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 41)**

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)

[algo.h 7](#__RefHeading___Toc5558_2519614503)

[Приложение 1. 9](#__RefHeading___Toc3283_4070117755)

[Контрольная работа №2 15](#__RefHeading___Toc2047_2758904751)

[2.1 Архитектура программной системы 15](#__RefHeading___Toc11495_2758904751)

[2.2 Использование индуктивного вычислителя 16](#__RefHeading___Toc11497_2758904751)

[Приложение 1 17](#__RefHeading___Toc11499_2758904751)

# Задание по варианту

На вход поступают элементы последовательности в виде точек на декартовой плоскости. Для каждого элемента рассматривается группа не менее чем соседних элементов, включающая его самого и предшествующих ему. Для каждого из элементов этой группы должно быть выполнено условие — точка находится внутри области, ограниченной заданной окружностью. Определить факт наличия такой группы в текущий момент времени и ее размер (количество элементов).

# 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

Алгоритм программы заключается в обработке групп чисел, соответственно, логически, он разбит на несколько функций. По условию, мы обрабатываем точки в декартовой системе координат, следовательно мы принимаем два значения за раз. Договоримся, что - очередная пара точек . Таким образом, для пересчета очередного поступившего значения, должно наступить событие:

Будем использовать следующие сокращения:

|  |  |
| --- | --- |
|  | Радиус заданной окружности |
|  | Радиус от центра окружности, до указанной точки |
|  | Итератор от 0 до |

Можем получить следующие базовые условия:

;

;

Имея набор базовых условий, можно сформировать предикаты,

- справедливо для ;

Результат: ;

Обратный предикат , справедлив в случае , тогда результат будет:

;

Предикат 3:

- справедливо для в таком случае, результат:

;

Предикат 4:

- справедливо для ;

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

Для выполнения поставленной задачи, в ходе программы, счетчик будет проходить значения от до , что в свою очередь будет являться индуктивным расширением.

Используемые условия, 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.

Для хранения переменных, используемых в программе, мы используем структуру АТД.

АТД Application

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

std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>> temp\_group;  
std::pair <int, int>init\_xy;

std::pair<int,int> circle\_center;  
std::pair<int,int> circle\_edge;  
int circle\_r;  
int const\_k;  
int iteration = 1;

Первое — вектор, по размеру , тип данных которого пара: и пара и пара int int. Можем рассмотреть данный тип данных с обратной стороны. На низшем уровне — это пара для рассматриваемой точки. На уровень выше, в первое значение пары становится радиус от центра окружности до точки. Поднимаясь еще на уровень выше, в первое значение – изначально . После обработки, он может быть изменен на .

Второе — пара изначальных значений .

Третье — пара x/y для центра окружности.

Четвертое — точка на окружности, для определения радиуса референсной окружности.

Пятое — вычисленный радиус такой окружности.

Шестое — введенное число k.

Седьмое — счетчик итерации.

application.h

В файле объявляется основная функция рассматриваемого приложения appRun(), которая отвечает за вызов других под-функций, соответственно за обработку поступающих значений .

В качестве функций, в .h файле объявляется прототип функции, а в .cpp файле приводится определение.

bool appGetConstantK(Application &app);  
bool appGetCircleDimensions(Application &app);  
bool appInitializeData(Application &app);  
bool appProcessCurrentXYRadius(Application &app);  
bool appProcessResult(Application &app);  
bool appGetOutputToUser(Application &app);

Рассмотрим каждую функцию в отдельности:

bool appGetConstantK(Application &app); - Получение числа K;

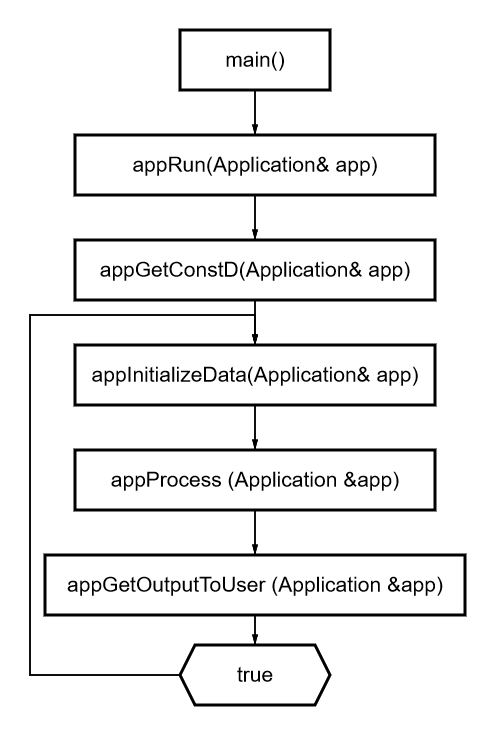
bool appGetCircleDimensions(Application &app); - Получение center и edge координат, референсной окружности;

bool appInitializeData(Application &app); - Считывание очередной пары координат;

bool appProcessCurrentXYRadius(Application &app); - единичная функция сборки вектора, по размеру k;

bool appProcessResult(Application &app); - проверка вхождения точек группы в референсную окружность;

bool appGetOutputToUser(Application &app); - выдача результата пользователю.

Можем представить общую структуру программы в виде блок-схемы:

# Приложение 1.

*//main.cpp  
//  
// Created by Anatejl on 15.04.2024.  
//*#include "application.h"  
#include <iostream>  
  
int main() {  
  
 std::cout << "An unarguably valuable piece of software, THE diamond!" << std::endl;  
  
 Application app;  
  
 int ret = appRun(app);  
  
 return ret;  
}

*//application.h  
//  
// Created by Anatejl on 15.04.2024.  
//*#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <climits>  
#include <utility>  
#include "algo.h"  
  
*//Data for program to handle.*struct Application {  
  
 int const\_D;  
  
 *//Assume 1- iteration 2 - element* std::pair<int, int> current\_element;  
 std::pair<int, int> last\_element;  
  
 *//Assume 1 - index 2 - value* std::pair<int, int> finalLeft;  
 std::pair<int, int> finalRight;  
 int finalCS = 0;  
 int tempCS = 0;  
};  
  
*// To execute application*int appRun(Application& app);  
bool appInitializeData(Application &app);  
bool appGetConstantD(Application &app);  
bool appProcess(Application &app);  
bool appGetOutputToUser(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

*//application.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "application.h"  
#include "algo.h"  
#include <iostream>  
  
bool appGetConstantD(Application &app) {  
  
 std::cout << "Input a D constant to compare:" << std::endl;  
 std::cin >> app.const\_D;  
 std::cout << app.const\_D << std::endl;  
  
 return true;  
}  
  
bool appInitializeData(Application &app) {  
  
 std::cin >> app.current\_element.second;  
  
 return true;  
}  
  
bool appProcess(Application &app) {  
  
 while(true) {  
  
 if(algo\_check\_first\_iteration(&app)){  
 break;  
 }  
  
 if(algo\_check\_breakage(&app)){  
 break;  
 }  
  
 if(!algo\_check\_ascending(&app)){  
 break;  
 }  
  
 if(!algo\_check\_D(&app)){  
 break;  
 }  
  
 break;  
 }  
  
 algo\_update\_last(&app);  
  
 return true;  
}  
  
bool appGetOutputToUser(Application &app) {  
  
 std::cout << app.current\_element.first << " - Iteration" << std::endl;  
 std::cout << "L - " << app.finalLeft.first << std::endl;  
 std::cout << "R - " << app.finalRight.first << std::endl << std::endl;  
  
 return true;  
}  
  
int appRun(Application &app) {  
  
 if (!appGetConstantD(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 *//Default condition is "true", consider using "!std::con.eof()" for testing purposes.* while(true) {  
 if (!appInitializeData(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!appProcess(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl << "No matches applicable." << std::endl;  
 return 1;  
 }  
  
 if (!appGetOutputToUser(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 ++app.current\_element.first;  
 }  
  
 return 0;  
}

*//algo.h  
//  
// Created by Anatejl on 20.04.2024.  
//*#ifndef NNTU\_ALGO\_H  
#define NNTU\_ALGO\_H#include "application.h"  
  
bool algo\_check\_first\_iteration(void \*object);  
bool algo\_check\_ascending(void \*object);  
bool algo\_check\_D(void \*object);  
bool algo\_check\_breakage(void \*object);  
void algo\_update\_last(void \*object);  
  
#endif *//NNTU\_ALGO\_H*

*//algo.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "algo.h"  
  
void algo\_update\_last(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 app.last\_element.first = app.current\_element.first;  
 app.last\_element.second = app.current\_element.second;  
 ++app.tempCS;  
  
}  
  
bool algo\_check\_first\_iteration(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.first == 0){  
 app.finalLeft.first = app.current\_element.first;  
 app.finalLeft.second = app.current\_element.second;  
 app.finalRight.first = app.current\_element.first;  
 app.finalRight.second = app.current\_element.second;  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_breakage(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.second <= app.last\_element.second){  
 app.tempCS = 0;  
  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_ascending(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.tempCS > app.finalCS) {  
  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_D(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.second - app.finalLeft.second > app.const\_D){  
 app.finalLeft.first = app.current\_element.first - app.tempCS;  
 app.finalRight.first = app.current\_element.first;  
 app.finalCS = app.tempCS;  
 return true;  
 }  
  
 return false;  
}

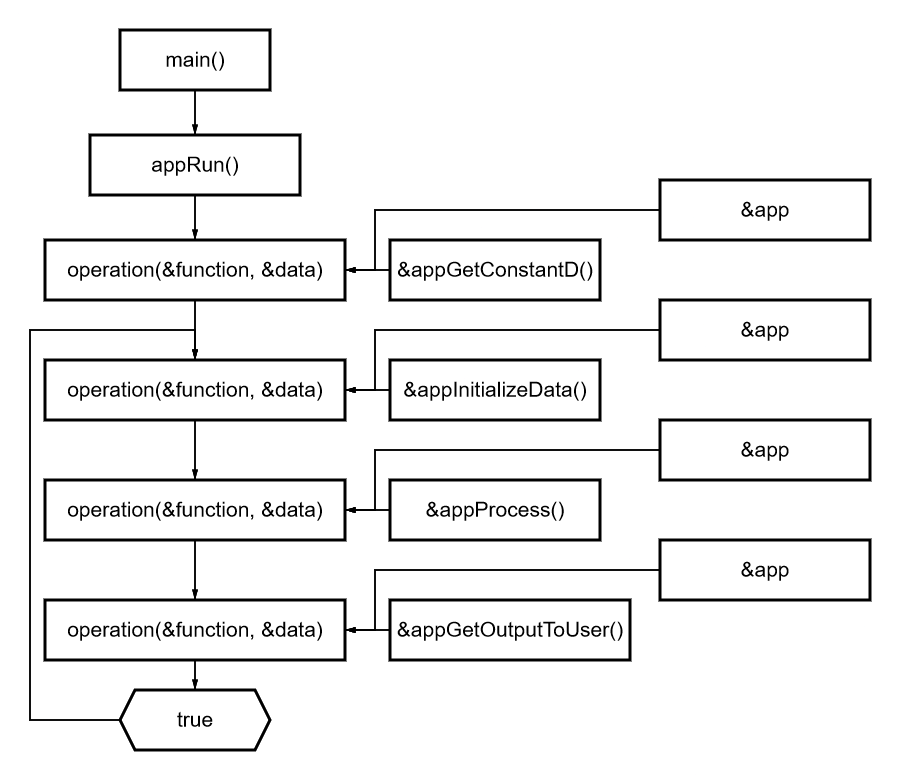
# Контрольная работа №2

**«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

Функция operation, которая соответствует назначению — callback из условия работы. Она принимает ссылку на функцию и ссылку на объявленный тип данных. В результате мы избегаем копирования одного и того же типа данных, по несколько раз.

В следствие использования данной структуры, можно наладить общение нескольких программ между друг другом, вследствие унифицированного элемента функции и набора данных.

//application.h

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

|  |  |
| --- | --- |
| Элемент | Назначение |
| typedef | аллиас на функцию |
| bool | тип функции |
| (\*Callback) | указатель на тип функции |
| (void \*object) | указатель на тип принимаемого аргумента, и его название |

// application.cpp

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

|  |  |
| --- | --- |
| Элемент | Назначение |
| bool | тип функции |
| operataion | название |
| (Callback callback | тип и имя принимаемого аргумента |
| void \*data) | указатель на любой тип данных, имя аргумента |
| return | возвращаемое значение |
| (\*callback) | результат работы вызываемой функции |
| (data) | данные с которыми работала программа. |

# Приложение 1

*//main.cpp  
//  
// Created by Anatejl on 15.04.2024.  
//*#include "application.h"  
#include <iostream>  
  
int main() {  
  
 std::cout << "An unarguably valuable piece of software, THE diamond!" << std::endl;  
  
 int ret = appRun();  
  
 return ret;  
}

*//application.h  
//  
// Created by Anatejl on 15.04.2024.  
//*#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <climits>  
#include <utility>  
#include "algo.h"  
  
*//Data for program to handle.*struct Application {  
  
 int const\_D;  
  
 *//Assume 1- iteration 2 - element* std::pair<int, int> current\_element;  
 std::pair<int, int> last\_element;  
  
 *//Assume 1 - index 2 - value* std::pair<int, int> finalLeft;  
 std::pair<int, int> finalRight;  
 int finalCS = 0;  
 int tempCS = 0;  
};  
  
typedef bool (\*Callback)(void \*object);  
bool operation(Callback callback, void \*data);  
  
*// To execute application*int appRun();  
bool appInitializeData(void \*object);  
bool appGetConstantD(void \*object);  
bool appProcess(void \*object);  
bool appGetOutputToUser(void \*object);  
  
#endif *//NNTU\_APPLICATION\_H*

*//application.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "application.h"  
#include "algo.h"  
#include <iostream>  
  
bool operation(Callback callback, void \*data) {  
 return (\*callback)(data);  
}  
  
bool appGetConstantD(void \*object) {  
  
 Application &app = \*((Application\*) object);  
  
 std::cout << "Input a D constant to compare:" << std::endl;  
 std::cin >> app.const\_D;  
 std::cout << app.const\_D << std::endl;  
  
 return true;  
}  
  
bool appInitializeData(void \*object) {  
  
 Application &app = \*((Application\*) object);  
  
 std::cin >> app.current\_element.second;  
  
 return true;  
}  
  
bool appProcess(void \*object) {  
  
 Application &app = \*((Application\*) object);  
  
 while(true) {  
  
 if(algo\_check\_first\_iteration(&app)){  
 break;  
 }  
  
 if(algo\_check\_breakage(&app)){  
 break;  
 }  
  
 if(!algo\_check\_ascending(&app)){  
 break;  
 }  
  
 if(!algo\_check\_D(&app)){  
 break;  
 }  
  
 break;  
 }  
  
 algo\_update\_last(&app);  
  
 return true;  
}  
  
bool appGetOutputToUser(void \*object) {  
  
 Application &app = \*((Application\*) object);  
  
 std::cout << app.current\_element.first << " - Iteration" << std::endl;  
 std::cout << "L - " << app.finalLeft.first << std::endl;  
 std::cout << "R - " << app.finalRight.first << std::endl << std::endl;  
  
 return true;  
}  
  
int appRun() {  
  
 Application app;  
  
 if (!operation(&appGetConstantD, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 *//Default condition is "true", consider using "!std::con.eof()" for testing purposes.* while(true) {  
 if (!operation(&appInitializeData, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!operation(&appProcess, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl << "No matches applicable." << std::endl;  
 return 1;  
 }  
  
 if (!operation(&appGetOutputToUser, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 ++app.current\_element.first;  
 }  
  
 return 0;  
}

*//algo.h  
//  
// Created by Anatejl on 20.04.2024.  
//*#ifndef NNTU\_ALGO\_H  
#define NNTU\_ALGO\_H#include "application.h"  
  
bool algo\_check\_first\_iteration(void \*object);  
bool algo\_check\_ascending(void \*object);  
bool algo\_check\_D(void \*object);  
bool algo\_check\_breakage(void \*object);  
void algo\_update\_last(void \*object);  
  
#endif *//NNTU\_ALGO\_H*

*//algo.cpp  
//  
// Created by Anatejl on 20.04.2024.  
//*#include "algo.h"  
  
void algo\_update\_last(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 app.last\_element.first = app.current\_element.first;  
 app.last\_element.second = app.current\_element.second;  
 ++app.tempCS;  
  
}  
  
bool algo\_check\_first\_iteration(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.first == 0){  
 app.finalLeft.first = app.current\_element.first;  
 app.finalLeft.second = app.current\_element.second;  
 app.finalRight.first = app.current\_element.first;  
 app.finalRight.second = app.current\_element.second;  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_breakage(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.second <= app.last\_element.second){  
 app.tempCS = 0;  
  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_ascending(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.tempCS > app.finalCS) {  
  
 return true;  
 }  
  
 return false;  
}  
  
bool algo\_check\_D(void \*object){  
  
 Application &app = \*((Application\*)object);  
  
 if(app.current\_element.second - app.finalLeft.second > app.const\_D){  
 app.finalLeft.first = app.current\_element.first - app.tempCS;  
 app.finalRight.first = app.current\_element.first;  
 app.finalCS = app.tempCS;  
 return true;  
 }  
  
 return false;  
}