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 43)**

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)

[АТД Application 6](#__RefHeading___Toc11491_2758904751)

[application.h 6](#__RefHeading___Toc11493_2758904751)

[Приложение 1. 9](#__RefHeading___Toc3283_4070117755)

[Контрольная работа №2 12](#__RefHeading___Toc2047_2758904751)

[2.1 Архитектура программной системы 12](#__RefHeading___Toc11495_2758904751)

[2.2 Использование индуктивного вычислителя 13](#__RefHeading___Toc11497_2758904751)

[Приложение 1 14](#__RefHeading___Toc11499_2758904751)

# Задание по варианту

Поиск максимального элемента, меньшего заданного значения, и подсчёт числа его повторений в последовательности. Тип элемента — действительный.

# 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

Принятые сокращения:

– порядковый номер элемента;

– очередной элемент;

– граница, по условию задачи;

– кол-во повторений максимального элемента;

– максимальный элемент, ниже границы.

- элемент выходной последовательности, включает в себя три значения ()

Отклик вычислителя Отклик вычислителя запишем как: .

Рассмотрим базовые условия:

;

;

;

;

Имея базовые условия, мы можем сформировать предикаты:

- справедливо для ;

Результат: ;

- справедливо для

Результат: ;

- справедливо для

Результат: ;

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

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

В следствии обработки входящих значений, предусмотренный счётчик , будет изменяться, что, соответственно, будет являться индуктивным расширением.

Кроме того, выполняется условие , то есть не допускается одновременное выполнение различных условий. Соответственно, модем составить обратное утверждение, .

Так же, можем сделать заключение о существовании очередного значения .

## 1.3. Архитектура программной реализации вычислителя.

Для обеспечения удобства взаимодействия с необходимыми переменными, нами вводится абстрактный тип данных (далее АТД), рассмотрим его:

АТД Application

Определяется структурный тип данных **Application**, содержащий четыре поля:

int cin\_read;  
 std::pair<int, int> max;  
 int threshold;  
 int iteration = 1;

1) Текущее значение - ;

2) Пара финальных значений;

1) Кол-во повторений максимального значения,

2) Максимальный элемент, ниже границы,

3) Граница, ниже которой ищем элементы;

4) Счётчик итерации, начинается с 1, т. к. мы засчитываем итерацию в конце цикла.

application.h

В заголовочном файле application.h объявляется глобальная функция int app\_run(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 , если произошла ошибка на одном из этапов, возвращает значение .

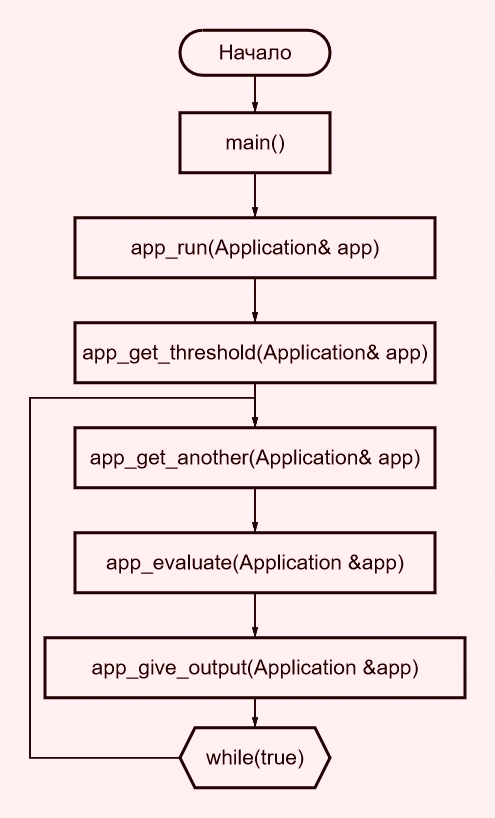
Для выполнения поставленных, условием, задач, в .h файле объявляется прототип функции, а в .cpp файле, определяются под-функции app\_run, а именно:

bool app\_get\_threshold(Application &app);  
 bool app\_get\_another(Application &app);  
 bool app\_evaluate(Application &app);  
 bool app\_give\_output(Application &app);

Рассмотрим каждую функцию в отдельности:

bool app\_get\_threshold(Application &app); - получение числа — границы, по условию;  
 bool app\_get\_another(Application &app); - получение очередного значения ;  
 bool app\_evaluate(Application &app); - выполнение проверки значения ;  
 bool app\_give\_output(Application &app); - вывод результата итерации на стандартное устройство.

Можем представить общую структуру программы в виде блок-схемы:



# Приложение 1.

//main.cpp

#include "application.h"  
  
int main(){  
 Application app;  
 int ret = app\_run(app);  
 return ret;  
}

//application.h

#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <utility>  
  
struct Application {  
 int cin\_read;  
 std::pair<int, int> max;  
 int threshold;  
 int iteration = 1;  
};  
  
int app\_run(Application &app);  
bool app\_get\_threshold(Application &app);  
bool app\_get\_another(Application &app);  
bool app\_evaluate(Application &app);  
bool app\_give\_output(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

//application.cpp

#include "application.h"  
#include <iostream>  
  
int app\_run(Application &app) {  
 if (!app\_get\_threshold(app)) {  
 std::cout << "INPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 *//use !std::cin.eof() for debugging!* while (true) {  
 if (!app\_get\_another(app)) {  
 std::cout << "SEQ. INPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 if (!app\_evaluate(app)) {  
 std::cout << "PROCESSING FAILURE, ABORT ABORT" << std::endl;  
 }  
 if (!app\_give\_output(app)) {  
 std::cout << "OUTPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 }  
 return 0;  
}  
  
bool app\_get\_threshold(Application &app) {  
 std::cout << "Input an int, to act as threshold:" << std::endl;  
 std::cin >> app.threshold;  
 if (std::cin.fail()) {  
 return false;  
 }  
 std::cout << app.threshold << std::endl;  
 return true;  
}  
  
bool app\_get\_another(Application &app) {  
 std::cin >> app.cin\_read;  
 if (std::cin.fail()) {  
 return false;  
 }  
 return true;  
}  
  
bool app\_evaluate(Application &app) {  
 if (app.iteration == 0) {  
 app.max.first = 1;  
 app.max.second = app.cin\_read;  
 return true;  
 }  
 if (app.cin\_read == app.max.second) {  
 ++app.max.first;  
 return true;  
 } else if (app.cin\_read > app.max.second && app.cin\_read < app.threshold) {  
 app.max.first = 1;  
 app.max.second = app.cin\_read;  
 }  
  
 return true;  
}  
  
bool app\_give\_output(Application &app) {  
 std::cout << app.iteration << " - " <<  
 app.max.second << " : " << app.max.first << " times." << std::endl;  
 ++app.iteration;  
 return true;  
}

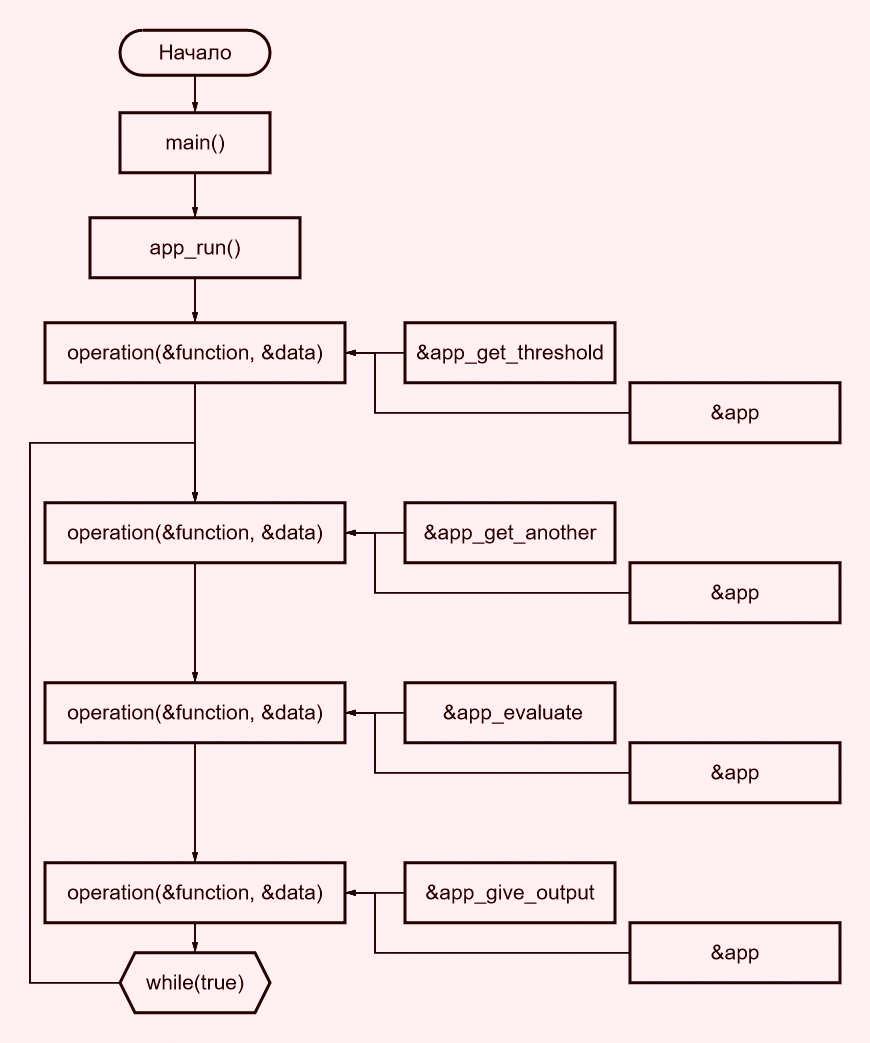
# Контрольная работа №2

**«Настройка индуктивного вычислителя**

**с использованием функции обратного вызова»**

## 2.1 Архитектура программной системы

Основная логика вычисления не претерпела изменений, с контрольной работы 1. Изменился способ вызова и обмена данными между функциями.

В следствие чего, 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 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.

# Приложение 1

*//main.cpp*

#include "application.h"  
  
int main(){  
 Application app;  
 int ret = app\_run(app);  
 return ret;  
}

*//application.h*

#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <utility>  
  
typedef bool (\*Callback)(void \*object);  
bool operation(Callback callback, void \*data);  
  
struct Application {  
 int cin\_read;  
 std::pair<int, int> max;  
 int threshold;  
 int iteration = 1;  
};  
  
int app\_run(Application &app);  
bool app\_get\_threshold(void \*raw\_app);  
bool app\_get\_another(void \*raw\_app);  
bool app\_evaluate(void \*raw\_app);  
bool app\_give\_output(void \*raw\_app);  
  
#endif *//NNTU\_APPLICATION\_H*

*//application.cpp*

#include "application.h"  
#include <iostream>  
  
bool operation(Callback callback, void \*data) {  
 return (\*callback)(data);  
}  
  
int app\_run(Application &app) {  
 if (!operation(&app\_get\_threshold, &app)) {  
 std::cout << "INPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 *//use !std::cin.eof() for debugging!* while (true) {  
 if (!operation(&app\_get\_another, &app)) {  
 std::cout << "SEQ. INPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 if (!operation(&app\_evaluate, &app)) {  
 std::cout << "PROCESSING FAILURE, ABORT ABORT" << std::endl;  
 }  
 if (!operation(&app\_give\_output, &app)) {  
 std::cout << "OUTPUT FAILURE, ABORT ABORT" << std::endl;  
 }  
 }  
 return 0;  
}  
  
bool app\_get\_threshold(void \*raw\_app) {  
 Application &app = \*(Application\*) raw\_app;  
 std::cout << "Input an int, to act as threshold:" << std::endl;  
 std::cin >> app.threshold;  
 if (std::cin.fail()) {  
 return false;  
 }  
 std::cout << app.threshold << std::endl;  
 return true;  
}  
  
bool app\_get\_another(void \*raw\_app) {  
 Application &app = \*(Application\*) raw\_app;  
 std::cin >> app.cin\_read;  
 if (std::cin.fail()) {  
 return false;  
 }  
 return true;  
}  
  
bool app\_evaluate(void \*raw\_app) {  
 Application &app = \*(Application\*) raw\_app;  
 if (app.iteration == 0) {  
 app.max.first = 1;  
 app.max.second = app.cin\_read;  
 return true;  
 }  
 if (app.cin\_read == app.max.second) {  
 ++app.max.first;  
 return true;  
 } else if (app.cin\_read > app.max.second && app.cin\_read < app.threshold) {  
 app.max.first = 1;  
 app.max.second = app.cin\_read;  
 }  
 return true;  
}  
  
bool app\_give\_output(void \*raw\_app) {  
 Application &app = \*(Application\*) raw\_app;  
 std::cout << app.iteration << " - " <<  
 app.max.second << " : " << app.max.first << " times." << std::endl;  
 ++app.iteration;  
 return true;  
}