

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 Каширин Кирилл Дмитриевич, группа М8О-208Б-20

Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 7: Связанный список. 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:

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 содержать 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.

Нельзя использовать:

1. Стандартные контейнеры `std`.

2. Шаблоны (template).
3. Различные варианты умных указателей (shared_ptr, weak_ptr).

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.cpp: реализация класса элемента связанного списка
10. tlinkedlist.cpp: реализация класса связанного списка

Дневник отладки

Возникли проблемы с добавлением элемента на заданную позицию. После тестирования проблему была устранена пересмотром присваивание указателю другого адреса.

Недочёты

Недочетов не заметил

Выводы

В данной лабораторной работе я создал класс-контейнер связанный список, функции для работы с ним. Освоил динамическое выделение памяти в C++ с помощью операций new и delete.

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

```
#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

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

```
#include "point.h"
```

```
class Hexagon : public Figure {
public:
    Hexagon();
    Hexagon(std::istream &is);
    Hexagon(Point a, Point b, Point c, Point d, Point e, Point f);
    Hexagon(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

```
#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(Hexagon& other):Hexagon(other.a,other.b,other.c,other.d,other.e,other.f)
{
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

```

#include <iostream>
#include "hexagon.h"

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

```


hlist_item.cpp

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

HListItem::HListItem(const Hexagon &hexagon) {
    this->hexagon = hexagon;
    this->next = nullptr;
}

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

HListItem::~HListItem() {}
```

tlinkedlist.h

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

class TLinkedList {
public:
    TLinkedList();
    int size_of_list;
    size_t Length();
    bool Empty();
    Hexagon& First();
    Hexagon& Last();
    TLinkedList(const TLinkedList& other);
    Hexagon& GetItem(size_t idx);
    void InsertFirst(const Hexagon &&hexagon);
    void InsertLast(const Hexagon &&hexagon);
    void RemoveLast();
    void RemoveFirst();
    void Insert(const Hexagon &&hexagon, 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 // HList_H
```

tlinkedlist.cpp

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

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

Hexagon& TLinkedList::GetItem(size_t idx){
    int k = 0;
    HListItem* obj = front;
    while (k != idx){
        k++;
        obj = obj->next;
    }
    return obj->hexagon;
}

Hexagon& TLinkedList::First() {
    return front->hexagon;
}

Hexagon& TLinkedList::Last() {
    return back->hexagon;
}
```

```

void TLinkedList::InsertLast(const Hexagon &&hexagon) {
    HListItem* obj = new HListItem(hexagon);
    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 << "Hexagon does not pop_back, because the Hexagon 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 Hexagon &&hexagon) {
    HListItem* obj = new HListItem(hexagon);
    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 << "Hexagon does not pop_front, because the Hexagon List is empty" << std::endl;
    } else {
        HListItem* del = front;
        front = del->next;
        delete del;
        size_of_list--;
    }
}

void TLinkedList::Insert(const Hexagon &hexagon, 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(hexagon);
        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" << 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;
        //  std::cout << "HListItem deleted" << std::endl;
    }
    size_of_list = 0;
    HListItem* front;
    HListItem* back;
}

std::ostream& operator<<(std::ostream& os, TLinkedList& 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;
    }
}

```

```

HListItem* obj = hl.front;
while(obj != nullptr) {
    if (obj->next != nullptr) {
        os << obj->hexagon << " " << "," << " ";
        obj = obj->next;
    } else {
        os << obj->hexagon;
        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 << "Hexagon List deleted" << std::endl;
    }
}

```

main.cpp

```

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

int main() {
    TLinkedList tlinkedlist;
    tlinkedlist.Empty();
    tlinkedlist.InsertLast(Hexagon(Point(1,2),Point(2,3),Point(3,4),Point(5,6),
    Point(7,8),Point(9,10)));
    tlinkedlist.InsertLast(Hexagon(Point(11,12),Point(12,13),Point(13,14),Point(14,15),
    Point(15,16),Point(16,17)));
    tlinkedlist.InsertLast(Hexagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),
    Point(21,22),Point(23,24)));
}

```

```

tlinkedlist.InsertLast(Hexagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),
Point(21,22),Point(23,24))));
std::cout << tlinkedlist;
tlinkedlist.RemoveLast();
std::cout << tlinkedlist.Length() << std::endl;
tlinkedlist.RemoveFirst();
tlinkedlist.InsertFirst(Hexagon(Point(2,3),Point(3,4),Point(4,5),Point(5,6),
Point(6,7),Point(7,8))));
tlinkedlist.Insert(Hexagon(Point(1,1),Point(2,3),Point(3,4),Point(5,6),
Point(7,8),Point(9,16)),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;
}

```