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**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**«Структура хранения данных: Стек»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[1.Введение 3](#_Toc533245674)

[2. Цели и задачи 4](#_Toc533245675)

[2.1. Используемые инструменты 4](#_Toc533245676)

[3. Руководство пользователя 6](#_Toc533245677)

[4. Руководство программиста 8](#_Toc533245678)

[4.1. Описание структуры программы 8](#_Toc533245679)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc533245680)

[5. Эксперименты 10](#_Toc533245681)

[6. Заключение 11](#_Toc533245682)

[7. Литература 12](#_Toc533245683)

[8. Приложения 13](#_Toc533245684)

[8.1. Приложение 1:Класс TExсeption 13](#_Toc533245685)

[8.2. Приложение 2:Класс TStack 13](#_Toc533245686)

[8.3. Приложение 3:Код программы тестирования и экспериментов 16](#_Toc533245687)

[8.4. Приложение 4:Тесты для класса 16](#_Toc533245688)

# 1.Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек.

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.

# 2. Цели и задачи

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

* добавления элемента в стек,
* извлечения элемента из стека (с удалением),
* проверка стека на полноту/пустоту.

Хранить данную структуру в этой лабораторной работе будем с помощью массива. Такая организация стека удобна, если элемент информации занимает в памяти фиксированное количество длины, например, какую-то одну условную единицу. При этом отпадает необходимость хранения в элементе стека явного указателя на следующий элемент стека, что экономит память.

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 методов шаблонного класса TStack согласно заданному интерфейсу.
2. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы TStack.
5. Модификация примера использования в тестовое приложение, позволяющее задавать стеки 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\_tstack.cpp (Рис.1):

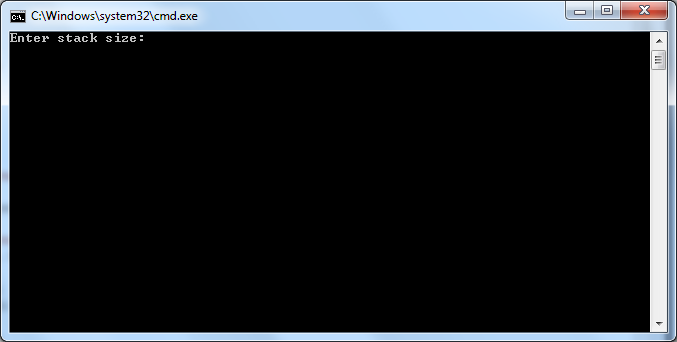


Рис.1.Ввод размера стека.

Пользователю предлагается ввести максимальный размер создаваемого стека. Если ввести отрицательное число, бросится исключение (Рис.2):

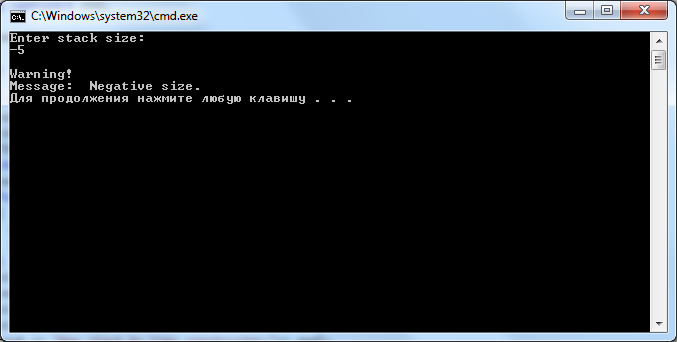


Рис.2. Ошибка.

При положительном значении создастся стек заданного размера для хранения значений любого типа. Затем пользователю предлагается заполнить все ячейки стека (Рис.3):

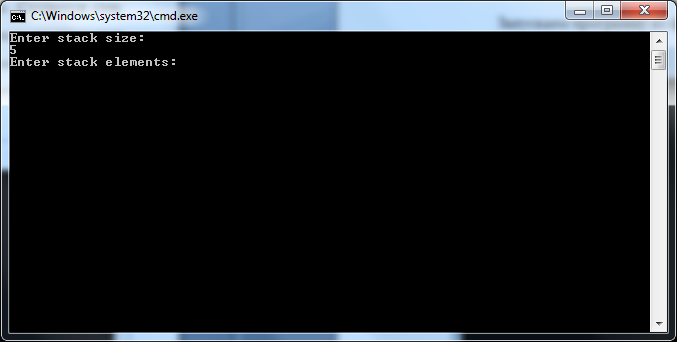


Рис.3. Заполнение стека.

После этого происходит печать стека на консоль. Удаляется один элемент. Снова выполняется печать того, что осталось в стеке и создается точно такой же стек с помощью конструктора копирования (Рис.4):

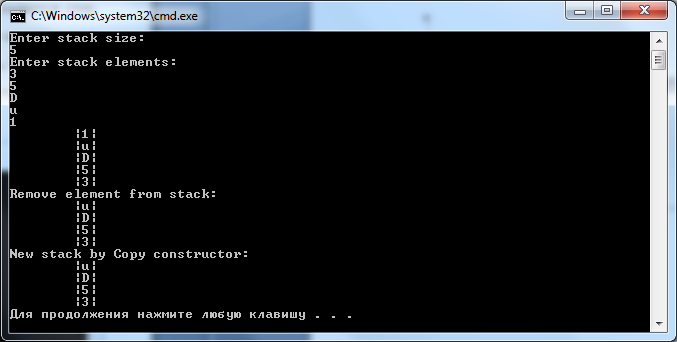


Рис.4.Пример операций со стеком

. Алгоритм можно повторить несколько раз, используя другие значения.

# 4. Руководство программиста

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

В данной работе будет использовано 2 класса:

* Класс «Стек» (TStack), реализованный с использованием массива
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

1. tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
2. exсeption.h – модуль с классом исключения TExсeption.

* sample\_tstack.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.

1. test\_main.cpp, test\_tstack.cpp – модуль с функциями тестирования для созданных классов. Содержат 20 тестов для класса TStack.

## 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. - 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 векторов на равенство. Возвращает true, если равенство выполняется, false в противном случае.

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

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

# 5. Эксперименты

В качестве примера рассмотрим перегрузку оператора присвоения для класса стека (TStack).

Теоретическая сложность выполнения алгоритма O(1).

Мы провели измерение присваивая стеку разное количество элементов: 10, 100, …, 1000000 элементов. Ниже вы можете увидеть график зависимости времени выполнения операции присвоения от количества элементов стека. (Рис.5) По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

Рис.5. График зависимости времени выполнения операции присвоения от размера стека.

По горизонтали – количество присваиваемых элементов.

По вертикали - время выполнения программы.

# 6. Заключение

В результате лабораторной работы была разработана структура данных - стек, 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://www.intuit.ru/studies/courses/648/504/lecture/11457

# 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);  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:Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_tstack.cpp** |
| #include <iostream>  #include <locale.h>  #include "tstack.h"  using namespace std;  int main()  {  try  {  int n;  cout << "Enter stack size:"<< endl;  cin >> n;  TStack<char> s1(n);  char symbol;  cout << "Enter stack elements:"<< endl;  for ( int i = 0; i < n; i++ )  {  cin >> symbol;  s1.Put(symbol);  }  s1.PrintStack();  cout << "Remove element from stack:"<< endl;  s1.Get();  s1.PrintStack();  cout << "New stack by Copy constructor:"<< endl;  TStack<char> s2(s1);  s2.PrintStack();  }  catch (TException exp)  {  exp.Show();  }  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "tstack.h"  #include "time.h"  using namespace std;  int main()  {  int max\_count = 0;  cout << "Enter number of checks - ";  cin >> max\_count;  clock\_t time;  clock\_t average\_time;  for ( unsigned size = 10; size < 1000000; size \*= 10 )  {  TStack<int> s1(size);  TStack<int> s2(size);  for ( int i = 0; i < size; i++ )  {  s1.Put(i);  }  average\_time = 0;  for (unsigned count = 0; count < max\_count; count++)  {  time = clock();  s1 = s2;  average\_time += clock() - time;  }  average\_time /= max\_count;  cout<< "Time is: " <<average\_time<<endl;  }  return 0;  } |

## 8.4. Приложение 4:Тесты для класса

|  |
| --- |
| **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\_EQ(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);  } |