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: Савченко Илья Владимирович
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Вариант 5:

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 5: Ромб, стек.

2. Описание программы

main.cpp	
struct Rhombus<T>	Фигура “ромб” (T - тип коорд.)
void FillRhombus(Rhombus<int>*, int, int)	Изменение высоты и ширины существующего ромба
void WriteRhombus(Rhombus<int>*)	Изменение ромба из стд. ввода
void PrintRhombus(const Rhombus<int>*)	Печать ромба
class CustomStack<T>	Стек
void CustomStack::Push(T*); T* CustomStack::Top(); T* CustomStack::Pop();	Основные функции стека
struct CustomStack<T>::iterator	Итератор на стеке, перегружает необходимые операторы
void CustomStack::insert(iterator&, T*); void CustomStack::erase(iterator&);	Функции добавления и удаления по итератору
iterator CustomStack::begin(); iterator CustomStack::end()	Возвращение итераторов на начало и конец
struct LessThan	Перегружен оператор скобок для подсчета кол-ва ромбов площадью меньше заданной
int main()	Драйвер

3. Набор тестов

Для ввода ромба указывается его полу-высота и полу-ширина

Центр ромба находится в центре координат.

Для ромбов

A: 1 2 (в ширину и 4 в высоту)

B: 2 3

C: 3 4

…

G: 7 8

(7 ромбов)

Кол-во ромбов площадью меньше 20 = 2 ромба

Ромб C имеет площадь 24

4. Результаты выполнения тестов

```
h - this message
p - push
r - remove by index
s - show
l - surface less than
x - exit
---
> p 1 2
> p 2 3
> p 3 4
> p 4 5
> p 5 6
> p 6 7
> p 7 8
> s
R: 7 8
R: 6 7
R: 5 6
R: 4 5
R: 3 4
R: 2 3
R: 1 2
---
> l 20
Less than 20 : 2 figs
```

5. Листинг программы

```
/**
 * Савченко И.В.
 * М8О-208Б-19
 * https://github.com/ShyFly46/oop_exercise_05
 *
 * Вариант 5:
 * Ромб, стек
 **/
```

```
#include<iostream>
#include<memory>    // smart_ptr
#include<iterator>
#include<cstddef>   // ptrdiff_t
#include<utility>    // pair
#include<algorithm> // for_each

using namespace std;

template<typename SCALAR>
struct Rhombus{
    using vertex_t = pair<SCALAR, SCALAR>;
    vertex_t a, b, c, d;
};

using MyFigure = Rhombus<int>;

void FillRhombus(Rhombus<int> *fig, int h, int w){
    if(h < 0) h = -h;
    if(w < 0) w = -w;

    fig->a.first = 0;
    fig->a.second = h;

    fig->b.first = w;
    fig->b.second = 0;

    fig->c.first = 0;
    fig->c.second = -h;

    fig->d.first = -w;
    fig->d.second = 0;
}

void WriteRhombus(Rhombus<int> *fig){
    int h, w;
    cin >> h >> w;
    FillRhombus(fig, h, w);
}

void PrintRhombusPtr(const MyFigure* fig){
    cout << "Rho: "
      << fig->a.second
        << " "
        << fig->b.first
        << "\n";
}
void PrintRhombus(const MyFigure& fig){
    cout << "R: "
      << fig.a.second
        << " "
        << fig.b.first
        << "\n";
}

template<class T>
class CustomStack{
```

```
    struct Node{
       unique_ptr<T> *p; // pointer
       Node *d; // down
       Node *u; // up
       Node(unique_ptr<T>* data_ptr, Node *next = nullptr)
         : p(data_ptr),
             d(next),
             u(nullptr)
       {}
       Node(T* data_ptr, Node *next = nullptr)
         : p(new unique_ptr<T>(data_ptr)),
             d(next),
             u(nullptr)
       {}

       ~Node(){
             delete p;
             if(d)
                   d->u = u;
             if(u)
                   u->d = d;
       }
    };

    Node *bottom,
        *top;
public:
    class iterator;
    CustomStack() : bottom(nullptr), top(nullptr) {}
    void Push(T *data){
       if(!top){
             bottom = new Node(data);
             top = bottom;
             return;
       }

       top->u = new Node(data, top);
       top = top->u;
    }

    T* Top(){
       if(!top)
             throw 1;
       return top->p->get();
    }

    T* Pop(){
       if(!top)
             throw 1;
       Node *toPop = top;
       top = top->d;
       if(top){
             top->u = nullptr;
       } else {
             bottom = nullptr;
       }
       T* toRet = toPop->p->release();
       toPop->u = nullptr;
```

```
        toPop->d = nullptr;
        delete toPop;
        return toRet;
    }

    void insert(iterator& i, T* data){
        if(!i)
              throw 3;
        Node* u = i.m_node;
        Node* d = u->d;
        Node* newN = new Node(data);
        newN->u = u;
        newN->d = d;
        u->d = newN;
        if(d)
              d->u = newN;
    }

    void erase(iterator& i){
        if(!i)
              throw 3;
        iterator copy = i;
        ++copy;
        if(i == top)
              top = i.m_node->d;
        if(i == bottom)
              bottom = i.m_node->u;
        delete i.m_node;
        i = copy;
    }

    bool IsEmpty(){
        return top;
    }

    iterator begin() {return iterator(top);}
    iterator end()   {return iterator(nullptr);}
};

template<typename T>
struct CustomStack<T>::iterator{
    friend class CustomStack<T>;

      using iterator_category = std::forward_iterator_tag;
      using difference_type   = std::ptrdiff_t;
      using value_type        = T;
      using pointer           = T*;  // or also value_type*
      using reference         = T&;  // or also value_type&

    iterator(Node *node) : m_node(node) {}

    reference operator*() {return *(m_node->p->get());}
    pointer operator->() {return m_node->p;}
    iterator& operator++(){
        if(!m_node)
              throw 2;
        m_node = m_node->d;
        return *this;
```

```
    }
    iterator operator++(int){
       if(!m_node)
             throw 2;
       iterator toRet(m_node);
       m_node = m_node->d;
       return toRet;
    }
    operator pointer() {return m_node->p->get();}
    operator bool() {return m_node;}
    friend bool operator==(const iterator& a, const iterator& b)
       {return a.m_node == b.m_node;}
    friend bool operator!=(const iterator& a, const iterator& b)
       {return a.m_node != b.m_node;}
private:
    Node *m_node;
};

using CuStack = CustomStack<MyFigure>;

struct LessThan{
    int than;
    LessThan(int than) : than(than) {}
    bool operator()(const MyFigure& fig){
       int w = fig.a.second;
       int h = fig.b.first;
       int sur = w * h * 2;
       return sur < than;
    }
};

void help(){
    cout << "h - this message\n"
      << "p - push\n"
        << "r - remove by index\n"
        << "s - show\n"
        << "l - surface less than\n"
        << "x - exit\n"
        << "---\n";
}

int main(){
    CuStack myStack;
    MyFigure *rh_ptr;
    LessThan lt(0);

    char command;

    help();

    cout << "> ";
    while(cin >> command){
       switch(command){
             case 'p': // push
                   rh_ptr = new MyFigure;
                   WriteRhombus(rh_ptr);
                   myStack.Push(rh_ptr);
                   break;
```

```
            case 'r':
            {
                  unsigned int index;
                  cin >> index;
                  CuStack::iterator del = myStack.begin();
                  try{
                        for(; index > 0; --index)
                              ++del;
                        myStack.erase(del);
                  } catch (int e){
                        cout << "Failed to delete\n";
                  }
                  break;
            }
            case 's':
                  for_each(myStack.begin(), myStack.end(), PrintRhombus);
                  cout << "---\n";
                  break;
            case 'h':
                  help();
                  break;
            case 'l':
                  cin >> lt.than;
                  cout << "Less than "
                       << lt.than
                         << " : "
                         << count_if(myStack.begin(), myStack.end(), lt)
                         << " figs\n";
                  break;
            case 'x':
                  return 0;
            default:
                  break;
      }
      cout << "> ";
   }

}
```

Список литературы

1. Writing a custom iterator in modern C++ [Электронный ресурс]. URL: https://www.internalpointers.com/post/writing-custom-iterators-modern-cpp
   (дата обращения 20.04.2021)