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 28)**

Выполнил:

Студент гр. 22-Рз Наумов А.А.

Проверил:

к.т.н., доцент кафедры ИРС Сидоров С.Б.

Нижний Новгород,

2024 г.

Оглавление

[Контрольная работа № 1. 3](#__RefHeading___Toc2035_2758904751)

[1.1. Постановка задачи 3](#__RefHeading___Toc2041_2758904751)

[1.2. Алгоритм индуктивной обработки 3](#__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)

[Sample: 8](#__RefHeading___Toc5558_2519614503)

[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 12](#__RefHeading___Toc11497_2758904751)

[Приложение 1 14](#__RefHeading___Toc11499_2758904751)

# 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.

Задание по варианту

Обнаружение наиболее длинного участка монотонного возрастания значений последовательных элементов, при условии что разность значений последнего и первого элементов участка не менее чем D. Результатом является интервал: (начальный номер, конечный номер). Тип элемента — целочисленный.

## 1.2. Алгоритм индуктивной обработки

Примем следующие сокращения, и каким значением инициализируются (если значение явно не указывается, то его упоминание пропускается):

– индекс левого элемента, обрабатываемой в данный момент последовательности;

– значение левого элемента, обрабатываемой в данный момент последовательности;

– индекс правого элемента, обрабатываемой в данный момент последовательности;

– значение правого элемента, обрабатываемой в данный момент последовательности;

()– количество элементов в текущей, обрабатываемой последовательности;

– индекс левого элемента, максимальной по кол-ву элементов последовательности, при разнице значений начального и конечного, на данный момент, элемента, больше, чем D;

– индекс правого элемента, максимальной по кол-ву элементов последовательности, при разнице начального и конечного, на данный момент, элемента, больше, чем D;;

()– максимальное количество элементов, для обработанных последовательностей, на данный момент.

- очередное поступившее значение;

- индекс поступившего элемента;

- булевое значение, вводится как ;

- предыдущее поступившее значение;

– число вводимое пользователем, по условию задачи.

- элемент выходной последовательности, представлен набором из двух элементов

Отклик вычислителя определим как .

Рассмотрим базовые условия:

;

;

;

;

Предикаты:

Результат: ;

Результат: ;

Результат: ;

Результат: ;

Результат:

Предикат соответствует случаю, когда программа находится на первой итерации. , когда итерация не является первой, и текущее, поступившее значение меньше предыдущего. , если итерация не является первой, поступившее значение больше предыдущего, и кол-во элементов в текущей последовательности больше, чем в предыдущей, максимально достигнутой вычислителем. , в случае, если итерация не является первой, текущее значение больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **меньше** . , когда итерация не является первой, текущее значение больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **больше** .

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

Условия, используемые в правиле пересчёта и охватывают все возможные ситуации, то есть . Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчёта 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.

Разберём используемый абстрактный тип данных.

АТД Application

Определяется структурный тип данных **Application**, содержащий следующие поля:

int tLi;  
 int tLv;  
 int tRi;  
 int tRv;  
 int fLi;  
 int fRi;  
 int tCS;  
 int fCS;  
 bool isFirst;  
 int n;  
 int D;  
 int xn;

Переменные по условию из пункта 1.2.

application.h

В заголовочном файле application.h объявляется глобальная функция int appRun(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 , 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 app\_get\_D(Application &app);  
 bool app\_get\_element(Application &app);  
 bool app\_process(Application &app);

bool app\_output\_match(Application &app);

Рассмотрим каждую функцию в отдельности:

bool app\_get\_D(Application &app); - Получение числа .

bool app\_get\_element(Application &app);- отвечает за получение очередного значения, от пользователя.  
 bool appProcess(Application &app); - Содержит основную логическую функцию обработки.  
 bool app\_output\_match(Application &app); - выводит промежуточный результат на стандартное устройство вывода.

**Sample:**

1 2 3 4 5 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3

**ASSERT {15, 27}**

**OUTPUT {15, 27}**

# Приложение 1.

*//main.cpp*#include "Application.h"  
  
int main() {  
  
 Application app;  
 int ret = app\_run(app);  
  
 return ret;  
}

*//application.h*#ifndef APPLICATION\_H  
#define APPLICATION\_Hstruct Application {  
 int tLi;  
 int tLv;  
 int tRi;  
 int tRv;  
 int fLi;  
 int fRi;  
 int tCS;  
 int fCS;  
 bool isFirst;  
 int n;  
 int D;  
 int xn;  
  
 Application(): tLi(0), tLv(0), tRi(0), tRv(0), fLi(0), fRi(0),  
 tCS(1), fCS(1), isFirst(true), n(0), D(0), xn(0) {}  
};  
  
bool app\_get\_D(Application &app);  
bool app\_get\_element(Application &app);  
bool app\_output\_match(Application &app);  
bool app\_process(Application &app);  
int app\_run(Application &app);  
  
#endif *// APPLICATION\_H*

*//application.cpp*#include <iostream>  
#include "Application.h"  
  
bool app\_get\_D(Application &app) {  
 std::cout << "Enter the value of D: ";  
 std::cin >> app.D;  
 std::cout << app.D << std::endl;  
 return true;  
}  
  
bool app\_get\_element(Application &app) {  
 std::cin >> app.xn;  
 return true;  
}  
  
bool app\_output\_match(Application &app) {  
 if (app.fLi != 0 && app.fRi != 0) {  
 std::cout << "Current best segment: from " << app.fLi + 1 << " to " << app.fRi + 1 << std::endl;  
 }  
 return true;  
}  
  
bool app\_process(Application &app) {  
 if (app.isFirst) {  
 app.tLv = app.tRv = app.xn;  
 app.isFirst = false;  
 }  
 if (app.tCS == 1) {  
 app.tLi = app.n - 1;  
 app.tLv = app.tRv;  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS = 2;  
 } else {  
 if (app.xn > app.tRv) {  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS++;  
 } else {  
 app.tLi = app.n;  
 app.tLv = app.xn;  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS = 1;  
 }  
 if (app.tCS > app.fCS && (app.tRv - app.tLv >= app.D)) {  
 app.fLi = app.tLi;  
 app.fRi = app.tRi;  
 app.fCS = app.tCS;  
 }  
 }  
 app.n++;  
 return true;  
}  
  
int app\_run(Application &app) {  
 app\_get\_D(app);  
 while (!std::cin.eof()) {  
 app\_get\_element(app);  
 app\_process(app);  
 app\_output\_match(app);  
 }  
 return 0;  
}

# 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 app\_Run, 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*#include "Application.h"  
  
int main() {  
  
 Application app;  
 int ret = app\_run(&app);  
  
 return ret;  
}

*//application.h*#ifndef APPLICATION\_H  
#define APPLICATION\_Hstruct Application {  
 int tLi;  
 int tLv;  
 int tRi;  
 int tRv;  
 int fLi;  
 int fRi;  
 int tCS;  
 int fCS;  
 bool isFirst;  
 int n;  
 int D;  
 int xn;  
  
 Application(): tLi(0), tLv(0), tRi(0), tRv(0), fLi(0), fRi(0),  
 tCS(1), fCS(1), isFirst(true), n(0), D(0), xn(0) {}  
};  
  
typedef bool (\*Callback)(void \*object);  
bool operation(Callback callback, void \*data);  
  
bool app\_get\_D(void\* object);  
bool app\_get\_element(void\* object);  
bool app\_output\_match(void\* object);  
bool app\_process(void\* object);  
int app\_run(void\* object);  
  
#endif *// APPLICATION\_H*

*//application.cpp*#include <iostream>  
#include "application.h"  
  
bool operation(Callback callback, void \*data) {  
 return (\*callback)(data);  
}  
  
bool app\_get\_D(void\* object) {  
 Application &app = \*((Application\*)object);  
 std::cout << "Enter the value of D: ";  
 std::cin >> app.D;  
 std::cout << app.D << std::endl;  
 return true;  
}  
  
bool app\_get\_element(void\* object) {  
 Application &app = \*((Application\*)object);  
 std::cin >> app.xn;  
 return true;  
}  
  
bool app\_output\_match(void\* object) {  
 Application &app = \*((Application\*)object);  
 if (app.fLi != 0 && app.fRi != 0) {  
 std::cout << "Current best segment: from " << app.fLi + 1 << " to " << app.fRi + 1 << std::endl;  
 }  
 return true;  
}  
  
bool app\_process(void\* object) {  
 Application &app = \*((Application\*)object);  
 if (app.isFirst) {  
 app.tLv = app.tRv = app.xn;  
 app.isFirst = false;  
 }  
 if (app.tCS == 1) {  
 app.tLi = app.n - 1;  
 app.tLv = app.tRv;  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS = 2;  
 } else {  
 if (app.xn > app.tRv) {  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS++;  
 } else {  
 app.tLi = app.n;  
 app.tLv = app.xn;  
 app.tRi = app.n;  
 app.tRv = app.xn;  
 app.tCS = 1;  
 }  
 if (app.tCS > app.fCS && (app.tRv - app.tLv >= app.D)) {  
 app.fLi = app.tLi;  
 app.fRi = app.tRi;  
 app.fCS = app.tCS;  
 }  
 }  
 app.n++;  
 return true;  
}  
  
int app\_run(void\* object) {  
 Application &app = \*((Application\*)object);  
 app\_get\_D(&app);  
 while (!std::cin.eof()) {  
 app\_get\_element(&app);  
 app\_process(&app);  
 app\_output\_match(&app);  
 }  
 return 0;  
}