Практическое занятие 1.
Спецификация: Исходные данные и результаты

Задание: Рассмотреть задачу из Лабораторной работы №2 на примере спецификации (пункты 1-7) из файла Пример-отчета-для-лабораторной-работы-2.*pdf*.

К каждой задаче должна прилагаться спецификация.
Спецификация задачи – полное описание решенной задачи, состоящее из следующих пунктов:

1. Постановка задачи
2. Уточненная постановка задачи
3. Пример решения
4. Таблица данных
5. Входная форма
6. Выходная форма
7. Аномальные ситуации
8. Функциональные и структурные тесты
9. Метод
10. Алгоритм
11. Программный код

Спецификация делается заранее при подготовке к лабораторной работе. Ее можно написать от руки, или сделать в электронном виде, а затем распечатать или с согласия преподавателя показать на экране компьютера.

♦ **Постановка задачи**
Берется из задачника. Например,

«Найти сумму всех положительных элементов массива.»

♦ **Уточненная постановка задачи**
Уточненная ВАМИ постановка задачи. Надо дополнить по своему усмотрению пропущенные детали (типы, структуры, имена, альтернативные решения). Например,

«Задан *одномерный целочисленный* массив *A*. Найти сумму *S* всех положительных элементов заданного массива.»

Почему *одномерный*? Потому, что обычно двумерный массив в задачах называют матрицей, а трех- и более мерные мы не будем рассматривать в данном курсе.
Почему *целочисленный*? Можно и *вещественный*… но сумму можно искать и целочисленного. А вот если бы надо было найти *сумму дробных частей* всех положительных элементов массива, то без вещественного типа элементов не обойтись.
Почему *A*? Можно назвать и по-другому… но тоже желательно покороче (1-3 буквы), например, *mas* (от слова ***мас***сив). Если массивов несколько, то им подойдут названия *A, B, C, X, Y, mas1, mas2, masA, masB*, …
Почему *S*? *S* – первая буква слова «сумма», но можно назвать и по-другому, например, *sum, summa* (от слова ***сум***ма), *svpem* (первые буквы фразы «сумма всех положительных элементов массива»)

♦ **Пример решения**
Решение задачи, проделанное вручную, помогает понять условие задачи, и, кроме того, этот пример может стать одним из тестов (пункт 8).

♦ **Таблица данных:**

Класс	Имя	Описание (смысл)	Тип	Структура	Формат
Входные					
Выходные					
Промежут					
очные					

Какие бывают данные? – Чтобы ответить на этот вопрос, надо ввести основание классификации, или признак, разделяющий данные на классы.

Ниже приведено несколько оснований классификации и соответствующих им классов данных.

Входные и выходные данные (по отношению к задаче)

Попробуйте дать определение входных данных.

Затем определите, пожалуйста, входные данные в следующих задачах.

1. Вычислить объем V цилиндра с высотой h и радиусом основания r.
   Входные данные: _____
2. Вычислить объем V цилиндра с высотой h=5 см и радиусом основания r.
   Входные данные: _____
3. Вычислить объем V цилиндра с высотой h=5см и радиусом основания r=7см.
   Входные данные: _____

А теперь посмотрим и определение входных данных и скорректируем Ваши результаты.

Входные данные – это данные,

1) без конкретных значений которых невозможно получение конкретного результата;

2) конкретные значения которых не заданы в условии задачи и не могут быть из него получены.

Проверьте себя, выделив входные данные в задачах:

4. Вычислить объемы n цилиндров с высотой h и радиусами оснований r, r+3, r+6, r+9, …..
   Входные данные: _____
5. Вычислить $S = \sum_{i=1}^{n} a_i$,где a – заданный массив из n элементов.

- В алгоритме входные данные представляются переменными. Значения их алгоритм запрашивает только в процессе выполнения – при выполнении операций ввода.

- Если используется ***интерактивный режим ввода данных (в диалоге)***, то для программы значений данных не существует до момента набора их на клавиатуре. По достижении операции ввода программа приостанавливается (“прерывание по вводу”) и ждет набора данных с клавиатуры.

Чтобы не испытывать дискомфорта перед пустым экраном, в программе в режиме диалога предусматривают *приглашения* – тексты, которые выводятся перед выполнением операторов вывода.

- При ***вводе данных из дискового файла*** данные заранее записываются в файл, откуда и читаются программой. *Приглашения* здесь не нужны. Для нас работа программы не прерывается вплоть до получения результата (или сбоя).

Промежуточные данные

В процессе решения задачи может потребоваться использование *вспомогательных переменных*.

В отличие от входных и выходных данных, состав которых определяется задачей, и по воле программиста выбираются только имена (возможно, структура), состав этих вспомогательных переменных может бесконечно варьироваться в зависимости от способа решения задачи и соображений (даже вкуса) программиста. Это – ***промежуточные данные.*** Они появляются по ходу решения задачи. Однако описание таких переменных сразу же, как только в них появляется необходимость, позволяет формализовать и более компактно описать дальнейший ход решения.

Чтобы разделить описания промежуточных данных и метода решения и в то же время получить текст метода, где все обозначения определены, удобно для описания промежуточных данных отвести

место (сначала пустое) в самом конце таблицы данных и заполнять эти строки по мере появления необходимости в промежуточных данных. В любом случае напишите слово «Промежуточные», пропустите столько же строк таблицы, сколько уже заполнено входными и выходными данными и приступайте к размышлению над решением задачи.

Все промежуточные данные обязательно должны быть описаны!

Типы данных (по смыслу задачи, по представлению в компьютере)

Тип данных определяет *размер* выделяемой ячейки памяти, *диапазон* и множество допустимых *операций*. Пока будем рассматривать только числовые типы, а именно – ***целый и вещественный***.

Целые				Вещественные			
Тип	Размер	Диапазон	Пример	Тип	Размер	Диапазон, точность	Пример
byte	1 байт	0..255	12 -125 $A5 {165 в 16с/с}	single	4	$e^{-45}..e^{38}$ 7..8 значащих цифр	-15.2 5e-2 $\{5\bullet10^{-2}\}$
shortint	1	-128..127					
word	2	0..65535		double	8	$e^{-324}..e^{308}$ 15..16 значащих цифр	
smallint	2	-32768..32767					
longint	4	-2 147 483 648.. 2 147 483 647		real	8		
integer	4						
cardinal	4	0.. 4 294 967 295		extended	10	$e^{-4951}..e^{4932}$ 19..20	
Int64	8	-2^63… 2^63-1					
UInt64	8	0…2^64-1					
a:=199 **div** 10; { → 19} a:= a **mod** 10; { → 9} a:=ord(a); { →a } a:=pred(a); { →a-1 } a:=succ(a); { →a+1}				b:= 9/2; { → 4.5} b:=int(4.5); { → 4.0} b:=frac(4.5); { → 0.5} b:= round(b); { → 1} b:= trunc(0.3); { → 0}			

Тип данного в задаче как правило можно определить из условия задачи. Так, количество каких-либо объектов – целое число; размеры, координаты – чаще всего вещественные и т.д.

В компьютере *целые числа* представляются *точно*.

Вещественные числа представляются *приближенно*, с некоторой точностью.

Точность задания входных данных задается при постановке задачи. Это – число знаков, обычно – знаков после точки, которым можно доверять. При измерении точность определяется возможностями измерительного прибора (например, обычная линейка дает точность 0.1 см). При вычислениях точность результата определяется математическими правилами.

Точность представления в компьютере определяется типом.

В разных языках программирования работа с вещественными числами может осуществляться по-разному. Поэтому, строго говоря, *сравнивать вещественные числа на равенство в общем случае не следует*. Так, например, сравнение «4.0 = 4.0» в языке Паскаль истинно, а в языке Фортран – всегда ложно. Если же в одной из частей равенства находится значение, заданное программистом с некоторой точностью, а в другой – значение, вычисленное с точностью машинной, то результат сравнения будет ложным, даже если с практической точки зрения величины равны. Пример.

2 =?= 2,1 – 0,2/2

2,1 – 0,2/2 = 2,1 - 0,1 = 2

При переводе в 2 СС некоторые конечные (в 10 СС) дроби становятся бесконечными (в 2СС) и округляются для сохранения в конечной памяти (по умолчанию тип *extended* – 10 байт). При вычислениях (деление) результат также округляется.

В результате $2 \neq 2{,}1 - 0{,}2/2$

Выход состоит в правильном *сравнении на равенство вещественных чисел с учетом точности* их представления. Два вещественных числа a и b, представленные с точностью t (t =0.1. 0.01 и т.д. – десятичные знаки, которым можно доверять), равны, если $|a-b| < t/2$.

(2 – (2.1 – 0.2/2) < 1e-7)

Тип данных в алгоритме необходимо описывать.

Диапазон данных – существует в любой задаче. Нельзя придумать задачу, где данные менялись бы например, от 10^{20} до 10^{-20}; если это физические данные, то они относятся к разным разделам физики.

Выход значений данных за границы диапазона – первая аномалия. Таким образом, задавая и проверяя диапазоны данных, мы можем реализовать свойство надежности программы.

Если в учебных задачах значения диапазона не следуют из условия, будем представлять некую мысленную модель и брать эти границы искусственно.

Преобразование типов (целое – вещественное):

Целому числу нельзя присвоить вещественное без предварительного преобразования (*Round*,*Trunc*)

Остальные присваивания: присваиваемое значение должно лежать в диапазоне переменной-приемника.

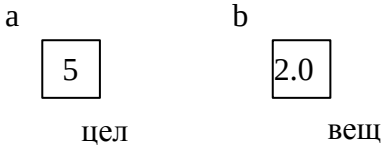
Формат. Если диапазон и точность определены, то определим схему для представления любого данного из заданного диапазона. Обозначим значком «+» знаковую позицию, крестиками XX…XX – числовые позиции, а точку представим как она есть.

Формат – схема представления числа из заданного диапазона, занимающего максимальное количество позиций. Пусть |a| <=100, точность 0.01.

Тогда формат: +XXX.XX (всего 7 позиций, две после точки; в Паскале это кодируется как :7:2).

Структура данных (по способу организации данных одного типа)

•• Простая переменная имеет одно имя и одно значение.



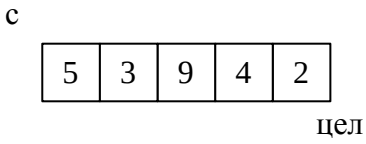
Описание:

тип – цел; имя – a; структура – простая переменная;

тип – вещ; имя – b; структура – простая переменная

•• Массив – совокупность *однородных* данных, располагающаяся в *соседних ячейках* памяти и имеющих *общее имя и различающихся по индексам*. В простейшем случае индексы – это номера элементов. Количество индексов, необходимых для однозначной идентификации элемента, определяет размерность массива.

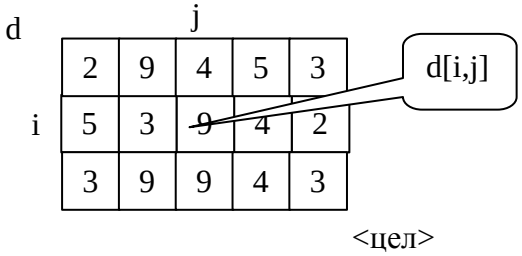
Одномерный массив – упорядоченная по номерам последовательность. Например,



Описание: Тип – цел; имя – c; структура – одномерный массив(5)

Элементы массива: c_1, c_2, … c_5, в общем виде – c_i или c[i].

Двумерный массив (матрица) – структура из строк и столбцов, например:



Описание: Тип – цел; имя – *d*; структура – двумерный массив (3x5);

Элементы массива: в общем виде – d_{ij} или $d_{i,j}$ или $d[i,j]$.

Здесь первый всегда индекс *строки*, затем *столбца*.

Константы и переменные (по «постоянству значений»)

Константа представляет сама себя (что написано, то и есть) и не изменяет значения при выполнении алгоритма. Например, 4 – числовая константа «четыре», ‘a’ – символьная константа “a”, ‘result’ – строковая константа «result».

Константы в задаче 4.4.0:

- числовая константа 100 в конструкции

```
const
  Nmax=100;
```

- числовые константы в форматах операторов вывода;
- там же – строковые константы (текст в одинарных кавычках).

Переменная – область памяти («ячейка»), имеющая имя. Значение переменной может изменяться; при этом старое значение стирается и заменяется новым. В любой момент переменная имеет некоторое значение, и фраза «значение переменной не определено» означает только, что в момент использования (как правило, первого использования) переменная не получила нужного начального значения вследствие небрежности программиста. *За заданием начальных значений должен следить разработчик алгоритма!*

Имя переменной – последовательность из 1-255 латинских букв, цифр и знака подчеркивания “_”, начинающаяся с буквы или знака подчеркивания. Например, правильные имена – *fd, r, a*12*d*23, *gamma, _*1, *a_*1.

Переменные описываются в разделе

```
var
    a, i, j: integer;  b: real;
    c: array [1..5] of byte;
```

♦ Входная форма

Поясняет пользователю программы, что должно появиться на экране и куда вводить значения.

Несоответствие вида экрана во время работы программы и входной формы, изображенной в спецификации, и отсутствие подсказок/приглашений дезориентирует пользователя.

При вводе данных из файла приглашения/пояснения не обязательны, важно расположение данных в файле, который должен создать пользователь.

♦ Выходная форма

Поясняет пользователю программы, что и где должно появиться на экране после выполнения программы. Для удобства пользователя и обычно перед выводом результатов выводят входные данные, чтобы и входные данные и результат находились перед глазами пользователя.

Несоответствие вида экрана во время работы программы и выходной формы, изображенной в спецификации, и отсутствие пояснений к результатам дезориентирует пользователя.

♦ Аномальные ситуации

Аномальные ситуации – ситуации возникающие при невозможности применить алгоритм для некоторых данных. На пример вычислить функцию для значения, лежащего вне области ее определения.

Каждая ситуация описывается и выдается соответствующая реакция, например сообщение и повторный ввод данных.

№	Описание	Условие возникновения	Реакция на аномалию

В условиях желательно *не* использовать связку $\vee$ (ИЛИ), объединяя разные ситуации в одну.

♦ Альтернативные ситуации

Если в задаче требуется найти в множестве объектов какой-либо объект, удовлетворяющий некоторому условию, необходимо предусмотреть в решении ситуацию, когда искомого объекта может не быть. Это – не аномалия, а вариант решения задачи.

Строго говоря, эта ситуация должна быть предусмотрена уже на уровне определения выходных данных как возможный вариант результатов и зафиксирована в описании выходных данных, выходной

форме и тестах. Реально это не всегда отслеживается и перетекает в пункт «Метод», а то и пропускается вообще (что не есть good).

Если Вы пропустили такую ситуацию, анализируя выходные данные, и разрешили ее только при описании метода, вернитесь выходным данным и скорректируйте их описание и – особенно – выходную форму: при отсутствии искомого должен быть соответствующий вывод.

Возможные варианты описываемой ситуации :

А) все получается автоматически, само собой, например, когда мы ищем число объектов, которые могут быть среди заданных объектов, а могут и не быть; в одном случае получим ненулевое значение искомого числа, в другом – нулевое (объектов нет);

Б) необходимо разделить случаи наличия и отсутствия искомого и обрабатывать их по-разному.

Например, требуется найти некое среднее арифметическое для объектов, удовлетворяющих условию. Так как среднее арифметическое=сумма/количество, а при отсутствии искомых объектов их количество равно нулю и делить на ноль нельзя, надо:

- определить упомянутое количество;
- проанализировать полученное значение;
- в зависимости от результатов анализа считать и выводить значение среднего арифметического или заранее предусмотренное сообщение.

Вариант б) более универсальный и в итоге более четкий и наглядный, но требует более тонкой работы.

В предлагаемом решении задачи из примера реализован вариант а): в случае отсутствия искомых точек напечатается значение *p*, равное 0.

♦ **Функциональные и структурные тесты**

С помощью тестов выясняется правильность работы алгоритма.

Подробнее про тесты позднее. Пока достаточно только функциональных тестов.

Для каждого альтернативного варианта решений нужен свой тест.

♦ **Метод – Алгоритм – Программа**

Алгоритм, алгорифм, одно из основных понятий (категорий) математики, не обладающих формальным определением в терминах более простых понятий, а абстрагируемых непосредственно из опыта. Алгоритмами являются, например, известные из начальной школы правила сложения, вычитания, умножения и деления столбиком, нахождение корней квадратного уравнения и правила перехода через улицу.

Алгоритм *– упорядоченная последовательность действий, формальное последовательное выполнение которых ведет к решению поставленной задачи за конечное число шагов.*

Программа (от греч. *programma* — объявление, распоряжение, указ), 1) план деятельности, работ. 2) Изложение основных положений и целей деятельности политических партии, организации или отдельного деятеля. 3) Краткое изложение содержания учебного предмета. 4) Упорядоченная последовательность действий для ЭВМ, реализующая алгоритм решения некоторой задачи.

Программа *– алгоритм, закодированный на одном из языком программирования.*

Увы, строго формальных правил здесь привести нельзя, как нельзя провести четкую границу между этапами «метод–алгоритм–программа». Однако можно выделить некую общую схему действий и предложить правила для ее описания.

Исходная посылка вполне тривиальна: прежде чем описывать алгоритм (формальную схему), надо по меньшей мере на неформальном уровне знать, как решить предъявленную задачу. Критерий такого знания – умение вручную получить результаты для любого конкретного примера. Пока этот уровень понимания не достигнут, следует терпеливо постигать смысл задачи. Иначе, уже на этапе отладки вы поможете понять, что написали код совсем другой программы.

Главное действо начинается, когда смысл задачи понят и определены ее входы-выходы. Итак, мы ***можем*** решить задачу, и надо теперь это умение ***выразить*** четким языком. Для этого и предназначен пункт «Метод», и главным образом к нему относятся приводимые рекомендации.

Цель их – помочь описать схему решения задачи максимально точно, с минимальными усилиями и так, чтобы получившийся текст был читабелен, *без многократного переписывания*. Если это удается, то перейти к алгоритму – дело техники.

♦ **Метод. Формулы и другие взаимосвязи между данными**

Если в Вашей задаче есть формулы или какие-либо общие закономерности, связывающие данные, выпишите их сразу же в разделе «Метод», введя, если надо, промежуточные переменные и вписав их в соответствующий раздел.

Это могут быть:

- геометрические и тригонометрические формулы (вычисление расстояний для точек; формулы для треугольников и др. фигур и т.д.);
- формулы для вычисления процентов, средних величин;
- и многое другое, зависящее от содержания конкретной задачи.

Пронумеруйте формулы. Далее в ходе описания метода можно повторно выписать эти формулы, а можно сослаться на них.

Выявление и запись возможных закономерностей как можно раньше уже неявно разбивает Вашу задачу на подзадачи и направляет ход решения в нужное русло**.**

♦ **Нисходящая разработка. Абстракции**

Приводимые далее рекомендации являются универсальными, хотя исходно были сформулированы как рекомендации для успешного применения нисходящего подхода.

1.Ключ к успеху: делать *формализованное и строгое описание* входов, функций и выходов всех абстракций ,то есть спецификацию соответствующих подзадач.

2.На каждом этапе следует *выделять главные*, максимально общие подзадачи; концентрировать внимание на самом существенном, *отвлекаясь от мелочей***.**

3.Точно, кратко, емко формулировать подзадачи: должно быть ясно, что и над чем делается, и что должно получиться. Это же – критерий *правильности выделения подзадач***.**

4.Если на каком-либо шаге встретились непредвиденные проблемы, не бойтесь *пересмотреть* неудачный проект.

5.На каждом этапе раскрывайте абстракцию в одну из базовых структур, таким образом, чтобы не терять контроля над *структурой* алгоритма.

<абстракция> → следование
→ ветвление
→ цикл

6.Реализацию каждой из абстракций (спецификация-алгоритм-программа) записывайте на отдельном листе – *наглядность***.**

7.Учитывая требование 2, на первом этапе проектирования целесообразно *отделить обработку* данных от ввода и вывода исходных данных и от вывода результатов.