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 *Клитная Анастасия Викторовна, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель работы

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

Задание

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

Требования к классу фигуры аналогичны требованиям из лаб.работы 1.

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 (==)

Класс-контейнер должен соджержать объекты фигур “по значению” (не по ссылке).

Класс-контейнер должен содержать набор следующих методов:

TODO: по поводу методов в личку

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Дневник отладки

Во время выполнения лабораторной работы программа была несколько раз отлажена, так как плохо работала функция удаления из дерева. После

нескольких отладок программа стала работать исправно.

Недочёты

Недочётов не было обнаружено.

Выводы

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

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
```

```
#include "point.h"

class Figure {
public:
   virtual double Area() = 0;
   virtual void Print(std::ostream &os) = 0;
   virtual size_t VertexesNumber() = 0;
   virtual ~Figure() {};
};

#endif
```

main.cpp

```
#include <iostream>
#include "octagon.h"
#include "tlinked_list.h"

int main(){
  TLinkedList tlinkedlist;
  tlinkedlist.Empty();

tlinkedlist.InsertLast(Octagon(Point(1,2),Point(2,3),Point(3,4),Point(5,6),Point(7,8),Poi
nt(9,10), Point(11,12), Point(12,13)));

tlinkedlist.InsertLast(Octagon(Point(13,14),Point(14,15),Point(15,16),Point(16,17),Poin
t(17,18),Point(18,19),Point(19,20),Point(20,21)));

tlinkedlist.InsertLast(Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Poin
t(21,22),Point(23,24), Point(25,26),Point(27,28)));

tlinkedlist.InsertLast(Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Poin
t(21,22),Point(23,24), Point(25,26),Point(27,28)));
  std::cout << tlinkedlist;
  tlinkedlist.RemoveLast();
  std::cout << tlinkedlist.Length() << std::endl;
  tlinkedlist.RemoveFirst();

tlinkedlist.InsertFirst(Octagon(Point(2,3),Point(3,4),Point(4,5),Point(5,6),Point(6,7),Poi
nt(7,8), Point(8,9), Point(9,10)));

tlinkedlist.Insert(Octagon(Point(1,1),Point(2,3),Point(3,4),Point(5,6),Point(7,8),Point(9,
10), Point(11,12),Point(13,18)),2);
  tlinkedlist.Empty();
  std::cout << tlinkedlist.First() << std::endl;
  std::cout << tlinkedlist.Last() << std::endl;
```

```
    std::cout << tlinkedlist.GetItem(2) << std::endl;

    tlinkedlist.Remove(2);
    std::cout << tlinkedlist;
    tlinkedlist.Clear();
    return 0;
  }
```

Octagon.cpp

```
#include "octagon.h"
#include <cmath>

Octagon::Octagon(): point_a(0,0), point_b(0,0), point_c(0,0), point_d(0,0), point_e(0,0),
point_f(0,0), point_g(0,0), point_h(0,0){
}

Octagon::Octagon(std::istream& is) {
   std::cout << "Enter the octagon's vertexes:" << std::endl;
   is >> point_a;
   is >> point_b;
   is >> point_c;
   is >> point_d;
   is >> point_e;
   is >> point_f;
   is >> point_g;
   is >> point_h;
 //   std::cout << "The octagon is created" << std::endl;
}

Octagon::Octagon(Point point_a1, Point point_b1,Point point_c1, Point point_d1, Point
point_e1, Point point_f1, Point point_g1, Point point_h1 ):
point_a(point_a1),point_b(point_b1),point_c(point_c1),point_d(point_d1),point_e(point_e1
),point_f(point_f1), point_g(point_g1),point_h(point_h1) {
}


/*void Octagon::Print(std::ostream& os) {
   std::cout << "Octagon: ";
   std::cout << point_a << ", ";
   std::cout << point_b << ", ";
   std::cout << point_c << ", ";
   std::cout << point_d << ", ";
   std::cout << point_e << ", ";
   std::cout << point_f << ", ";
   std::cout << point_g << ", ";
   std::cout << point_h << std::endl;
}
```

```
*/
size_t Octagon::VertexesNumber() {
    size_t number = 8;
    return number;
}

Octagon& Octagon::operator = (const Octagon& other) {
  if (this == &other) return *this;
  point_a = other.point_a;
  point_b = other.point_b;
  point_c = other.point_c;
  point_d = other.point_d;
  point_e = other.point_e;
  point_f = other.point_f;
  point_g = other.point_g;
  point_h = other.point_h;
  return *this;
}

Octagon& Octagon::operator == (const Octagon& other) {
  if (this == &other){
    std::cout << "Octagons are equal" << std::endl;
  } else {
    std::cout << "Octagons are not equal" << std::endl;
  }
}

double Octagon::Area() {
   double q = abs(point_a.X() * point_b.Y() + point_b.X() * point_c.Y() + poiny_c.X() *
point_d.Y() + point_d.X() * point_e.Y() + point_e.X() * point_f.Y() + point_f.X() *
point_g.Y() + point_g.X() * point_h.Y() + point_h.X() * point_a.Y() - point_b.X() *
point_a.Y() - point_c.X() * point_b.Y() - point_d.X() * point_c.Y() - point_e.X() * point_d.Y()
- point_f.X() * point_e.Y() - point_g.X() * point_f.Y() - point_h.X() * point_g.Y() -
point_a.X() * point_h.Y());
   double s = q / 2;
   return s;

}

Octagon::~Octagon() {
}

std::ostream& operator<<(std::ostream& os, Octagon& p) {
  os << p.point_a << p.point_b << p.point_c << p.point_d << p.point_e <<
p.point_f<<p.point_g<<p.point_h;
  return os;
}
```

Octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include "figure.h"

class Octagon : public Figure{
public:
    Octagon();
    Octagon(std::istream& is);
    Octagon(Point point_a, Point point_b, Point point_c, Point point_d, Point point_e, Point
point_f, Point point_g, Point point_h );
    size_t VertexesNumber();
    Octagon(Octagon &other);
    double Area();
    //void Print(std::ostream& os);
    virtual ~Octagon();
    Octagon& operator=(const Octagon& other);
    Octagon& operator==(const Octagon& other);
    friend std::ostream& operator<<(std::ostream& os, Octagon& p);

private:
    Point point_a, point_b, point_c, point_d, point_e, point_f, point_g, point_h, ;
};

#endif // OCTAGON_H
```

Point.cpp

```
#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_;
}

double Point::dist(Point& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx * dx + dy * dy);
}

double Point::X(){
   return x_;
```

```
};

double Point::Y(){
   return 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;
}
```

Point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>
#include <vector>
#include <cmath>

class Point {
public:
    Point();
    Point(std::istream& is);
    Point(double x, double y);

    double dist(Point& other);
    double X();
    double Y();


    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);

    friend class Square;
    friend class Octagon;
    friend class Triangle;

private:
    double x_;
    double y_;
};
```

```
#endif // POINT_H
```

tlinked_list.cpp

```
#include <iostream>
#include "tlinked_list.h"

TLinkedList::TLinkedList() {
  size_of_list = 0;
  HListItem* front;
  HListItem* back;
  std::cout << "Octagon List created" << std::endl;
}

TLinkedList::TLinkedList(const TLinkedList& other){
  front = other.front;
  back = other.back;
}
bool TLinkedList::Empty() {
  return size_of_list;
}

size_t TLinkedList::Length() {
  return size_of_list;
}

Octagon& TLinkedList::GetItem(size_t idx){
  int k = 0;
  HListItem* obj = front;
  while (k != idx){
    k++;
    obj = obj->next;
  }
  return obj->hexagon;
}
HListItem& TLinkedList::First() {
  return front->octagon;
}
HListItem& TLinkedList::Last() {
  return back->octagon;
}
void TLinkedList::InsertLast(const Octagon &&octagon) {
  HListItem* obj = new HListItem(octagon);
  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) {
    std::cout << "Octagon does not pop_back, because the Octagon List is empty" << std::
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 Octagon &&octagon) {
    HListItem* obj = new HListItem(octagon);
    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) {
      std::cout << "Octagon does not pop_front, because the Octagon List is empty" <<
std:: endl;
    } else {
    HListItem* del = front;
    front = del->next;
```

```
    delete del;
    size_of_list--;
    }
}
void TLinkedList::Insert(const Octagon &&octagon,size_t position) {
  if (position <0) {
    std::cout << "Position < zero" << std::endl;
  } else if (position > size_of_list) {
    std::cout << " Position > size_of_list" << std::endl;
  } else {
    HListItem* obj = new HListItem(octagon);
    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 ) {
    std:: cout << "Position " << position << " > " << "size " << size_of_list << " Not correct
erase" << std::endl;
  } else if (position < 0) {
    std::cout << "Position < 0" << std::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;
}
std::ostream& operator<<(std::ostream& os, TLinkedList& ol) {
  if (hl.size_of_list == 0) {
    os << "The octagon list is empty, so there is nothing to output" << std::endl;
  } else {
    os << "Print Octagon List" << std::endl;
    HListItem* obj = ol.front;
    while(obj != nullptr) {
      if (obj->next != nullptr) {
        os << obj->octagon << " " << "," << " ";
        obj = obj->next;
      } else {
        os << obj->octagon;
        obj = obj->next;
      }
    }
    os << std::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;
    std::cout << "Octagon List deleted" << std::endl;
  }
}
```

tlinked_list.h

```
#ifdef TLINKED_LIST_H
#define TLINKED_LIST_H
#include <iostream>
#include "tlinked_list_item.h"
#include "octagon.h"

class TLinkedList {
public:
 TLinkedList();
 int size_of_list;
 size_t Lenght();
 bool Empty();
 TLinkedList& First();
 TLinkedList& Last();
 TLinkedList(const TLinkedList& other);
 Octagon& GetItem(size_t idx);
 //void Empty();
 void InsertFirst(const Octagon &&octagon);
 void InsertLast(const Octagon &&octagon);
 void RemoveLast();
 void RemoveFirst();
 void Insert(Octagon &&octagon, size_t position);
 void Remove(size_t position);
 void Clear();
 friend std::ostream& operator<<(std::ostream& os, TLinkedList& list);
 ~TLinkedList();
private:
 HListItem *front;
 HListItem *back;
};
#endif //TLINKED_LIST_H
```

tlinked_list_item.cpp

```
#include "tlinked_list_item.h"
#include "octagon.h"
#include <iostream>

 HListItem:: HListItem(const Octagon& octagon) {
    this->octagon = octagon;
    this->next = nullptr;

}
std::ostream& operator<<(std::ostream& os,  HListItem& obj){
  os << "["<<obj.octagon << "]"<<std::endl;
  return os;
}
 HListItem::~ HListItem(){
}
```

tlinked_list_item.h

```
#include<iostream>
#include "octagon.h"

class  HListItem {
public:
     HListItem(const Octagon& octagon);
    friend std::ostream& operator<<(std::ostream& os,  HListItem& obj);
    ~HListItem();
    HListItem* next;
    Octagon octagon;

};
```