МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Структура хранения данных: Мультистек»**

**Выполнил:** студент группы 381903-3

Пасека Иван Евгеньевич

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Содержание

[1. Введение 3](#_Toc536638418)

[2. Цели и задачи 4](#_Toc536638419)

[2.1. Используемые инструменты 4](#_Toc536638420)

[3. Руководство пользователя 5](#_Toc536638421)

[4. Руководство программиста 6](#_Toc536638422)

[4.1. Описание структуры программы 6](#_Toc536638423)

[4.2. Описание функций и процедур, их алгоритмов 6](#_Toc536638424)

[5. Эксперименты 9](#_Toc536638425)

[6. Заключение 10](#_Toc536638426)

[7. Литература 11](#_Toc536638427)

[8. Приложения 12](#_Toc536638428)

[8.1. Приложение 1:Класс TExсeption 12](#_Toc536638429)

[8.2. Приложение 2:Класс TStack 12](#_Toc536638430)

[8.3. Приложение 3:Класс TNewStack 15](#_Toc536638431)

[8.4. Приложение 4:Класс TMStack 16](#_Toc536638432)

[8.5. Приложение 5:Код программы тестирования и экспериментов 19](#_Toc536638433)

[8.6. Приложение 6:Тесты для классов 20](#_Toc536638434)

# 1. Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Мультистек.

Стек (англ.stack– стопка) – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала. В стеках используется метод доступа к элементам LIFO ( Last Input – First Output, "последним пришел – первым вышел"). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно сначала взять верхнюю.

Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется вершиной стека. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека. Например, если записаны в стек числа 1, 2, 3, то при последующем извлечении получим 3,2,1.

Разработка класса Мультистек обеспечит работу с несколькими стеками.

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – мультистек и выполнение основных операций над ней:

* добавления элементов в мультистек,
* извлечения элементов из мультистека (с удалением),
* проверка мультистека на полноту/пустоту,

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация класса TNewStack, построенного на основе существующего класса TStack.
2. Реализация класса TMStack, построенного на основе класса TNewStack.
3. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
4. Обеспечение работоспособности тестов и примера использования.
5. Реализация заготовок тестов, покрывающих все методы созданных классов.
6. Модификация примера использования в тестовое приложение, позволяющее задавать мультистеки и осуществлять основные операции над ними.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

Запускаем программу из файла sample\_tmultistack.cpp. Программа выведет пример корректной работы основных операций с мультистеком . (Рис.1):

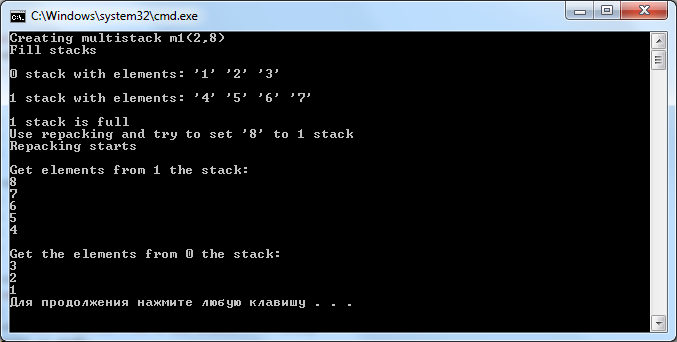


Рис.1.Пример операций с мультистеком.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано 4 класса:

* Класс «Стек» (TStack), реализованный с использованием массива.
* Класс TNewStack, построенного на основе класса TStack.
* Класс «Мультистек» (TMStack), построенного на основе класса TNewStack.
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

* tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
* tnewstack.h– модуль с классом TNewStack, в котором определен интерфейс и реализация его методов.
* tmultistack.h– модуль с классом TMStack, в котором определен интерфейс шаблонного класса Мультистек и реализация его методов.
* exсeption.h – модуль с классом исключения TExсeption.
* sample\_tmultistack.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
* test\_main.cpp, test\_tstack.cpp, test\_ tmultistack.cpp – модуль с функциями тестирования для созданных классов. Содержит 20 тестов для класса TStack и 12 тестов для класса TMStack.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TStack:

template <typename StackType>

class TStack

1. TStack(int n = 0) - конструктор класса, принимающий размер стека. По умолчанию создается стек размера 0 с позицией вершины стека 0.

2. TStack(TStack<StackType> &S) - конструктор копирования.

Принимает ссылку на объект класса TStack.

3. ~TStack() - деструктор. Освобождает выделенную под вектор память.

4. TStack& operator=(const TStack<StackType>& stack) – перегрузка оператора присваивания одного стека другому.

Присваивает полям первого объекта класса поля второго объекта класса.

5. int GetSize() - возвращает размер стека.

6. StackType Get() – метод изъятия элемента из вершины стека с удалением.

7. void Put(StackType A) - метод, позволяющий добавить новый элемент в стек.

8. bool IsFull() – метод проверки стека на полноту.

9. bool IsEmpty() – метод проверки стека на пустоту.

10. void PrintStack() - метод отображения текущих элементов стека.

11. bool operator==(const TStack<StackType>& stack) const – перегрузка оператора сравнения.

Выполняется проверка стеков на равенство. Возвращает true, если равенство выполняется, false в противном случае.

12. bool operator!=(const TStack<StackType>& stack) const - перегрузка оператора сравнения.

Выполняется проверка стеков на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию методов шаблонного класса TNewStack:

template <typename NewStackType>

class TNewStack :public TStack<NewStackType>

1. TNewStack(int \_size = 0, NewStackType\* \_memory = 0) - конструктор класса.

2. TNewStack(TNewStack <NewStackType> &NS) - конструктор копирования.

Принимает ссылку на объект класса TNewStack.

3. int GetFreeMemory() – метод для получения свободной памяти (количество свободных позиций)

4. int GetSize() – метод, возвращающий размер.

5. int GetTop() – метод, возвращающий первый элемент.

6. void SetMemory(int \_size, NewStackType\* \_memory) – метод выделении памяти под массив.

7. void Put(NewStackType \_A) – метод добавления элемента.

8. NewStackType Get() – метод изъятия элемента из вершины стека с удалением.

Рассмотрим реализацию методов шаблонного класса TMStack:

template <typename MultiStackType>

class TMStack

1. TMStack(int \_n, int \_size) - конструктор класса, принимающий количество стеков и общий размер мультистека.

2. TMStack(TMStack<MultiStackType> &A) – конструктор копирования.

Принимает ссылку на объект класса TMStack.

3. int GetSize(){ return size; } - возвращает размер стека мультистека.

4. MultiStackType Get(int \_n) - метод изъятия элемента из вершины стека с удалением.

5. void Set(int \_n, MultiStackType p) - метод, позволяющий добавить новый элемент в один из стеков мультистека. Принимает индекс стека и добавляемый элемент.

6. bool IsFull(int \_n) - метод проверки мультистека на полноту.

7. bool IsEmpty(int \_n) – метод проверки мультистека на пустоту.

8. int GetFreeMemory() – метод оценки объема свободной памяти.

9. void Repack(int k) – перепаковка памяти - динамическое распределение памяти путем переписи части хранимых значений в другую область ( другой стек ).

# 5. Эксперименты

В качестве примера рассмотрим перегрузку оператора присвоения для класса мультистека (TMStack).

Теоретическая сложность выполнения алгоритма O(1).

Мы провели измерение присваивая мультистеку разное количество элементов: 10, 100, …, 100000 элементов. Ниже вы можете увидеть график зависимости времени выполнения операции присвоения от количества элементов мультистека. (Рис.2) По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

Рис.2. График зависимости времени выполнения операции присвоения от размера мультистека.

По горизонтали – количество присваиваемых элементов.

По вертикали - время выполнения программы.

# 6. Заключение

В результате лабораторной работы была разработана структура данных - мультистек, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Созданный шаблонный класс был протестированы с использованием Google Tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на методе класса.

# 7. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Тестирование с использованием Google Test

(http://neerc.ifmo.ru/wiki/index.php?title=Тестирование\_с\_использованием\_Google\_Test#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D1.8F\_main.28.29)

1. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
2. https://ru.wikipedia.org/wiki/Стек

# 8. Приложения

## 8.1. Приложение 1:Класс TExсeption

|  |
| --- |
| **exception.h** |
| #pragma once  #include <iostream>  #include <string>  class TException  {  private:  std::string str;  public:  TException(std::string \_str);  void Show();  };  TException::TException(std::string \_str) : str(\_str) {}  void TException::Show()  {  std::cout << "\nWarning! \nMessage: " << str << std::endl;  } |

## 8.2. Приложение 2:Класс TStack

|  |
| --- |
| **tstack.h** |
| #pragma once  #include <iostream>  #include "exception.h"  using namespace std;  template <typename StackType>  class TStack  {  protected:  int size;  int top;  StackType\* memory;  public:  int GetSize() { return size; }  TStack(int n = 0);  TStack(TStack<StackType> &S);  TStack& operator=(const TStack<StackType>& stack);  ~TStack() { delete[] memory; }  StackType Get();  void Put(StackType A);  bool IsFull();  bool IsEmpty();  void PrintStack();  bool operator!=(const TStack<StackType>& stack) const;  bool operator==(const TStack<StackType>& stack) const;  };  //----------------------------------------------------------------------  template <class StackType>  TStack <StackType> ::TStack(int n)  {  if ( n < 0 )  throw TException(" Negative size.");  else if ( n == 0 )  {  size = 0;  top = 0;  memory = NULL;  }  else  {  size = n;  top = 0;  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = 0;  }  }  //----------------------------------------------------------------------  template <class StackType>  TStack <StackType> ::TStack(TStack <StackType> &S)  {  size = S.size;  top = S.top;  if ( size == 0 )  memory = NULL;  else  {  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = S.memory[i];  }  }  //----------------------------------------------------------------------  template <class StackType>  TStack<StackType>& TStack<StackType> ::operator=(const TStack<StackType>& stack)  {  if ( this != &stack )  {  top = stack.top;  if ( size != stack.size )  {  size = stack.size;  delete[] memory;  memory = new StackType[size];  }  for ( int i = 0; i < size; i++ )  {  memory[i] = stack.memory[i];  }  }  return \*this;  }  //----------------------------------------------------------------------  template <class StackType>  void TStack<StackType> ::Put(StackType A)  {  if ( IsFull() )  {  throw TException("Stack is full");  }  else  {  memory[top] = A;  top++;  }  }  //----------------------------------------------------------------------  template <class StackType>  StackType TStack<StackType> ::Get()  {  if ( IsEmpty() )  throw TException("Stack is empty");  else  {  top--;  return memory[top];  }  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::IsFull()  {  //ternary operator  return top >= size ? true : false;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::IsEmpty()  {  return !top;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::operator==(const TStack<StackType>& stack) const  {  if ( top != stack.top || size != stack.size )  return false;  for ( int i = 0; i < top; i++ )  {  if ( memory[i] != stack.memory[i] )  return false;  }  return true;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::operator!=(const TStack<StackType>& stack) const  {  return !( \*this == stack );  }  //----------------------------------------------------------------------  template <class StackType>  void TStack<StackType>:: PrintStack()  {  for ( int i = top-1; i >= 0; i-- )  cout << "\t|" << memory[i] << "|" << endl;  } |

## 8.3. Приложение 3:Класс TNewStack

|  |
| --- |
| **tnewstack.h** |
| #pragma once  #include <iostream>  #include "tstack.h"  #include "exception.h"  template <typename NewStackType>  class TNewStack :public TStack<NewStackType>  {  public:  TNewStack(int \_size = 0, NewStackType\* \_memory = 0);  TNewStack(TNewStack <NewStackType> &NS);  int GetFreeMemory();  int GetSize();  int GetTop();  void SetMemory(int \_size, NewStackType\* \_memory);  void Put(NewStackType \_A);  NewStackType Get();  };  //--------------------------------------------------------------------------  template <typename NewStackType>  TNewStack<NewStackType>::TNewStack(int \_size, NewStackType\* \_memory)  {  TStack<NewStackType>::size = \_size;  TStack<NewStackType>::top = 0;  if (TStack<NewStackType>::memory != NULL)  delete [] TStack<NewStackType>::memory;  TStack<NewStackType>::memory = \_memory;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  TNewStack<NewStackType>::TNewStack(TNewStack <NewStackType> &NS)  {  TStack<NewStackType>::size = NS.TStack<NewStackType>::size;  TStack<NewStackType>::top = NS.TStack<NewStackType>::top;  if (TStack<NewStackType>::memory != NULL)  delete [] TStack<NewStackType>::memory;  TStack<NewStackType>::memory = NS.TStack<NewStackType>::memory;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  int TNewStack<NewStackType>::GetFreeMemory()  {  return (TStack<NewStackType>::size - TStack<NewStackType>::top);  }  //--------------------------------------------------------------------------  template <typename NewStackType>  int TNewStack<NewStackType>::GetSize()  {  return TStack<NewStackType>::size;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  int TNewStack<NewStackType>::GetTop()  {  return TStack<NewStackType>::top;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  void TNewStack<NewStackType>::SetMemory(int \_size, NewStackType\* \_memory)  {  TStack<NewStackType>::size = \_size;  TStack<NewStackType>::memory = \_memory;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  void TNewStack<NewStackType>::Put(NewStackType \_A)  {  TStack<NewStackType>::memory[TStack<NewStackType>::top] = \_A;  TStack<NewStackType>::top++;  }  //--------------------------------------------------------------------------  template <typename NewStackType>  NewStackType TNewStack<NewStackType>::Get()  {  TStack<NewStackType>::top--;  return TStack<NewStackType>::memory[TStack<NewStackType>::top];  } |

## 8.4. Приложение 4:Класс TMStack

|  |
| --- |
| **tmultistack.h** |
| #pragma once  #include "exception.h"  #include "tnewstack.h"  template <typename MultiStackType>  class TMStack  {  protected:  int size;  MultiStackType\* memory;  int n;  TNewStack<MultiStackType>\*\* newS;  int GetFreeMemory();  void Repack(int k);  public:  TMStack(int \_n, int \_size);  TMStack(TMStack<MultiStackType> &A);  int GetSize() { return size; }  MultiStackType Get(int \_n);  void Set(int \_n, MultiStackType p);  bool IsFull(int \_n);  bool IsEmpty(int \_n);  };  //--------------------------------------------------------------------------  template <typename MultiStackType>  TMStack<MultiStackType> ::TMStack(int \_n, int \_size)  {  if (\_n <= 0 || \_size <= 0)  throw TException ("Negative leight.");  n = \_n;  size = \_size;  newS = new TNewStack<MultiStackType> \*[n];  memory = new MultiStackType[size];  int el\_size = 0;  for (int i = 0; i < n; i++)  {  el\_size = size / n;  if (i == n -1) el\_size += size % n;  newS[i] = new TNewStack<MultiStackType>(el\_size, &memory[i\*size/n]);  }  }  //--------------------------------------------------------------------------  template <typename MultiStackType>  TMStack<MultiStackType> ::TMStack(TMStack &A)  {  size = A.size;  n = A.n;  memory = new MultiStackType[size];  newS = new TNewStack<MultiStackType> \*[n];  for (int i = 0; i < size; i++)  memory[i] = A.memory[i];  int\* smth = new int[n];  for (int i = 0; i < n; i++)  smth[i] = A.newS[i]->TNewStack<MultiStackType>::GetSize();  int countingSize = 0;  newS[0] = new TNewStack<MultiStackType>(\*(A.newS[0]));  newS[0]->TNewStack<MultiStackType>::SetMemory(smth[0], memory);  for (int i = 1; i < n; i++)  {  newS[i] = new TNewStack<MultiStackType>(\*(A.newS[i]));  int temp = smth[0] + countingSize;  newS[i]->TNewStack<MultiStackType>::SetMemory(smth[i], &memory[temp]);  countingSize += smth[i];  }  }  //--------------------------------------------------------------------------  template<typename MultiStackType>  void TMStack<MultiStackType>::Set(int \_n, MultiStackType p)  {  if (IsFull(\_n))  Repack(\_n);  newS[\_n]->Put(p);  }  //--------------------------------------------------------------------------  template<typename MultiStackType>  MultiStackType TMStack<MultiStackType>::Get(int \_n)  {  if (IsEmpty(\_n))  throw TException("Empty");  return newS[\_n]->Get();  }  //--------------------------------------------------------------------------  template<typename MultiStackType>  bool TMStack<MultiStackType>::IsFull(int \_n)  {  return (newS[\_n]->GetFreeMemory() <= 0);  }  //--------------------------------------------------------------------------  template<typename MultiStackType>  bool TMStack<MultiStackType>::IsEmpty(int \_n)  {  return (newS[\_n]->GetFreeMemory() == newS[\_n]->GetSize());  }  //--------------------------------------------------------------------------  template <typename MultiStackType>  int TMStack<MultiStackType>::GetFreeMemory()  {  int count = 0;  for (int i = 0; i < n; i++)  {  count += newS[i]->GetFreeMemory();  }  return count;  }  //--------------------------------------------------------------------------  template <typename MultiStackType>  void TMStack<MultiStackType>::Repack(int k)  {  cout << "Repacking starts" << endl;  int FreeForNow = GetFreeMemory();  if ( FreeForNow == 0)  throw TException("Error");  int eq\_add = FreeForNow / n;  int full\_add = FreeForNow % n;  int\* sizeNewOne = new int[n];  MultiStackType\*\* startNewOne = new MultiStackType\*[n];  MultiStackType\*\* startOldOne = new MultiStackType\*[n];  for (int i = 0; i < n; i++)  sizeNewOne[i] = eq\_add + newS[i]->GetTop();  sizeNewOne[k] += full\_add;  startNewOne[0] = startOldOne[0] = memory;  for (int i = 1; i < n; i++)  {  startNewOne[i] = startNewOne[i - 1] + sizeNewOne[i - 1];  startOldOne[i] = startOldOne[i - 1] + newS[i - 1]->GetSize();  }  for (int i = 0; i < n; i++)  {  if (startNewOne[i] <= startOldOne[i])  for (int j = 0; j < newS[i]->GetTop(); j++)  startNewOne[i][j] = startOldOne[i][j];  else  {  int s = i + 1;  for (s; s < n; s++)  if (startNewOne[s] <= startOldOne[s])  break;  for (int j = s - 1; j >= i; j--)  for (int r = newS[j]->GetTop() - 1; r >= 0; r--)  startNewOne[j][r] = startOldOne[j][r];  i = s - 1;  }  }  for (int i = 0; i < n; i++)  newS[i]->SetMemory(sizeNewOne[i], startNewOne[i]);  delete[] sizeNewOne;  delete[] startNewOne;  delete[] startOldOne;  } |

## 8.5. Приложение 5:Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_tmultistack.cpp** |
| #include <iostream>  #include "tmultistack.h"  using namespace std;  int main()  {  cout << "Creating multistack m1(2,8)" << endl;  TMStack<int> m1(2, 8);  cout << "Fill stacks" << endl;  cout << endl << "0 stack with elements: '1' '2' '3'" << endl;  m1.Set(0, 1);  m1.Set(0, 2);  m1.Set(0, 3);  cout << endl << "1 stack with elements: '4' '5' '6' '7'" << endl;  m1.Set(1, 4);  m1.Set(1, 5);  m1.Set(1, 6);  m1.Set(1, 7);  cout << endl << " check 1 stack is full" << endl;  cout << "Use repacking and try to set '8' to 1 stack" << endl;  m1.Set(1, 8);  cout << endl << "Get elements from 1 the stack: " << endl;  cout << m1.Get(1) << endl;  cout << m1.Get(1) << endl;  cout << m1.Get(1) << endl;  cout << m1.Get(1) << endl;  cout << m1.Get(1) << endl;  cout << endl << "Get the elements from 0 the stack: " << endl;  cout << m1.Get(0) << endl;  cout << m1.Get(0) << endl;  cout << m1.Get(0) << endl;  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "time.h"  #include "tmultistack.h"  using namespace std;  int main()  {  unsigned max\_count = 0;  cout << "Enter number of checks - ";  cin >> max\_count;  clock\_t time;  clock\_t average\_time;  for ( unsigned size = 10; size < 100000; size \*= 10 )  {  TMStack <int> m1(2, size);  TMStack <int> m2(2, size);  for ( unsigned i = 0; i < size; i++ )  {  m1.Set(1, i);  }  average\_time = 0;  for (unsigned count = 0; count < max\_count; count++)  {  time = clock();  m1 = m2;  average\_time += clock() - time;  }  average\_time /= max\_count;  cout<< "Time is: " <<average\_time<<endl;  }  return 0;  } |

## 8.6. Приложение 6:Тесты для классов

|  |
| --- |
| **test\_main.cpp** |
| #include <gtest.h>  int main(int argc, char \*\*argv)  {  ::testing::InitGoogleTest(&argc, argv);  return RUN\_ALL\_TESTS();  } |

|  |
| --- |
| **test\_tstack.cpp** |
| #include "tstack.h"  #include <gtest.h>  TEST(TStack, can\_create\_stack)  {  ASSERT\_NO\_THROW(TStack<int> s1);  }  TEST(TStack, new\_stack\_is\_empty)  {  TStack<int> s1(3);  EXPECT\_EQ(true, s1.IsEmpty());  }  TEST(TStack, can\_create\_stack\_with\_positive\_size)  {  ASSERT\_NO\_THROW(TStack<int> s1(5));  }  TEST(TStack, throw\_when\_create\_stack\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TStack<int> s1(-5));  }  TEST(TStack, can\_create\_copied\_stack)  {  TStack<int> s1(5);  ASSERT\_NO\_THROW(TStack<int> s2(s1));  }  TEST(TStack, can\_put\_elem\_in\_stack)  {  TStack<int> s1(3);  ASSERT\_NO\_THROW(s1.Put(1));  }  TEST(TStack, stack\_with\_elem\_isnt\_empty)  {  TStack<int> s1(3);  s1.Put(1);  EXPECT\_FALSE(s1.IsEmpty());  }  TEST(TStack, cant\_put\_in\_full\_stack)  {  TStack<int> s1(3);  for ( int i = 0; i < 3; i++ )  {  s1.Put(i);  }  ASSERT\_ANY\_THROW(s1.Put(3));  }  TEST(TStack, can\_get\_elem\_from\_stack)  {  TStack<int> s1(1);  s1.Put(1);  int res = s1.Get();  EXPECT\_EQ(1, res);  }  TEST(TStack, cant\_get\_from\_empty\_stack)  {  TStack<int> s1(3);  ASSERT\_ANY\_THROW(s1.Get());  }  TEST(TStack, get\_returns\_last\_put\_elem)  {  TStack<int> s1(3);  int elem1 = 1, elem2 = 2, res;  s1.Put(elem1); s1.Put(elem2);  res = s1.Get();  EXPECT\_TRUE((res = elem2) && (res != elem1));  }  TEST(TStack, not\_empty\_assignment)  {  TStack<int> s1(10);  TStack<int> s2(10);  s1.Put(10);  s1.Put(20);  s1.Put(40);  s2 = s1;  EXPECT\_EQ(40, s2.Get());  EXPECT\_EQ(s2.Get(), 20);  EXPECT\_EQ(s2.Get(), 10);  }  TEST(TStack, can\_use\_isempty\_correctly)  {  TStack<int> s1(5);  ASSERT\_TRUE(s1.IsEmpty());  }  TEST(TStack, can\_use\_isempty\_incorrectly)  {  TStack<int> s1(2);  s1.Put(1);  ASSERT\_FALSE(s1.IsEmpty());  }  TEST(TStack, can\_use\_isfull\_correctly)  {  TStack<int> s1(1);  s1.Put(10);  ASSERT\_TRUE(s1.IsFull());  }  TEST(TStack, can\_use\_isfull\_incorrectly)  {  TStack<int> s1(5);  ASSERT\_FALSE(s1.IsFull());  }  TEST(TStack, compare\_equal\_stacks\_return\_true)  {  TStack<int> s1(2), s2(2);  s1.Put(10);  s1.Put(20);  s2 = s1;  EXPECT\_TRUE(s2==s1);  }  TEST(TStack, compare\_stack\_with\_itself\_return\_true)  {  TStack<int> s1(5);  EXPECT\_TRUE(s1==s1);  }  TEST(TStack, stacks\_with\_different\_size\_are\_not\_equal)  {  TStack<int> s1(5), s2(4);  EXPECT\_FALSE(s1==s2);  }  TEST(TStack, compare\_non\_equal\_stacks\_return\_false)  {  TStack<int> s1(2), s2(2);  s1.Put(10);  s1.Put(10);  s2.Put(10);  s2.Put(20);  EXPECT\_FALSE(s1==s2);  } |

|  |
| --- |
| **test\_tmultistack.cpp** |
| #include "tmultistack.h"  #include <gtest.h>  TEST(TMStack, cant\_create\_multistack\_with\_negative\_length)  {  ASSERT\_ANY\_THROW(TMStack<int> m1(3, -1));  }  TEST(TMStack, can\_create\_multistack\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TMStack<int> m1(3, 12));  }  TEST(TMStack, new\_multistack\_is\_empty)  {  TMStack<int> m1(3, 12);  EXPECT\_EQ(true, m1.IsEmpty(1));  }  TEST(TMStack, multistack\_with\_element\_is\_not\_empty)  {  TMStack<int> m1(3, 12);  m1.Set(1, 5);  EXPECT\_EQ(false, m1.IsEmpty(1));  }  TEST(TMStack, can\_copy\_multistack)  {  TMStack<int> m1(3, 12);  ASSERT\_NO\_THROW(TMStack<int> m2(m1));  }  TEST(TMStack, can\_set\_element\_in\_empty\_stack)  {  TMStack<int> m1(3, 12);  ASSERT\_NO\_THROW(m1.Set(1, 2));  }  TEST(TMStack, can\_set\_element\_in\_not\_full\_stack)  {  TMStack<int> m1(2, 4);  m1.Set(0, 12);  m1.Set(0, 12);  m1.Set(0, 12);  EXPECT\_NO\_THROW(m1.Set(0, 12));  }  TEST(TMStack, cant\_set\_element\_in\_full\_multistack)  {  TMStack<int> m1(2, 5);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  EXPECT\_ANY\_THROW(m1.Set(1, 12));  }  TEST(TMStack, can\_get\_element)  {  TMStack<int> m1(5, 15);  m1.Set(2, 12);  EXPECT\_EQ(12, m1.Get(2));  }  TEST(TMStack, cant\_get\_element\_from\_empty\_multistack)  {  TMStack<int> m1(2, 5);  EXPECT\_ANY\_THROW(m1.Get(1));  }  TEST(TMStack, correct\_order\_return\_value)  {  TMStack <int> m1(2, 5);  m1.Set(1, 1);  m1.Set(1, 2);  m1.Set(1, 3);  ASSERT\_EQ(3, m1.Get(1));  ASSERT\_EQ(2, m1.Get(1));  ASSERT\_EQ(1, m1.Get(1));  }  TEST(TMStack, can\_use\_isfull\_correctly)  {  TMStack<int> m1(5, 25);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  m1.Set(1, 12);  EXPECT\_EQ(true, m1.IsFull(1));  } |