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 18)

Выполнил:

Студент гр. 22-Рз [redacted]

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. 6

application.h.....7

Приложение 1.....9

Контрольная работа №2.....11

2.1 Архитектура программной системы.....11

2.2 Использование индуктивного вычислителя.....12

Приложение 1.....14

Задание по варианту

Подсчет количества «равнинных участков». Равнинным участком считается группа соседних элементов (более K элементов) исходной последовательности с одинаковым значением. 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

Примем следующие сокращения:

x – текущее значение;

n – порядковый номер;

ti – кол-во элементов текущего равнинного участка;

tv – элемент текущего равнинного участка;

k – K из условия;

lp – предыдущее, максимальное количество элементов равнинного участка;

fi – выдаваемое пользователю, кол-во элементов максимального равнинного участка;

fv – выдаваемое пользователю, элемент максимального равнинного участка.

y – элемент выходной последовательности, состоит из трёх значений: $(n,\ fv,\ fi)$;

Отклик вычислителя q определим как $q=\{< y >\ \ ,\forall x_n\}$.

Указав используемые переменные, перейдём к рассмотрению базовых условий:

$r_1(n=0)$;

$r_2(x_n=tv)$;

$r_3(ti>lp \wedge ti>K)$;

На основе описанных, базовых, условий, можно сформировать предикаты, отметим, что если переменная не включена в результат работы предиката, то её значение остаётся без изменений:

$R_1(): r_1$ - для $n=0$;

Результат: $R_1(.) \rightarrow ti=0,\ tv=x,\ q=< y >,\ n=n+1$;

$R_2(): \neg r_1 \wedge r2$ - для $n>0 \wedge x=tv$;

Результат: $R_2(.) \rightarrow ti=ti+1,\ q=< y >,\ n=n+1$;

$R_3(): \neg r_1 \wedge \neg r_2$ - для $n>0 \wedge x \neq tv$;

Результат: $R_3(.) \rightarrow ti=1,\ tv=x,\ lp=fi,\ q=< y >,\ n=n+1$;

$R_4(): \neg r_1 \wedge r_2 \wedge r_3$ - для $n>0 \wedge x=tv \wedge ti>lp \wedge ti>K$;

Результат: $R_4(.) \rightarrow fi=ti;\ fv=tv,\ q=< y >,\ n=n+1$;

Приведённые условия являются предикатами, т.е. результатом каждого из условий является логическое значение истина/ложь. Точкой обозначен набор аргументов, необходимых для вычисления значения предиката.

Во время выполнения программы, значение счётчика *ti* изменяется, для соответствия поставленным задачам, что является индуктивным расширением.

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

Кроме того, выполняется условие $\forall x_n : R_4 \rightarrow R_1 \wedge R_2 \wedge R_3 \wedge R_4 = false$, то есть не допускается одновременное выполнение различных условий. И обратное утверждение, $\neg \forall x_n : R_4 \rightarrow R_1 \wedge R_2 \wedge R_3 \wedge R_4 = true$. Истинное значение предиката обозначает факт наступления связанного с ним события.

1.3. Архитектура программной реализации вычислителя.

Разберём используемый абстрактный тип данных.

В файле application.h определяется два АТД, рассмотрим их в порядке, представленном в коде программы.

1) АТД temp_data — два поля:

```
// 1-n, 2-x
std::pair <int, int> cin_read_current;
// 1-streak 2-x
std::pair <int, int> temp_counter;
```

Первое поле — текущий элемент, первое значение — счетчик n, второе — x, текущее значение, считанное из потока.

Второе поле — текущее плато, первое значение — счетчик кол-ва элементов плато, второе значение — само значение плато.

2) АТД Application — четыре поля:

```
temp_data temp_data;
//k
int constK;
//lp
int last_plato;
// 1-streak, 2-x
std::pair <int, int> final;
```

Первое поле — включение АТД temp_data в основной АТД Application.

Второе поле — считанное, из стандартного устройства ввода, число K.

Третье поле — значение кол-ва элементов предыдущего плато.

Четвертое — пара целочисленых, первое — кол-во элементов, максимального на текущий момент, плато. Второе — элемент текущего плато.

application.h

В заголовочном файле application.h объявляется функция int appRun(Application& app). Упомянутая функция отвечает за правильное исполнение приложения. Принимает данные с стандартного устройства ввода, обрабатывает, согласно приведённым в пункте 1.2 правилам, и выводит соответствующий результат на стандартное устройство вывода.

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

```
bool appInitializeK(Application &app);
bool appInitializeData(Application &app);
bool appProcessData(Application &app);
bool appGetOutput(Application &app);
```

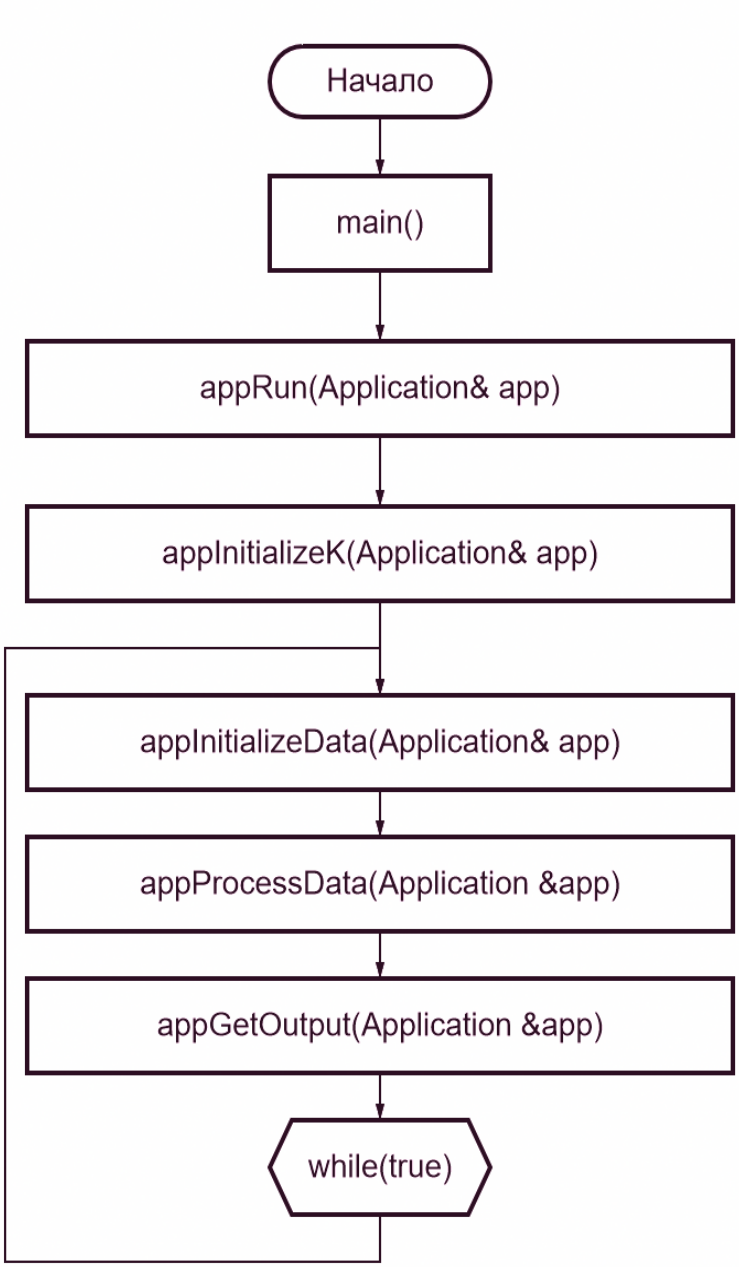
Рассмотрим каждую функцию в отдельности:

bool appInitializeK(Application &app); - Получение числа K;

bool appInitializeData(Application &app); - Получение очередного x;

bool appProcessData(Application &app); - Обработка x_n, запись результата плато на шаге n;

bool appGetOutput(Application &app); - выводит промежуточный результат на стандартное устройство вывода. Для упрощения восприятия, представим программу в качестве блок-схемы.


Начало
main()
appRun(Application& app)
appInitializeK(Application& app)
appInitializeData(Application& app)
appProcessData(Application &app)
appGetOutput(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 <utility>
struct temp_data {
    // 1-n, 2-x
    std::pair <int, int> cin_read_current;
    // 1-streak 2-x
    std::pair <int, int> temp_counter;
};

//Data for program to handle
struct Application {
    temp_data temp_data;
    //k
    int constK;
    //lp
    int last_plato;
    // 1-streak, 2-x
    std::pair <int, int> final;
};

int appRun(Application &app);
bool appInitializeK(Application &app);
bool appInitializeData(Application &app);
```

```
bool appProcessData(Application &app);
bool appGetOutput(Application &app);

#endif //NNTU_APPLICATION_H
```

Application.cpp

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

int appRun(Application &app) {
    if (!appInitializeK(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    while (!std::cin.eof()) {
        //Get Value from cin
        if (!appInitializeData(app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!appProcessData(app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!appGetOutput(app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
    }
    return 0;
}

bool appInitializeK(Application &app) {
    std::cin >> app.constK;
    //assign controlled value to i counter
    app.temp_data.cin_read_current.first = INT_MAX;
    if (std::cin.fail()) {
        return false;
    }
    return true;
}

bool appInitializeData(Application &app) {
    std::cin >> app.temp_data.cin_read_current.second;
    //assign 0 or ++ to counter, based on i
```

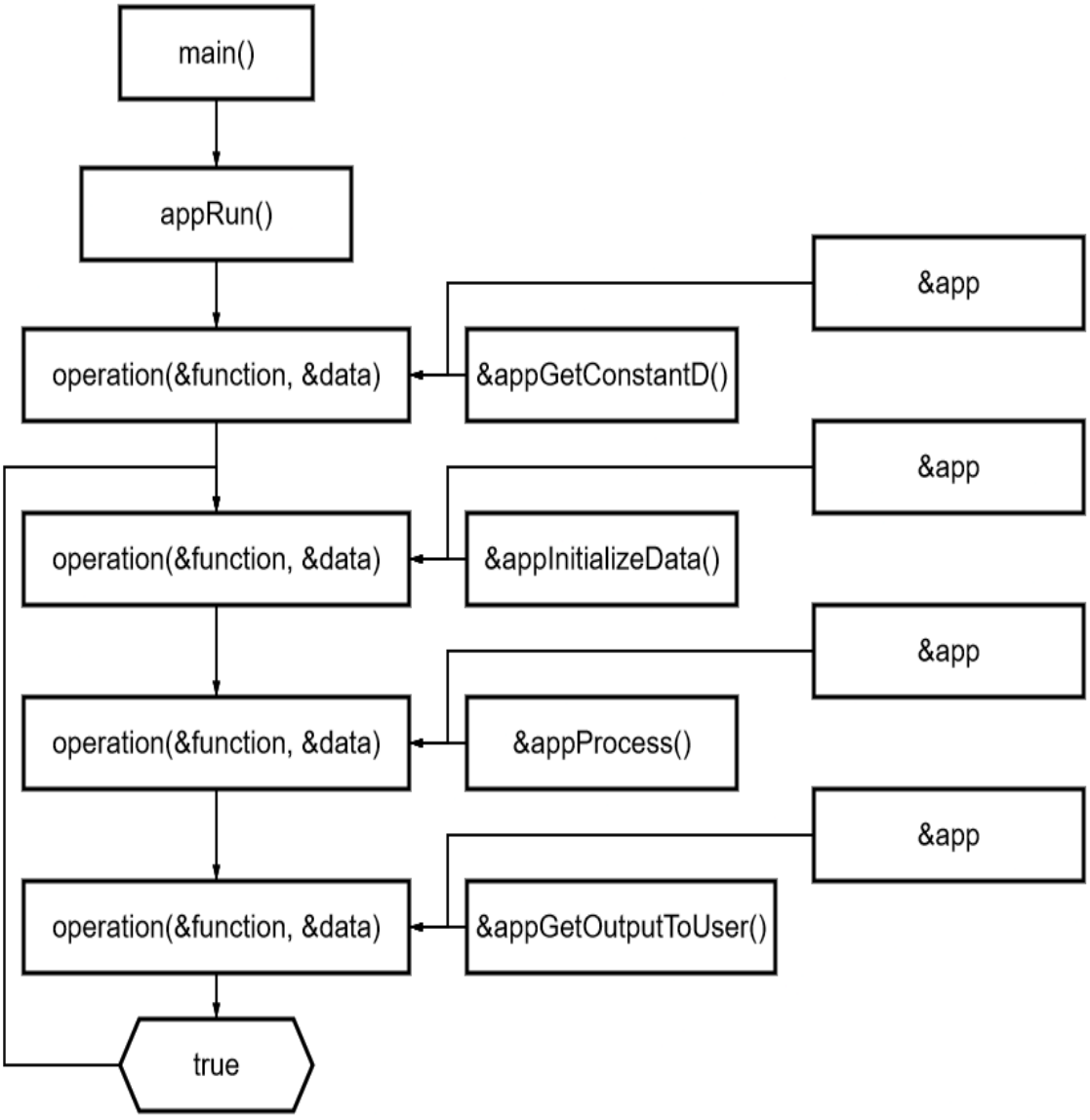
```
    if (app.temp_data.cin_read_current.first == INT_MAX) {
        app.temp_data.cin_read_current.first = 0;
    } else {
        ++app.temp_data.cin_read_current.first;
    }
    return true;
}

bool appProcessData(Application &app) {
    //check for first iteration, do special things for it
    if (app.temp_data.cin_read_current.first == 0) {
        ++app.temp_data.temp_counter.first;
        app.temp_data.temp_counter.second = app.temp_data.cin_read_current.second;
        return true;
    }
    //check for x_n == x_n-1
    if (app.temp_data.cin_read_current.second == app.temp_data.temp_counter.second) {
        ++app.temp_data.temp_counter.first;
    } else {
        app.temp_data.temp_counter.first = 1;
        app.temp_data.temp_counter.second = app.temp_data.cin_read_current.second;
        app.last_plato = app.final.first;
    }
    //check for current plato counter > K,
    // if so then assign current values to final to display
    if (app.temp_data.temp_counter.first > app.last_plato &&
app.temp_data.temp_counter.first > app.constK) {
        app.final.first = app.temp_data.temp_counter.first;
        app.final.second = app.temp_data.temp_counter.second;
    }
    return true;
}

bool appGetOutput(Application &app) {
    std::cout << app.temp_data.cin_read_current.first << " - Iteration: ";
    std::cout << app.final.first << " of " << app.final.second << 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.

В следствие внедрения данной методологии вызова функций, изменилась структура программы. Представлена ниже.



2.2 Использование индуктивного вычислителя

Запрошенная callback конструкция, может быть представлена в виде функции operation, которая будет брать два аргумента, ссылку на bool функцию, которую необходимо исполнить, и ссылку на АТД, т. е. Данные, которые operation передаст, как аргумент, для вызываемой функции. Опишем её:

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

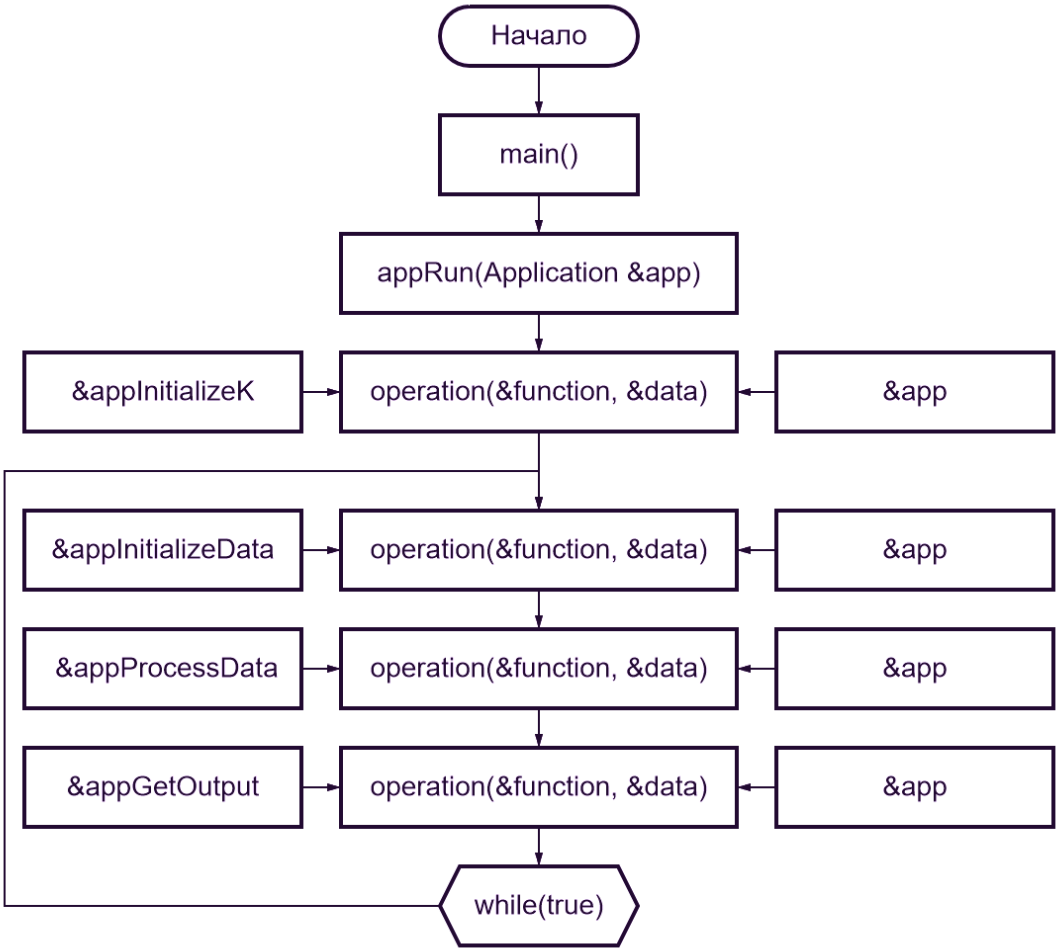
(void *data) — указатель типа аргумента, название

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.

В следствие изменения стиля вызова функций, структура программы, притерпела небольшие изменения, рассмотрим обновлённую блок-схему:



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 <utility>

struct temp_data {

    // 1-n, 2-value
    std::pair <int, int> cin_read_current;
    // 1-streak
    std::pair <int, int> temp_counter;

};
struct Application {

    temp_data temp_data;
    int constK;
    int last_plato;

    // 1-streak, 2-value
    std::pair <int, int> final;

};
```

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

int appRun();
bool appInitializeK(void *p_app);
bool appInitializeData(void *p_app);
bool appProcessData(void *p_app);
bool appGetOutput(void *p_app);

#endif //NNTU_APPLICATION_H
```

application.cpp

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

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

bool appInitializeK(void *p_app) {
    Application &app = *(Application *) p_app;
    std::cin >> app.constK;
    //assign controlled value to i counter
    app.temp_data.cin_read_current.first = INT_MAX;
    if (std::cin.fail()) {
        return false;
    }
    return true;
}

bool appInitializeData(void *p_app) {
    Application &app = *(Application *) p_app;
    std::cin >> app.temp_data.cin_read_current.second;
    //assign 0 or ++ to counter, based on i
    if (app.temp_data.cin_read_current.first == INT_MAX) {
        app.temp_data.cin_read_current.first = 0;
    } else {
        ++app.temp_data.cin_read_current.first;
    }
```

```
    return true;
}

bool appProcessData(void *p_app) {
    Application &app = *(Application *) p_app;
    //check for first iteration, do special things for it
    if (app.temp_data.cin_read_current.first == 0) {
        ++app.temp_data.temp_counter.first;
        app.temp_data.temp_counter.second =
app.temp_data.cin_read_current.second;
        return true;
    }
    //check for x_n == x_n-1
    if (app.temp_data.cin_read_current.second ==
app.temp_data.temp_counter.second) {
        ++app.temp_data.temp_counter.first;
    } else {
        app.temp_data.temp_counter.first = 1;
        app.temp_data.temp_counter.second =
app.temp_data.cin_read_current.second;
        app.last_plato = app.final.first;
    }
    //check for current plato counter > K,
    // if so then assign current values to final to display
    if (app.temp_data.temp_counter.first > app.last_plato &&
app.temp_data.temp_counter.first > app.constK) {
        app.final.first = app.temp_data.temp_counter.first;
        app.final.second = app.temp_data.temp_counter.second;
    }
    return true;
}

bool appGetOutput(void *p_app) {
    Application &app = *(Application *) p_app;
    std::cout << app.temp_data.cin_read_current.first << " - Iteration: ";
    std::cout << app.final.first << " of " << app.final.second << std::endl;
    return true;
}

int appRun() {
```

```
    Application app;
    if (!operation(&appInitializeK, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }
    while (!std::cin.eof()) {
        if (!operation(&appInitializeData, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appProcessData, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
        if (!operation(&appGetOutput, &app)) {
            std::cout << "DATA INPUT FAILURE." << std::endl;
            return 1;
        }
    }
    return 0;
}
```