Лекция 06
БГТУ, ФИТ, 1 семестр

Основы программной инженерии (ПОИТ)

Стили программирования. Структурное программирование

План лекции:
- парадигмы программирования;
- императивное программирование;
- декларативное программирование;
- структурное программирование.

1. На прошлых лекциях:

> **Система программирования:** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки. Каждая система программирования должна иметь некоторый встроенный в нее язык программирования, предназначенный для общения разработчика с используемыми инструментами.

> **Интегрированная среда разработки (integrated development environment – IDE):** – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

Примеры IDE (визуальные среды):
Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator, ...

Структура языка программирования:

Стандарты языков программирования

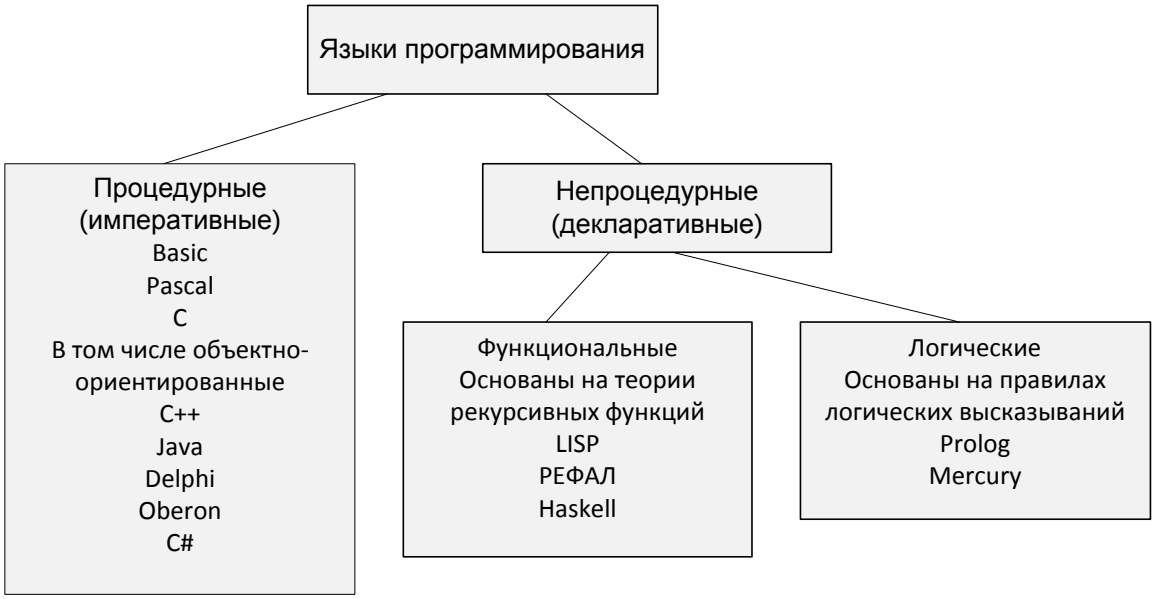
Язык программирования обычно представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

Для многих широко распространенных языков программирования созданы ***международные стандарты***.

Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка. В рамках таких комитетов продолжается разработка и модернизация языков программирования, решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

Стандарт языка программирования: Visual C++ 2020 доступно в Visual Studio начиная с версии 16.11 – это реализация стандарта C++20 или **ISO/IEC 14882:2020**.

2. Парадигмы (стили) программирования



Язык программирования строится в соответствии с базовой моделью вычислений и парадигмой программирования.

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

Языки классифицируют по важнейшим признакам:

- ***эволюционным*** – поколения языков (1GL, 2GL, 3GL, 4GL, 5GL, ...);
- ***функциональным*** – по назначению, исполняемым функциям (описательные, логические, математические);
- ***уровню языка*** – то есть уровню обобщения в словах-операторах языка (низкого, среднего, высокого, ...);
- ***области применения*** –системные, сетевые, встроенные и пр.

Поколения языков (Generations of Languages)
Каждое из последующих поколений по своей функциональной мощности качественно отличается от предыдущего.

- 1GL - первое поколение: ***Машинные языки***. Появились в середине 40-х годов XX века.
- 2GL - второе поколение: ***Ассемблеры***. Фактически это те же машинные языки, но более красиво «обернутые». Появились в конце 50-х годов XX века
- 3GL - третье поколение: ***Процедурные языки***. Появились в начале 60-х годов XX века. К этому поколению относят универсальные языки высокого уровня, с помощью которых можно решать задачи из любых областей (например, Algol-60).
- 4GL - четвертое поколение: ***Языки поддержки сложных структур данных*** (например, SQL). Появились в конце 60-х годов XX века.
- 5GL - пятое поколение: ***Языки искусственного интеллекта*** (например, Prolog). Появились в начале 70-х годов XX века.
- 6GL - шестое поколение: ***Языки нейронных сетей*** (самообучающиеся языки). Исследовательские работы в этой области начались в середине 80-х годов XX века.

Классификация языков по поддерживаемым методологиям появилась примерно в 80-х годах XX века.

Пример функции возведения в квадрат в некоторых языках программирования.

Императивный **C**:	Функциональный **Scheme**:

int square(int x)
{
  return x * x;
}
	(define square
	(lambda (x)
	(* x x)))

Конкатенативный **Joy**:	Конкатенативный **Factor**:
DEFINE square == dup * .	: square (x -- y) dup *;

Стиль (парадигма) программирования определяет базовые концепции языков программирования и их сочетания.

Методология включает в себя модель вычислений для данного стиля.

Методология разработки программного обеспечения – совокупность методов, применяемых на различных стадиях жизненного цикла программного обеспечения.

Стили программирования естественно классифицируются по трем признакам:

низкоуровневые языки;

высокоуровневые языки;

глобальность либо локальность действий и условий.

Неструктурное программирование характерно для наиболее ранних языков программирования. Сложилось в середине 40-х с появлением первых языков программирования.

Основные признаки:

- строки как правило нумеруются;
- из любого места программы возможен переход к любой строке;

Выделяют четыре базовых стиля программирования.

1. Структурное программирование (действия и условия локальны);
2. Программирование от состояний (действия глобальны, условия локальны);
3. Программирование от событий (действия локальны, условия глобальны);
4. Сентенциальное программирование (действия и условия глобальны).

В программировании от ***состояний*** процесс представляется как смена состояний системы.

Новое состояние возникает в результате действия, изменяющего старое состояние.

Выбор этого действия зависит от проверки условий.

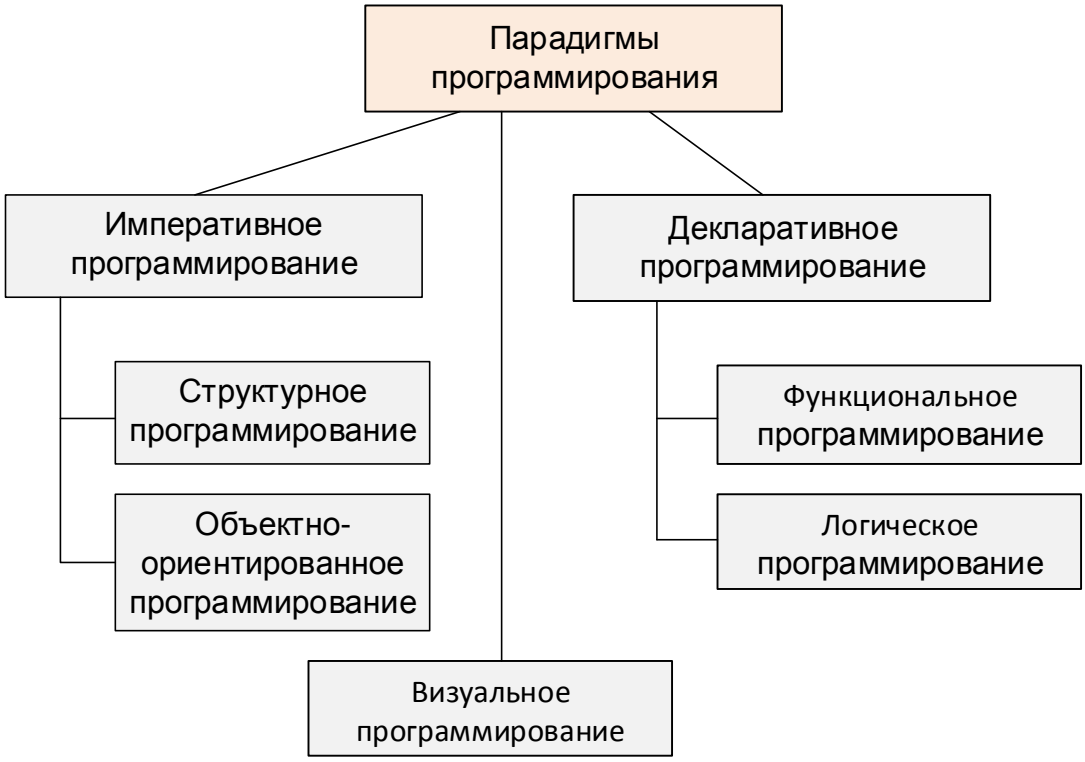
Математической моделью программы служит конечный автомат.

Разрешено использование оператора безусловного перехода goto либо использование объектов, обменивающихся информацией через общее поле памяти.

В ***сентенциальном*** стиле каждый шаг программы проверяет весь объем текстовой информации на соответствие образцу, находит и, согласно найденному правилу преобразования, производит преобразование.

В программировании от ***событий*** условие состоит в том, что в системе произошло некоторое событие, которое влечет обработку данного действия обработчиком события.

3. Основные парадигмы программирования:



императивное	программа = последовательность действий, связанных условными и безусловными переходами
процедурное	программа = последовательность процедур, каждая из которых есть последовательность элементарных действий и вызовов процедур, структурированных с помощью структурных операторов ветвления и цикла
объектно-ориентированное	программа = несколько взаимодействующих объектов, функциональность (действия) и данные распределяются между этими объектами
функциональное	программа = система определений функций, описание того, что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается
логическое	программа = система определений и правил вида «условие => новый факт»
сентенциальное	программа = система правил вида «шаблон => трансформирующее действие»
событийное	программа = система правил вида «событие => новые события» + диспетчер событий
автоматное	программа = конечный автомат или автомат специального типа

Типы входных данных в различных парадигмах

Тип входных данных	Парадигмы программирования
аргументы	функциональное
глобальные	автоматное, сентенциальное, продукционное
локальные	(создание временных локальных объектов используется во многих стилях)
события	событийное

Организация хранилища

Парадигмы программирования	структура хранимых данных	тип императива
автоматное	***состояние***	диаграмма переходов
логическое	***факты*** (n-арные отношения)	правила вида (логическое условие → новые факты)
сентенциальное	***текст*** (произвольные данные, обычно записанные на некотором формальном языке)	правила вида (шаблон → трансформация)

Пример: телефон имеет следующие режимы (макро-состояния):

- ожидание звонка
- входящий звонок
- установлено соединение (идет разговор)
- просмотр телефонной книжки
- навигация по меню

Поток ***входных данных*** — это последовательность нажимаемых клавиш и принимаемые (в параллельном режиме) телефоном сигналы.

Глобальными переменными (памятью) являются все настройки, адресная книга, списки вызовов, SMS сообщения, флаги (информация) о пропущенных вызовах или полученных SMS сообщениях и др.

[Множество ***состояний*** телефона] =
[Множество режимов] × [Множество состояний памяти].

Проектирование системы заключается в следующем:

- выделение базовых сущностей (функций, объектов, состояний);
- установление взаимных связей (распределении ответственности – функциональности) между этими сущностями;
- определение, какие данные в каких функциях (объектах, состояниях) будут видны (доступны для чтения и модификации).

Императивное программирование — программирование от «глаголов». Программы представляют собой последовательность действий с уловными и безусловными переходами.
Программист *мыслит* в терминах действий и выстраивает последовательности действий в более сложные макро-действия (процедуры).

Пример:

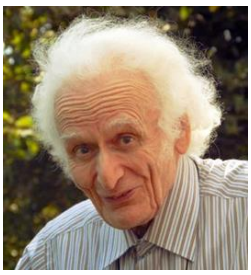
```
Procedure Вскипятить_чайник

begin
  Зажечь плиту;
  Взять чайник;
  Налить в чайник воды;
  Поставить на плиту;
  Подождать 5 минут;
end
begin
  if Чайник не пуст then
    Вылить из чайника воду;
  Вскипятить_чайник;
end.
```

Базовые признаки языка программирования и стиля:

- тип локализации (видимости) данных:
  - данные глобальны (Global);
  - данные локальны (Local);
  - данные передаются как аргументы функции (Flow);
  - данные отсутствуют (NONE);
- структура хранимых данных:
  - хранилище фактов;
  - хранилище объектов (переменные, структуры данных, объекты);
  - состояние;
- принцип доступа к данным:
  - именованный;
  - адресный;
  - сложный (по запросам):
    - SQL (Relational DBs);
    - Logical queries (Prolog);
  - Линейный:
    - FIFO (очередь);
    - FILO (стек);
- метод описания логики работы программы:
  - действия + условные переходы;
  - математические зависимости функций друг от друга;
  - диаграмма переходов;
  - правила вида «логическое условие → изменение данных»;
  - правила вида «шаблон → изменение данных»;
- принцип взаимной организации императивных и декларативных данных:
  - инкапсуляция в одном (объектно-ориентированный подход);
  - декларативных данных как таковых нет, есть аргументы, поступающие на вход функциям (функциональный подход);
  - простая логика действий привязана к состояниям глобальных данных;
  - ....

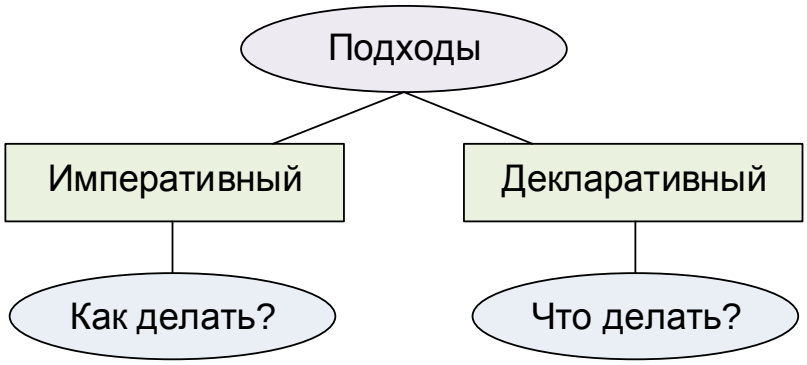
Программировать в различных стилях программирования можно в рамках одного о того же языка.


Коррадо Бём

Концепция структурного программирования основана на теореме Бёма-Якопини: любая вычислимая функция может быть представлена комбинацией трёх управляющих структур:

- последовательности действий;
- ветвления;
- повторения (итерации).

Основные подходы к программированию:



Императивное программирование (от греч. imper — действие) предполагает, что программа явно описывает алгоритм решения конкретной задачи (действия исполнителя), т.е. описывает как решать поставленную задачу.

В основе императивного программирования находятся два основных понятия:
– алгоритм (отсюда название алгоритмические языки)
– архитектура ЭВМ, основанная на модели вычислений фон Неймана.

Допускается выполнение действий над данными определённого типа.

- используются допустимые для данного типа операции и функции:
  - арифметические;
  - логические;
  - символьные;
  - строковые.

Управление ходом исполнения алгоритма.

- Используются операторы, реализующие основные управляющие структуры:
  - следование;
  - ветвление;
  - цикл;
  - вызов подпрограммы (возможно).

Декларативное программирование (лат. declaratio – объявление, подход возник в 60-х годах) – это предварительная реализация «решателя» для целого класса задач.
Тогда для решения конкретной задачи этого класса достаточно декларировать в терминах данного языка только её условие:
(исходные данные + необходимый вид результата)
«Решатель» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.

Пример. Найти сумму элементов массива.

Псевдокод:	***Псевдокод:***

// императивная парадигма
`let array = [1, 2, 3, 4, 5, 6];`
`let result = 0`
`for (let i = 0; i < array.length; i++)`
`result += array[i];`
	// декларативная парадигма
	`let array = [1, 2, 3, 4, 5, 6];`
	`array.reduce((prev, current) =>`
	`prev + current, 0)`

В ***императивной*** парадигме разработчик пишет для компьютера инструкции, которым тот следует. Инструкции будут примерно следующие:

- определить массив `array` из 6 элементов с заданными значениями;
- определить переменную `result` и присвоить ей значение 0;
- определить индекс цикла ***i*** со значением 0;
- начало цикла;
- добавить к переменной `result` элемент массива `array[i]`;
- прибавить к переменной i единицу;
- повторять цикл, пока значение переменной i меньше количества элементов массива `array`;
- конец цикла;

То есть, программист говорит, что нужно сделать (программирование от глаголов) и в каком порядке, а компьютер выполняет приказы.

В ***декларативной*** парадигме программист просто пишет следующее:

- дан массив array из 6 элементов с заданными значениями;
- получить сумму элементов массива array.

Приведенный в примере метод используется для последовательной обработки каждого элемента массива с сохранением промежуточного результата. Здесь
`prev` – последний результат вызова функции (промежуточный результат).
`current` – текущий элемент массива, элементы перебираются по очереди слева-направо.
При первом запуске `prev` – исходное значение, с которого начинаются вычисления и оно равно нулю.

4. Структурное программирование

Термин структурное программирование ввёл Э.Дейкстра в 1975 году:

- Э.Дейкстра «Заметки по структурному программированию» (в составе сборника «Структурное программирование» / М.: Мир, 1975.

Структурное программирование – стиль написания программ без ***goto***.
В структурном программировании необходимо:
составить правильную логическую схему программы;
реализовать ее средствами языка программирования;
Программа – множество вложенных блоков (иерархия блоков), каждый из которых имеет один вход и один выход.
Передача управления между блоками и операторами внутри блока (на каждом уровне дерева) выполняется последовательно.

Технология структурного программирования – нисходящее проектирование, т.е. от общего к частному, от внешней конструкции к внутренней.

Теоретическим фундаментом структурного программирования является теорема о структурировании Бёма-Якопини.

Структурное программирование характеризуется:

- ограниченным использованием условных и безусловных переходов;
- широким использованием подпрограмм и управляющих структур (циклов, ветвлений, и т.п.);
- блочной структурой.

Любая вычислимая конструкция может быть представлена комбинацией трёх управляющих структур:

- последовательности;
- ветвления;
- повторения (итерации).

Последовательность – это последовательное выполнение инструкций/блоков.
Ветвление – это выполнение либо одной, либо другой инструкции/блока в зависимости от значения некого булева (логического) выражения.
Итерация – это многократное выполнение инструкции/блока пока некое булево выражение истинно.

Структурное программирование – методология и технология разработки программных средств, основанная на трёх базовых конструкциях:

- следование;
- ветвление;
- цикл.

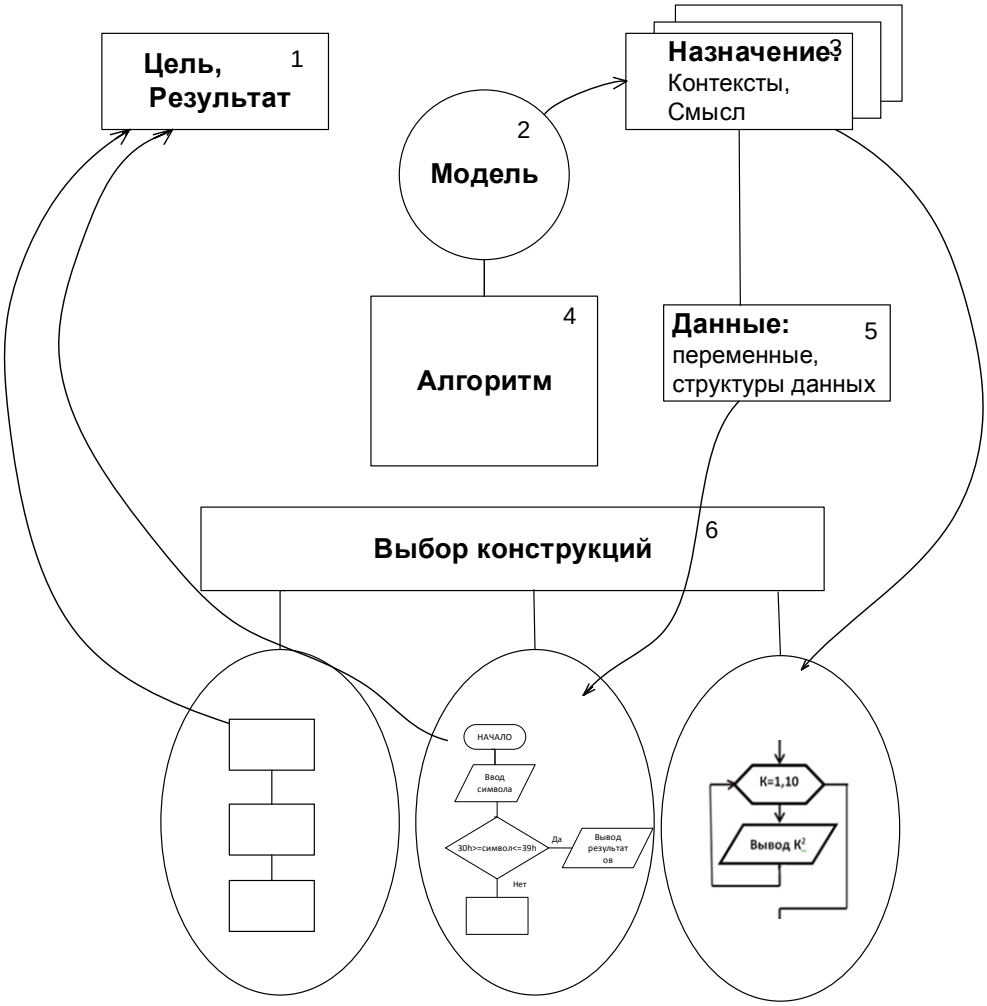
Принципы разработки:

- программирование «сверху-вниз» (нисходящее программирование);
- модульное программирование с иерархическим упорядочением связей между модулями/подпрограммами «От общего к частному»

Этапы проектирования:

- формулировка целей (результатов) работы программы;
- представление процесса работы программы (модель);
- выделение из модели фрагментов: определение переменных и их назначения, стандартных программных контекстов.

Этапы структурного проектирования



1. Исходным состоянием процесса проектирования является формулировка цели алгоритма, или результата, который должен быть получен при его выполнении. Формулировка производится на естественном языке.
2. Создается модель происходящего процесса, используются графические или другие способы представления (например, образные изображения, позволяющие лучше понять выполнение алгоритма в динамике).
3. Выполняется сбор информации, касающейся характеристик алгоритма. Например, наличие определенных переменных и их «смысл», а также соответствующие им программные контексты.
4. В модели выделяется наиболее существенная часть, для которой строится алгоритм (в одной из форм представления).
5. Определяются переменные, необходимые для формального представления алгоритма и формулируется их назначение.
6. Выбирается одна из конструкций - *простая последовательность действий, условная конструкция* или *цикл*.
7. Для вложенных конструкций необходимо вернуться на предыдущие этапы проектирования.

Технология структурного программирования базируется на следующих методах:

- нисходящее проектирование;
- пошаговое проектирование;
- структурное проектирование (программирование без goto);
- одновременное проектирование алгоритма и данных;
- ***модульное проектирование;***
- ***модульное, нисходящее, пошаговое тестирование.***

Нисходящее проектирование программы состоит в процессе формализации от самой внешней синтаксической конструкции алгоритма к самой внутренней; в движении от общей формулировки алгоритма к частной формулировке, составляющей его действия;

Структурное проектирование заключается в замене словесной формулировки алгоритма на одну из синтаксических конструкций – *последовательность*, *условие* или *цикл*.
Использование оператора безусловного перехода goto запрещается из принципиальных соображений;

Пошаговое проектирование состоит в том, что на каждом этапе проектирования в текст программы вносится только одна конструкция языка.

Одновременное проектирование алгоритма и структур данных. При нисходящей пошаговой детализации программы необходимые для работы структуры данных и переменные появляются по мере перехода от неформальных определений к конструкциям языка, то есть процессы детализации алгоритма и данных идут параллельно.

Цель (результат) = действие + цель (результат) вложенной конструкции.

Последовательность действий, связанных результатом.

- представить действие в виде последовательности шагов;
- между различными шагами существуют связи через общие переменные, результат выполнения шага и последующие шаги используют этот результат.

Программирование без goto.
Операторы continue, break и return: служат для более «мягкого» нарушения структурированной логики выполнения программы:

- **continue** – переход завершающую часть цикла;
- **break** – выход из внутреннего цикла;
- **return** – выход из текущего модуля (функции).