

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 13)**

Выполнил:  
Студент гр. 22-Рз      Кузнецов Д.С.

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>Заголовочный файл 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

Исключение элементов, значения которых находятся вне интервала  $[a, b]$ , границы которого изменяются по заданному линейному закону  $a(n) = v_a \cdot n + a_0$ ,  $b(n) = v_b \cdot n + b_0$ , где  $n$  — номер элемента в исходной последовательности. 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  $x_i$ , 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

Используемые переменные и принятые сокращения:

$x_n$  — очередное значение;

$n$  — индекс текущего элемента;

$a$  — левая граница, вычисляется по формуле  $a(n) = v_a \cdot n + a_0$

$v_a$  — множитель левой границы, вводится пользователем;

$a_0$  — нулевое значение  $a$ , вводится пользователем;

$b$  — левая граница, вычисляется по формуле  $b(n) = v_b \cdot n + b_0$

$v_b$  — множитель левой границы, вводится пользователем;

$b_0$  — нулевое значение  $b$ , вводится пользователем;

$inBounds$  — логическое значение, используется для записи вердикта, принадлежит ли  $x_n$  участку  $[a, b]$ ;

$y_1$  – первый вариант элемента выходной последовательности, состоит из двух элементов

$(n, x_n)$ ;

$y_2$  – второй вариант элемента выходной последовательности, состоит из одного элемента  $(n)$ ;

В начале выполнения программы, значения  $a, b$  неизвестны, они вычисляются в ходе работы, и напрямую используются, в приведённых ниже логических условиях, без записи в отдельную переменную. В свою очередь  $x_n$  получается от пользователя в ходе работы программы. Отклик вычислителя запишем как:  $q = \begin{cases} \langle y_1 \rangle & , inBounds = true \\ \langle y_2 \rangle & , inBounds = false \end{cases}$ .

Переменные границы  $[a, b]$  вычисляются по формулам  $a(n) = v_a \cdot n + a_0$ ,  $b(n) = v_b \cdot n + b_0$ , соответственно. Значения  $v_a, a_0, v_b, b_0$  получаются от пользователя на первой итерации программы, при  $n=0$ .

Рассмотрим базовое условие:

$r_2(x_n > a \wedge x_n < b)$ ;

Зная эти условия, можно сформировать предикаты, отметим, что если переменная не упоминается в результате предиката, то её значение не изменяется.

$R_1(): r_1$  - если  $(x_n > a \wedge x_n < b)$ ;

Результат:  $R_1(): \rightarrow inBounds = true, q = \langle y_1 \rangle, n = n + 1, a = a(n + 1), b = b(n + 1)$ ;

$R_2(): \neg r_1$  - если  $(x_n < a \wedge x_n < b) \vee (x_n > a \wedge x_n > b)$

Результат:  $R_2(): \rightarrow inBounds = false, q = \langle y_2 \rangle, n = n + 1, a = a(n + 1), b = b(n + 1)$ ;

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

В ходе выполнения программы, значение счётчика  $n$ , претерпевает изменения, в результате индуктивной операции над  $x_n$ , соответственно работа счётчика будет считаться индуктивным расширением.

Кроме того, выполняется условие  $\forall x_n \rightarrow (R_1 \vee R_2) = true$ , т.е. Для любого поступившего значения  $x_n$  выполнится предикат  $R_1$  или  $R_2$ , соответственно, одновременное выполнение не допускается.

### 1.3. Архитектура программной реализации вычислителя.

В ходе разработки программы, было принято решение, использовать абстрактный тип данных, для хранения всех используемых переменных. В предложенной программе, он представлен как тип с именем *Application*.

#### АТД Application

Определяется структурный тип данных **Application**, содержащий шесть полей:

```
int iteration = 1;
int cin_read;
bool in_bounds = false;
int va = INT_MAX;
int a0 = INT_MAX;
int vb = INT_MAX;
int b0 = INT_MAX;
```

Выбранные имена переменных говорят сами за себя, но тем не менее, разберём из предназначение. 1 — счётчик итерации программы, для учёта индекса  $n$ . 2 — очередное значение  $x_n$ , 3 — логическая переменная, обозначающая вхождение  $x_n$  в  $[a, b]$ , или нет. Последние четыре значения — константы, вводимые пользователем, по условию задачи.

#### Заголовочный файл application.h

В нем объявляется глобальная функция `appRun`, возвращающая целочисленное значение. Эта функция отвечает за исполнение программы, 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 appGetData(Application &app);  
int appGetA(Application &app);  
int appGetB(Application &app);  
bool appProcess(Application &app);  
bool appDoOutput(Application &app);
```

Рассмотрим каждую функцию в отдельности:

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

`int appGetA(Application &app);` - считывание  $v_a$  и  $a_0$ , если вызвана в первый раз.

Или просчет границы  $a$ , для значения  $x_n$ .

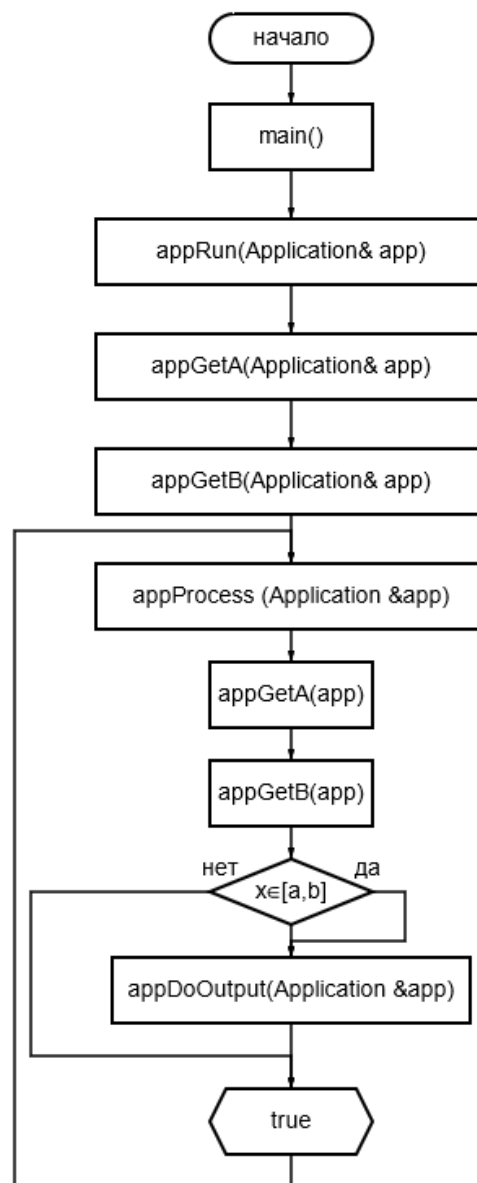
`int appGetB(Application &app);` - считывание  $v_b$  и  $b_0$ , если вызвана в первый раз.

Или просчет границы  $b$ , для значения  $x_n$ .

`bool appProcess(Application &app);` - вердикт, входит ли, очередное значение  $x_n$  в границы  $[a, b]$ .

`bool appDoOutput(Application &app)` — если входит, то вывод  $x_n$  на стандартное устройство вывода;

Можем представить общую структуру программы в виде 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 <climits>
```

```
struct Application {  
    int iteration = 1;  
    int cin_read;  
    bool in_bounds = false;  
    int va = INT_MAX;  
    int a0 = INT_MAX;  
    int vb = INT_MAX;  
    int b0 = INT_MAX;  
};
```

```
int appRun(Application& app);  
bool appGetData(Application &app);  
int appGetA(Application &app);  
int appGetB(Application &app);  
bool appProcess(Application &app);  
bool appDoOutput(Application &app);
```

```
#endif //NNTU_APPLICATION_H
```

```
//application.cpp
```

```
#include "application.h"
```

```
#include <iostream>
```

```
int appRun(Application &app) {  
    if (!appGetA(app)) {  
        std::cout << "DATA INPUT FAILURE." << std::endl;  
        return 1;  
    }  
}
```

```

if (!appGetB(app)) {
    std::cout << "DATA INPUT FAILURE." << std::endl;
    return 1;
}
//use !std::cin.eof() for testing
while(true) {
    if (!appGetData(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    appProcess(app);
    appDoOutput(app);
    ++app.iteration;
}
return 0;
}

int appGetA(Application &app) {
    if (app.va == INT_MAX && app.a0 == INT_MAX) {
        std::cin >> app.va >> app.a0;
        return true;
    }
    else{
        int calculated = app.va * app.iteration + app.a0;
        return calculated;
    }
}

int appGetB(Application &app) {
    if (app.vb == INT_MAX && app.b0 == INT_MAX) {
        std::cin >> app.vb >> app.b0;
        return true;
    }
    else{
        int calculated = app.vb * app.iteration + app.b0;
        return calculated;
    }
}

bool appGetData(Application &app) {
    std::cin >> app.cin_read;
    if(std::cin.fail()){
        return false;
    }
    return true;
}

bool appProcess(Application &app) {

```

```

    if(app.cin_read > appGetA( app) && app.cin_read < appGetB(app)){
        app.in_bounds = true;
    }
    return false;
}

bool appDoOutput(Application &app) {
    if(app.in_bounds){
        std::cout << app.iteration << " - " << app.cin_read << std::endl;
        //DEBUG
        std::cout << "[" << appGetA(app) << " ÷ " << appGetB(app) << "]" << std::endl <<
std::endl;
    }
    else{
        std::cout << app.iteration << " - provided value is out of bounds. " << std::endl;
    }
    app.in_bounds = false;
    return true;
}

```

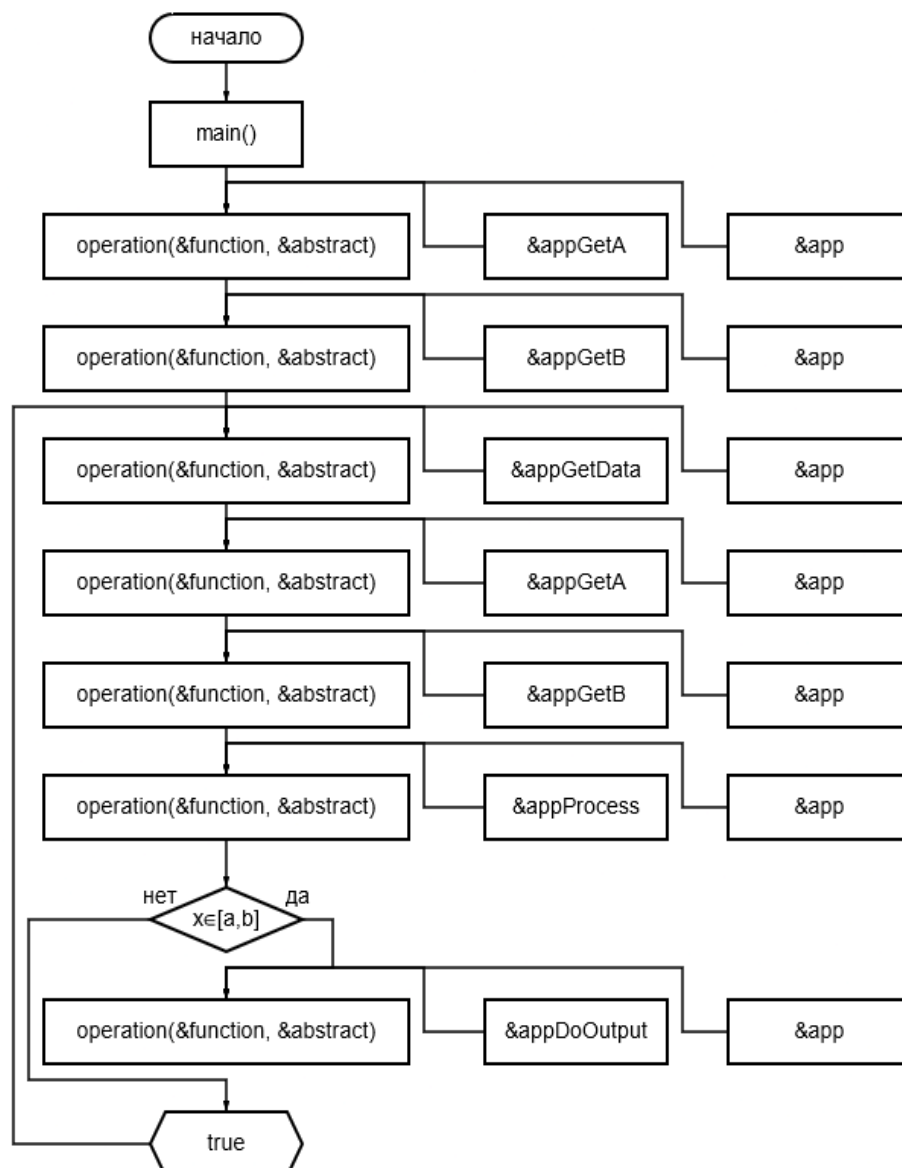
## Контрольная работа №2

### «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 АД Application в основной функции appRun, вместо main. Но теперь, так как, функция, вызванная через callback должна возвращать bool значение, мы не можем в качестве возвращаемого значения функций appGetA и appGetB, указывать результат вычислений. Его нужно передавать через переменную в АД. Что и было сделано. В Application, добавлены два целочисленных поля — current\_A, и current\_B.

Обновленная блок-схема представлена ниже.



## 2.2 Использование индуктивного вычислителя

По условию, вводится функция `operation()`, которая принимает два аргумента - ссылку на исполняемую функцию и ссылку на переменную АД. Рассмотрим структуру функции `operation`.

В `application.h`:

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

`typedef` — псевдоним типа данных

`bool` — тип возвращаемого значения данных

`(*Callback)` — указатель имени функции

`(void *abstract)` — указатель типа аргумента, название

В `application.cpp`:

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

`bool` — тип возвращаемого значения

`operation` — имя

`Callback` - тип первого аргумента

`callback` — имя первого аргумента

`void *abstract` — указатель тип второго аргумента — любой

`return` — возвращаемое значение

`(*callback)` – результат работы вызванной функции

`(abstract)` – данные, которые эта функция использовала.

# 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 <climits>

//Data for program to handle
struct Application {
    int iteration = 1;
    int cin_read;
    bool in_bounds = false;

    /*Since we can't invoke function for it return its value as int,
    now we store A/B as a distinct variables, rather than gaining
    A/B reading by straight invoking its corresponding function*/
    int current_A, current_B;

    int va = INT_MAX;
    int a0 = INT_MAX;
    int vb = INT_MAX;
    int b0 = INT_MAX;
};

typedef bool (*Callback)(void *abstract);
bool operation(Callback callback, void *abstract);

// To execute application

int appRun();
bool appGetData(void *abstract);
bool appGetA(void *abstract);
bool appGetB(void *abstract);
bool appProcess(void *abstract);
bool appDoOutput(void *abstract);
```

```
#endif //NNTU_APPLICATION_H
```

## APPLICATION.CPP

```
#include "application.h"
```

```
#include <iostream>
```

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

```
bool appGetA(void *abstract) {  
    Application &app = *(Application*) abstract;  
    if (app.va == INT_MAX && app.a0 == INT_MAX) {  
        std::cin >> app.va >> app.a0;  
    }  
    else {  
        app.current_A = app.va * app.iteration + app.a0;  
    }  
    return true;  
}
```

```
bool appGetB(void *abstract) {  
    Application &app = *(Application*) abstract;  
    if (app.vb == INT_MAX && app.b0 == INT_MAX) {  
        std::cin >> app.vb >> app.b0;  
    }  
    else {  
        app.current_B = app.vb * app.iteration + app.b0;  
    }  
    return true;  
}
```

```
bool appGetData(void *abstract) {  
    Application &app = *(Application*) abstract;  
    std::cin >> app.cin_read;  
    if(std::cin.fail()){  
        return false;  
    }  
    return true;  
}
```

```
bool appProcess(void *abstract) {  
    Application &app = *(Application*) abstract;  
    if(app.cin_read > app.current_A && app.cin_read < app.current_B){  
        app.in_bounds = true;  
    }  
    return false;  
}
```

```

}

bool appDoOutput(void *abstract) {
    Application &app = *(Application*) abstract;
    if(app.in_bounds){
        std::cout << app.iteration << " - " << app.cin_read << std::endl;
        //DEBUG
        std::cout << "[" << app.current_A << " ÷ "
            << app.current_B << "]" << std::endl << std::endl;
    }
    else{
        std::cout << app.iteration << " - provided value is out of bounds. " << std::endl;
    }
    app.in_bounds = false;
    return true;
}

int appRun() {
    Application app;
    if (!operation(&appGetA, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    if (!operation(&appGetB, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    //use !std::cin.eof() for testing
    while(true) {
        if (!operation(&appGetData, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appGetA, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appGetB, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        operation(&appProcess, &app);
        operation(&appDoOutput, &app);
        ++app.iteration;
    }
    return 0;
}

```