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 №8**

**«Стек на списке»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[1.Введение 3](#_Toc718796)

[2. Цели и задачи 4](#_Toc718797)

[2.1. Используемые инструменты 4](#_Toc718798)

[3. Руководство пользователя 6](#_Toc718799)

[4. Руководство программиста 8](#_Toc718800)

[4.1. Описание структуры программы 8](#_Toc718801)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc718802)

[5. Эксперименты 11](#_Toc718803)

[6. Заключение 12](#_Toc718804)

[7. Литература 13](#_Toc718805)

[8. Приложения 14](#_Toc718806)

[8.1. Приложение 1:Класс TExсeption 14](#_Toc718807)

[8.2. Приложение 2:Класс TElement 14](#_Toc718808)

[8.3. Приложение 3:Класс TList 15](#_Toc718809)

[8.4. Приложение 4:Класс TStackList 19](#_Toc718810)

[8.5. Приложение 5:Код программы тестирования и экспериментов 21](#_Toc718811)

[8.6. Приложение 6:Тесты для класса 22](#_Toc718812)

# 1.Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек на списке.

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 на списке и выполнение основных операций над ним:

* добавления элемента в стек,
* извлечения элемента из стека (с удалением),
* проверка стека 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 методов шаблонного класса TStackList 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 TStackList.
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 sample\_tstacklist.cpp (Рис.1):

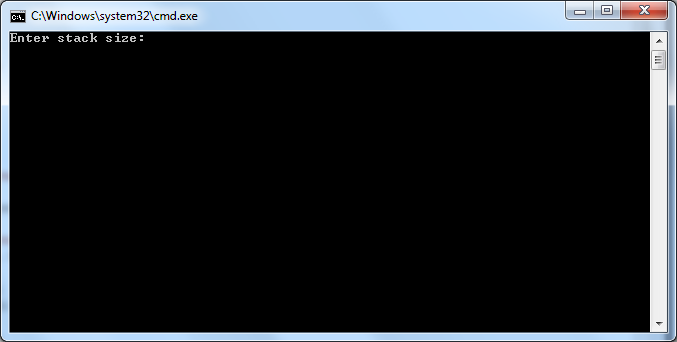


Рис.1.Ввод размера стека.

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

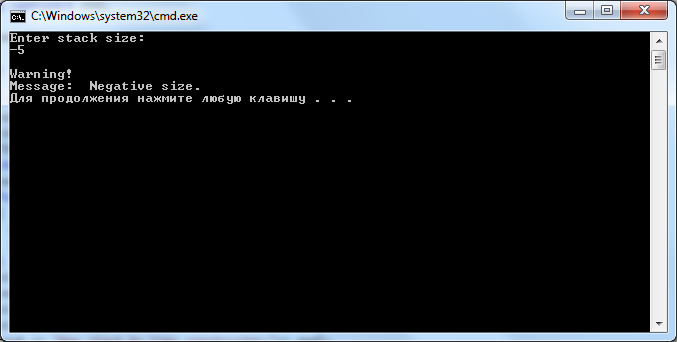


Рис.2. Ошибка.

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

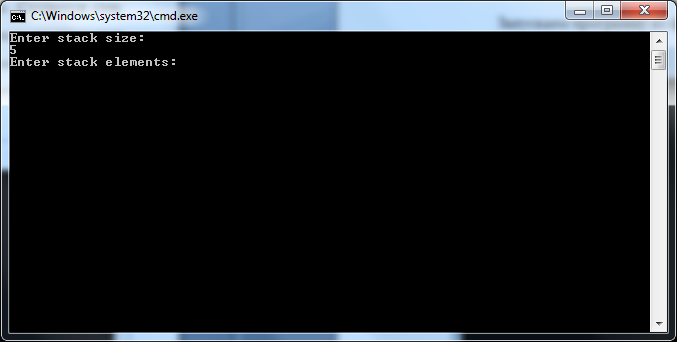


Рис.3. Заполнение стека.

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):

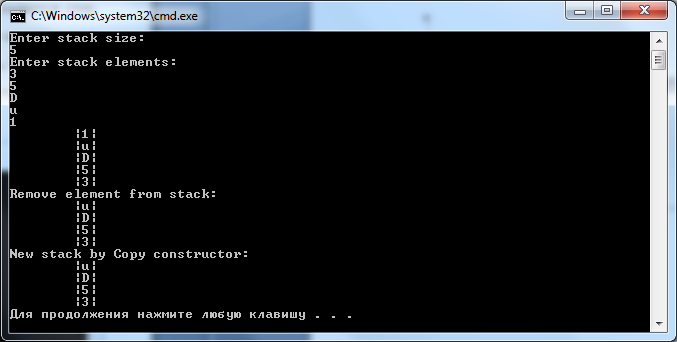


Рис.4.Пример операций со стеком.

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 4 класса:

* Класс «Элемент» (TElement) – реализует узел списка.
* Класс «Список» (TList), реализованный с использованием указателей.
* Класс «Стек» (TStackList), реализованный на основе класса TList.
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

1. telement.h– модуль с классом TElement, в котором определен интерфейс шаблонного класса Элемент и реализация его методов.
2. tlist.h– модуль с классом TList, в котором определен интерфейс шаблонного класса Список и реализация его методов.
3. tstacklist.h– модуль с классом TStackList, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
4. exсeption.h – модуль с классом исключения TExсeption.

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

1. test\_main.cpp, test\_tstacklist.cpp – модуль с функциями тестирования для созданных классов. Содержат 18 тестов для класса TStackList.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TElement:

template <class ElementType>

class TElement

1. TElement(ElementType \_elem = 0, TElement<ElementType>\* \_next = 0) - конструктор класса с параметрами, принимающий значение элемента и указатель на следующий элемент. По умолчанию значения установятся в ноль .

2. TElement(TElement<ElementType> &Elem) - конструктор копирования.

Принимает ссылку на объект класса TElement.

3. ElementType GetElem() - метод класса, позволяющий получить значение элемента.

4. TElement\* GetNext() – метод класса, позволяющий получить указатель на следующий элемент.

5. void Set(ElementType \_elem) - метод класса, позволяющий установить значение элемента.

6. void SetNext(TElement<ElementType>\* next) - метод класса, позволяющий установить указатель.

Рассмотрим реализацию методов шаблонного класса TList:

template <class ListType>

class TList

1. TList() – конструктор класса по умолчанию.

2. TList(TList<ListType> &List) - конструктор копирования.

Принимает ссылку на объект класса TList.

3. ~TList() - деструктор. Освобождает выделенную память.

4. void Put(int \_n, ListType elem) - метод, позволяющий добавить новый элемент в список на определенную позицию.

5. void PutStart(ListType List) - метод, позволяющий добавить новый элемент в начало списка.

6. void PutEnd(ListType List) - метод, позволяющий добавить новый элемент в конец списка.

7. ListType Get(int \_n) - метод изъятия элемента на определенной позиции из списка с удалением.

8. ListType GetStart() - метод изъятия элемента из начала списка с удалением.

9. ListType GetEnd() - метод изъятия элемента с конца списка с удалением.

10. bool IsFull() - метод проверки списка на полноту.

11. bool IsEmpty() - метод проверки списка на пустоту.

12. void PrintList() - метод отображения текущих элементов списка.

Рассмотрим реализацию методов шаблонного класса TStackList:

template <class StackListType>

class TStackList : public TList<StackListType>

1. TStackList<StackListType>(int \_size = 10) - конструктор класса, принимающий размер стека. По умолчанию создается стек размера 10 с позицией вершины стека 0.

2. TStackList<StackListType>(TStackList<StackListType> &A) - конструктор копирования.

Принимает ссылку на объект класса TStackList.

3. ~TStackList() - деструктор. Освобождает выделенную память.

4. void Put(StackListType A) - метод, позволяющий добавить новый элемент в стек.

4. StackListType Get() - метод изъятия элемента из вершины стека с удалением.

5. int GetMaxSize() - возвращает максимальный размер стека.

6. int GetSize() - возвращает размер стека.

7. bool IsEmpty() - метод проверки стека на пустоту.

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

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

# 5. Эксперименты

В качестве примера рассмотрим операции добавления/изъятия элемента для класса стека (TStackList).

Теоретическая сложность выполнения алгоритмов O(1).

Мы провели измерение скорости добавления/изъятия элемента, при разном количестве элементов в стеке: 10, 100, …, 1000000 элементов. Ниже вы можете увидеть таблицу зависимости времени выполнения операции от количества элементов списка. (Таблица.1).

По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритмов равна теоретической.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | метод Put () | метод Get() |
| 10 | 0 | 0 |
| 100 | 0 | 0 |
| 1000 | 0 | 0 |
| 10000 | 0 | 0 |
| 100000 | 0 | 0 |

Таблица.1. Зависимость времени выполнения операций от размера стека.

# 6. Заключение

В результате лабораторной работы была разработана структура данных – стек на списке, 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 TElement

|  |
| --- |
| **telement.h** |
| #pragma once  #include <iostream>  using namespace std;  template <class ElementType>  class TElement  {  private:  ElementType elem;  TElement<ElementType>\* next;  public:  TElement(ElementType \_elem = 0, TElement<ElementType>\* \_next = 0);  TElement(TElement<ElementType> &Elem);  ElementType GetElem();  TElement\* GetNext();  void Set(ElementType \_elem);  void SetNext(TElement<ElementType>\* next);  };  template <class ElementType>  TElement<ElementType>::TElement(ElementType \_elem, TElement<ElementType>\* \_next)  {  elem = \_elem;  if ( \_next == 0 )  next = 0;  else  next = \_next;  }  template <class ElementType>  TElement<ElementType>::TElement(TElement<ElementType> &Elem)  {  elem = Elem.elem;  next = Elem.next;  }  template <class ElementType>  ElementType TElement<ElementType>::GetElem()  {  return elem;  }  template <class ElementType>  TElement<ElementType>\* TElement<ElementType>::GetNext()  {  return next;  }  template <class ElementType>  void TElement<ElementType>::Set(ElementType \_elem)  {  elem = \_elem;  }  template <class ElementType>  void TElement<ElementType>::SetNext(TElement<ElementType>\* \_next)  {  next = \_next;  } |

## 8.3. Приложение 3:Класс TList

|  |
| --- |
| **tlist.h** |
| #pragma once  #include <iostream>  #include "exception.h"  #include "telement.h"  using namespace std;  template <class ListType>  class TList  {  protected:  int count;  TElement<ListType>\* start;  public:  TList();  TList(TList<ListType> &List);  ~TList();  void Put(int \_n, ListType elem);  void PutStart(ListType List);  void PutEnd(ListType List);  ListType Get(int \_n);  ListType GetStart();  ListType GetEnd();  bool IsFull();  bool IsEmpty();  void PrintList();  };  template <class ListType>  TList<ListType>::TList()  {  count = 0;  start = 0;  }  template <class ListType>  TList<ListType>::TList(TList<ListType> &List)  {  count = List.count;  TElement<ListType> \*tmp1 = List.start;  TElement<ListType> \*tmp2;  if ( List.start == 0 )  start = 0;  else  {  start = new TElement<ListType>(\*List.start);  tmp2 = start;  while (tmp1->TElement<ListType>::GetNext() != 0)  {  tmp2->TElement<ListType>::SetNext(new TElement<ListType>(\*(tmp1->TElement<ListType>::GetNext())));  tmp1 = tmp1->TElement<ListType>::GetNext();  tmp2 = tmp2->TElement<ListType>::GetNext();  }  }  }  template <class ListType>  TList<ListType>::~TList()  {  while ( start != 0 )  {  TElement<ListType>\* temp = start;  start = start->GetNext();  delete temp;  }  }  template<class ListType>  void TList<ListType>::Put(int \_n, ListType elem)  {  if (\_n < 1 || \_n > count - 1)  throw TException("Uncurrent index.");  else  {  int i = 0;  TElement<ListType>\* li = start;  while (i != \_n - 1)  {  li = li->GetNext();  i++;  }  TElement<ListType>\* temp = new TElement<ListType>(elem, li->GetNext());  li->SetNext(temp);  }  count++;  }  template <class ListType>  void TList<ListType>::PutStart(ListType List)  {  if ( start == 0 )  {  TElement<ListType>\* temp = new TElement<ListType>(List, 0);  start = temp;  }  else  {  TElement<ListType>\* temp = new TElement<ListType>(List, start);  start = temp;  }  count++;  }  template <class ListType>  void TList<ListType>::PutEnd(ListType List)  {  if (IsEmpty())  {  start = new TElement<ListType>(List, 0);  }  else  {  TElement<ListType>\* temp = start;  while ( temp->GetNext() != 0 )  temp = temp->GetNext();  temp->SetNext(new TElement <ListType>(List, 0));  }  count++;  }  template<class ListType>  ListType TList<ListType>::Get(int \_n)  {  if (IsEmpty() )  throw TException( "List is empty." );  if ( \_n < 1 || \_n > count - 1 )  throw TException( "Uncurrent index." );  else  {  int i = 0;  count--;  TElement<ListType>\* li = start;  TElement<ListType>\* di = start->GetNext();  while ( i != \_n - 1 )  {  li = di;  di = di->GetNext();  i++;  }  ListType temp = di->GetElem();  li->SetNext( di->GetNext() );  delete di;  return temp;  }  }  template <class ListType>  ListType TList<ListType>::GetStart()  {  if (!IsEmpty())  {  ListType temp = start->GetElem();  start = start->GetNext();  count--;  return temp;  }  else  throw TException("Have not any elem");  }  template <class ListType>  ListType TList<ListType>::GetEnd()  {  if (!IsEmpty())  {  TElement<ListType>\* temp = start;  while ((temp->GetNext())->GetNext() != 0)  temp = temp->GetNext();  ListType tempE = (temp->GetNext())->GetElem();  temp->SetNext(0);  count--;  return tempE;  }  else  throw TException("Have not any elem");  }  template <class ListType>  bool TList<ListType>::IsFull()  {  try  {  TElement<ListType>\* List = new TElement<ListType>();  if ( List != 0 )  {  delete List;  return true;  }  else  return false;  }  catch (...)  {  return false;  }  return true;  }  template <class ListType>  bool TList<ListType>::IsEmpty()  {  return ( start == 0 );  }  template<class ListType>  void TList<ListType>::PrintList()  {  if ( start == 0 )  throw TException( "List is empty." );  else  {  TElement<ListType>\* li = start;  while ( li->GetNext() != 0 )  {  cout << "\t|" << li->GetElem() << "|" << endl;  li = li->GetNext();  }  cout << "\t|" << li->GetElem() << "|" << endl;  }  } |

## 8.4. Приложение 4:Класс TStackList

|  |
| --- |
| **tstacklist.h** |
| #pragma once  #include <iostream>  #include "exception.h"  #include "tlist.h"  using namespace std;  template <class StackListType>  class TStackList : public TList<StackListType>  {  protected:  int size;  public:  TStackList<StackListType>(int \_size = 10);  TStackList<StackListType>(TStackList<StackListType> &A);  ~TStackList();  void Put(StackListType A);  StackListType Get();  int GetMaxSize();  int GetSize();  bool IsEmpty();  bool IsFull();  void PrintStack();  };  template <class StackListType>  TStackList<StackListType>::TStackList(int \_size) : TList<StackListType>()  {  if ( \_size <= 0 )  throw TException( "Negative size" );  size = \_size;  }  template <class StackListType>  TStackList<StackListType>::TStackList(TStackList<StackListType> &A) : TList<StackListType>(A)  {  TList<StackListType>::count = A.count;  }  template<class StackListType>  TStackList<StackListType>::~TStackList()  {}  template <class StackListType>  void TStackList<StackListType>::Put(StackListType A)  {  if ( this->IsFull() )  throw TException( "StackList is Full" );  TList<StackListType>::PutStart(A);  }  template <class StackListType>  StackListType TStackList<StackListType>::Get()  {  if ( this->IsEmpty() )  throw TException( "StackList is Empty" );  return TList<StackListType>::GetStart();  }  template <class StackListType>  int TStackList<StackListType>::GetMaxSize()  {  return size;  }  template<class StackListType>  int TStackList<StackListType>::GetSize()  {  return TList<StackListType>::count;  }  template <class StackListType>  bool TStackList<StackListType>::IsEmpty()  {  if ( TList<StackListType>::count == 0 )  return true;  return false;  }  template<class StackListType>  bool TStackList<StackListType>::IsFull()  {  if ( TList<StackListType>::count == size )  return true;  return false;  }  template<class StackListType>  void TStackList<StackListType>::PrintStack()  {  if ( this->IsEmpty() )  throw TException( "StackList is Empty" );  TList<StackListType>::PrintList();  } |

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

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

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "tstacklist.h"  #include "time.h"  using namespace std;  int main()  {  clock\_t time\_start;  clock\_t total\_time;  for ( unsigned size = 10; size < 1000000; size \*= 10 )  {  TStackList<int> p1(size);  for ( unsigned i = 0; i < size-1; i++ )  {  p1.Put(i);  }  total\_time = 0;  time\_start = clock();  /\*p1.Put(1);\*/  p1.Get();  total\_time += clock() - time\_start;  /\*cout<< "Size is: " << size << " Time(put) is: " << total\_time <<endl;\*/  cout<< "Size is: " << size << " Time(get) is: " << total\_time <<endl;  }  return 0;  } |

## 8.6. Приложение 6:Тесты для класса

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

|  |
| --- |
| **test\_tstacklist.cpp** |
| #include "tstacklist.h"  #include <gtest.h>  TEST(TStackList, can\_create\_stacklist)  {  ASSERT\_NO\_THROW(TStackList<int> s1);  }  TEST(TStackList, new\_stacklist\_is\_empty)  {  TStackList<int> s1(3);  EXPECT\_EQ(true, s1.IsEmpty());  }  TEST(TStackList, can\_create\_stacklist\_with\_positive\_size)  {  ASSERT\_NO\_THROW(TStackList<int> s1(5));  }  TEST(TStackList, throw\_when\_create\_stacklist\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TStackList<int> s1(-5));  }  TEST(TStackList, can\_create\_copied\_stacklist)  {  TStackList<int> s1(5);  ASSERT\_NO\_THROW(TStackList<int> s2(s1));  }  TEST(TStackList, can\_get\_max\_size)  {  TStackList<int> s1(10);  ASSERT\_EQ(s1.GetMaxSize(), 10);  }  TEST(TStackList, can\_put\_elem\_in\_stacklist)  {  TStackList<int> s1(3);  ASSERT\_NO\_THROW(s1.Put(1));  }  TEST(TStackList, stacklist\_with\_elem\_isnt\_empty)  {  TStackList<int> s1(3);  s1.Put(1);  EXPECT\_EQ(false, s1.IsEmpty());  }  TEST(TStackList, cant\_put\_in\_full\_stacklist)  {  TStackList<int> s1(1);  s1.Put(1);  ASSERT\_ANY\_THROW(s1.Put(3));  }  TEST(TStackList, can\_get\_elem\_from\_stacklist)  {  TStackList<int> s1;  s1.Put(1);  int res = s1.Get();  EXPECT\_EQ(1, res);  }  TEST(TStackList, cant\_get\_from\_empty\_stacklist)  {  TStackList<int> s1(3);  ASSERT\_ANY\_THROW(s1.Get());  }  TEST(TStackList, get\_returns\_last\_put\_elem)  {  TStackList<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(TStackList, not\_empty\_assignment)  {  TStackList<int> s1(10);  TStackList<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(TStackList, can\_use\_isempty\_correctly)  {  TStackList<int> s1(3);  ASSERT\_TRUE(s1.IsEmpty());  }  TEST(TStackList, can\_use\_isempty\_incorrectly)  {  TStackList<int> s1(2);  s1.Put(1);  ASSERT\_FALSE(s1.IsEmpty());  }  TEST(TStackList, can\_use\_isfull\_correctly)  {  TStackList<int> s1(1);  s1.Put(10);  ASSERT\_TRUE(s1.IsFull());  }  TEST(TStackList, can\_use\_isfull\_incorrectly)  {  TStackList<int> s1(5);  ASSERT\_FALSE(s1.IsFull());  }  TEST(TStackList, can\_get\_size)  {  TStackList<int> S(5);  S.Put(1);  S.Put(2);  ASSERT\_EQ(S.GetSize(), 2);  } |