Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «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: Пермяков Никита Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Разработать шаблоны классов. 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 с помощью std::for_each;
    - Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count_if);

Вариант: 19

Контейнер:

- Очередь

Структура:

- Прямоугольник

Цель:

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

2. Описание программы

1. Программа выполняет определённые действия по введённым командам:
    a. 0 — выход из программы;
    b. 1,2,3 - создание фигуры (Квадрат, Прямоугольник, Треугольник соответственно), обновление вершин через ввод и проверка;
    c. 4 - вывод данных вершин;
    d. 5 - вывод площади
2. Функция Print() возвращает содержимое кортежа. Она определена для tuple.
3. Функция Square() вычисляет площадь данной фигуры или совокупности точек в кортеже в зависимости от типа фигуры по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.

3. Набор тестов

Пояснение:

1) На ввод подается число выбора действия из текстового интерфейса
2) При обновлении координат фигуры – выводится номер пары координат фигуры
3) Выводится информация о состоянии tuple фигур, происходит проверка тестами

Test 1

```
1

1

0 0 2 0 2 2 0 2

3

6

9

6

1

2
```

1

3

7

Test 2

1

1

4 0 4 2 0 2 0 0

1

1

3 0 3 5 0 5 0 0

3

4

5

6

12

7

Test 3

1

1

4 0 4 2 0 2 0 0

1

1

3 0 3 5 0 5 0 0

3

1

2

7 0 7 5 0 5 0 0

1

3

5

4

7

4. Результаты выполнения тестов

Test 1

1

1

0 0 Add item back of queue or iterator position

2 0 2 2 command: 1 - <Push> or 2 - <Insert>

0 2

3

6

push --> Input points: 9

6

1

2

print -->

1

3

7[0, 0] [2, 0] [2, 2][0, 2]

area --> Input area: The number figures with area less than given 1

area --> Input area: The number figures with area less than given 0

Delete item from front queue or iterator position

command: 1 - <Pop> or 2 - <Erase>

pop -->print -->

Test 2

command: 1 - <Push> or 2 - <Insert>

3 0 3 5 push --> Input points: 0 5 0 0

Add item back of queue or iterator position

3

4

5

6

command: 1 - <Push> or 2 - <Insert>

push --> Input points: 12

7print -->

[4, 0] [4, 2] [0, 2][0, 0] <- [3, 0] [3, 5] [0, 5][0, 0]

front --> [4, 0] [4, 2] [0, 2][0, 0]

back --> [3, 0] [3, 5] [0, 5][0, 0]

area --> Input area: The number figures with area less than given 1

Test 3

Add item back of queue or iterator position

3 0 3 5 command: 1 - <Push> or 2 - <Insert>

0 5 0 0

push --> Input points: 3

1

2

7 print -->

0 7 5 0 [4, 0] [4, 2] [0, 2][0, 0] <- [3, 0] [3, 5] [0, 5][0, 0]

5 0 0

1

Add item back of queue or iterator position

command: 1 - <Push> or 2 - <Insert>

3

5

4

7insert --> Input points: --> Input index: print -->

[4, 0] [4, 2] [0, 2][0, 0] <- [7, 0] [7, 5] [0, 5][0, 0] <- [3, 0] [3, 5] [0, 5][0, 0]

back --> [3, 0] [3, 5] [0, 5][0, 0]

front --> [4, 0] [4, 2] [0, 2][0, 0]

5. Листинг программы

main.cpp

```
#include <iostream>
#include <utility>
#include <algorithm>
#include <locale>

#include "queue.h"
#include "rectangle.h"
#include "vertex.h"
#include "vector.h"


void printHelp() {
    std::cout << std::endl << "'1'           - Add" << std::endl;
    std::cout << "'2'           - Remove" << std::endl;
    std::cout << "'3'           - Print" << std::endl;
    std::cout << "'4'           - Front" << std::endl;
    std::cout << "'5'           - Back" << std::endl;
    std::cout << "'6'           - Count if" << std::endl;
    std::cout << "'7'           - Exit" << std::endl;
}

int main() {
       Queue<Rectangle<int>> q;
       int cmd, i, cnt;
       double area;

       while (true) {
       // printHelp();
              std::cin >> cmd;
              switch(cmd) {
             case 1: {
                         std::cout << "Add item back of queue or iterator position"
<< std::endl;
                         std::cout << "command: 1 - <Push> or 2 - <Insert>" <<
```

```
std::endl;
                        std::cin >> cmd;
                if(cmd == 1) {
                    Rectangle<int> rec;
                              std::cout << "push --> Input points: ";
                    try {
                                  std::cin >> rec;
                         } catch (std::exception &e) {
                                  std::cout << e.what() << std::endl;
                                  break;
                             }
                                q.Push(rec);
                                continue;
                }
                        else if (cmd == 2) {
                                 Rectangle<int> rec;
                                 std::cout << "insert --> Input points: ";
                                 try {
                                        std::cin >> rec;
                                 } catch (std::exception &e) {
                                        std::cout << e.what() << std::endl;
                                        break;
                                 }
                                 std::cout << "       --> Input index: ";
                                 std::cin >> i;
                                 Queue<Rectangle<int>>::ForwardIterator it =
q.Begin();
                                 for (cnt = 0; cnt < i; cnt++) {
                                        it++;
                                 }
                                 q.Insert(it, rec);
                                 continue;
                           }
                           else {
                                 std::cout << "[Error 1] Invalid input" <<
std::endl;
                                 std::cin.clear();
                                 std::cin.ignore(30000, '\n');
                                 break;
                           }
                    }
                    case 2: {
                           std::cout << "Delete item from front queue or iterator
position" << std::endl;
                        std::cout << "command: 1 - <Pop> or 2 - <Erase>" <<
std::endl;
                           std::cin >> cmd;
                           if (cmd == 1) {
                                 std::cout << "pop -->";
                                 q.Pop();
                                 continue;
```

```
                        }
                        else if(cmd == 2) {
                                std::cout << "erase --> Input index: ";
                                std::cin >> i;
                                Queue<Rectangle<int>>::ForwardIterator it =
q.Begin();
                                for (cnt = 0; cnt < i; cnt++) {
                                        it++;
                                }
                                q.Erase(it);
                                continue;
                        }
                        else {
                                std::cout << "[Error 2] Invalid input" <<
std::endl;
                                std::cin.clear();
                                std::cin.ignore(30000, '\n');
                                break;
                        }
                }
                case 3: {
                        std::cout << "print -->" << std::endl;
                        q.Print();
                        continue;
                }
                case 4: {
                        std::cout << "front --> ";
                        Rectangle<int> value;
                        try {
                                value = q.Front();
                        } catch (std::exception &e) {
                                std::cout << e.what() << std::endl;
                                break;
                        }
                        std::cout << value << std::endl;
                        continue;
                }
                case 5: {
                        std::cout << "back --> ";
                        Rectangle<int> value;
                        try {
                                value = q.Back();
                        }
                        catch (std::exception &e) {
                                std::cout << e.what() << std::endl;
                                break;
                        }
                        std::cout << value << std::endl;
                        continue;
                }
                case 6: {
```

```
                        std::cout << "area --> Input area: ";
                        std::cin >> area;
                        std::cout << "The number figures with area less than
given "
                              << std::count_if(q.Begin(), q.End(),
[area](Rectangle<int> t){
                              return Area(t) < area;
                        }) << std::endl;
                        continue;
                  }
                  case 7: {
                        return 0;
                  }
                  default: {
                        std::cout << "[Error 3] Invalid input" << std::endl;
                        std::cin.clear();
                        std::cin.ignore(30000, '\n');
                  }
            }
      }
      return 0;
}
```

point.h

```
#pragma once



template<typename T>
std::istream &operator>>(std::istream &is, std::pair<T, T> &v) {
      is >> v.first >> v.second;
      return is;
}

template<typename T>
std::ostream &operator<<(std::ostream &os, const std::pair<T,T> &v) {
      os << "[" << v.first << ", " << v.second << "]";
      return os;
}
```

rectangle.h

```
#pragma once
#include <utility>
#include <iostream>

#include "vector.h"

template<typename T>
struct Rectangle {
	std::pair<T,T> vertices[4];
};

template<typename T>
typename Rectangle<T>::std::template pair<T,T> Center(const Rectangle<T> &t);

template<typename T>
double Area(const Rectangle<T> &t);

template<typename T>
std::ostream &Print(std::ostream &os, const Rectangle<T> &t);

template<typename T>
std::istream &Read(std::istream &is, Rectangle<T> &t);

template<typename T>
std::istream &operator>>(std::istream &is, Rectangle<T> &t);

template<typename T>
std::ostream &operator<<(std::ostream &os, const Rectangle<T> &t);

template<typename T>
typename Rectangle<T>::std::template pair<T,T> Center(const Rectangle<T> &t) {
	T x, y;
	x = (t.vertices[0].first + t.vertices[1].first + t.vertices[2].first +
t.vertices[3].first) / 4;
	y = (t.vertices[0].second + t.vertices[1].second + t.vertices[2].second +
t.vertices[3].second) / 4;
```

```
	return std::make_pair(x, y);
}

template<typename T>
double Area(const Rectangle<T> &t) {
	double hor = std::max(abs(t.vertices[0].first - t.vertices[1].first),
abs(t.vertices[1].first - t.vertices[2].first));
	double ver = std::max(abs(t.vertices[0].second - t.vertices[1].second),
abs(t.vertices[1].second - t.vertices[2].second));
	return hor * ver;
}

template<typename T>
std::ostream &Print(std::ostream &os, const Rectangle<T> &t) {
	for (int i = 0; i < 4; i++) {
		os << t.vertices[i];
		if (i != 2) {
			os << " ";
		}
	}
	return os;
}

template<typename T>
std::istream &Read(std::istream &is, Rectangle<T> &t) {
	for (int i = 0; i < 4; i++) {
		is >> t.vertices[i].first >> t.vertices[i].second;
	}
	double AB = Length(t.vertices[0], t.vertices[1]),
		BC = Length(t.vertices[1], t.vertices[2]),
		CD = Length(t.vertices[2], t.vertices[3]),
		DA = Length(t.vertices[0], t.vertices[3]);
	if (AB != CD || BC != DA) {
		throw std::logic_error("Vectors must not be with different
length.");
	}
```

```
	if (!is_parallel(t.vertices[0], t.vertices[1], t.vertices[2], t.vertices[3])){
		throw std::logic_error("Vectors must not be dont parallel.");
	}
	return is;
}

template<typename T>
std::istream &operator>>(std::istream &is, Rectangle<T> &t) {
	return Read(is, t);
}

template<typename T>
std::ostream &operator<<(std::ostream &os, const Rectangle<T> &t) {
	return Print(os, t);
}
```

vector.h

```
#pragma once
#include <utility>
#include <cmath>
#include <iostream>

#include "vertex.h"

template<typename T>
struct Vector {
	Vector(T x_cord, T y_cord) : p1{x_cord}, p2{y_cord} {};
	Vector(std::pair<T, T> &p1, std::pair<T, T> &p2) : p1{p2.first - p1.first}, p2{p2.second - p1.second} {};

	double operator*(const Vector<T> &a) const {
		return (this->p1 * a.p1) + (this->p2 * a.p2);
	}

	Vector<T> &operator=(const Vector<T> &a) {
```

```
            this->p1 = a.p1;
            this->p2 = a.p2;
            return *this;
      }

      T p1, p2;
};

template<typename T>
double Length(const std::pair<T, T> &A, const std::pair<T, T> &B) {
      return sqrt(pow((B.first - A.first), 2) +
                        pow((B.second - A.second), 2));
}

template<typename T>
bool is_parallel(const std::pair<T, T> &A, const std::pair<T, T> &B, const
std::pair<T, T> &C, const std::pair<T, T> &D) {
      return (B.second - A.second) * (D.first - C.first) - (D.second - C.second)
* (B.first - A.first) == 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

project(lab5)

add_executable(lab5 main.cpp)

set_property(TARGET lab5 PROPERTY CXX_STANDARD 11)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

6. Ссылка на репозиторий

https://github.com/nikit34/oop_exercise_05

7. Объяснение результатов работы программы

Контейнер «очередь» реализован с помощью умных указателей. Queue содержит умный указатель std::shared_ptr на первый элемент очереди, умный указатель std::weak__ptr на последний элемент и размер очереди size_t size.
Элемент очереди - класс Node, который содержит умный указатель std::shared_ptr на следующий элемент очереди, умный указатель std::weak_ptr на предыдущий элемент и значение.
Указатель на предыдущий элемент очереди сделан с помощью weak_ptr, чтобы можно было легко удалять элемент из очереди, изменяя только указатели next, тем самым то, на что указывали раньше shared_ptr`ы, удаляется, т. к. на него указывают только weak_ptr`ы, а weak_ptr содержит слабую ссылку и не учитывается при подсчете количества указателей на какой-то объект.

8. Вывод

В ходе работы были получены навыки использования умных указателей и итераторов.
Умные указатели, при использовании позволяют сэкономить время на выявление утечек памяти и исправления их.
Итераторы предоставляют уровень абстракции, скрывающий внутреннюю структуру контейнера от пользователя. Это также позволяет ускорить и обезопасить использование различных реализаций контейнеров с похожим интерфейсом.
В ООП такое поведение называется параметрическим полиморфизмом.

9. Список литературы

1. Перегрузка операторов C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/5.14.php

(дата обращения: 29.09.2020).

2. Битовые операции C++ [Электронный ресурс]. URL: http://www.c-cpp.ru/books/bitovye-operatory

(дата обращения: 29.09.2020).