# **Лабораторная работа № 05**

**Студент**: Сазонов Вадим Кириллович , м8О-201Б-20

# **Тема: Основы работы с коллекциями: итераторы**

**Цель:**

* Изучение основ работы с коллекциями, знакомство с шаблоном проектирования «Итератор»;

## **Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом.
2. Получить у преподавателя вариант задания.
3. Реализовать задание своего варианта в соответствии с поставленными требованиями.
4. Подготовить тестовые наборы данных.
5. Создать репозиторий на GitHub.
6. Отправить файлы лабораторной работы в репозиторий.
7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

## **Требования к программе**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться **oop\_exercise\_05** (в случае использования Windows **oop\_exercise\_05.exe).**

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть<https://github.com/login/oop_exercise_05>.

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

· main.cpp //файл с заданием работы

· CMakeLists.txt // файл с конфигураций CMake

· report.doc // отчет о лабораторной работе

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения (правильные многоугольники), т.е. равносторонними (кроме трапеции и прямоугольника). Для хранения координат фигур необходимо использовать шаблон std::pair.

Например:

template <class T>

struct Square{

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

};

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr). Опционально использование std::unique\_ptr;

2. В качестве параметра шаблона коллекция должна принимать тип данных - фигуры;

3. Реализовать forward\_iterator по коллекции;

4. Коллекция должны возвращать итераторы begin() и end();

5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);

6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);

7. При выполнении недопустимых операций (например выход за границы коллекции или удаление несуществующего элемента) необходимо генерировать исключения;

8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)

9. Коллекция должна содержать метод доступа:

* стек – pop, push, top;
* очередь – pop, push, top;
* список, Динамический массив – доступ к элементу по оператору [].

10. Реализовать программу, которая:

* позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;
* позволяет удалять элемент из коллекции по номеру элемента;
* выводит на экран введенные фигуры c помощью std::for\_each;
* выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

Вариант 11:

|  |  |
| --- | --- |
| Прямоугольник | Список |

**Код программы:**

**Main.cpp**

#include <iostream>

#include <memory>

#include "rectangle.h"

#include "mylist.h"

#include <algorithm>

void print\_rect\_and\_area(std::shared\_ptr<Rectangle> rect){

std::cout << \*rect << "Area is equal " << rect->Area() << std::endl;

}

int main(void)

{

TLinkedList<Rectangle> list;

std::shared\_ptr<Rectangle> ptr = std::shared\_ptr<Rectangle>(new Rectangle(1,2,3,4));

list.InsertLast(ptr);

ptr = std::shared\_ptr<Rectangle>(new Rectangle(2,3,5,6));

list.InsertLast(ptr);

ptr = std::shared\_ptr<Rectangle>(new Rectangle(1,0,100,6));

list.Insert(ptr,0);

std::for\_each (list.begin(),list.end(), print\_rect\_and\_area);

std::cout<<std::endl;

std::cout<< \*list[2] << std::endl;

list.Remove(1);

std::for\_each (list.begin(),list.end(), print\_rect\_and\_area);

return 0;

}

**Iter.h**

#pragma once

#include <memory>

using namespace std;

template<typename node, typename T>

class TIterator {

private:

shared\_ptr<node> ptr;

public:

TIterator(shared\_ptr<node> ptr)

{

this->ptr = ptr;

}

shared\_ptr<T> operator\*()

{

return ptr->getValuePtr();

}

shared\_ptr<T> operator->()

{

return ptr->getValuePtr();

}

TIterator<node, T> operator++()

{

return ptr = ptr->getNextItemPtr();

}

TIterator<node, T> operator++(int)

{

TIterator iter(\*this);

++(\*this);

return iter;

}

bool operator==(TIterator<node, T> const& other)

{

return ptr == other.ptr;

}

bool operator!=(TIterator<node, T> const& other)

{

return !(\*this == other);

}

};

Mlistelem.h

#pragma once

#include <memory>

using namespace std;

template <class T>

class TLinkedListItem

{

private:

shared\_ptr<T> pValue;

shared\_ptr<TLinkedListItem<T>> pNextItem;

public:

TLinkedListItem() {}

TLinkedListItem(shared\_ptr<T> pValue, shared\_ptr<TLinkedListItem<T>> pNextItem);

virtual ~TLinkedListItem();

shared\_ptr<T> getValuePtr();

shared\_ptr<TLinkedListItem<T>> getNextItemPtr();

void setNextItemPtr(shared\_ptr<TLinkedListItem<T>> pNextItem);

};

Mlistelem.cpp

#include <memory>

#include "rectangle.h"

#include "mlistelem.h"

using namespace std;

template<class T>

inline TLinkedListItem<T>::TLinkedListItem(shared\_ptr<T> pValue, shared\_ptr<TLinkedListItem<T>> pNextItem)

{

this->pValue = pValue;

this->pNextItem = pNextItem;

}

template<class T>

TLinkedListItem<T>::~TLinkedListItem()

{

}

template<class T>

shared\_ptr<T> TLinkedListItem<T>::getValuePtr()

{

return pValue;

}

template<class T>

shared\_ptr<TLinkedListItem<T>> TLinkedListItem<T>::getNextItemPtr()

{

return pNextItem;

}

template<class T>

void TLinkedListItem<T>::setNextItemPtr(shared\_ptr<TLinkedListItem<T>> pNextItem)

{

this->pNextItem = pNextItem;

}

template class TLinkedListItem<Rectangle>;

// template ostream& operator<<(ostream& os, const TLinkedListItem<Rectangle>& item);

Mylist.h

#pragma once

#include <iostream>

#include <memory>

#include "mlistelem.h"

#include "iter.h"

using namespace std;

template <class T>

class TLinkedList

{

private:

size\_t length;

shared\_ptr<TLinkedListItem<T>> pFirstItem;

shared\_ptr<TLinkedListItem<T>> pLastItem;

public:

TLinkedList();

TLinkedList(const TLinkedList& other);

virtual ~TLinkedList();

shared\_ptr<T> First();

shared\_ptr<T> Last();

shared\_ptr<T> GetItem(size\_t position);

void InsertFirst(shared\_ptr<T> pValue);

void InsertLast(shared\_ptr<T> pValue);

void Insert(shared\_ptr<T> pValue, size\_t position);

void RemoveFirst();

void RemoveLast();

void Remove(size\_t position);

void Clear();

bool Empty();

size\_t Length();

template <class T2>

friend std::ostream& operator<<(std::ostream& os, const TLinkedList<T2>& list);

std::shared\_ptr<T> operator[](int pos){

shared\_ptr<TLinkedListItem<T>> curr = pFirstItem;

int cur\_pos = 0;

while(curr){

if (cur\_pos != pos){

curr = curr->getNextItemPtr();

cur\_pos++;

continue;

}

return curr->getValuePtr();

}

throw runtime\_error("Out of range");

};

TIterator<TLinkedListItem<T>, T> begin();

TIterator<TLinkedListItem<T>, T> end();

};

Mylist.cpp

#include <iostream>

#include <memory>

#include <stdexcept>

#include "mylist.h"

#include "rectangle.h"

template<typename T>

TLinkedList<T>::TLinkedList()

{

pFirstItem = nullptr;

pLastItem = nullptr;

length = 0;

}

template<typename T>

TLinkedList<T>::TLinkedList(const TLinkedList<T>& other)

{

pFirstItem = nullptr;

pLastItem = nullptr;

length = 0;

shared\_ptr<TLinkedListItem<T>> pCurrentItem = other.pFirstItem;

while (pCurrentItem != nullptr)

{

InsertLast(pCurrentItem->getValuePtr());

pCurrentItem = pCurrentItem->getNextItemPtr();

}

}

template<typename T>

shared\_ptr<T> TLinkedList<T>::First()

{

if (Empty())

throw runtime\_error("Cannon get the item from empty list");

return pFirstItem->getValuePtr();

}

template<typename T>

shared\_ptr<T> TLinkedList<T>::Last()

{

if (Empty())

throw runtime\_error("Cannon get the item from empty list");

return pLastItem->getValuePtr();

}

template<typename T>

void TLinkedList<T>::InsertFirst(shared\_ptr<T> pValue)

{

shared\_ptr<TLinkedListItem<T>> pNewItem(new TLinkedListItem<T>(pValue, pFirstItem));

pFirstItem = pNewItem;

if (Empty())

pLastItem = pNewItem;

length++;

}

template<typename T>

void TLinkedList<T>::InsertLast(shared\_ptr<T> pValue)

{

shared\_ptr<TLinkedListItem<T>> pNewItem(new TLinkedListItem<T>(pValue, nullptr));

if (pLastItem != nullptr)

pLastItem->setNextItemPtr(pNewItem);

pLastItem = pNewItem;

if (Empty())

pFirstItem = pNewItem;

length++;

}

template<typename T>

void TLinkedList<T>::Insert(shared\_ptr<T> pValue, size\_t position)

{

if (position == 0)

{

InsertFirst(pValue);

return;

}

else if (position == length)

{

InsertLast(pValue);

return;

}

else if (position > length)

throw runtime\_error("Specified poition is out of range");

int i = 0;

shared\_ptr<TLinkedListItem<T>> pCurrentItem = pFirstItem;

shared\_ptr<TLinkedListItem<T>> pPreviousItem = nullptr;

while (pCurrentItem != nullptr)

{

if (i == position)

break;

pPreviousItem = pCurrentItem;

pCurrentItem = pCurrentItem->getNextItemPtr();

i++;

}

shared\_ptr<TLinkedListItem<T>> pNewItem(new TLinkedListItem<T>(pValue, pCurrentItem));

pPreviousItem->setNextItemPtr(pNewItem);

length++;

}

template<typename T>

void TLinkedList<T>::RemoveFirst()

{

if (Empty())

throw runtime\_error("Cannon remove the item from empty list");

shared\_ptr<TLinkedListItem<T>> pNextItem = pFirstItem->getNextItemPtr();

pFirstItem = pNextItem;

length--;

if (Empty())

pLastItem = nullptr;

}

template<typename T>

void TLinkedList<T>::RemoveLast()

{

if (Empty())

throw runtime\_error("Cannon remove the item from empty list");

shared\_ptr<TLinkedListItem<T>> pCurrentItem = pFirstItem;

shared\_ptr<TLinkedListItem<T>> pPreviousItem = nullptr;

while (pCurrentItem != nullptr)

{

if (pCurrentItem == pLastItem)

break;

pPreviousItem = pCurrentItem;

pCurrentItem = pCurrentItem->getNextItemPtr();

}

if (pPreviousItem != nullptr)

pPreviousItem->setNextItemPtr(nullptr);

pLastItem = pPreviousItem;

length--;

if (Empty())

pFirstItem = nullptr;

}

template<typename T>

void TLinkedList<T>::Remove(size\_t position)

{

if (Empty())

throw runtime\_error("Cannon remove the item from empty list");

if (position == 0)

{

RemoveFirst();

return;

}

else if (position == length - 1)

{

RemoveLast();

return;

}

else if (position >= length)

throw runtime\_error("Specified poition is out of range");

int i = 0;

shared\_ptr<TLinkedListItem<T>> pCurrentItem = pFirstItem;

shared\_ptr<TLinkedListItem<T>> pPreviousItem = nullptr;

while (pCurrentItem != nullptr)

{

if (i == position)

break;

pPreviousItem = pCurrentItem;

pCurrentItem = pCurrentItem->getNextItemPtr();

i++;

}

pPreviousItem->setNextItemPtr(pCurrentItem->getNextItemPtr());

length--;

}

template<typename T>

shared\_ptr<T> TLinkedList<T>::GetItem(size\_t position)

{

if (Empty())

throw runtime\_error("Cannon get the item from empty list");

if (position >= length)

throw runtime\_error("Specified position is out of range");

int i = 0;

shared\_ptr<TLinkedListItem<T>> pCurrentItem = pFirstItem;

while (pCurrentItem != nullptr)

{

if (i == position)

return pCurrentItem->getValuePtr();

pCurrentItem = pCurrentItem->getNextItemPtr();

i++;

}

throw runtime\_error("Something went wrong");

}

template<typename T>

bool TLinkedList<T>::Empty()

{

return length == 0;

}

template<typename T>

size\_t TLinkedList<T>::Length()

{

return length;

}

template<typename T>

void TLinkedList<T>::Clear()

{

shared\_ptr<TLinkedListItem<T>> pCurrentItem = pFirstItem;

while (pCurrentItem != nullptr)

{

shared\_ptr<TLinkedListItem<T>> pNextItem = pCurrentItem->getNextItemPtr();

pCurrentItem = pNextItem;

}

pFirstItem = nullptr;

pLastItem = nullptr;

length = 0;

}

template<typename T>

TLinkedList<T>::~TLinkedList()

{

Clear();

}

template<typename T2>

ostream& operator<<(ostream& os, const TLinkedList<T2>& list)

{

shared\_ptr<TLinkedListItem<T2>> pCurrentItem = list.pFirstItem;

while (pCurrentItem != nullptr)

{

os << pCurrentItem->getValuePtr()->Area();

if (pCurrentItem != list.pLastItem)

os << " -> ";

pCurrentItem = pCurrentItem->getNextItemPtr();

}

return os;

}

template<typename T>

TIterator<TLinkedListItem<T>, T> TLinkedList<T>::begin()

{

return TIterator<TLinkedListItem<T>, T>(pFirstItem);

}

template<typename T>

TIterator<TLinkedListItem<T>, T> TLinkedList<T>::end()

{

return TIterator<TLinkedListItem<T>, T>(nullptr);

}

template class TLinkedList<Rectangle>;

template ostream& operator<<(ostream& os, const TLinkedList<Rectangle>& list);

Rectangle.h

#pragma once

#include <iostream>

class Rectangle

{

public:

Rectangle(float x1, float y1, float first\_side, float second\_side)

{

x = x1;

y = y1;

this->first\_side = first\_side;

this->second\_side = second\_side;

}

float Area(){

return first\_side\*second\_side;

}

friend std::ostream& operator<<(std::ostream& os, Rectangle& rect){

os << "Rectangle: left lower corner coords " << "(" << rect.x << "," << rect.y << ")." << "First side is " << rect.first\_side << " and second side is " << rect.second\_side << std::endl;

}

private:

float x;

float y;

float first\_side;

float second\_side;

};

**Результат работы программы:**

Rectangle: left lower corner coords (1,0).First side is 100 and second side is 6

Area is equal 600

Rectangle: left lower corner coords (1,2).First side is 3 and second side is 4

Area is equal 12

Rectangle: left lower corner coords (2,3).First side is 5 and second side is 6

Area is equal 30

Rectangle: left lower corner coords (2,3).First side is 5 and second side is 6

Rectangle: left lower corner coords (1,0).First side is 100 and second side is 6

Area is equal 600

Rectangle: left lower corner coords (2,3).First side is 5 and second side is 6

Area is equal 30