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 [redacted]

Проверил:

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

Нижний Новгород,
2024 г.

Оглавление

Задание по варианту

Обработка заключается в суммировании всех поступающих элементов. Найти наибольшее значение суммы, не меньшее заданного значения S. 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 *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

Максимальную сумму обозначим как E. В программе присутствует счётчик итерации (далее *n*) который будет индуктивно расширяться при окончании одного цикла программы. Так же, для того, чтобы добавлять в сумму любые значения, даже отрицательные, x_n предварительно суммируется к рабочей сумме (далее *wE*). Если *wE* оказывается больше *E*, то значение *E* обновляется в соответствии с новым значением. y – элемент выходной последовательности, имеет две вариации, y_1 и y_2. $y_1=(n)$, $y_2=(n,\ E)$. Отклик вычислителя q определим как $q=\begin{Bmatrix} <y_1> & ,wE>E \\ <y_2> & ,E>S \end{Bmatrix}$.

В начале работы программы, переменные имеют следующие значения:

$E = 0$, $wE = 0$, $n = 1$, S вводится пользователем, с стандартного устройства, при запуске программы.

Рассмотрим базовые условия:

$r_1(wE > E)$;

$r_2(E > S)$;

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 x_n добавляется к wE, таким образом - $wE = wE + x_n$.

$R_1() : r_1$ - справедливо для $wE > E$;

Результат: $R_1(.) \rightarrow E = wE, q = < y_1 >, \ n = n + 1$;

$R_2() : r_1 \wedge r_2$ - справедливо для $E > S$;

Результат: $R_2(.) \rightarrow q = < y_2 >, \ n = n + 1$;

Условия, используемые в пересчёте E 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.

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 — счётчик текущей итерации программы,

2 — хранилище текущего значения,

3 — введённое пользователем число *S*,

4 — общая сумма, ранее рассматривалась как *E*;

5 — «рабочая» сумма *wE*;

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 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 appGetRead(Application &app);
bool appGetS(Application &app);
bool appProcess(Application &app);
bool appMakeAnOutput(Application &app);
```

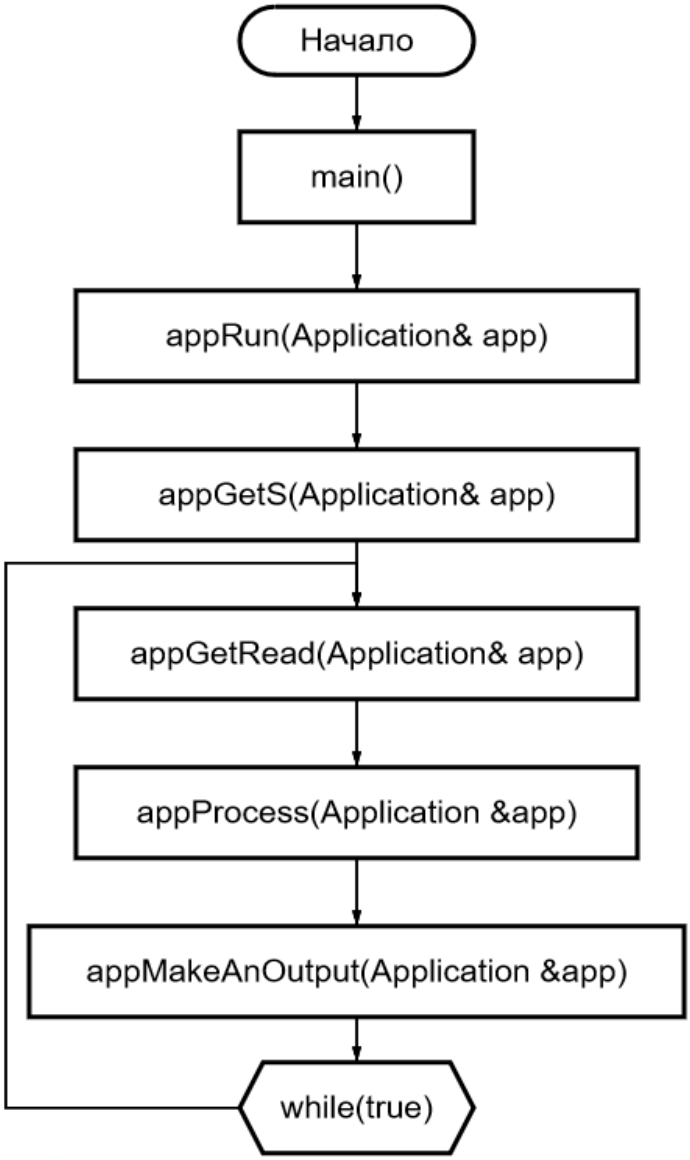
Рассмотрим каждую функцию в отдельности:

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

`bool appGetS(Application &app);` - получение числа *S*.

`bool appProcess(Application &app);` - отвечает за добавление к рабочей сумме *wE*, и добавление к финальной сумме *E*.

`bool appMakeAnOutput(Application &app);` - проверяет на S, выводит результат пользователю.


Начало
main()
appRun(Application& app)
appGetS(Application& app)
appGetRead(Application& app)
appProcess(Application &app)
appMakeAnOutput(Application &app)
while(true)

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_H

struct 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;
```

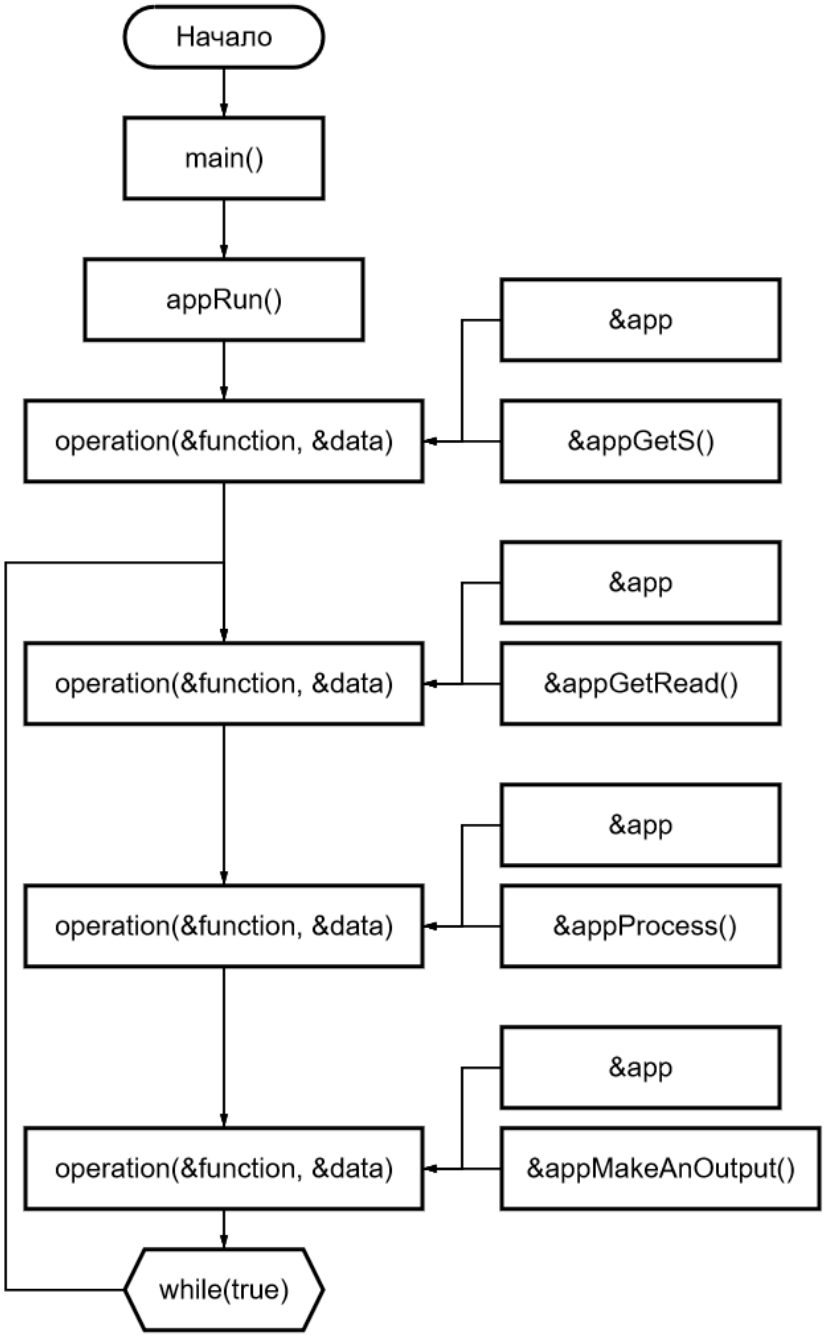
```
        //DEBUG ENTRY
        std::cout << "DEBUG: " << app.tempSum << std::endl;
    }
    else {
        std::cout << "No valid result, for now..." << std::endl;
    }

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

Application.h

```
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

struct 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;
}
```