МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им.

Р.Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий

Кафедра «Информационные радиосистемы»

**Индуктивные операции и функции высших порядков**

**(Вариант 28)**

Выполнил:

Студент гр. 22-Рз Наумов А.А.

Проверил:

к.т.н., доцент кафедры ИРС Сидоров С.Б.

Нижний Новгород,

2024 г.

Оглавление

[1. Постановка задачи 3](#__RefHeading___Toc2465_4070117755)

[2. Архитектура программной системы 4](#__RefHeading___Toc2467_4070117755)

[Application 4](#__RefHeading___Toc6530_2651826584)

[Vector 5](#__RefHeading___Toc6532_2651826584)

[3. Алгоритм обработки данных 5](#__RefHeading___Toc3121_2204344327)

[Приложение 1. 6](#__RefHeading___Toc3283_4070117755)

[Релизная версия 1.0 6](#__RefHeading___Toc3133_22043443271)

[//main.cpp 6](#__RefHeading___Toc3187_2651826584)

[//application.h 6](#__RefHeading___Toc3189_2651826584)

[//vector.h 6](#__RefHeading___Toc3191_2651826584)

[//application.cpp 7](#__RefHeading___Toc3193_2651826584)

[//vector.cpp 8](#__RefHeading___Toc3195_2651826584)

[Промежуточная версия 0.9 9](#__RefHeading___Toc3133_2204344327)

[//main.cpp 9](#__RefHeading___Toc3197_2651826584)

[//application.h 10](#__RefHeading___Toc3137_2204344327)

[//IrNumber.h 10](#__RefHeading___Toc3139_2204344327)

[//application.cpp 10](#__RefHeading___Toc3141_2204344327)

[//processing.cpp 11](#__RefHeading___Toc3143_2204344327)

[Промежуточная версия 0.5 12](#__RefHeading___Toc3285_4070117755)

[//main.cpp 12](#__RefHeading___Toc3145_2204344327)

[//header.h 12](#__RefHeading___Toc3147_2204344327)

[//initVector.cpp 13](#__RefHeading___Toc3149_2204344327)

[//processing.cpp 13](#__RefHeading___Toc3151_2204344327)

[Промежуточная версия 0.3 13](#__RefHeading___Toc3287_4070117755)

[//main.cpp 13](#__RefHeading___Toc3153_2204344327)

[//header.h 14](#__RefHeading___Toc3155_2204344327)

[//initVector.cpp 14](#__RefHeading___Toc3157_2204344327)

[//processing.cpp 14](#__RefHeading___Toc3159_2204344327)

# Задание по варианту

Обнаружение наиболее длинного участка монотонного возрастания значений последовательных элементов, при условии что разность значений последнего и первого элементов участка не менее чем D. Результатом является интервал: (начальный номер, конечный номер). Тип элемента — целочисленный.

# Контрольная работа № 1.

# «Реализация индуктивной обработки

# последовательности элементов»

## 1.1. Постановка задачи

Освоение способов разработки алгоритмов выполнения индуктивных операций над последовательностью данных, построение индуктивных функций методом индуктивных расширений. Изучение общей схемы программной реализации индуктивной функции и схемы обработки последовательности элементов с использованием индуктивной функции.

## 1.2. Алгоритм индуктивной обработки

В начале работы программы, у нас в распоряжении имеется абстрактный тип данных (далее АТД), в котором нам присутствует вектор изначальных значений, считываемых с стандартного устройства ввода, переменная D. Переменные для хранения финальных значений, переменная для хранения стрика текущей обрабатываемой последовательности. Подробнее мы их рассмотрим в пункте 1.3.

Пойдем по порядку алгоритма индуктивной обработки. В самом начале, у целочисленных переменных app.finalLeft и app.finalRight приравнены к INT\_MAX (**2.147.483.647), макросом из заголовочного файла climits, имеющим максимальное значение типа данных int в 32х битной системе. Этим мы можем отследить, изменялась ли переменная в течение работы программы. Так же, задавать поведение строго в первой итерации цикла.**

**Таким образом, значение *xn ,*будет обрабатываться циклом for(), для каждого элемента входной последовательности. Цикл ходит от первого элемента вектора входных значений, до последнего элемента.**

**Перед началом углубления в функцию обработки, стоит сказать, что, местами, в очереди условий используется сравнение с последующим объектом. Это обусловлено методом обработки полученных значений. Следуя логике представленного программного кода, сначала считываются все значения с стандартного устройства ввода, и только после этого начинается стадия обработки.**

**За индуктивную обработку отвечает функция** bool appProcessDataIntoFinalResult (Application &app) функция возвращает логическое значение true/false, в зависимости от успешности получения искомых индексов элементов входной последовательности. В начале, целочисленной переменной tempConsequenceStreak присваивается значение 0, что знаменует собой, на данный момент, отсутствие, подходящей возрастающей последовательности. После этого вводится цикл, о нем было сказано выше. При условии, что текущий элемент меньше последующего, последовательность имеет больше одного элемента, и он не является последним, значение переменной счётчика стрика текущей последовательности увеличивается на 1. Соответственно, при нарушении одного из условий, мы падаем во внутрь условия первого уровня.

Первый уровень условий отвечает за «грубый» отбор элементов, по кол-ву элементов последовательности, значению итератора и условии превосходства значения последующего элемента над текущим. В середине условия находится второй уровень проверки, его рассмотрим чуть позже. В конце расположена операция сброса стрика последовательности. Таким образом, данное условие можно представить в виде математической записи в следующем виде: (Будем считать, что n\_min и n\_max — минимальный и максимальный элемент входной последовательности, соответственно. Так же, что tempConsequenceStreak = tempCS и finalConsequenceStreak = finalCS.), в следствие, обратное условие будет: , в таком случае, значения результирующих индексов не изменяются, а к счётчику tempConsequenceStreak, добавляется 1. Так же, если значение упало в это условие, то по окончании, вне зависимости от исхода условий глубже, счётчик последовательности tempConsequenceStreak сбрасывается на 0. И так, второй уровень условий.

Он представляет собой группу проверок, первая из которых: превосходство значения текущего стрика (tempConsequenceStreak) над финальным, хранящемся в АТД Application, finalConsequecneStreak И превосходства разницы значений последнего и первого элемента подходящей последовательности над введённой константой D (constD). Вторая группа вводится в параллельном режиме, логическим оператором ИЛИ, и используется для случая когда, в следствие обработки последовательности должны получиться индексы 0, 1. В математическом выражении второй уровень будет выглядеть следующим образом: . На третьем уровне условий, мы приближаемся к присвоению финальных индексов.

Условие третьего уровня отвечает за выбор метода присвоения левого индекса. в следствие попадания проверяемого элемента под условие, левый индекс finalLeft получит значение , в противном случае, значение . Вне зависимости от этого, правому индексу finalRight присваивается значение текущего индекса - , и текущий стрик из tempCS записывается в переменную финального стрика — finalCS.

Если по окончании цикла for(), значения финальных индексов небыли изменены, и размер входной последовательности равен, то в качестве финальных индексов присваиваются 0/0 finalLeft/finalRight соответственно.

Если, тем не менее, ни одно из условий не помогло установить наличие возрастающей последовательности, удовлетворяющей нашим условиям, в частности, индексу D, то функция возвращает false флаг, и исполнение программы прекращается с текстом ошибки “DATA INPUT FAILURE”. В противном случае, возвращается флаг true, и программа продолжает выполнение, согласно заданным функциям.

Работоспособность представленной логики была проверена с помощью google тестов, в количестве 22 штук, от самых тривиальных, до усложнённых с несколькими последовательностями, включая значения в отрицательной части. Тесты находятся в приложении 2.

## 1.3. Архитектура программной реализации вычислителя.

Разберём используемые абстрактные типы данных.

## Application

Определяется структурный тип данных **Application**, содержащий четыре поля: **Vector Initial, Vector final, unsigned n, unsigned R.**

В заголовочном файле *application.h* объявляется глобальная функция

*int app\_run(application& app)*. Эта функция отвечает за исполнение внутренних операций АТД *application,* а именно: получение необходимых данных от пользователя, обработка полученных значений, и вывод результата в консоль. При успешном выполнении возвращает значение 0, если произошла ошибка при вводе данных, возвращает значение 1.

За выполнение указанных выше задач в файле application.cpp объявляются функции с модификатором *static:*

*static* bool appGetN(Application& app);

static bool appGetR(Application& app);

static bool appGetXY(Application& app, unsigned n);

static bool appProcessVector(Application& app);

static void appOutputResult(Application& app);

Рассмотрим каждую функцию в отдельности:

*static* bool appGetN(Application& app); - Отвечает за получение числа n от пользователя.

static bool appGetR(Application& app); - Отвечает за получение числа R от пользователя.

static bool appGetXY(Application& app, unsigned n); - Отвечает за получение пар X/Y от пользователя.

static bool appProcessVector(Application& app); - Отвечает за обработку полученных значений, и подсчёт пар, подпадающих под условие задачи.

static void appOutputResult(Application& app); - Отвечает за вывод результирующих данных обратно пользователю.

## Vector

В этом модуле определяется структурный тип данных для представления n-мерного массива *Vector*, содержащий два поля, coordX и coordY, представленного типом *vector* из стандартной библиотеки с элементами типа *int* (целочисленные) и поле unsigned counter, которое используется для валидации вывода.

В заголовочном файле *vector.h* объявляются функции:

void vectorResize(Vector& v, unsigned size);

Vector vectorProcess(const Vector& initial, const Vector& final, unsigned R, unsigned n);

unsigned vectorGetSizeOfFinal(Vector& v);

Рассмотрим эти функции в отдельности:

void vectorResize(Vector& v, unsigned size); - Отвечает за изменение размера массива.

Vector vectorProcess(const Vector& initial, const Vector& final, unsigned R, unsigned n); - Отвечает за обработку данных векторов.

unsigned vectorGetSizeOfFinal(Vector& v); - Отвечает за получение количества элементов финального вектора.

## 

# Контрольная работа №2

# «Настройка индуктивного вычислителя

# с использованием функции обратного вызова»

# Приложение 1.

## Версия 1.0

*//main.cpp  
//  
// Created by Anatejl on 15.04.2024.  
//*#include "application.h"  
#include <iostream>  
  
int main() {  
  
 std::cout << "An unarguably valuable piece of software, THE diamond!" << std::endl;  
  
 Application app;  
  
 int ret = appRun(app);  
  
 return ret;  
}

*//application.h  
//  
// Created by Anatejl on 15.04.2024.  
//*#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H  
  
#include <climits>  
#include "vector.h"  
  
*//Data for program to handle.*struct Application {  
  
 Vector valueArray;  
 int constD;  
  
 int finalLeft = INT\_MAX;  
 int finalRight = INT\_MAX;  
 int finalConsequenceStreak = 0;  
};  
  
*// To execute application*int appRun(Application& app);  
  
bool appInitializeData(Application &app);  
bool appGetConstantD(Application &app);  
bool appProcessDataIntoFinalResult(Application &app);  
bool appGetOutputToUser(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

*//application.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "application.h"  
#include "vector.h"  
#include <iostream>  
  
  
int appRun(Application &app) {  
  
 if (!appGetConstantD(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!appInitializeData(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!appProcessDataIntoFinalResult(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl << "No matches applicable." << std::endl;  
 return 1;  
 }  
  
 if (!appGetOutputToUser(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 return 0;  
}  
  
bool appGetConstantD(Application &app) {  
  
 std::cout << "Input a D constant to compare:" << std::endl;  
 std::cin >> app.constD;  
 std::cout << app.constD << std::endl;  
  
 return true;  
}  
  
bool appInitializeData(Application &app) {  
  
 app.valueArray = vectorValueArrayInitialize(app.valueArray);  
 std::cout << "Input array has been successfully processed." << std::endl;  
  
 return true;  
}  
  
bool appProcessDataIntoFinalResult(Application &app) {  
  
 int tempConsequenceStreak = 0;  
  
 for (int i = 0; i < vectorGetSize(app.valueArray); ++i) {  
 if (  
 (i != vectorGetSize(app.valueArray)-1 && app.valueArray.value[i] > app.valueArray.value[i + 1])  
 ||  
 (i == vectorGetSize(app.valueArray)-1)  
 ){  
  
 if ((tempConsequenceStreak > app.finalConsequenceStreak &&  
 app.valueArray.value[i] -app.valueArray.value[i - (tempConsequenceStreak-1)] > app.constD) ||  
 app.valueArray.value[i] -app.valueArray.value[i - tempConsequenceStreak] > app.constD)  
 {  
 if((app.finalLeft == INT\_MAX && app.finalRight == INT\_MAX) &&  
 (i == tempConsequenceStreak || i == tempConsequenceStreak+1)){  
 app.finalLeft = i - tempConsequenceStreak;  
 }  
 else{  
 app.finalLeft = i - (tempConsequenceStreak-1);  
 }  
 app.finalRight = i;  
 app.finalConsequenceStreak = tempConsequenceStreak;  
 }  
  
 tempConsequenceStreak = 0;  
 }  
  
 ++tempConsequenceStreak;  
 }  
  
 if (app.finalLeft == INT\_MAX && app.finalRight == INT\_MAX && vectorGetSize(app.valueArray) == 1) {  
 app.finalLeft = 0;  
 app.finalRight = 0;  
 }  
  
 if (app.finalLeft == INT\_MAX && app.finalRight == INT\_MAX) {  
 return false;  
 }  
  
 return true;  
}

bool appGetOutputToUser(Application &app) {  
  
 std::cout << std::endl << app.finalLeft << " - Left Index" << std::endl;  
 std::cout << app.finalRight << " - Right Index" << std::endl;  
  
 return true;  
}

*//vector.h  
//  
// Created by Anatejl on 20.04.2024.  
//*#ifndef NNTU\_VECTOR\_H  
#define NNTU\_VECTOR\_H  
  
#include <vector>  
  
struct Vector{  
  
 std::vector<int> value;  
  
};  
  
Vector vectorValueArrayInitialize(const Vector& valueArray);  
int vectorGetSize(Vector& v);  
  
#endif *//NNTU\_VECTOR\_H*

*//vector.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "vector.h"  
#include <iostream>  
  
Vector vectorValueArrayInitialize(const Vector &valueArray) {  
  
 Vector temporaryVector;  
 int counter = 0;  
  
 while (!std::cin.eof()) {  
  
 int temporary\_value\_storage = 0;  
  
 for (int k = 0; k <= counter; k++) {  
  
 std::cin >> temporary\_value\_storage;  
 temporaryVector.value.push\_back(temporary\_value\_storage);  
  
 counter++;  
  
 if (std::cin.eof()) {  
 break;  
 }  
 }  
 }  
  
 *//Demonstrate input data* std::cout << "Provided vector consists of " << vectorGetSize(temporaryVector) << " entries." << std::endl;  
 std::cout << "Values as follows:" << std::endl;  
  
 for (int i : temporaryVector.value) {  
 std::cout << i << " ";  
 }  
 std::cout << std::endl;  
  
 return temporaryVector;  
  
}  
  
int vectorGetSize(Vector &v) {  
  
 return v.value.size();  
}

# Приложение 2.

Тесты для программы из приложения 1.

*//test.cpp*

*//*  
*// Created by Anatejl on 30.04.2024.*  
*//*  
#ifndef INDUCTANCE\_V2\_APPLICATION\_TESTS\_CPP  
#define INDUCTANCE\_V2\_APPLICATION\_TESTS\_CPP  
  
#include "E:\Git\NNTU\_INF\googletest\googlemock\include\gmock\gmock.h"  
#include "E:\Git\NNTU\_INF\googletest\googletest\include\gtest\gtest.h"  
  
#include "application.h"  
  
using namespace std;  
  
Application init\_application(std::vector<int> &&init\_values, int D) {  
 Application test\_application;  
 test\_application.constD = D;  
 test\_application.valueArray.value.reserve(init\_values.size());  
 for (auto &val: init\_values) {  
 test\_application.valueArray.value.push\_back(val);  
 }  
  
 return test\_application;  
}  
  
TEST(IndexFindCorrectly, Simple\_1) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 4, 5}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(4, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_2) {  
Application test\_app = init\_application(std::vector<int>{1}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(0, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_3) {  
Application test\_app = init\_application(std::vector<int>{1, 123}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(1, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_4) {  
Application test\_app = init\_application(std::vector<int>{11, 12, 11}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(1, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_5) {  
Application test\_app = init\_application(std::vector<int>{1, 4, 1, 3, 6}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(2, test\_app.finalLeft);  
ASSERT\_EQ(4, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_6) {  
Application test\_app = init\_application(std::vector<int>{4, 3, 2, 1}, 0);  
ASSERT\_FALSE(appProcessDataIntoFinalResult(test\_app));  
}  
  
TEST(IndexFindCorrectly, Simple\_7) {  
Application test\_app = init\_application(std::vector<int>{4, 3, 2, 1, 3, 2}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(3, test\_app.finalLeft);  
ASSERT\_EQ(4, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_8) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 1, 1, 1, 1, 2, 1}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(3, test\_app.finalLeft);  
ASSERT\_EQ(7, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_9) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 3, 2, 1, 1}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(3, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_10) {  
Application test\_app = init\_application(std::vector<int>{6, 4, 2, 1, 5, 8, 2, 3, 7, 9, 2, 3, 4, 2}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(6, test\_app.finalLeft);  
ASSERT\_EQ(9, test\_app.finalRight);  
}  
  
TEST(IndexFindCorrectly, Simple\_11) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 2}, 0);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(0, test\_app.finalLeft);  
ASSERT\_EQ(2, test\_app.finalRight);  
}  
  
TEST(CannotFindIndex, D\_Bigger\_1) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 4, 5}, 10);  
ASSERT\_FALSE(appProcessDataIntoFinalResult(test\_app));  
  
}  
  
TEST(CannotFindIndex, D\_Bigger\_2) {  
Application test\_app = init\_application(std::vector<int>{16, 1, 3, 4, 5, 1, 9}, 10);  
ASSERT\_FALSE(appProcessDataIntoFinalResult(test\_app));  
}  
  
  
TEST(CannotFindIndex, D\_Bigger\_3) {  
Application test\_app = init\_application(std::vector<int>{5, 4, 3, 2}, 1);  
ASSERT\_FALSE(appProcessDataIntoFinalResult(test\_app));  
}  
  
TEST(IndexWithD, Filttered\_by\_D\_1) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 3, 4, 5, 1, 12, 13}, 10);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(5, test\_app.finalLeft);  
ASSERT\_EQ(7, test\_app.finalRight);  
}  
  
TEST(IndexWithD, Filttered\_by\_D\_2) {  
Application test\_app = init\_application(std::vector<int>{1, 2, 2, 2, 2, 2, 1, 5, 2, 3, 3, 3, 4, 4, 5, 2, 2}, 1);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(8, test\_app.finalLeft);  
ASSERT\_EQ(14, test\_app.finalRight);  
}  
  
TEST(IndexWithD, Filttered\_by\_D\_3) {  
Application test\_app = init\_application(std::vector<int>{5, 1, 9, 1, 2, 3, 10}, 7);  
ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
ASSERT\_EQ(3, test\_app.finalLeft);  
ASSERT\_EQ(6, test\_app.finalRight);  
}  
  
TEST(NegativeUInt, NegativeUInt) {  
 unsigned int huy = -1;  
 ASSERT\_EQ(-1, huy);  
}  
  
TEST(NegativeUInt, Negative\_unit\_becomes\_false) {  
 int op = -1;  
 int op2 = 1;  
 ASSERT\_TRUE(op2);  
}  
  
TEST(Custom, One) {  
 Application test\_app = init\_application(std::vector<int>{1, 2, 3, 4, 1, 3, 1, INT\_MAX - 5, INT\_MAX - 4, INT\_MAX - 3, INT\_MAX - 2, INT\_MAX - 1}, 0);  
 ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
 ASSERT\_EQ(6, test\_app.finalLeft);  
 ASSERT\_EQ(11, test\_app.finalRight);  
}  
  
TEST(Custom, Two) {  
 Application test\_app = init\_application(std::vector<int>{-4, -3, -2, -1, -3, -2, -6, INT\_MIN + 5, INT\_MIN + 6, INT\_MIN + 7, INT\_MIN + 8, INT\_MIN + 9, INT\_MIN + 12, INT\_MIN + 25}, 0);  
 ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
 ASSERT\_EQ(7, test\_app.finalLeft);  
 ASSERT\_EQ(13, test\_app.finalRight);  
}  
  
TEST(Custom, Three) {  
 Application test\_app = init\_application(std::vector<int>{INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3, INT\_MIN + 3}, -1);  
 ASSERT\_TRUE(appProcessDataIntoFinalResult(test\_app));  
  
 ASSERT\_EQ(0, test\_app.finalLeft);  
 ASSERT\_EQ(7, test\_app.finalRight);  
}  
  
int main(int argc, char \*\*argv) {  
 ::testing::InitGoogleTest(&argc, argv);  
  
 return RUN\_ALL\_TESTS();  
}  
  
  
#endif *//INDUCTANCE\_V2\_APPLICATION\_TESTS\_CPP*