**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: Бирюков В. В.
Группа: 80-207
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

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 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 с помощью std::for_each.
    - выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count_if).

Вариант 17: треугольник, очередь.

2. Описание программы

Шаблонный класс Triangle хранит треугольник как координаты его левой вершины (в виде пары) и длину стороны.

Шаблонный класс Queue представляет собой коллекцию типа “очередь”. Очередь хранит элементы в виде линейного двусвязного списка, для удобства вставки и удаления за константное время. Очередь поддерживает следующие методы доступа: top – просмотр верхнего элемента, push – добавление элемента в начало очереди, pop – удаление элемента с конца очереди, insert – вставка элемента по итератору (на

позицию до итератора), erase – удаление элемента по итератору.

Итератор очереди хранит указатель на соответствующий узел очереди и содержит минимально необходимый набор методов для forward_iterator (перегруженные разыменование, инкремент и неравенство)

3. Набор тестов

Программа принимает на вход команды, вводимые пользователем. Реализованы следующие команды: добавление элемента в начало очереди (`a TRIANGLE`), удаление элемента из конца очереди (`d`), просмотр элемента в конце очереди (`t`), вставка элемента по индексу (`i INDEX TRIANGLE`), удаление элемента по индексу (`e INDEX`), печать элементов очереди (`p`), подсчет количество элементов, площадь которых меньше заданной (`f SQUARE`).

Треугольник вводится в виде трех чисел: координат вершины и длины стороны.

Тест 1:

```
> a 1 1 1
> a 2 2 2
> a 3 3 3
> p
(3, 3) (4.5, 5.59808) (6, 3)
(2, 2) (3, 3.73205) (4, 2)
(1, 1) (1.5, 1.86603) (2, 1)
> d
> d
> p
(3, 3) (4.5, 5.59808) (6, 3)
> a 1 1 1
> p
(1, 1) (1.5, 1.86603) (2, 1)
(3, 3) (4.5, 5.59808) (6, 3)
> i 1 2 2 2
> p
(1, 1) (1.5, 1.86603) (2, 1)
(2, 2) (3, 3.73205) (4, 2)
(3, 3) (4.5, 5.59808) (6, 3)
> e 0
> p
(2, 2) (3, 3.73205) (4, 2)
(3, 3) (4.5, 5.59808) (6, 3)
> i 2 4 4 4
> p
(2, 2) (3, 3.73205) (4, 2)
```

```
(3, 3) (4.5, 5.59808) (6, 3)
(4, 4) (6, 7.4641) (8, 4)
> t
(4, 4) (6, 7.4641) (8, 4)
> f 10
3
> f 5
2
> a 6 6 6
> p
(6, 6) (9, 11.1962) (12, 6)
(2, 2) (3, 3.73205) (4, 2)
(3, 3) (4.5, 5.59808) (6, 3)
(4, 4) (6, 7.4641) (8, 4)
> f 10
3
> q
```

Тест 2 (исключительные ситуации):

```
> p
> t
Error: Queue is empty
> d
Error: Queue is empty
> a 1 6 0
Error: Invalid triangle parameters
> a 1 1 1
> a 2 2 2
> a 3 3 3
> p
(3, 3) (4.5, 5.59808) (6, 3)
(2, 2) (3, 3.73205) (4, 2)
(1, 1) (1.5, 1.86603) (2, 1)
> i -1 9 9 9
Error: Out of bounds
> e 5
Error: Out of bounds
> p
(3, 3) (4.5, 5.59808) (6, 3)
(2, 2) (3, 3.73205) (4, 2)
(1, 1) (1.5, 1.86603) (2, 1)
> q
```

## 4. Листинг программы

```
// triangle.hpp
#pragma once

#include <utility>
#include <stdexcept>
```

```cpp
#include <cmath>

template <class T>
class Triangle {
public:
  std::pair<T,T> x;
  T a;

  Triangle() : x(), a() {}
  Triangle(T x1, T x2, T a) : x(x1,x2), a(a) {
    if (a <= 0) {
      throw std::invalid_argument("Error: Invalid triangle parameters");
    }
  }

  double square() const {
    return sqrt(3) / 4 * a * a;
  }
  template <class A>
  friend std::ostream& operator<<(std::ostream&, const Triangle<A>&);
  template <class A>
  friend std::istream& operator>>(std::istream&, const Triangle<A>&);
};

template <class T>
std::ostream& operator<<(std::ostream& os, const Triangle<T>& tr) {
  os << "(" << tr.x.first << ", " << tr.x.second << ") "
     << "(" << tr.x.first + 1.0 / 2 * tr.a << ", "
     << tr.x.second + sqrt(3) / 2 * tr.a << ") "
     << "(" << tr.x.first + tr.a << ", " << tr.x.second << ")";
  return os;
}

template <class T>
std::istream& operator>>(std::istream& is, Triangle<T>& tr) {
  is >> tr.x.first >> tr.x.second >> tr.a;
  if (tr.a <= 0) {
    throw std::invalid_argument("Error: Invalid triangle parameters");
  }
  return is;
}

// queue.hpp
#pragma once

#include <memory>
#include <stdexcept>

template <class T>
class Queue {
private:
  class Node {
  public:
    std::unique_ptr<T> data;
    std::shared_ptr<Node> next;
    std::weak_ptr<Node> prev;
```

```
    Node(const T& value) : data(new T(value)), next(), prev() {}
  };

public:
  class Forward_iterator {
  private:
    std::shared_ptr<Node> ptr;

  public:
    using iterator_category = std::forward_iterator_tag;
      using value_type = T;
      using difference_type = size_t;
      using pointer = T*;
      using reference = T&;
    T& operator*() {
      return *(ptr->data);
    }
    Forward_iterator& operator++() {
      if (ptr == nullptr) {
        throw std::runtime_error("Error: Out of bounds");
      }
      ptr = ptr->next;
      return *this;
    }
    bool operator!=(const Forward_iterator& other) {
      return ptr != other.ptr;
    }
    bool operator==(const Forward_iterator& other) {
      return ptr == other.ptr;
    }
    Forward_iterator(std::shared_ptr<Node> ptr) : ptr(ptr){}

    friend class Queue<T>;
  };

private:
  std::shared_ptr<Node> head;
  std::shared_ptr<Node> tail;

public:
  Forward_iterator begin();
  Forward_iterator end();
  void insert(Forward_iterator&, const T&);
  void erase(Forward_iterator&);
  const T& top();
  void pop();
  void push(const T&);

  Queue();
};

template <class T>
Queue<T>::Queue() : tail(new Node(T())) {
  head = tail;
}
```

```
template <class T>
const T& Queue<T>::top() {
  if (tail->prev.lock() == nullptr) {
    throw std::runtime_error("Error: Queue is empty");
  }
  return *(tail->prev.lock()->data);
}

template <class T>
void Queue<T>::pop() {
  if (tail->prev.lock() == nullptr) {
    throw std::runtime_error("Error: Queue is empty");
  }
  tail = tail->prev.lock();
}

template <class T>
void Queue<T>::push(const T &value) {
  std::shared_ptr<Node> node(new Node(value));
  node->next = head;
  head->prev = node;
  head = node;
}

template <class T>
typename Queue<T>::Forward_iterator Queue<T>::begin() {
  return Forward_iterator(head);
}

template <class T>
typename Queue<T>::Forward_iterator Queue<T>::end() {
  return Forward_iterator(tail);
}

template <class T>
void Queue<T>::insert(typename Queue<T>::Forward_iterator& iter,
                      const T& value) {
  if (iter.ptr == nullptr) {
    throw std::runtime_error("Error: Out of bounds");
  }
  if (iter == begin()) {
    push(value);
  } else {
    std::shared_ptr<Node> node(new Node(value));
    node->next = iter.ptr;
    node->prev = iter.ptr->prev;
    iter.ptr->prev.lock()->next = node;
    iter.ptr->prev = node;
  }
}

template <class T>
void Queue<T>::erase(typename Queue<T>::Forward_iterator& iter) {
  if (iter.ptr == nullptr || iter == end()) {
    throw std::runtime_error("Error: Out of bounds");
```

```
  }
  if (iter == begin()) {
    head = head->next;
  } else {
    iter.ptr->prev.lock()->next = iter.ptr->next;
  }
}

// main.cpp
#include <iostream>
#include <algorithm>

#include "queue.hpp"
#include "triangle.hpp"

int main() {
  Queue<Triangle<int>> queue;

  std::cout << "a TRIANGLE - Push" << std::endl
            << "d - Pop" << std::endl
            << "t - Top" << std::endl
            << "i INDEX TRIANGLE - Insert" << std::endl
            << "e INDEX - Erase" << std::endl
            << "p - Print" << std::endl
            << "f SQUARE - Filter" << std::endl
            << "q - Exit" << std::endl;

  char command;
  std::cout << "> ";
  while (std::cin >> command && command != 'q') {
    try {
      if (command == 'a') {
        Triangle<int> tr;
        std::cin >> tr;
        queue.push(tr);

      } else if (command == 'd') {
        queue.pop();

      } else if (command == 't') {
        std::cout << queue.top() << std::endl;

      } else if (command == 'i') {
        int index;
        std::cin >> index;
        Triangle<int> tr;
        std::cin >> tr;
        if (index < 0) {
          throw std::runtime_error("Error: Out of bounds");
        }
        auto iter = queue.begin();
        while (index--) {
          ++iter;
        }
        queue.insert(iter, tr);
```

```
      } else if (command == 'e') {
        int index;
        std::cin >> index;
        if (index < 0) {
          throw std::runtime_error("Error: Out of bounds");
        }
        auto iter = queue.begin();
        while (index--) {
          ++iter;
        }
        queue.erase(iter);

      } else if (command == 'p') {
        std::for_each(queue.begin(), queue.end(),
         [](const Triangle<int>& tr) { std::cout << tr << std::endl; });

      } else if (command == 'f') {
        int square;
        std::cin >> square;
        std::cout << std::count_if(queue.begin(), queue.end(),
                        [&square](const Triangle<int>& tr)
                        { return tr.square() < square; }) << std::endl;
      }
    } catch (const std::exception& e) {
      std::cout << e.what() << std::endl;
    }
    std::cout << "> ";
  }
}
```

5. Выводы

В ходе лабораторной работы я познакомился с созданием коллекций, подходящих для алгоритмов из стандартной библиотеки C++, а также с использованием умных указателей.

Литература


1. Справочник по языку C++ [Электронный ресурс]. URL: https://ru.cppreference.com (дата обращения: 15.11.20).
2. Sample C++/STL custom iterator [Электронный ресурс]. URL: https://gist.github.com/jeetsukumaran/307264 (дата обращения: 13.11.20).