**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 5**

Тема: Основы работы с коллекциями: итераторы

Студент: Волков

Группа: М80-207Б-19

Преподаватель: Чернышев Л. Н.

Дата:

Оценка:

1. **Постановка задачи**

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, 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 аз границы коллекции или удаление не существующего 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 3: Прямоугольник — стек*

1. **Описание программы**

Задание разбито на несколько файлов: *main.cpp, Stack.h, Rectangle.h.*

В *main.cpp* находится набор тестов, нужных для тестирования программы.

В *Rectangle.h* находится класс rectangle, в котором есть координаты фигуры и заданы стороны. Также есть метод, вычисляющий площадь фигуры

В *Stack.h* написана коллекция, с которой я работаю. Для начала делается структура *Node,* в которой содержатся данные элемента структуры и ссылка на предыдущий элемент стека, написанный при помощи умных указателей. Далее перегружается операторы для того, чтобы была совместимость со стандартными алгоритмами. Далее идет описание самого стека. В нем данные хранятся при помощи указателя shared\_ptr<Node<T>>. Это сделано для того, чтобы можно было хранить в стеке данные мне фигуры. Далее идет описание стандартных методов стека. Чего-то необычного там нет.

Напоследок идет класс итератора, который написан внутри Stack. Это было сделано для удобства обращения к итератору. Далее идут using, задающие характеристики нашего итератора. Это надо, опять же, для совместимости со стандартными алгоритмами. В нем же идет перегрузка операторов, нужных для совместимости

1. **Наборы и результаты выполнения тестов**

*Тест 1:*

Input count of rectangles  
2  
Input rectangles.  
To end entering press the EOF button.  
First, enter the X and Y - point of vertex  
Second, enter the sides of rectangle - A and B  
0 0  
2 2  
Input rectangles.  
To end entering press the EOF button.  
First, enter the X and Y - point of vertex  
Second, enter the sides of rectangle - A and B  
1 1  
2 3  
  
This is all rectangles  
Coordinates: {(1, 1); (1, 3); (4, 3); (4, 1)}  
  
Coordinates: {(0, 0); (0, 2); (2, 2); (2, 0)}  
  
2  
  
Enter your index to insert  
3  
Input rectangle.  
First, enter the X and Y - point of vertex  
Second, enter the sides of rectangle - A and B  
3 4  
1 1  
  
This is all rectangles  
Coordinates: {(1, 1); (1, 3); (4, 3); (4, 1)}  
  
Coordinates: {(0, 0); (0, 2); (2, 2); (2, 0)}  
  
Coordinates: {(3, 4); (3, 5); (4, 5); (4, 4)}  
  
  
Enter your index to erase  
1  
  
This is all rectangles  
Coordinates: {(0, 0); (0, 2); (2, 2); (2, 0)}  
  
Coordinates: {(3, 4); (3, 5); (4, 5); (4, 4)}  
  
  
Enter your minimal area  
1  
Number of rectangles with area greater or equal, than 1: 2  
  
Process finished with exit code 0

*Тест 2:*

Input count of rectangles  
2  
Input rectangles.  
To end entering press the EOF button.  
First, enter the X and Y - point of vertex  
Second, enter the sides of rectangle - A and B  
0 0  
2 2  
Input rectangles.  
To end entering press the EOF button.  
First, enter the X and Y - point of vertex  
Second, enter the sides of rectangle - A and B  
1 1  
2 3  
  
This is all rectangles  
Coordinates: {(1, 1); (1, 3); (4, 3); (4, 1)}  
  
Coordinates: {(0, 0); (0, 2); (2, 2); (2, 0)}  
  
  
Enter your index to insert  
5  
terminate called after throwing an instance of 'std::invalid\_argument'  
 what(): You entered wrong index  
  
Process finished with exit code 3

1. **Листинг программы**

**main.cpp**

/\*Волков Мавтей Андреевич  
 \*  
 \* 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, 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 аз границы коллекции или удаление не существующего элемента)  
 необходимо генерировать исключения;  
8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)  
9. Коллекция должна содержать метод доступа:  
o Стек – pop, push, top;  
10. Реализовать программу, которая:  
o Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;  
o Позволяет удалять элемент из коллекции по номеру элемента;  
o Выводит на экран введенные фигуры c помощью std::for\_each;  
o Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if);  
 Вариант 3: прямоугольни - стек  
\*/  
#include "Stack.h"  
#include <algorithm>  
  
int main() {  
 Stack<Rectangle<int>> st;  
 int x, y, a,b;  
 size\_t n;  
 std::cout << "Input count of rectangles" << std::endl;  
 std::cin >> n;  
  
 for (int i = 0; i < n; ++i){  
 std::cout << "Input rectangles.\nTo end entering press the EOF button.\n"  
 "First, enter the X and Y - point of vertex\n"  
 "Second, enter the sides of rectangle - A and B" << std::endl;  
 std::cin >> x >> y;  
 std::cin >> a >> b;  
 st.push(Rectangle<int>(std::pair<int,int> {x,y}, a, b));  
 }  
 auto printStack = [](const Node<Rectangle<int>>& obj){  
 std::cout << obj << std::endl;  
 };  
 std::cout << "\nThis is all rectangles" << std::endl;  
 std::for\_each(st.begin(), st.end(), printStack);  
  
 std::cout << "\nEnter your index to insert" << std::endl;  
 std::cin >> n;  
 try{  
 if(st.getSize()+1 < n){  
 throw std::invalid\_argument("You entered wrong index");  
 }  
 else{  
 auto it = st.begin();  
 while(--n){  
 ++it;  
 }  
 std::cout << "Input rectangle.\n"  
 "First, enter the X and Y - point of vertex\n"  
 "Second, enter the sides of rectangle - A and B" << std::endl;  
 std::cin >> x >> y >> a >> b;  
 st.insert(it, Rectangle<int>(std::pair<int,int>{x,y},a,b));  
 }  
 }  
 catch (std::invalid\_argument& e) {  
 std::terminate();  
 }  
  
 std::cout << "\nThis is all rectangles" << std::endl;  
 std::for\_each(st.begin(), st.end(), printStack);  
  
 std::cout << "\nEnter your index to erase" << std::endl;  
 std::cin >> n;  
 try{  
 auto it = st.begin();  
 while(--n){  
 ++it;  
 }  
 st.erase(it);  
 }  
 catch (std::runtime\_error& e) {  
 std::cout << e.what() << std::endl;  
 }  
  
 std::cout << "\nThis is all rectangles" << std::endl;  
 std::for\_each(st.begin(), st.end(), printStack);  
  
 int minArea;  
 std::cout << "\nEnter your minimal area" << std::endl;  
 std::cin >> minArea;  
 auto checkMinArea = [&minArea](const Node<Rectangle<int>>& rec){  
 return rec.data.area() >= minArea;  
 };  
 std::cout << "Number of rectangles with area greater or equal, than " << minArea << ": "  
 << std::count\_if(st.begin(), st.end(), checkMinArea) << std::endl;  
 return 0;  
}

**Stack.h**

#ifndef OOP\_EXERCISE\_05\_STACK\_H  
#define OOP\_EXERCISE\_05\_STACK\_H  
  
#include "Rectangle.h"  
  
template<class T>  
struct Node;  
  
template<class T>  
void operator++(std::shared\_ptr<Node<T>> &curNode) {  
 if (curNode) {  
 curNode = curNode->prev;  
 }  
 else {  
 throw std::out\_of\_range("Iterator has already in nullptr");  
 }  
}  
  
template<class T>  
bool operator!=(const Node<T> &lhs, const Node<T> &rhs) {  
 return &lhs != &rhs;  
}  
  
template<class T>  
bool operator==(const Node<T> &lhs, const Node<T> &rhs) {  
 return &lhs.data == &rhs.data;  
}  
  
template<class T>  
std::ostream &operator<<(std::ostream &cout, const Node<T> &node) {  
 cout << node.data;  
 return cout;  
}  
  
template<class T>  
struct Node {  
 T data;  
 std::shared\_ptr<Node> prev;  
  
  
 Node() : prev(nullptr) {  
 }  
  
 Node(const T &val) : data(val) {  
 }  
  
 friend void operator++<>(std::shared\_ptr<Node<T>> &n);  
  
 friend bool operator!=<>(const Node<T> &lhs, const Node<T> &rhs);  
  
 friend bool operator==<>(const Node<T> &lhs, const Node<T> &rhs);  
  
 friend std::ostream &operator<< <>(std::ostream &cout, const Node<T> &obj);  
};  
  
  
template<class T>  
class Stack {  
private:  
 size\_t size = 0;  
 std::shared\_ptr<Node<T>> head;  
public:  
  
 class iterator {  
 private:  
 std::shared\_ptr<Node<T>> iter;  
  
 public:  
 using iterator\_category = std::forward\_iterator\_tag;  
 using difference\_type = std::ptrdiff\_t;  
 using value\_type = T;  
 using pointer = T \*;  
 using reference = T &;  
  
 iterator() : iter(nullptr) {  
 }  
  
 explicit iterator(const std::shared\_ptr<Node<T>> &anotherIter) : iter(anotherIter) {  
 }  
  
 bool isNull() {  
 return iter == nullptr;  
 }  
  
 friend void operator++(iterator &it) {  
 ++it.iter;  
 }  
  
 friend bool operator!=(const Stack<T>::iterator &lhs, const Stack<T>::iterator &rhs) {  
 return lhs.iter != rhs.iter;  
 }  
  
 friend std::ostream &operator<<(std::ostream &out, const iterator &it) {  
 out << \*it.iter;  
 return out;  
 }  
  
 Node<T> &operator\*() {  
 return \*iter;  
 }  
 };  
  
 Stack() noexcept: head(nullptr) {  
 }  
  
 void pop() {  
 if (head) {  
 head = head->prev;  
 --size;  
 }  
 else {  
 throw std::runtime\_error("Stack is empty");  
 }  
 }  
  
 void push(const T &val) {  
 Node<T> \*newNode = new Node<T>(val);  
 std::shared\_ptr<Node<T>> newPtr{newNode};  
 newPtr->prev = head;  
 head = newPtr;  
 ++size;  
 }  
  
 T top() {  
 if (head) {  
 return head->data;  
 }  
 else {  
 throw std::runtime\_error("Nullptr in head");  
 }  
 }  
  
 void insert(Stack<T>::iterator &it, const T &elem) {  
 std::unique\_ptr<Node<T>> newNode{new Node<T>(elem)};  
 std::shared\_ptr<Node<T>> newPtr = std::move(newNode);  
 std::shared\_ptr<Node<T>> prevPtr = head;  
  
 if (prevPtr) {  
 if (!it.isNull()) {  
 while (\*prevPtr->prev != \*it) {  
 ++prevPtr;  
 }  
 }  
 else {  
 while (prevPtr->prev != nullptr) {  
 ++prevPtr;  
 }  
 }  
 newPtr->prev = prevPtr->prev;  
 prevPtr->prev = newPtr;  
 }  
 else {  
 head = newPtr;  
 }  
 ++size;  
 }  
  
 void erase(iterator &it) {  
 if (it.isNull()) {  
 throw std::runtime\_error("Iterator is in nullptr state");  
 }  
 else {  
 if (\*it == \*head) {  
 head = head->prev;  
 }  
 else {  
 std::shared\_ptr<Node<T>> prevPtr = head;  
 while (\*prevPtr->prev != \*it) {  
 ++prevPtr;  
 }  
 prevPtr->prev = prevPtr->prev->prev;  
 }  
 --size;  
 }  
  
 }  
  
 iterator begin() {  
 return iterator(head);  
 }  
  
 iterator end() {  
 return iterator(nullptr);  
 }  
  
 size\_t getSize() {  
 return size;  
 }  
};  
  
#endif //OOP\_EXERCISE\_05\_STACK\_H

**Rectangle.h**

#ifndef OOP\_EXERCISE\_05\_RECTANGLE\_H  
#define OOP\_EXERCISE\_05\_RECTANGLE\_H  
  
#include <iostream>  
#include <memory>  
#include <exception>  
  
template<class T>  
struct Rectangle{  
 using vertex\_t = std::pair<T,T>;  
 vertex\_t point;  
 T a;  
 T b;  
  
 Rectangle(const vertex\_t& \_point, const T& \_a, const T& \_b ):a(\_a), point(\_point),b(\_b){}  
  
 T area() const {  
 return a\*b;  
 }  
};  
  
template<class T>  
std::ostream& operator << (std::ostream& cout, const std::pair<T,T>& val){  
 cout << "(" << val.first << ", " << val.second << ")";  
 return cout;  
}  
  
template<class T>  
std::ostream& operator << (std::ostream& cout, const Rectangle<T>& obj){  
 cout << "Coordinates: {"  
 << std::pair<T,T>(obj.point.first,obj.point.second)<< "; "  
 << std::pair<T,T>(obj.point.first,obj.point.second + obj.a)<< "; "  
 << std::pair<T,T>(obj.point.first + obj.b,obj.point.second + obj.a)<< "; "  
 << std::pair<T,T>(obj.point.first + obj.b,obj.point.second)<< "}\n ";  
  
 return cout;  
}  
  
#endif //OOP\_EXERCISE\_05\_RECTANGLE\_H

1. **Вывод**

В ходе выполнения работы я узнал, как реализовать итераторы на языке С++, чтобы они были совместимы со стандартными функциями. Узнал про умные указатели *std::unique\_ptr, std::shared\_ptr* и в чем их разница. Изучил работу *std::for\_each* и *std::count\_if* и реализовал из взаимодействие с моей коллекцией.

**Список литературы**

1. *std::shared\_ptr — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/memory/shared_ptr> (дата обращения 15.11.2020).

1. *Делаем свой итератор / Хабр — Habr*

URL: <https://en.cppreference.com/w/cpp/memory/shared_ptr> (дата обращения 15.11.2020).

1. *std::for\_each — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/algorithm/for_each> (дата обращения 15.11.2020).

1. *std::count\_if — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/algorithm/for_each> (дата обращения 15.11.2020).