

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**

<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>7</b>
<b>АТД Application.....</b>	<b>7</b>
<b>application.h.....</b>	<b>8</b>
<b>algo.h.....</b>	<b>8</b>
<b>Приложение 1.....</b>	<b>11</b>
<b>Контрольная работа №2.....</b>	<b>17</b>
<b>2.1 Архитектура программной системы.....</b>	<b>17</b>
<b>2.2 Использование индуктивного вычислителя.....</b>	<b>18</b>
<b>Приложение 1.....</b>	<b>19</b>

## 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.

# 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

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

$tLi$  — индекс левого элемента, обрабатываемой в данный момент последовательности;

$tLv$  — значение левого элемента, обрабатываемой в данный момент последовательности;

$tRi$  — индекс правого элемента, обрабатываемой в данный момент последовательности;

$tRv$  — значение правого элемента, обрабатываемой в данный момент последовательности;

$tCS$  ( $tempCS$ ) — количество элементов в текущей, обрабатываемой последовательности;

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

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

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

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

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

$x_n$  - очередное поступившее значение;

$lx$  - предыдущее поступившее значение;

$\Delta v$  - разница значений первого и последнего элемента максимальной, подходящей, на данный момент, последовательности,  $x_n - fLv$ ;

$\Delta i$  - разница индексов первого и последнего элемента максимальной, подходящей, на данный момент, последовательности,  $n - fCS$ ;

$n$  - индекс поступившего элемента; (Равен 0, т. к. это итератор)

$D$  – цифра, вводимая пользователем, по условию задачи.

$y$  - элемент выходной последовательности, представлен набором из трех элементов  
 $y = (n, fLi, fRi)$

Отклик вычислителя  $q$  определим как  $q = \{ \langle y \rangle, \forall x_n \}$ .

Рассмотрим каждое из условий:

$$r_1(n=0);$$

$$r_2(x_n \leq lx);$$

$$r_3(tCS > fCS);$$

$$r_4(\Delta v > D);$$

Имея набор базовых условий, можно сформировать предикаты. Так же, для переменных, которые претерпевают изменения, приведены правила пересчёта. Если переменная не пересчитывается, то правило пересчёта **не приводится**.

$R_1(): r_1$  - справедливо для  $n=0$ ;

Результат:  $R_1(.) \rightarrow fLi=n, fRi=n, fLv=x_n, fRv=x_n, q=<y>, lx=x_n$ ;

$R_2(): \neg r_1 \wedge r_2$  - справедливо для  $x_n < lx$ ;

Результат:  $R_2(.) \rightarrow tCS=0, \Delta x=x_n - fLv, \Delta i=n - tCS, q=<y>, lx=x_n$ ;

$R_3(): \neg r_1 \wedge \neg r_2 \wedge \neg r_3$  - справедливо для  $n>0 \wedge x_n > x_{n-1} \wedge tCS < fCS$ ;

Результат:  $R_3(.) \rightarrow tCS=tCS+1, \Delta x=x_n - fLv, \Delta i=n - tCS, q=<y>, lx=x_n$ ;

$R_4(): \neg r_1 \wedge \neg r_2 \wedge r_3 \wedge \neg r_4$  - справедливо для  $(n>0) \wedge (x_n > x_{n-1}) \wedge (tCS > fCS) \wedge (\Delta v < D)$ ;

Результат:  $R_4(.) \rightarrow tCS=tCS+1, \Delta x=x_n - fLv, \Delta i=n - tCS, q=<y>, lx=x_n$ ;

$R_5(): \neg r_1 \wedge \neg r_2 \wedge r_3 \wedge r_4$  - справедливо для  $(n>0) \wedge (x_n > x_{n-1}) \wedge (tCS > fCS) \wedge (\Delta v > D)$ ;

Результат:

$R_5(.) \rightarrow tCS=tCS+1, \Delta x=x_n - fLv, \Delta i=n - tCS, fLi=\Delta i, fRi=x_n, fCS=tCS, q=<y>, lx=x_n$

Предикат  $R_1(.)$  соответствует случаю, когда программа находится на первой итерации.  $R_2(.)$ , когда итерация не является первой, и текущее, поступившее значение меньше предыдущего.  $R_3(.)$ , если итерация не является первой, поступившее значение больше предыдущего, и кол-во элементов в текущей последовательности больше, чем в предыдущей, максимально достигнутой вычислителем.  $R_4(.)$ , в случае, если итерация не является первой, текущее значение  $x$  больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **меньше**  $D$ .  $R_5(.)$ , когда итерация не является первой, текущее значение  $x$  больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **больше**  $D$ .

В общем случае вид формулы, по которой вычисляется новое значение величины, определяется набором условий  $R=(R_1(.), R_2(.), R_3(.), R_4(.), R_5(.))$ .

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

Условия, используемые в правиле пересчёта *fLi* и *fRi* охватывают все возможные ситуации, то есть  $R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 = true$ . Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчета величины.

Кроме того, выполняется условие  $\forall x_n: R_5 \rightarrow R_1 \cap R_2 \cap R_3 \cap R_4 \cap R_5 = false$ , то есть не допускается одновременное выполнение различных условий. И обратное утверждение,  $\neg \forall x_n: R_5 \rightarrow R_1 \cap R_2 \cap R_3 \cap R_4 \cap R_5 = true$ . 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 семь полей:

```
int const_D;  
std::pair<int, int> current_element;  
std::pair<int, int> last_element;  
std::pair<int, int> finalLeft;  
std::pair<int, int> finalRight;  
int tempCS = 0;  
int finalCS = 0;
```

Первое — константа  $D$ , используемая для сравнения значений крайних элементов последовательности. Логика использования пары интов следующая, первое значение — индекс, второе — значение. Следуя ей, объявляются переменные — текущий элемент, последний элемент, финальный левый и финальный правый элементы. Целочисленными, так же, объявляются переменные *tempCS/finalCS*, хранящие временный стрик текущей последовательности, и максимальный достигнутый за время работы программы, соответственно.

## 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 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 appInitializeData(Application &app);  
bool appGetConstantD(Application &app);  
bool appProcess(Application &app);  
bool appGetOutputToUser(Application &app);
```

Рассмотрим каждую функцию в отдельности:

`bool appInitializeData(Application &app);` - отвечает за получение очередного значения, от пользователя.

`bool appGetConstantD(Application &app);` - Получение числа  $D$ .

`bool appProcess(Application &app);` - Содержит основную логическую функцию обработки.

`bool appGetOutputToUser(Application &app);` - выводит промежуточный результат на стандартное устройство вывода.

## algo.h



В файле объявляются функции для базовых условий  $r$ , и одна вспомогательная с типом `void`, а именно:

```
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);
```

Все они принимают указатель на тип переменную по типу данных АТД Application.

Рассмотрим каждую функцию в отдельности:

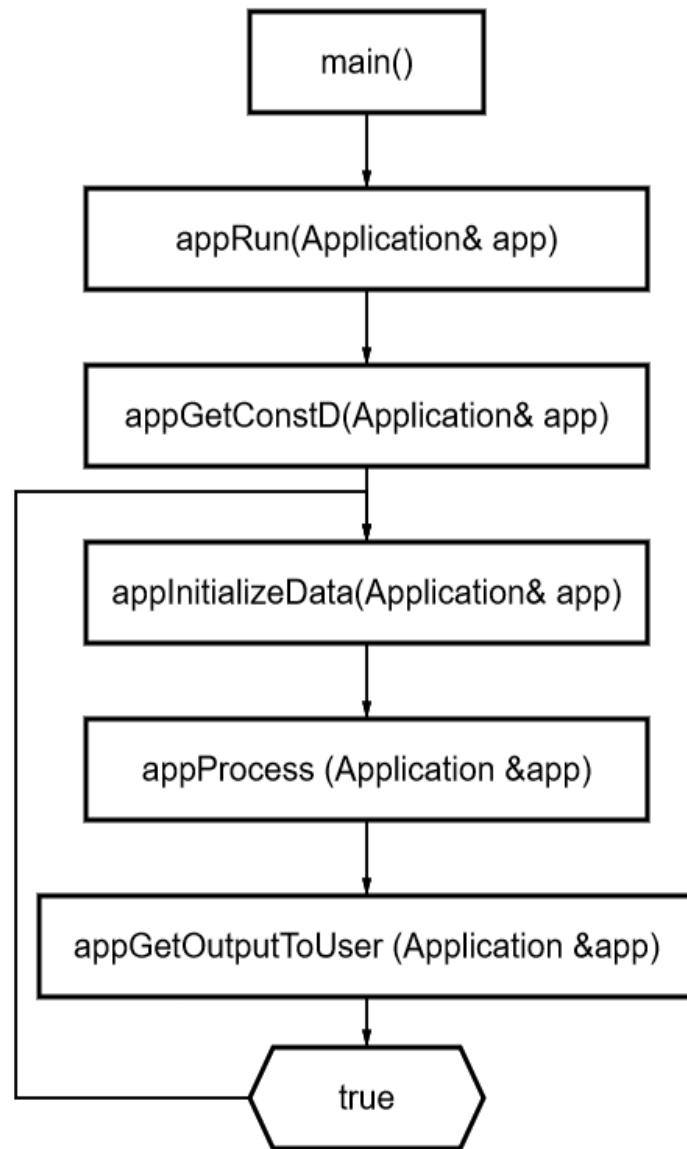
`bool algo_check_first_iteration(void *object);` - соответствует выражению  $r_1(n=0) \rightarrow (fLi \wedge fRi)=n \wedge (fLv \wedge fRv)=x_n$ . Выполняя действия и возвращая *true*, в случае  $n=0$ . Словами, приравнивает левый и правый индексы к параметрам текущего, индекс и значение.

`bool algo_check_ascending(void *object);` - соответствует выражению  $r_2(x_n \leq x_{n-1}) \rightarrow tCS=0$ . Выполняя действия и возвращая *true*, в случае  $x_n \leq x_{n-1}$ . Словами, если текущий элемент меньше предыдущего, то счетчик текущей последовательности сбрасывается, и возвращается *true*.

`bool algo_check_breakage(void *object);` - соответствует выражению  $r_3(tCS > fCS)$ . Действий функция не выполняет, только проверяет превосходство текущей последовательности над максимально достигнутой ранее. Если так, то вернется *true*.

`bool algo_check_breakage(void *object);` - соответствует выражению  $r_4(x_n - fLv > D) \rightarrow fLi = n - tCS \wedge fRi = n \wedge fCS = tCS$ . Основное условие наступления пересчета максимальных индексов. Если значение текущего минус значение первого элемента текущей последовательности больше  $D$ , то максимальный левый индекс приравнивается к  $n$  минус  $k$ -во элементов текущей последовательности, правый индекс к  $n$ , и максимальное  $k$ -во элементов приравнивается к текущему.

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;  
}
```



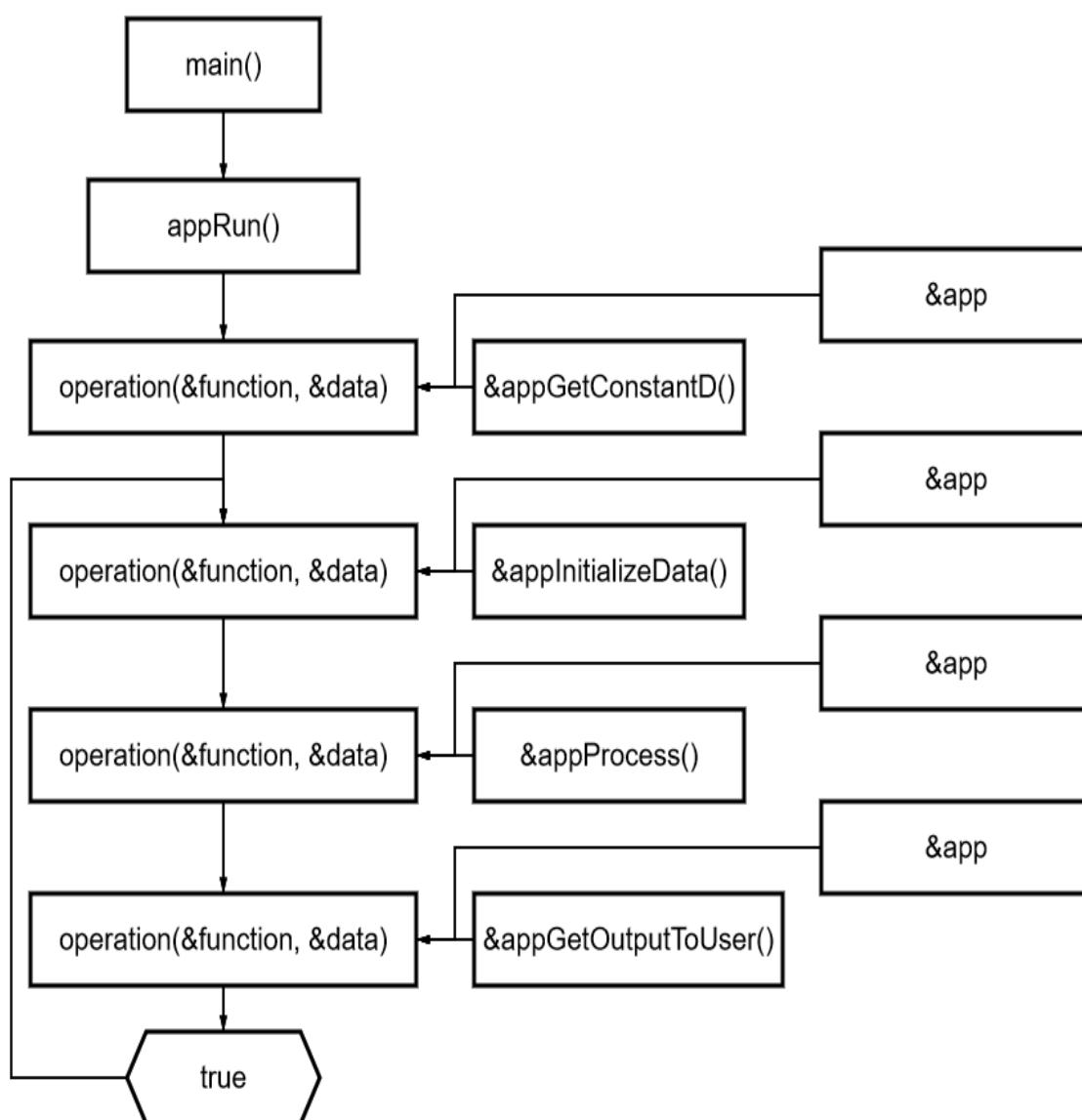
## Контрольная работа №2

### «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
<code>typedef</code>	аллиас на функцию
<code>bool</code>	тип функции
<code>(*Callback)</code>	указатель на тип функции
<code>(void *object)</code>	указатель на тип принимаемого аргумента, и его название

```
// application.cpp  
bool operation(Callback callback, void *data) {  
    return (*callback)(data);  
}
```

Элемент	Назначение
<code>bool</code>	тип функции
<code>operataion</code>	название
<code>(Callback callback</code>	тип и имя принимаемого аргумента
<code>void *data)</code>	указатель на любой тип данных, имя аргумента
<code>return</code>	возвращаемое значение
<code>(*callback)</code>	результат работы вызываемой функции
<code>(data)</code>	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;
}

```