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

# ЛАБОРАТОРНАЯ РАБОТА №7 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 шаблонами классов;

Построение итераторов для динамических структур данных.

## Задание

Используя структуру данных, разработанную для лабораторной работы №4, спроектировать и разработать **итератор** для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен позволять работать с любыми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например:

Нельзя использовать:

Стандартные контейнеры std.

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 возникли некоторые проблемы с реализацией списка, но все они были исправлены.

# Недочёты

Недочётов не было обнаружено.

### Выводы

Лабораторная работа №7 позволила мне реализовать свой класс Iterator на языке C++, были освоены базовые навыки работы с итераторами, создаваемыми вручную, и итерирование по созданному контейнеру.

## 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
```

# 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(std::istream &is) {
  is >> x >> y;
}


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;
}

bool operator == (Point &p1, Point& p2) {
  return (p1.x == p2.x && p1.y == p2.y);
}
```

# Point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

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

```
  friend bool operator == (Point& p1, Point& p2);
  friend class Pentagon;
  double X();
  double Y();
  friend std::istream& operator>>(std::istream& is, Point& p);
  friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
  double x;
  double y;
};

#endif
```

**hlist_item.h**

```
#ifndef HLISTITEM_H
#define HLISTITEM_H
#include <iostream>
#include "octagon.h"
#include <memory>

template <class T> class HListItem {
public:
  HListItem(const std::shared_ptr<Octagon> &octagon);
  template <class A> friend std::ostream& operator<<(std::ostream& os, HListItem<A> &obj);
  ~HListItem();
  std::shared_ptr<T> octagon;
  std::shared_ptr<HListItem<T>> next;
  std::shared_ptr<HListItem<T>> SetNext(std::shared_ptr<HListItem<T>> &next_);
  std::shared_ptr<HListItem<T>> GetNext();
  std::shared_ptr<T>& GetValue();
};
#include "hlist_item.inl"
#endif //HLISTITEM_H
```

**hlist_item.inl**

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

template <class T> HListItem<T>::HListItem(const std::shared_ptr<Octagon> &octagon) {
  this->octagon = octagon;
  this->next = nullptr;
}
template <class T> std::shared_ptr<HListItem<T>>
HListItem<T>::SetNext(std::shared_ptr<HListItem<T>> &next_) {
  std::shared_ptr<HListItem<T>> prev = this->next;
```

```
  this->next = next_;
  return prev;
}
template <class T> std::shared_ptr<T>& HListItem<T>::GetValue() {
  return this->octagon;
}
template <class T> std::shared_ptr<HListItem<T>> HListItem<T>::GetNext() {
  return this->next;
}
template <class A> std::ostream& operator<<(std::ostream& os,HListItem<A> &obj) {
  os << "[" << obj.octagon << "]" << std::endl;
  return os;
}
template <class T> HListItem<T>::~HListItem() {
}
```

**main.cpp**

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

int main() {

  TLinkedList<Octagon> tlinkedlist;
  std::cout << tlinkedlist.Empty() << std::endl;
  tlinkedlist.InsertLast(std::shared_ptr<Octagon>(new
Octagon(Point(1,2),Point(2,3),Point(3,4),Point(5,6),Point(7,8),Point(9,10), Point(11,12),
Point(12,13))));
  tlinkedlist.InsertLast(std::shared_ptr<Octagon>(new
Octagon(Point(13,14),Point(14,15),Point(15,16),Point(16,17),Point(17,18),Point(18,19),Point(
19,20),Point(20,21))));
  tlinkedlist.InsertLast(std::shared_ptr<Octagon>(new
Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Point(21,22),Point(23,24),
Point(25,26),Point(27,28))));
  tlinkedlist.InsertLast(std::shared_ptr<Octagon>(new
Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Point(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(std::shared_ptr<Octagon>(new
Octagon(Point(2,3),Point(3,4),Point(4,5),Point(5,6),Point(6,7),Point(7,8),
Point(8,9),Point(9,10))));
  tlinkedlist.Insert(std::shared_ptr<Octagon>(new
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);

  std::cout << tlinkedlist.Empty() << std::endl;
```

```
  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),po
int_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
```

**titerator.h**

```
#include <memory>
#ifndef INC_5_LABA__TITERATOR_H_
#define INC_5_LABA__TITERATOR_H_
template <class node, class T> class Titerator {
public:
  Titerator(std::shared_ptr<node> n) { node_ptr = n; }
  std::shared_ptr<T> operator*() { return node_ptr->GetValue(); }
  std::shared_ptr<T> operator->() { return node_ptr->GetValue(); }
  void operator++() { node_ptr = node_ptr->GetNext(); }
  Titerator operator++(int) {
    Titerator other(*this);
    ++(*this);
    return other;
  }
  bool operator==(Titerator const &i) { return node_ptr == i.node_ptr; };
  bool operator!=(Titerator const &i) { return node_ptr != i.node_ptr; };

private:
  std::shared_ptr<node> node_ptr;
```

```
};

#endif // INC_5_LABA__TITERATOR_H_
```

**tlinkedlist.h**

```
#ifndef HLIST_H
#define HLIST_H
#include <iostream>
#include "hlist_item.h"
#include "octagon.h"
#include <memory>
#include "titerator.h"

template <class T> class TLinkedList {
public:
  TLinkedList();
  int size_of_list;
  size_t Length();
  std::shared_ptr<T>& First();
  std::shared_ptr<Octagon>& Last();
  std::shared_ptr<Octagon>& GetItem(size_t idx);
  bool Empty();
  TLinkedList(const std::shared_ptr<TLinkedList> &other);
  void InsertFirst(const std::shared_ptr<Octagon> &&octagon);
  void InsertLast(const std::shared_ptr<Octagon> &&octagon);
  void RemoveLast();
  void RemoveFirst();
  void Insert(const std::shared_ptr<Octagon> &&octagon, size_t position);
  void Remove(size_t position);
  void Clear();
  template <class A> friend std::ostream& operator<<(std::ostream& os, TLinkedList<A>&
list);
  ~TLinkedList();
  Titerator<HListItem<T>, T> begin();
  Titerator<HListItem<T>, T> end();
private:
  std::shared_ptr<HListItem<T>> front;
  std::shared_ptr<HListItem<T>> back;
};
#include "tlinkedlist.inl"
#endif //HList_H
```

**tlinkedlist.inl**

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

template <class T>
```

```
Titerator<HListItem<T>, T> TLinkedList<T>::begin() {
  return Titerator<HListItem<T>, T> (front);
}

template <class T>
Titerator<HListItem<T>, T> TLinkedList<T>::end() {
  return Titerator<HListItem<T>, T>(back);
}

template <class T> TLinkedList<T>::TLinkedList() {
  size_of_list = 0;
  std::shared_ptr<HListItem<T>> front = nullptr;
  std::shared_ptr<HListItem<T>> back = nullptr;
  std::cout << "Octagon List created" << std::endl;
}
template <class T> TLinkedList<T>::TLinkedList(const std::shared_ptr<TLinkedList> &other){
  front = other->front;
  back = other->back;
}
template <class T> size_t TLinkedList<T>::Length() {
  return size_of_list;
}
template <class T> bool TLinkedList<T>::Empty() {
  return size_of_list;
}
template <class T> std::shared_ptr<Hexagon>& TLinkedList<T>::GetItem(size_t idx){
  int k = 0;
  std::shared_ptr<HListItem<T>> obj = front;
  while (k != idx){
    k++;
    obj = obj->GetNext();
  }
  return obj->GetValue();
}
template <class T> std::shared_ptr<T>& TLinkedList<T>::First() {
    return front->GetValue();
}
template <class T> std::shared_ptr<Hexagon>& TLinkedList<T>::Last() {
  return back->GetValue();
}
template <class T> void TLinkedList<T>::InsertLast(const std::shared_ptr<Octagon>
&&octagon) {
  std::shared_ptr<HListItem<T>> obj (new HListItem<T>(octagon));
 // std::shared_ptr<HListItem<T>> obj =
std::make_shared<HListItem<T>>(HListItem<T>(octagon));
  if(size_of_list == 0) {
    front = obj;
    back = obj;
    size_of_list++;
```

```
    return;
  }
  back->SetNext(obj); // = obj;
  back = obj;
  obj->next = nullptr; // = nullptr;
  size_of_list++;
}
template <class T> void TLinkedList<T>::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;
    }
    std::shared_ptr<HListItem<T>> prev_del = front;
    while (prev_del->GetNext() != back) {
      prev_del = prev_del->GetNext();
    }
    prev_del->next = nullptr;
    back = prev_del;
    size_of_list--;
    }
}
template <class T> void TLinkedList<T>::InsertFirst(const std::shared_ptr<Octagon>
&&octagon) {
    std::shared_ptr<HListItem<T>> obj (new HListItem<T>(octagon));
    if(size_of_list == 0) {
      front = obj;
      back = obj;
    } else {
      obj->SetNext(front); // = front;
      front = obj;
    }
    size_of_list++;
}
template <class T> void TLinkedList<T>::RemoveFirst() {
    if (size_of_list == 0) {
      std::cout << "Octagon does not pop_front, because the Octagon List is empty" << std::
endl;
    } else {
    std::shared_ptr<HListItem<T>> del = front;
    front = del->GetNext();
    size_of_list--;
    }
}
template <class T> void TLinkedList<T>::Insert(const std::shared_ptr<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 {
    std::shared_ptr<HListItem<T>> obj (new HListItem<T>(octagon));
    if (position == 0) {
      front = obj;
      back = obj;
    } else {
      int k = 0;
      std::shared_ptr<HListItem<T>> prev_insert = front;
      std::shared_ptr<HListItem<T>> next_insert;
      while(k+1 != position) {
        k++;
        prev_insert = prev_insert->GetNext();
      }
      next_insert = prev_insert->GetNext();
      prev_insert->SetNext(obj); // = obj;
      obj->SetNext(next_insert); // = next_insert;
    }
    size_of_list++;
  }
}
template <class T> void TLinkedList<T>::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;
      std::shared_ptr<HListItem<T>> prev_erase = front;
      std::shared_ptr<HListItem<T>> next_erase;
      std::shared_ptr<HListItem<T>> del;
      while( k+1 != position) {
        k++;
        prev_erase = prev_erase->GetNext();
      }
      next_erase = prev_erase->GetNext();
      del = prev_erase->GetNext();
      next_erase = del->GetNext();
      prev_erase->SetNext(next_erase); // = next_erase;
    }
    size_of_list--;
```

```
  }
}
template <class T> void TLinkedList<T>::Clear() {
  std::shared_ptr<HListItem<T>> del = front;
  std::shared_ptr<HListItem<T>> prev_del;
  if(size_of_list !=0 ) {
    while(del->GetNext() != nullptr) {
      prev_del = del;
      del = del->GetNext();
    }
    size_of_list = 0;
    //   std::cout << "HListItem deleted" << std::endl;
  }
  size_of_list = 0;
  std::shared_ptr<HListItem<T>> front;
  std::shared_ptr<HListItem<T>> back;
}
template <class T> std::ostream& operator<<(std::ostream& os, TLinkedList<T>& ol) {
  if (ol.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;
    std::shared_ptr<HListItem<T>> obj = ol.front;
    while(obj != nullptr) {
      if (obj->GetNext() != nullptr) {
        os << obj->GetValue() << " " << "," << " ";
        obj = obj->GetNext();
      } else {
        os << obj->GetValue();
        obj = obj->GetNext();
      }
    }
    os << std::endl;
  }
  return os;
}
template <class T> TLinkedList<T>::~TLinkedList() {
  std::shared_ptr<HListItem<T>> del = front;
  std::shared_ptr<HListItem<T>> prev_del;
  if(size_of_list !=0 ) {
    while(del->GetNext() != nullptr) {
      prev_del = del;
      del = del->GetNext();
    }
    size_of_list = 0;
    std::cout << "Octagon List deleted" << std::endl;
  }
}
```