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

ЛАБОРАТОРНАЯ РАБОТА №7

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

Студент *Каширин Кирилл Дмитриевич, группа М8О-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 7: Связанный список. 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:

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

Нельзя использовать:

1. Стандартные контейнеры std.

Программа должна позволять:

1. Вводить произвольное количество фигур и добавлять их в контейнер.
2. Распечатывать содержимое контейнера.
3. 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. figure.h: описание абстрактного класса фигур
3. point.h: описание класса точки
4. hexagon.h: описание класса шестиугольника, 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. hexagon.cpp: реализация класса шестиугольника, наследующегося от figures
9. hlist_item.inl: реализация класса элемента связанного списка
10. tlinkedlist.inl: реализация класса связанного списка

Дневник отладки

Недочёты

Недочетов не заметил

Выводы

В данной лабораторной работе я познакомился с шаблонами, которые работают с различными типами данных. Преимущество шаблона в том, что используется обобщенное программирование, код может использоваться многократно. Но есть и свои недодастки, а именно увеличивается время компиляции программы из-за того, что для каждого типа параметра шаблона компилятор создаст свой бинарный код.

Исходный код:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
#include "point.h"
class Figure {
public:
    virtual double Area() = 0;
    virtual size_t VertexesNumber() = 0;
    virtual ~Figure() {};
};

#endif // FIGURE_H
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

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

  double dist(Point& other);

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

  double x();
  double y();

private:
  double x_;
  double y_;
};

#endif // POINT_H
```

point.cpp

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

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 Point::x(){
  return x_;
}

double Point::y(){
  return y_;
}
```

hexagon.h

```cpp
#ifndef HEXAGON_H
#define HEXAGON_H
#include <iostream>
#include "figure.h"
```

```cpp
#include "point.h"
#include <memory>

class Hexagon : public Figure {
public:
    Hexagon();
    Hexagon(std::istream &is);
    Hexagon(Point a, Point b, Point c, Point d, Point e, Point f);
    Hexagon(std::shared_ptr<Hexagon>& other);
    double Area();
    size_t VertexesNumber();
    virtual ~Hexagon();
    Hexagon& operator=(const Hexagon& other);
    Hexagon& operator==(const Hexagon& other);
    friend std::ostream& operator<<(std::ostream& os, Hexagon& h);
private:
    Point a, b, c, d, e, f;
};

#endif // HEXAGON_H
```

hexagon.cpp

```cpp
#include <iostream>
#include "hexagon.h"
#include <cmath>

Hexagon::Hexagon(): a(0,0),b(0,0),c(0,0),d(0,0),e(0,0),f(0,0) {
}
Hexagon::Hexagon(std::istream &is) {
    is >> a;
    is >> b;
    is >> c;
    is >> d;
    is >> e;
    is >> f;
}
Hexagon::Hexagon(Point a1, Point b1,Point c1, Point d1, Point e1, Point f1): a(a1),b(b1)
}
double Hexagon::Area() {
    return 0.5*abs(a.x()*b.y()+b.x()*c.y()+c.x()*d.y()+d.x()*e.y()+e.x()*f.y()+f.x()*a.y
}
Hexagon::~Hexagon() {
```

```
}
size_t Hexagon::VertexesNumber() {
    return 6;
}
Hexagon::Hexagon(std::shared_ptr<Hexagon>& other):Hexagon(other->a,other->b,other->c,oth
}
Hexagon& Hexagon::operator = (const Hexagon& other) {
  if (this == &other) return *this;
  a = other.a;
  b = other.b;
  c = other.c;
  d = other.d;
  e = other.e;
  f = other.f;
  //std::cout << "Hexagon copied" << std::endl;
  return *this;
}
Hexagon& Hexagon::operator == (const Hexagon& other) {
  if (this == &other){
    std::cout << "Hexagons are equal" << std::endl;
  } else {
    std::cout << "Hexagons are not equal" << std::endl;
  }
}
std::ostream& operator<<(std::ostream& os, Hexagon& h) {
  os << h.a << h.b << h.c << h.d << h.e << h.f;
  return os;
}
```

hlist_item.h
hlist_item.inl
tlinkedlist.h

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

template <class T> class TLinkedList {
```

```
public:
  TLinkedList();
  int size_of_list;
  size_t Length();
  Hexagon& First();
  Hexagon& Last();
  Hexagon& GetItem(size_t idx);
  bool Empty();
  TLinkedList(const std::shared_ptr<TLinkedList> &other);
  void InsertFirst(std::shared_ptr<Hexagon> &&hexagon);
  void InsertLast(std::shared_ptr<Hexagon> &&hexagon);
  void RemoveLast();
  void RemoveFirst();
  void Insert(std::shared_ptr<Hexagon> &&hexagon, size_t position);
  void Remove(size_t position);
  void Clear();
  template <class A> friend std::ostream& operator<<(std::ostream& os,const TLinkedList<
  ~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>(nullptr);
}

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 << "Hexagon 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> Hexagon& TLinkedList<T>::GetItem(size_t idx){
  int k = 0;
  std::shared_ptr<HListItem<T>> obj = front;
  while (k != idx){
    k++;
    obj = obj->next;
  }
  return *obj->hexagon;
}
template <class T> Hexagon& TLinkedList<T>::First() {
    return *front->hexagon;
}
template <class T> Hexagon& TLinkedList<T>::Last() {
  std::shared_ptr<HListItem<T>> node = this->front;
    while(node->next != nullptr) {
        node = node->next;
    }
    return *node->hexagon;
}
template <class T> void TLinkedList<T>::InsertLast( std::shared_ptr<Hexagon> &&hexagon)
  std::shared_ptr<HListItem<T>> obj (new HListItem<T>(hexagon));
 // std::shared_ptr<HListItem<T>> obj = std::make_shared<HListItem<T>>(HListItem<T>(hexa
  if(size_of_list == 0) {
    front = obj;
    front->next = nullptr;
    size_of_list++;
    return;
```

```
  }
  std::shared_ptr<HListItem<T>> end = this->front;
  while(end->next != nullptr) {
    end = end->next;
  }
  end->next = obj;
  obj->next = nullptr;
  size_of_list++;
}
template <class T> void TLinkedList<T>::RemoveLast() {
  if (size_of_list == 0) {
    std::cout << "Hexagon does not pop_back, because the Hexagon List is empty" << std::
  } else {
    if (size_of_list == 1) {
      RemoveFirst();
      size_of_list--;
      return;
    }
    std::shared_ptr<HListItem<T>> prev_del = front;
    std::shared_ptr<HListItem<T>> save;
    while (prev_del->next != nullptr) {
      save = prev_del;
      prev_del = prev_del->next;
    }
    save->next = nullptr;
    prev_del->next = nullptr;
    size_of_list--;
    }
}
template <class T> void TLinkedList<T>::InsertFirst( std::shared_ptr<Hexagon> &&hexagon)
    std::shared_ptr<HListItem<T>> obj (new HListItem<T>(hexagon));
    if(size_of_list == 0) {
      front = obj;
    } else {
      obj->next = front;
      front = obj;
    }
    size_of_list++;
}
template <class T> void TLinkedList<T>::RemoveFirst() {
    if (size_of_list == 0) {
      std::cout << "Hexagon does not pop_front, because the Hexagon List is empty" << st
```

```
    } else {
    std::shared_ptr<HListItem<T>> del = front;
    front = del->next;
    size_of_list--;
    }
}
template <class T> void TLinkedList<T>::Insert(std::shared_ptr<Hexagon> &&hexagon,size_t
  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>(hexagon));
    if (position == 0) {
      front = 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->next;
      }
      next_insert = prev_insert->next;
      prev_insert->next = obj; // = obj;
      obj->next = 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 c
  } 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->next;
      }
      next_erase = prev_erase->next;
      del = prev_erase->next;
      next_erase = del->next;
      prev_erase->next = 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->next != nullptr) {
      prev_del = del;
      del = del->next;
    }
    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,const TLinkedList<T>& hl) {
  if (hl.size_of_list == 0) {
    os << "The hexagon list is empty, so there is nothing to output" << std::endl;
  } else {
    os << "Print Hexagon List" << std::endl;
    std::shared_ptr<HListItem<T>> obj = hl.front;
    for(int i = 0; i < hl.size_of_list - 1; i++)
    {
        os << *obj->hexagon << " " << "," << " ";
        obj = obj->next;
    }
    os << *obj->hexagon;
    os << std::endl;
    }
```

```
    return os;
}
template <class T> TLinkedList<T>::~TLinkedList() {
}
```

main.cpp

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

int main() {

  TLinkedList<Hexagon> tlinkedlist;
  std::cout << tlinkedlist.Empty() << std::endl;
  tlinkedlist.InsertLast(std::shared_ptr<Hexagon>(new Hexagon(Point(1,2),Point(2,3),Poin
  tlinkedlist.InsertLast(std::shared_ptr<Hexagon>(new Hexagon(Point(11,12),Point(12,13),
  tlinkedlist.InsertLast(std::shared_ptr<Hexagon>(new Hexagon(Point(17,18),Point(18,19),
  tlinkedlist.InsertLast(std::shared_ptr<Hexagon>(new Hexagon(Point(17,18),Point(18,19),
  std::cout << tlinkedlist;
  tlinkedlist.RemoveLast();
  std::cout << tlinkedlist.Length() << std::endl;
  tlinkedlist.RemoveFirst();
  tlinkedlist.InsertFirst(std::shared_ptr<Hexagon>(new Hexagon(Point(2,3),Point(3,4),Poi
  tlinkedlist.Insert(std::shared_ptr<Hexagon>(new Hexagon(Point(1,1),Point(2,3),Point(3,
  std::cout << tlinkedlist.Empty() << std::endl;
  std::cout << tlinkedlist.First() << std::endl;
  std::cout << tlinkedlist.Last() << std::endl;
  std::cout << tlinkedlist;
  std::cout << tlinkedlist.GetItem(2) << std::endl;
  tlinkedlist.Remove(2);
  std::cout << tlinkedlist;
  tlinkedlist.Remove(2);
  std::cout << tlinkedlist;
  Iterator:
  for (auto i : tlinkedlist) {
    std::cout << *i << std::endl;
  }
  tlinkedlist.Clear();
  return 0;
}
```