МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №5 по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Зинин Владислав Владимирович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

### Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с классами.

Знакомство с умными указателями.

### Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.  
  
Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.

Класс-контейнер должен соджержать объекты используя std:shared\_ptr<…>.

Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std.

Шаблоны (template).

Объекты «по-значению»

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер.

Распечатывать содержимое контейнера.

Удалять фигуры из контейнера.

**Описание программы**

Исходный код лежит в 10 файлах:

1. main.cpp - основная программа, взаимодействие с пользователем посредством команд из меню

2. include/figure.h - описание абстрактного класса фигур

3. include/point.h - описание класса точки

4. include/TVector.cpp - реализация функций контейнера первого уровня (в моем случае вектора)

5. include/TVector.h – реализация класса контейнера первого уровня (в моем случае вектора)

6. include/rhombus.h - описание класса ромба, наследующегося от figures

7. include/point.cpp - реализация класса точки

8. include/TVectorItem.cpp – реализация функций вспомогательного класса для контейнера

9. include/TVectorItem.h – описание вспомогательного класса для контейнера

10. include/rhombus.cpp: реализация класса ромба, наследующегося от figure

**Дневник отладки**

Во время выполнения лабораторной работы неисправностей почти не возникало, все было отлажено сразу же.

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №5 позволила мне полностью осознать концепцию умных указателей в языке С++ и отточить навыки в работе с ними. Всё прошло успешно.

**Исходный код**

**Figure.h**

|  |
| --- |
| #ifndef FIGURE\_H |
|  |

|  |
| --- |
| #define FIGURE\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Figure |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| virtual ~Figure(){}; |
|  |

|  |
| --- |
| virtual double Area() = 0; |
|  |

|  |
| --- |
| virtual size\_t VertexesNumber() = 0; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //FIGURE\_H

**Point.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Point::Point(): x\_(0.0), y\_(0.0) {} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Point::Point(double x, double y): x\_(x), y\_(y) {} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Point::Point(std::istream &is) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| is >> x\_ >> y\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::istream& operator>>(std::istream& is, Point& p) { |
|  |

|  |
| --- |
| is >> p.x\_ >> p.y\_; |
|  |

|  |
| --- |
| return is; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::ostream& operator<<(std::ostream& os, Point& p) { |
|  |

|  |
| --- |
| os << "(" << p.x\_ << ", " << p.y\_ << ")"; |
|  |

|  |
| --- |
| return os; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double get\_x(Point &other) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return other.x\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double get\_y(Point &other) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return other.y\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void Point::set\_x(Point &other, double x) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| other.x\_ = x; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void Point::set\_y(Point &other, double y) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| other.y\_ = y; |
|  |

}

**Point.h**

|  |
| --- |
| #ifndef POINT\_H |
|  |

|  |
| --- |
| #define POINT\_H |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Point |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| Point(); |
|  |

|  |
| --- |
| Point(double x, double y); |
|  |

|  |
| --- |
| Point(std::istream &is); |
|  |

|  |
| --- |
| double dist(Point &other); |
|  |

|  |
| --- |
| friend double get\_x(Point &other); |
|  |

|  |
| --- |
| friend double get\_y(Point &other); |
|  |

|  |
| --- |
| void set\_x(Point &other, double x); |
|  |

|  |
| --- |
| void set\_y(Point &other, double y); |
|  |

|  |
| --- |
| friend std::istream& operator>>(std::istream& is, Point& p); |
|  |

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& os, Point& p); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| double x\_, y\_; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //POINT\_H

**TVector.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVector.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*----init----\*/ |
|  |

|  |
| --- |
| TVector::TVector() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size = 0; |
|  |

|  |
| --- |
| std::cout << "TVector created" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*----bool----\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool TVector::Empty() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return size == 0?1:0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void TVector::InsertLast(std::shared\_ptr<Rhombus> &&rhomb) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> value (new TVectorItem(rhomb)); |
|  |

|  |
| --- |
| if(size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this->first = value; |
|  |

|  |
| --- |
| this->first->next = nullptr; |
|  |

|  |
| --- |
| this->first = value; |
|  |

|  |
| --- |
| size++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> end = this->first; |
|  |

|  |
| --- |
| while(end->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| end = end->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| end->next = value; |
|  |

|  |
| --- |
| value->next = nullptr; |
|  |

|  |
| --- |
| size++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void TVector::Resize(const size\_t new\_size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == new\_size || new\_size < 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else if(new\_size > size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size\_t iter = new\_size - size; |
|  |

|  |
| --- |
| for(int i = 0; i < iter; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| InsertLast(std::shared\_ptr<Rhombus>(new Rhombus())); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else{ |
|  |

|  |
| --- |
| size\_t iter = new\_size; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> end = this->first; |
|  |

|  |
| --- |
| for(int i = 0; i < iter; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| end = end->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| end->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| size = new\_size; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void TVector::RemoveLast() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "List is empty" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> del = this->first; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> del = this->first; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> save; |
|  |

|  |
| --- |
| while(del->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| save = del; |
|  |

|  |
| --- |
| del = del->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| save->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void TVector::Remove(size\_t idx) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(idx < 1 || idx > size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Invalid erase!" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> del; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> prev\_del; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> next\_del = this->first; |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| if(idx == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| del = this->first; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| this->first = next\_del; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int i = 1; i < idx; ++i) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| prev\_del = next\_del; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| del = next\_del; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| prev\_del->next = next\_del; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*-----Rhombus-----\*/ |
|  |

|  |
| --- |
| const Rhombus& TVector::Last() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> node = this->first; |
|  |

|  |
| --- |
| while(node->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| node = node->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*node->rhomb; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----destructor---\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TVector::~TVector() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "TVector deleted" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----size\_t---\*/ |
|  |

|  |
| --- |
| size\_t TVector::Length() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return size; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----operator---\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus& TVector::operator[](const size\_t idx) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> idx\_rhomb = this->first; |
|  |

|  |
| --- |
| for(int i = 1; i < idx; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| idx\_rhomb = idx\_rhomb->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*idx\_rhomb->rhomb; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::ostream& operator<<(std::ostream& os, TVector& obj) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(obj.size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << "TList is empty" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << "Print rhombus" << std::endl; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> print = obj.first; |
|  |

|  |
| --- |
| os << '['; |
|  |

|  |
| --- |
| for(int i = 0; i < obj.size - 1; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << print->rhomb->Area() << " " << "," << " "; |
|  |

|  |
| --- |
| print = print->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| os << print->rhomb->Area() << ']'; |
|  |

|  |
| --- |
| os << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return os; |
|  |

}

**TVector.h**

|  |
| --- |
| #ifndef TVECTOR\_H |
|  |

|  |
| --- |
| #define TVECTOR\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVectorItem.h" |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <memory> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class TVector |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| /\*-----init-----\*/ |
|  |

|  |
| --- |
| TVector(); |
|  |

|  |
| --- |
| /\*-----void-----\*/ |
|  |

|  |
| --- |
| void Remove(size\_t idx); |
|  |

|  |
| --- |
| void Resize(const size\_t new\_size); |
|  |

|  |
| --- |
| void InsertLast(std::shared\_ptr<Rhombus> &&rhomb); |
|  |

|  |
| --- |
| void RemoveLast(); |
|  |

|  |
| --- |
| /\*-----Rhombus-----\*/ |
|  |

|  |
| --- |
| const Rhombus& Last(); |
|  |

|  |
| --- |
| /\*-----bool-----\*/ |
|  |

|  |
| --- |
| bool Empty(); |
|  |

|  |
| --- |
| /\*-----size\_t-----\*/ |
|  |

|  |
| --- |
| size\_t Length(); |
|  |

|  |
| --- |
| /\*----operator-----\*/ |
|  |

|  |
| --- |
| Rhombus& operator[] (const size\_t idx); |
|  |

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& os, TVector& obj); |
|  |

|  |
| --- |
| /\*-----destructor-----\*/ |
|  |

|  |
| --- |
| ~TVector(); |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| size\_t size; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> first; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif//TVECTOR\_H

**TVectorItem.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVectorItem.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TVectorItem::TVectorItem(std::shared\_ptr<Rhombus>& rhomb) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this->rhomb = rhomb; |
|  |

|  |
| --- |
| this->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::ostream& operator<<(std::ostream& os, TVectorItem& obj) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << obj.rhomb << " "; |
|  |

|  |
| --- |
| return os; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TVectorItem::~TVectorItem() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "TVectorItem deleted" << std::endl; |
|  |

}

**TVectorItem.h**

|  |
| --- |
| #ifndef TVECTORITEM\_H |
|  |

|  |
| --- |
| #define TVECTORITEM\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <memory> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class TVectorItem |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| TVectorItem(std::shared\_ptr<Rhombus>& rhomb); |
|  |

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& os, TVectorItem& obj); |
|  |

|  |
| --- |
| ~TVectorItem(); |
|  |

|  |
| --- |
| std::shared\_ptr<Rhombus> rhomb; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem> next; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //TVECTORITEM\_H

**Main.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVector.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int main() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TVector list; |
|  |

|  |
| --- |
| /\*-----Test push\_front---\*/ |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,2), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,3), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,4), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,5), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,6), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,7), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| /\*-----Test pop\_front---\*/ |
|  |

|  |
| --- |
| list.RemoveLast(); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| list.RemoveLast(); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| /\*-----Test push\_back---\*/ |
|  |

|  |
| --- |
| // list.push\_front(std::shared\_ptr<Rhombus>(new Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3)))); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| // /\*-----Test pop\_back---\*/ |
|  |

|  |
| --- |
| // list.pop\_front(); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| // /\*-----Test clear---\*/ |
|  |

|  |
| --- |
| // list.clear(); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // list.push\_front(std::shared\_ptr<Rhombus>(new Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3)))); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| // /\*-----Test insert---\*/ |
|  |

|  |
| --- |
| // list.insert(std::shared\_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 1); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| // list.insert(std::shared\_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 3); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| // list.insert(std::shared\_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 2); |
|  |

|  |
| --- |
| // std::cout << list << std::endl; |
|  |

|  |
| --- |
| /\*-----Test erase---\*/ |
|  |

|  |
| --- |
| list.Resize(2); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| std::cout << "--------------" << std::endl; |
|  |

|  |
| --- |
| std::cout << list.Length() << std::endl; |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| std::cout << list[2] << std::endl; |
|  |

|  |
| --- |
| list.Resize(4); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| list.Resize(4); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| return 0; |
|  |

}

**Rhombus.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <math.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| a.set\_x(a, 1); |
|  |

|  |
| --- |
| a.set\_y(a, 1); |
|  |

|  |
| --- |
| b.set\_x(b, 2); |
|  |

|  |
| --- |
| b.set\_y(b, 2); |
|  |

|  |
| --- |
| c.set\_x(c, 0); |
|  |

|  |
| --- |
| c.set\_y(c, 3); |
|  |

|  |
| --- |
| d.set\_x(d, -1); |
|  |

|  |
| --- |
| d.set\_y(d, -1); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus(std::istream &is) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| is >> a; |
|  |

|  |
| --- |
| is >> b; |
|  |

|  |
| --- |
| is >> c; |
|  |

|  |
| --- |
| is >> d; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus(Point pa, Point pb, Point pc, Point pd): a(pa), b(pb), c(pc), d(pd) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Rhombus created" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // void Rhombus::Print(std::ostream &os) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // os << "Rhombus" << std::endl; |
|  |

|  |
| --- |
| // os << a << ',' << b << ',' << c << ',' << d << std::endl; |
|  |

|  |
| --- |
| // } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double Rhombus::Area() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return 0.5 \* fabs(get\_x(a)\*get\_y(b) + get\_x(b)\*get\_y(c) + get\_x(c)\*get\_y(d) + get\_x(d)\*get\_y(a) - get\_x(b)\*get\_y(a) - get\_x(c)\*get\_y(b) - get\_x(d)\*get\_y(c) - get\_x(a)\*get\_y(d)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::~Rhombus() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Rhombus deleted" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| size\_t Rhombus::VertexesNumber() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return 4; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::ostream& operator<<(std::ostream& os, Rhombus& p) { |
|  |

|  |
| --- |
| os << p.a << p.b << p.c << p.d; |
|  |

|  |
| --- |
| return os; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus& Rhombus::operator=(const Rhombus& other) { |
|  |

|  |
| --- |
| if (this == &other) |
|  |

|  |
| --- |
| return \*this; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| this->a = other.a; |
|  |

|  |
| --- |
| this->b = other.b; |
|  |

|  |
| --- |
| this->c = other.c; |
|  |

|  |
| --- |
| this->d = other.d; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout << "Rhombus copied" << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return \*this; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Point get\_a(const Rhombus& other) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // return other.a; |
|  |

|  |
| --- |
| // } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Point get\_b(Rhombus& other) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // return other.b; |
|  |

|  |
| --- |
| // } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Point get\_c(Rhombus& other) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // return other.c; |
|  |

|  |
| --- |
| // } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Point get\_d(Rhombus& other) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // return other.d; |
|  |

// }

**Rhombus.h**

|  |
| --- |
| #ifndef RHOMBUX\_H |
|  |

|  |
| --- |
| #define RHOMBUX\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
| #include "figure.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Rhombus : public Figure |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| Rhombus(); |
|  |

|  |
| --- |
| Rhombus(std::istream &is); |
|  |

|  |
| --- |
| Rhombus(Point a, Point b, Point c, Point d); |
|  |

|  |
| --- |
| double Area(); |
|  |

|  |
| --- |
| // void Print(std::ostream &os); |
|  |

|  |
| --- |
| size\_t VertexesNumber(); |
|  |

|  |
| --- |
| Rhombus& operator=(const Rhombus& other); |
|  |

|  |
| --- |
| // friend Point get\_a(Rhombus& other); |
|  |

|  |
| --- |
| // friend Point get\_b(Rhombus& other); |
|  |

|  |
| --- |
| // friend Point get\_c(Rhombus& other); |
|  |

|  |
| --- |
| // friend Point get\_d(Rhombus& other); |
|  |

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& os, Rhombus& p); |
|  |

|  |
| --- |
| virtual ~Rhombus(); |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| Point a, b, c, d; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //RHOMBUX\_H