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 13)

Выполнил:

Студент гр. 22-Рз [redacted]

Проверил:

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

Нижний Новгород,
2024 г.

Оглавление

Задание по варианту

Исключение элементов, значения которых находятся вне интервала $[a, b]$, границы которого изменяются по заданному линейному закону $a(n) = v_a \cdot n + a_0$, $b(n) = v_b \cdot n + b_0$, где n — номер элемента в исходной последовательности. 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 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

Перевычисление очередного значения x_n происходит в функции *appProcess*, которая, в свою очередь, вызывается в функции *appRun*. В общем виде, пересчет

$R(): r_1 \curlywedge r_2$

Рассмотрим каждое из условий:

$r_1(x_n > a)$;

$r_2(x_n < b)$;

Соответственно, зная эти условия, можно сформулировать предикаты,

$R_1(): r_1$ - если $x_n > a$;

Результат: $R_1(.) \to -$;

$R_2(): \neg r_1$ - если $x_n < a$;

Результат: $R_2(.) \to -$;

$R_3(): r_1 \wedge r_2$ - если $x_n > a \wedge x_n < b$;

Результат: $R_3(.): \to x_n \in [a, b]$;

$R_4(): r_1 \wedge \neg r_2$ - если $x_n > a \wedge x_n > b$;

Результат: $R_4(.) \to x_n \notin [a, b]$;

$R_5(): \neg r_1 \wedge \neg r_2$ - если $x_n < a \wedge x_n > b$;

Результат: $R_5(.) \to x_n \notin [a, b]$;

$R_6(): \neg r_1 \wedge r_2$ - если $x_n < a \wedge x_n < b$;

Результат: $R_6(.) \to x_n \notin [a, b]$;

В общем случае вид формулы, по которой вычисляется новое значение величины, определяется набором условий $R = (R_1(.), R_2(.), R_3(.), R_4(.), R_5(.), R_6(.))$.

Приведенные 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 счетчика *iteration*, претерпевает изменения, в результате индуктивной операции над x_n, соответственно работа счетчика будет считаться индуктивным расширением.

Условия, используемые в правиле пересчета охватывают все возможные ситуации, то есть $R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 = true$. Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчета величины.

Кроме того, выполняется условие $\forall x_n : R_3 \to R_1 \cap R_2 \cap R_3 \cap R_4 \cap R_5 \cap R_6 = false$, то есть не допускается одновременное выполнение различных условий. Истинное значение предиката обозначает факт наступления связанного с ним события.

1.3. Архитектура программной реализации вычислителя.

В ходе разработки программы, было принято решение, использовать абстрактный тип данных, для хранения всех используемых переменных. В предложеной программе, он представлен как тип с именем *Application*.

АТД Application

Определяется структурный тип данных **`Application`**, содержащий шесть полей:

```
int iteration = 1;
int cin_read;
int va = INT_MAX;
int a0 = INT_MAX;
int vb = INT_MAX;
int b0 = INT_MAX;
```

Выбранные имена переменных говорят сами за себя, но тем не менее, разберем из предназначение. 1 — счетчик итерации программы, для учета индекса n. 2 – очередное значение x_n. Последние четыре значения — константы, вводимые пользователем, по условию задачи.

Заголовочный файл application.h

В нем объявляется глобальная функция `appRun`, возвращающая целочисленное значение. Эта функция отвечает за исполнение программы, 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 appGetData(Application &app);
int appGetA(Application &app);
int appGetB(Application &app);
bool appProcess(Application &app);
bool appDoOutput(Application &app);
```

Рассмотрим каждую функцию в отдельности:

bool appGetData(Application &app); - получение очередного значения x_n;

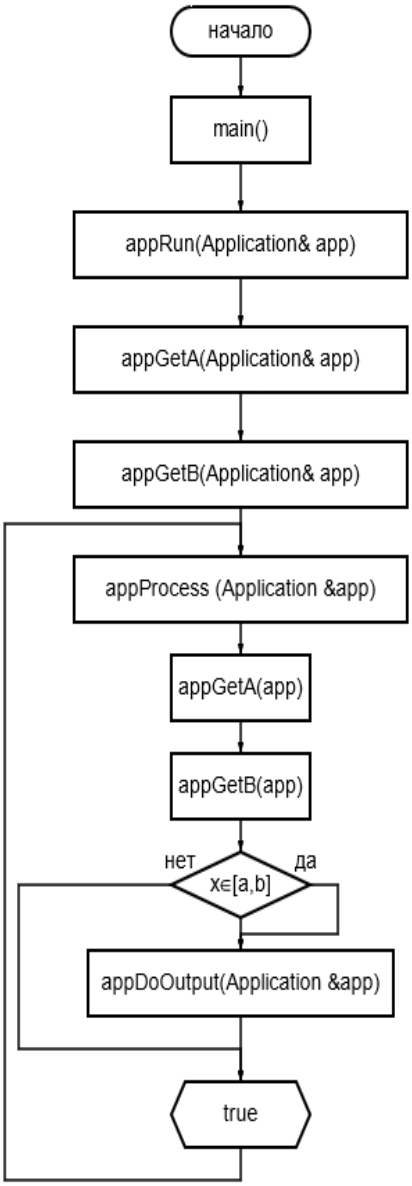
int appGetA(Application &app); - считывание va и $a0$, если вызвана в первый раз. Или просчет границы a, для значения x_n.

int appGetB(Application &app); - считывание vb и $b0$, если вызвана в первый раз. Или просчет границы b, для значения x_n.

bool appProcess(Application &app); - вердикт, входит ли, очередное значение x_n в границы $[a,b]$.

bool appDoOutput(Application &app) — если входит, то вывод x_n на стандартное устройство вывода;

Можем представить общую структуру программы в виде блок-схемы:



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>

struct Application {
    int iteration = 1;
    int cin_read;
    int va = INT_MAX;
    int a0 = INT_MAX;
    int vb = INT_MAX;
    int b0 = INT_MAX;
};

int appRun(Application& app);
bool appGetData(Application &app);
int appGetA(Application &app);
int appGetB(Application &app);
bool appProcess(Application &app);
bool appDoOutput(Application &app);

#endif //NNTU_APPLICATION_H

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

int appRun(Application &app) {
    if (!appGetA(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
```

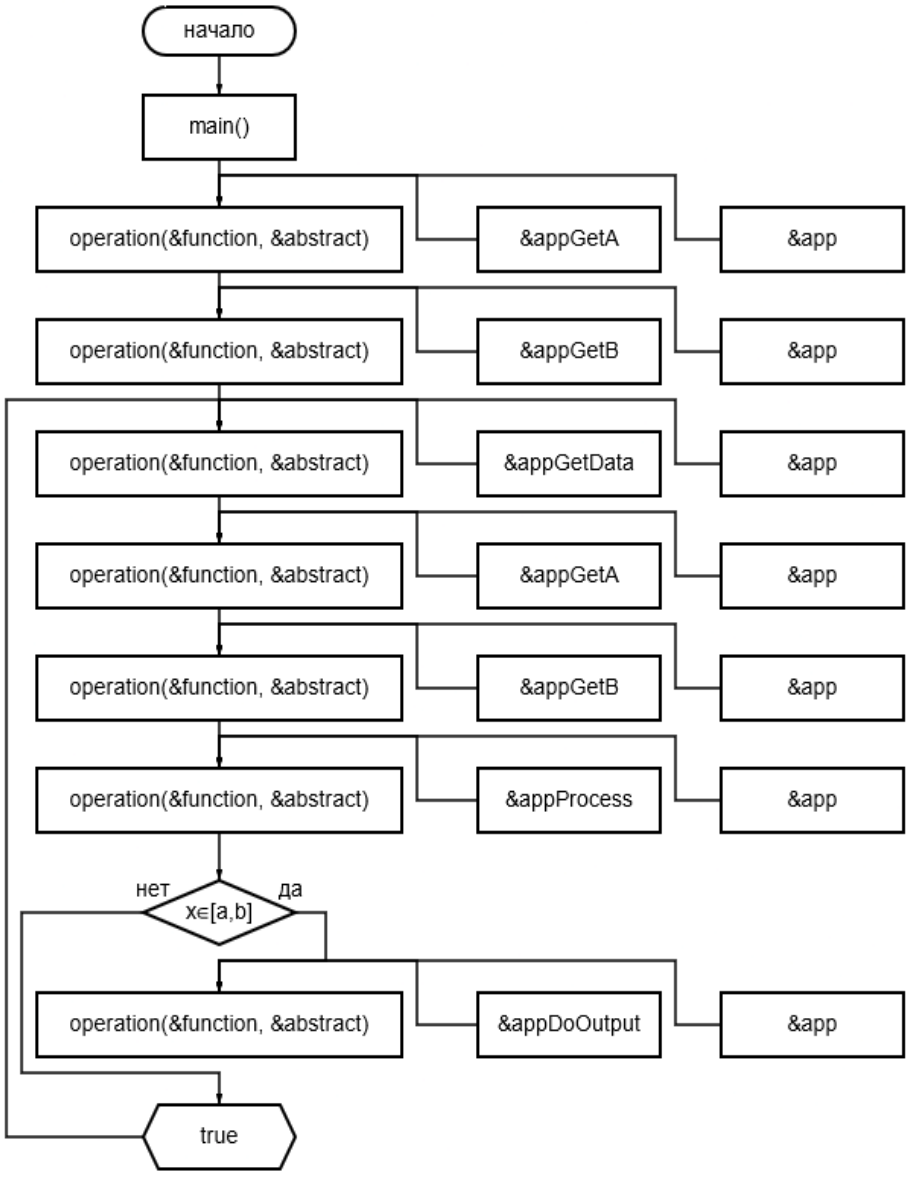
```
    }
    if (!appGetB(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
//use !std::cin.eof for testing
while(true) {
        if (!appGetData(app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }

        if (appProcess(app)) {
            appDoOutput(app);
        }
        ++app.iteration;
    }
    return 0;
}

int appGetA(Application &app) {
    if (app.va == INT_MAX && app.a0 == INT_MAX) {
        std::cin >> app.va >> app.a0;
        return true;
    }
    else{
        int calculated = app.va * app.iteration + app.a0;
        return calculated;
    }
}

int appGetB(Application &app) {
    if (app.vb == INT_MAX && app.b0 == INT_MAX) {
        std::cin >> app.vb >> app.b0;
        return true;
    }
    else{
        int calculated = app.vb * app.iteration + app.b0;
        return calculated;
    }
}

bool appGetData(Application &app) {
    std::cin >> app.cin_read;
    if(std::cin.fail()){
        return false;
```

```
    }
    return true;
}

bool appProcess(Application &app) {
    if(app.cin_read > appGetA( app) &&  app.cin_read < appGetB(app)){
      return true;
    }
    return false;
}

bool appDoOutput(Application &app) {
    std::cout << app.iteration << " - " << app.cin_read << std::endl;
    //DEBUG
    std::cout << appGetA(app) << " ÷ " << appGetB(app) << std::endl << std::endl;
    return true;
}
```

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. Но теперь, так как, функция, вызванная через `callback` должна возвращать bool значение, мы не можем в качестве возвращаемого значения функций `appGetA` и `appGetB`, указывать результат вычислений. Его нужно передавать через переменную в АТД. Что и было сделано. В `Application`, добавлены два целочисленных поля — `current_A`, и `current_B`.
Обновленная блок-схема представлена ниже.



2.2 Использование индуктивного вычислителя

По условию, вводится функция operation(), которая принимает два аргумента - ссылку на исполняемую функцию и ссылку на переменную АТД. Рассмотрим структуру функции operation.

В application.h:

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

typedef — псевдоним типа данных

bool — тип возвращаемого значения данных

(*Callback) — указатель имени функции

(void *abstract) — указатель типа аргумента, название

В application.cpp:

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

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

Приложение 1

MAIN.cpp

```
#include "application.h"
#include <iostream>

int main() {
    int ret = appRun();
    return ret;
}
```

APPLICATION.H

```
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <climits>

//Data for program to handle
struct Application {
    int iteration = 1;
    int cin_read;

    /*Since we can't invoke function for it return its value as int,
    now we store A/B as a distinct variables, rather than gaining
    A/B reading by straight invoking its corresponding function*/
    int current_A, current_B;

    int va = INT_MAX;
    int a0 = INT_MAX;
    int vb = INT_MAX;
    int b0 = INT_MAX;
};

typedef bool (*Callback)(void *abstract);
bool operation(Callback callback, void *abstract);

// To execute application

int appRun();
bool appGetData(void *abstract);
bool appGetA(void *abstract);
bool appGetB(void *abstract);
bool appProcess(void *abstract);
bool appDoOutput(void *abstract);
```

```
#endif //NNTU_APPLICATION_H
```

APPLICATION.CPP

```
#include "application.h"
#include <iostream>

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

bool appGetA(void *abstract) {
    Application &app = *(Application*) abstract;
    if (app.va == INT_MAX && app.a0 == INT_MAX) {
        std::cin >> app.va >> app.a0;
    }
    else {
        app.current_A = app.va * app.iteration + app.a0;
    }
    return true;
}

bool appGetB(void *abstract) {
    Application &app = *(Application*) abstract;
    if (app.vb == INT_MAX && app.b0 == INT_MAX) {
        std::cin >> app.vb >> app.b0;
    }
    else {
        app.current_B = app.vb * app.iteration + app.b0;
    }
    return true;
}

bool appGetData(void *abstract) {
    Application &app = *(Application*) abstract;
    std::cin >> app.cin_read;
    if(std::cin.fail()){
        return false;
    }
    return true;
}

bool appProcess(void *abstract) {
    Application &app = *(Application*) abstract;
    if(app.cin_read > app.current_A && app.cin_read < app.current_B){
      return true;
```

```
    }
    return false;
}

bool appDoOutput(void *abstract) {
    Application &app = *(Application*) abstract;
    std::cout << app.iteration << " - " << app.cin_read << std::endl;
    //DEBUG
    std::cout << app.current_A << " ÷ " << app.current_B << std::endl << std::endl;
    return true;
}

int appRun() {
    Application app;
    if (!operation(&appGetA, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    if (!operation(&appGetB, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    //use !std::cin.eof() for testing
    while(true) {
        if (!operation(&appGetData, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appGetA, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appGetB, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (operation(&appProcess, &app)) {
            operation(&appDoOutput, &app);
        }
        ++app.iteration;
    }
    return 0;
}
```