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 10)**

Выполнил:

Студент гр. 22-Рз \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

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

Нижний Новгород,

2024 г.

Оглавление

[Задание по варианту 3](#__RefHeading___Toc2465_4070117755)

[Контрольная работа № 1. 4](#__RefHeading___Toc2035_2758904751)

[1.1. Постановка задачи 4](#__RefHeading___Toc2041_2758904751)

[1.2. Алгоритм индуктивной обработки 4](#__RefHeading___Toc11489_2758904751)

[1.3. Архитектура программной реализации вычислителя. 6](#__RefHeading___Toc8446_3110321970)

[АТД Application 6](#__RefHeading___Toc11491_2758904751)

[application.h 6](#__RefHeading___Toc11493_2758904751)

[Приложение 1. 8](#__RefHeading___Toc3283_4070117755)

[Контрольная работа №2 11](#__RefHeading___Toc2047_2758904751)

[2.1 Архитектура программной системы 11](#__RefHeading___Toc11495_2758904751)

[2.2 Использование индуктивного вычислителя 12](#__RefHeading___Toc11497_2758904751)

[Приложение 1 13](#__RefHeading___Toc11499_2758904751)

# Задание по варианту

Обработка заключается в суммировании всех поступающих элементов. Найти наибольшее значение суммы, не меньшее заданного значения . Тип элемента — действительное число.

# 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.

## 1.2. Алгоритм индуктивной обработки

Максимальную сумму обозначим как E. В программе присутствует счётчик итерации (далее ) который будет индуктивно расширяться при окончании одного цикла программы. Так же, для того, чтобы добавлять в сумму любые значения, даже отрицательные, предварительно суммируется к рабочей сумме (далее ). Если оказывается больше , то значение обновляется в соответствии с новым значением. – элемент выходной последовательности, имеет две вариации, и . , . Отклик вычислителя определим как .

В начале работы программы, переменные имеют следующие значения:

= 0, = 0, = 1, вводится пользователем, с стандартного устройства, при запуске программы.

Рассмотрим базовые условия:

;

;

Имея набор базовых условий, можно сформировать предикаты, отметим, что если переменная не приводится в результате выполнения предиката, то она остаётся **без изменений**. Перед началом проверки предикатов, текущее значение добавляется к , таким образом - .

- справедливо для ;

Результат: ;

- справедливо для ;

Результат: ;

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

## 1.3. Архитектура программной реализации вычислителя.

Разберём используемый абстрактный тип данных.

АТД Application

Определяется структурный тип данных **Application**, содержащий пять полей:

int i = 1;  
 int cin\_read;  
 int constS;  
 int finalSum = 0;

int tempSum = 0;

1 — cчётчик текущей итерации программы,

2 — хранилище текущего значения,

3 — введённое пользователем число ,

4 — общая сумма, ранее рассматривалась как ;

5 — «рабочая» сумма ;

application.h

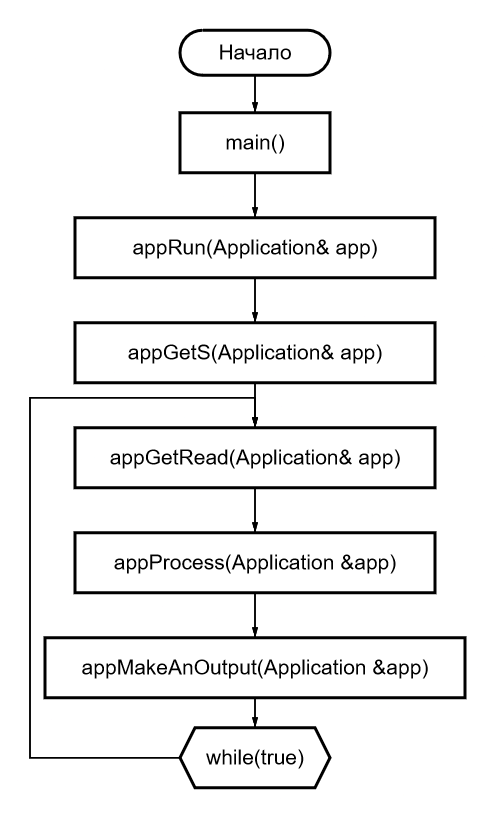
В заголовочном файле application.h объявляется глобальная функция int appRun(Application& app). Эта функция отвечает за исполнение программы, а соответственно, напрямую взаимодействует с АТД A*pplication,* а именно: получает данные от пользователя, обрабатывает их, и выводит результат в стандартное устройство вывода. При успешном выполнении возвращает значение , 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 appGetRead(Application &app);  
 bool appGetS(Application &app);  
 bool appProcess(Application &app);  
 bool appMakeAnOutput(Application &app);

Рассмотрим каждую функцию в отдельности:

bool appGetRead(Application &app);- отвечает получение очередного значения, от пользователя.  
 bool appGetS(Application &app); - получение числа .  
 bool appProcess(Application &app); - отвечает за добавление к рабочей сумме , и добавление к финальной сумме .  
 bool appMakeAnOutput(Application &app); - проверяет на S, выводит результат пользователю.



# Приложение 1.

Main.cpp

#include "application.h"  
#include <iostream>  
  
int main() {  
 std::cout << "Hello" << std::endl;  
 Application app;  
 int ret = appRun(app);  
 return ret;  
}

application.h

#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_Hstruct Application {  
 int i = 1;  
 int cin\_read;  
 int constS;  
 int finalSum = 0;

int tempSum = 0;  
};  
int appRun(Application& app);  
bool appGetRead(Application &app);  
bool appGetS(Application &app);  
bool appProcess(Application &app);  
bool appMakeAnOutput(Application &app);  
  
#endif *//NNTU\_APPLICATION\_H*

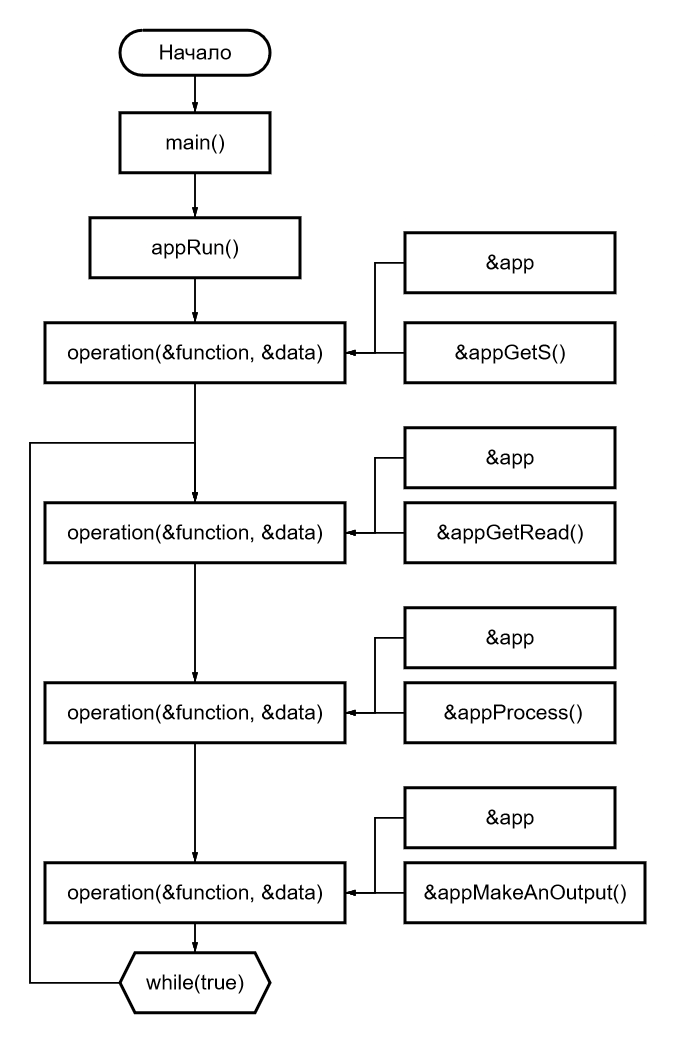
application.cpp

#include "application.h"  
#include <iostream>  
  
int appRun(Application &app) {  
 if (!appGetS(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 while(true) {  
 if (!appGetRead(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!appProcess(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!appMakeAnOutput(app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 ++app.i;  
 }  
 return 0;  
}  
  
bool appGetS(Application &app) {  
 std::cout << "Input an S constant:" << std::endl;  
 std::cin >> app.constS;  
 std::cout << app.constS << std::endl << std::endl;  
 return true;  
}  
  
bool appGetRead(Application &app) {  
 std::cin >> app.cin\_read;  
 if(std::cin.fail()){  
 return false;  
 }  
 return true;  
}  
  
bool appProcess(Application &app) {  
 app.tempSum += app.cin\_read;  
 if(app.tempSum > app.finalSum){  
 app.finalSum = app.tempSum;  
 }  
 return true;  
}  
  
bool appMakeAnOutput(Application &app) {  
 std::cout << app.i << " - ";  
 *//Output results* if (app.finalSum > app.constS) {  
 std::cout << app.finalSum << std::endl;  
 *//DEBUG ENTRY* std::cout << "DEBUG: " << app.tempSum << std::endl;  
 }  
 else {  
 std::cout << "No valid result, for now..." << std::endl;  
 }  
  
 return true;  
}

# Контрольная работа №2

**«Настройка индуктивного вычислителя с использованием функции обратного вызова»**

## 2.1 Архитектура программной системы

Алгоритм и логика выполнения пересчета, и выдачи результата, остался таким же, как в контрольной работе 1. Но изменился стиль вызова функций, и стиль передачи переменной АТД. Обновленная блок-схема представлена ниже:

## 2.2 Использование индуктивного вычислителя

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

В application.h:

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

typedef — псевдоним типа данных

bool — тип данных

(\*Callback) — указатель типа функции

(void \*data) — указатель типа аргумента, название

В application.cpp:

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

bool — тип функции

operation — название

Callback - тип первого аргумента

сallback — 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 << "Hello" << std::endl;  
 int ret = appRun();  
 return ret;  
}

Application.h

#ifndef NNTU\_APPLICATION\_H  
#define NNTU\_APPLICATION\_Hstruct Application {  
 int i = 1;  
 int cin\_read;  
 int constS;  
 int finalSum = 0;

int tempSum = 0;

};  
typedef bool (\*Callback)(void \*data);  
bool operation(Callback callback, void \*data);  
int appRun();  
bool appGetRead(void \*data);  
bool appGetS(void \*data);  
bool appProcess(void \*data);  
bool appMakeAnOutput(void \*data);  
  
#endif *//NNTU\_APPLICATION\_H*

Application.cpp

#include "application.h"  
#include <iostream>  
  
bool operation(Callback callback, void \*data) {  
 return (\*callback)(data);  
}

bool appGetS(void \*data) {  
 Application &app = \*(Application\*) data;  
 std::cout << "Input an S constant:" << std::endl;  
 std::cin >> app.constS;  
 std::cout << app.constS << std::endl << std::endl;  
 return true;  
}  
  
bool appGetRead(void \*data) {  
 Application &app = \*(Application\*) data;  
 std::cin >> app.cin\_read;  
 if(std::cin.fail()){  
 return false;  
 }  
 return true;  
}  
  
bool appProcess(void \*data) {  
 Application &app = \*(Application\*) data;  
 app.tempSum += app.cin\_read;  
 if(app.tempSum > app.finalSum){  
 app.finalSum = app.tempSum;  
 }  
 return true;  
}  
  
bool appMakeAnOutput(void \*data) {  
 Application &app = \*(Application\*) data;  
 std::cout << app.i << " - ";  
 *//Output results* if (app.finalSum > app.constS) {  
 std::cout << app.finalSum << std::endl;  
 *//DEBUG ENTRY  
 //std::cout << "DEBUG: " << app.tempSum << std::endl;* }  
 else {  
 std::cout << "No valid result, for now..." << std::endl;  
 }  
 return true;  
}

int appRun() {  
 Application app;  
 if (!operation(&appGetS, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 while(true) {  
 if (!operation(&appGetRead, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
  
 if (!operation(&appProcess, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 if (!operation(&appMakeAnOutput, &app)) {  
 std::cout << "DATA INPUT FAILURE." << std::endl;  
 return 1;  
 }  
 ++app.i;  
 }  
 return 0;  
}