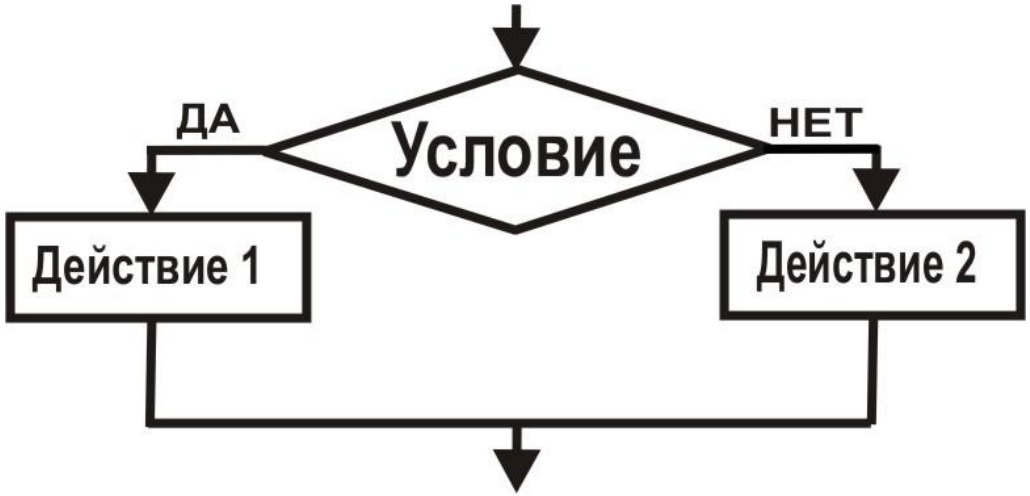
Алгоритмы. Программирование



29.11.2012

Базовые понятия

Алгоритм — набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий. (c) Wikipedia

Алгоритм каждому определенному набору входных данных ставит в соответствие некоторый набор выходных данных, т.е. вычисляет (реализует) функцию. При рассмотрении конкретных вопросов в теории алгоритмов всегда имеется в виду какая-то конкретная модель алгоритма.

«Свойства» алгоритма

Дискретность - алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. Каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего

Определенность - каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Выполнение алгоритма не требует никаких дополнительных указаний или сведений о решаемой задаче.

Результативность (конечность) - алгоритм должен приводить к решению задачи за конечное число шагов;

Массовость - алгоритм решения задачи разрабатывается в общем виде, то есть, он должен быть применим для некоторого класса задач, различающихся только исходными данными. При этом исходные данные

Правила построения алгоритмов

Первое правило – при построении алгоритма, прежде всего, необходимо задать множество объектов, с которыми будет работать алгоритм. Формализованное (закодированное) представление этих объектов носит название *данных*. Алгоритм приступает к работе с некоторым набором данных, которые называются входными, и в результате своей работы выдает данные, которые называются выходными. Таким образом, алгоритм преобразует входные данные в выходные.

Это правило позволяет сразу отделить алгоритмы от «методов» и «способов». Пока мы не имеем *формализованных входных данных*, мы не можем построить алгоритм.

Правила построения алгоритмов

Второе правило – для работы алгоритма требуется память. В памяти размещаются входные данные, с которыми алгоритм начинает работать, промежуточные данные и выходные данные, которые являются результатом работы алгоритма. Память является дискретной, т.е. состоящей из отдельных ячеек. Поименованная ячейка памяти носит название *переменной*. В теории алгоритмов размеры памяти не ограничиваются, т.е. считается, что мы можем предоставить алгоритму любой необходимый для работы объем памяти.

В реальной же жизни для работы программы, которая работает по данному алгоритму размеры памяти более, чем конечны. Так, например, для хранения переменной типа LongInt в Pascal требуется всего 4 байта, тогда как для хранения массива 1000×1000×1000 элементов потребуется 4 гигабайта.

Правила построения алгоритмов

Третье правило – дискретность. Алгоритм строится из отдельных шагов (действий, операций, команд). Множество шагов, из которых составлен алгоритм, конечно.

Четвертое правило – детерминированность. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки.

Пятое правило – сходимость (результативность). Алгоритм должен завершать работу после конечного числа шагов. При этом необходимо указать, что считать результатом работы алгоритма.

Виды алгоритмов

1. Механические алгоритмы (детерминированные, жесткие) - задают определенные действия, обозначая их в единственной и достоверной последовательности, обеспечивая тем самым однозначный требуемый или искомый результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм;

2. Гибкие алгоритмы (стохастические, эвристические)

2.1 Вероятностные (стохастические) алгоритмы дают программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата;

2.2 Эвристические алгоритмы - достижение конечного результата программы действий однозначно не предопределено, так же как не обозначена вся последовательность действий, не выявлены все действия исполнителя.

Виды алгоритмов

3. **Линейные алгоритмы** - наборы команд (указаний), выполняемых последовательно во времени друг за другом

4. **Разветвляющиеся алгоритмы** - алгоритмы, содержащие хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов;

5. **Циклические алгоритмы** - алгоритмы, предусматривающие многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.

Способы записи алгоритмов

- **словесная** (записи на естественном языке);
- **графическая** (изображения из графических символов);
- **псевдокоды** (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- **программная** (тексты на языках программирования).

Словесная запись алгоритма

Рассмотрим пример на алгоритме нахождения максимального из двух значений:

- определим форматы переменных X, Y, M, где X и Y - значения для сравнения, M - переменная для хранения максимального значения;
- получим два значения чисел X и Y для сравнения;
- сравним X и Y;
- если X меньше Y, значит большее число Y;
- поместим в переменную M значение Y;
- если X не меньше (больше) Y, значит большее число X;
- поместим в переменную M значение X.

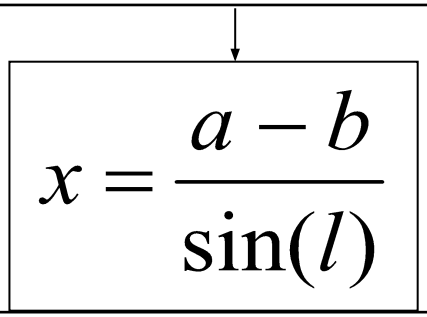
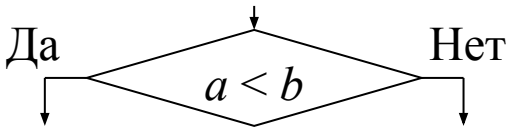
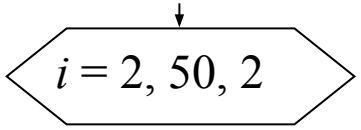
Проблемы:

1. Описание строго не формализуемо
2. Запись многословна
3. Неоднозначное толкование отдельных предписаний алгоритма

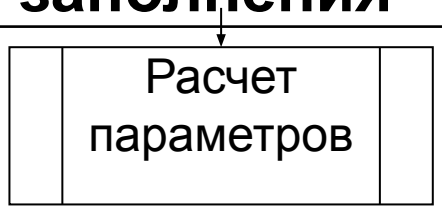
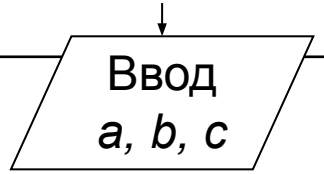
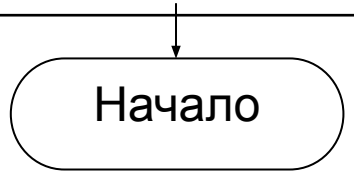
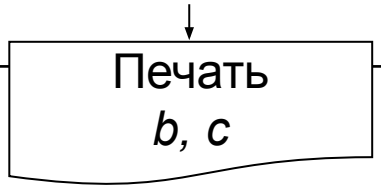
Графическая запись алгоритма

Структурная (блок-, граф-) схема алгоритма - графическое изображение алгоритма в виде схемы связанных между собой с помощью стрелок (линий перехода) блоков - графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

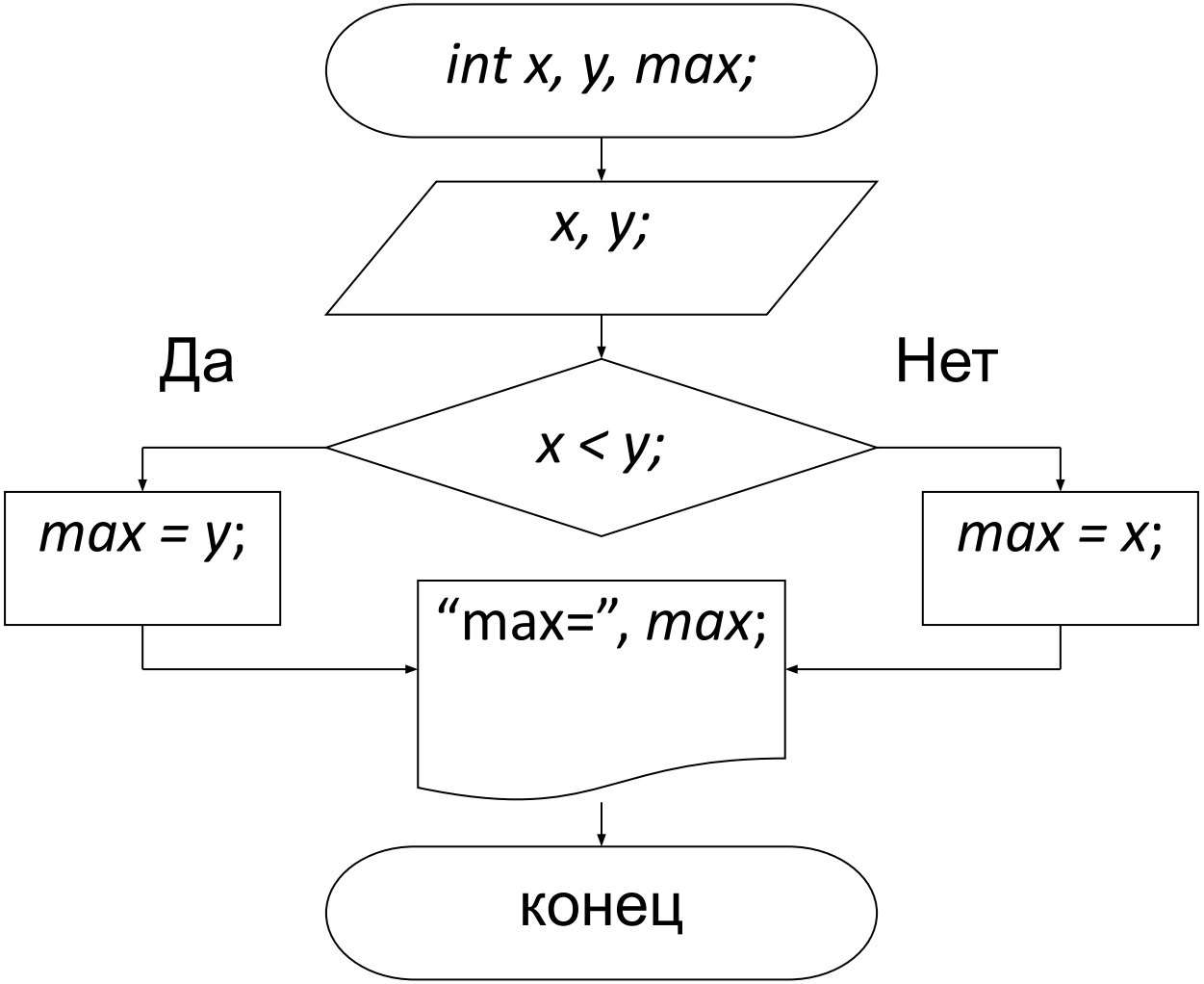
Типовые обозначения структурной схемы

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Ветвление		Проверка условий
Модификация		Начало цикла

Типовые обозначения структурной схемы

Название символа	Обозначение и пример заполнения	Пояснение
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод–вывод		Ввод-вывод в общем виде
Пуск–останов		Начало, конец алгоритма, вход и выход в подпрограмму
Документ		Вывод результатов на печать

Пример структурной схемы алгоритма



Запись алгоритма в псевдокоде

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками.

<Имя> **Посл.**
Выполнить <действие 1>
Выполнить <действие 2>
<Имя> **конец**

Выбор:

<Имя> **Выбор** <условие действия 1>
Выполнить <действие 1>
<Имя> **или** <условие действия 2>
Выполнить <действие 2>
<Имя> **конец**;

Повторение:

<Имя> **Повт**, пока не <условие действия>
Выполнить <действие 1>
<Имя> **конец**

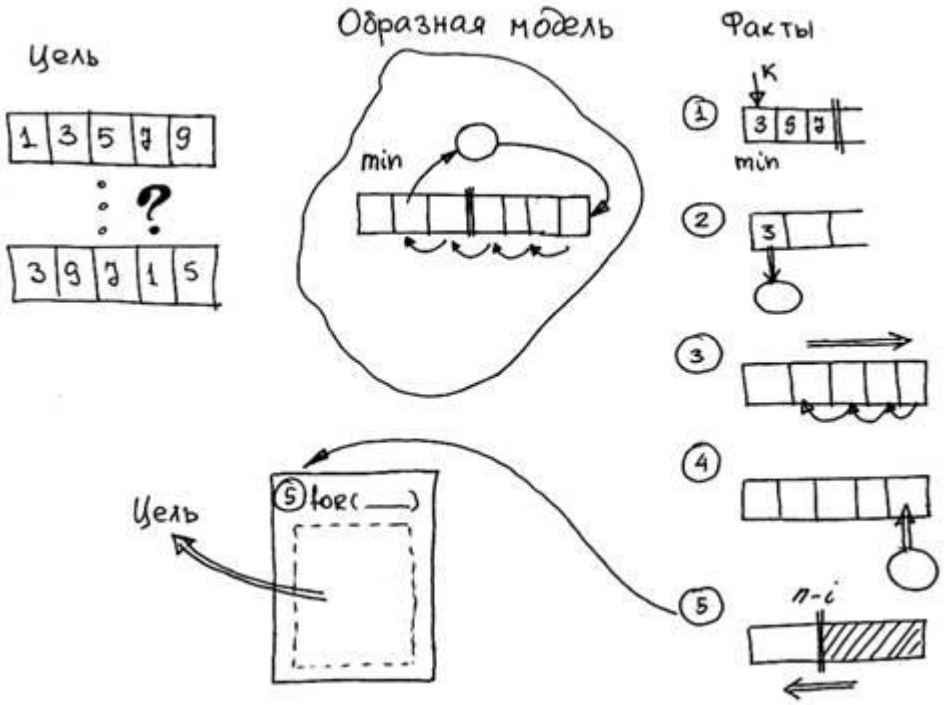
Программное представление алгоритма

На практике в качестве исполнителей алгоритмов используются компьютеры, поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на «понятном» ему языке. И здесь на первый план выдвигается необходимость *точной записи команд, не оставляющей места для произвольного толкования их исполнителем*.

Язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке - программой.

Этапы проектирования программы

1. Определение результата работы программы.
2. Формирование образной модели программы.
3. Формулирование фактов, касающихся программы.
4. Выстраивание программы из набора фактов.
5. Последовательное приближение к результату.



Этапы проектирования программы

1. Результат работы программы. Целью выполнения любой программы является получение результата, а результат – это данные с определенными свойствами. Например, целью программы сортировки является создание последовательности из имеющихся данных, расположенных в порядке возрастания. Точно так же любой промежуточный шаг программы имеет свою цель: получить данные с нужными свойствами в нужном месте. Как правило, постановка задачи начинается с формулировки результата.

Этапы проектирования программы

2. Образная модель программы. Формальное проектирование программы не продвинется ни на шаг, если программист «не видит», как это происходит. То есть первоначально действующая модель программы должна присутствовать в голове. Это – область образного мышления! Изобразительные средства здесь уместны любые – словесные, графические. Здесь работают интуиция, аналогия, фантазия и другие элементы творческого процесса. На этом уровне справедлив тезис, что программирование – это искусство

Этапы проектирования программы

3. Факты, касающиеся программы. Формальная сторона проектирования начинается с перечисления фактов, касающихся образной модели программы. К таковым относятся: переменные и их смысловая интерпретация, известные программные решения и соответствующие им стандартные программные контексты. Речь идет не об окончательных фрагментах программы(!), а о фрагментах, которые могут войти в готовую программу. Иногда при их включении потребуется доопределить некоторые параметры (например, границы выполнения цикла, которые не видны на этапе сбора фактов).

Этапы проектирования программы

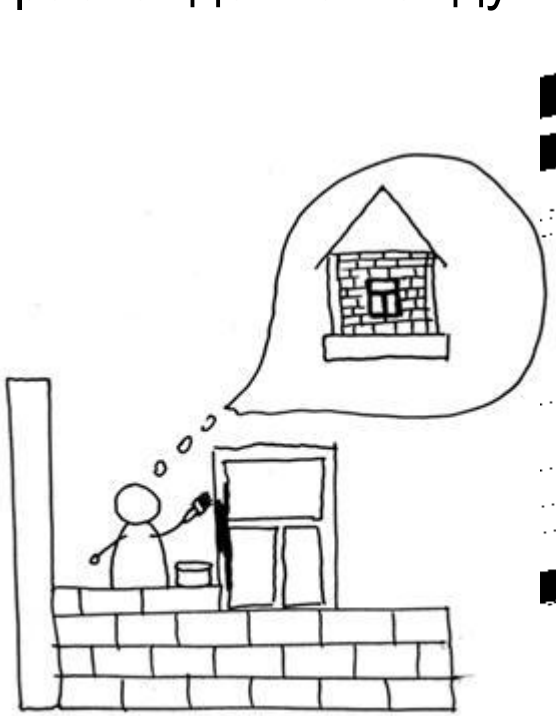
4. Выстраивание программы из набора фактов. Эта часть процесса программирования вызывает наибольшие затруднения, ибо здесь начинается то, что извне обычно и воспринимается как «программирование»: написание текста программы. Особенность заключается в том, что обычно фрагменты взаимосвязаны друг с другом прямо по структуре алгоритма или косвенно через данные. Различие подходов состоит в том, в какой последовательности в программу включаются фрагменты (по отношению к гипотетической готовой программе), с какой стороны начать этот процесс и в каком направлении двигаться.

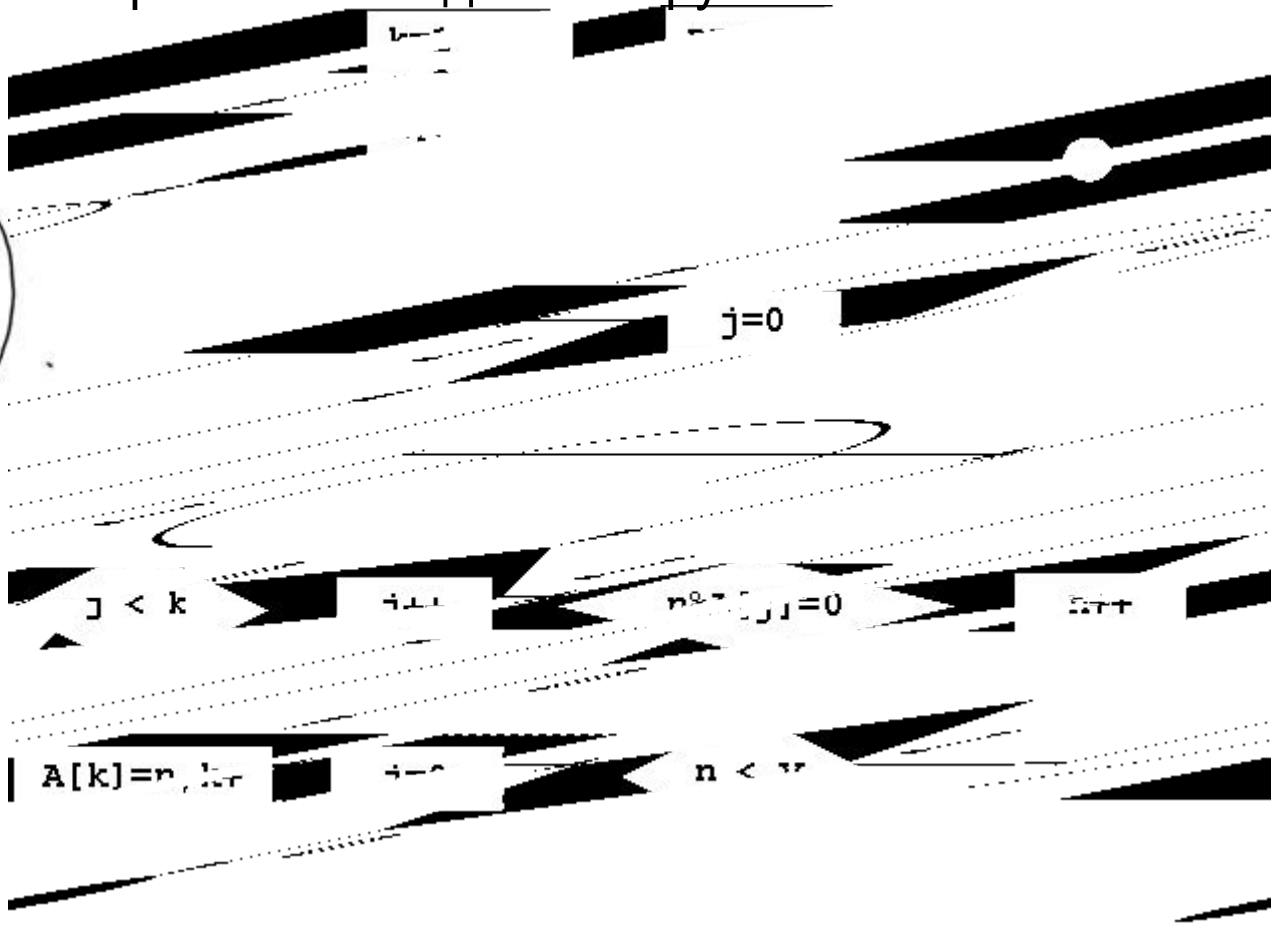
Этапы проектирования программы

5. Последовательное приближение к результату. Сложную программу не всегда удается спроектировать за один этап. Цикл «результат – образная модель – факты – выстраивание программы» может повторяться. При выстраивании программы может оказаться, что ненаписанная часть программы нуждается в дополнительном осмыслении, начиная с образной модели. При этом само направление выстраивания программы, т.е. собственно технология программирования, имеют большое значение: она в большей или меньшей степени гарантируется правильность и неприкосновенность уже написанного

«Историческое проектирование»

«Историческое» проектирование соответствует естественному ходу рассуждений. Программист просто записывает очередной оператор, который по его мнению должен выполняться программой в процессе ее работы. Ошибочность такого принципа состоит в том, что текст программы и последовательность ее выполнения - это не одно и то же и расхождение между ними рано или поздно обнаружится.





«Восходящее проектирование»

Восходящее проектирование – проектирование программы «изнутри», от самой внутренней конструкции к внешней. Привлекательность этого подхода обусловлена тем, что внутренние конструкции программы - это частности, которые всегда более «на виду», чем внешние конструкции, реализующие обобщенные действия. Частности составляют большую часть фактов в образной модели программы и, что самое ценное, могут быть непосредственно записаны на языке программирования.

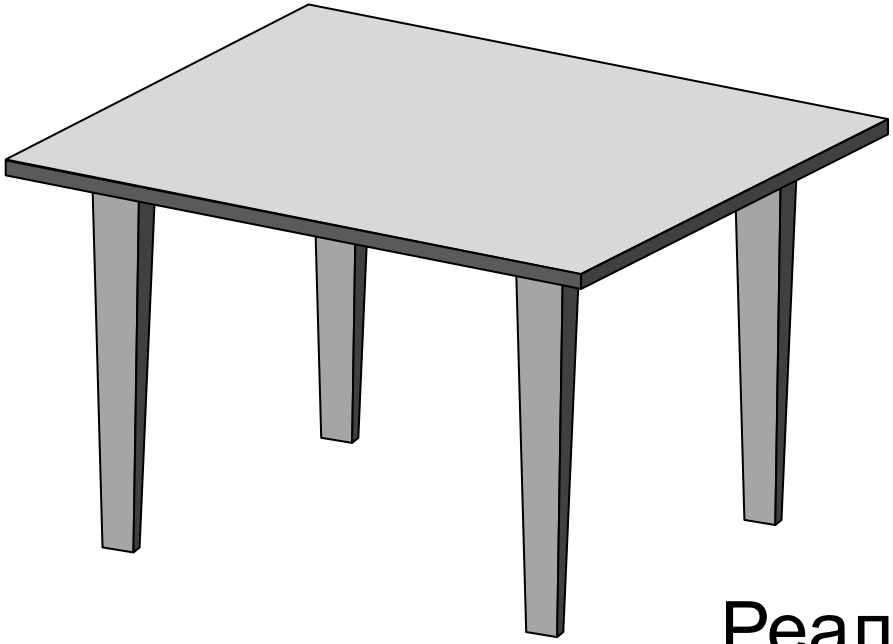
Недостатки :

- не факт, что программу удастся «свести» в единое целое, особенно сложную;
- поскольку параметры внутренних конструкций могут зависеть от внешних, то внутренние конструкции - не есть «истины в последней инстанции» и по мере написания программы тоже должны корректироваться.

«Нисходящее проектирование»

Нисходящее (структурное) проектирование - проектирование программы, начиная с самой внешней ее конструкции. Самое трудное, но самое правильное движение в направлении от общего к частному. Первая трудность заключается в неочевидности выбора самой внешней (общей, объемлющей) конструкции – частности всегда виднее. Вторая, менее очевидная: ненаписанная часть программы (внутреннее содержимое конструкции) также должна быть сформулирована в общем виде, т.е. словесно. Отсюда следует, что нисходящее проектирование должно сочетать в тексте программы формальное (то есть записанное на языке программирования) и неформальное (то есть словесное или даже образное) представление

«Нисходящее проектирование»



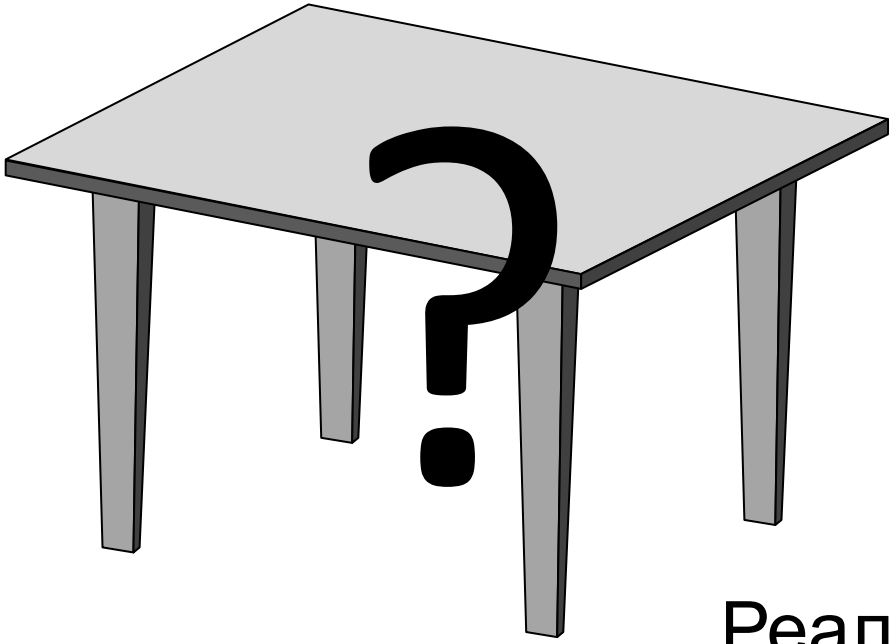
Задача: Нам нужно сделать стол.

Подзадачи:
1. Команда А делает крышку стола
2. Команда Б делает ножки стола
3. Команда В делает крепления ножек к крышке

Итог: Сбор стола

Реализация

«Восходящее проектирование»



Задачи:

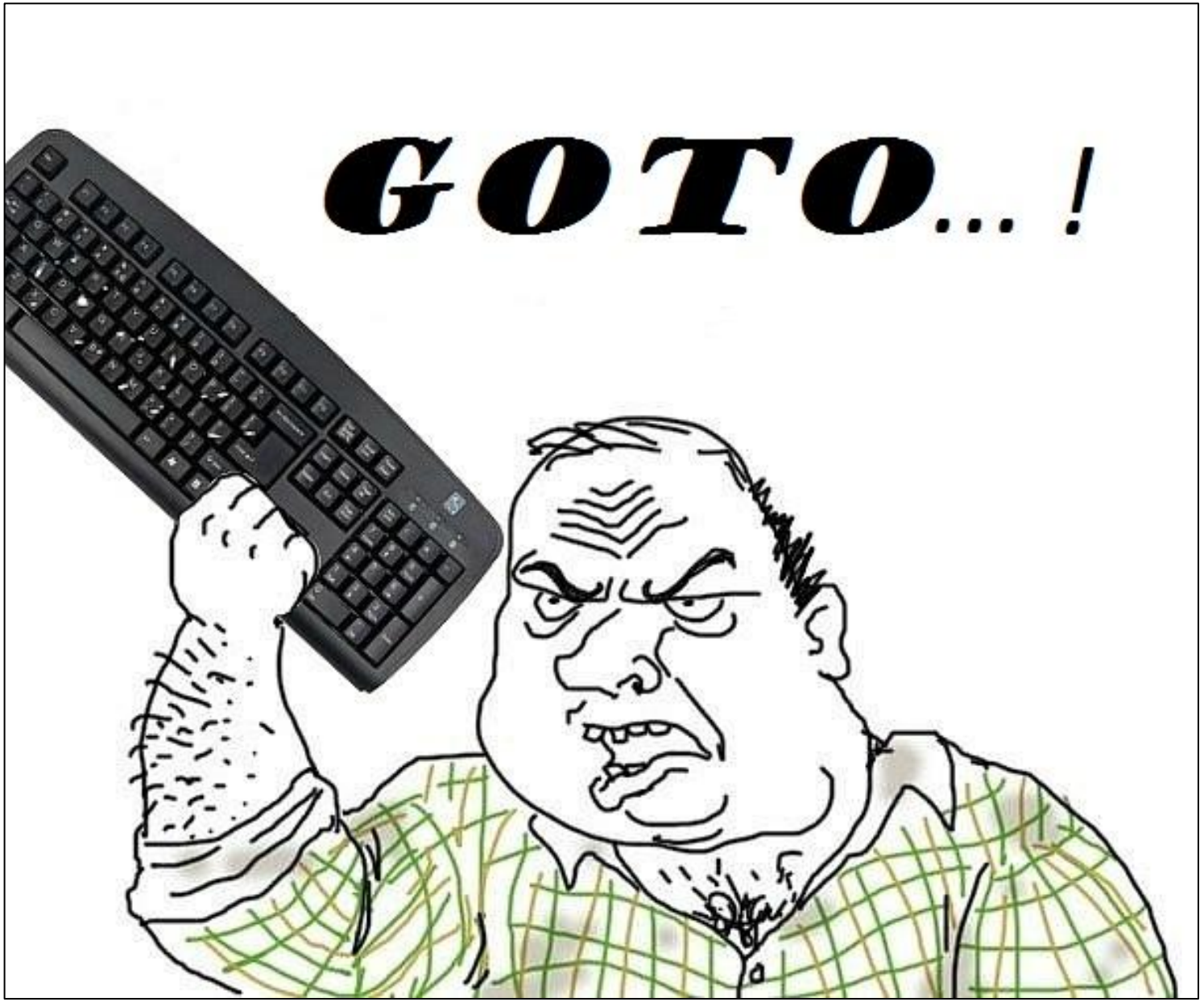
1. Команда А делает крышку стола
2. Команда Б делает ножки стола
3. Команда В делает крепления ножек к крышке

Итог: Сбор стола

Реализация

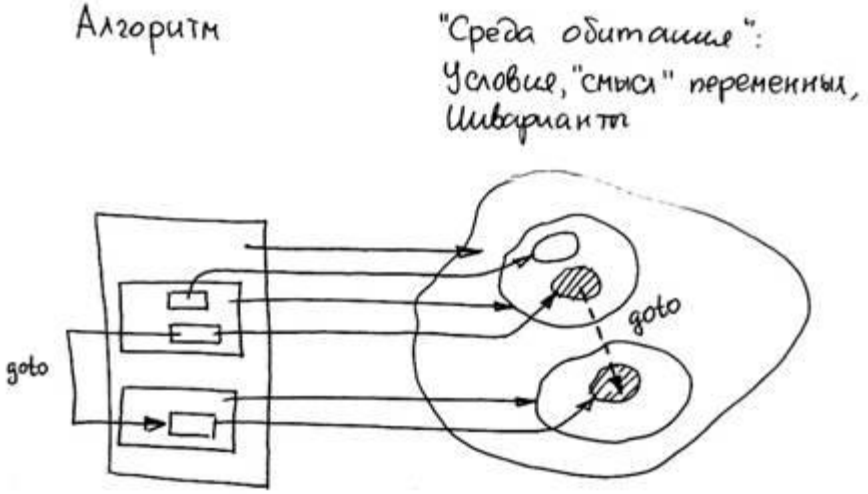
Криворукие идиоты, 6#@^%

«Нисходящее проектирование»



Проектирование без GOTO

«Среда обитания» программы. Каждая конструкция языка не просто встраивается в программу, а определяет свойства используемых ею данных, «смысл» переменных, которые появились в программе одновременно с ней. Поэтому при использовании исключительно вложенных конструкций мы получим в каждой точке программы определенный набор выполняемых условий, своего рода «среду обитания» алгоритма. Эти переменные являются исходными данными для очередного шага детализации алгоритма.



Проектирование без GOTO

Почему «программирование без GOTO»? Нисходящее пошаговое проектирование исключает использование оператора goto, более того, запрещает его применение как нарушающего структуру программы. GOTO страшен не тем, что «неправильно» связывает разные части алгоритма, а в том, что переводит алгоритм из одних «условий обитания» в другие: в точке перехода они составлены без учета того, что кто-то сюда войдет «не по правилам».

Чрезвычайными обстоятельствами, вынуждающими прибегнуть к помощи оператора goto, являются глобальные нарушения логики выполнения программы, например грубые неисправимые ошибки во входных данных. В таких случаях делается переход из нескольких вложенных конструкций либо в конец программы, либо к повторению некоторой ее части.

«Грязное проектирование»

Под «грязным» программированием обычно понимается написание программы, грубо воспроизводящей требуемое поведение. Такая программа может быть быстро разработана и отлажена, а затем использована для уяснения последующих шагов, либо для наложения «заплаток» с целью получения требуемого результата. Хотя этот «не есть хорошо» с точки зрения технологии проектирования, но может быть оправдано при следующих условиях:

- «грязная» программа воспроизводит требуемое поведение на самом верхнем уровне.
- в дальнейшем в нее могут встраиваться контексты и фрагменты, *не меняющие ее поведения*, но конкретизирующие ее в нужном направлении.

Базовые понятия программирования

Транслятор - программа, которая конвертирует программу, написанную на одном языке, в программу на другом языке так, чтобы обе давали идентичные результаты вычислений.

Трансляторы далее классифицируются на *ассемблеры, компиляторы и препроцессоры*.

Интерпретатор преобразовывает каждую инструкцию программы непосредственно в машинный код и немедленно выполняет ее перед переходом к следующей инструкции.

Базовые понятия программирования

Ассемблер транслирует программу, написанную на компоновочном языке (язык ассемблера/assembly language), в **машинный код**. Каждая команда в программе, написанной на ассемблере, имеет почти взаимно однозначное соответствие командам в машинном коде. Другими словами, код операции и операнд, из которых состоит каждая инструкция ассемблера, обозначаются читаемым именем (т. е. мнемоническим кодом операции) и десятичным числом, вместо двоичного представления соответству

Базовые понятия программирования

Компилятор - транслятор со сложной структурой, который преобразовывает программу, написанную на языке высокого уровня, в машинный код или, в некоторых случаях, в программу на ассемблере.

Препроцессор - языковый процессор, который выполняет предварительную трансляцию исходных кодов типа замены алфавитно-цифровых выражений двоичной формой и вставкой определенных файлов, создавая модификацию исходного текста, который должен быть далее обработан транслятором. Препроцессор также удобно использовать, когда новый язык высокого уровня создается путем добавления элементов к существующему языку высокого уровня. Например, программа на C++ может транслироваться в ее эквивалент на C препроцессором.

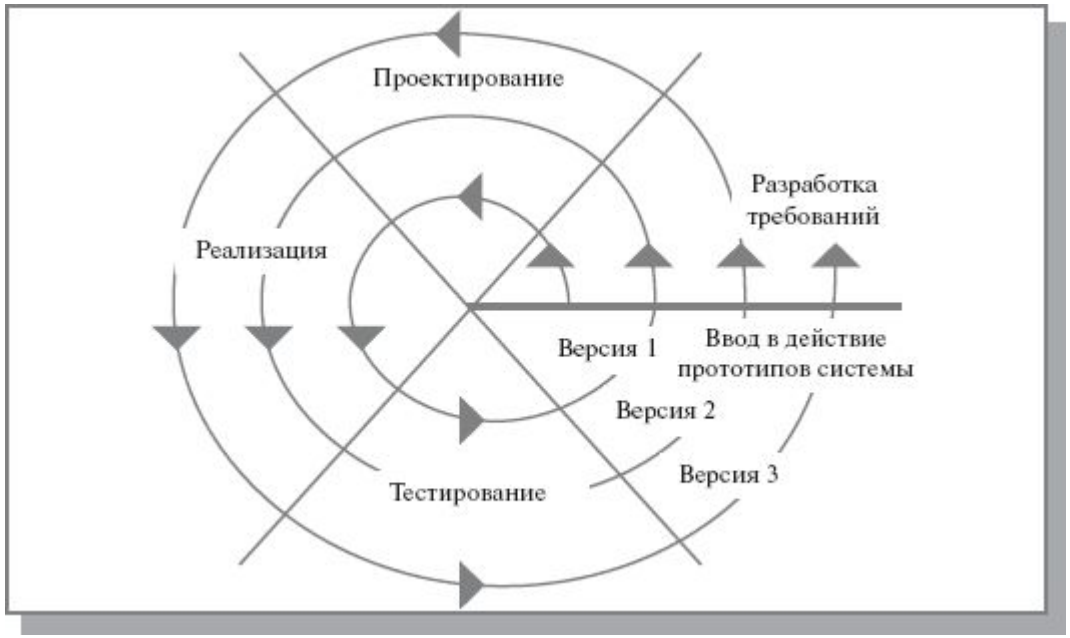
Базовые понятия программирования

Интерпретатор выполняет каждую инструкцию программы, поскольку она преобразована в машинный код без создания программы-цели, в то время как ассемблеры и компиляторы производят программы-цели без выполнения их. Интерпретаторы удобны для быстрого нахождения ошибок в программах, но не подходят для больших программ из-за низкого времени интерпретации.

Жизненный цикл программного обеспечения

Жизненный цикл включает в себя **шесть** этапов:

- анализ требований,
- определение спецификаций,
- проектирование,
- кодирование,
- тестирование,
- сопровождение



Мозгогвоздь

В наличии у студента Сидорчука Васи имеются сомбреро, балалайка, секундомер и электрический трансформатор. Необходимо определить высоту здания института.

Студенты групп первого курса института решили сыграть в игру «Сессия нахаляву». Для участия собралось 127 человек. В первом туре 126 студентов составят 63 пары, победители которых выйдут в следующий тур, и еще 1 студент выходит во второй тур без игры. В следующем туре — 64 игрока сыграют 32 матча и т.д. Сколько всего матчей понадобится, чтобы определить счастливчика-победителя?

Осуществите тестирование бутылки с кетчупом.

Супермегамозгогвозд

Жизнь забросила вас в центр ровной и совершенно круглой постапокалиптической пустыни. На ваше счастье она обнесена колючей проволокой, за которой бегает кибернетический пёс-терминатор.

Пес категорически вас ненавидит и хочет уничтожить, лишь колючая проволока мешает ему. Если вы сможете добежать до колючей проволоки и перелезть через нее, вы сможете сесть в вертолет и улететь, если только пёс не добежит до этой точки раньше вас.

Проблема в том, что скорость пса-терминатора превышает вашу ровно в 4 раза. Разумеется, у терминатора стопроцентное зрение, он никогда не спит и мыслит о вашей поимке очень логично. Он будет делать все возможное, чтобы поймать вас.

Как же вам спастись от пса-терминатора ?