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

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 x_n, 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`.)$\left(n \neq \left(n_{max}\right) - 1 \wedge \left(x_n > x_{(n+1)}\right)\right) \vee n \neq \left(n_{max}\right) - 1$, в следствие, обратное условие будет: $x_n < x_{(n+1)}$, 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: $tempCS > finalCS \wedge \left(x_n - x_{(n-(tempCS-1))} > D\right) \vee \left(x_n - x_{(n-(tempCS))} > D\right)$. На третьем уровне условий, мы приближаемся к присвоению финальных индексов.

Условие третьего уровня отвечает за выбор метода присвоения левого индекса. $(finalLeft = \text{INT_MAX} \wedge finalRight = \text{INT_MAX}) \wedge (n = tempCS \vee n = tempCS + 1)$в следствие попадания проверяемого элемента под условие, левый индекс `finalLeft` получит значение $n - tempCS$, в противном случае, значение $n - (tempCS - 1)$. Вне зависимости от этого, правому индексу `finalRight` присваивается значение текущего индекса - n, и текущий стрик из `tempCS` записывается в переменную финального стрика — `finalCS.`

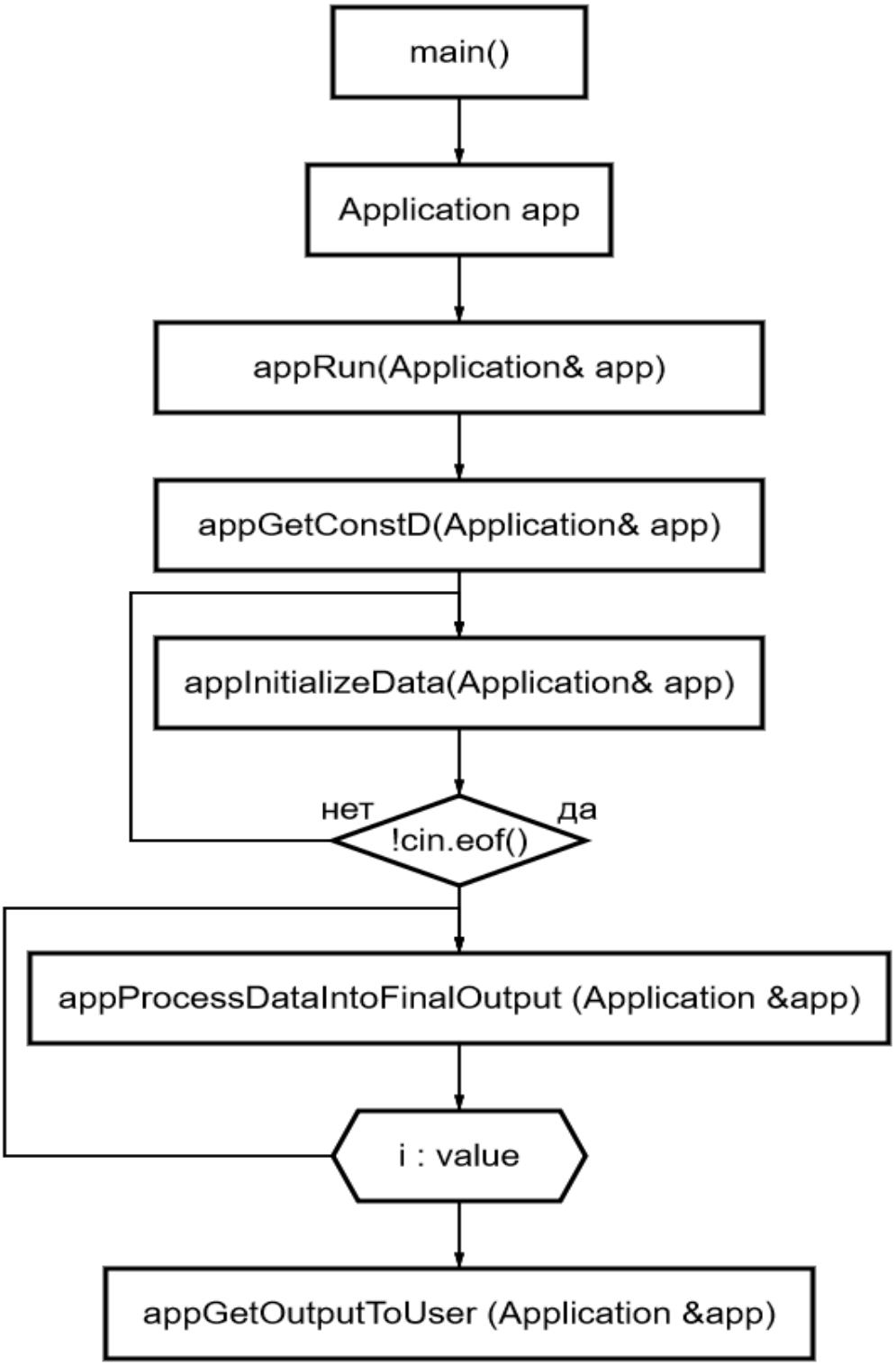
Если по окончании цикла `for()`, значения финальных индексов небыли изменены, и размер входной последовательности равен, то в качестве финальных индексов присваиваются `0/0 finalLeft/finalRight` соответственно.

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.



АТД Application

Определяется структурный тип данных **`Application`**, содержащий пять полей: `Vector valueArray`, `int constD`, `int finalLeft`, `int finalRight`, `int finalConsequenceStreak`.

Первое — вектор, собираемый из введенной пользователем последовательности. Второе — константа `D`, так же, получаемая через стандартное устройство ввода. Три последних элемента, начинающихся с слова `final`, хранят результирующие данные, соответственно своим названиям. Финальный левый и правый индекс, а так же, длину конечной, совпавшей, последовательности. Как Вы успели заметить, я написал код на плюсах в camelCase, да. На следующем курсе все будет в snake_case.

application.h

В заголовочном файле `application.h` объявляется глобальная функция `int appRun(Application& app).` Эта функция отвечает за исполнение работу программы, а соответственно, напрямую взаимодействует с АТД A*pplication*, а именно: получает данные от пользователя, обрабатывает их, и выводит результат в стандартное устройство вывода. При успешном выполнении возвращает значение 0, если произошла ошибка на одном из этапов, возвращает значение 1.

Для выполнения поставленных, условием, задач, в `.h` файле объявляется прототип функции, а в `.cpp` файле, определяются под-функции `appRun`, а именно:

```
bool appInitializeData(Application &app);
bool appGetConstantD(Application &app);
bool appProcessDataIntoFinalResult(Application &app);
bool appGetOutputToUser(Application &app);
```

Рассмотрим каждую функцию в отдельности:

`bool appInitializeData(Application &app)`- отвечает за сборку вектора исходных данных, полученных от пользователя.

`bool appGetConstantD(Application &app)`- Получение числа D.

`bool appProcessDataIntoFinalResult(Application &app)`- Содержит основную логическую функцию обработки.

`bool appGetOutputToUser(Application &app)`- выводит конечный результат на стандартное устройство вывода.

АТД Vector

В этом модуле определяется структурный тип данных для представления n-мерного массива введенных пользователем данных, состоит из одного поля представленного типом `vector`, из стандартной библиотеки с элементами типа `int` (целочисленные).

В заголовочном файле `vector.h` объявляются функции:

```
Vector vectorValueArrayInitialize(const Vector& valueArray);
int vectorGetSize(Vector& v);
```

Рассмотрим эти функции в отдельности:

`Vector vectorValueArrayInitialize(const Vector& valueArray)`- прототип функции, собирающей вектор изначальных значений.

`int vectorGetSize(Vector& v)`- шорткат для получения размера вектора, с помощью функции .size().

Приложение 1.

```
//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.valueAr-
ray);
    std::cout << "Input array has been successfully pro-
cessed." << 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.valueAr-
ray.value[i] > app.valueArray.value[i + 1])
            ||
        (i == vectorGetSize(app.valueArray)-1)
        ){

            if ((tempConsequenceStreak > app.finalConse-
quenceStreak &&
            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.final-
Right == INT_MAX) &&
                    (i == tempConsequenceStreak || i == temp-
ConsequenceStreak+1)){
                    app.finalLeft = i - tempConsequenceS-
treak;
                }
                else{
                    app.finalLeft = i - (tempConsequenceS-
```

```
treak-1);
                }
                app.finalRight = i;
                app.finalConsequenceStreak = tempConse-
quenceStreak;
            }

            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.val-
ue.push_back(temporary_value_storage);

            counter++;

            if (std::cin.eof()) {
                break;
            }
        }
    }

    //Demonstrate input data
    std::cout << "Provided vector consists of " << vectorGet-
Size(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 "C:\Users\l3t\Documents\GitHub\NNTU_INF\googletest\googlemock\include\
gmock\gmock.h"
//#include "C:\Users\l3t\Documents\GitHub\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 +
```

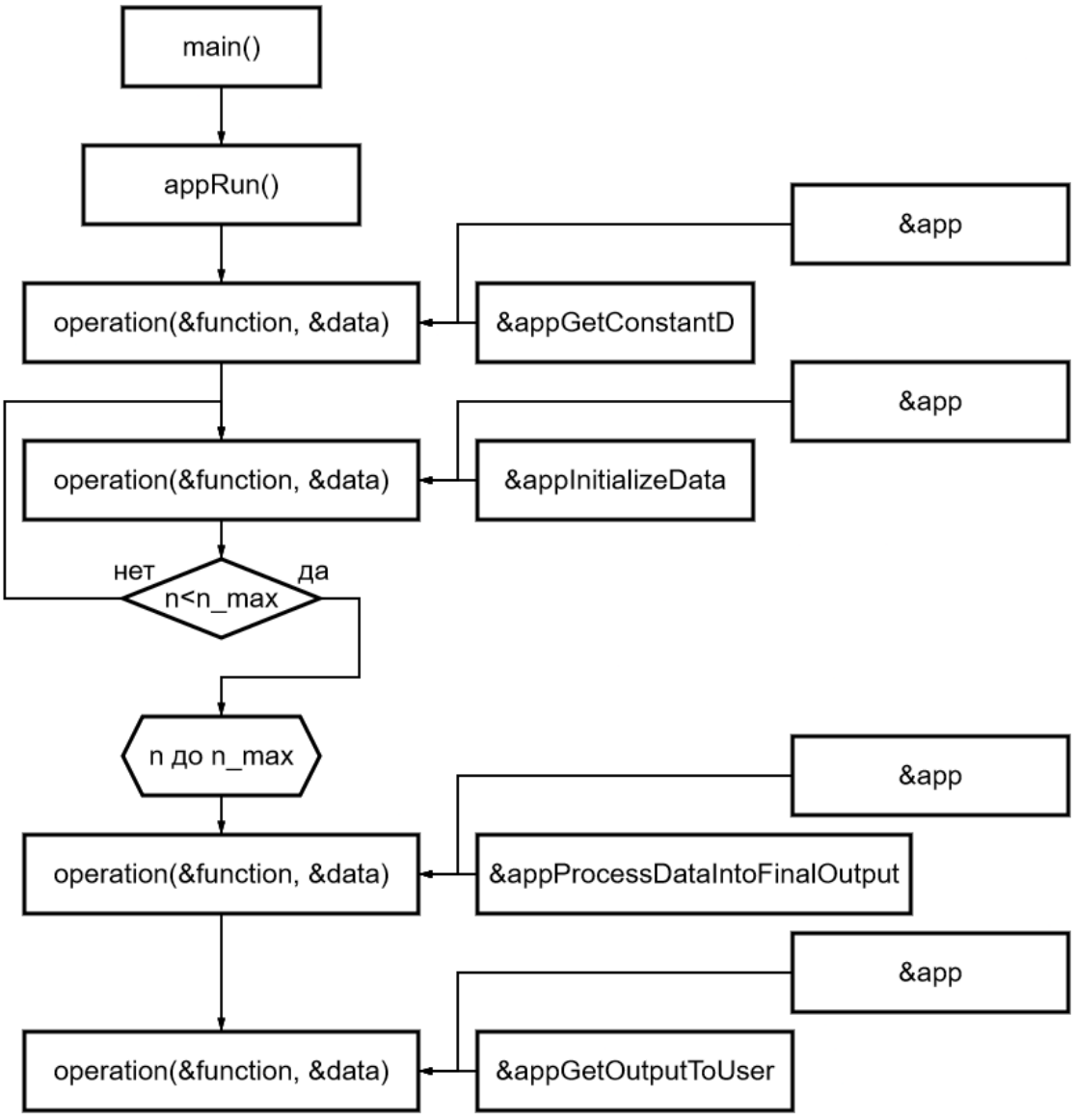
```
12, INT_MIN + 25}, 0);
    ASSERT_TRUE(appProcessDataIntoFinalResult(test_app));

    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
```

Контрольная работа №2
«Настройка индуктивного вычислителя с использованием функции обратного вызова»

2.1 Архитектура программной системы

Основная логика вычисления не претерпела изменений, с контрольной работы 1. Изменился стиль вызова и способ обмена данными между функциями. Теперь мы объявляем АТД Application в основной функции `appRun`, вместо main.

В следствие внедрения данной методологии вызова функций, изменилась структура программы. Представлена ниже.



2.2 Использование индуктивного вычислителя

Функция operation, которая соответствует назначению — callback из условия работы. Она принимает ссылку на функцию и ссылку на объявленный тип данных. В результате мы избегаем копирования одного и того же типа данных, по несколько раз.

В следствие использования данной структуры, можно наладить общение нескольких программ между друг другом, вследствие унифицированного элемента функции и набора данных.

```
//application.h

typedef bool (*Callback)(void *object);
bool operation(Callback callback, void *data);
```

Элемент	Назначение
`typedef`	аллиас на функцию
`bool`	тип возвращаемого значения
`(*Callback)`	указатель на тип функции
`(void *object)`	указатель на тип принимаемого аргумента, и его название

```
// application.cpp
bool operation(Callback callback, void *data) {
    return (*callback)(data);
}
```

Элемент	Назначение
`bool`	тип возвращаемого значения
`operataion`	название
`(Callback callback`	тип и имя принимаемого аргумента
`void *data)`	указатель на любой тип данных, имя аргумента
`return`	возвращаемое значение
`(*callback)`	результат работы вызываемой функции
`(data)`	данные с которыми работала программа.

Приложение 1

```
//main.cpp
//
// Created by Anatejl on 15.04.2024.
//

#include "application.h"
#include <iostream>

int main() {

    std::cout << std::endl << "An unarguably valuable piece of software, THE diamond!" <<
std::endl;

    int ret = appRun();

    return ret;
}

//application.h
//
// Created by Anatejl on 15.04.2024.
//

#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <climits>
#include "vector.h"

struct Application {

    Vector valueArray = {};

    int constD = 0;
    int finalLeft = INT_MAX;
    int finalRight = INT_MAX;
    int finalConsequenceStreak = 0;
    int tempConsequenceStreak = 0;
    int processIterator = 0;
};

typedef bool (*Callback)(void *object);
bool operation(Callback callback, void *data);
```

```
int appRun();
bool appInitializeData(void *app);
bool appGetConstantD(void *app);
bool appProcessDataIntoFinalResult(void *app);
bool appGetOutputToUser(void *app);

#endif //NNTU_APPLICATION_H


//application.cpp
//
// Created by Anatejl on 20.04.2024.
//

#include "application.h"
#include <iostream>

bool operation(Callback callback, void *data) {
    return (*callback)(data);
}

bool appGetConstantD(void *app) {

    Application &tempApp = *(Application *) app;

    std::cout << "Input a D constant to compare:" << std::endl;
    std::cin >> tempApp.constD;
    std::cout << tempApp.constD << std::endl;

    if(std::cin.fail()) {
        return false;
    }

    return true;
}

bool appInitializeData(void *app) {

    Application &tempApp = *(Application *) app;

    tempApp.valueArray = vectorValueArrayInitialize(tempApp.valueArray);

    return true;
}
```

```
bool appProcessDataIntoFinalResult(void *app) {

    Application &tempApp = *(Application *) app;

    //DENOTE FOR GTEST ONLY
    //for (int i = 0; i < tempApp.valueArray.value.size(); ++i) {

    if ((tempApp.processIterator != vectorGetSize(tempApp.valueArray) - 1 &&
        tempApp.valueArray.value[tempApp.processIterator] >
tempApp.valueArray.value[tempApp.processIterator + 1])
        ||
        (tempApp.processIterator == vectorGetSize(tempApp.valueArray) - 1)){

            if (tempApp.tempConsequenceStreak > tempApp.finalConsequenceStreak &&
                (tempApp.valueArray.value[tempApp.processIterator] -
                 tempApp.valueArray.value[tempApp.processIterator -
(tempApp.tempConsequenceStreak - 1)] >
                 tempApp.constD ||
                 tempApp.valueArray.value[tempApp.processIterator] -
                 tempApp.valueArray.value[tempApp.processIterator -
tempApp.tempConsequenceStreak] > tempApp.constD)) {

                if ((tempApp.finalLeft == INT_MAX && tempApp.finalRight == INT_MAX)
&&
                    (tempApp.processIterator == tempApp.tempConsequenceStreak ||
                     tempApp.processIterator == tempApp.tempConsequenceStreak + 1)) {
                    tempApp.finalLeft = tempApp.processIterator -
tempApp.tempConsequenceStreak;
                }
                else {
                    tempApp.finalLeft = tempApp.processIterator -
(tempApp.tempConsequenceStreak - 1);
                }

                tempApp.finalRight = tempApp.processIterator;
                tempApp.finalConsequenceStreak = tempApp.tempConsequenceStreak;
            }

        tempApp.tempConsequenceStreak = 0;
    }


    if (tempApp.finalLeft == INT_MAX && tempApp.finalRight == INT_MAX &&
vectorGetSize(tempApp.valueArray) == 1) {
        tempApp.finalLeft = 0;
        tempApp.finalRight = 0;
```

```
    }

    ++tempApp.tempConsequenceStreak;
    ++tempApp.processIterator;

    //DENOTE FOR GTEST ONLY
    //if (i == tempApp.valueArray.value.size() - 1 && tempApp.finalLeft == INT_MAX &&
tempApp.finalRight == INT_MAX) {
    //    return false;
    //}
    //}

    return true;
}

bool appGetOutputToUser(void *app) {

    Application &tempApp = *(Application *) app;

    std::cout << std::endl << tempApp.finalLeft << " - Left Index" << std::endl;
    std::cout << tempApp.finalRight << " - Right Index" << std::endl;

    return true;
}

int appRun() {

    Application app;

    // 1 - Get D const
    if (!operation(&appGetConstantD, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }

    // 2 - Read through cin
    while (!std::cin.eof()) {

        if (!operation(&appInitializeData, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
    }

    // 3 - Process data
    for (int i : app.valueArray.value) {
```

```
        operation(&appProcessDataIntoFinalResult, &app);

        if (i == app.valueArray.value.size() - 1 && app.finalLeft == INT_MAX &&
app.finalRight == INT_MAX) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
    }

    // 4 - Display result
    operation(&appGetOutputToUser, &app);

    return 0;
}


//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;

    while (!std::cin.eof()) {
        int temporary_value_storage = 0;

        std::cin >> temporary_value_storage;
        temporaryVector.value.push_back(temporary_value_storage);

        if (std::cin.eof()) {
            break;
        }
    }

    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 к коду из первого приложение.

```
//test.cpp
//
// Created by Anatejl on 30.04.2024.
//
#ifndef CALLBACK_V1_APPLICATION_TESTS_CPP
#define CALLBACK_V1_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 "C:\Users\l3t\Documents\GitHub\NNTU_INF\googletest\googlemock\include\
gmock\gmock.h"
#include "C:\Users\l3t\Documents\GitHub\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));

//ASSERT_TRUE(appProcessDataIntoFinalResult(test_app));
//ASSERT_TRUE(test_app.finalLeft == 0 || test_app.finalLeft == 4 || test_app.finalLeft ==
INT_MAX);
//ASSERT_TRUE(test_app.finalRight == 0 || test_app.finalRight == 4 || test_app.finalRight
== INT_MAX);
}

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 huy = -1;
    int huy2 = 1;
    //ASSERT_FALSE(huy);
    ASSERT_TRUE(huy2);
}

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 //CALLBACK_APPLICATION_TESTS_CPP
```