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 №6**

**«Список на указателях»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[1.Введение 3](#_Toc14752)

[2. Цели и задачи 5](#_Toc14753)

[2.1. Используемые инструменты 5](#_Toc14754)

[3. Руководство пользователя 7](#_Toc14755)

[4. Руководство программиста 8](#_Toc14756)

[4.1. Описание структуры программы 8](#_Toc14757)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc14758)

[5. Эксперименты 10](#_Toc14759)

[6. Заключение 11](#_Toc14760)

[7. Литература 12](#_Toc14761)

[8. Приложения 13](#_Toc14762)

[8.1. Приложение 1:Класс TExсeption 13](#_Toc14763)

[8.2. Приложение 2:Класс TElement 13](#_Toc14764)

[8.3. Приложение 3:Класс TList 14](#_Toc14765)

[8.4. Приложение 4:Код программы тестирования и экспериментов 18](#_Toc14766)

[8.5. Приложение 5:Тесты для класса 19](#_Toc14767)

# 1.Введение

Списки являются чрезвычайно гибкой структурой, так как их легко сделать большими или меньшими, и их элементы доступны для вставки или удаления в любой позиции списка. Списки также можно объединять или разбивать на меньшие списки. Списки регулярно используются в приложениях, например в программах информационного поиска, трансляторах программных языков или при моделировании различных процессов.

В математике список представляет собой последовательность элементов определенного типа ( elementtype ). Представим список в виде последовательности элементов, разделенных запятыми: a1, а2, ..., аn, где n≥ 0 и всё ai имеют тип elementtype. Количество элементов n - длина списка. Если n ≥ 1, то а1 называется первым элементом, а аn—последним элементом списка. В случае n = 0 имеем пустой список, который не содержит элементов.

Важное свойство списка заключается в том, что его элементы можно линейно упорядочить в соответствии с их позицией в списке. Говорим, что элемент ai предшествует ai+1 для і = 1, 2, ..., n — 1 и ai следует за ai-1 для і =2, 3, .., n. Также будем говорить, что элемент аi имеет позицию і.

Реализация списков с помощью указателей

Для реализации однонаправленных списков используются указатели, связывающие последовательные элементы списка. Эта реализация освобождает нас от использования непрерывной области памяти для хранения списка и, следовательно, от необходимости перемещения элементов списка при вставке или удалении элементов. Однако ценой за это удобство становится дополнительная память для хранения указателей.

В этой реализации список состоит из ячеек, каждая из которых содержит элемент списка и указатель на следующую ячейку списка. Если список состоит из элементов а1 , а2, ..., аn, то для і = 1, 2, ..., n-1 ячейка, содержащая элемент аi, имеет также указатель на ячейку, содержащую элемент ai+1. Ячейка, содержащая элемент аn, имеет указатель nil (нуль). Имеется также ячейка header (заголовок), которая указывает на ячейку, содержащую а1. Ячейка header не содержит элементов списка. В случае пустого списка заголовок Имеет указатель nil, не указывающий ни на какую ячейку. На рис. 2.2 показан связанный список описанного вида.

|  |
| --- |
|  |
|  | http://bookwu.net/imgs/1432734111image012.jpg |

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – линейный односвязный список на указателях и выполнение основных операций над ним:

* добавления в начало элемента списка.
* добавления в конец элемента списка.
* добавления, извлечения промежуточного элемента списка.
* извлечения с удалением элемента из начала списка,
* извлечения с удалением элемента из конца списка,
* проверка списка 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 TList 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 TList.
5. Модификация примера использования в тестовое приложение, позволяющее задавать списки 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\_tlist.cpp.

Программа выведет пример правильной работы основных операций со списком. (Рис.1):

1. добавления в начало элемента списка.
2. добавления в конец элемента списка.
3. добавления, извлечения промежуточного элемента списка.
4. извлечения с удалением элемента из начала списка.
5. извлечения с удалением элемента из конца списка.

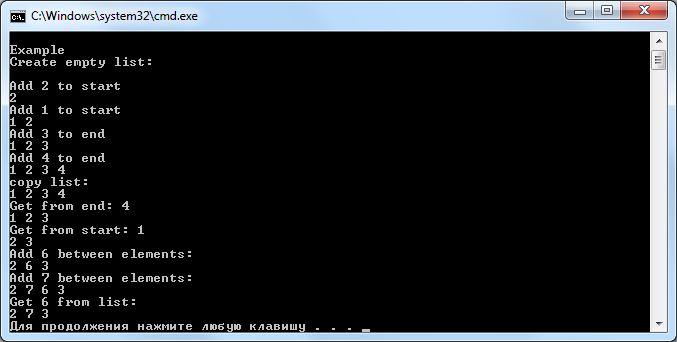


Рис.1.Пример.

# 4. Руководство программиста

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

В данной работе будет использовано 3 класса:

* Класс «Элемент» (TElement) – реализует узел списка.
* Класс «Список» (TList), реализованный с использованием указателей.
* 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. exсeption.h – модуль с классом исключения TExсeption.
4. sample\_tlist.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
5. test\_main.cpp, test\_tlist.cpp, test\_telement.cpp – модуль с функциями тестирования для созданных классов. Содержит 22 теста для класса TList и 6 тестов для класса TElement.

## 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.

# 5. Эксперименты

В качестве примера рассмотрим операцию добавления элемента в конец и в начало списка для класса TList.

Теоретическая сложность выполнения алгоритмов O(1).

Мы провели измерение скорости добавления элемента в конец и в начало списка при разном количестве элементов: 10, 100, …, 1000000. 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 алгоритма равна теоретической.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | метод PutEnd() | метод PutStart() |
| 10 | 0 | 0 |
| 100 | 0 | 0 |
| 1000 | 0 | 0 |
| 10000 | 0 | 0 |
| 100000 | 0 | 0 |

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

# 6. Заключение

В результате лабораторной работы была разработана структура данных – односвязный линейный список на указателях, 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. <http://bookwu.net/book_algoritmy-i-struktury-dannyh_1245/12_2.2.-realizaciya-spiskov>
3. Д. Кнут. Искусство программирования. (3-е издание) Т.1.

# 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()  {  start = 0;  }  template <class ListType>  TList<ListType>::TList(TList<ListType> &List)  {  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 << li->GetElem() << " ";  li = li->GetNext();  }  cout << li->GetElem() << " ";  }  } |

## 8.4. Приложение 4:Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_tlist.cpp** |
| #include "tlist.h"  using namespace std;  int main()  {  cout << endl << "Example" << endl;  cout << "Create empty list:" << endl;  TList<int> list1;  cout << endl << "Add 2 to start" << endl;  list1.PutStart(2);  list1.PrintList();  cout << endl <<"Add 1 to start" << endl;  list1.PutStart(1);  list1.PrintList();  cout << endl <<"Add 3 to end" << endl;  list1.PutEnd(3);  list1.PrintList();  cout << endl <<"Add 4 to end" << endl;  list1.PutEnd(4);  list1.PrintList();  cout << endl <<"copy list:" << endl;  TList<int> list2(list1);  list1.PrintList();  cout << endl <<"Get from end: " << list1.GetEnd() << endl;  list1.PrintList();  cout << endl <<"Get from start: " << list1.GetStart() << endl;  list1.PrintList();  cout << endl <<"Add 6 between elements:" << endl;  list1.Put(1, 6);  list1.PrintList();  cout << endl <<"Add 7 between elements:" << endl;  list1.Put(1, 7);  list1.PrintList();  cout << endl <<"Get 6 from list:" << endl;  list1.Get(2);  list1.PrintList();  cout << endl;  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "tlist.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 )  {  TList<int> list1;  for ( unsigned i = 0; i < size; i++ )  {  list1.PutStart(i);  }  total\_time = 0;  time\_start = clock();  /\*list1.PutEnd(1);\*/  list1.PutStart(1);  total\_time += clock() - time\_start;  cout<< "Size is: " << size << " Time(put to the start) is: " << total\_time <<endl;  /\*cout<< "Size is: " << size << " Time(put to the end) is: " << total\_time <<endl;\*/  }  return 0;  } |

## 8.5. Приложение 5:Тесты для класса

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

|  |
| --- |
| **test\_tlist.cpp** |
| #include <gtest.h>  #include "tlist.h"  TEST(TList, can\_create\_list)  {  ASSERT\_NO\_THROW(TList<int> list);  }  TEST(TList, can\_create\_right\_copy\_of\_list)  {  TList <int> list;  list.PutStart(1);  TList<int> list2(list);  ASSERT\_EQ(list.GetStart(), list2.GetStart());  }  TEST(TList, can\_put\_elem\_into\_list)  {  TList <int> list;  list.PutStart(4);  list.PutStart(3);  list.PutStart(2);  list.Put(2, 1);  ASSERT\_EQ(list.Get(2), 1);  }  TEST(TList, can\_put\_elem\_to\_the\_start\_of\_list)  {  TList <int> list;  ASSERT\_NO\_THROW(list.PutStart(1));  }  TEST(TList, can\_put\_elem\_to\_the\_end\_of\_list)  {  TList <int> list;  ASSERT\_NO\_THROW(list.PutEnd(1));  }  TEST(TList, throw\_when\_put\_elem\_to\_incorrent\_index)  {  TList<int> list;  list.PutStart(1);  list.PutStart(1);  list.PutStart(1);  ASSERT\_ANY\_THROW(list.Put(0, 1));  }  TEST(TList, throw\_when\_get\_elem\_when\_list\_is\_empty)  {  TList<int> list;  ASSERT\_ANY\_THROW(list.Get(1));  }  TEST(TList, can\_get\_elem\_to\_index)  {  TList<int> list;  list.PutStart(3);  list.PutStart(2);  list.PutStart(1);  int res = list.Get(1);  ASSERT\_EQ(2, res);  }  TEST(TList, throw\_when\_get\_elem\_with\_incorrent\_index)  {  TList<int> list;  list.PutStart(1);  list.PutStart(1);  list.PutStart(1);  ASSERT\_ANY\_THROW(list.Get(0));  }  TEST(TList, can\_get\_elem\_from\_the\_start\_of\_list)  {  TList<int> list;  list.PutStart(2);  list.PutStart(1);  int res = list.GetStart();  ASSERT\_EQ(1, res);  }  TEST(TList, can\_get\_elem\_from\_the\_end\_of\_list)  {  TList<int> list;  list.PutStart(2);  list.PutStart(1);  int res = list.GetEnd();  ASSERT\_EQ(2, res);  }  TEST(TList, throw\_when\_get\_from\_start\_when\_list\_is\_empty)  {  TList<int> list;  ASSERT\_ANY\_THROW(list.GetStart());  }  TEST(TList, throw\_when\_get\_from\_end\_when\_list\_is\_empty)  {  TList<int> list;  ASSERT\_ANY\_THROW(list.GetEnd());  }  TEST(TList, can\_use\_isempty\_correctly)  {  TList<int> list;  ASSERT\_TRUE(list.IsEmpty());  }  TEST(TList, can\_use\_isempty\_incorrectly)  {  TList<int> list;  list.PutStart(1);  ASSERT\_FALSE(list.IsEmpty());  }  TEST(TList, can\_use\_isfull\_correctly)  {  TList<int> list;  list.PutStart(1);  ASSERT\_TRUE(list.IsFull());  } |

|  |
| --- |
| **test\_telement.cpp** |
| #include <gtest.h>  #include "telement.h"  TEST(TElement, can\_create\_element)  {  ASSERT\_NO\_THROW(TElement<int> element);  }  TEST(TElement, can\_copy\_element)  {  TElement<int> element(2,0);  TElement<int> element2(element);  ASSERT\_EQ(element.GetElem(), element2.GetElem());  }  TEST(TElement, can\_get\_element)  {  TElement<int> element(5, 0);  int res = element.GetElem();  ASSERT\_EQ(5, res);  }  TEST(TElement, can\_get\_next)  {  TElement<int>\* elem = 0;  TElement<int> element(5, elem);  ASSERT\_EQ(element.GetNext(), elem);  }  TEST(TElement, can\_create\_elem\_default)  {  TElement<int> elem;  ASSERT\_EQ(elem.GetElem(), 0);  }  TEST(TElement, can\_set\_next)  {  TElement<int>\* li = 0;  TElement<int> elem;  elem.SetNext(li);  ASSERT\_EQ(elem.GetNext(), li);  } |