

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**

<b>Задание по варианту.....</b>	<b>3</b>
<b>Контрольная работа № 1.....</b>	<b>4</b>
<b>1.1. Постановка задачи.....</b>	<b>4</b>
<b>1.2. Алгоритм индуктивной обработки.....</b>	<b>4</b>
<b>1.3. Архитектура программной реализации вычислителя.</b>	
<b>.....</b>	<b>6</b>
<b>АТД Application.....</b>	<b>6</b>
<b>application.h.....</b>	<b>6</b>
<b>Приложение 1.....</b>	<b>9</b>
<b>Контрольная работа №2.....</b>	<b>12</b>
<b>2.1 Архитектура программной системы.....</b>	<b>12</b>
<b>2.2 Использование индуктивного вычислителя.....</b>	<b>13</b>
<b>Приложение 1.....</b>	<b>14</b>

## **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  $x_i$ , где  $i$  — порядковый номер элемента, начиная с 1. Реализовать указанную в варианте обработку  $f(X)$  последовательности элементов  $X = \langle x_1, x_2, x_3, \dots \rangle$  с использованием схемы индуктивной обработки на пространстве последовательностей. Полученный набор выходных значений  $Y = \langle y_1, y_2, y_3, \dots \rangle$  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

Обработка, и пречест осуществляется в функции `app_evaluate`, рассмотрим применяемые условия:

$$R(): \neg r_1 \cap \neg r_2 \cap r_3 \cap r_4$$

Принятые сокращения:

$n$  — порядковый номер элемента;

$x_n$  — элемент с индексом  $n$ ;

$th$  — граница, по условию задачи;

$i_{max}$  — кол-во повторений максимального элемента;

$x_{max}$  – максимальный элемент, ниже границы.

Рассмотрим базовые условия:

$$r_1(n=0);$$

$$r_2(x_n = x_{max});$$

$$r_3(x_n > x_{max});$$

$$r_4(x_n < th);$$

Имея базовые условия, мы можем сформировать предикаты:

$$R_1(): r_1 - \text{справедливо для } n=0;$$

$$\text{Результат: } R_1(.) \rightarrow i_{max}=1; x_{max}=x_n;$$

$$R_2(): \neg r_1 \cap r_2 - \text{справедливо для } n \neq 0 \wedge x_n = x_{max}$$

$$\text{Результат: } R_2(.) \rightarrow ++i_{max};$$

$$R_3(): \neg r_1 \cap \neg r_2 \cap r_3 \cap r_4 - \text{справедливо для } n \neq 0 \wedge x_n \neq x_{max} \wedge x_n > x_{max} \wedge x_n < th$$

$$\text{Результат: } R_3(.) \rightarrow i_{max}=1; x_{max}=x_n;$$

В общем случае вид формулы, по которой вычисляется новое значение величины  $x_{max}$ , определяется набором условий  $R=(R_3(.))$ . А значение  $i_{max}$ , вычисляется следующим набором условий  $R=(R_2(.))$ .

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  $x_n$  значений, предусмотренный счётчик *iteration*, будет изменяться, что, соответственно, будет являться индуктивным расширением.

Условия, используемые в правиле пересчёта  $x_n$  охватывают все возможные ситуации, то есть  $R_1 \cup R_2 \cup R_3 = true$ . Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчёта величины.

Кроме того, выполняется условие  $\forall x_n: R_3 \rightarrow R_1 \cap R_2 \cap R_3 = false$ , то есть не допускается одновременное выполнение различных условий. Соответственно, можем составить обратное утверждение,  $\neg \forall x_n: R_3 \rightarrow R_1 \cap R_2 \cap R_3 = true$ .

Так же, можем сделать заключение о существовании очередного значения  $x_n$   $\exists x_n: R_3 \rightarrow x_n \in [0, th]$ .

### 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 -  $x_n$ ;
- 2) Пара финальных значений;
  - 1) Кол-во повторений максимального значения,  $i_{max}$
  - 2) Максимальный элемент, ниже границы,  $x_{max}$
- 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 *Application*, 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 `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:

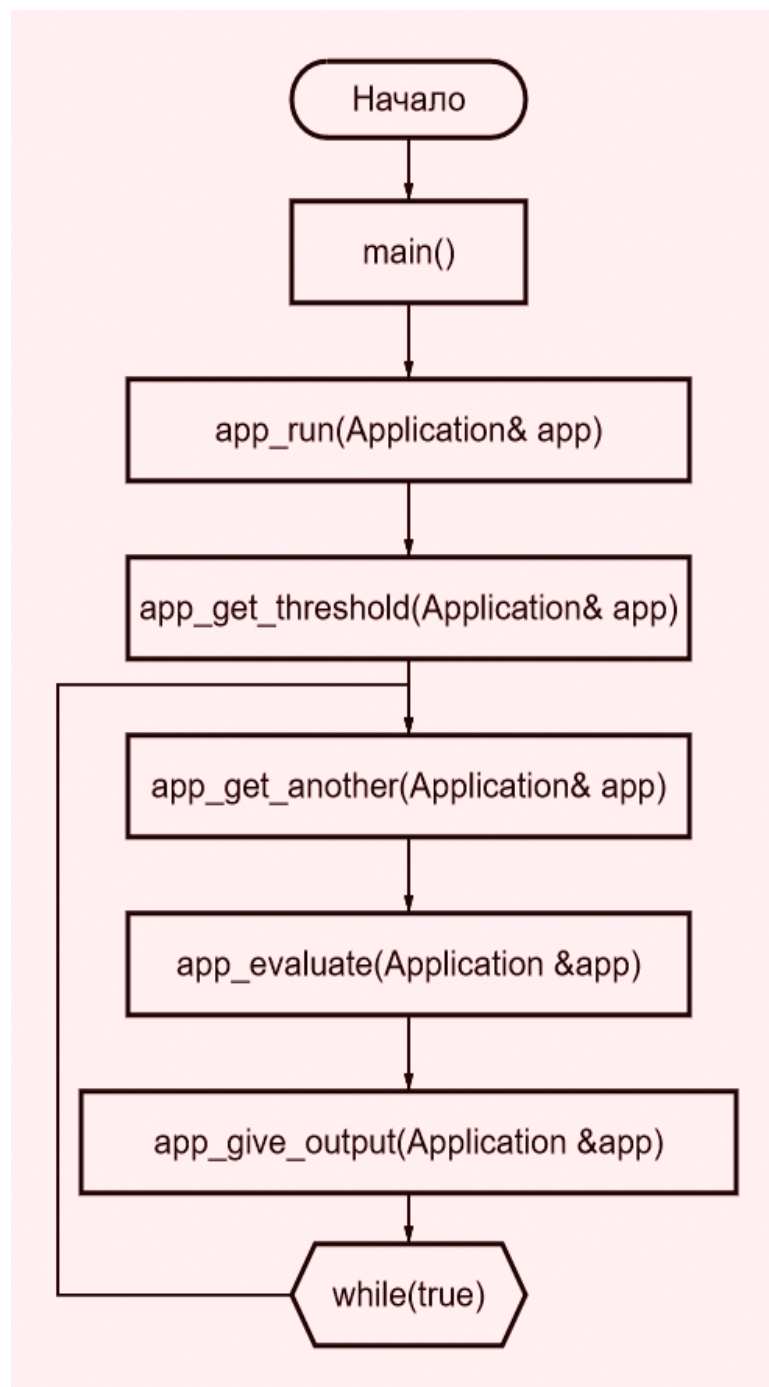
`bool app_get_threshold(Application &app);` - получение числа — границы, по условию;

`bool app_get_another(Application &app);` - получение очередного значения  $x_n$ ;

`bool app_evaluate(Application &app);` - выполнение проверки значения  $x_n$ ;

`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;
}

```

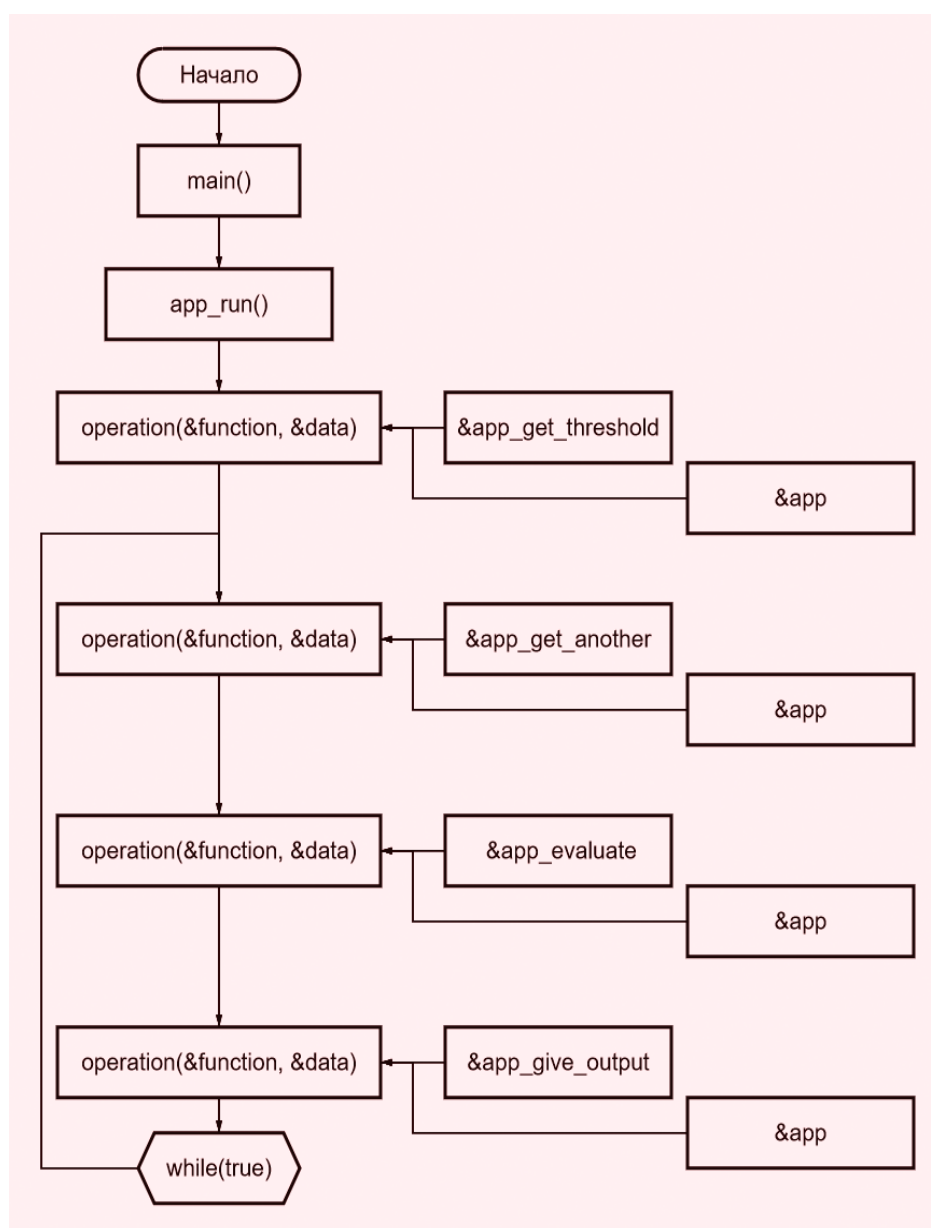
## Контрольная работа №2

### «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

callback — 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;
}

```