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 Конев А.А.

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 7](#__RefHeading___Toc11493_2758904751)

[Приложение 1. 9](#__RefHeading___Toc3283_4070117755)

[Контрольная работа №2 14](#__RefHeading___Toc2047_2758904751)

[2.1 Архитектура программной системы 14](#__RefHeading___Toc11495_2758904751)

[2.2 Использование индуктивного вычислителя 15](#__RefHeading___Toc11497_2758904751)

[Приложение 1 16](#__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

По условию, мы обрабатываем точки в декартовой системе координат, следовательно мы принимаем два значения за раз. Договоримся, что - очередная пара точек .

Примем следующие сокращения:

|  |  |
| --- | --- |
|  | Радиус заданной окружности |
|  | Радиус от центра окружности, до указанной точки |
|  | Кол-во элементов в одной группе |
|  | Итератор от 0 до |
|  | Индекс текущего элемента |
|  | Очередное значение |
|  | Счётчик количества точек, попавших в окружность |
|  | Первый вариант элемента выходной последовательности, содержит три элемента () |
|  | Второй вариант элемента выходной последовательности, содержит три элемента () |

Обозначим отклик вычислителя как

По смыслу, программа разбита на две части, сначала рассмотрим алгоритм обработки поступающих значений . Отметим, что эта часть не является основным индуктивным вычислителем, поэтому в качестве результата, не приводится. Обозначим её дополнительной буквой в индексе — С. Если пересчёт переменной, явно, не приводится, то стоит считать, что её значение осталось неизменным.

Базовые условия алгоритма просчёта вхождения группы точек в окружность:

Предикаты алгоритма обработки точек:

Результат

,

Результат

Базовые условия основного вычислителя:

;

;

Предикаты основного вычислителя, на основе базовых условий:

,

Результат

,

Результат

Для выполнения поставленной задачи, в ходе программы, счётчик 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 пересчёта величины.

## 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:

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

3)std::pair<int,int> circle\_center;  
 4) std::pair<int,int> circle\_edge;  
 5) int circle\_r;  
 6) int const\_k;  
 7) int iteration = 1;

8) bool isInBounds = false;

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.

Восьмое — логическое значение, входит ли текущая группа в окружность, или нет

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;

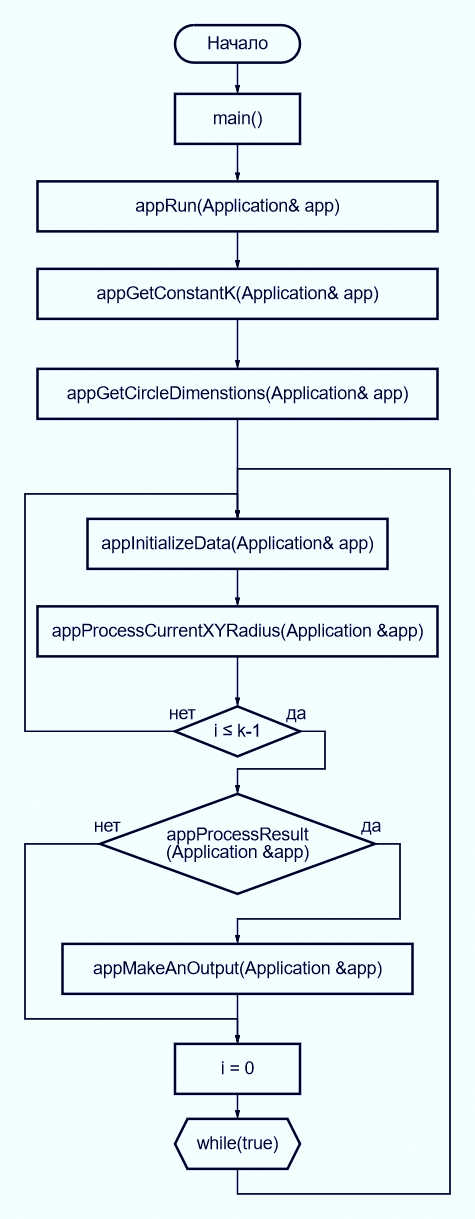
bool appInitializeData(Application &app); - Считывание очередной пары координат;

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

#include "application.h"  
#include <iostream>  
  
int main() {  
  
 std::cout << "main.cpp" << std::endl;  
  
 Application app;  
  
 int ret = appRun(app);  
  
 return ret;  
}

application.h

#pragma once  
#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_H#include <climits>  
#include <utility>  
#include <vector>  
struct Application {  
  
 *//assume 1 - bool of whether radius belongs to the target  
 // 2 - 1- Computed R of a given set of XY,  
 // 2 - initial XY* std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>> temp\_group;  
 std::pair <int, int>init\_xy;  
  
 *//CIRCLE  
 //Assume 1-X 2-Y* std::pair<int,int> circle\_center;  
 std::pair<int,int> circle\_edge;  
 int circle\_r;  
 int const\_k;  
 int iteration = 1;

bool isInBounds = false;

};  
int appRun(Application& app);  
  
bool appGetConstantK(Application &app);  
bool appGetCircleDimensions(Application &app);  
bool appInitializeData(Application &app);  
bool appProcessCurrentXYRadius(Application &app);  
bool appProcessResult(Application &app);  
bool appGetOutputToUser(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

*application.cpp*

#include "application.h"  
#include <iostream>  
#include <cmath>  
  
int appRun(Application &app) {  
 if (!appGetConstantK(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!appGetCircleDimensions(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 while (!std::cin.eof()) {  
 int i;  
 while (i <= app.const\_k) {  
 if (i <= app.const\_k - 1) {  
 if (!appInitializeData(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!appProcessCurrentXYRadius(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 }  
 else {  
 appProcessResult(app);  
 appGetOutputToUser(app);  
 app.temp\_group.clear();  
 }  
 ++i;  
 }  
 ++app.iteration;  
 i = 0;  
 }  
 return 0;  
}  
  
bool appGetConstantK(Application &app) {  
 std::cout << "Input a K constant:" << std::endl;  
 std::cin >> app.const\_k;  
 std::cout << app.const\_k << std::endl;  
 return true;  
}  
  
bool appGetCircleDimensions(Application &app) {  
 std::cout << "Input CENTER 'X Y' coordinate of circle: " << std::endl;  
 std::cin >> app.circle\_center.first >> app.circle\_center.second;  
 std::cout << app.circle\_center.first <<"/"<< app.circle\_center.second << std::endl;  
  
 std::cout << "Input EDGE 'X Y' coordinate of circle: " << std::endl;  
 std::cin >> app.circle\_edge.first >> app.circle\_edge.second;  
 std::cout << app.circle\_edge.first <<"/"<< app.circle\_edge.second << std::endl;  
  
 app.circle\_r = sqrt(  
 pow(app.circle\_edge.first - app.circle\_center.first, 2) +  
 pow(app.circle\_edge.second - app.circle\_center.second, 2));  
  
 return true;  
}  
bool appInitializeData(Application &app) {  
  
 std::cin >> app.init\_xy.first >> app.init\_xy.second;  
  
 return true;  
}  
bool appProcessCurrentXYRadius(Application &app) {  
 app.temp\_group.push\_back(std::make\_pair(false,std::make\_pair(sqrt(  
 pow(app.init\_xy.first - app.circle\_center.first, 2) +  
 pow(app.init\_xy.second - app.circle\_center.second, 2)),  
 std::make\_pair(app.init\_xy.first, app.init\_xy.second))));  
  
 return true;  
}  
  
bool appProcessResult(Application &app) {  
  
 int counter = 0;  
  
 for (int j = 0; j < app.temp\_group.size(); ++j) {  
 if (app.temp\_group[j].second.first < app.circle\_r) {  
 app.temp\_group[j].first = true;  
 ++counter;  
 }  
 }  
 if (counter == app.const\_k){  
 app.isInBounds = true;  
 }  
 else{  
 app.isInBounds = false;  
 }  
  
 return false;  
}  
  
bool appGetOutputToUser(Application &app) {  
 if(app.isInBounds) {  
 std::cout << app.iteration << " Iteration - group is" << " inside the circle, their values are:" << std::endl;  
 for (int i = 0; i < app.const\_k; ++i) {  
  
 std::cout << "(" << app.temp\_group[i].second.second.first << "/" << app.temp\_group[i].second.second.second  
 << ") ";  
 }  
 std::cout << std::endl;  
 }  
 else{  
 std::cout << app.iteration << " Iteration - group is OUT OF BOUNDS,"<< "their values are:" << std::endl;  
 for (int i = 0; i < app.const\_k; ++i) {  
  
 std::cout << "(" << app.temp\_group[i].second.second.first << "/" << app.temp\_group[i].second.second.second  
 << ") ";  
 }  
 std::cout << std::endl;  
 }  
 app.isInBounds = false;  
 return true;  
}

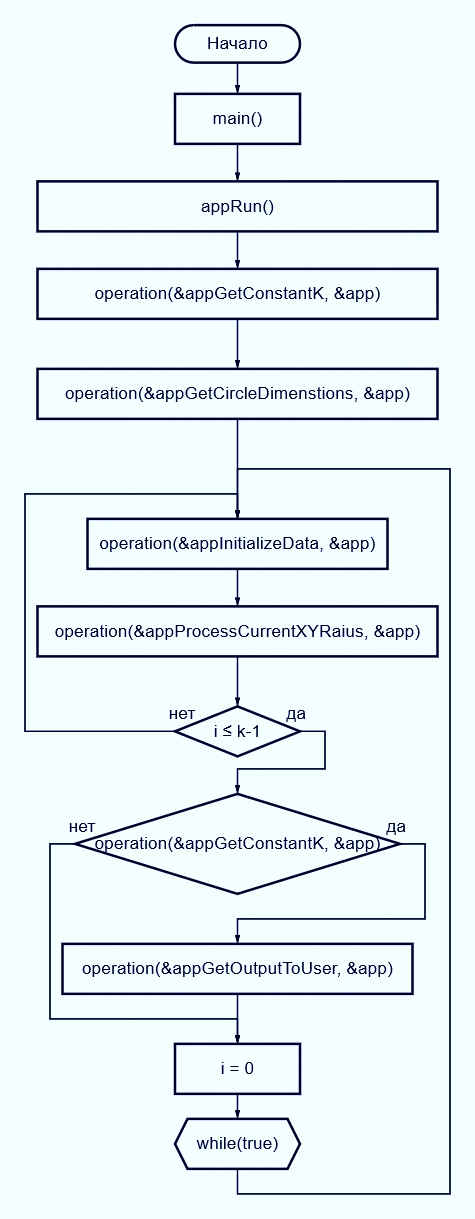
# Контрольная работа №2

**«Настройка индуктивного вычислителя**

**с использованием функции обратного вызова»**

## 2.1 Архитектура программной системы

Алгоритм обработки поступающих значений не претерпел изменений, но был видоизменён стиль вызова функций, и передачи им данных для работы. Теперь мы используем функцию , передаём ей в качестве аргументов, ссылку на функцию для запуска, и ссылку на данные, с которыми предстоит работать.

Логично предположить, что исходя из внесённых изменений, немного изменилась структура программы, обновлённая версия, в виде блок-схемы, представлена ниже.

## 2.2 Использование индуктивного вычислителя

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

//application.h

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

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

// application.cpp

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

|  |  |
| --- | --- |
| 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 |
| (ADT) | данные с которыми работала программа. |

# Приложение 1

main.cpp

#include "application.h"  
#include <iostream>  
  
int main() {  
  
 std::cout << "main.cpp" << std::endl;  
  
 int ret = appRun();  
  
 return ret;  
}

application.h

#include <climits>  
#include <utility>  
#include <vector>  
struct Application {  
  
 *//assume 1 - bool of whether radius belongs to the target  
 // 2 - 1- Computed R of a given set of XY,  
 // 2 - initial XY* std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>> temp\_group;  
 std::pair <int, int>init\_xy;  
  
 *//CIRCLE  
 //Assume 1-X 2-Y* std::pair<int,int> circle\_center;  
 std::pair<int,int> circle\_edge;  
 int circle\_r;  
 int const\_k;  
 int iteration = 1;

bool isInBounds = false;

};  
  
typedef bool (\*Callback)(void \*ADT);  
bool operation(Callback callback, void \*ADT);  
int appRun();  
  
bool appGetConstantK(void \*object);  
bool appGetCircleDimensions(void \*object);  
bool appInitializeData(void \*object);  
bool appProcessCurrentXYRadius(void \*object);  
bool appProcessResult(void \*object);  
bool appGetOutputToUser(void \*object);  
  
#endif *//NNTU\_APPLICATION\_H*

application.cpp

#include "application.h"  
#include <iostream>  
#include <cmath>  
  
bool operation(Callback callback, void \*ADT){  
 return (\*callback)(ADT);  
}  
  
bool appGetConstantK(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 std::cout << "Input a K constant:" << std::endl;  
 std::cin >> app.const\_k;  
 std::cout << app.const\_k << std::endl;  
 return true;  
}  
  
bool appGetCircleDimensions(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 std::cout << "Input CENTER 'X Y' coordinate of circle: " << std::endl;  
 std::cin >> app.circle\_center.first >> app.circle\_center.second;  
 std::cout << app.circle\_center.first <<"/"<< app.circle\_center.second << std::endl;  
  
 std::cout << "Input EDGE 'X Y' coordinate of circle: " << std::endl;  
 std::cin >> app.circle\_edge.first >> app.circle\_edge.second;  
 std::cout << app.circle\_edge.first <<"/"<< app.circle\_edge.second << std::endl;  
  
 app.circle\_r = sqrt(  
 pow(app.circle\_edge.first - app.circle\_center.first, 2) +  
 pow(app.circle\_edge.second - app.circle\_center.second, 2));  
  
 return true;  
}  
  
bool appInitializeData(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 std::cin >> app.init\_xy.first >> app.init\_xy.second;  
  
 return true;  
}  
  
bool appProcessCurrentXYRadius(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 app.temp\_group.push\_back(std::make\_pair(false,std::make\_pair(sqrt(  
 pow(app.init\_xy.first - app.circle\_center.first, 2) +  
 pow(app.init\_xy.second - app.circle\_center.second, 2)),  
 std::make\_pair(app.init\_xy.first, app.init\_xy.second))));  
  
 return true;  
}  
  
bool appProcessResult(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 int counter = 0;  
  
 for (int j = 0; j < app.temp\_group.size(); ++j) {  
 if (app.temp\_group[j].second.first < app.circle\_r) {  
 app.temp\_group[j].first = true;  
 ++counter;  
 }  
 }  
  
 if (counter == app.const\_k){  
 app.isInBounds = true;  
 }  
 else{  
 app.isInBounds = false;  
 }  
  
 return false;  
}  
  
bool appGetOutputToUser(void \*object) {  
  
 Application &app = \*(Application\*) object;  
  
 if(app.isInBounds) {  
 std::cout << app.iteration << " Iteration - group is" << " inside the circle, their values are:" << std::endl;  
  
 for (int i = 0; i < app.const\_k; ++i) {  
  
 std::cout << "(" << app.temp\_group[i].second.second.first << "/" << app.temp\_group[i].second.second.second  
 << ") ";  
  
 }  
 std::cout << std::endl;  
 }  
 else{  
 std::cout << app.iteration << " Iteration - group is OUT OF BOUNDS,"<< "their values are:" << std::endl;  
 for (int i = 0; i < app.const\_k; ++i) {  
  
 std::cout << "(" << app.temp\_group[i].second.second.first << "/" << app.temp\_group[i].second.second.second  
 << ") ";  
  
 }  
 std::cout << std::endl;  
 }  
 app.isInBounds = false;  
 return true;  
}  
  
int appRun() {  
  
 Application app;  
  
 if (!operation(&appGetConstantK, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!operation(&appGetCircleDimensions, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 while (!std::cin.eof()) {  
 int i;  
 while (i <= app.const\_k) {  
 if (i <= app.const\_k - 1) {  
 if (!operation(&appInitializeData, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!operation(&appProcessCurrentXYRadius, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 }  
 else {  
 operation(&appProcessResult, &app);  
 operation(&appGetOutputToUser, &app);  
 app.temp\_group.clear();  
 }  
 ++i;  
 }  
 ++app.iteration;  
 i = 0;  
 }  
 return 0;  
}