

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №6 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:

Знакомство с шаблонами классов;

Построение шаблонов динамических структур данных.

Задание

Необходимо спроектировать и запрограммировать на языке C++ **шаблон класса-контейнера** первого уровня, содержащий **одну фигуру (колонка фигура 1)**, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы №1;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
- Шаблон класса-контейнера должен содержать объекты используя `std::shared_ptr<...>`.

Нельзя использовать:

- Стандартные контейнеры `std`.

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.inl - 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.inl – 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 были некие неисправности в работе шаблонов и компиляции программы, однако окончательный вариант полностью исправен.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №6 позволила мне полностью осознать одну из базовых и фундаментальных концепций языка C++ - работу с так называемыми шаблонами (templates).

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
```

TVector.h

```
#ifndef TVECTOR_H
```

```
#define TVECTOR_H
```

```
#include <iostream>
```

```
#include "TVectorItem.h"
```

```
#include "rhombus.h"
```

```
#include <memory>
```

```
template <class T> 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);
```

```
template <class B> friend std::ostream& operator<<(std::ostream& os, TVector<B> &obj);
```

```
/*-----destructor-----*/
```

```
~TVector();
```

```
private:
```

```
size_t size;
```

```
std::shared_ptr<TVectorItem<T>> first;
```

```

};

#include "TVector.inl"

#endif//TVECTOR_H
TVector.inl
#include <iostream>
#include "TVector.h"

/*----init----*/

template <class T> TVector<T>::TVector()
{
    size = 0;
    std::cout << "TVector created" << std::endl;
}

/*----bool----*/

template <class T> bool TVector<T>::Empty()
{
    return size == 0?1:0;
}

template <class T> void TVector<T>::InsertLast(std::shared_ptr<Rhombus> &&rhomb)
{
    std::shared_ptr<TVectorItem<T>> value (new TVectorItem<T>(rhomb));
    if(size == 0)
    {
        this->first = value;
        this->first->next = nullptr;
        this->first = value;
        size++;
    }
    else
    {
        std::shared_ptr<TVectorItem<T>> end = this->first;
        while(end->next != nullptr)
        {

```

```

end = end->next;
}
end->next = value;
value->next = nullptr;
size++;
}
}

template <class T> void TVector<T>::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<T>> end = this->first;
for(int i = 0; i < iter; i++)
{
end = end->next;
}
end->next = nullptr;
}
size = new_size;
}

template <class T> void TVector<T>::RemoveLast()

```

```

{
if(size == 0)
{
std::cout << "List is empty" << std::endl;
}
else
{
if(size == 1)
{
size--;
std::shared_ptr<TVectorItem<T>> del = this->first;
}
else
{
std::shared_ptr<TVectorItem<T>> del = this->first;
std::shared_ptr<TVectorItem<T>> save;
while(del->next != nullptr)
{
save = del;
del = del->next;
}
size--;
save->next = nullptr;
}
}
}

template <class T> void TVector<T>::Remove(size_t idx)
{
if(idx < 1 || idx > size)
{
std::cout << "Invalid erase!" << std::endl;
}
else
{

```



```

std::shared_ptr<TVectorItem<T>>> del;
std::shared_ptr<TVectorItem<T>>> prev_del;
std::shared_ptr<TVectorItem<T>>> 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-----*/
template <class T> const Rhombus& TVector<T>::Last()
{
std::shared_ptr<TVectorItem<T>>> node = this->first;
while(node->next != nullptr)
{
node = node->next;
}
return *node->rhomb;
}
/*-----destructor---*/

```

```

template <class T> TVector<T>::~TVector()
{
std::cout << "TVector deleted" << std::endl;
}

/*----size_t---*/

template <class A> size_t TVector<A>::Length()
{
return size;
}

/*----operator---*/

template <class T> Rhombus& TVector<T>::operator[](const size_t idx)
{
std::shared_ptr<TVectorItem<T>> idx_rhomb = this->first;
for(int i = 1; i < idx; i++)
{
idx_rhomb = idx_rhomb->next;
}
return *idx_rhomb->rhomb;
}

template <class T> std::ostream& operator<<(std::ostream& os, TVector<T>& obj)
{
if(obj.size == 0)
{
os << "TList is empty" << std::endl;
}
else
{
os << "Print rhombus" << std::endl;
std::shared_ptr<TVectorItem<T>> print = obj.first;
os << 'T';
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;
}

```

TVectorItem.h

```

#ifndef TVECTORITEM_H
#define TVECTORITEM_H

#include <iostream>
#include "rhombus.h"
#include <memory>

template <class T> class TVectorItem
{
public:
TVectorItem(std::shared_ptr<Rhombus>& rhomb);
template <class B> friend std::ostream& operator<<(std::ostream& os, TVectorItem<B> &obj);
~TVectorItem();
std::shared_ptr<T> rhomb;
std::shared_ptr<TVectorItem<T>> next;
};
#include "TVectorItem.inl"
#endif //TVECTORITEM_H

```

TVectorItem.inl

```

#include <iostream>
#include "TVectorItem.h"

template <class T> TVectorItem<T>::TVectorItem(std::shared_ptr<Rhombus>& rhomb)
{
this->rhomb = rhomb;
this->next = nullptr;
}

template <class B> std::ostream& operator<<(std::ostream& os, TVectorItem<B> &obj)

```

```

{
os << obj.rhomb << " ";
return os;
}

template <class T> TVectorItem<T>::~~TVectorItem()
{
std::cout << "TVectorItem deleted" << std::endl;
}

```

Main.cpp

```

#include <iostream>
#include "TVector.h"

int main()
{

TVector<Rhombus> 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 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;
}

```

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

```

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.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();
size_t VertexesNumber();
friend std::ostream& operator<<(std::ostream& os, Rhombus& p);
virtual ~Rhombus();
private:
Point a, b, c, d;
};

#endif //RHOMBUX_H

```


