# *Алгоритмизация.*

# *Процедурно-ориентированное программирование.*

*Вопросы:*

1. *Алгоритм.*
2. *Блок-схема.*
3. *Процедурное программирование*
4. *Структурное программирование*

# *Ключевые слова * Key words*

| | |
|---|---|
| *Алгоритм* | *Algorithm* |
| *Блок-схема* | *Flow chart* |
| *Модуль* | *Module* |
| *Процедура* | *Procedure* |
| *Процедурная декомпозиция* | *Procedure decomposition* |
| *Процедурное программирование* | *Procedure programming* |
| *Модульное программирование* | *Modular programming* |
| *Структурное программирование* | *Structured programming* |

# *Идея:*

*<u>Чтобы решить задачу, надо</u>*

- Разработать последовательность элементарных действий

  →разработать ***алгоритм***

- Каждое действие преобразовать в инструкции, понятные компьютеру

  → написать ***текст программы***

# *Алгоритм*

- точно определённая последовательность ***действий*** для решения задачи.

Страница из «Китаб ал джабр **ал Хорезми»,** старейшей арабской работы по алгебре



# *Процедура*

- законченная точно определённая последовательность ***операций*** для решения отдельной задачи.

```
Procedure Vvod_ID(var x,y:byte);
Begin
…
…
End;
```

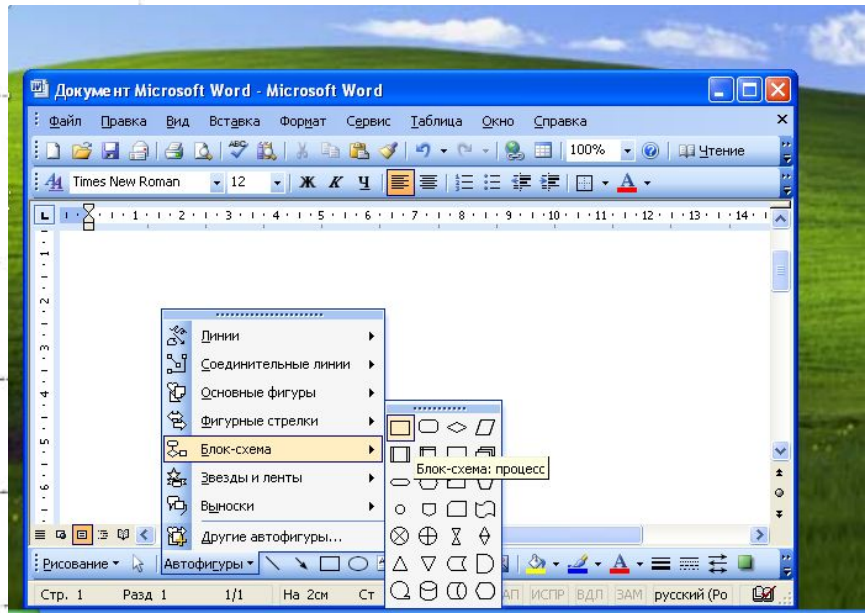
# Особенности процедурного (модульного) программирования

- *Процедурная декомпозиция* - разделение большой программы на отдельные части, *процедуры (модули)* - облегчает разработку, отладку и сопровождение программы.
- В свою очередь модуль также представляет собой совокупность процедур.
- Программа всегда имеет начальную процедуру и окончание.
- *Для начала действия последующей процедуры необходимо завершение всех действий предшествующей процедуры.*
- Вся программа может быть представлена в виде *направленной последовательности* графических блоков:

# Блок-схема алгоритма

(ГОСТ 19.701-90)

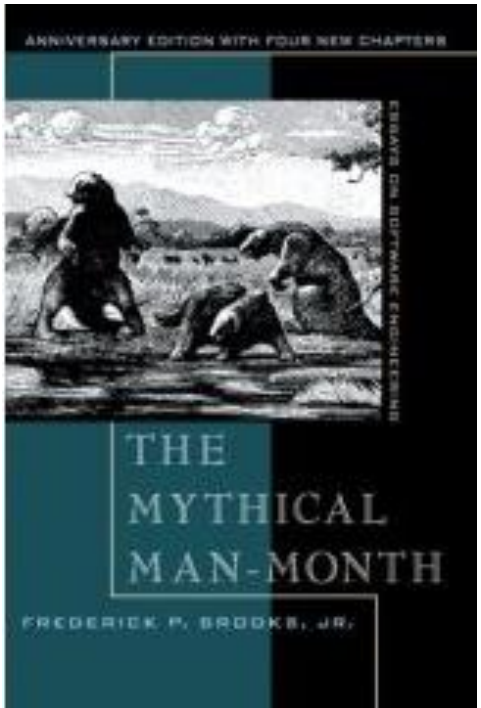
| Название блока | Обозначение | Назначение блока |
|---|---|---|
| Терминатор | Действие | Начало, завершение программы или подпрограммы |
| Процесс | Действие | Обработка данных (вычисления, пересылки и т. п.) |
| Данные | Данные | Операции ввода-вывода |
| Решение | Условие | Ветвления, выбор, итерационные и поисковые циклы |
| Подготовка | Действия | Счетные циклы |
| Граница цикла | Начало<br>Конец | Любые циклы |
| Предопределенный процесс | Имя | Вызов процедур |
| Соединитель | Имя | Маркировка разрывов линий |
| Комментарий | --- Комментарий | Пояснения к операциям |

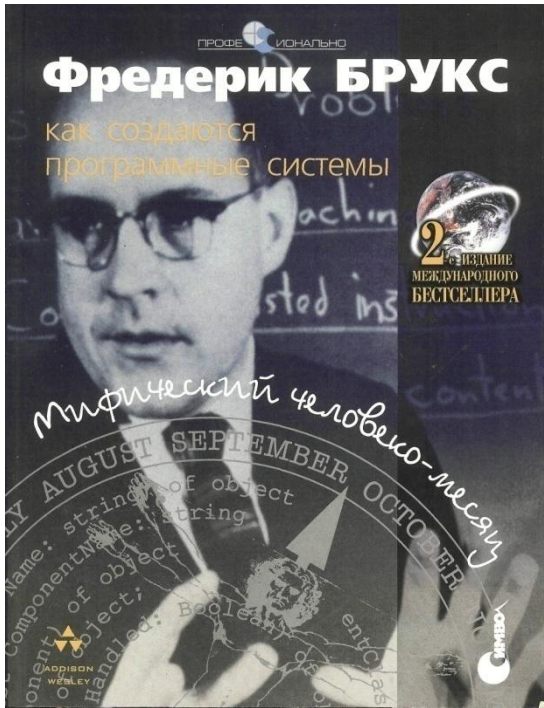


# Алгоритм функциональности галчонка

# Сложность программы = f (количество строк программного кода)?

Фредерик Брукс.
«Мифический человеко-месяц,
или Как создаются программные системы»



# Закон Брукса ☹️

**Глава 2. Мифический человеко-месяц**

2.1 Программные проекты чаще проваливаются из-за нехватки календарного времени, чем по всем остальным причинам, вместе взятым.

2.2 Чтобы приготовить вкусную пищу, нужно время; некоторые задачи нельзя ускорить, не испортив результат.

2.3 Все программисты являются оптимистами: "Все будет хорошо".

2.4 Поскольку программист работает с чистыми идеями, мы не ожидаем особых трудностей при реализации.

2.5 Но сами наши *идеи* бывают ошибочными - отсюда и ошибки в программах.

2.6 Наши методы оценивания, основанные на учете затрат, смешивают затраты с полученным результатом. *Человеко-месяц является ошибочным и опасным заблуждением, поскольку предполагает, что месяцы и количество людей можно менять местами.*

2.7 Разделение задачи между несколькими людьми вызывает дополнительные затраты на обучение и обмен информацией.

2.8 Мое практическое правило: 1/3 времени - на проектирование, 1/6 - на написание программы, 1/4 - на тестирование компонентов и 1/4 - на системное тестирование.

2.9 Как научной дисциплине нам не хватает методов оценки.

2.10 Поскольку мы не уверены в своих оценках сроков работы, нам часто не достает смелости упрямо отстаивать их под нажимом руководства и клиентов.

**2.11 *Закон Брукса: если проект не укладывается в сроки, то добавление рабочей силы задержит его еще больше.***

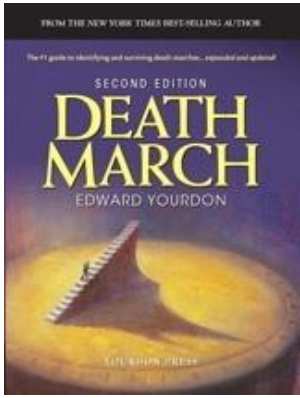
2.12 Добавление рабочей силы увеличивает общий объем затрат тремя путями: труд по перекраиванию задач и происходящее при этом нарушение работы, обучение новых людей, дополнительное общение.

# Закон Платта ☹

*«Любой проект по разработке ПО потребует **в три раза** больше времени, чем вы рассчитываете, даже если вы учитываете этот закон»*

Дэвид С.Платт,
*«Software Legend»*
по признанию *Microsoft*
в 2002г.


# «Death March» ☹




*Edward Yourdon. «Death March. The Complete Software Developers's Guide to Surviving "Mission Impossible" Projects»*

Эдвард Йордон. «Путь камикадзе. Как разработчику программного обеспечения выжить в безнадёжном проекте»



Эдвард Йордон - автор и соавтор более двух десятков книг, включая "Путь камикадзе", "Закат и падение американского программиста" ,"Подъем и возрождение американского программиста".

В июне 1977 г. он был официально объявлен членом Зала славы компьютеров, объединяющим таких выдающихся людей, как Чарльз Бэббидж, Сеймур Крей, Джемс Мартин, Грейс Хоппер, Джеральд Вайнберг и Билл Гейтс.

Широко известный как соавтор популярной методологии Коуда/Йордона, он создал и возглавил YOURDON - консалтинговую компанию, которая обучила уже более 250 000 человек во всем мире.

# Ветвления – «корень зла»?!

- Безусловный переход:
  GOTO *<номер строки/метка>*

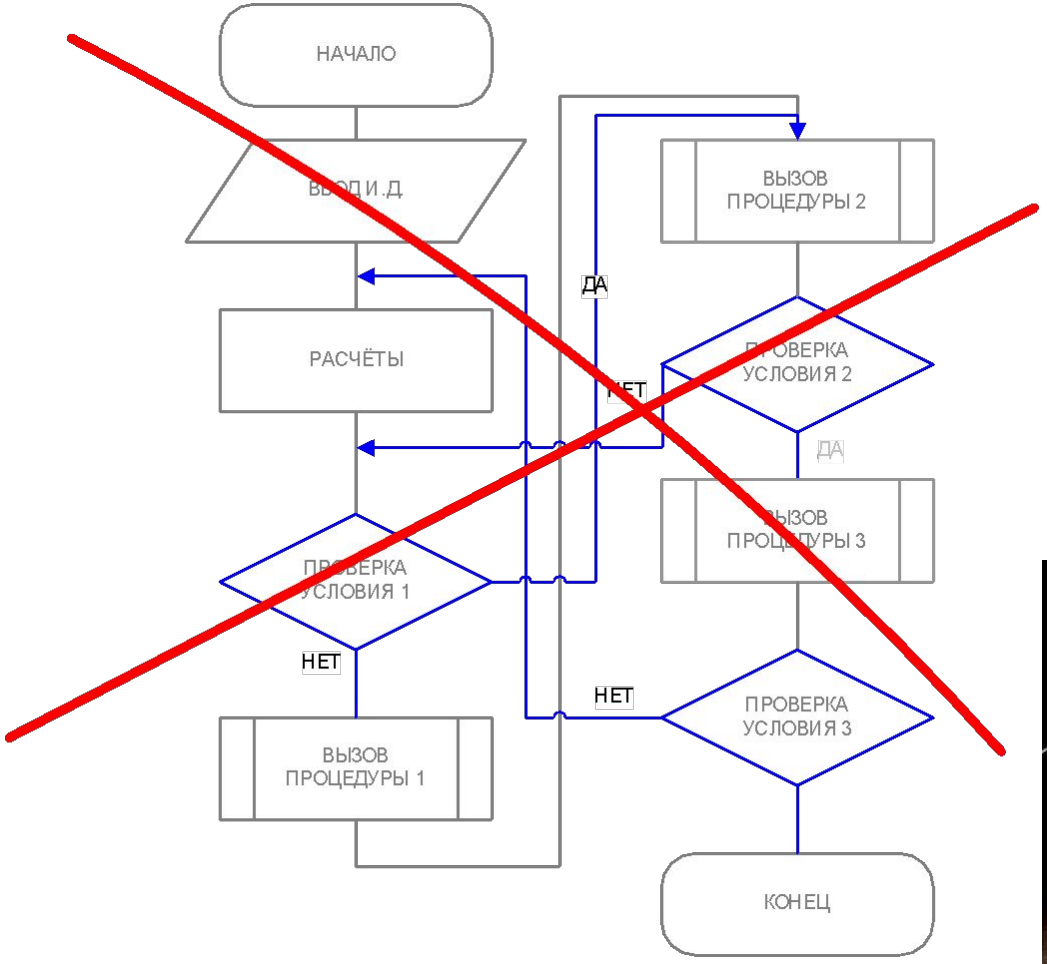
- Условный переход:
  IF *<условие>*
  THEN *<путь1>*
  ELSE *<путь2>*

- Цикл:
  DO  WHILE <условие>
    <тело цикла>
  ENDDO

  REPEAT
    <тело цикла>
  UNTIL <условие>

- Выбор варианта:
  DO CASE <выражение> OF
    <значение1>: *<путь1>*
    <значение2>: *<путь3>*
    …
  ENDDO

# За программирование без «GOTO»!



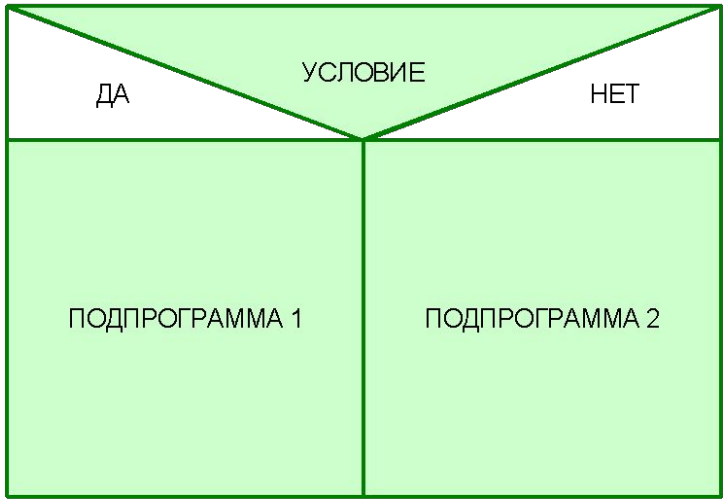

# *Структурное программирование*

- методология и технология разработки программных комплексов, основанная на принципах:
  - *нисходящего программирования;*
  - *модульного программирования.*

Эпоху Структурного Программирования начали «Заметки о структурном программировании» (1960) датчанина *Эдсгера Дейкстры*.

<u>Основа методологии</u>: процедурная декомпозиция *на всех уровнях* проектирования программной системы.

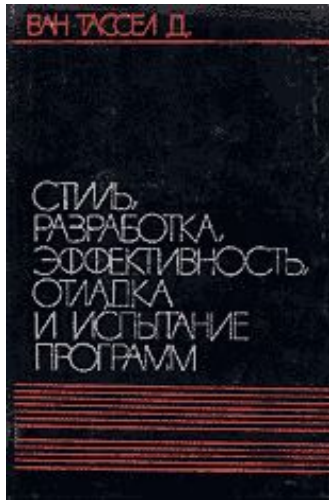
# «Структурирующая» блок-схема: вместо ветвления – линейный участок!





# Правила хорошего стиля!

1. *Старайтесь, чтобы имена переменных отражали смысл их содержимого*
2. *Не используйте одну и ту же переменную в разных смыслах*
3. *Пишите комментарии! Одна строка = один оператор + один комментарий*
4. *Выделяйте отступами блоки, вложенные циклы и условные операторы*
5. *Оптимальный по размерам модуль целиком должен помещаться на экране*
6. *Избегайте неявного преобразования типов данных*
7. *…*



# Расчёт сложных процентов: условие задачи

***Дано***: капитал **Q** вкладывается в предприятие с ежегодным приростом **D**%.

***Определить***: текущую величину капитала в течение первых **N** лет.

ТАК ВОТ ОНО КАКОЕ СЧАСТЬЕ!!!

# Расчёт сложных процентов: решение задачи

***Условие задачи.***
***Дано***: капитал **Q** вкладывается в предприятие с ежегодным приростом **D**%.
***Определить***: текущую величину капитала в течение первых **N** лет.

***Решение:***

Прирост D% означает увеличение капитала в (1+D/100) раз.

Введём коэффициент увеличения $D_{raz} = 1 + D/100$.

Тогда текущая величина капитала по итогам:

- 1-го года равняется $Q * D_{raz};$
- 2-го года равняется $Q * D_{raz} * D_{raz};$
- …;
- N-го года равняется $Q * D_{raz.} * D_{raz} * \ldots * D_{raz}$ (умножать N раз).

# Расчёт сложных процентов: BASIC-программа и блок-схема алгоритма

```
10 PRINT “Расчёт сложных процентов
20 INPUT “Введите Q, D, N”, Q, D, N
30 D_RAZ=1+D/100
40 J=1
50 Q=Q*D_RAZ
60 PRINT J,Q
70 J=J+1
80 IF J<=N THEN GOTO 50
90 END
```