**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: итераторы

Студент: Баранников Степан Алексеевич

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата: 25.11.2019

Оценка:

Москва, 2019

Постановка задачи

Создать шаблон динамической коллекцию, согласно варианту задания:

Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr).

(Опционально использование std::unique\_ptr;)

В качестве параметра шаблона коллекция должна принимать тип данных;

Коллекция должна содержать метод доступа:

Стек – pop, push, top;

Очередь – pop, push, top;

Список, Динамический массив – доступ к элементу по оператору [];

Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти – является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);

Коллекция должна использовать аллокатор для выделеления и освобождения памяти для своих элементов.

Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально – vector).

Реализовать программу, которая:

Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор;

Позволяет удалять элемент из коллекции по номеру элемента;

Выводит на экран введенные фигуры c помощью std::for\_each;

Вариант 2: Квадрат, Стек, Список.

Описание программы

Шаблон квадрата написан в Shape.h, вместе шаблонными функциями Area, Centre и Print, которые ищут его площадь, центр фигуры и печатают его соответственно.

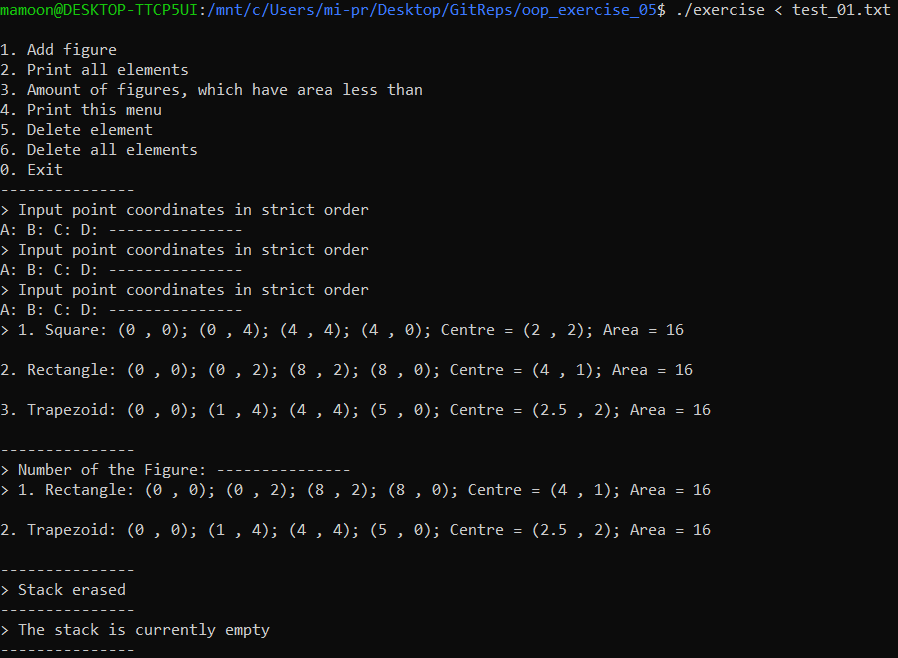
Шаблон стека расписан в Stack.h. Он выполняет все стандартные для стека функции (pop, push, top). В нем также расписан собственный итератор forward\_iterator для шаблонных функций, работающих с итераторами (std::count\_if, std::for\_each). В шаблон передается тип аллокатора, по умолчанию берется std::allocator.

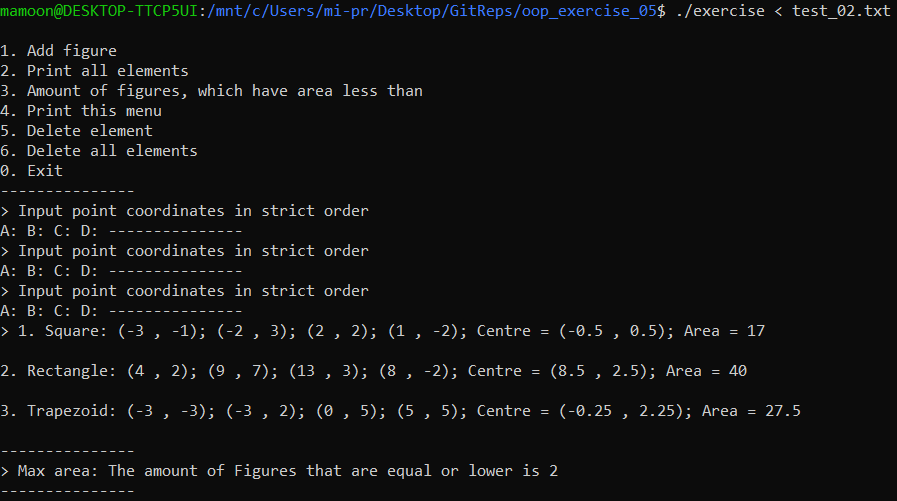
В Allocator.h содержится аллокатор, минимально соответствующий чертам аллокатора, заданым в стандарте. В нем реализованы стандартный конструктор и конструктор из аллокатора, шаблонная структура rebind, позволяющая перешивать аллокатор на другой тип, методы allocate и deallocate, выделяющие и отпускающие память, contruct и destroy, создающие и уничтожающие типы.

В main.cpp содержится меню, позволяющее работать со стеком, держащий в себе тип Square, тип точек которого int.

ССЫЛКА НА GITHUB: <https://github.com/mamongo/oop_exercise_06>

1. Набор testcasestest\_01.txt  
   1  
   0 0 0 4 4 4 4 0  
   1  
   0 0 0 2 8 2 8 0  
   1  
   0 0 1 4 4 4 5 0  
   2  
   5  
   1  
   2  
   6  
   2  
   0  
   test\_02.txt  
   1  
   -3 -1 -2 3 2 2 1 -2  
   1  
   4 2 9 7 13 3 8 -2  
   1  
   -3 -3 -3 2 0 5 5 5  
   2  
   3  
   30  
   0
2. Результаты выполнения тестов.

./exercise < test\_01.txt  


./exercise < test\_02.txt  


1. Листинг программы

Shapes.h

|  |
| --- |
| #ifndef SHAPE\_H #define SHAPE\_H  #include <iostream> #include <utility> #include <cmath>  typedef std::pair<double, double> DoublePoint;  template <typename T> class Figure { public:  typedef std::pair<T, T> Point;   bool Square = false, Rectangle = false, Trapezoid = false;  bool Abstract = false;   Figure() : centre(DoublePoint(0,0)), area(0)  {  for (int i = 0; i < 4; ++i)  p[i].first = p[i].second = 0;  }   DoublePoint centre;  double area{ 0 };  Point p[4]; };  bool DoubleEqual(double lhs, double rhs){  const double EPS = 0.00001;  if (lhs > rhs)  return (lhs - rhs) < EPS ? true : false;  else  return (rhs - lhs) < EPS ? true : false; }  bool IsRight(std::pair<double, double> a, std::pair<double, double> b, std::pair<double, double> c){  std::pair<double, double> vec1 = { b.first - a.first, b.second - a.second };  std::pair<double, double> vec2 = { c.first - a.first, c.second - a.second };   double result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (DoubleEqual(result, 0)) return true;  return false; }  bool IsRight(std::pair<int, int> a, std::pair<int, int> b, std::pair<int, int> c){  std::pair<int, int> vec1 = { b.first - a.first, b.second - a.second };  std::pair<int, int> vec2 = { c.first - a.first, c.second - a.second };   int result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (result == 0) return true;  return false; }  //----------------------------------------------------------------------------------------------------------- template <class T> inline double distance(std::pair<T, T> a, std::pair<T, T> b){  return (b.first - a.first) \* (b.first - a.first) + (b.second - a.second) \* (b.second - a.second); }   template <class T> bool IsRectangle(std::pair<T, T> p[4]){  std::pair<T, T> null(0, 0);  if (p[0] == null && p[1] == null && p[2] == null && p[3] == null) return false;   if (  IsRight(p[2], p[1], p[3]) &&  IsRight(p[3], p[2], p[0]) &&  IsRight(p[1], p[0], p[2]) &&  IsRight(p[0], p[3], p[1])  ) return true;  return false; } //----------------------------------------------------------------------------------------------------------- template <class T> bool IsSquare(std::pair<T, T> p[4]){  if (  IsRectangle(p) &&  DoubleEqual(distance(p[0], p[1]), distance(p[1], p[2])) &&  DoubleEqual(distance(p[1], p[2]), distance(p[2], p[3])) &&  DoubleEqual(distance(p[2], p[3]), distance(p[3], p[0])) &&  DoubleEqual(distance(p[3], p[0]), distance(p[0], p[1]))  ) return true;  return false; } //----------------------------------------------------------------------------------------------------------- template <class T> bool IsTrapezoid(std::pair<T, T> p[4]){  std::pair<T, T> null(0, 0);  if (p[0] == null && p[1] == null && p[2] == null && p[3] == null) return false;   if (  !DoubleEqual(distance(p[1], p[2]), distance(p[3], p[0])) &&  DoubleEqual(distance(p[0], p[1]), distance(p[2], p[3])) &&  DoubleEqual(distance(p[0], p[2]), distance(p[1], p[3]))  ) return true;  return false; } //-----------------------------------------------------------------------------------------------------------  //----------------------------------------------- template <class T> DoublePoint Centre(Figure<T>& fig){  if(fig.Square || fig.Rectangle || fig.Trapezoid){  DoublePoint res(0, 0);  for (int i = 0; i < 4; ++i)  {  res.first += fig.p[i].first;  res.second += fig.p[i].second;  }  res.first /= 4;  res.second /= 4;  return res;  } }  //----------------------------------------------- template <class T> double Area(Figure<T>& fig){  return abs(fig.p[0].first \* fig.p[1].second + fig.p[1].first \* fig.p[2].second + fig.p[2].first \* fig.p[3].second + fig.p[3].first \* fig.p[0].second - fig.p[1].first \* fig.p[0].second - fig.p[2].first \* fig.p[1].second - fig.p[3].first \* fig.p[2].second - fig.p[0].first \* fig.p[3].second)/2; }  //----------------------------------------------------------------------------------------------------------- template <class T> std::ostream& operator << (std::ostream& os, const std::pair<T, T>& p) {  os << '(' << p.first << " , " << p.second << ')';  return os; }  template <class T> std::ostream& operator << (std::ostream& os, Figure<T>& fig) {  if (fig.Abstract)  os << "Abstract: ";  else if(fig.Trapezoid)  os << "Trapezoid: ";  else if(fig.Rectangle)  os << "Rectangle: ";  else if(fig.Square)  os << "Square: ";  os << fig.p[0] << "; " << fig.p[1] << "; " << fig.p[2] << "; " << fig.p[3] << "; Centre = " << fig.centre << "; Area = " << fig.area << std::endl; }  //----------------------------------------------------------------------------------------------------------- bool operator == (std::pair<int, int> lhs, std::pair<int, int> rhs) {  return lhs.first == rhs.first && lhs.second == rhs.second; }  bool operator == (std::pair<double, double> lhs, std::pair<double, double> rhs) {  return DoubleEqual(lhs.first, rhs.first) && DoubleEqual(lhs.second, rhs.second); }  template <class T> std::istream& operator >> (std::istream& is, Figure<T>& fig) {  cout << "Input point coordinates in strict order\nA: ";  is >> fig.p[0].first >> fig.p[0].second;  cout << "B: ";  is >> fig.p[1].first >> fig.p[1].second;  cout << "C: ";  is >> fig.p[2].first >> fig.p[2].second;  cout << "D: ";  is >> fig.p[3].first >> fig.p[3].second;   if(IsSquare(fig.p))  fig.Square = true;  else if(IsRectangle(fig.p))  fig.Rectangle = true;  else if(IsTrapezoid(fig.p))  fig.Trapezoid = true;  else   fig.Abstract = true;   fig.centre = Centre(fig);  fig.area = Area(fig);  return is; }  #endif |

Stack.h

|  |
| --- |
| #ifndef STACK\_H #define STACK\_H  #include <memory> #include <exception> #include <algorithm>  template<class T> class Stack { public:  Stack()  {  container = nullptr;  \_size = capacity = 0;  }   Stack(Stack&& other)  {  capacity = other.capacity;  \_size = other.\_size;  container = other.container;  other.container = nullptr;  }   ~Stack() { container = nullptr; }   class forward\_iterator {  public:  //The next using's actually make our pointer compatible with pointer templates  using iterator\_category = std::forward\_iterator\_tag;  using difference\_type = std::ptrdiff\_t;  using value\_type = T ;  using pointer = T\*;  using reference = T&;   forward\_iterator() { value = nullptr; }  forward\_iterator(pointer arg) { value = arg; }  ~forward\_iterator() { };   forward\_iterator& operator = (pointer arg)  {  value = arg;  return \*this;  }   forward\_iterator& operator ++ ()  {  ++value;  return \*this;  }   forward\_iterator operator ++ (int)  {  return value++;  }   T& operator \*()  {  return \*value;  }   bool operator == (const forward\_iterator& other) const  {  return value == other.value ? 1 : 0;  }   bool operator != (const forward\_iterator& other) const  {  return value != other.value ? 1 : 0;  }   pointer value;  };   size\_t size(void) const  {  return \_size;  }   bool empty(void) const  {  return \_size == 0;  }    forward\_iterator begin(void) const  {  return forward\_iterator(container.get());  }   forward\_iterator end(void) const  {  return forward\_iterator(container.get() + \_size);  }    Stack& push(T value)  {  insert(\_size, value);  return \*this;  }   Stack& pop(void)  {  erase(\_size - 1);  return \*this;  }   T top(void)  {  if (!empty())  return \*(container.get() + \_size - 1);  else throw std::logic\_error("Stack is empty");  }   void insert(size\_t index, T val)  {  if (index > \_size || index < 0) throw std::logic\_error("Outside the bounds");  if (capacity == \_size)  {  if (capacity == 0)  {  container.reset(new T);  capacity = 1;  }  else   {  capacity \*= 2;  T\* temp = new T[capacity];  std::copy(container.get(), container.get() + \_size, temp);  container.reset(temp);  }  }   for (size\_t i = \_size; i != index; --i)  \*(container.get() + i) = \*(container.get() + i - 1);  \*(container.get() + index) = val;  ++\_size;  }   void erase(size\_t index)  {  if (\_size == 0) throw std::logic\_error("Stack is empty");  else if (index >= \_size || index < 0) throw std::logic\_error("Outside the bounds");  for (size\_t i = index + 1; i != \_size; ++i)  \*(container.get() + i - 1) = \*(container.get() + i);  --\_size;  }   void erase()  {  capacity = \_size = 0;  container = nullptr;  }  private:  std::shared\_ptr<T> container;  size\_t \_size, capacity; };  template<class T> void swap(Stack<T>& lhs, Stack<T>& rhs) {  std::swap(lhs, rhs); //or use std::move semantics }  #endif |

main1.cpp

|  |
| --- |
| /\* 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; 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:  o Стек - pop, push, top; o Очередь - pop, push, top;  o Список, Динамический массив - доступ к элементу по оператору []; 10. Реализовать программу, которая:  o Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;  o Позволяет удалять элемент из коллекции по номеру элемента; o Выводит на экран введенные фигуры c помощью std::for\_each;  o Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if);  Вариант 18: Квадрат, Прямоугольник, Трапеция.  Баранников Степан Алексеевич M8O-201Б \*/ #include <iostream> #include <string> #include <exception> using namespace std; #include "Shape.h" #include "Stack.h"  struct stack\_empty : public std::exception {  const char\* what() const throw ()  {  return "The stack is currently empty";  } };  void PrintStack(const Stack< Figure<double> >& stack) {  int num = 1;  for\_each(stack.begin(), stack.end(), [&num](auto& e)  {  cout << num << ". " << e << endl;  ++num;  }); }  int GetNum(void) {  int k;  bool got = false;  while (!got)  {  cin >> k;  if (!cin.good())  {  cout << "Bad input ignored: " << flush;  cin.clear();  cin.ignore(256, '\n');  }  else got = true;  }  return k; }  void print\_menu(){  cout  << "\n1. Add figure\n"  << "2. Print all elements\n"  << "3. Amount of figures, which have area less than\n"  << "4. Print this menu\n"  << "5. Delete element\n"  << "6. Delete all elements\n"  << "0. Exit\n"  << flush; }   int main() {  Stack< Figure<double> > fig\_stack;  Figure<double> buff;  int k = 1;  size\_t num;  double MaxArea;  print\_menu();  do{  cout << "---------------\n> ";  try{  k = GetNum();  if(!cin) throw runtime\_error("Failed to get input");  switch (k){  case 1:  cin >> buff;  fig\_stack.push(buff);  k = 1;  break;  case 2:  if (fig\_stack.empty()) throw stack\_empty();  PrintStack(fig\_stack);  break;  case 3:  if (fig\_stack.empty()) throw stack\_empty();  cout << "Max area: ";  cin >> MaxArea;  cout << "The amount of Figures that are equal or lower is " << count\_if( fig\_stack.begin(), fig\_stack.end(), [&MaxArea](auto& fig) -> bool  {  return ( !fig.Abstract && (fig.area < MaxArea || DoubleEqual(fig.area, MaxArea)) );  }  ) << endl;  break;  case 4:  print\_menu();  break;  case 5:  if (fig\_stack.empty()) throw stack\_empty();  cout << "Number of the Figure: ";  num = GetNum();  if (num < 1 || num > fig\_stack.size()) throw runtime\_error("Incorrect index");  fig\_stack.erase(num - 1);  break;  case 6:  fig\_stack.erase();  cout << "Stack erased" << endl;  break;  case 0:  break;  default:  cout << "No such number" << endl;  }  }   catch (stack\_empty& err)  {  cout << err.what() << endl;  }  catch (runtime\_error& err)  {  cerr << err.what() << endl;  }  catch (logic\_error & err)  {  cerr << "In stack: " << err.what() << endl;  }  catch (...)  {  cerr << "Exception thrown, but undefined. Please open an issue at GitHub with steps to reproduce ." << endl;  }  } while (k);  return 0; } |

1. Вывод:

Программа умеет находить площадь любого Четырехугольника, и если нет заданного по заданию четырехугольника, то программа выводит данные с подписью `Abstract:`, в иных случаях все сделано по заданию.

Обычно, собственно реализованные структуры данных не имеют

итераторов, что не позволяет использовать их вместе со стандартными

шаблонными функциями. Создание для них собственного итератора

позволяет обойти это.

## Список литературы

* Справочник по языке C++ [Электронный ресурс]. URL: <https://en.cppreference.com/w/cpp/memory/shared_ptr> (дата обращения: 08.11.2019).