МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №4

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

Студент *Маринин Иван Сергеевич, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 10: Трапеция, Квадрат, Прямоугольник. Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур, согласно варианту задания. 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:
   * InsertFirst() – метод, добавляющий элемент в начало списка
   * InsertLast() – метод, добавляющий фигуру в конец списка
   * Insert() - метод, добавляющий фигуру в произвольное место списка
   * RemoveFirst() - метод, удаляющий первый элемент списка
   * RemoveLast() - метод, удаляющий последний элемент списка
   * Remove() - метод, удаляющий произвольный элемент списка
   * Empty() - метод, проверяющий пустоту списка
   * Length() - метод, возвращающий длину массива
   * operator<< – выводит связанный список в соответствии с заданным форматом в поток вывода
   * Clear() - метод, удаляющий все элементы контейнера, но позволяющий пользоваться им.

**Описание программы**

Исходный файл лежит в 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством комманд из меню
2. figure.h: описание абстрактного класса фигур
3. point.h: описание класса точки
4. square.h: описание класса квадрата, наследующегося от figures
5. hlist\_item.h: описание класса элемента связанного списка
6. tlinkedlist.h: описание класса связанного списка
7. point.cpp: реализация класса точки
8. square.cpp: реализация класса квадрата, наследующегося от figures
9. hlist\_item.cpp: реализация класса элемента связанного списка
10. tlinkedlist.cpp: реализация класса связанного списка

**Дневник отладки**

Возникли небольшие проблемы при отладке программы. После тестирования они были устранены.

ivanmarinin@MacBook-Air-Ivan lab\_2 % ./a.out

Square List created

Print Square List

(1; 2)(1; 3)(2; 3)(2; 2) , (11; 12)(11; 13)(12; 13)(12; 12) , (21; 22)(21; 23)(22; 23)(22; 22) , (31; 32)(31; 33)(32; 33)(32; 32)

3

1

(2; 3)(2; 4)(3; 4)(3; 3)

(21; 22)(21; 23)(22; 23)(22; 22)

(1; 1)(1; 2)(2; 2)(2; 1)

Print Square List

(2; 3)(2; 4)(3; 4)(3; 3) , (11; 12)(11; 13)(12; 13)(12; 12) , (21; 22)(21; 23)(22; 23)(22; 22)

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №4 - это модернизация последних лабораторных 2 семестра. Если на 1 курсе я реализовывал связный список при помощи структур на языке СИ, то сейчас я реализовал связный список при помощи ООП на языке С++. Лабораторная прошла успешно, я повторил старый материал и узнал, усвоил много нового.

**Исходный код**

hlist\_item.cpp

#include <iostream>

#include "hlist\_item.h"

HListItem::HListItem(const Square &square) {

this->square = square;

this->next = nullptr;

}

ostream& operator<<(ostream& os,HListItem& obj) {

os << "[" << obj.square << "]" << endl;

return os;

}

HListItem::~HListItem() {

}

point.cpp

#include "point.h"

#include <cmath>

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

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

Point::Point(istream &is) {

is >> x\_ >> y\_;

}

double Point::dist(Point& other) {

double dx = (other.x\_ - x\_);

double dy = (other.y\_ - y\_);

return sqrt(dx\*dx + dy\*dy);

}

istream& operator>>(istream& is, Point& p) {

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

return is;

}

ostream& operator<<(ostream& os, Point& p) {

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

return os;

}

double Point::x(){

return x\_;

}

double Point::y(){

return y\_;

}

square.cpp

#include <iostream>

#include "square.h"

#include <cmath>

Square::Square(): a(0,0),b(0,0),c(0,0),d(0,0) {}

Square::Square(istream &is) {

is >> a;

is >> b;

is >> c;

is >> d;

}

Square::Square(Point a1, Point b1,Point c1, Point d1): a(a1),b(b1),c(c1),d(d1) {}

double Square::Area() {

return pow(abs((a.y() - b.y())), 2);

}

Square::~Square() {}

size\_t Square::VertexesNumber() {

return 4;

}

Square::Square(Square& other):Square(other.a,other.b,other.c,other.d) {}

Square& Square::operator = (const Square& other) {

if (this == &other) return \*this;

a = other.a;

b = other.b;

c = other.c;

d = other.d;

return \*this;

}

Square& Square::operator== (const Square& other) {

if (this == &other)

cout << "Squares are equal" << endl;

else

cout << "Squares are not equal" << endl;

}

ostream& operator<<(ostream& os, Square& h) {

os << h.a << h.b << h.c << h.d;

return os;

}

tlinkedlist.cpp

#include <iostream>

#include "tlinkedlist.h"

TLinkedList::TLinkedList() {

size\_of\_list = 0;

HListItem\* front;

HListItem\* back;

cout << "Square List created" << endl;

}

TLinkedList::TLinkedList(const TLinkedList& other){

front = other.front;

back = other.back;

}

size\_t TLinkedList::Length() {

return size\_of\_list;

}

bool TLinkedList::Empty() {

return size\_of\_list;

}

Square& TLinkedList::GetItem(size\_t idx){

int k = 0;

HListItem\* obj = front;

while (k != idx){

k++;

obj = obj->next;

}

return obj->square;

}

Square& TLinkedList::First() {

return front->square;

}

Square& TLinkedList::Last() {

return back->square;

}

void TLinkedList::InsertLast(const Square &&square) {

HListItem\* obj = new HListItem(square);

if(size\_of\_list == 0) {

front = obj;

back = obj;

size\_of\_list++;

return;

}

back->next = obj;

back = obj;

obj->next = nullptr;

size\_of\_list++;

}

void TLinkedList::RemoveLast() {

if (size\_of\_list == 0) {

cout << "Square does not pop\_back, because the Square List is empty" << endl;

}

else {

if (front == back) {

RemoveFirst();

size\_of\_list--;

return;

}

HListItem\* prev\_del = front;

while (prev\_del->next != back) {

prev\_del = prev\_del->next;

}

prev\_del->next = nullptr;

delete back;

back = prev\_del;

size\_of\_list--;

}

}

void TLinkedList::InsertFirst(const Square &&square) {

HListItem\* obj = new HListItem(square);

if(size\_of\_list == 0) {

front = obj;

back = obj;

} else {

obj->next = front;

front = obj;

}

size\_of\_list++;

}

void TLinkedList::RemoveFirst() {

if (size\_of\_list == 0) {

cout << "Square does not pop\_front, because the Square List is empty" << endl;

} else {

HListItem\* del = front;

front = del->next;

delete del;

size\_of\_list--;

}

}

void TLinkedList::Insert(const Square &&square,size\_t position) {

if (position <0) {

cout << "Position < zero" << endl;

} else if (position > size\_of\_list) {

cout << " Position > size\_of\_list" << endl;

} else {

HListItem\* obj = new HListItem(square);

if (position == 0) {

front = obj;

back = obj;

} else {

int k = 0;

HListItem\* prev\_insert = front;

HListItem\* next\_insert;

while(k + 1 != position) {

k++;

prev\_insert = prev\_insert->next;

}

next\_insert = prev\_insert->next;

prev\_insert->next = obj;

obj->next = next\_insert;

}

size\_of\_list++;

}

}

void TLinkedList::Remove(size\_t position) {

if ( position > size\_of\_list ) {

cout << "Position " << position << " > " << "size " << size\_of\_list << " Not correct erase" << endl;

}

else if (position < 0) {

cout << "Position < 0" << endl;

}

else {

if (position == 0) {

RemoveFirst();

}

else {

int k = 0;

HListItem\* prev\_erase = front;

HListItem\* next\_erase;

HListItem\* del;

while( k+1 != position) {

k++;

prev\_erase = prev\_erase->next;

}

next\_erase = prev\_erase->next;

del = prev\_erase->next;

next\_erase = del->next;

delete del;

prev\_erase->next = next\_erase;

}

size\_of\_list--;

}

}

void TLinkedList::Clear() {

HListItem\* del = front;

HListItem\* prev\_del;

if(size\_of\_list !=0 ) {

while(del->next != nullptr) {

prev\_del = del;

del = del->next;

delete prev\_del;

}

delete del;

size\_of\_list = 0;

}

size\_of\_list = 0;

HListItem\* front;

HListItem\* back;

}

ostream& operator<<(ostream& os, TLinkedList& hl) {

if (hl.size\_of\_list == 0) {

os << "The square list is empty, so there is nothing to output" << endl;

} else {

os << "Print Square List" << endl;

HListItem\* obj = hl.front;

while(obj != nullptr) {

if (obj->next != nullptr) {

os << obj->square << " " << "," << " ";

obj = obj->next;

} else {

os << obj->square;

obj = obj->next;

}

}

os << endl;

}

return os;

}

TLinkedList::~TLinkedList() {

HListItem\* del = front;

HListItem\* prev\_del;

if(size\_of\_list !=0 ) {

while(del->next != nullptr) {

prev\_del = del;

del = del->next;

delete prev\_del;

}

delete del;

size\_of\_list = 0;

cout << "Square List deleted" << endl;

}

}

main.cpp

#include <iostream>

#include "tlinkedlist.h"

int main() {

TLinkedList tlinkedlist;

tlinkedlist.Empty();

tlinkedlist.InsertLast(Square(Point(1,2),Point(1,3),Point(2,3),Point(2, 2)));

tlinkedlist.InsertLast(Square(Point(11,12),Point(11,13),Point(12,13),Point(12, 12)));

tlinkedlist.InsertLast(Square(Point(21,22),Point(21,23),Point(22,23),Point(22,22)));

tlinkedlist.InsertLast(Square(Point(31,32),Point(31,33),Point(32,33),Point(32,32)));

cout << tlinkedlist;

tlinkedlist.RemoveLast();

cout << tlinkedlist.Length() << endl;

tlinkedlist.RemoveFirst();

tlinkedlist.InsertFirst(Square(Point(2,3),Point(2,4),Point(3,4),Point(3,3)));

tlinkedlist.Insert(Square(Point(1,1),Point(1,2),Point(2,2),Point(2, 1)),2);

cout << tlinkedlist.Empty() << endl;

cout << tlinkedlist.First() << endl;

cout << tlinkedlist.Last() << endl;

cout << tlinkedlist.GetItem(2) << endl;

tlinkedlist.Remove(2);

cout << tlinkedlist;

tlinkedlist.Clear();

return 0;

}