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 18)**

Выполнил:

Студент гр. 22-Рз \_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

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

Нижний Новгород,

2024 г.

Оглавление

[Задание по варианту 3](#__RefHeading___Toc2465_4070117755)

[Контрольная работа № 1. 4](#__RefHeading___Toc2035_2758904751)

[1.1. Постановка задачи 4](#__RefHeading___Toc2041_2758904751)

[1.2. Алгоритм индуктивной обработки 4](#__RefHeading___Toc11489_2758904751)

[1.3. Архитектура программной реализации вычислителя. 6](#__RefHeading___Toc5564_2519614503)

[application.h 7](#__RefHeading___Toc11493_2758904751)

[Приложение 1. 9](#__RefHeading___Toc3283_4070117755)

[Контрольная работа №2 11](#__RefHeading___Toc2047_2758904751)

[2.1 Архитектура программной системы 11](#__RefHeading___Toc11495_2758904751)

[2.2 Использование индуктивного вычислителя 12](#__RefHeading___Toc11497_2758904751)

[Приложение 1 14](#__RefHeading___Toc11499_2758904751)

# Задание по варианту

Подсчет количества «равнинных участков». Равнинным участком считается группа соседних элементов (более K элементов) исходной последовательности с одинаковым значением. 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.

На вход системы последовательно и неограниченно во времени поступают элементы, где — порядковый номер элемента, начиная с 1. Реализовать указанную в варианте обработку последовательности элементов с использованием схемы индуктивной обработки на пространстве последовательностей. Полученный набор выходных значений рассматривается как результирующая последовательность. Значения элементов исходной последовательности должны запрашиваться у пользователя (приниматься со стандартного устройства ввода) по одному за раз. Сформированные выходные значения требуется выдавать сразу после их формирования на стандартное устройство вывода. Реализация обработки должна быть приведена в отдельном модуле.

## 1.2. Алгоритм индуктивной обработки

Пересчет выдаваемого результата осуществляется функцией appProcessData, имея очередное значение , необходимо соблюсти следующие условия, чтобы значение успешно прошло учет в текущее плато:

Примем следующие сокращения:

– текущее значение;

– порядковый номер;

– кол-во значений текущего стрика;

– значение текущего стрика;

– K из условия;

– предыдущее, максимальное значение стрика;

– выдаваемое пользователю, кол-во значений подходящего стрика;

– выдаваемое пользователю, значение подходящего стрика.

Указав используемые переменные, перейдем к рассмотрению базовых условий:

;

;

;

;

На основе описанных, базовых, условий, можно сформировать предикаты:

- справедливо для ;

Результат: ;

- справедливо для ;

Результат: ;

- справедливо для ;

Результат: ;

- справедливо для ;

Результат: ;

В общем случае вид формулы, по которой вычисляется новое значение величины, определяется набором условий .

Приведенные условия являются предикатами, т.е. результатом каждого из условий является логическое значение истина/ложь. Точкой обозначен набор аргументов, необходимых для вычисления значения предиката.

Во время выполнения программы, значение счетчика изменяется, для соответствия поставленным задачам, что является индуктивным расширением. Очередной элемент

Условия, используемые в правиле пересчета и охватывают все возможные ситуации, то есть . Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчета величины.

Кроме того, выполняется условие , то есть не допускается одновременное выполнение различных условий. И обратное утверждение, . Истинное значение предиката обозначает факт наступления связанного с ним события.

## 1.3. Архитектура программной реализации вычислителя.

Разберём используемый абстрактный тип данных.

В файле application.h определяется два АТД, рассмотрим их в порядке, представленном в коде программы.

**1) АТД temp\_data — два поля:**

*// 1-n, 2-x*

std::pair <int, int> cin\_read\_current;  
 *// 1-streak 2-x* std::pair <int, int> temp\_counter;

Первое поле — текущий элемент, первое значение — счетчик n, второе — x, текущее значение, считанное из потока.

Второе поле — текущее плато, первое значение — счетчик кол-ва элементов плато, второе значение — само значение плато.

**2) АТД Application — четыре поля:**

temp\_data temp\_data;  
 *//k* int constK;  
 *//lp* int last\_plato;  
 *// 1-streak, 2-x* std::pair <int, int> final;

Первое поле — включение АТД temp\_data в основной АТД Application.

Второе поле — считанное, из стандартного устройства ввода, число K.

Третье поле — значение кол-ва элементов предыдущего плато.

Четвертое — пара целочисленых, первое — кол-во элементов, максимального на текущий момент, плато. Второе — элемент текущего плато.

application.h

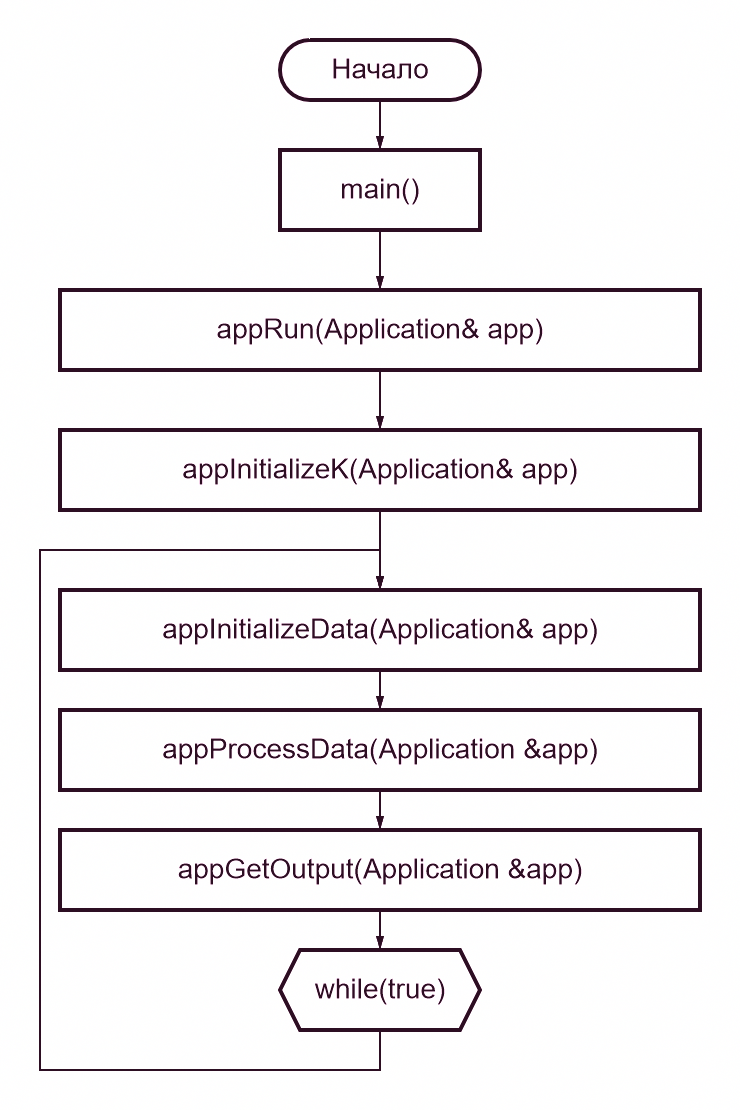
В заголовочном файле application.h объявляется функция int appRun(Application& app). Упомянутая функция отвечает за правильное исполнение приложения. Принимает данные с стандартного устройства ввода, обрабатывает, согласно приведённым в пункте 1.2 правилам, и выводит соответствующий результат на стандартное устройство вывода.

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

bool appInitializeK(Application &app);  
 bool appInitializeData(Application &app);  
 bool appProcessData(Application &app);  
 bool appGetOutput(Application &app);

Рассмотрим каждую функцию в отдельности:

bool appInitializeK(Application &app); - Получение числа ;  
 bool appInitializeData(Application &app); - Получение очередного ;  
 bool appProcessData(Application &app); - Обработка , запись результата плато на шаге ;  
 bool appGetOutput(Application &app); - выводит промежуточный результат на стандартное устройство вывода. Для упрощения восприятия, представим программу в качестве блок-схемы.



# Приложение 1.

Main.cpp

#include "application.h"  
#include <iostream>  
  
int main() {  
 Application app;  
 int ret = appRun(app);  
 return ret;  
}

Application.h

*/*#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <utility>struct temp\_data {  
 *// 1-n, 2-x* std::pair <int, int> cin\_read\_current;  
 *// 1-streak 2-x* std::pair <int, int> temp\_counter;  
};  
  
*//Data for program to handle*struct Application {  
 temp\_data temp\_data;  
 *//k* int constK;  
 *//lp* int last\_plato;  
 *// 1-streak, 2-x* std::pair <int, int> final;  
};  
  
int appRun(Application &app);  
bool appInitializeK(Application &app);  
bool appInitializeData(Application &app);  
bool appProcessData(Application &app);  
bool appGetOutput(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

Application.cpp

#include "application.h"  
#include <iostream>  
  
int appRun(Application &app) {  
 if (!appInitializeK(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 while (!std::cin.eof()) {  
 *//Get Value from cin* if (!appInitializeData(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!appProcessData(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!appGetOutput(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 }  
 return 0;  
}  
  
bool appInitializeK(Application &app) {  
 std::cin >> app.constK;  
 *//assign controlled value to i counter* app.temp\_data.cin\_read\_current.first = INT\_MAX;  
 if (std::cin.fail()) {  
 return false;  
 }  
 return true;  
}  
  
bool appInitializeData(Application &app) {  
 std::cin >> app.temp\_data.cin\_read\_current.second;  
 *//assign 0 or ++ to counter, based on i* if (app.temp\_data.cin\_read\_current.first == INT\_MAX) {  
 app.temp\_data.cin\_read\_current.first = 0;  
 } else {  
 ++app.temp\_data.cin\_read\_current.first;  
 }  
 return true;  
}  
  
bool appProcessData(Application &app) {  
 *//check for first iteration, do special things for it* if (app.temp\_data.cin\_read\_current.first == 0) {  
 ++app.temp\_data.temp\_counter.first;  
 app.temp\_data.temp\_counter.second = app.temp\_data.cin\_read\_current.second;  
 return true;  
 }  
 *//check for x\_n == x\_n-1* if (app.temp\_data.cin\_read\_current.second == app.temp\_data.temp\_counter.second) {  
 ++app.temp\_data.temp\_counter.first;  
 } else {  
 app.temp\_data.temp\_counter.first = 1;  
 app.temp\_data.temp\_counter.second = app.temp\_data.cin\_read\_current.second;  
 app.last\_plato = app.final.first;  
 }  
 *//check for current plato counter > K,  
 // if so then assign current values to final to display* if (app.temp\_data.temp\_counter.first > app.last\_plato && app.temp\_data.temp\_counter.first > app.constK) {  
 app.final.first = app.temp\_data.temp\_counter.first;  
 app.final.second = app.temp\_data.temp\_counter.second;  
 }  
 return true;  
}  
  
bool appGetOutput(Application &app) {  
 std::cout << app.temp\_data.cin\_read\_current.first << " - Iteration: ";  
 std::cout << app.final.first << " of " << app.final.second << std::endl;  
 return true;  
}

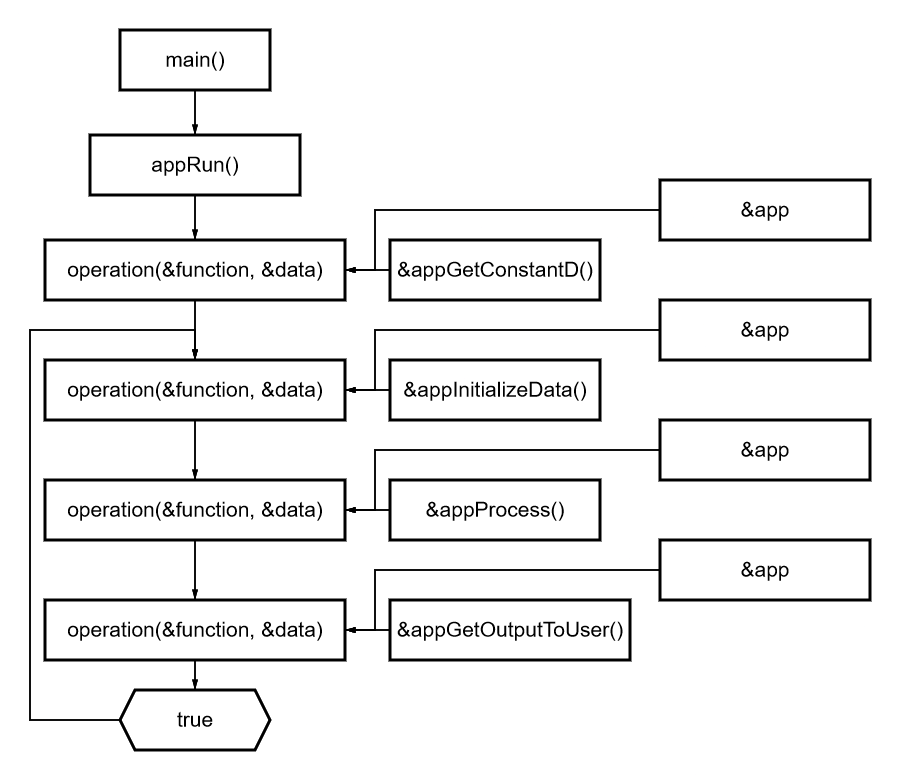
# Контрольная работа №2

**«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

Запрошенная callback конструкция, может быть представлена в виде функции operation, которая будет брать два аргумента, ссылку на bool функцию, которую необходимо исполнить, и ссылку на АТД, т. е. Данные, которые operation передаст, как аргумент, для вызываемой функции. Опишем её:

В application.h:

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

typedef — псевдоним типа данных

bool — тип возвращаемого значения

(\*Callback) — указатель типа функции

(void \*data) — указатель типа аргумента, название

В application.cpp:

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

bool — тип возвращаемого значения

operation — название

Callback - тип первого аргумента

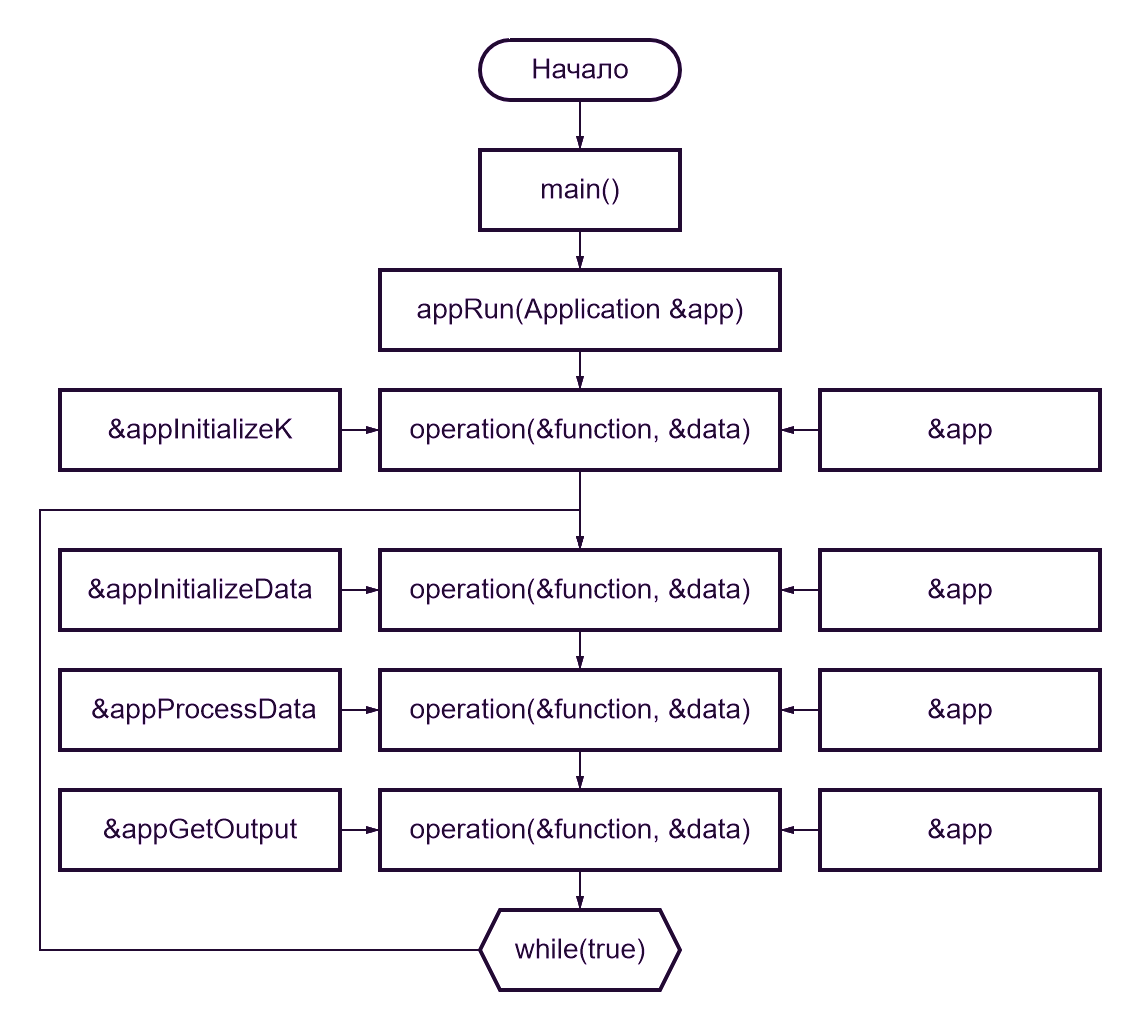
сallback — имя первого аргумента

void \*data — указатель тип второго аргумента — любой

return — возвращаемое значение

(\*callback) – результат работы вызванной функции

(data) – данные, которые эта функция использовала.

В следствие изменения стиля вызова функций, структура программы, притерпела небольшие изменения, рассмотрим обновлённую блок-схему:

# Приложение 1

Main.cpp

#include "application.h"  
#include <iostream>  
  
int main() {  
  
 int ret = appRun();  
  
 return ret;  
}

application.h

#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <utility>  
struct temp\_data {  
  
 *// 1-n, 2-value* std::pair <int, int> cin\_read\_current;  
 *// 1-streak* std::pair <int, int> temp\_counter;  
  
};struct Application {  
  
 temp\_data temp\_data;  
 int constK;  
 int last\_plato;  
  
 *// 1-streak, 2-value* std::pair <int, int> final;  
  
};  
typedef bool (\*Callback)(void \*object);  
bool operation(Callback callback, void \*data);  
int appRun();  
bool appInitializeK(void \*p\_app);  
bool appInitializeData(void \*p\_app);  
bool appProcessData(void \*p\_app);  
bool appGetOutput(void \*p\_app);  
  
#endif *//NNTU\_APPLICATION\_H*

application.cpp

#include "application.h"  
#include <iostream>  
  
bool operation(Callback callback, void \*data) {  
 return (\*callback)(data);  
}  
  
bool appInitializeK(void \*p\_app) {  
 Application &app = \*(Application \*) p\_app;  
 std::cin >> app.constK;  
 *//assign controlled value to i counter* app.temp\_data.cin\_read\_current.first = INT\_MAX;  
 if (std::cin.fail()) {  
 return false;  
 }  
 return true;  
}  
  
bool appInitializeData(void \*p\_app) {  
 Application &app = \*(Application \*) p\_app;  
 std::cin >> app.temp\_data.cin\_read\_current.second;  
 *//assign 0 or ++ to counter, based on i* if (app.temp\_data.cin\_read\_current.first == INT\_MAX) {  
 app.temp\_data.cin\_read\_current.first = 0;  
 } else {  
 ++app.temp\_data.cin\_read\_current.first;  
 }  
  
 return true;  
}  
  
bool appProcessData(void \*p\_app) {  
 Application &app = \*(Application \*) p\_app;  
 *//check for first iteration, do special things for it* if (app.temp\_data.cin\_read\_current.first == 0) {  
 ++app.temp\_data.temp\_counter.first;  
 app.temp\_data.temp\_counter.second = app.temp\_data.cin\_read\_current.second;  
 return true;  
 }  
 *//check for x\_n == x\_n-1* if (app.temp\_data.cin\_read\_current.second == app.temp\_data.temp\_counter.second) {  
 ++app.temp\_data.temp\_counter.first;  
 } else {  
 app.temp\_data.temp\_counter.first = 1;  
 app.temp\_data.temp\_counter.second = app.temp\_data.cin\_read\_current.second;  
 app.last\_plato = app.final.first;  
 }  
 *//check for current plato counter > K,  
 // if so then assign current values to final to display* if (app.temp\_data.temp\_counter.first > app.last\_plato && app.temp\_data.temp\_counter.first > app.constK) {  
 app.final.first = app.temp\_data.temp\_counter.first;  
 app.final.second = app.temp\_data.temp\_counter.second;  
 }  
 return true;  
}  
  
bool appGetOutput(void \*p\_app) {  
 Application &app = \*(Application \*) p\_app;  
 std::cout << app.temp\_data.cin\_read\_current.first << " - Iteration: ";  
 std::cout << app.final.first << " of " << app.final.second << std::endl;  
 return true;  
}  
  
int appRun() {  
 Application app;  
 if (!operation(&appInitializeK, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 while (!std::cin.eof()) {  
 if (!operation(&appInitializeData, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!operation(&appProcessData, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!operation(&appGetOutput, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 }  
 return 0;  
}