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

Методические указания к контрольным работам по дисциплине
«Алгоритмы и методы организации программных систем»
для студентов направления подготовки бакалавра
11.03.01 «Радиотехника» заочной формы обучения

Нижний Новгород
2024

Составитель С.Б. Сидоров

УДК 519.6

Индуктивные операции и функции высших порядков: Метод. указания к контрольным работам по дисциплине «Алгоритмы и методы организации программных систем» для студентов направления подготовки бакалавра 11.03.01 «Радиотехника» заочной формы обучения / НГТУ; Сост.: С.Б. Сидоров. Н. Новгород, 2024.– 18 с.

Изложены краткие сведения о методах разработки алгоритмов выполнения индуктивных операций и применении функций высшего порядка в программных системах. Приведены схемы построения индуктивных расширений. Рассмотрены вопросы организации программных компонент при использовании функций обратного вызова. Приведена общая постановка задач, определен порядок выполнения контрольных работ, указаны требования к отчету, а также сформулированы вопросы для самопроверки.

Редактор

Подп. к печ._____. Формат 60x84 1/16 . Бумага_____. Печать офсетная.
Печ. л. 0,9 . Уч.-изд.л._____. Тираж_____экз. Заказ_____.

Нижегородский государственный технический университет им. Р.Е. Алексеева.

Типография НГТУ.
Адрес университета и полиграфического предприятия:
603950, ГСП-41, г. Нижний Новгород, ул. Минина, 24.

1 Контрольная работа №1 «Реализация индуктивной обработки последовательности элементов»

1.1 Цель работы

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.

1.2 Краткие сведения

1.2.1 Формулировка задачи

На вход системы последовательно и неограниченно во времени поступают элементы x_i, где i — порядковый номер элемента, начиная с 1. Все элементы имеют одну и ту же природу, то есть являются однотипными. Над набором элементов $\boldsymbol{X}^{(n)}=\langle x_1, x_2, x_3, \ldots, x_n\rangle$ выполняется некоторая операция $f(\boldsymbol{X}^{(n)})$. Результатом ее выполнения является набор выходных значений $\boldsymbol{Y}^{(n)}=\langle y_1^{(n)}, y_2^{(n)}, y_3^{(n)}, \ldots, y_{k(n)}^{(n)}\rangle$. Все элементы выходного набора также имеют одну и ту же природу, то есть являются однотипными. При этом типы элементов на входе и выходе системы в общем случае отличаются.

Указанная операция выполняется каждый раз при поступлении очередного элемента x_{n+1} для набора $\boldsymbol{X}^{(n+1)}$.

1.2.2 Классы операций

Все операции над набором данных подразделяются на два класса:

- вычисление значения характеристики последовательности;
- порождение элементов последовательности

Для класса вычисления значения характеристики количество элементов в выходном наборе $k(n)\equiv 1$. Эта характеристика может быть скалярной величиной, может состоять из набора элементов разной природы или представлять собой набор однотипных значений. Существенным является конечный размер выходного элемента, не зависящий от n.

Для этого класса можно определить выходную последовательность $\boldsymbol{Y}'^{(n)}=\langle y_1^{(1)}, y_1^{(2)}, y_1^{(3)}, \ldots, y_1^{(n)}\rangle$, расширяющуюся во времени по мере поступления на вход системы обработки очередных входных элементов x_n.

Отличием операций, относящихся к другому классу является неограниченный рост количества элементов $k(n)$ в выходном наборе $\boldsymbol{Y}^{(n)}$ так, что $\nexists K : \forall n \Rightarrow k(n) < K$.

Для данного класса в качестве выходной последовательности $\boldsymbol{Y}'^{(n)}$ можно рассмотреть $\boldsymbol{Y}^{(n)}$, то есть $\boldsymbol{Y}'^{(n)} = \boldsymbol{Y}^{(n)}$, но только при условии что $\forall n \Rightarrow Y^{(n+1)} = Y^{(n)} * q^{(n+1)}$, где $q^{(n+1)} = \langle y_{k(n)+1}^{(n+1)}, \ldots, y_{k(n+1)}^{(n+1)} \rangle$ и $*$ — операция присоединения.

В этом случае выходная последовательность $\boldsymbol{Y}'^{(n)} = \Delta * q^{(1)} * q^{(2)} * \ldots * q^{(n)}$. Заметим, что в данных обозначениях для первого класса операций $\forall k : q^{(k)} = \langle y_1^{(k)} \rangle$.

1.2.3 Индуктивная функция

Функция $f : \Omega(x) \to \Omega(Y)$ называется индуктивной, если $f(\omega * x) = f(\omega) * q(\omega * x)$, причем $q(\omega * x) = F'_q(S(\omega * x), x)$, а $S(\omega * x)$ можно вычислить, зная $S(\omega)$ и x, т. е. если для нее существует функция $F_S(y, x)$ такая, что

$$S(\omega * x) = F_S(S(\omega), x) . \qquad (1)$$

Для последовательности ω с количеством элементов (длиной) n введем обозначения $S^{(n)} = S(\omega)$ и $q^{(n)} = q(\omega)$. С учетом этого:

$$\begin{aligned} &S^{(n+1)} = F_S(S^{(n)}, x) , \\ &q^{(n+1)} = F'_q(S^{(n+1)}, x) = F'_q(F_S(S^{(n)}, x)) = F_q(S^{(n)}, x) . \end{aligned} \qquad (2)$$

В этом случае вектор-функцию $F(s, x) = \langle F_q(s, x), F_S(s, x) \rangle$ назовем функцией перевычисления. Она позволяет рассчитать новые значения $q^{(n+1)}$ и $S^{(n+1)}$ для расширенной последовательности, используя текущее значение $S^{(n)}$ и очередной поступивший элемент x входной последовательности. Предполагается, что $S(\Delta) = S^{(0)}$ известно.

Величину S можно понимать как состояние вычислителя. Для выполнения обработки элементов входной последовательности необходимо использовать некоторые вспомогательные величины. Они будут рассчитываться каждый раз при поступлении очередного элемента последовательности. Совокупность таких величин примем в качестве рассматриваемого S, т. е. $S = \langle s_1, s_2, \ldots, s_k \rangle$.

Для индуктивных функций можно получить в общем виде простой однопроходный алгоритм обработки последовательно поступающих данных:

```
вычислить начальное S = f(Δ)
пока есть непрочитанные элементы
нц
```

получить очередное значение x
вычислить $\langle q, S\rangle = \langle F_q(S,x), F_S(S,x)\rangle$
выдать q
кц

При этом расчет нового значения S и формирование выхода q выполняются одновременно, на основе текущего состояния S. Учитывая (2), алгоритм можно записать следующим образом:

вычислить начальное $S = f(\Delta)$
пока есть непрочитанные элементы
нц
получить очередное значение x
вычислить $S = F_S(S,x)$
вычислить $q = F'_q(S,x)$
выдать q
кц

В этом случае при вычислении q используется обновленное состояние S. Выбор варианта алгоритма пересчета определяется конкретной задачей, исходя из критерия более простой и понятной формы записи правила формирования выхода вычислителя.

Примеры индуктивных функций:

сумма элементов последовательности

$S = \langle s\rangle$,
$s^{(0)} = 0$,
$s = s + x$,
$q = F'_q(S,x) = \langle s\rangle$.

длина последовательности

$S = \langle n\rangle$,
$n^{(0)} = 0$,
$n = n + 1$,
$q = F'_q(S,x) = \langle n\rangle$.

последний элемент

$S = \langle v\rangle$,
$v^{(0)}$ — любое,
$v = x$,
$q = F'_q(S,x) = \langle v\rangle$.

1.2.4 Индуктивное расширение

Подавляющее большинство полезных функций не являются индуктивными, поскольку в записи их функции перевычисления в правой части, кроме s и x присутствуют другие переменные величины. Однако такие функции могут быть приведены к классу индуктивных посредством рассмотрения их индуктивного расширения.

Пусть для целевой задачи получено некоторое правило формирования выхода вычислителя, как отклика на поступление одного очередного элемента входной последовательности. Пусть также в записи этого правила присутствуют величины, отличные от входного элемента и составных частей описания T_y. Совокупность таких величин обозначим как $\vec{v}=(v_1, v_2, \ldots v_r)$. Рассмотрим новую операцию $f_z(x)$, результатом которой является набор, состоящий из элементов $z=(y, \vec{v})$ типа T_z. Новый тип является составным типом и включает в себя элемент типа T_y и набор величин $\vec{v}$, т. е. $T_z=(T_y, \vec{v})$. Операцию $f_z(x)$ назовем **индуктивным расширением** операции $f(x)$, если она удовлетворяет определению (1). Для нее выходная последовательность $Z^{'(n)}=\Delta * q_z^{(1)} * q_z^{(2)} * \ldots * q_z^{(n)}$.

Здесь $q_z^{(n+1)}=\langle z_{k(n)+1}^{(n+1)}, \ldots z_{k(n+1)}^{(n+1)} \rangle$ — отклик вычислителя, выполняющего операцию $f_z(x)$, на поступление очередного элемента x_{n+1} входной последовательности *X*. Для полученного индуктивного расширения $S_z(\omega * x)=\Phi_S(S_z(\omega), x)$, а сама $\Phi_S()$ вместе с $\Phi_q(S_z, x)$ составляет новую функцию перевычисления $\Phi(z, x)$.

Поскольку исходная задача состоит в выполнении операции $f(x)$, то для получения результата ее выполнения из элементов выходной последовательности *Z* необходимо выделять составляющую *y*.

С практической точки зрения при рассмотрении индуктивного расширения, компоненты $\vec{v}$ добавляются в исходное состояние *S* и приводятся правила их пересчета, а также указываются начальные значения.

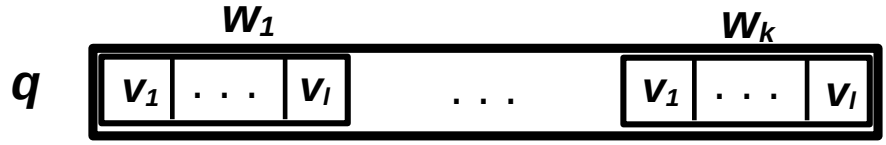
Если в правилах пересчета $\vec{v}$ будут присутствовать новые необходимые величины, то следует применить еще раз аналогичную схему, но уже для операции $f_z(x)$. Таким образом, переход к индуктивной операции в общем случае является итерационной процедурой.

1.2.5 Математические формы записи функций перевычисления

Применение схемы индуктивного вычислителя основывается на математической форме записи индуктивной операции. Рассмотрим некоторые способы представления функции перевычисления. Введем ряд обозначений.

Не уменьшая общности, элемент входной последовательности $x=(\chi_1, \ldots, \chi_m)$. Элементы набора χ_i могут быть разнотипными, и иметь достаточно произвольную природу. На отдельные части будем ссылаться в виде χ_i без указания их явной принадлежности к *x*. Элемент выходной последовательности $y=(v_1, \ldots, v_l)$ и на отдельные части описания

выходного элемента будем ссылаться в виде v_i. Отклик вычислителя на поступивший элемент обозначим как $q=\langle w_1,\ldots,w_k\rangle$. Отклик является упорядоченным набором, фактически последовательностью длины *k*. Все w_i имеют одну природу, а именно, элементами *y*. Структуру отклика вычислителя можно изобразить в следующем виде:



Индуктивное расширение вычислителя $ext=(e_1,\ldots e_r)$. Элементы набора могут быть разнотипными и иметь достаточно произвольную природу. На e_i будем ссылаться без указания их явной принадлежности к *ext*.

В записи функции перевычисления указываются правила пересчета для каждой из величин $v_{i,w}$ и e_j.В общем случае вид формулы, по которой вычисляется новое значение величины, определяется набором условий $R=(R_1(.),R_2(.),\ldots,R_n(.))$. Каждое условие в наборе является предикатом, то есть функция, результатом которой является истина или ложь. Точкой обозначен набор аргументов, необходимых для вычисления значения предиката. В качестве аргументов можно использовать элементы из *x*, *ext*, *y*. В последнем случае, если количество элементов в *q* возможно более одного, необходимо уточнение к какому элементу *w* относится v_i.

Каждому виду условия соответствует своя функция расчета нового значения элемента перевычисляемой величины. У таких функций набор аргументов формируется по тем же правилам, что и для предикатов. Например, правило пересчета v_i можно записать в следующем виде:

$$v_i=\begin{cases} f_1(.), если\ R_1(.) \\ f_2(.), если\ R_2(.) \\ \cdots \\ f_n(.), если\ R_n(.) \end{cases}$$

Условия, используемые в правиле пересчета некоторой величины, должны охватывать все возможные ситуации, то есть $R_1\cup R_2\cup\ldots\cup R_n=true$. Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчета величины. Кроме того должно быть выполнено $\forall l,s: l\neq s\Rightarrow R_l\cap R_s=false$, то есть не допускается одновременного выполнения различных условий. В противном случае присутствует противоречие с точки зрения правила пересчета величины.

В ряде случаев правило пересчета величин удобно представить в другой форме. Рассматриваются всевозможные события, в связи с

поступлением очередного элемента на вход вычислителя, существенные с точки зрения выполняемой вычислителем операции. Эти события должны быть взаимоисключающими, но одно из них обязательно должно иметь место. Описание каждого события представляется в виде предиката $R_i(.)$. Истинное значение предиката обозначает факт наступления связанного с ним события. Для каждого события приводятся правила пересчета всех величин в следующем виде:

$$\text{если } R_k(.) \Rightarrow \begin{cases} q = \langle \ldots \rangle \\ v_{i,w} = f_{i,w}(.) \\ \cdots \\ e_j = g_j(.) \end{cases}$$

1.2.6 Архитектура программной компоненты вычисления индуктивной функции

В области программной реализации вычислитель индуктивной функции определяется набором следующих программных компонент:

- тип элемента входной последовательности (`OperationInputItem`);
- тип элемента выходной последовательности (`OperationOutputItem`);
- тип результата обработки **q** (`OperationOutput`);
- тип, описывающий набор параметров обработки (`OperationParameters`);
- тип, описывающий индуктивное расширение операции (`OperationExtension`);
- абстрактный тип данных индуктивной операции (`Operation`).

Поскольку природа элементов входной и выходной последовательности является исходной информацией для решаемой задачи, определение типа каждого из них необходимо поместить в свой программный модуль, представленный заголовочным файлом, например `input_item.h` и `output_item.h` соответственно.

Индуктивную операцию удобно представить в виде абстрактного типа данных в отдельном программном модуле. Поскольку типы описания результата обработки, набора параметров и индуктивного расширения являются продуктом решения задачи, то есть частью разрабатываемого вычислителя, то их определение помещается в заголовочный файл модуля абстрактного типа данных.

Определение начального значения результата обработки для пустой входной последовательности реализуется в функции инициализации этого типа. Центральное место занимает функция обработки с двумя аргументами — значением очередного элемента входной последовательности и переменной, куда необходимо поместить результат обработки **q**.

1.3 Общая постановка задачи

На вход системы последовательно и неограниченно во времени поступают элементы x_i, где i — порядковый номер элемента, начиная с 1. Реализовать указанную в варианте обработку $f(\boldsymbol{X})$ последовательности элементов $\boldsymbol{X}=\langle x_1, x_2, x_3, \ldots \rangle$ с использованием схемы индуктивной обработки на пространстве последовательностей. Полученный набор выходных значений $\boldsymbol{Y}=\langle y_1, y_2, y_3, \ldots \rangle$ 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.

1.4 Порядок выполнения

1. Из постановки задачи получите математическое описание элемента входной последовательности данных и описание элемента выходной последовательности.
2. Разработайте алгоритм индуктивной обработки данных и запишите его в математической форме в соответствии с рекомендациями раздела 1.2.5.
3. Подготовьте тесты для решаемой задачи, охватывающие как можно больше различных сценариев поступления данных и проверьте на них правильность полученного математического решения. При выявленных ошибках выполните доработку алгоритма. Данный этап необходимо выполнять до успешного прохождения всех подготовленных тестов.
4. Определите структурные типы данных для представления в программе элемента входной последовательности и элемента выходной последовательности в соответствии с рекомендациями раздела 1.2.6.
5. Определите абстрактный тип данных для представления индуктивного вычислителя вместе с вспомогательными структурными типами данных в соответствии с рекомендациями раздела 1.2.6.
6. Получите реализацию функций абстрактного типа данных.
7. Получите реализацию тестовой программы для проверки ранее подготовленных тестов.
8. Проведите тестирование полученной программной реализации и, при необходимости, выполните её отладку для устранения выявленных на этапе тестирования ошибок. Данный этап необходимо выполнять до успешного прохождения всех подготовленных тестов.
9. Напишите отчет о выполнении контрольной работы в соответствии с рекомендациями раздела 1.5.
10. В целях самоконтроля ответьте на вопросы раздела 1.6.

1.5 Структура отчета

Отчет о выполнении контрольной работы должен содержать **титульный лист** с названием работы, фамилиями студента и преподавателя, а также названием университета, города и года выполнения работы.

Первым разделом является «**Постановка задачи»**, в котором полностью приводится как общая формулировка задачи, так и указанный преподавателем вариант, перечень исходных данных и требуемых результатов, а также контрольные примеры для уточнения постановки задачи и проверки правильности ее решения.

В следующем разделе «**Алгоритм индуктивной обработки**» приводится математическая запись алгоритма обработки разработанного в соответствии с постановкой задачи. Указывается найденное индуктивное расширение с пояснением каждого его элемента. Приводятся начальные значения всех компонент индуктивного вычислителя с пояснением принятого решения.

В разделе «**Архитектура программной реализации вычислителя**» приводится описание абстрактного типа данных (АТД), используемого для представления индуктивного вычислителя, предваряемое описанием вспомогательных типов данных. Описание абстрактного типа данных должно включать определение структурного типа данных с пояснением назначения каждого элемента структуры, а также перечень функций, реализующих операции АТД.

В последнем разделе «**Приложение**» приводится исходный текст программной реализации индуктивного вычислителя и основной программы, демонстрирующей его применение в соответствии с общей постановкой задачи.

1.6 Контрольные вопросы

1. Какие операции называются индуктивными?
2. Какая модель поступления данных на вход блока обработки рассматривается применительно к индуктивным операциям?
3. Какие классы операций над последовательностью данных выделяют? Приведите примеры операций разных классов.
4. Поясните специфику выходной последовательности для операции, являющейся индуктивной.
5. Какие вычислительные проблемы можно решить, при использовании метода индуктивной обработки последовательности данных?
6. Дайте определение индуктивной функции и приведите примеры индуктивных функций и не являющихся таковыми.

7. Что такое функция перевычисления?
8. Поясните понятие индуктивного расширения.
9. Какова причина необходимости задания начального значения компонентам индуктивного вычислителя?
10. Перечислите составные части индуктивного вычислителя.
11. Какая структура элемента входной последовательности данных выбрана Вами для разработанного индуктивного вычислителя?
12. Какая структура элемента выходной (результирующей) последовательности данных выбрана Вами для разработанного индуктивного вычислителя?
13. Поясните определение структуры параметров обработки реализованного индуктивного вычислителя.
14. Детально объясните определение структурного типа данных для представления индуктивного расширения с точки зрения его соответствия математического представления индуктивного вычислителя .
15. Поясните разработанный Вами алгоритм обработки данных на основе математической записи.
16. Дайте подробное объяснение программной реализации функции обработки элемента входной последовательности.
17. Опишите реализованную схему тестовой программы.

2 Контрольная работа №2 «Настройка индуктивного вычислителя с использованием функции обратного вызова»

2.1 Цель работы

Знакомство с понятием функции высшего порядка. Изучение и практическое применение схем представления функции обратного вызова и их использования в задачах.

2.2 Краткие сведения

2.2.1 Понятие функции высшего порядка

Функцией высшего порядка называется функция, у которой хотя бы один из аргументов является указателем на функцию. Сигнатура функции имеет следующую структуру:

```
тип operationA (…, тип2 (*F)(описание аргументов), ... );
```

В целях упрощения восприятия описания функции высшего порядка часто используется другой способ указания ее сигнатуры, при котором предварительно определяется специальный тип данных для описания функции, передаваемой в качестве аргумента в функцию высшего порядка:

```
typedef тип2 (*operationF)(описание аргументов);
```

В этом случае сигнатура функции высшего порядка может быть представлена в следующем виде:

```
тип operationA (…, OperationF f, ... );
```

Использование функций высшего порядка упрощает повторное использование программной реализации алгоритма, за который ответственна эта функция, в других программных системах. При этом программная реализация функции становится более универсальной и легко адаптируется под конкретные требования путем ее настройки через аргумент в виде указателя на функцию.

2.2.2 Функции обратного вызова

При выполнении программной системы типовой ситуацией является обращение с запросом некоторого вида одной программной компоненты к другой компоненте программной системы. В ряде случаев исполнение запроса может потребовать дополнительной коммуникации между вызывающей и вызываемой компонентами.

Например, вызываемой компоненте для выполнения действия может потребоваться некоторая помощь со стороны вызывающей компоненты, такая как предоставление дополнительной информации или проведение некоторой обработки данных. Другим примером является ситуация, когда в процессе выполнения вызываемой компоненты происходят события определенного вида и вызывающей компоненте отправляются уведомления о данных событиях.

В обоих приведенных ситуациях реализация вызываемой компоненты (`Operation`) с учетом возможности ее повторного использования не должна привязываться к конкретной вызывающей компоненте (`Caller`). Обеспечить указанное требование можно с помощью функции, адрес которой передается в вызываемую компоненту. Такие вспомогательные функции, являющиеся коммуникативными элементами, называются функциями обратного вызова (`callback`). Их коммуникативная роль наглядно иллюстрируется на приведенном ниже рисунке.

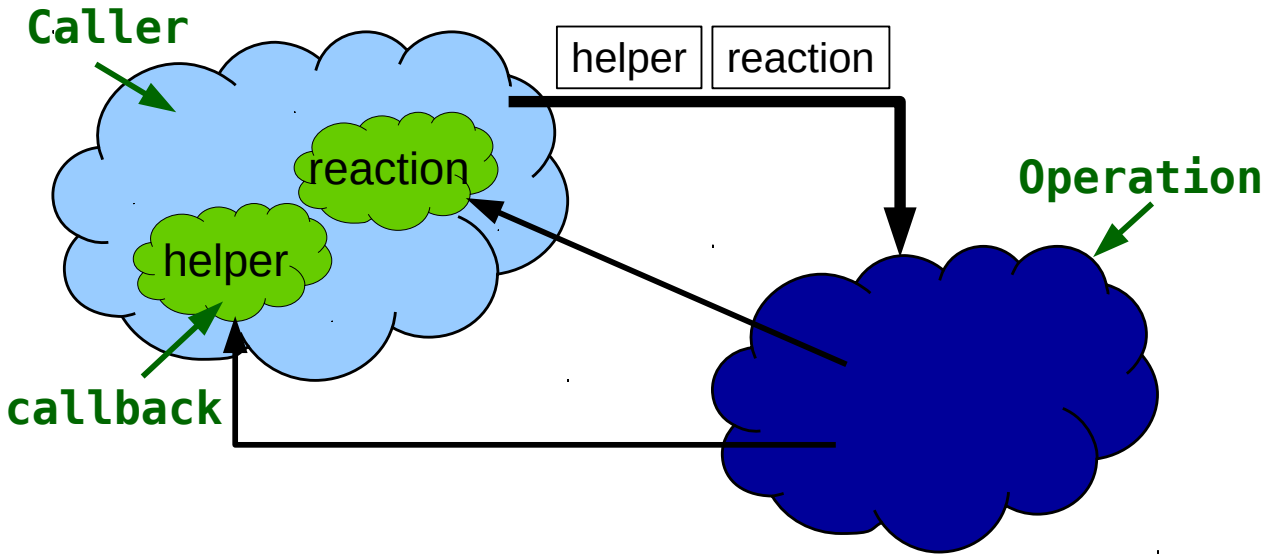


Рисунок: Взаимодействия компонент через функции обратного вызова

Очевидно, что в этом случае интерфейс вызываемой компоненты с точки зрения вызывающей компоненты является ни чем иным, как функцией высшего порядка.

2.2.3 Общая схема метода

Внедрение в программную систему функции обратного вызова, как коммуникативного элемента между вызывающей и вызываемой компонентами, состоит в определении следующих программных элементов:

- описание типа вызывающей компоненты;
- реализация функции обратного вызова, как части вызывающей компоненты;

- определение типа данных для представления функции обратного вызова;
- реализация вызываемой компоненты;
- использование вызываемой компоненты

Для описания вызывающей компоненты обычно используется структурный тип данных, в который в виде набора отдельных полей включается информация, необходимая для выполнения функции обратного вызова:

```
struct Caller {
     //описание необходимых свойств
}
```

В сигнатуре функции обратного вызова обязательно присутствует как минимум один аргумент, посредством которого в функцию передается информация о вызывающей компоненте. Поскольку вызываемая компонента не должна владеть сведениями о точном типе того, кто ее вызвал, то эта информация обычно представляется не типизированным указателем. Кроме одного обязательного аргумента, функция обратного вызова может принимать дополнительные аргументы, состав которых определяется задачей, в контексте решения которой будет применяться эта функция. С другой стороны, функция обратного вызова, как часть вызывающей компоненты, владеет информацией о природе (типе) этой компоненты. Вследствие отмеченного, не типизированный указатель можно безопасно преобразовать к указателю на вызывающую компоненту и тем самым обеспечить доступ из функции к значению свойств этой компоненты. С учетом сказанного выше общая схема определения функции обратного вызова примет следующий вид:

```
void callbackFunction(void* clientData, . . . ) {
   Caller& client = *(Caller*)clientData;
   //действия
   return;
}
```

Для удобства описания вызываемой компоненты и, что не менее важно, его понимания определяется тип данных для представления функции обратного вызова. Определение типа основывается на сигнатуре функции обратного вызова и имеет следующий вид:

```
typedef void (*Callback)(void* clientData, . . . );
```

В приведенной общей схеме многоточием обозначено описание дополнительных аргументов функции обратного вызова.

Вызываемая компонента является функцией высшего порядка, так как, помимо всего прочего, принимает информацию о функции обратного вызова и какие данные ей необходимо передать при обращении. Как

правило, соответствующие аргументы в сигнатуре вызываемой компоненты приводятся в конце списка. Тем самым подчеркивается, что они имеют смысл дополнительной информации, позволяющей адаптировать исполнение алгоритма под задачу вызывающей компоненты. С учетом отмеченного выше общая схема определения функции, реализующей вызываемую компоненту, примет следующий вид:

```
тип operation(..., Callback callback, void* clientData) {
   //. . .
   (*callback)(clientData, . . .)
   //. . .
}
```

Вызывающая компонента при обращении к вызываемой должна сообщить ей информацию о функции обратного вызова и значении аргумента, который необходимо будет передать в функцию обратного вызова. Это значение, как правило, является адресом переменной, описывающей саму вызывающую компоненту. Указанная переменная имеет рассмотренный ранее тип данных, а ее значение должно быть сформировано некоторым способом до обращения к вызываемой компоненте. Таким образом, использование вызываемой компоненты ревализуется в соответствии со следующей общей схемой:

```
//. . .
Caller caller;
//занесение информации в caller
//. . .
operation(..., callbackFunction, &caller);
//. . .
```

2.3 Общая постановка задачи

Реализовать процедуру использования индуктивного вычислителя из задания к контрольной работе №1 в виде функции высшего порядка. При этом операции получения очередного элемента входной последовательности и выдачи очередного элемента результирующей последовательности представить в виде функций обратного вызова.

Дополнительно процедура использования вычислителя должна учитывать возможность поступления конечного числа элементов на вход обработки и завершения своей работы в этом случае. Для обеспечения указанного требования функция получения элемента входной последовательности должна дополнительно возвращать признак завершения потока данных на входе.

Выполнение операции получения очередного элемента входной последовательности заключается в чтении значения элемента из потока ввода. Аналогично выполнение операции вывода элемента выходной

последовательности состоит в записи значения в поток вывода. При этом требуется обеспечить выполнение дополнительной настройки операции выдачи посредством указания символа разделителя, используемого для отделения значений последовательных элементов в потоке вывода.

2.4 Порядок выполнения

1. В качестве основы взять программу, полученную как результат выполнения первой контрольной работы.
2. Создайте модуль для реализации функции использования вычислителя.
3. В созданном модуле определите типы данных, необходимые для представления в виде функций обратного вызова операций получения элемента входной последовательности и выдачи элемента выходной последовательности.
4. В созданном модуле создайте и получите реализацию функцию использования вычислителя в виде функции высшего порядка. При этом рекомендуется воспользоваться реализацией тестовой программой из первой контрольной работы, выполнив необходимую доработку в соответствии с предъявляемыми требованиями.
5. В основном модуле программы реализуйте функцию получения элемента входной последовательности с учетом предъявляемых к ней требований.
6. В основном модуле программы реализуйте функцию выдачи элемента выходной последовательности.
7. В основной (тестовой) программе обеспечьте создание вычислителя с его настройкой, настройку операций ввода и вывода, а также вызов функции использования вычислителя.
8. Подготовьте тесты для решаемой задачи, охватывающие как можно больше различных сценариев поступления данных.
9. Проведите тестирование полученной программной реализации и при необходимости выполните её отладку для устранения выявленных на этапе тестирования ошибок. Данный этап необходимо выполнять до успешного прохождения всех подготовленных тестов.
10. Напишите отчет о выполнении контрольной работы в соответствии с рекомендациями раздела 2.5.

2.5 Структура отчета

Отчет о выполнении контрольной работы должен содержать **титульный лист** с названием работы, фамилиями студента и преподавателя, а также названием университета, города и года выполнения работы.

Первым разделом является «**Постановка задачи»**, в котором полностью приводится как общая формулировка задачи, так и указанный преподавателем вариант, перечень исходных данных и требуемых результатов, а также контрольные примеры для уточнения постановки задачи.

В следующем разделе «**Архитектура программной системы**» приводится перечень программных компонент, используемых в полученной программной системе, с пояснением назначения каждой компоненты. Также раздел должен содержать схему внутреннего устройства системы, на которой изображаются перечисленные компоненты и их взаимодействие.

В разделе «**Использование индуктивного вычислителя**» приводится описание функции, обеспечивающей управление обработкой последовательности входных данных. Описание должно включать интерфейс функции с подробным пояснением назначения каждого ее аргумента, и алгоритм ее работы, представленный либо в словесной форме, либо в виде блок-схемы. Также поясняются реализованные функции обратного вызова с описанием структур данных, включающих информацию, необходимую для выполнения этих функций.

В последнем разделе «**Приложение**» приводится исходный текст программной реализации и основной программы, демонстрирующей его применение в соответствии с общей постановкой задачи.

2.6 Контрольные вопросы и задания

1. Что влияет на выбор конкретного способа обработки данных?
2. В чем состоит проблема выбора оптимальной реализации некоторого алгоритма обработки данных?
3. Что такое повторное использование программной компоненты? Каковы критерии возможности его применения в разрабатываемой программной системе?
4. Предложите возможные способы обеспечения повторного использования и проведите анализ качества каждого из этих способов.
5. Что такое функция высшего порядка?
6. Поясните возможные способы применения функций обратного вызова.
7. Приведите типовую сигнатуру функции обратного вызова и поясните ее составные части.
8. Укажите компоненты общей схемы метода обратного вызова и поясните их назначение.
9. Как при использовании индуктивного вычислителя учитывается возможность поступления конечного числа элементов на вход обработки?

10. Поясните реализацию операции получения очередного элемента входной последовательности
11. Объясните реализацию операции выдачи очередного элемента результирующей последовательности.
12. Дайте подробное объяснение программной реализации функции использования индуктивного вычислителя.
13. Опишите реализованную схему тестовой программы.

3 Список литературы

1. **Сидоров, С. Б.** Алгоритмы и методы организации программных систем: учеб. пособие / С.Б. Сидоров [и др.]; НГТУ им. Р.Е.Алексеева. – Нижний Новгород, 2023. – 146 с.
2. **Кушниренко, А.Г.** Программирование для математиков : учеб. пособие / А. Г. Кушниренко, Г. В. Лебедев. – М. : Наука, 1988. — 384 с.
3. **Гамма, Э.** Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. — СПб. : Питер, 2001. — 368 с.