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: Ивенкова Любовь Васильевна
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

Содержание

1. Постановка задачи

Вариант: 2
Задача:

Создать шаблонный класс квадрата с публичными полями. Создать шаблонную коллекцию “стек”, который реализован с помощью умных указателей. В качестве параметров в него должны поступать фигуры. Реализовать forward iterator по стеку, а так же begin(), end(), pop(), push(), top(). Реализовать метод вставки на позицию итератора inrase и метод удаления из позиции итератора erase.

Реализованная программа должна:

- позволять добавлять фигуры в стек;
- позволять удалять фигуры из стека;
- выводить все введенные фигуры с помощью std::for_each;
- выводить количество объектов, площадь которых меньше заданной, с помощью std::count_if.

2. Описание программы

Программа принимает в себя данные из консоли и из файла, при перенаправлении потока ввода вывода, и выполняет заданные действия. При запуске появляется меню выбора операций, после выбора которой вводятся данные. В список операций входит *добавление фигуры в стек, удаление фигуры из стека, вывод всех координат фигур в стеке, вывод количества фигур, площадь которых меньше заданной*.

Программа состоит из трех файлов: main.cpp, stack.h и figure.h:

- figure.h - описание классов фигур. Они представлены шаблонными классами, в которых присутствуют лишь публичные поля. Функция get_coords передает в класс координаты на основе стороны, которую ввел пользователь.

- stack.h - реализация стека с помощью умных указателей.

  Класс Node - элемент стека. Он содержит фигуру и указатели на предыдущий и следующий элементы.
  Класс Forward_iterator - реализация итератора, который может совместим со стандартными алгоритмами. Содержит перегрузку операторов: ++, ->, *, ==, !=. Является дружественным классом для класса Stack.
  Класс Stack. Содержит указатели на первый и последний элементы стека, а также его размер.
  void Push(T& elem) - добавление нового элемента elem в конец стека.
  void Insert(Forward_iterator iter, T& obj) - добавление элемента obj на

позицию итератора iter.
void Erase(Forward_iterator iter) - удаление элемента из позиции итератора iter.
bool Empty() - проверка стека на наличие элементов в нем.
void Pop() - удаление последнего элемента стека.
Forward_iterator Top() - возвращает верхний элемент стека.
Forward_iterator Begin() - возвращает первый элемент стека.
Forward_iterator End() - возвращает последний элемент стека (NULL).

- main.cpp - главный файл. В нем создается стек. Считывается выбор пользователя, исходя из которого выбираются дальнейшие действия.

  void menu() - вывод меню.
  void error() - вывод сообщения об ошибке.

Переменные классов

class Figure
using coords = std::pair<T, T>;
coords a, b, c, d; - координаты квадрата.
class Stack
std::shared_ptr<Node> head, tail; - указатели на верхний и нижний(первый) элементы.
long long size; - размер стека.
class Forward_iterator
std::shared_ptr<Node> node; - указатель на фигуру.
class Node
std::shared_ptr<Node> next, previous; - указатель на следующий и предыдущий элементы.
T value; - фигура.

3. Набор тестов

Таблица 1. Тест 1

1 6 1	-добавить в конец стека новый квадрат со стороной 6
1 7 1	-добавить в конец стека новый квадрат со стороной 7
1 9 2 3	-добавить новый квадрат со стороной 9 на 3 позицию в стеке (ошибка: такого итератора в стеке нет)
3	-вывести все фигуры в стеке (выведет квадраты со сторонами 6 и 7)
2 2	-удалить фигуру на 2 позиции в стеке
3	-вывести все фигуры в стеке (выведет квадрат со стороной 6)
0	-выход из программы

Таблица 2. Тест 2

1 8 2 2	-добавить новый квадрат со стороной 8 на 2 позицию в стеке (ошибка: такого итератора в стеке нет)
1 9 2 1	-добавить новый квадрат со стороной 9 на 1 позицию в стеке
3	-вывести все фигуры в стеке (квадрат со стороной 9)
1 10 1	-добавить в конец стека новый квадрат со стороной 10
1 5 1	-добавить в конец стека новый квадрат со стороной 5
3	-вывести все фигуры в стеке (квадраты со сторонами 9, 10 и 5)
4 90	-вывести количество фигур, площадь которых меньше 90
0	-выход из программы

4. Результат выполнения тестов

Тест 1:

```
Enter 0-4 to:
1 - add square to stack
2 - delete figure from stack
3 - print all coordinates of figures in stack
4 - print the number of figures with an area less than...
0 - exit
Your choice: 1
Enter the side of the square: 6
Insert at the end of the stack [1] or at a specific position [2]? 1
What's next?
>>1
Enter the side of the square: 7
Insert at the end of the stack [1] or at a specific position [2]? 1
What's next?
>>1
Enter the side of the square: 9
Insert at the end of the stack [1] or at a specific position [2]? 2
Enter the position number on the stack 3
There is no such iterator
What's next?
>>3
(0, 0), (6, 0), (6, 6), (0, 6)
(0, 0), (7, 0), (7, 7), (0, 7)
What's next?
>>2Enter the position number on the stack 2
What's next?
>>3
(0, 0), (6, 0), (6, 6), (0, 6)
What's next?
>>0
The program is closed, goodbye!
```

Тест 2:

```
Enter 0-4 to:
1 - add square to stack
2 - delete figure from stack
3 - print all coordinates of figures in stack
4 - print the number of figures with an area less than...
```

```
0 - exit
Your choice: 1
Enter the side of the square: 8
Insert at the end of the stack [1] or at a specific position [2]? 2
Enter the position number on the stack 2
There is no such iterator
What's next?
>>1
Enter the side of the square: 9
Insert at the end of the stack [1] or at a specific position [2]? 2
Enter the position number on the stack 1
What's next?
>>3
(0, 0), (9, 0), (9, 9), (0, 9)
What's next?
>>1
Enter the side of the square: 10
Insert at the end of the stack [1] or at a specific position [2]? 1
What's next?
>>1
Enter the side of the square: 5
Insert at the end of the stack [1] or at a specific position [2]? 1
What's next?
>>3
(0, 0), (9, 0), (9, 9), (0, 9)
(0, 0), (10, 0), (10, 10), (0, 10)
(0, 0), (5, 0), (5, 5), (0, 5)
What's next?
>>4
Enter area: 90
The number of figures with an area less than 90 is 2
What's next?
>>0
The program is closed, goodbye!
```

5. Листинг программы

main.cpp

```
/*https://github.com/Li-Iven/OOP/tree/main/oop_exercise_05
Ивенкова Любовь, 208 группа*/
#include <iostream>
#include"figure.h"
#include"stack.h"
#include<algorithm>

void menu() {
    std::cout << "Enter 0-4 to:" << std::endl;
    std::cout << "1 - add square to stack" << std::endl;
    std::cout << "2 - delete figure from stack" << std::endl;
    std::cout << "3 - print all coordinates of figures in stack" << std::endl;
    std::cout << "4 - print the number of figures with an area less than..." << std::endl;
```

```
    std::cout << "0 - exit" << std::endl;
}

void error() {
    std::cout << "Incorrect value!" << std::endl;
}

int main()
{
    int a, b, choice;
    long double area;
    Stack<Figure<int>> stack;
    Figure<int> sq;
    menu();
    std::cout << "Your choice: ";
    do {
        std::cin >> choice;
        switch(choice) {
        case 1:
            std::cout << "Enter the side of the square: ";
            std::cin >> a;
            sq = Figure<int>(a);
            std::cout << "Insert at the end of the stack [1] or at a specific position [2]? ";
            std::cin >> choice;
            if (choice == 1) {
                stack.Push(sq);
            }
            else if (choice == 2) {
                std::cout << "Enter the position number on the stack ";
                try {
                    std::cin >> b;
                    if (stack.Size() == 0) {
                        b--;
                        if (b<0 || b>stack.Size()) {
                            throw b;
                        }
                    }
                }
                catch (const int a) {
                    std::cout << "There is no such iterator\n";
                    break;
                }
                Stack<Figure<int>>::Forward_iterator it = stack.Begin();
                int i = 1;
                if (i != b) {
                    for (i; i < b; ++i) {
                        ++it;
```

```
            }
        }
        stack.Insert(it, sq);
    }
    else {
        error();
    }
    break;
case 2: {
    std::cout << "Enter the position number on the stack ";
    std::cin >> b;
    try {
        if (b<0 || b>stack.Size()) {
            throw b;
        }
    }
    catch (const int a) {
        std::cout << "There is no such iterator\n";
        break;
    }
    Stack<Figure<int>>::Forward_iterator it = stack.Begin();
    for (int i = 1; i < b; ++i) {
        ++it;
    }
    stack.Erase(it);
    break;
}
case 3: {
    auto print = [](Figure<int>& elem) {
        std::cout << "("<<elem.a.first << ", " << elem.a.second<<"), ";
        std::cout << "(" << elem.b.first << ", " << elem.b.second << "), ";
        std::cout << "(" << elem.c.first << ", " << elem.c.second << "), ";
        std::cout << "(" << elem.d.first << ", " << elem.d.second << ")"<<std::endl;
    };
    std::for_each(stack.Begin(), stack.End(), print);
    break;
}
case 4:
{
    std::cout << "Enter area: ";
    std::cin >> area;
    try {
        if (area < 0) {
            throw area;
        }
    }catch(const long double a){
        std::cout << "You entered negative area\n";
```

```
            break;
        }
        long double count = std::count_if(stack.Begin(), stack.End(), [area](Figure<int>& elem)
{return area > elem.d.second * elem.d.second; });
        std::cout << "The number of figures with an area less than " << area << " is " << count <<
std::endl;
        break;
    }
    case 0:
        std::cout << "The program is closed, goodbye!\n";
        return false;
        break;
    default:
        std::cout << "Incorrect values! Try again.\n";
        break;
    }
    std::cout << "What's next?\n";
    std::cout << ">>";
  } while (choice);
  return 0;
}
```

stack.h

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

template<typename T>
class Stack {
private:
        class Node {
        private:
                std::shared_ptr<Node> next, previous;
                T value;

        public:
                Node() : value(nullptr), next(nullptr), previous(nullptr){}
                Node(T val): value(val), next(nullptr), previous(nullptr) {}
                ~Node(){}
                friend class Stack;
        };

        std::shared_ptr<Node> head, tail;
        long long size;
```

```
public:
	Stack(): head(nullptr), tail(nullptr), size(0) {}
	~Stack() {}
	long long Size() {
		return size;
	}
	class Forward_iterator {
	public:
		using value_type = T;
		using reference = T&;
		using pointer = T*;
		using difference_type = ptrdiff_t;
		using iterator_category = std::forward_iterator_tag;
		Forward_iterator(std::shared_ptr<Node> elem=nullptr) :node(elem) {}
		~Forward_iterator() {}
		friend Stack;
		bool operator==(const Forward_iterator& other) const{
			return node == other.node;
		}

		bool operator!=(const Forward_iterator& other) const{
			return node != other.node;
		}

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

		Forward_iterator operator++(int index) {
			Forward_iterator tmp(node);
			node = node->next;
			return tmp;
		}

		std::shared_ptr<Node> operator->() {
			return this->node;
		}

		T& operator*() {
			return this->node->value;
		}

	private:
		std::shared_ptr<Node> node;
	};
```

```
void Push(T& elem) {
	if (size == 0) {
		tail = std::make_shared<Node>(elem);
		head = tail;
	}
	else {
		std::shared_ptr<Node> node = std::make_shared<Node>(elem);
		node->previous = head;
		head->next = node;
		head = node;
	}
	size++;
}

void Insert(Forward_iterator iter, T& obj) {
	if (iter == Begin()) {
		if (tail == nullptr) {
			Push(obj);
			return;
		}
		else {
			std::shared_ptr<Node> tmp = std::make_shared<Node>(obj);
			tail->previous = tmp;
			tmp->next = tail;
			tail = tmp;
		}
	}else if(iter == End()){
		std::shared_ptr<Node> tmp = std::make_shared<Node>(obj);
		tmp->next = head;
		tmp->previous = head->previous;
		head->previous = tmp;
	}
	else {
		std::shared_ptr<Node> tmp = std::make_shared<Node>(obj);
		tmp->next = iter.node;
		iter->previous->next = tmp;
		tmp->previous = iter->previous;
		iter->previous = tmp;
	}
	size++;
}

void Erase(Forward_iterator iter) {
	if (iter == tail) {
		if (size == 0) {
			throw "List is empty";
		}
```

```
                else if (size == 1) {
                        head = nullptr;
                        tail = nullptr;
                }
                else {
                        tail->next->previous = nullptr;
                        tail = tail->next;
                }
                size--;
                return;
        }
        else if (iter == head) {
                std::shared_ptr<Node> tmp = head->previous;
                head->previous->next = nullptr;
                head = tmp;
                size--;
                return;
        }
        else {
                iter->next->previous = iter->previous;
                iter->previous->next = iter->next;
                size--;
                return;
        }
    }

    bool Empty() {
        return size==0;
    }

    Forward_iterator Top() {
        return head;
    }

    Forward_iterator Begin() {
        return tail;
    }

    Forward_iterator End() {
        return Forward_iterator{};
    }
};
```

figure.h

```
#pragma once
template<typename T>
struct Figure {
	using coords = std::pair<T, T>;
	coords a, b, c, d;
	Figure(T a) {
		get_coords(*this, a);
	}
	Figure() {}
	~Figure(){}
};

template<typename T, typename S>
void get_coords(T& elem, S side) {
	elem.a.first = elem.a.second = elem.d.first = elem.b.second = 0;
	elem.b.first = elem.c.first = elem.c.second = elem.d.second = side;
}
```

6. Вывод

Благодаря данной лабораторной работе я ознакомилась с умными указателями std::shared_ptr, std::weak_ptr, std::unique_ptr. Узнала стандартные алгоритмы std::count_if и std::for_each. Смогла реализовать собственную коллекцию “стек”. Реализовала forward_iterator по коллекции.

7. Список используемых источников

1. std::for_each [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/algorithm/for_each (Дата обращения: 12.04.2021)
2. std::count, std::count_if [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/algorithm/count (Дата обращения: 13.04.2021)
3. Обработка исключений [Электронный ресурс]. URL: https://ravesli.com/urok-182-obrabotka-isklyuchenij/ (Дата обращения: 13.04.2021)
4. Forward iterators in C++ [Электронный ресурс]. https://www.geeksforgeeks.org/forward-iterators-in-cpp/ (Дата обращения: 13.04.2021)
5. Умный указатель std::shared_ptr [Электронный ресурс]. https://ravesli.com/urok-194-std-shared_ptr/ (Дата обращения 13.04.2021)
6. Умные указатели и семантика перемещения [Электронный ресурс].

https://ravesli.com/urok-189-umnye-ukazateli-i-semantika-peremeshheniya/ (Дата обращения: 13.04.2021)
7. Smart pointers [Электронный ресурс]. https://eax.me/cpp-smart-pointers/ (Дата обращения: 13.04.2021)