Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «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: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Чернышев Л. Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Изучить основы работы с коллекциями, познакомиться с шаблоном проектирования «Итератор».

Создать шаблон динамической коллекции согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей. 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: pop, push, top;
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 2. Квадрат, стек.

2. Описание программы

Будем использовать шаблонный класс квадрата из предыдущей лабораторной работы. Для реализация стека и итераторы используем std::shared_ptr, чтобы при вызове деструктора итератора не уничтожался сам стек. Для вставки и удаления из позицию итератора нужно представлять стек как список, потому что нужно изменять указатели соседних элементов. При использовании итератора, если произошёл выход за границу или пытаемся удалить пустой итератор, будем генерировать исключение std::runtime_error, тем самым обезопасив программу.

3. Набор тестов

Программа принимает на вход положительное число N - количество фигур в стеке. В следующих N строках следует описание каждого квадрата: координаты левого нижнего угла и длина стороны квадрата. После этих строк программа принимает на вход индекс элемента для удаления из стека и минимальную площадь квадрата.

Тест №1

```
5
0
0 0 10
1
1 1 9
0
10 10 10
0
0 0 9
4
5 5 5
3
26
```

Тест №2

```
5
0
-10 -10 1
1
-5 -5 5
1
7 7 2
2
10 10 3
3
0 0 4
5
9
```

Тест №3

```
5
0
-10 -10 1
1
-5 -5 5
1
7 7 2
2
10 10 3
3
0 0 4
1
9
```

Тест №4

```
1
0
-1000 -1000 1000
1
10
```

4. Результат выполнения тестов

Программа выводит весь стек до и после удаления элемента. Затем программа выводит количество квадратов с площадью, не меньшей заданной.

Тест №1

```
Your input:
Square {(0, 0), (0, 9), (9, 9), (9, 0)}
Square {(10, 10), (10, 20), (20, 20), (20, 10)}
Square {(0, 0), (0, 10), (10, 10), (10, 0)}
Square {(1, 1), (1, 10), (10, 10), (10, 1)}
Square {(5, 5), (5, 10), (10, 10), (10, 5)}
After erase:
Square {(0, 0), (0, 9), (9, 9), (9, 0)}
Square {(10, 10), (10, 20), (20, 20), (20, 10)}
Square {(1, 1), (1, 10), (10, 10), (10, 1)}
Square {(5, 5), (5, 10), (10, 10), (10, 5)}
Number of figures with square greater or equal, than 26: 3
```

Тест №2

```
Your input:
Square {(-10, -10), (-10, -9), (-9, -9), (-9, -10)}
Square {(7, 7), (7, 9), (9, 9), (9, 7)}
Square {(10, 10), (10, 13), (13, 13), (13, 10)}
Square {(0, 0), (0, 4), (4, 4), (4, 0)}
Square {(-5, -5), (-5, 0), (0, 0), (0, -5)}
After erase:
Square {(-10, -10), (-10, -9), (-9, -9), (-9, -10)}
Square {(7, 7), (7, 9), (9, 9), (9, 7)}
Square {(10, 10), (10, 13), (13, 13), (13, 10)}
Square {(0, 0), (0, 4), (4, 4), (4, 0)}
Number of figures with square greater or equal, than 9: 2
```

Тест №3
Your input:
Square {(-10, -10), (-10, -9), (-9, -9), (-9, -10)}
Square {(7, 7), (7, 9), (9, 9), (9, 7)}
Square {(10, 10), (10, 13), (13, 13), (13, 10)}
Square {(0, 0), (0, 4), (4, 4), (4, 0)}
Square {(-5, -5), (-5, 0), (0, 0), (0, -5)}
Input index to erase from stack
Square {(7, 7), (7, 9), (9, 9), (9, 7)}
Square {(10, 10), (10, 13), (13, 13), (13, 10)}
Square {(0, 0), (0, 4), (4, 4), (4, 0)}
Square {(-5, -5), (-5, 0), (0, 0), (0, -5)}
Number of figures with square greater or equal, than 9: 3

Тест №4
Your input:
Square {(-1000, -1000), (-1000, 0), (0, 0), (0, -1000)}
After erase:
Number of figures with square greater or equal, than 10: 0

5. Листинг программы

Программа разделена на файлы: square.hpp, stack.hpp, main.cpp. В первом реализация шаблонного класса квадрата, во втором реализация шаблонной коллекции стека и итератора. В main.cpp взаимодействие с коллекцией.

square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include <iostream>
#include <tuple>

template<class T>
struct TSquare {
	/* Cords of left bottom corner, side */
	std::pair<T, T> Cord;
	T Side;

	TSquare(const std::pair<T, T> & cord, T side) : Cord(cord),
Side(side) {}
};

template<class T>
T CalcSquare(const TSquare<T> & Sq) {
	return Sq.Side * Sq.Side;
}

template<class T>
std::ostream & operator << (std::ostream & out, const TSquare<T> &
sq) {
	out << "Square {";
	out << std::pair<T, T>(sq.Cord.first, sq.Cord.second) << ", ";
	out << std::pair<T, T>(sq.Cord.first, sq.Cord.second +
sq.Side) << ", ";
	out << std::pair<T, T>(sq.Cord.first + sq.Side, sq.Cord.second
+ sq.Side) << ", ";
	out << std::pair<T, T>(sq.Cord.first + sq.Side,
sq.Cord.second);
	out << "}";
	return out;
}

template<class T1, class T2>
std::ostream & operator << (std::ostream & out, const
std::pair<T1, T2> & p) {
	out << "(" << p.first << ", " << p.second << ")";
	return out;
}

#endif /* SQUARE_HPP */
```

stack.hpp

```
#ifndef STACK_HPP
#define STACK_HPP

#include <iostream>
#include <memory>

template<class T>
class TStackNode;

template<class T>
class TStack;

template<class T>
void operator ++ (std::shared_ptr< TStackNode<T> > & curStackNode)
{
	if (curStackNode) {
		curStackNode = curStackNode->Next;
	} else {
		throw(std::runtime_error("Iterator points to nullptr!"));
	}
}

template<class T>
bool operator != (const TStackNode<T> & lhs, const TStackNode<T> &
rhs) {
	return &lhs.Data != &rhs.Data;
}

template<class T>
bool operator == (const TStackNode<T> & lhs, const TStackNode<T> &
rhs) {
	return &lhs.Data == &rhs.Data;
}

template<class T>
std::ostream & operator << (std::ostream & out, const
TStackNode<T> & curStackNode) {
	out << curStackNode.Data;
	return out;
}

template<class T>
class TStackNode {
public:
	T Data;
	std::shared_ptr< TStackNode<T> > Next;
public:
	TStackNode() noexcept : Data(), Next(nullptr) {};
	explicit TStackNode(const T & elem) noexcept : Data(elem),
Next(nullptr) {};
```

```
     friend void operator ++ <> (std::shared_ptr< TStackNode<T> > &
curStackNode);
     friend bool operator != <> (const TStackNode<T> & lhs, const
TStackNode<T> & rhs);
     friend bool operator == <> (const TStackNode<T> & lhs, const
TStackNode<T> & rhs);
     friend std::ostream & operator << <> (std::ostream & out,
const TStackNode<T> & curStackNode);
     friend class TStack<T>;
};

template<class T>
class TStack {
private:
     std::shared_ptr< TStackNode<T> > TopNode;
public:
     class iterator {
     private:
          std::shared_ptr< TStackNode<T> > ptr;
     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() : ptr(nullptr) {}
          iterator(const std::shared_ptr< TStackNode<T> > &
anotherPtr) : ptr(anotherPtr) {}

          bool IsNullptr() {
               return ptr == nullptr;
          }

          friend void operator ++ (iterator & it) {
               ++it.ptr;
          }

          friend bool operator != (const iterator & lhs, const
iterator & rhs) {
               return lhs.ptr != rhs.ptr;
          }

          friend std::ostream & operator << (std::ostream & out,
const iterator & it) {
               out << *it.ptr;
               return out;
          }
          TStackNode<T>& operator * () {
               return *ptr;
          }
     };
```

```
        iterator begin();
        iterator end();

        TStack() noexcept : TopNode(nullptr) {};

        void Pop();
        void Push(const T & elem);
        T Top();
        void Erase(iterator it);
        void Insert(iterator it, const T & elem);
};


template<class T>
typename TStack<T>::iterator TStack<T>::begin() {
        return TStack<T>::iterator(TopNode);
}

template<class T>
typename TStack<T>::iterator TStack<T>::end() {
        return TStack<T>::iterator(nullptr);
}


template<class T>
void TStack<T>::Pop() {
        if (TopNode) {
                TopNode = TopNode->Next;
        } else {
                throw(std::runtime_error("Stack is empty!"));
        }
}

template<class T>
void TStack<T>::Push(const T & elem) {
        TStackNode<T>* newNode = new TStackNode(elem);
        std::shared_ptr< TStackNode<T> > newNodeShared(newNode);
        newNodeShared->Next = TopNode;
        TopNode = newNodeShared;
}

template<class T>
T TStack<T>::Top() {
        if (TopNode) {
                return TopNode->Data;
        } else {
                throw(std::runtime_error("Stack is empty!"));
        }
}
```

```
template<class T>
void TStack<T>::Erase(TStack<T>::iterator it) {
     if (it.IsNullptr()) {
          throw(std::runtime_error("Iterator points to nullptr!"));
     } else {
          if (*it == *TopNode) {
               TopNode = TopNode->Next;
          } else {
               std::shared_ptr< TStackNode<T> > prevNode = TopNode;
               while (*prevNode->Next != *it) {
                    ++prevNode;
               }
               prevNode->Next = prevNode->Next->Next;
          }
     }
}

template<class T>
void TStack<T>::Insert(TStack<T>::iterator it, const T & elem) {
     TStackNode<T>* newNode = new TStackNode(elem);
     std::shared_ptr< TStackNode<T> > newNodeShared(newNode);
     if (TopNode) {
          if (*it == *TopNode) {
               newNodeShared->Next = TopNode;
               TopNode = newNodeShared;
               return;
          }
          std::shared_ptr< TStackNode<T> > prevNode = TopNode;
          if (it.IsNullptr()) {
               while (prevNode->Next != nullptr) {
                    prevNode = prevNode->Next;
               }
               prevNode->Next = newNodeShared;
          } else {
               while (*prevNode != *it) {
                    prevNode = prevNode->Next;
               }
               newNodeShared->Next = prevNode->Next;
               prevNode->Next = newNodeShared;
               std::swap(prevNode->Data, prevNode->Next->Data);
          }
     } else {
          TopNode = newNodeShared;
     }

}

#endif /* STACK_HPP */
```

main.cpp

```
#include <algorithm>
#include "stack.hpp"
#include "square.hpp"

/*
 * Инютин М А М8О-207Б-19
 * 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: pop, push, top;
 * 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).
 */

int main() {
     size_t n;
     auto Print = [](const TStackNode< TSquare<int> > & sq) {
          std::cout << sq << std::endl;
     };
     std::cout << "Input number of squares in stack" << std::endl;
     std::cin >> n;
     int cordX, cordY, side;
     TStack< TSquare<int> > st;
     for (size_t i = 0; i < n; ++i) {
          size_t n;
          std::cout << "Input index to insert a square" <<
std::endl;
          std::cin >> n;
          std::cout << "Input square as follows: x y a" <<
std::endl;
```

```
            std::cout << "x, y is a left bottom corner cords" <<
std::endl;
            std::cout << "a is square side" << std::endl;
            std::cin >> cordX >> cordY >> side;
            try {
                TStack< TSquare<int> >::iterator it = st.begin();
                while (n--) {
                    ++it;
                }
                st.Insert(it, TSquare<int>(std::pair<int,
int>(cordX, cordY), side));
            } catch (std::runtime_error & exception) {
                std::cout << exception.what() << std::endl;
            }
        }
        std::cout << "Your input:" << std::endl;
        std::for_each(st.begin(), st.end(), Print);
        std::cout << "Input index to erase from stack" << std::endl;
        std::cin >> n;
        try {
            TStack< TSquare<int> >::iterator it = st.begin();
            while (n > 1) {
                ++it;
                --n;
            }
            st.Erase(it);
        } catch (std::runtime_error & exception) {
            std::cout << exception.what() << std::endl;
        }
        std::cout << "After erase:" << std::endl;
        std::for_each(st.begin(), st.end(), Print);
        int minimalSquare;
        std::cout << "Input minimal square" << std::endl;
        std::cin >> minimalSquare;
        auto MatchSqaure = [minimalSquare](const TStackNode<
TSquare<int> > & sq) {
            return !(CalcSquare(sq.Data) < minimalSquare);
        };
        std::cout << "Number of figures with square greater or equal,
than " << minimalSquare << ": ";
        std::cout << std::count_if(st.begin(), st.end(), MatchSqaure)
<< std::endl;
        return 0;
}
```

6. Выводы

В ходе выполнения работы я узнал, как реализовать итераторы на языке C++, чтобы они были совместимы со стандартными функциями. Узнал про умные указатели std::unique_ptr, std::shared_ptr и в чём их разница. Изучил работу std::for_each и std::count_if и реализовал их взаимодействие с моей коллекцией.

Список литературы

1. *std::shared_ptr – cppreference.com*
   URL: https://en.cppreference.com/w/cpp/memory/shared_ptr
   (дата обращения 12.11.2020)
2. *Делаем свой итератор / Хабр — Habr.com*
   URL: https://habr.com/ru/post/265491/
   (дата обращения 12.11.2020)
3. *std::for_each – cppreference.com*
   URL: https://en.cppreference.com/w/cpp/algorithm/for_each
   (дата обращения 12.11.2020)
4. *std::count, std::count_if – cppreference.com*
   URL: https://en.cppreference.com/w/cpp/algorithm/count
   (дата обращения 12.11.2020)