(p[1], p[2]), distance(p[3], p[0])) &&  DoubleEqual(distance(p[0], p[1]), distance(p[2], p[3])) &&  DoubleEqual(distance(p[0], p[2]), distance(p[1], p[3]))  ) return true;  return false; } //-----------------------------------------------------------------------------------------------------------  //----------------------------------------------- template <class T> DoublePoint Centre(Figure<T>& fig){  if(fig.Square || fig.Rectangle || fig.Trapezoid){  DoublePoint res(0, 0);  for (int i = 0; i < 4; ++i)  {  res.first += fig.p[i].first;  res.second += fig.p[i].second;  }  res.first /= 4;  res.second /= 4;  return res;  } }  template <class T> DoublePoint Centre(tuple< pair<T, T>, pair<T, T>, pair<T, T>, pair<T, T> >& fig) {  DoublePoint res(0, 0);   res.first += get<0>(fig).first;  res.second += get<0>(fig).second;  res.first += get<1>(fig).first;  res.second += get<1>(fig).second;  res.first += get<2>(fig).first;  res.second += get<2>(fig).second;  res.first += get<3>(fig).first;  res.second += get<3>(fig).second;   res.first /= 4;  res.second /= 4;   return res; } //----------------------------------------------- template <class T> double Area(Figure<T>& fig){  //double a = distance(fig.p[0], fig.p[1]), b = distance(fig.p[1], fig.p[2]), c = distance(fig.p[2], fig.p[3]), d = distance(fig.p[3], fig.p[0]);  //double half\_per = (a + b + c + d)/2;  //if(!fig.Abstract)  return abs(fig.p[0].first \* fig.p[1].second + fig.p[1].first \* fig.p[2].second + fig.p[2].first \* fig.p[3].second + fig.p[3].first \* fig.p[0].second - fig.p[1].first \* fig.p[0].second - fig.p[2].first \* fig.p[1].second - fig.p[3].first \* fig.p[2].second - fig.p[0].first \* fig.p[3].second)/2;  //else   // return sqrt((half\_per - a) \* (half\_per - b) \* (half\_per - c) \* (half\_per - d)); } template <class T> double Area(tuple< pair<T, T>, pair<T, T>, pair<T, T>, pair<T, T> >& fig) {  return abs(get<0>(fig).first \* get<1>(fig).second + get<1>(fig).first \* get<2>(fig).second + get<2>(fig).first \* get<3>(fig).second + get<3>(fig).first \* get<0>(fig).second - get<1>(fig).first \* get<0>(fig).second - get<2>(fig).first \* get<1>(fig).second - get<3>(fig).first \* get<2>(fig).second - get<0>(fig).first \* get<3>(fig).second)/2;  } //----------------------------------------------------------------------------------------------------------- template <class T> ostream& operator << (ostream& os, const pair<T, T>& p) {  os << '(' << p.first << " , " << p.second << ')';  return os; }  template <class T> void print(ostream& os, Figure<T>& fig) {  if (fig.Abstract)  os << "Abstract: ";  else if(fig.Square)  os << "Square: ";  else if(fig.Rectangle)  os << "Rectangle: ";  else if(fig.Trapezoid)  os << "Trapezoid: ";  os << fig.p[0] << "; " << fig.p[1] << "; " << fig.p[2] << "; " << fig.p[3] << "; Centre = " << fig.centre << "; Area = " << fig.area << endl; }  template <class T> void print(ostream& os, tuple< pair<T, T>, pair<T, T>, pair<T, T>, pair<T, T> >& fig) {  int switcher = 0;  pair<T, T> p[4];  p[0] = get<0>(fig);  p[1] = get<1>(fig);  p[2] = get<2>(fig);  p[3] = get<3>(fig);   if (IsSquare(p)) switcher = 2;  else if (IsRectangle(p)) switcher = 1;  else if (IsTrapezoid(p)) switcher = 3;  else switcher = 0;   DoublePoint centre = Centre(fig);  double area = Area(fig);   switch (switcher)  {  case 3:  os << "Trapezoid: ";  break;  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 << endl; } //----------------------------------------------------------------------------------------------------------- bool operator == (pair<int, int> lhs, pair<int, int> rhs) {  return lhs.first == rhs.first && lhs.second == rhs.second; }  bool operator == (pair<double, double> lhs, pair<double, double> rhs) {  return DoubleEqual(lhs.first, rhs.first) && DoubleEqual(lhs.second, rhs.second); }  template <class T> Figure<T>::Figure(istream& is) {  cout << "Input point coordinates clockwise or counter clockwise\nA: ";  is >> p[0].first >> p[0].second;  cout << "B: ";  is >> p[1].first >> p[1].second;  cout << "C: ";  is >> p[2].first >> p[2].second;  cout << "D: ";  is >> p[3].first >> p[3].second;   if(IsSquare(p))  Square = true;  else if(IsRectangle(p))  Rectangle = true;  else if(IsTrapezoid(p))  Trapezoid = true;  else Abstract = true;   centre = Centre(\*this);  area = Area(\*this); } |

main1.cpp

|  |
| --- |
| //Barannikov Stepan M8o-201b /\* \*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; \*Вариант задания - 18: Квадрат, Прямоугольник, Трапеция. \*/ #include "Shape.h"  int main() {  cout << "Creating Square" << endl;  Figure<double> sqr(cin);  cout << "Creating Rectangle" << endl;  Figure<double> rect(cin);  cout << "Creating Trapezoid" << endl;  Figure<double> trap(cin);   print(cout, sqr);  print(cout, rect);  print(cout, trap);  } |

main2.cpp

|  |
| --- |
| //Barannikov Stepan M8o-201b /\* \*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; \*Вариант задания - 18: Квадрат, Прямоугольник, Трапеция. \*/ #include "Shape.h"  int main() {  cout << "Tuples\_Tuples\_Tuples\_Tuples\_Tuples\_Tuples\_Tuples\_Tuples\_Tuples\_" << endl << endl;  cout << "IntCoordinates\_IntCoordinates\_IntCoordinates\_" << endl;  typedef pair<int, int> IntPair;  typedef pair<double, double> DoublePair;  IntPair a, b, c, d;   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);   a = make\_pair(0, 0);  b = make\_pair(1, 4);  c = make\_pair(4, 4);  d = make\_pair(5, 0);  tuple<IntPair, IntPair, IntPair, IntPair> TrapArgs(a,b,c,d);  print(cout, TrapArgs);   cout << endl << endl << "DoubleCoordinates\_DoubleCoordinates\_DoubleCoordinates\_" << endl;  DoublePair e, f, g, h;   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);   e = make\_pair(0.5, 0.5);  f = make\_pair(1.5, 4.5);  g = make\_pair(4.5, 4.5);  h = make\_pair(5.5, 0.5);  tuple<DoublePair, DoublePair, DoublePair, DoublePair> TrapArgsDouble(e, f, g, h);  print(cout, TrapArgsDouble);   e = make\_pair(-1.8, -1.5);  f = make\_pair(1.5, 4.5);  g = make\_pair(4.5, 4.5);  h = make\_pair(5.5, 0.5);  tuple<DoublePair, DoublePair, DoublePair, DoublePair> AbstrArgsDouble(e, f, g, h);  print(cout, AbstrArgsDouble); } |

1. Вывод:

Программа умеет находить площадь любого Четырехугольника, и если нет заданного по заданию четырехугольника, то программа выводит данные с подписью `Abstract:`, в иных случаях все сделано по заданию.

## Список литературы

1. Шилдт, Герберт. С++: базовый курс, 3-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2018. - 624 с. : ил. - Парал. тит. англ.
2. Справочник по языку С++ [Электронный ресурс]. URL: [http://www.cplusplus.com/reference](http://www.cplusplus.com/reference/deque/) (дата обращения: 25.10.2019).
3. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 27.10.2019).