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: Шубин Григорий Сергеевич
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

a. Реализовать шаблон класса “Ромб”. Параметр шаблона - скалярный тип данных, задающий тип данных для координат.
b. Создать шаблон динамической коллекции “Стек”. Коллекция должна быть реализована при помощи умных указателей.
c. Реализовать forward_iterator по коллекции. Итератор должен быть совместимым со стандартными алгоритмами.
d. Коллекция должна содержать методы begin и end.
e. Коллекция должна содержать методы вставки и удаления с позиции итератора.
f. При выполнении недопустимых операций необходимо генерировать исключения.
g. Коллекция должна содержать методы доступа push(), pop(), top().
h. Программа должна позволять вводить с клавиатуры фигуры и добавлять в коллекцию.
i. Программа должна позволять удалять фигуры с любой позиции коллекции.
j. Программа должна выводить на экран все фигуры при помощи std::for_each.
k. Программа должна выводить на экран количество фигур, площадь которых меньше заданной, при помощи std::count_if.

2. Описание программы

Класс rhombus

Класс rhombus представляет собой структуру для хранения фигур-ромбов. Ромб задаётся координатами центра и длинами диагоналей. Первая введённая диагональ параллельна оси абсцисс, вторая - оси ординат.

В классе определена функция вычисления площади ромба (половина произведения диагоналей), переопределены операторы сравнения на равенство и неравенство, а также определены функции считывания и печати ромба.

Класс stack

В классе stack реализованы все методы для работы с коллекцией “Стек”. По заданию необходимо уметь вставлять элементы на любую позицию стека и удалять их, поэтому мой стек реализован на основе линейного однонаправленного списка.

В качестве вспомогательной структуры для элемента стека я реализовал класс item, в котором содержится текущий элемент стека и ссылка на следующий элемент. Для ссылки используется умный указатель std::shared_ptr. В структуре переопределены операторы присваивания и сравнения элементов стека.

Класс стек хранит в себе следующие атрибуты: head и tail - первый и последний элементы стека соответственно. Изначально они совпадают. Хранение этих элементов необходимо для дальнейшей работы с итераторами.

Внутри класса “Стек” реализован класс iterator. Для итератора переопределены все необходимые операторы (разыменование, сравнение, инкремент) для его совместимости со стандартными алгоритмами. Также в классе описаны все поля, необходимые для корректной работы итератора (iterator_traits). Итератор использует умные указатели std::shared_ptr.

В стеке реализованы следующие методы:

- T& top() - возвращает элемент на вершине стека. Если стек пустой, то генерируется исключение.
- void pop() - удаление вершины стека. Если стек пустой, то генерируется исключение.
- void push(T) - добавление элемента на вершину стека.
- iterator begin() const, iterator end() const - возвращает итераторы на начало и конец стека соответственно
- void insert(iterator&, T) - вставляет элемент на позицию итератора. Проверка корректности итератора перекладывается на программиста.
- void erase(iterator&) - удаление элемента на позиции итератора. Проверка корректности итератора перекладывается на программиста.

main.cpp

В функции main реализован интерфейс для взаимодействия с пользователем согласно заданию. При запуске программы пользователю будут показаны все возможные команды, поддерживаемые программой. Если пользователь введёт неверную команду, то программа выведет соответствующее сообщение. При вводе команды производится вызов соответствующих методов класса stack.

Печать всех фигур реализована при помощи стандартной функции std::for_each. Подсчёт количества фигур с площадью, меньшей заданной реализован при помощи функции std::count_if. Для упрощения реализации использовались лямбда-выражения.

3. Тестирование программы

В качестве тестовых данных программе подаётся набор команд. Интерфейс для взаимодействия с программой:

1. Добавить ромб в стек
2. Удалить ромб из вершины стека
3. Посмотреть элемент на вершине стека
4. Вставить элемент на позицию итератора
5. Удалить элемент с позиции итератора
6. Печать всех фигур
7. Посчитать количество фигур с площадью меньше заданной
0. Выход

test1.txt

Тест

```
1 0 0 2 5 // добавить ромб с центром (0,0) и диагоналями 2 и 5
1 -1 1 10 2 // добавить ромб с центром (-1, 1) и диагоналями 10 и 2
1 50 -100 25 45 // добавить ромб с центром (50, -100) и диагоналями 25 и 45
3 // посмотреть вершину стека
6 // печать всех фигур
7 20 // посчитать количество фигур с площадью меньше 20
2 // удалить элемент из вершины стека
3 // посмотреть вершину стека
6 // печать всех фигур
2 // удалить элемент из вершины стека
2 // удалить элемент из вершины стека (стек становится пустым)
2 // удалить элемент из вершины стека (должна появиться ошибка)
3 // посмотреть вершину стека (стек пустой -> будет ошибка)
6 // печать всех фигур
0 // завершение работы
```

Результат

```
Pushed
Pushed
Pushed
Top: Rhombus {(37.5; -100), (50; -77.5), (62.5; -100), (50; -122.5)}
Rhombus {(37.5; -100), (50; -77.5), (62.5; -100), (50; -122.5)}
Rhombus {(-6; 1), (-1; 2), (4; 1), (-1; 0)}
Rhombus {(-1; 0), (0; 2.5), (1; 0), (0; -2.5)}
The amount of figures with square < 20 is 2
Popped
Top: Rhombus {(-6; 1), (-1; 2), (4; 1), (-1; 0)}
Rhombus {(-6; 1), (-1; 2), (4; 1), (-1; 0)}
Rhombus {(-1; 0), (0; 2.5), (1; 0), (0; -2.5)}
Popped
Popped
Stack is empty
Stack is empty

Process finished with exit code 0
```

test2.txt

Тест

```
1 10 10 3 1 // добавить ромб с центром (10, 10) и диагоналями 3 и 1
1 2 4 20 9 // добавить ромб с центром (2, 4) и диагоналями 20 и 9
1 -10 -5 5 10 // добавить ромб с центром (-10, -5) и диагоналями 5 и 10
6 // печать всех фигур
4 1 4 2 2 2 // добавить ромб с центром (1, 4) и диагоналями 2 и 2 на позицию
2
6 // печать всех фигур
5 3 // удалить ромб с 3 позиции
6 // печать всех фигур
4 0 0 10 15 0 // добавить ромб с центром (0,0) и диагоналями 10 и 15 на
позицию 0
6 // печать всех фигур
5 2 // удалить ромб с 2 позиции
5 0 // удалить ромб с 0 позиции
```

```
5 0 // удалить ромб с 0 позиции
5 1000 // удалить ромб с 1000 позиции (должна быть ошибка)
6 // печать всех фигур
0  // завершение работы
```

Результат

```
Pushed
Pushed
Pushed
Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}
Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}
Rhombus {(8.5; 10), (10; 10.5), (11.5; 10), (10; 9.5)}
Inserted
Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}
Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}
Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}
Rhombus {(8.5; 10), (10; 10.5), (11.5; 10), (10; 9.5)}
Erased
Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}
Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}
Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}
Inserted
Rhombus {(-5; 0), (0; 7.5), (5; 0), (0; -7.5)}
Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}
Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}
Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}
Erased
Erased
Erased
Invalid position
Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}

Process finished with exit code 0
```

4. Листинг программы

rhombus.h

```
#ifndef OOP_LAB5_RHOMBUS_H
#define OOP_LAB5_RHOMBUS_H

#include <iostream>

template<class T>
class rhombus {
private:
    std::pair<T, T> center;
    double diagonal1;
    double diagonal2;
public:
    rhombus() : diagonal1(0), diagonal2(0) {
        center = std::make_pair(0, 0);
    }
```

```
    rhombus(T x_center, T y_center, double diag1, double diag2) :
diagonal1(diag1), diagonal2(diag2) {
        center = std::make_pair(x_center, y_center);
    }

    double square() { return diagonal1 * diagonal2 * 0.5;}

    bool operator==(rhombus<T> &other) {
        if (center != other.center) {return false;}
        if (diagonal1 != other.diagonal1) {return false;}
        if (diagonal2 != other.diagonal2) {return false;}
        return true;
    }

    bool operator!=(rhombus<T> &other) {return !(*this == other);}

    template<class T1>
    friend std::ostream &operator<<(std::ostream &out, rhombus<T1> &r);

    template<class T1>
    friend std::istream &operator>>(std::istream &in, rhombus<T1> &r);


};

template<class T>
std::istream &operator>>(std::istream &in, rhombus<T> &r) {
    in >> r.center.first >> r.center.second >> r.diagonal1 >> r.diagonal2;
}

template<class T>
std::ostream &operator<<(std::ostream &out, rhombus<T> &r) {
    out << "Rhombus {(" << r.center.first - r.diagonal1 * 0.5 << "; " <<
r.center.second << "), (";
    out << r.center.first << "; " << r.center.second + r.diagonal2 * 0.5 <<
"), (";
    out << r.center.first + r.diagonal1 * 0.5 << "; " << r.center.second <<
"), (";
    out << r.center.first << "; " << r.center.second - r.diagonal2 * 0.5 <<
")}";
}


#endif //OOP_LAB5_RHOMBUS_H
```

stack.h

```
#ifndef OOP_LAB5_STACK_H
#define OOP_LAB5_STACK_H

#include <memory>
#include <exception>

template<class T>
class stack {
private:
    struct item {
        T value;
```

```
        std::shared_ptr<item> next = nullptr;
        item() = default;
        item(T val) : value(val) {}
        item &operator=(item const &other) {
            value = other.value;
            next = other.next;
            return *this;
        }

        bool operator!=(item &other) {
            if (value == other.value) {
                if (next == nullptr && other.next == nullptr) {
                    return false;
                }
                return next != other.next;
            }
            return value != other.value;
        }
    };

    item head;
    item tail = head;
public:
    class iterator {
    private:
        std::shared_ptr<item> node;
        friend class stack;

    public:
        // iterator traits
        using difference_type = ptrdiff_t;
        using value_type = T;
        using reference = T &;
        using pointer = std::shared_ptr<T>;
        using iterator_category = std::forward_iterator_tag;

        iterator(item node_) { node = std::make_shared<item>(node_);}

        iterator &operator++() {
            node = node->next;
            return *this;
        }

        reference operator*() {return node->value;}

        pointer operator->() {return &node->value;}

        iterator& operator=(iterator& other) {node = other.node;}

        bool operator!=(iterator &other) {return *node != *(other.node);}

        bool operator!=(iterator &&other) {return *node != *(other.node);}

        bool operator==(iterator &other) {return !(*this != other);}

        bool operator==(iterator &&other) {return !(*this != other);}
    };
```

```
    T &top() {
        if (head != tail) {return head.value;}
        throw std::runtime_error("Stack is empty");
    }

    void pop() {
        if (head != tail) {
            head = *head.next;
        } else {
            throw std::runtime_error("Stack is empty");
        }
    }

    void push(T val) {
        item new_head(val);
        new_head.value = val;
        new_head.next = std::make_shared<item>(head);
        head = new_head;
    }

    // iterator to the top
    iterator begin() const {
        return iterator(head);
    }

    // iterator to the last element
    iterator end() const {
        return iterator(tail);
    }

    // insert to iterator's pos
    void insert(iterator &it, T val) {
        if (it == begin()) {
            push(val);
        } else {
            item new_node(*it);
            new_node.next = it.node->next;
            *it = val;
            it.node->next = std::make_shared<item>(new_node);
        }
    }

    // erase from iterator's pos
    void erase(iterator &it) {
        if (it == begin()) {
            pop();
        } else if (it == end()) {
            tail = *it;
        } else {
            *it = it.node->next->value;
            it.node->next = it.node->next->next;
        }
    }
};
#endif //OOP_LAB5_STACK_H
```

main.cpp

```
#include <iostream>
#include <algorithm>

#include "rhombus.h"
#include "stack.h"

void print_menu() {
    std::cout << "1. Push rhombus to stack" << std::endl;
    std::cout << "2. Pop rhombus from the stack" << std::endl;
    std::cout << "3. Check the element on the top of the stack" << std::endl;
    std::cout << "4. Insert rhombus to the position" << std::endl;
    std::cout << "5. Erase rhombus from the position" << std::endl;
    std::cout << "6. Print all figures" << std::endl;
    std::cout << "7. Calculate the amount of figures with square less than
..." << std::endl;
    std::cout << "0. Exit" << std::endl;
    std::cout << std::endl;
    std::cout << "(to add a rhombus type the in coords of the center and
lengths of diagonals)" << std::endl;
    std::cout << std::endl;
}

int main() {
    stack<rhombus<int>> s;
    print_menu();
    char cmd;
    while (std::cin >> cmd) {
        if (cmd == '1') {
            rhombus<int> r;
            std::cin >> r;
            s.push(r);
            std::cout << "Pushed" << std::endl;

        } else if (cmd == '2') {
            try {
                s.pop();
                std::cout << "Popped" << std::endl;
            }
            catch (std::exception &ex) {
                std::cout << ex.what() << std::endl;
            }
        } else if (cmd == '3') {
            try {
                auto t = s.top();
                std::cout << "Top: " << t << std::endl;
            }
            catch (std::exception &ex) {
                std::cout << ex.what() << std::endl;
            }
        } else if (cmd == '4') {
            rhombus<int> r;
            std::cin >> r;
            unsigned int pos;
            std::cin >> pos;
            auto iter = s.begin();
```

```
            try {
                if (iter == s.end() && pos != 0) {
                    throw std::runtime_error("Invalid position");
                }
                for (unsigned int i = 0; i < pos; ++i) {
                    ++iter;
                    if (iter == s.end() && i != pos - 1) {
                        throw std::runtime_error("Invalid position");
                    }
                }
                s.insert(iter, r);
                std::cout << "Inserted" << std::endl;
            }
            catch (std::exception &ex) {
                std::cout << ex.what() << std::endl;
            }
        } else if (cmd == '5') {
            unsigned int pos;
            std::cin >> pos;
            auto iter = s.begin();
            try {
                if (iter == s.end()) {
                    throw std::runtime_error("Invalid position");
                }
                for (unsigned int i = 0; i < pos; ++i) {
                    ++iter;
                    if (iter == s.end() && i != pos) {
                        throw std::runtime_error("Invalid position");
                    }
                }
                s.erase(iter);
                std::cout << "Erased" << std::endl;
            }
            catch (std::exception &ex) {
                std::cout << ex.what() << std::endl;
            }
        } else if (cmd == '6') {
            std::for_each(s.begin(), s.end(), [](rhombus<int> r) {
                std::cout << r << std::endl;
            });

        } else if (cmd == '7') {
            double num;
            std::cin >> num;
            int count = std::count_if(s.begin(), s.end(), [num](auto &r) {
                return r.square() < num;
            });
            std::cout << count << std::endl;

        } else if (cmd == '0') {
            break;
        } else {
            std::cout << "Invalid cmd" << std::endl;
        }
    }
}
```

5. Выводы

В данной лабораторной работе был изучен шаблон проектирования “итератор”. Итератор - это модернизированный указатель, который умеет перемещаться по всем элементам контейнера. Итератор позволяет абстрагироваться от внутреннего устройства коллекций. Именно поэтому в стандартной библиотеке алгоритмов C++ все функции реализованы на итераторах. Это делает их универсальными для всех контейнеров.

В рамках данной работы был реализован пользовательский класс для итератора. Стандартные функции могут принимать пользовательские итераторы только в том случае, если в них переопределены операции разыменования, инкремента и оператор сравнения, а также указаны все используемые типы данных (iterator_traits).

Также в работе использовались механизмы генерации исключительных ситуаций и лямбда-выражения - важные инструменты для проектирования классов и функций в C++.

Список используемых источников

1. Руководство по языку C++ [Электронный ресурс]. URL: https://www.cplusplus.com/ (дата обращения 02.12.2020).
2. Шаблоны классов в C++ [Электронный ресурс]. URL: http://cppstudio.com/post/5188/ (дата обращения 02.12.2020).
3. Умные указатели в C++ [Электронный ресурс]. URL: https://eax.me/cpp-smart-pointers/ (дата обращения 03.12.2020).
4. Итераторы в C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/7.3.php (дата обращения 03.12.2020).