МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент Абдуль-Хади Филипп, группа М8О-207Б-21

Преподаватель Дорохов Евгений Павлович

Условие

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну фигуру (квадрат)**. Классы должны удовлетворять следующим правилам:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. Классы фигур должны содержать набор следующих методов:
   * Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
   * Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1.
   * Оператор копирования (=)
   * Оператор сравнения с такими же фигурами (==)
3. Класс-контейнер должен соджержать объекты фигур “по значению” (не по ссылке).
4. Класс-контейнер должен содержать набор следующих методов:  
   size()  
   empty()

operator[]

resize()

push\_back()

pop\_back()

erase(size\_t pos)

clear()

operator<<

Описание программы

Исходный код лежит в 7 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством ввода команд
2. square.cpp: реализация класса квадрата
3. square.h: класс квадрата
4. tvector\_item.cpp: реализация класса контейнера для класса квадрата
5. tvector\_item.h: класс контейнер для класса квадрата
6. tvector.cpp: реализация вектора
7. tvector.h: класс вектора

Дневник отладки

|  |  |
| --- | --- |
| **Проблема** | **Исправление** |
| Неудобно собирать без make файла | Создал make файл |

Недочёты

Отсутствуют

Выводы

Перегрузка операторов очень удобная штука с широкой областью применения.

Исходный код ниже:

main.cpp

**#include <iostream>**

**#include "square.h"**

**#include "tvector.h"**

**using** **namespace** std;

**int** **main**()

{

cout << "Comands:" << endl;

cout << "a - add new square (a [side\_len])" << endl;

cout << "d - erase square by index (d [idx])" << endl;

cout << "s - set square by index (s [idx] [side\_len])" << endl;

cout << "p - print all containing squares (p)" << endl;

cout << "q - quit (q)" << endl;

**char** running = 1;

TVector \*vect = **new** TVector();

**char** cmd;

**while**(running)

{

cout << "> ";

cin >> cmd;

**switch**(cmd)

{

**case** 'a':

{

**int** sl;

cin >> sl;

vect->push\_back(Square(sl));

**break**;

}

**case** 'd':

{

**int** di;

cin >> di;

vect->erase(di);

**break**;

}

**case** 's':

{

**int** si,sl;

cin >> si >> sl;

Square **csq**(sl);

(\*vect)[si] = csq;

**break**;

}

**case** 'p':

{

cout << \*vect << endl;

**break**;

}

**case** 'q':

{

running = 0;

**break**;

}

**default**:

cout << "wrong input" << endl;

}

}

**delete** vect;

}

square.cpp

**#include "square.h"**

Square::Square(std::istream &is)

{

is >> \***this**;

}

Square::Square(**int** sideLen)

{

side\_len = sideLen;

}

Square::Square()

{

side\_len = 0;

}

**size\_t** **Square::VertexesNumber**()

{

**return** 4;

}

**double** **Square::Area**()

{

**return** side\_len\*side\_len;

}

**void** **Square::Print**(std::ostream& os)

{

os << \***this**;

}

Square& Square::**operator**=(**const** Square& sq)

{

side\_len=sq.side\_len;

**return** \***this**;

}

**bool** Square::**operator**==(**const** Square& sq)

{

**return** side\_len==sq.side\_len;

}

std::istream& **operator**>>(std::istream& is, Square& r)

{

is >> r.side\_len;

**return** is;

}

std::ostream& **operator**<<(std::ostream& os, Square& r)

{

os << "Square: " << r.side\_len << 'x' << r.side\_len;

**return** os;

}

square.h

**#ifndef SQUARE\_H**

**#define SQUARE\_H**

**#include <iostream>**

**class** **Square**

{

**public**:

Square(std::istream &is);

Square(**int** sideLen);

Square();

**size\_t** **VertexesNumber**();

**double** **Area**();

**void** **Print**(std::ostream& os);

Square& **operator**=(**const** Square& sq);

**bool** **operator**==(**const** Square& sq);

**friend** std::istream& **operator**>>(std::istream& is, Square& r);

**friend** std::ostream& **operator**<<(std::ostream& os, Square& r);

**private**:

**int** side\_len;

};

**#endif**

tvector\_item.cpp

**#include "tvector\_item.h"**

**#include <iostream>**

TVectorItem::TVectorItem(**const** et\_tvectoritem& elem)

{

**this**->element = elem;

}

TVectorItem::TVectorItem(**const** TVectorItem& another)

{

**this**->element = another.element;

}

**const** et\_tvectoritem **TVectorItem::getElement**()

{

**return** **this**->element;

}

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

{

os << "Item: " << obj.element;

**return** os;

}

TVectorItem::~TVectorItem()

{

}

tvector\_item.h

**#ifndef TVECTORITEM\_H**

**#define TVECTORITEM\_H**

**#define et\_tvectoritem Square**

**#include<iostream>**

**#include "square.h"**

**class** **TVectorItem**

{

**public**:

TVectorItem(**const** et\_tvectoritem& elem);

TVectorItem(**const** TVectorItem& another);

**const** et\_tvectoritem **getElement**();

~TVectorItem();

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

**private**:

et\_tvectoritem element;

};

**#endif**

tvector.cpp

**#include "tvector.h"**

**#include <iostream>**

**#include <cstring>**

TVector::TVector()

{

vals = NULL;

len = 0;

rLen = 0;

}

**int** **TVector::size**()

{

**return** len;

}

**char** **TVector::empty**()

{

**return** len == 0;

}

et\_tvector& TVector::**operator**[](**int** idx)

{

**return** vals[idx];

}

**void** **TVector::resize**(**int** newsize)

{

vals = (et\_tvector\*)realloc((**void**\*)vals, **sizeof**(et\_tvector)\*newsize);

}

**void** **TVector::push\_back**(**const** et\_tvectoritem& sq)

{

**if**(rLen)

{

**if**(len>=rLen)

{

rLen<<=1;

resize(rLen);

}

}

**else**

{

rLen=1;

resize(rLen);

}

vals[len]=TVectorItem(sq);

len++;

}

et\_tvectoritem **TVector::pop\_back**()

{

et\_tvector ret = vals[len-1];

erase(len-1);

**return** ret.getElement();

}

**void** **TVector::erase**(**int** pos)

{

memmove((**void**\*)&(vals[pos]),(**void**\*)&(vals[pos+1]),**sizeof**(et\_tvector)\*(len-pos-1));

len--;

**if**(len==rLen>>1)

resize(len);

}

**void** **TVector::clear**()

{

**if**(!empty())

{

**delete** vals;

vals = NULL;

len = 0;

rLen = 0;

}

}

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

{

os << '[';

**for**(**int** i = 0; i < obj.len; i++)

{

os << obj.vals[i];

**if**(i != obj.len - 1)

os << ", ";

}

os << ']';

**return** os;

}

TVector::~TVector()

{

clear();

}

tvector.h

**#ifndef TVECTOR\_H**

**#define TVECTOR\_H**

**#define et\_tvector TVectorItem**

**#include<iostream>**

**#include "tvector\_item.h"**

**class** **TVector**

{

**public**:

TVector();

~TVector();

**int** **size**();

**char** **empty**();

et\_tvector& **operator**[](**int** idx);

**void** **resize**(**int** newsize);

**void** **push\_back**(**const** et\_tvectoritem& sq);

et\_tvectoritem **pop\_back**();

**void** **erase**(**int** pos);

**void** **clear**();

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

**private**:

et\_tvector \*vals;

**int** len;

**int** rLen;

};

**#endif**