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**

**Отчет по учебной практике**

**«Структура хранения данных: Стек на массиве»**

**Выполнил:** студент группы 381903-3

Стеценко Данил Михайлович

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ Лебедев И.Г

Нижний Новгород

2020

Оглавление

[1. Введение. 2](#_Toc533106246)

[2. Постановка задачи. 3](#_Toc533106247)

[3. Руководство пользователя. 4](#_Toc533106248)

[4. Руководство программиста. 5](#_Toc533106249)

[4.1. Описание структуры программы. 5](#_Toc533106250)

[4.2. Описание структур данных. 5](#_Toc533106251)

[4.3. Описание алгоритмов. 6](#_Toc533106252)

[5. Заключение. 7](#_Toc533106261)

[6. Литература. 8](#_Toc533106262)

**7.** Приложения………………………………………………………………………….9

# Введение.

**Стек** — [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), представляющий собой [список элементов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), организованных по принципу FILO – первым вошел, последним вышел. Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Из стека нельзя извлечь произвольный элемент. Есть только вершина стека, и мы можем или положить туда элемент, или забрать его оттуда.

# Постановка задачи.

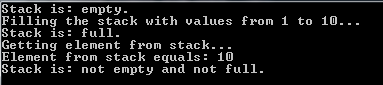
В данной лабораторной работе нужно разработать эффективную библиотеку для хранения и работы с такой структурой данных, как стек.

Для этого нам нужно:

* Описать и реализовать класс **TStack**.
* Протестировать класс **TStack** с помощью Google Test.
* Продемонстрировать работу класса **TStack**.

# Руководство пользователя.

Пример работы класса **TStack**:



*Рис 1. Пример работы программы*

Программа работает следующим образом:

* Создается стек (в данном примере его размер равен 10).
* Проверяем стек на пустоту.
* Заполняем стек значениями от 1 до 10.
* Проверяем стек на полноту.
* Забираем элемент с вершины стека.
* Проверяем стек на полноту и пустоту.

# Руководство программиста.

# Описание структуры программы.

Программа состоит из модулей:

* **Stack** – содержит в себе файл **stack\_main.cpp** с реализацией примера использования класса **TStack.**
* **StackLib –** содержит в себе файл **Stack.h**, в котором описан и реализован класс **TStack**.
* **Test** – содержит в себе файл **test\_tstack.cpp**, в котором находится набор тестов, для проверки работоспособности класса **TStack**.

# Описание структур данных.

**Класс TStack.**

Класс **TStack** является шаблонным и содержит два поля со спецификатором **protected**:

* **int top** – вершина стека.
* **int size** – размер стека.
* **T\* mas** – указатель на область памяти для хранения стека.

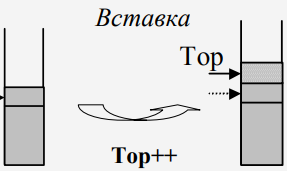
Далее идут методы класса со спецификатором доступа **public**:

* **TStack(int \_size=0)** – конструктор инициализатор.
* **TStack(TStack<T> &obj)** – конструктор копирования.
* **~TStack()** – деструктор.
* **void Put(T elem)** – функция, помещающая элемент в вершину стека.
* **T Get()** – функция, возвращающая элемент с вершины стека.
* **bool IsEmpty()** – функция, проверяющая стек на пустоту.
* **bool IsFull()** – функция, проверяющая стек на полноту.

# Описание алгоритмов.

**Добавление элемента в стек.**

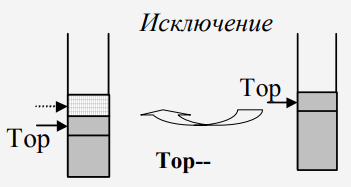
Добавляя элемент в стек, мы помещаем его в вершину стека, на которую указывает поле **top**. После добавления элемента в стек, значение поля **top** увеличивается на 1.



*Рис. 2 Добавление элемента в стек.*

**Удаление элемента из стека.**

Удаляя элемент из стека, мы забираем его из первой непустой ячейки. На эту непустую ячейку указывает поле **top** со значением, уменьшенным на 1.



*Рис. 3 Удаление элемента из стека.*

# Заключение.

В данной лабораторной работе мне удалось реализовать библиотеку для хранения и работы со стеком, а именно:

* Удалось реализовать класс **TStack**.
* Удалось протестировать методы класса **TStack**, а также обеспечить их работоспособность.

Таким образом, я смог реализовать структуру данных под названием стек.

# Литература.

1. Ссылка из Википедии про стек:

<https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA>

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015.

# Приложения

|  |
| --- |
| **Steck.h** |
| #ifndef STACK\_H  #define STACK\_H  #include <iostream>  #include <fstream>  using namespace std;  template <typename T>  class TStack  {  private:  T\* stackPtr;  const int size;  int num;  public:  TStack(int = 25);  TStack(const TStack<T>&);  ~TStack();  inline int IsEmpty(void) const;  inline int IsFull(void) const;  inline void put(const T&);  inline T deleteElem();  inline const T& Peek(int) const;  inline int getStackSize() const;  inline T\* getPtr() const;  inline int getNum() const;  inline int min\_elem();  inline int max\_elem();  friend ostream& operator<<(ostream& out, const TStack& st)  {  for (int ix = st.num - 1; ix >= 0; ix--)  cout << st.stackPtr[ix] << endl;  return out;  }  };  template <typename T>  TStack<T>::TStack(int maxSize) : size(maxSize)  {  if (maxSize < 0)  {  throw logic\_error("ERROR");  }  stackPtr = new T[size];  num = 0;  }  template <typename T>  inline int TStack<T>::IsEmpty() const  {  return stackPtr == NULL;  for (int i = 0; i < size; i++)  {  if (stackPtr[i] == 0)  {  continue;  }  else  {  throw logic\_error("ERROR");  }  }  }  template <typename T>  inline int TStack<T>::IsFull() const  {  for (int i = 0; i < size; i++)  {  if (stackPtr[i] != 0)  {  continue;  }  else  {  throw logic\_error("ERROR");  }  }  }  template <typename T>  TStack<T>::TStack(const TStack<T>& otherStack) : size(otherStack.getStackSize())  {  stackPtr = new T[size];  num = otherStack.getNum();  for (int ix = 0; ix < num; ix++)  {  stackPtr[ix] = otherStack.getPtr()[ix];  }  }  template <typename T>  TStack<T>::~TStack()  {  if (this->stackPtr != NULL)  {  delete[] stackPtr;  }  num = 0;  }  template <typename T>  inline void TStack<T>::put(const T& value)  {  if (num > size - 1 || num < 0)  {  throw logic\_error("ERROR");  }  stackPtr[num++] = value;  }  template <typename T>  inline T TStack<T>::deleteElem()  {  if (num < NULL)  {  throw logic\_error("ERROR");  }  stackPtr[--num];  return stackPtr[num];  }  template <class T>  inline const T& TStack<T>::Peek(int Elem) const  {  if (Elem > num)  {  throw logic\_error("ERROR");  }  return stackPtr[num - Elem];  }  template <typename T>  inline int TStack<T>::getStackSize() const  {  return size;  }  template <typename T>  inline T\* TStack<T>::getPtr() const  {  return stackPtr;  }  template <typename T>  inline int TStack<T>::getNum() const  {  return num;  }  template<typename T>  inline int TStack<T>::max\_elem()  {  int res = stackPtr[0];  for (int i = 1; i < size; i++)  {  if (stackPtr[i] > res)  {  res = stackPtr[i];  }  }  return res;  }  template<typename T>  inline int TStack<T>::min\_elem()  {  int res = stackPtr[0];  for (int i = 1; i < size; i++)  {  if (stackPtr[i] < res)  {  res = stackPtr[i];  }  }  return res;  }  #endif |