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

Стеценко Данил Михайлович

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

**Содержание**

[1. Введение 3](#_30j0zll)

[2. Цели и задачи 4](#_2s8eyo1)

[2.1. Используемые инструменты 4](#_17dp8vu)

[3. Руководство пользователя 5](#_3rdcrjn)

[4. Руководство программиста 6](#_26in1rg)

[4.1. Описание структуры программы 6](#_lnxbz9)

[4.2. Описание функций и процедур, их алгоритмов 6](#_35nkun2)

[5. Эксперименты 9](#_44sinio)

[6. Заключение 10](#_2jxsxqh)

[7. Литература 11](#_z337ya)

[8. Приложения 12](#_3j2qqm3)

# 1. Введение

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 Stacklib, построенного на основе существующего класса Stack.
2. Реализация класса MultiStack, построенного на основе класса MultiStack.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы созданных классов.
5. 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):

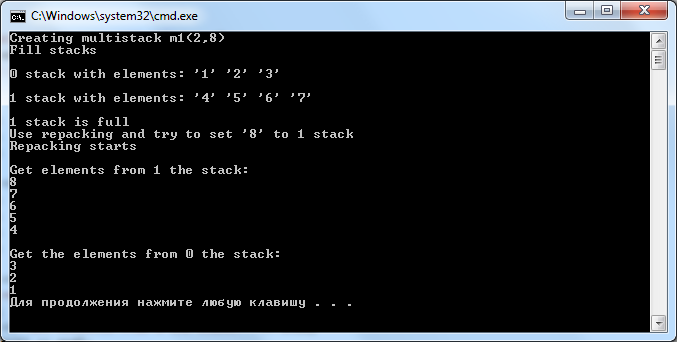


Рис.1.Пример операций с мультистеком.

# 4. Руководство программиста

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

В данной работе будет использовано 4 класса:

* Класс «Стек» (Stacklib), реализованный с использованием массива.
* Класс «Мультистек» (MultiStack), построенного на основе класса MultiStack.

## 4.1. Описание структуры программы

Модульная структура программы:

* Stack.h– модуль с классом Stacklib, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
* MultiStack.h– модуль с классом MultiStack, в котором определен интерфейс шаблонного класса Мультистек и реализация его методов.
* 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 Steck

|  |
| --- |
| Steck.h |
| #ifndef \_STACK\_  #define \_STACK\_  #include <iostream>  #include <fstream>  using namespace std;  template <class T>  class TStack  {  protected:  int length;  T\* data;  int top;  bool f;  public:  TStack(int size = 1, bool \_f = true);  TStack(TStack<T>& \_v);  ~TStack();  TStack<T>& operator =(TStack<T>& \_v);  void Push(T d);  void SetData(T\* \_x, int size, int \_top);  void Resize(int \_size);  T Get();  int IsEmpty(void) const;  int IsFull(void) const;  int GetSize();  int GetCount();  inline int min\_elem();  inline int max\_elem();  template <class T1>  friend ostream& operator<< (ostream& ostr, const TStack<T1>& A);  template <class T1>  friend istream& operator >> (istream& istr, TStack<T1>& A);  };  template <class T1>  ostream& operator<< (ostream& ostr, const TStack<T1>& A) {  for (int i = 0; i < A.top; i++) {  ostr << A.data[i] << endl;  }  return ostr;  }  template <class T1>  istream& operator >> (istream& istr, TStack<T1>& A) {  int count;  istr >> count;  for (int i = 0; i < A.count; i++) {  T1 d;  istr >> d;  A.Push(d);  }  return istr;  }  template<class T>  inline TStack<T>::TStack(int size, bool \_f)  {  if (size > 0)  {  this->length = size;  this->f = \_f;  if (f)  {  data = new T[length];  for (int i = 0; i < length; i++)  {  data[i] = 0;  }  }  this->top = 0;  }  else  {  throw logic\_error("Error");  }  }  template <class T>  TStack<T>::TStack(TStack<T>& \_v)  {  length = \_v.length;  top = \_v.top;  f = \_v.f;  if (f)  {  data = new T[length];  for (int i = 0; i < length; i++)  {  data[i] = \_v.data[i];  }  }  else  {  data = \_v.data;  }  }  template <class T>  TStack<T>::~TStack()  {  length = 0;  if (f)  {  if (data != NULL)  {  delete[] data;  data = 0;  }  }  }  template<typename T>  inline int TStack<T>::max\_elem()  {  T res = data[0];  for (int i = 1; i < length; i++)  {  if (data[i] > res)  {  res = data[i];  }  }  return res;  }  template<typename T>  inline int TStack<T>::min\_elem()  {  T res = data[0];  for (int i = 1; i < length; i++)  {  if (data[i] < res)  {  res = data[i];  }  }  return res;  }  template <class T>  TStack<T>& TStack<T>::operator =(TStack<T>& \_v)  {  if (this == &\_v)  return \*this;  this->length = \_v.length;  f = \_v.f;  if (f)  {  if (data != NULL)  {  delete[] data;  }  this->data = new T[length];  for (int i = 0; i < length; i++)  {  data[i] = \_v.data[i];  }  }  else  {  data = \_v.data;  }  this->top = \_v.top;  return \*this;  }  template<class T>  inline void TStack<T>::Push(T d)  {  if (top - 1 > length)  {  throw "Error";  }  data[top] = d;  top++;  }  template<class T>  inline void TStack<T>::SetData(T\* \_x, int size, int \_top)  {  if (f)  {  if (data != NULL)  {  delete[] data;  }  this->length = size;  f = false;  data = \_x;  top = \_top;  }  }  template<class T>  inline void TStack<T>::Resize(int \_size)  {  if (\_size <= 0)  {  throw logic\_error("Error");  }  if (data == NULL)  {  delete[] data;  data = new T[\_size];  }  else  {  T\* temp = new T[\_size];  for (int i = 0; i < this->GetCount(); i++)  {  temp[i] = Get();  }  delete[] data;  data = new T[\_size];  for (int i = 0; i < 1; i++)  {  data[i] = temp[i];  }  }  }  template<class T>  inline T TStack<T>::Get()  {  if (top <= 0)  {  throw logic\_error("Error");  }  T d = data[top - 1];  top--;  return d;  }  template<class T>  inline int TStack<T>::IsEmpty(void) const  {  return (top == 0);  }  template<class T>  inline int TStack<T>::IsFull(void) const  {  return (top - 1 > length);  }  template <class T>  int TStack<T>::GetSize()  {  return length;  }  template<class T>  inline int TStack<T>::GetCount()  {  return top;  }  #endif |

## 8.2. Приложение 2:Класс MultiStack

## 

|  |
| --- |
| **MultiStack.h** |
| #ifndef \_MULTISTACK\_  #define \_MULTISTACK\_  #include <iostream>  #include <fstream>  #include <math.h>  #include"Steck.h"  using namespace std;  template <class T>  class TMultiStack  {  protected:  int length;  int stackCount;  TStack<T>\* stacks;  T\*\* oldData;  T\* data;  void StackRelocation(int i);  public:  TMultiStack(int size = 1, int stackCount = 1);  TMultiStack(TMultiStack<T>& \_v);  ~TMultiStack();  TMultiStack<T>& operator =(TMultiStack<T>& \_v);  T Get(int i);  void Push(T d, int i);  bool IsEmpty(int i) const;  bool IsFull(int i) const;  int GetSize();  void Resize(int size = 1, int stackCount = 1);  template <class T1>  friend ostream& operator<< (ostream& ostr, const TMultiStack<T1>& A);  template <class T1>  friend istream& operator >> (istream& istr, TMultiStack<T1>& A);  };  template<class T1>  inline istream& operator>>(istream& istr, TMultiStack<T1>& A)  {  int stCount = 0;  istr >> stCount;  int size = 0;  istr >> size;  A.Resize(size, stCount);  for (int i = 0; i < A.stCount; i++)  {  istr >> A.stacks[i];  }  return istr;  }  template<class T1>  inline ostream& operator<<(ostream& ostr, const TMultiStack<T1>& A)  {  for (int i = 0; i < A.stackCount; i++)  {  ostr << (A.stacks[i]) << endl;  }  return ostr;  }  #endif |

## 8.3. Приложение 3:Тесты для классов

## 

|  |
| --- |
| **test\_MultiStack.cpp** |
| #include "MultiStack.h"  #include "MultiStack.cpp"  #include "../gtest/gtest.h"  TEST(TMultiStack, can\_create\_MultiStack\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TMultiStack<int> mst(9, 3));  }  TEST(TMultiStack, cant\_create\_MultiStack\_with\_null)  {  ASSERT\_ANY\_THROW(TMultiStack<int> mst(0));  }  TEST(TMultiStack, check\_empty\_MultiStack)  {  TMultiStack<int> mst(9, 3);  EXPECT\_EQ(true, mst.IsEmpty(1));  }  TEST(TMultiStack, check\_for\_non\_existent\_empty\_MultiStack)  {  TMultiStack<int> mst(9, 3);  ASSERT\_ANY\_THROW(mst.IsEmpty(4));  }  TEST(TMultiStack, check\_negative\_empty\_MultiStack)  {  TMultiStack<int> mst(9, 3);  ASSERT\_ANY\_THROW(mst.IsEmpty(-5));  }  TEST(TMultiStack, check\_IsFull\_MultiStack)  {  TMultiStack<int> mst(9, 3);  EXPECT\_EQ(false, mst.IsFull(1));  }  TEST(TMultiStack, check\_for\_non\_existent\_IsFull\_MultiStack)  {  TMultiStack<int> mst(9, 3);  ASSERT\_ANY\_THROW(mst.IsFull(4));  }  TEST(TMultiStack, check\_negative\_IsFull\_MultiStack)  {  TMultiStack<int> mst(9, 3);  ASSERT\_ANY\_THROW(mst.IsFull(-5));  }  TEST(TMultiStack, can\_push\_with\_mem)  {  TMultiStack<int> mst(9, 3);  ASSERT\_NO\_THROW(mst.Push(10, 1));  }  TEST(TMultiStack, cant\_push\_with\_no\_mem)  {  TMultiStack<int> mst(9, 3);  ASSERT\_ANY\_THROW(mst.Push(10, 4));  }  TEST(TMultiStack, can\_copy\_stack)  {  TMultiStack<int> mst1(9, 3);  mst1.Push(10, 1);  ASSERT\_NO\_THROW(TMultiStack<int> mst2(mst1));  }  TEST(TMultiStack, can\_assing\_stack)  {  TMultiStack<int> mst1(9, 3);  mst1.Push(10, 1);  TMultiStack<int> mst2(9, 3);  mst2 = mst1;  EXPECT\_EQ(10, mst2.Get(1));  }  TEST(TMultiStack, can\_get)  {  TMultiStack<int> mst(9, 3);  mst.Push(10, 1);  ASSERT\_NO\_THROW(mst.Get(1));  }  TEST(TMultiStack, cant\_get\_nonexistent)  {  TMultiStack<int> mst(9, 3);  mst.Push(10, 1);  ASSERT\_ANY\_THROW(mst.Get(-4));  }  TEST(TMultiStack, cant\_get\_negative)  {  TMultiStack<int> mst1(9, 3);  mst1.Push(10, 1);  ASSERT\_ANY\_THROW(mst1.Get(4));  }  TEST(TMultiStack, can\_GetSize)  {  TMultiStack<int> mst1(9, 3);  ASSERT\_NO\_THROW(mst1.GetSize());  }  TEST(TMultiStack, can\_Resize)  {  TMultiStack<int> mst1(1, 1);  mst1.Push(10, 0);  ASSERT\_NO\_THROW(mst1.Resize(5, 0));  }  TEST(TMultiStack, cant\_Resize\_negative)  {  TMultiStack<int> mst1(1, 1);  mst1.Push(10, 0);  ASSERT\_ANY\_THROW(mst1.Resize(-5, 0));  }  TEST(TMultiStack, can\_StackRelocation)  {  TMultiStack<int> mst(9, 3);  mst.Push(11, 0);  mst.Push(12, 1);  mst.Push(12, 1);  mst.Push(12, 1);  ASSERT\_NO\_THROW(mst.Push(15, 1));  } |