Дед ПахМелыч

**Основы алгоритмов**

2010

**Оглавление**

Глава 1. Введение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

Глава 2. Алгоритмы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

Глава 3. Условия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

Глава 4. Циклы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

Глава 5. Функции \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

Глава 6. Массивы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

Глава 7. Методы и подходы составления алгоритмов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_16

Глава 8. Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_17

**Глава 1**

**Введение**

Алгоритмы в нашей жизни встречаются повседневно, благодаря своей близости к реальной жизни, они позволяют нам легко реализовывать всевозможные программы на различных языках программирования, принимать решения в реальной жизни и просчитывать ситуацию. Конечно, в программировании многое зависит от синтаксиса и назначения языка, но искусственный язык стараются сделать как можно понятней и удобней программисту, что приведет к более продуктивной работе кодера (программиста). Алгоритмы используются в скриптовых кодах, т.е. когда пишется сценарий приложения. Сценарий - это по сути тот же самый алгоритм, но написанный на языке программирования. В нем заранее расписано, что и при каких обстоятельствах должна делать программа. Для обычных людей сценарием является, например, инструкция, в котором "языком программирования" выступает обычный язык (китайский, русский, английский и т.д.), а сами действия, описанные в ней - это алгоритм.

Данная книга прежде всего рассчитана на новичков в программровании и не предназначена только для одного определенного языка программирования, стороной могут проходить те, кто изучает разметочные языки программирования (html, xml, wml и др.). Алгоритмы также изучаются в курсе информатики, и литературы по ним достаточно много. В данной книге будут рассмотрены в основном математические задачи (Хотя на их основе решаются более конкретные задачи).

**Глава 2**

**Алгоритмы**

Итак, теперь мы можем ближе перейти к теме этой небольшой книги. Для начала нам нужно определиться, что же такое алгоритм. Возьму стандартное определение, не буду изобретать велосипед, тем более с определением этим я согласен. Алгоритм - это последовательность действии, приводящих к достижению цели. Есть и другое определение понятия алгоритм - запись метода решения задачи. Алгоритмы встречаются не только в программировании, но и в жизни, что и видно из определения. Приведу простой пример алгоритма:

1 Проснуться

2 Сходить в ванную

3 Позавтракать

4 Почистить зубы

5 Одеться

6 Выйти на улицу

7 Доехать до работы

Теперь, прежде чем графически изобразить этот алгоритм, я введу понятие блок схемы. Блок-схема – это графическое отображение алгоритма, в которой каждое действие записано в отдельном графическом блоке (фигуре), и эти блоки связаны между собой линиями связи. Давайте нарисуем блок-схему предыдущего примера:

Проснуться

Сходить в ванную

Позавтракать

Почистить зубы

Одеться

Выйти на улицу

Доехать до работы

Рассмотрим схему поподробней. В овальных блоках записываются действия начала и конца алгоритма, иногда их опускают. В схемах действия записываются в прямоугольные блоки. Они выполняются сверху вниз (за редким исключением, где линии связи будут отображаться стрелками, которые указывают направление выполнения алгоритма). Линии связи должны идти под прямыми углами.

В этом примере я написал алгоритм поездки на работу. Прошу заметить, именно эту задачу можно решить и другим путем - все таки каждый может придумать свой метод решения задачи, свой алгоритм. Но если поменять некоторые действия местами, алгоритм будет ложным, например, поменяйте местами первый и последний пункт алгоритма. Навярника вы улыбнетесь. Этим я хочу объяснить, что программист должен хорошо орудовать алгоритмом ( а так же кодом своей программы ), т.е. менять алгоритм для достижения одной и той же цели, переделывать свой алгоритм в другой и для другой цели, но имеющий общие части и др. Чтобы уметь так работать с кодом, алгоритмом надо иметь навык программирования, хотя в принципе и без этого можно уметь "играть" кодом, но это уже талант...

Иногда в программе требуется, чтобы пользователь ввёл некоторые данные, в блок-схеме это записывается так:

Введите значение

В одном блоке можно ввести несколько значений, чтобы не удлинять блок-схему.

Двинемся дальше. Алгоритмы делят на две группы: линейные и нелинейные. В линейных алгоритмах действия перечисляются непосредственно друг за другом. В нелинейных алгоритмах в цепочке действий есть ответвления. Чаще всего это выражается применением условии. Как правило в речи аналог этому союз если, например: если сегодня будет хорошая погода, я пойду гулять, иначе останусь дома. В этом примере выполнится одно из двух действий: пойду гулять или останусь дома. Одновременно они выполниться не могут ( в нашем примере они еще и противоречивы ), а выполнится одно в зависимости от выполнения условия, т.е. При хорошей погоде пойду гулять, а при плохой останусь дома. Условие в нашем примере "если сегодня будет хорошая погода". То есть в алгоритмах имеются элементы логики, без которых не обходятся многие программы, с помощью логики программу можно сделать "умной", интерактивной. В последующих главах мы поподробнее затронем эти вопросы.

В практике программиста нелинейные алгоритмы встречаются гораздо чаще линейных, поэтому их надо хорошо понимать. Если человек умеет хорошо создавать нелинейные алгоритмы, тем более сложные, то он без особого труда освоит любой алгоритмический язык. Другое дело некоторые такие языки имеют тяжелый синтаксис, но его освоить проще, имея умения составлять программы, хоть и в словесной форме. С внедрением объектно-ориентированного программирования, многие программисты считают алгоритмические языки слишком простыми для написания современных программ, но на мой взгляд без этой основы не возможно программирование, хоть в этой основе и нет крайне удобных функции для программирования. К тому же без знаний алгоритмов будет сложнее изучать принципы объектно-ориентированного программирования.

**Глава 3**

**Условия**

Двинемся чуть далше. Часто в повседневной практике нам приходится совершать выбор, например: в магазине мы думаем, купить ли хлеб по такой цене; когда переходим дорогу, смотрим на светофор и, если на нем горит зеленый цвет, переходим ее. Совершенно ясно, что такие действия совершаем мы нелинейно - мы делаем выбор. В первом случае критерием условия была цена хлеба, а во втором - цвет светофора. Теперь давайте составим наши алгоритмы:

***А)***

*1 заходим в магазин*

*2 смотрим цены*

*3а если цена хлеба меньше или равна 10 рублей:*

*3.1 покупаем хлеб*

*3б иначе*

*3.2 не покупаем хлеб*

В блок схеме условие обозначается ромбом с ответвлением линии связи на два направления.

Заходим в магазин

Смотрим цены

Цена хлеба меньше или равна 10р?

Да Нет

Не покупаем хлеб

Покупаем хлеб

Видно, что алгоритм разделился на два направления (ветки) после вопроса (Ветка Да и ветка Нет). Два этих направления потом **должны обязательно сойтись**! В каждой ветке последовательность действий пишется по тем же правилам, что и любой алгоритм (Разве не надо писать блоки начала и конца).

***Б)***

*1 подходим к дороге*

*2 смотрим на светофор*

*3а если горит зеленый цвет:*

*3.1 переходим дорогу*

*3б иначе*

*3.2 не переходим дорогу*

Подходим кдороге

Смотрим на светофор

Горит зелёный цвет?

Да Нет

Не переходим дорогу

Переходим дорогу

Как видите, условие выражается при помощи слова "если", слово "иначе" указывает, что надо делать, если условие не выполняется. Если слова "иначе" нет,то при невыполнении условия ничего не надо делать. Условия часто применяют при решении математических задач, при составлении программ и т.д. Попробуйте составить словесный алгоритм решения квадратного уравнения через дискрименант.

Вообще, условие в целом имеет следующую структуру:

1)Вопрос

2а) дейчтвия если ответ положительный

2б) действия если ответ отрицательный.

И графически:

Базовая

Вопрос

Да Нет структура

Действия Действия

Иногда в одной из веток может и не быть никаких действий, но линия ветки чертится всё-рано.

Можно создавать и многоуровневые условия, которые в программировании имеют два метода. Но на блок схеме они будут выглядеть одинаково.

Первый заключается в словесной форме применением фразы иначе если. Рассмотрим пример:

1 Сдаем в институте экзамены

**2а** если сдал все экзамены:

2.1 Меня переводят на следующий курс

**2б** иначе если не сдал один экзамен:

2.2 Меня оставляют на второй год обучения

**2в** Иначе

2.3 Меня отчисляют из института, и я счастливый иду в армию

Сдаём в институте экзамены

Сдал все экзамены

Да Нет

Меня переводят на следующий курс

Не сдал один экзамен

Да Нет

Меня оставляют на второй год

Меня отчисляют из института, ия счастливый иду в армию

Ещё раз обращаю внимание на то, что все ветки алгоритма сходятся по базовой структре.

Из примера видно, что наш алгоритм разветвляется на три части, в зависимости от успешности сдачи экзамена. Такой прием сложных условий также часто встречается в программировании. Таким образом можно создать какое угодное количество условий.

Иногда такой синтаксис можно реализовать гораздо проще, если в условии проверяется какой-то один общий параметр. В прошлом примере таковым являлся параметр успешности сдачи экзамена. Перепишем наш алгоритм:

1 Сдаем в институте экзамены

2 Успешность сдачи экзаменов:

2.1 Сдал все экзамены - Меня переводят на следующий курс

2.2 Не сдал один экзамен - Меня оставляют на второй год обучения

2.3 Не сдал два экзамена и больше - Меня отчисляют из института, и я счастливый иду в армию

Здесь блок-схема аналогична прошлому примеру

Видно, что алгоритм стал покороче и более читабельным. В принципе вам считать, какой из приведенных методов вам удобней.

**Глава 4**

**Циклы**

Иногда требуется много раз выполнять какое-либо действие, например, когда мы пишем список чисел: мы будем писать числа последовательно, например пока не дойдем до числа 10. Конечно, такой алгоритм можно решить и линйным способом, но представьте каким бы был алгоритм, если нам требовалось записать список чисел от 1 до 1000000... Поэтому напишем алгоритм с циклом:

1 выполняем цикл до тех пор, пока число итерации не будет равно 10

* 1. пишем число итераций

i:=1

i<=10

Да Нет

Печать i

i:=i+1

Как видно, блок схема содержит условие, но в данном случе одна из веток отходит назад, что и позволяет выполнять действия в этой ветке несколько раз. В данном примере для удобства я ввёл переменную i, в которую я записываю значение оператором «:=». Циклы бывают двух типов: ДО и ПОСЛЕ. В данном случае это цикл ДО, так как *условие* цикла стоит *в начале*, перед веткой, которая должна повторяться. Благодаря условию мы проверяем, надо ли выполнять дальше итерацию цикла. Это условие также называют критерием окончания цикла. С ним надо быть осторожней. Если бы я забыл в итерационной ветке действие i:=i+1, то переменная I не изменялась бы, поэтому условие всегда было бы истинным, и цикл стал бы бесконечным.

Итерация - это один проход через тело цикла. Тело цикла - это действия, которые надо выполнить циклом n-е количество раз. Как видно, в цикле используется условие, без него бы цикл "зациклился". В языках программирования цикл можно прервать внутри цикла:

1 выполняем цикл до тех пор, пока число итерации не будет равно 100

1.1 пишем число итераций

1.2 если число итераций равно 10

1.2.1 выходим из цикла

Графически изобразить прерывании цикла можно, но это противоречит основным правилам составления блок-схем.

В этои случае будут написаны числа от 1 до 10, условие внутри цикла "глушит" в нашем случае 90 итераций. Рассмотрим еще пример: нам требуется вывести список чисел 0 2 4 6 8 10, т.е. только четные числа, вот решение:

1. начальное число=0, выполняем цикл до тех пор, пока начальное число не будет равно 10; начальное число = начальное число + 2

1.1 пишем начальное число

i:=0

i<=10

Да Нет

Печать i

i:=i+2

В этом условии нельзя использовать число итераций, поэтому мы ввели свое число, которое мы увеличиваем с каждой итерацией на 2, до тех пор, пока оно не будет равно 10. Прошу заметить, если бы я в условии написал любое нечетное число, цикл стал бы бесконечным... Цикл также можно прервать внутри цикла, аналогично примеру с итерациями.

В блок схеме цикл реализуется с помощью оператора условия, рассмотренного в третей главе, в котором одно из оветвлений отбрасывает на нужное количество шагов назад. Например:

1 Начальное число равно 1

2 Выводимому числу присваиваем значение начального числа

3 Пишем выводимое число

4 Выводимое число увеличиваем на 1

5а Если выводимое число роавно 10:

5.1 конец прогаммы

5б иначе

5.2 Переходим к пункту 3

Блок схема данного примера была приведена ранее.

В данном примере будут написаны числа от 1 до 10. Такой метод практически не встречается в программировании, так как существуют более удобные операторы цикла, которые мы рассматривали в этой гнлаве. Заметьте, я в алгоритме употребил термие "присваиваем значение", это я сдедал не просто так. В программировании слово "равно" лишь сранивает два каких-либо значения, но не изменяет. А присваивание значения происходит другим оператором. Операторы сравнения и присваивания значения часто путают. Вообще все значения хранятся в переменных - в ячейках, областях памяти, которые имеют свой адрес. Кто-то может слышал понятие массив - это та же самая переменная, но которая может хранить много значений и в разной структуре, их анаологом в математике являются матрицы. Эти понятия присуще любому алгоритмическому языку.

Рассмотрим теперь на блок схемах базовые структуры циклов ДО и ПОСЛЕ. Они оличаются лишь расположением условия в цикле. В цикле ДО оно стоит *перед* итерационной веткой, а в цикле ПОСЛЕ – *после* неё. В цикле ПОСЛЕ итерационная ветка выполнится хотя бы раз, не зависимо от алгоритма.

Цикл ДО:

Условие

Итерационная ветка

Выход из цикла

Здесь в итерационной ветке должны содержаться действия, которые рано или поздно изменят ответ в условии цикла (чтобы цикл не стал бесконечным). Также и до условия могут содержаться действия, которые задают начальные значения для проверки условия ( в предыдущих примерах мы присваивали начальное значение переменной i). Также я не стал подписывать на схеме, где ветка да, а где ветка нет, так как в зависимости от алгоритма, итерационная ветка может находиться в любом нужном условии (но ответ условия не должен меняться с каждой итерацией).

Цикл ПОСЛЕ:

Итерационная ветка

Условие

Конец цикла

Здесь всё аналогично циклу ДО.

**Глава 5**

**Функции**

Иногда требуется выполнять какие либо действия несколько раз, но циклы применять нельзя, поскольку эти действия надо выполнять в разных частях программы и иметь разные параметры. Спасают положение функции и процедуры, в них записывают нужные действия и затем вызывают их в нужных местах. При программировании на разных языках программирования такой подход сильно экономит и упрощает код программы. В качестве примера напишем алгоритм чистки зубов утром, днем и вечером:

1 функция А:

1.1. Зазходим в ванную комнату

1.2 достаем зубную щетку

1.3 достаем зубную пасту

1.3 наносим пасту на щетку

1.4 чистим зубы

1.5 поласкаем рот

1.6 полоскаем щетку

1.7 убираем щетку

1.8 убираем пасту

1.9 уходим из ванной комнаты

2 если наступило утро:

2.1 выполняем функцию А

3 если наступил день:

3.1 выполняем функцию А

4 если наступил вечер:

4.1 выполняем функцию А

Блок –схема:

Здесь функция описывется блоком:

Функция A

Здесь в левой колонке обычно записывют входные данные, а в правой выходные.

Входные данные – это данные, которые передаются функции (процедуре), как начальные значения из основной программы. Выходные данные функция (процедура) передаёт основной программе после своего выполнения. Функция, в отличие от процелуры передаёт лишь один выходной параметр. Функции и процедуры описываются отдельными блок схемами, но вместо блоков начала и конца пишутся блоки с входными и выходными параметрами. Функции и процедуры можно вызывать сколь угодно раз. Блок-схему вы уже можете составить сами, пользуясь уже полученными знаниями (Пункт 1 это функция, а пункт 2 – основная программа).

Теперь представьте алгоритм без применения функции... В реальности этот алгоритм будет немного сложнее. При помощи функции пишут различные компоненты программы.

**Глава 6**

**Массивы**

Хочу немного поподробней остановиться на массивах. Как я уже написал - это пременная, которая может иметь несколько значений и иметь различную структуру. Применять массивы удобно для хранения какой либьо однотипной информации, информацию, относящуюся к какому-нибудь конкретному объекту и т.п. Например можно хранить информацию о машине:

Массив машина:

[1] марка

[2] цвет

[3] год выпуска

[4] пробег

В квадратных скобках записан элемент идентификационный номер массива. В каждом элементе хранится нужная информация (марка, цвет, год выпуска, пробег), и по идентификационному номеру элемента мы можем получить нужную нам информацию. В каждый элемент можно записать также элемент, тогда выйдет двумерный массив ( можно и более многоуровневые массивы создать ), получаются как бы массивы внутри массива:

Массив машина:

[1]

[1] Цвет

[2] тип кузова

[2]

[1] год выпуска

[2] пробег

Здесь мы видим двумерный массив. В элементе 1 мы видим два элемента, в которые мы записали внешние данные машины (цвет и тип кузова), в элементе 2 мы также записали два элемента, характеризующие технические параметры машины (год выпуска и пробег). Доступ к такти элементам уже идет через два идентификационных номера - элемента первого уровня и элемента второго уровня. Например, обратившись к [1][2] мы получим информацию о типе кузова.

Применение массивов позволяет структурировать информацию, упростить написание алгоритма. Например, с помощью цикла можно в двап счета отобразить информацию из массива, нежели ее отображать из каждой переменной. Массивы встречаются во многих программах, во многих языках программирования существуют встроенные служебные массивы. Значения массивов можно обрабатывать, как и значения переменных. В них можно хранить различную информацию и т.д.

**Глава 7**

**Методы и подходы составления алгоритмов**

Стиль составления алгоритмов может быть разным, но нужно придерживаться некоторых простых правил, которые упростят ваш алгоритм, уменьшат риск запутывания в алгоритмах, в общем сделают жизнь удоюней. В первую орчередь я советую программистам комментировать свой сценарии ( во всех языках есть такая функция ), так как в сложных сценариях легко запутаться, к тому же сторонним программистам тяжело будет разобраться. Также старайтесь разбивать свои задачи на логические части - на модули. Это достигается применением функций. Такой подход повышает читабельность программы, позволяет легко модифицировать и преструктуризировать прогаммы, позволяет использовать ее компоненты в любом месте программы, организует структуру программы. Ведь когда весь код свален в кучу-малу разобраться очень тяжело. Часто используемые модули, функции, компоненты удобно организовать в отдельную библиотеку, которую можно применять и в других программах. Для уменьшения количества багов и дыр тщательно прорабатывайте алгоритм и код сценария его реализации.

**Глава 8**

**Заключение**

Мы ознакомились с основными возможностями алгоритмов, с помощью которых можно написать практически любую программу, не рассмотрены лишь мелкие нюансы, с которыми можно ознакомиться при изучении какого-либо языка программирования. Также целью книги не было изучение объектно-ориентированного подхода при написании программ, по которым пишутся огромные учебные пособия. В комментах к этой книге оставляйте свои замечания и пожелания.

SMARTPARAD.RU