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

[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)

[Заголовочный файл 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

Исключение элементов, значения которых находятся вне интервала , границы которого изменяются по заданному линейному закону , , где — 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, 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

Используемые переменные и принятые сокращения:

– очередное значение;

— индекс текущего элемента;

– левая граница, вычисляется по формуле

– множитель левой границы, вводится пользователем;

– нулевое значение a, вводится пользователем;

– левая граница, вычисляется по формуле

– множитель левой границы, вводится пользователем;

– нулевое значение b, вводится пользователем;

– логическое значение, используется для записи вердикта, принадлежит ли участку ;

– первый вариант элемента выходной последовательности, состоит из двух элементов

();

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

В начале выполнения программы, значения неизвестны, они вычисляются в ходе работы, и напрямую используются, в приведённых ниже логических условиях, без записи в отдельную переменную. В свою очередь получается от пользователя в ходе работы программы. Отклик вычислителя запишем как: .

Переменные границы [a, b] вычисляются по формулам , *,* соответственно. Значения получаются от пользователя на первой итерации программы, при .

Рассмотрим базовое условие:

;

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

- если ;

Результат: ;

- если

Результат: ;

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 , претерпевает изменения, в результате индуктивной операции над , соответственно работа счётчика будет считаться индуктивным расширением.

Кроме того, выполняется условие , т.е. Для любого поступившего значения выполнится предикат или , 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 предложеной программе, он представлен как тип с именем .

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 . 2 – очередное значение , 3 — логическая переменная, обозначающая вхождение в , 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 , если произошла ошибка на одном из этапов, возвращает значение .

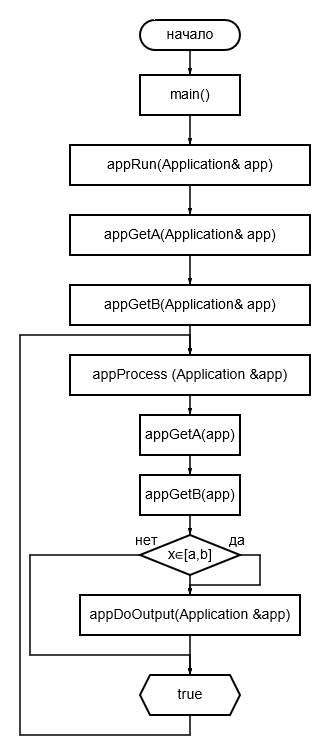
Для выполнения поставленных, условием, задач, в .h файле объявляется прототип функции, а в .cpp файле, определяются под-функции appRun, а именно:

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 ;  
 int appGetA(Application &app); - считывание и , если вызвана в первый раз. Или просчет границы , для значения .  
 int appGetB(Application &app); - считывание и , если вызвана в первый раз. Или просчет границы , для значения .  
 bool appProcess(Application &app); - вердикт, входит ли, очередное значение в границы .  
 bool appDoOutput(Application &app) — если входит, то вывод 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;  
}

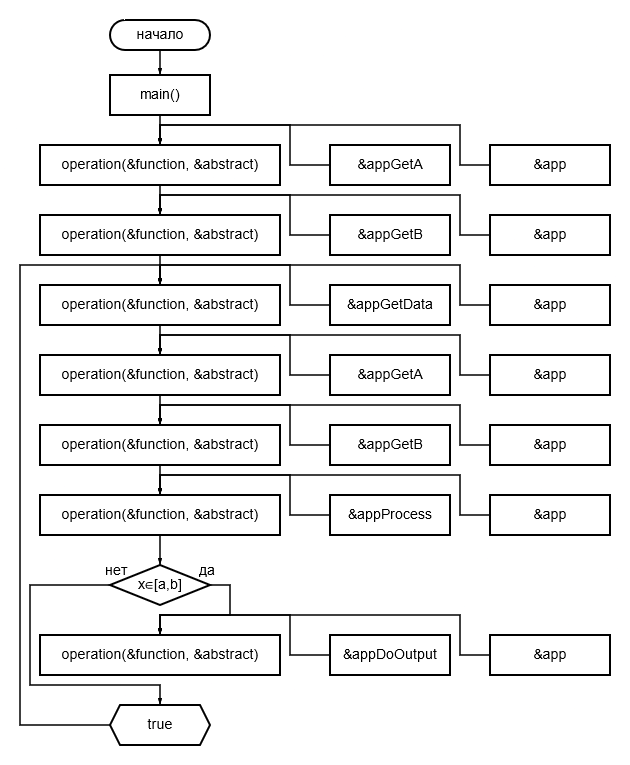
# Контрольная работа №2

**«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 АТД 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 АТД. 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 АТД. 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

сallback — 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;  
}