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 1)

Выполнил:
Студент гр. 22-Рз Чеботарёв Р.А.

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

Нижний Новгород,
2024 г.

Оглавление

Задание по варианту.....3

Контрольная работа № 1.....4

1.1. Постановка задачи.....4

1.2. Алгоритм индуктивной обработки.....4

1.3. Архитектура программной реализации вычислителя. 5

АТД Application.....6

application.h.....6

Приложение 1.....8

Контрольная работа №2.....11

2.1 Архитектура программной системы.....11

2.2 Использование индуктивного вычислителя.....12

Приложение 1.....13

Задание по варианту

На вход поступают целочисленные элементы последовательности. Необходимо обнаружить события, когда соседние элементы отличаются более чем на R. Описание события на выходе должно включать время наступления события (номер текущего элемента) и значения 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 *xi*, где *i* — порядковый номер элемента, начиная с 1. Реализовать указанную в варианте обработку $f(X)$ последовательности элементов $X = \langle x1, x2, x3, \ldots \rangle$ с использованием схемы индуктивной обработки на пространстве последовательностей. Полученный набор выходных значений $Y = \langle y1, y2, y3, \ldots \rangle$ 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

Основной алгоритм программы содержится в функции processCurrentValue, для пересчёта очередного, поступившего значения x_n, необходимо соблюсти следующие условия

$R(): \neg r_1 \cap \neg r_2 \cap r_3 \cap r_4$

Стоит подчеркнуть, используемые переменные, и их принятые сокращения:

n – индекс значения — текущая итерация;

x_n – текущее значение;

x_{n-1} – предыдущее значение;

R – число R, введённое пользователем;

Рассмотрим базовое условие:

$r(x_n - x_{n-1} > R)$;

Имея базовое условие, можно сформировать предикат,

$R_1() : r_1$ - справедливо для $x_n - x_{n-1} > R$;

Результат: $R_1() \rightarrow x_n - x_{n-1} \cap R = true$;

Следовательно, мы можем сформулировать обратный предикат, для остальных случаев

$R_2() : \neg r_1$ – для остальных случаев - $x_n - x_{n-1} \leq R$;

Результат: $R_2() \rightarrow x_n - x_{n-1} \cap R = false$

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

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 следствии обработки входящих x_n значений, предусмотренный счётчик *curIndex*, будет изменяться, что, соответственно, будет являться индуктивным расширением.

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

Кроме того, выполняется условие $\forall x_n : R_1 \rightarrow R_1 \cap R_2 = false$, то есть не допускается одновременное выполнение различных условий. И обратное утверждение, $\neg \forall x_n : R_1 \rightarrow R_1 \cap R_2 = true$. Истинное значение предиката обозначает факт наступления связанного с ним события. $\exists x_n : R_1 \rightarrow x_n \in R$ - будет являться конечным заключением, о рассматриваемой программе.

1.3. Архитектура программной реализации вычислителя.

Для обеспечения удобства взаимодействия с необходимыми переменными, нами вводится абстрактный тип данных (далее АТД), рассмотрим его:

АТД Application

Определяется структурный тип данных **`Application`**, содержащий четыре поля:

```
int prevValue = INT_MIN;
int curValue = INT_MIN;
int curIndex = 0;
int constR = 0;
```

1) Предыдущее значение - x_{n-1};

2) Текущее значение - x_n;

3) Итерационный индекс программы;

4) Число R, вводится пользователем;

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 A*pplication*, 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.

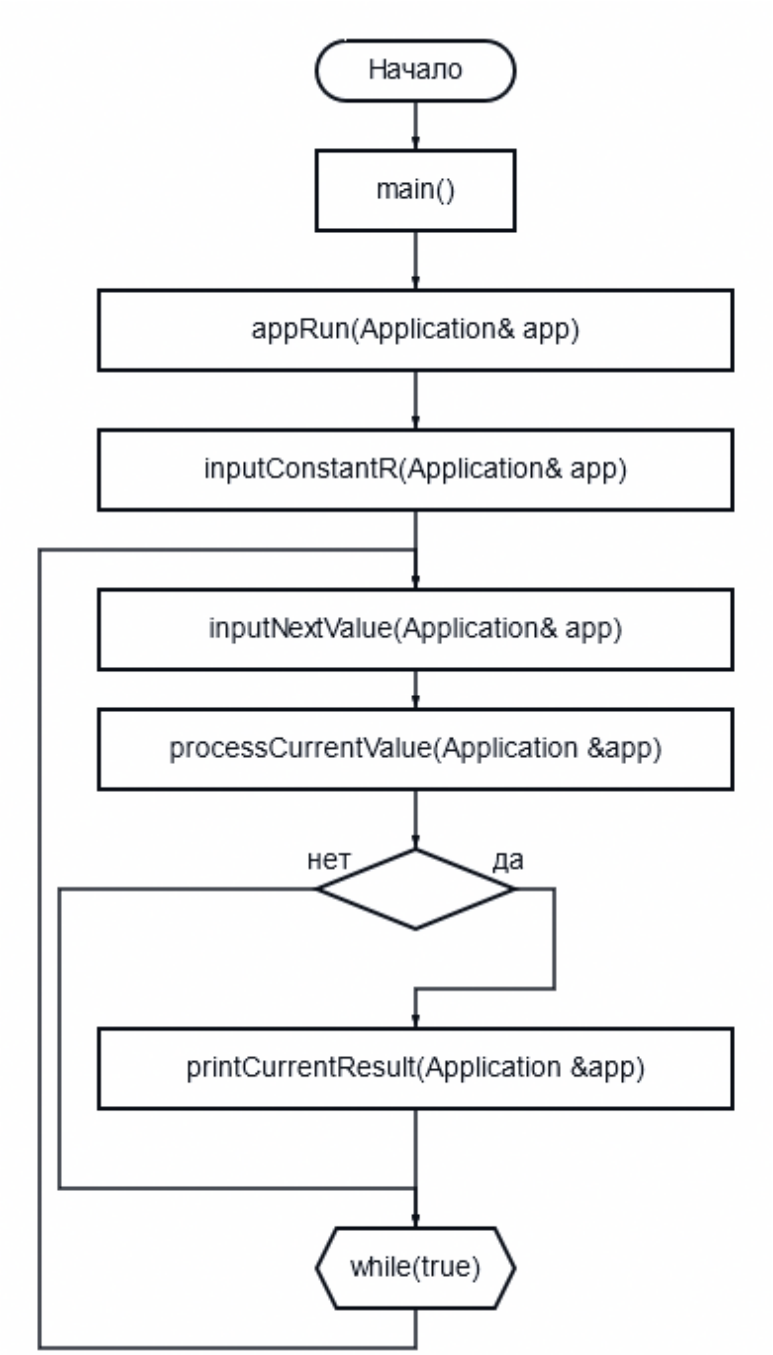
Для выполнения поставленных, условием, задач, в `.h` файле объявляется прототип функции, а в `.cpp` файле, определяются под-функции `appRun`, а именно:

```
bool inputConstantR(Application &app);
bool inputNextValue(Application &app);
bool processCurrentValue(Application &app);
bool printCurrentResult(Application &app);
```

Рассмотрим каждую функцию в отдельности:

`bool inputConstantR(Application &app);`- получение значения R, от пользователя;
`bool inputNextValue(Application &app);`- получение очередного числа x_n;
`bool processCurrentValue(Application &app);` обработка, согласно предикатам;
`bool printCurrentResult(Application &app);` - вывод результата, на итерации, пользователю

Можем представить общую структуру программы в виде блок-схемы:


Начало
main()
appRun(Application& app)
inputConstantR(Application& app)
inputNextValue(Application& app)
processCurrentValue(Application &app)
нет
да
printCurrentResult(Application &app)
while(true)

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>
#include <iostream>

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

struct Application {
    int prevValue = INT_MIN;
    int curValue = INT_MIN;
    int curIndex = 0;
    int constR = 0;
};

int appRun(Application &app);
std::string formatResult(Application &app, bool isFirst);
bool inputConstantR(Application &app);
bool inputNextValue(Application &app);
bool processCurrentValue(Application &app);
bool printCurrentResult(Application &app);

#endif //NNTU_APPLICATION_H
```

```
//application.cpp
#include "application.h"

bool operation(Callback callback, void *data) {
```

```
    return (*callback)(data);
}

int appRun(Application &app) {

    inputConstantR(app);

    //Consider using !std::cin.eof() for testing purposes.
    while (!std::cin.eof()) {

        if(!inputNextValue(app)){
            return 1;
        }

        if (processCurrentValue(app)) {
            printCurrentResult(app);
        }
    }
    return 0;
}

bool inputConstantR(Application &app) {

    std::cout << "Input a R constant, to compare adjacent values:" << std::endl;
    std::cin >> app.constR;

    return true;
}

bool inputNextValue(Application &app) {

    if (std::cin.eof()) {
        return false;
    }
    app.prevValue = app.curValue;
    std::cin >> app.curValue;
    ++app.curIndex;

    return true;
}
```

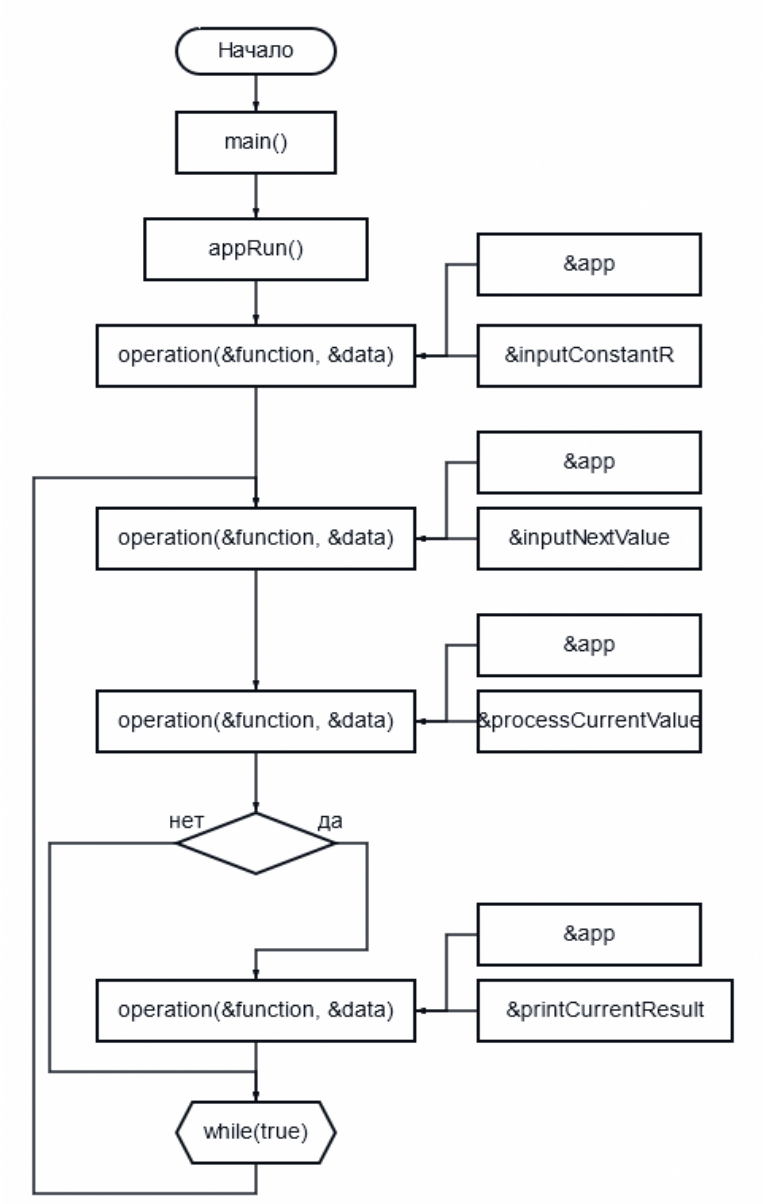
```
bool processCurrentValue(Application &app) {
    return std::abs(static_cast<long>(app.prevValue) - app.curValue) >
app.constR;
}

bool printCurrentResult(Application &app) {
    std::cout << formatResult(app, app.curIndex == 1) << std::endl;
    return true;
}

std::string formatResult(Application &app, bool isFirst) {
    return "{\n\tindex: " + std::to_string(app.curIndex) +
           (!isFirst ? ",\n\tprevious value: " + std::to_string(app.prevValue) : "") +
           ",\n\tcurrent value: " + std::to_string(app.curValue) + "\n}";
}
```

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.

В application.h:

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

typedef — псевдоним типа данных

bool — тип возвращаемого значения

(*Callback) — указатель типа функции

(void *data) — указатель типа аргумента, название

В application.cpp:

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

bool — тип возвращаемого значения
operation — название
Callback - тип первого аргумента
callback — имя первого аргумента
void *data — указатель тип второго аргумента — любой
return — возвращаемое значение
(*callback) – результат работы вызванной функции
(data) – данные, которые эта функция использовала.

Приложение 1

```
//main.cpp
#include "application.h"
#include <iostream>

int main() {
    std::cout << "VAR 1 CODE" << std::endl;
    Application app;
    int ret = appRun(app);
    return ret;
}

//application.h
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <climits>
#include <iostream>

typedef bool (*Callback)(void *object);

bool operation(Callback callback, void *data);

//Data for program to handle.
struct Application {
    int prevValue = INT_MIN;
    int curValue = INT_MIN;
    int curIndex = 0;
    int constR = 0;
};

// To execute application
int appRun(Application &app);

std::string formatResult(Application &app, bool isFirst);
bool inputConstantR(void *rawApp);
bool inputNextValue(void *rawApp);
bool processCurrentValue(void *rawApp);
```

```
bool printCurrentResult(void *rawApp);

#endif //NNTU_APPLICATION_H


//application.cpp
#include "application.h"

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

int appRun(Application &app) {

    operation(&inputConstantR, &app);

    //Consider using !std::cin.eof() for testing purposes.
    while (!std::cin.eof()) {

        if(!operation(&inputNextValue, &app)){
            return 1;
        }

        if (operation(&processCurrentValue, &app)) {
            operation(&printCurrentResult, &app);
        }
    }
    return 0;
}

bool inputConstantR(void *rawApp) {

    Application &app = *((Application *) rawApp);
    std::cout << "Input a R constant, to compare adjacent values:" << std::endl;
    std::cin >> app.constR;
    return true;
}

bool inputNextValue(void *rawApp) {
    Application &app = *((Application *) rawApp);
    if (std::cin.eof()) {
```

```
        return false;
    }
    app.prevValue = app.curValue;
    std::cin >> app.curValue;
    ++app.curIndex;
    return true;
}

bool processCurrentValue(void *rawApp) {
    Application &app = *((Application *) rawApp);
    return std::abs(static_cast<long>(app.prevValue) - app.curValue) >
app.constR;
}

bool printCurrentResult(void *rawApp) {
    Application &app = *((Application *) rawApp);
    std::cout << formatResult(app, app.curIndex == 1) << std::endl;
    return true;
}

std::string formatResult(Application &app, bool isFirst) {
    return "{\n\tindex: " + std::to_string(app.curIndex) +
           (!isFirst ? ",\n\tprevious value: " + std::to_string(app.prevValue) : "") +
           ",\n\tcurrent value: " + std::to_string(app.curValue) + "\n}";
}
```