Языки программирования

Литература:

1. C++, Turbo Pasckal, QBasik: Эволюция языков программирования langprog.far/historylangprog.html.
2. Информатика/Курносов А.П., Кулев С.А., Улезько А.В. и др.; Под ред. А.П. Курносова. М., 2005.
3. Макарова Н.В. Информатика /под ред. Проф. Н. В. Макаровой. М.,2000.
4. Немнюгин С.А. Turbo Pascal. Программирование на языке высокого уровня: Учебник для вузов. СПб., 2008.
5. Островский В.А. Информатика: учеб. для вузов. М., 2000.

Немного истории: поколения языков

В середине 50-х годов вычислительные машины получили широкое распространение в университетах и научно-исследовательских институтах США и Западной Европы, и тогда же наступило время стремительного прогресса в области программирования.

1 этап

1943 – 1945 гг. – появление первого в истории языка программирования высокого уровня ***Plankalkül***, созданного немецким инженером Конрадом Цузе.

1950-ые гг. – появление языка ***Алгол.***

1957 г. – ***Fortran (FORmula TRANslator), LISP***.

В период **1960 — 1970 гг.** были разработаны основные парадигмы языков программирования, используемые в настоящее время.

1-ый этап - этап операционного программирования.

Период развития ЭВМ 1-ого поколения. ЭВМ «понимают» только цифровые команды, и программы состояли из множества строк, содержащих цифры, интерпретируемые центральным процессором.

Например, команда **05 825 631** трактовалась как сложение двух чисел (код **05**), записанных в ячейки с номерами **825** и **631**.

С появлением ЭВМ 2-ого поколения появились языки программирования типа Ассемблер и автокод. Команда сложения записывается с использованием служебных слов – **ADD** (сложить) **PR1**, **ZET**, где ADD – код команды, PR1, ZET – имена ячеек.

Перевод программы (трансляция), записанных таким образом в цифровое представление, осуществляется с помощью специальных программ, называемых ассемблерами.

Принцип работы: программа собирается из мелких деталей, отдельных операций и имеет достаточно простую структуру.

Решаемые задачи - расчетные.

2 этап

1964 г. – язык ***APL* (*A Programming Language)*** оказал влияние на функциональное программирование и стал первым языком, поддерживавшим обработку массивов. Язык ***PL/1 (Programming Language 1)*** был создан для научных, инженерных и бизнес-ориентированных вычислений.

Язык ***Симула*** впервые включал поддержку *объектно-ориентированного программирования*.

В середине **1970-х гг.** группа специалистов представила язык ***Smalltalk***, который был уже всецело объектно-ориентированным.

В период с **1969** по **1973 гг.** велась разработка языка ***C***, популярного и по сей день.

В **1972 г.** был создан ***Пролог*** — первый язык *логического* программирования.

В **1978 г.** в языке ***ML*** была реализована расширенная система полиморфной типизации, положившая начало типизированным языкам функционального программирования.

Развиваются языки программирования высокого уровня.

В них реализуются новые идеи: *подпрограммы* и *раздельная компиляция* (***Fortran 2***); блочная структура и типы данных (***Алгол 60***); описание данных и работа с файлами (***Кобол***); обработка списков и указателей (***LISP***).

В следующих версиях языков продолжается развитие: ***PL/1*** (Фортран+Кобол+Алгол), ***Паскаль*** (развитие Алгол 60), ***Симула*** (классы и абстрактные данные).

Реализуются технологии *нисходящего* и *восходящего* программирования.

Нисходящее программирование

Принцип нисходящего программирования – разбиение большой задачи на подзадачи, которые могут рассматриваться отдельно.

Основные правила применения данной технологии:

- формализованное и строгое описание входов функций и выходов всех модулей программы и системы;
- согласованная разработка структур данных и алгоритмов;
- ограничение на размер модулей.

При этом процесс нисходящей разработки программы может продолжаться до тех пор, пока не будет достигнут уровень базовых конструкций.

К нисходящей технологии следует отнести и то, что называется ***модульным программированием***. Достаточно независимые фрагменты задачи оформляются как модули.

Создаются библиотеки модулей, определяется механизм включения модулей в разрабатываемую программу. Модуль должен иметь строго определенный интерфейс и скрытую часть, одну точку входа и одну точку выхода.

Восходящая технология конструирования программ – решение складывается из «отдельных кирпичиков», из известных решений подзадач.

Данной технологией оговаривается определенный принцип декомпозиции и иерархическая структура программы.

Важнейшей составляющей этой технологии является ***структурное программирование*** (языки программирования ***Паскаль***, ***Модула-2***).

Пионером структурного программирования считается Эдсгер Вибе Дейкстра, который в 1965 г. предположил, что оператор GOTO может быть исключен из языков программирования.

Характерные черты структурного стиля программирования:

- простота и ясность (программа легко читается и анализируется);
- использование только базовых конструкций;
- отсутствие сетевых структур в программе;
- отсутствие многоцелевых функциональных блоков;
- отсутствие сложных арифметических и логических конструкций;
- расположение в строке программы не более одного оператора языка программирования;
- содержательность имен переменных.

3 этап

В **1960 — 1970-х гг.** активно велись споры о необходимости поддержки *структурного программирования* в тех или иных языках.

1980-ые гг. - период консолидации. Язык ***C++*** объединил в себе черты объектно-ориентированного и системного программирования Был стандартизирован язык ***Ада***, производный от ***Паскаля*** и предназначенный для использования в бортовых системах управления военными объектами.

Изучаются перспективы так называемых языков пятого поколения, которые включают в себя конструкции логического программирования. Функциональные языки приняли в качестве стандарта языки ***ML*** и ***LISP***.

В **1990- гг.** в связи с активным развитием Интернета распространение получили языки, позволяющие создавать сценарии для веб-страниц — главным образом ***Perl***, развившийся из скриптового инструмента для Unix-систем, и ***Java.***

Языки программирования

Язык программирования – это формальный язык для записи алгоритмов в виде, допускающем их автоматическую подготовку к выполнению на компьютере.

Язык программирования - это система обозначений, служащая для точного описания программ или алгоритмов для ЭВМ. Языки программирования являются искусственными языками. От естественных языков они отличаются ограниченным числом “слов” и очень строгими правилами записи команд (операторов).

Для преобразования программы в машинный код служит специальной программное средство – **транслятор.**

Трансляторы делятся на две группы по их функционалу: ***компиляторы*** и ***интерпретаторы***.

Интерпретатор преобразует команды исходного текста программы в машинные команды и немедленно их выполняет.

Особенности со знаком «+»	**Особенности со знаком «-»**
Программа выполняется по строкам исходного текста, что позволяет проверить правильность написания программы с точки зрения синтаксиса языка. При обнаружении ошибок появляется сообщение об ошибке, исполнения программы приостанавливается. Основное достоинство – результаты выполнения программы сразу же видны	**Интерпретация программ довольно медленный процесс. Заметно усложняется в случае, если программа состоит из нескольких модулей.** **Для запуска созданной программы на конкретном компьютере необходимо наличие собственно программы-интерпретатора.**

Компилятор просматривает текст программы (иногда несколько раз) и создает последовательность данных, которая называется ***объектным кодом.***

Объектный код еще не является полным аналогом программы. Необходим дополнительный этап, который называется *редактированием связей, или компоновкой.* На этом этапе происходит объединение объектного кода программы и объектного кода подпрограмм, взятых из внешних библиотек.

Результатом этого этапа является так называемый *исполнимый код* – он представляет собой набор машинных команд, реализующих алгоритм, записанный в программе.

Исполнимый код может запускаться автономно на любом компьютере подходящей платформы. Операцию компоновки кода выполняет отдельная программа, которая называется ***редактором связей*** или ***компоновщиком***.

В современных системах программирования компоновщик часто объединяют с компилятором, так что для пользователя оба этапа выглядят как один. Подобный двухступенчатый процесс облегчает создание больших и сложных программ.

Основные требования, предъявляемые к языкам программирования:

наглядность - использование в языке по возможности уже существующих символов;

единство - использование одних и тех же символов для обозначения одних и тех же или родственных понятий в разных частях алгоритма. Количество этих символов должно быть по возможности минимальным;

гибкость - возможность относительно удобного, несложного описания распространенных приемов математических вычислений с помощью имеющегося в языке ограниченного набора изобразительных средств;

модульность - возможность описания сложных алгоритмов в виде совокупности простых модулей, которые могут быть составлены отдельно и использованы в различных сложных алгоритмах;

однозначность - недвусмысленность записи любого алгоритма. Отсутствие ее могло бы привести к неправильным ответам при решении задач.

Любой алгоритм, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату.

В зависимости от *степени детализации предписаний,* как правило, определяется уровень языка программирования — чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

- машинные;

- машинно-ориентированные (*ассемблеры*);

- машинно-независимые (***языки высокого уровня***).

Машинные языки и ***машинно-ориентированные языки*** — это языки **низкого уровня**, требующие указания мелких деталей процесса обработки данных.

Языки же высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Разные типы процессоров имеют разные наборы команд.

Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он называется языком программирования низкого уровня.

Здесь имеется в виду, что операторы языка близки к машинному коду и ориентированы на конкретные команды процессора.

Языки низкого уровня

Язык ассемблера — это машинно-зависимый язык низкого уровня, в котором короткие мнемонические имена соответствуют отдельным машинным командам. Используется для представления в удобочитаемой форме программ, записанных в машинном коде.

Язык ассемблера позволяет программисту пользоваться текстовыми кодами, по своему усмотрению присваивать символические имена регистрам компьютера и памяти, а также задавать удобные для себя способы адресации.

Кроме того, он позволяет использовать различные системы счисления (например, десятичную или шестнадцатеричную) для представления числовых констант, использовать в программе комментарии и др.

Языки высокого уровня

Языки высокого уровня - были разработаны для того, чтобы освободить программиста от учета технических особенностей конкретных компьютеров, их архитектуры. Уровень языка характеризуется степенью его близости к естественному, человеческому языку.

Важным преимуществом языков высокого уровня является их универсальность, независимость от ЭВМ. Программа, написанная на таком языке, может выполняться на разных машинах.

При переходе на другую ЭВМ программа не требует переделки.

Программа, написанная на языке высокого уровня, легко может быть понята любым специалистом, который знает язык и характер задачи.

Преимущества языков высокого уровня

- алфавит языка высокого уровня значительно *шире* алфавита машинного языка, что существенно повышает наглядность текста программы;
- набор операций, допустимых для использования, *не зависит от набора машинных операций*, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- формат предложений достаточно *гибок и удобен для использования*, что позволяет с помощью одного предложения задать достаточно содержательный этап обработки данных;
- требуемые операции задаются с помощью *общепринятых математических обозначений*;
- данным в языках высокого уровня присваиваются *индивидуальные имена*, выбираемые программистом;
- в языке может быть предусмотрен значительно более *широкий набор типов данных* по сравнению с набором машинных типов данных.

Состав языка

Разговорный язык	Язык программирования
Состав: -**алфавит -** конечный набор графических символов (букв), используемый в определенной *системе письма* для передачи элементов звуковой речи (ГОСТ 7.79—2000) **;** **- лексика** –словарный состав языка **;** **- грамматика -** изучает закономерности построения правильных осмысленных речевых отрезков на этом языке (словоформ, предложений, текстов); **-фонетика –** изучает звуки речи, звуковое строение языка**;** **- синтаксис -** строение словосочетаний и предложений и функциональное взаимодействие в них различных частей речи (часть грамматика).	Состав: **-алфавит -** фиксированный для данного языка набор символов, которые могут быть использованы при написании программы, никакие другие символы в тексте не допускаются**;** **- синтаксис -** правила построения из символов алфавита специальных конструкций, с помощью которых составляется алгоритм. Этот набор правил устанавливает, какие комбинации символов являются осмысленными предложениями на этом языке**;** **- семантика -** система правил толкования конструкций (слов) языка.

Классификация языков высокого уровня

Традиционно языки высокого уровня делятся на:

- **процедурные (императивные)**;
- **логические**;
- **объектно-ориентированные**.

Процедурные языки

Процедурные (иногда называют ***императивными***) языки предназначены для однозначного описания алгоритмов. При решении задачи процедурные языки требуют в той или иной форме явно записать процедуру ее решения.

Одни из наиболее известных:

Fortran и ***Algol*** - языки, предназначенные для решения научно-технических задач.

Cobol – для решения экономических задач.

Basic – для решения небольших вычислительных задач в диалоговом режиме.

В то же время в середине 60-х годов начали разрабатывать алгоритмические языки широкой ориентации – универсальные языки. Обычно они строились по принципу объединения возможностей узко-ориентированных языков.

Среди них наиболее известны: ***PL/1***, ***Pascal***, ***C***, ***C+*** , ***Modula***, ***Ada***.

Однако, как любое универсальное средство, такие широко-ориентированные языки во многих конкретных случаях оказываются менее эффективными

Логические языки

Логические языки ориентированы не на запись алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания. В этих языках указывается что дано и что требуется получить. При этом поиск решения задачи возлагается непосредственно на ЭВМ.

Наиболее известные и используемые: ***Prolog***, ***Lisp***, ***Mercury*** и др.

Развитие систем искусственного интеллекта опирается на парадигму логического программирования.

Объектно-ориентированные языки

Принцип ***объектно-ориентированного программирования:*** программа представляется в виде совокупности объектов, каждый из которых является представителем определенного класса, а классы образуют иерархию наследования.

Принцип объектно-ориентированных языков заключается в стремлении связать данные с обрабатывающими эти данные процедурами в единое целое - **объект**.

К объектно-ориентированным языкам относятся такие известные языки, как: ***C++***, ***Object Pascal***, ***Java***.

Базовые понятия объектно-ориентированного подхода

- **объект**;
  - **класс**;
  - **поле данных (*свойство*)** — это характеристика объекта и его параметров. Все объекты наделены определенными свойствами, совокупность которых выделяет (определяет) объект.
  - **метод** — это набор действий (*функция* или *процедура*) над объектом или его свойствами, определяют поведение объектов;
  - **событие** — это характеристика ***изменения состояния*** объекта.

Объект

Представляет собой совокупность свойств (параметров) определенных сущностей и методов их обработки (программных средств). Каждый объект имеет определенный тип. Можно сказать, что все является **объектом**.

Объект: автомобиль

Объект: студент

Класс

Каждый объект является элементом определенного **класса**.

Класс - это совокупность объектов, характеризующихся общностью применяемых к ним методов обработки или свойств.

Класс – тип данных, описывающий внутреннюю структуру и поведение производных от него объектов.

Классы и объекты

Класс: автомобиль

Объект: автомобиль, гос. номер ...

Класс: студент

Объект: студент «Дуб»

Важнейшие принципы объектно-ориентированного подхода:

- **абстрагирование;**
- **инкапсуляция**;
- **наследование**
- **полиморфизм.**

Абстрагирование позволяет выделять какие-либо свойства объектов и затем на основе этих свойств создавать конкретные классы, а другие свойства рассматривать как несущественные.

Выбор правильного набора абстракций для заданной предметной области является основной задачей объектно-ориентированного программирования.

Абстрагирование

Поля класса Студент:

- *Университет;*
- *Факультет;*
- *Курс;*
- *Группа.*

Поля данных

Инкапсуляция

Под **инкапсуляцией** (объединением) понимается скрытие ***полей*** объекта с целью обеспечения доступа к ним только посредством ***методов*** класса (т. е. скрытие деталей, несущественных для использования объекта).

Инкапсуляция позволяет пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством ***публичных методов*** (предоставляемого интерфейса)

Методы

Метод состоит их некоторого количества ***операторов*** для выполнения какого-то действия, имеет набор входных *аргументов* и *возвращаемое значение.*

В зависимости от того, какой уровень доступа предоставляют методы, выделяют:

- ***Public*** – открытый интерфейс, общий для всех пользователей данного класса;
- ***Private*** – закрытый интерфейс, доступный только изнутри данного класса ;
- ***Protected*** – защищенный интерфейс, внутренний для всех наследников данного класса.

Наследование

Наследование - механизм языка, позволяющий описать новый класс на основе уже существующего суперкласса (родительского, базового класса).

Класс-потомок (подкласс) может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.

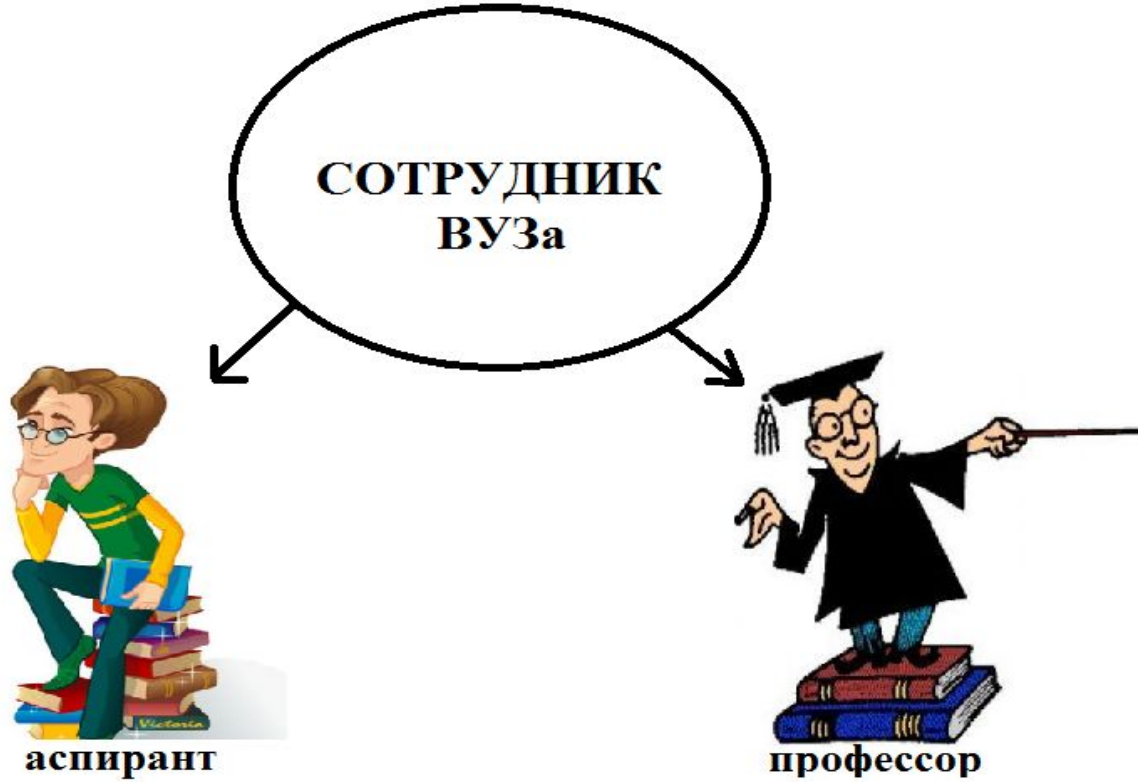
Механизм наследования позволяет строить иерархии классов, т.е. класс может иметь образованные от него подклассы. При построении подклассов осуществляется наследование данных и методов обработки объектов исходного класса.

Виды наследования:

- **простое;**
- **множественное.**

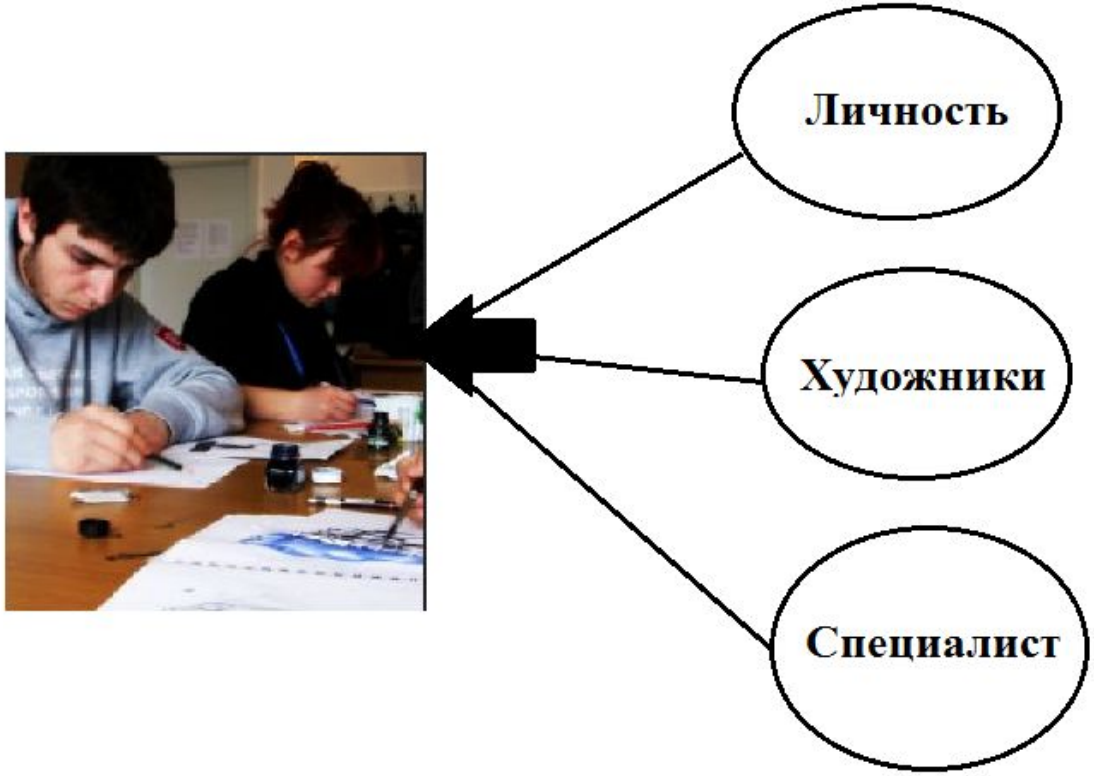
Простое наследование. Класс, от которого произошло наследование, называется *базовым* или *родительским*. Классы, которые произошли от базового, называются *потомками*, *наследниками* или *производными классами.* Объекты создаются только на основе *производных классов*, наследованных от абстрактного.

Например, *базовый* класс - «*сотрудник ВУЗа*», от которого наследуются классы «*аспирант*», «*профессор*» и т. д.



Множественное наследование.

При множественном наследовании у класса может быть более одного суперкласса. В этом случае класс наследует методы (функциональность) суперклассов.



Полиморфизм

Полиморфизмом называется единообразная обработка разнотипных данных, т.е. одно и то же имя можно использовать для обработки общих для класса действий.

Например, операция «+» для *целых, длинных целых, символьных переменных* и *чисел с плавающей запятой*.

Типы полиморфизма

Существует 3 принципиально разных типа полиморфизма:

- ***ситуативный*** - функция описывает разные реализации (возможно, с различным поведением) для ограниченного набора явно заданных типов и их комбинаций.
- ***параметрический*** - код написан отвлеченно от конкретного типа данных и поэтому свободно используется с любыми новыми типами данных;
- ***полиморфизм подтипов*** – позволяет функции, выполняющейся на одном типе **Z**, корректно выполняться на аргументах типа **J**, являющегося подтипом **Z**.

Другая классификация языков программирования

- **процедурные**;
- **логические**;
- **объектно-ориентированные**;
- **структурные;**
- **функциональные;**
- **аспектно-ориентированные.**

Структурные языки программирования

В основе лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э.Дейкстрой, разработана и дополнена Николаусом Виртом.

Идея данной методологии:

1. Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

последовательное исполнение — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;

ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;

цикл — многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

2. Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция **вызова подпрограммы**. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.

3. Разработка программы ведётся пошагово, методом «сверху вниз».

Теорема о структурном программировании (теорема Бома-Якопини)

Любую схему алгоритма можно представить в виде композиции вложенных блоков ***begin*** и ***end***, условных операторов ***if***, ***then***, ***else***, циклов с предусловием (***while***) и может быть дополнительных логических переменных (флагов).

Эта теорема была сформулирована итальянскими математиками Коррадо Бомом и Джузеппе Якопини в 1966 году и говорит о том, как можно избежать использования оператора перехода ***GOTO***.

Состав структурного языка

- ***символы*** **-** основные неделимые знаки, в терминах которых пишутся все тексты на языке;
- ***слова*** **-** элементарные конструкции - минимальные единицы языка, имеющие самостоятельный смысл; образуются из основных символов языка;
- ***словосочетания*** – выражения - состоят из элементарных конструкций и символов, задают правила вычисления некоторого значения;
- ***предложения*** – операторы - задают полное описание некоторого действия, которое необходимо выполнить.

Символы

Основными символами структурного языка, составляющими его алфавит являются ***буквы***, ***цифры*** и ***специальные символы***:

1. 26 латинских строчных и 26 латинских прописных букв
2. _ (подчеркивание)
3. 10 цифр: **0 1 2 3 4 5 6 7 8 9**;
4. знаки операций: **+ - * / = <> < > <= >= := @**
5. ограничители: **. , ' ( ) [ ] (. .) { } (* *) .. : ;**
6. спецификаторы: **^ # $**
7. ***служебные*** (зарезервированные) слова.

Элементарные конструкции

Элементарные конструкции включают в себя ***имена***, ***числа*** и ***строки***.

Имя - это последовательность букв и цифр, начинающаяся с буквы, называет элементы языка – константы, метки, типы, переменные, процедуры, функции, модули, объекты.

Числа обычно записываются в десятичной системе счисления, могут быть целыми и действительными.

Целые числа записываются в форме без десятичной точки, например:

217 -45 8954 +483.

Действительные числа записываются в форме с десятичной точкой или с использованием десятичного порядка (E):

28.6 0.65 -0.018 4.0

5E12 -1.72E9 73.1E-16

Строки

Строки представляют собой последовательность символов, записанную между апострофами.

Если в строке в качестве содержательного символа необходимо употребить сам апостроф, то следует записать два апострофа.

Примеры строк:

'СТРОКА' 'STRING'

'ПРОГРАММА'

'АД''ЮТАНТ'

Типы данных

Тип определяет:

- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;

- внутреннюю форму представления данных в ЭВМ;

- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

Типы данных:

- ***целые*** - определяют константы, переменные и функции, значения которых реализуются множеством целых чисел, допустимых в данной ЭВМ;
- ***действительные*** - определяет данные, которые реализуются подмножеством действительных чисел, допустимых в данной ЭВМ;
- ***логические (Boolean)*** – определяет данные, которые могут принимать логические значения TRUE и FALSE;
- ***символьный (Char)***;
- ***адресный (Pointer).***

Целые

тип	диапазон значений	требуемая память
Shortint	-128 .. 127	1 байт
Integer	-32768 .. 32767	2 байта
Longint	-2147483648 .. 2147483647	4 байта
Byte	0 .. 255	1 байт
Word	0 .. 65535	2 байта

Действительные

тип	диапазон значений	количество цифр мантиссы	требуемая память
Real	2.9e-39 .. 1.7e+38	11	6 байт
Single	1.5e-45 .. 3.4e+38	7	4 байта
Double	5.0e-324 .. 1.7e+308	15	8 байта
Extended	3.4e-4932 ...1e+4932	19	10 байт
Comp	-9.2e+18 .. 9.2e+18	19	8 байта

Стандартные операции

Над переменными можно выполнять следующие арифметические операции:

- Сложение **+**,
- Вычитание **-**,
- Умножение *****,
- Деление **div**,
- получение остатка от деления **mod**.

Например:

17 **div** 2 = 8, 3 **div** 5 = 0.

17 **mod** 2 = 1, 3 **mod** 5 = 3.

Стандартные функции

abs(x) - абсолютное значение аргумента
arctan(x) - арктангенс аргумента
sin(x) - синус аргумента
cos(x) - косинус аргумента
exp(x) - экспонента аргумента
Ln(x) - натуральный логарифм

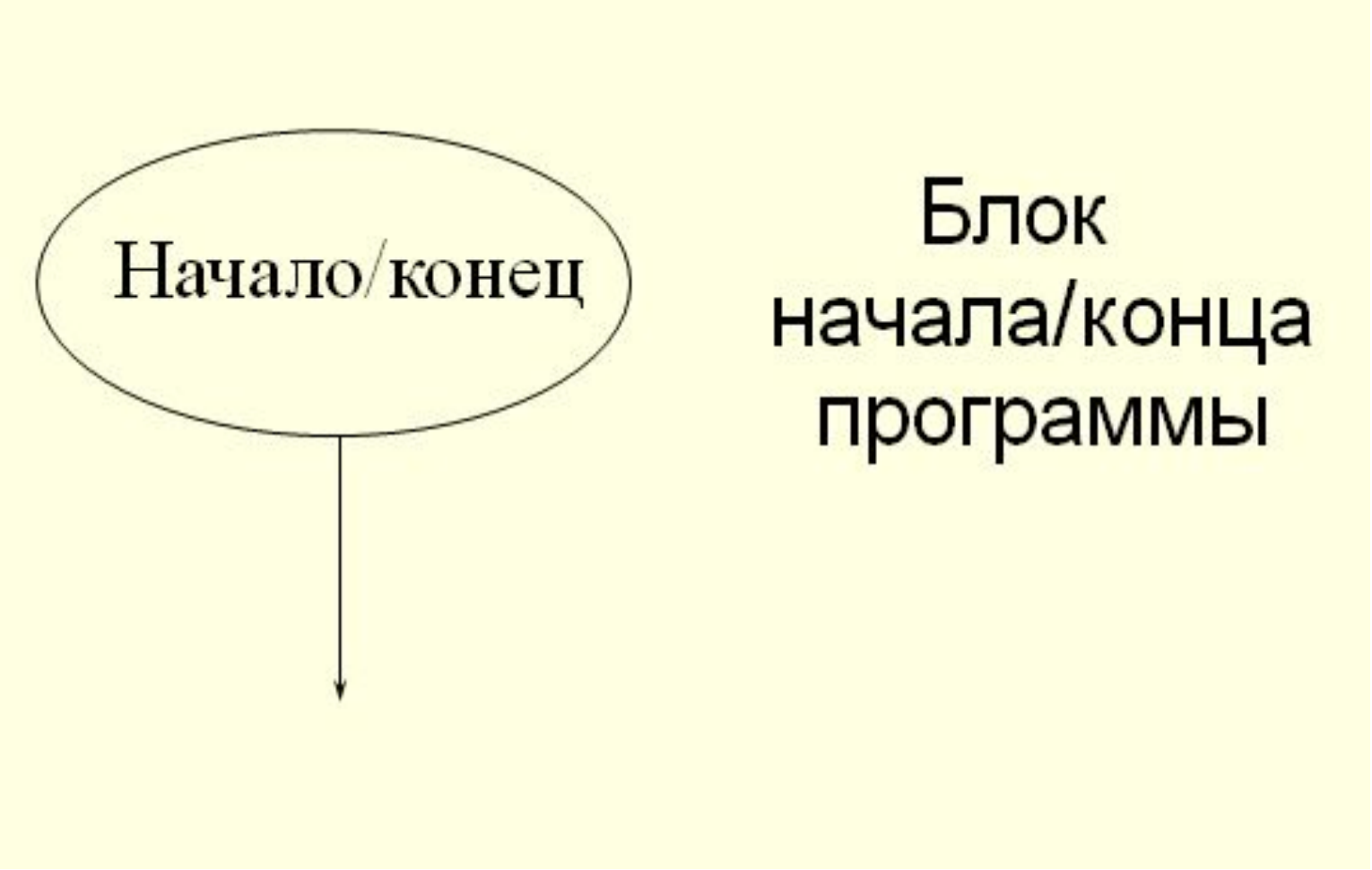
Стандартные функции

sqr(x) - квадрат аргумента
sqrt(x) - квадратный корень из аргумента
Pi - число π=3,14159265358979323 85
Frac(x) - дробная часть числа
int(x) - целая часть числа
Random -
равномерное псевдослучайное число от 0 до 1
Random(x) -
равномерное псевдослучайное число от 0 до x
Randomize -
инициализатция датчика псевдослучайных чисел

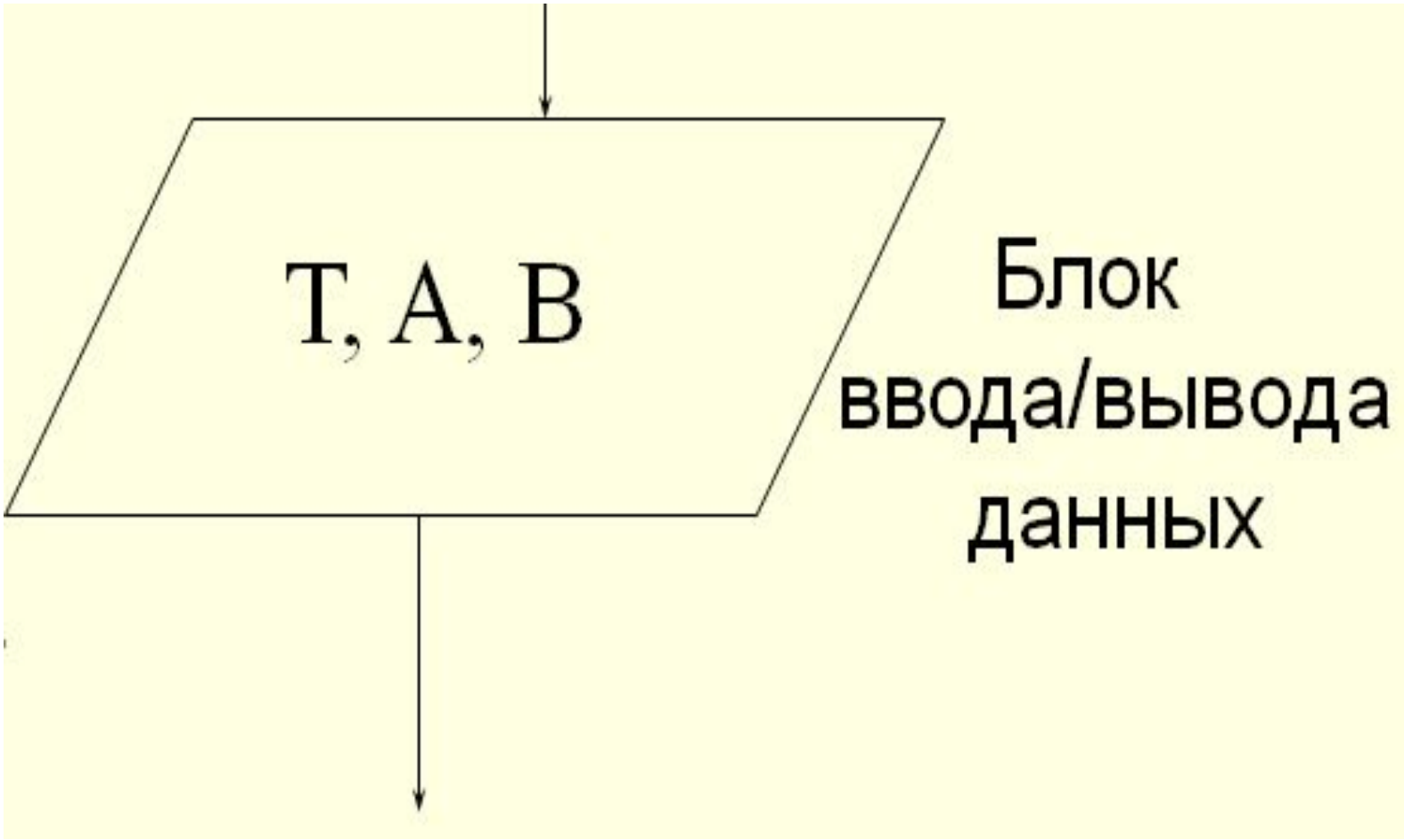
Операции отношений

Операция	Действие	Выражение	Результат
=	равно	A = B	если A равно B, True
>	больше	A > B	если A больше B, True
<	меньше	A < B	если A меньше B, True
<>	не равно	A <> B	если A не равно B, True
>=	больше или равно	A >= B	если A больше или равно B, True
<=	меньше или равно	A <= B	если A меньше или равно B, True

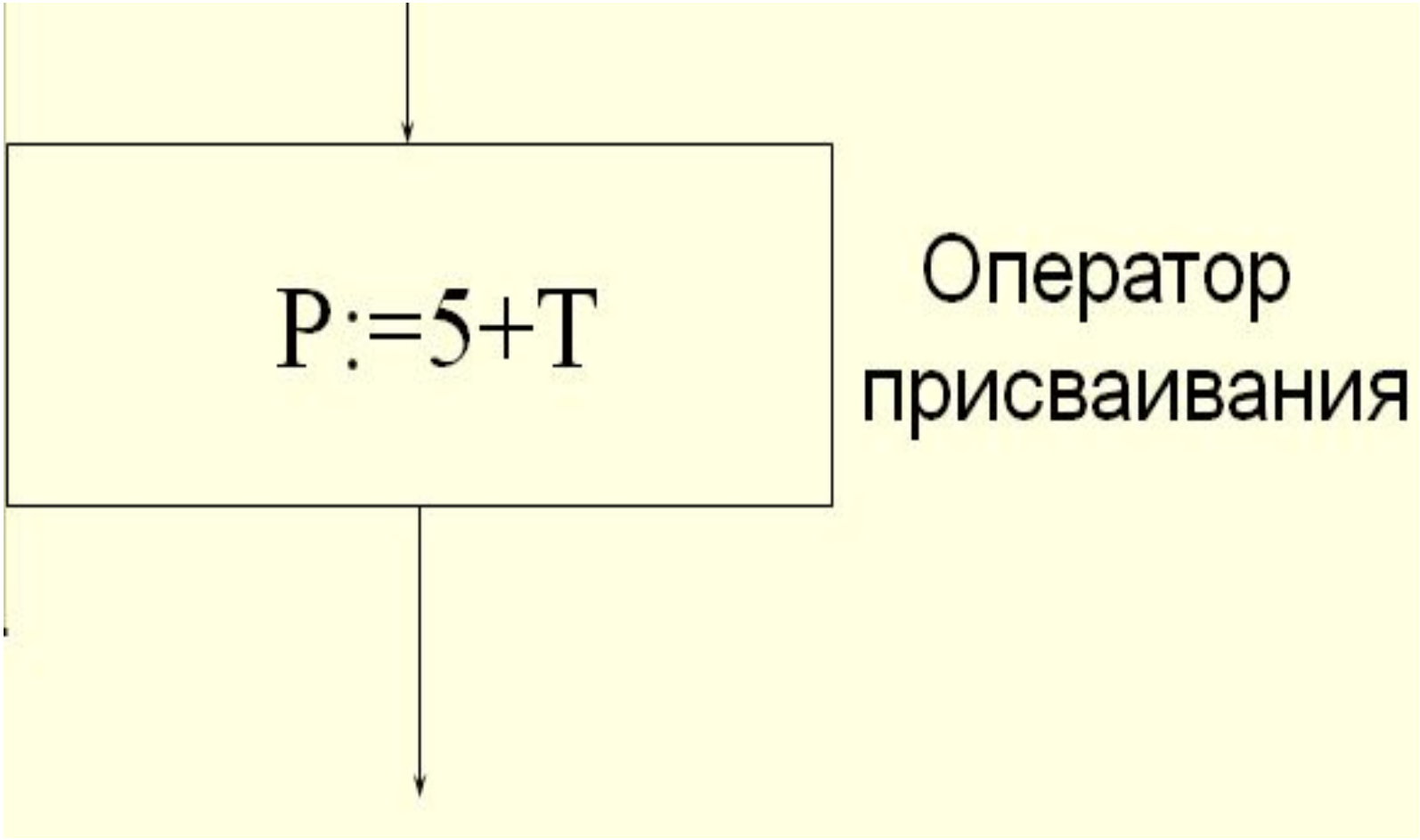
Основные алгоритмические конструкции



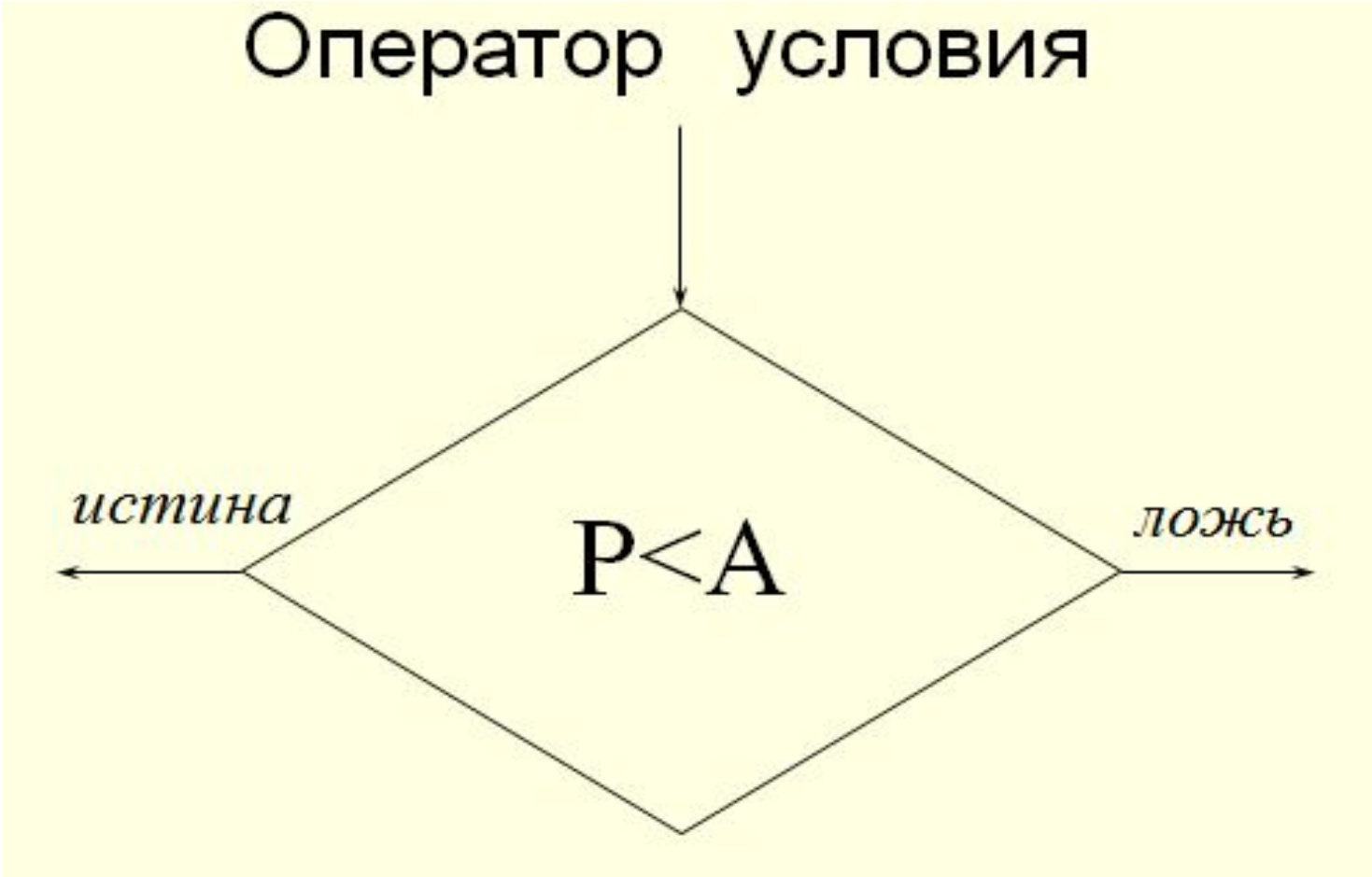
Основные алгоритмические конструкции



Основные алгоритмические конструкции



Основные алгоритмические конструкции



Линейные алгоритмы

В классе **N** учеников. После контрольной работы было получено:

A-пятерок,

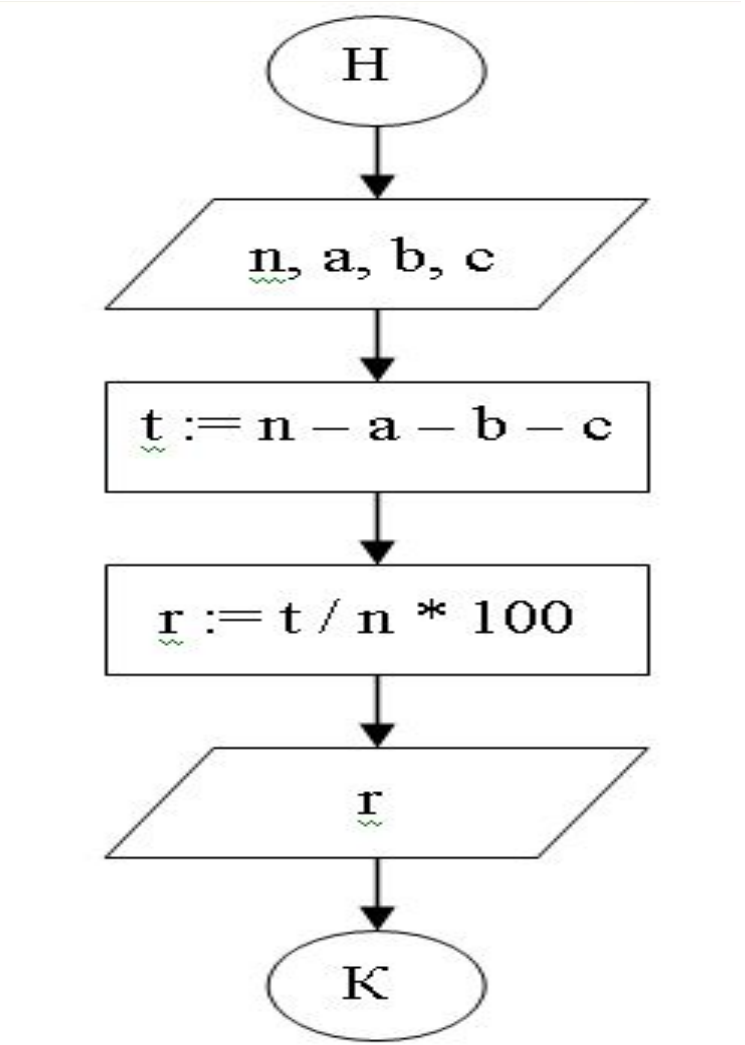
B-четверок,

C-двоек,

остальные тройки.

Найти процент троек. (Процент троек вычисляется по формуле **R=T/N*100**, где **T** - число троек.)

Составление линейного алгоритма



Алгоритмы со сложными операторами. Оператор условия.

1) **IF** <условие> **THEN** <действие >

[Если условие принимает значение «TRUE», то выполняется действие после служебного слова THEN. В противном случае выполняется действие, описанное в следующем операторе].

2) **IF** <условие> **THEN** <действие 1>

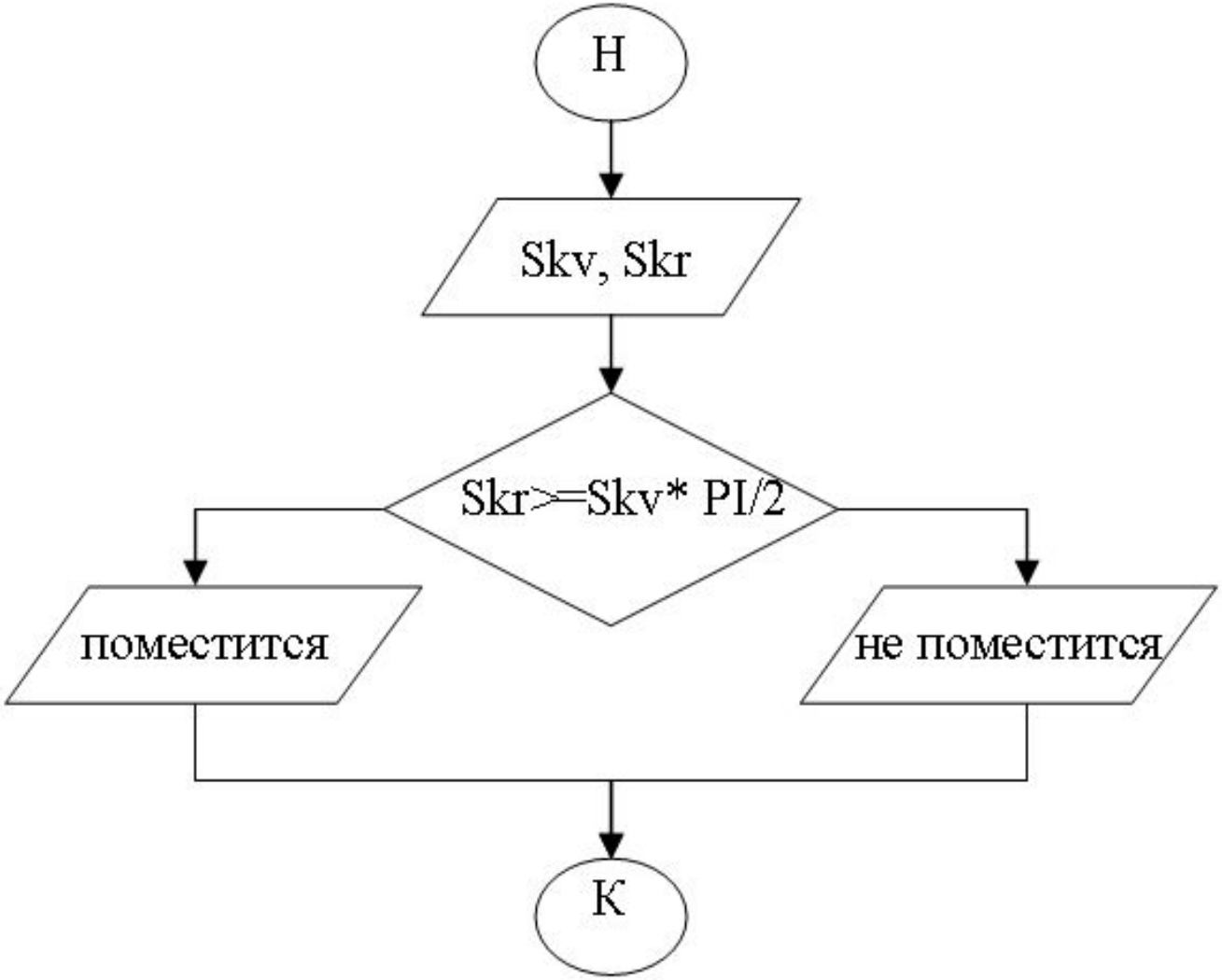
ELSE < действие 2>

[Если условие принимает значение «TRUE», то выполняется действие после служебного слова THEN. В противном случае выполняется действие после служебного слова ELSE].

Пример алгоритма

Известны площадь круга **Skr** и площадь квадрата **Skv**. Определить, поместится ли квадрат в круг.

Квадрат поместится в круг при условии, если Skr >= Skv * PI/2


Н
Skv, Skr
Skr>=Skv* PI/2
поместится
не поместится
К

Алгоритмы со сложными операторами. Операторы цикла.

1. цикл **с параметром :** (**FOR … DO** <действия>**)**;
2. цикл **с предусловием:**

(**WHILE** <логическое выражение> **DO** <оператор>);

3. цикл **с постусловием:**

(**REPEAT** <последовательность операторов> **UNTIL** <логическое выражение>).

Цикл с параметром

Общий вид оператора цикла с параметром имеет вид:

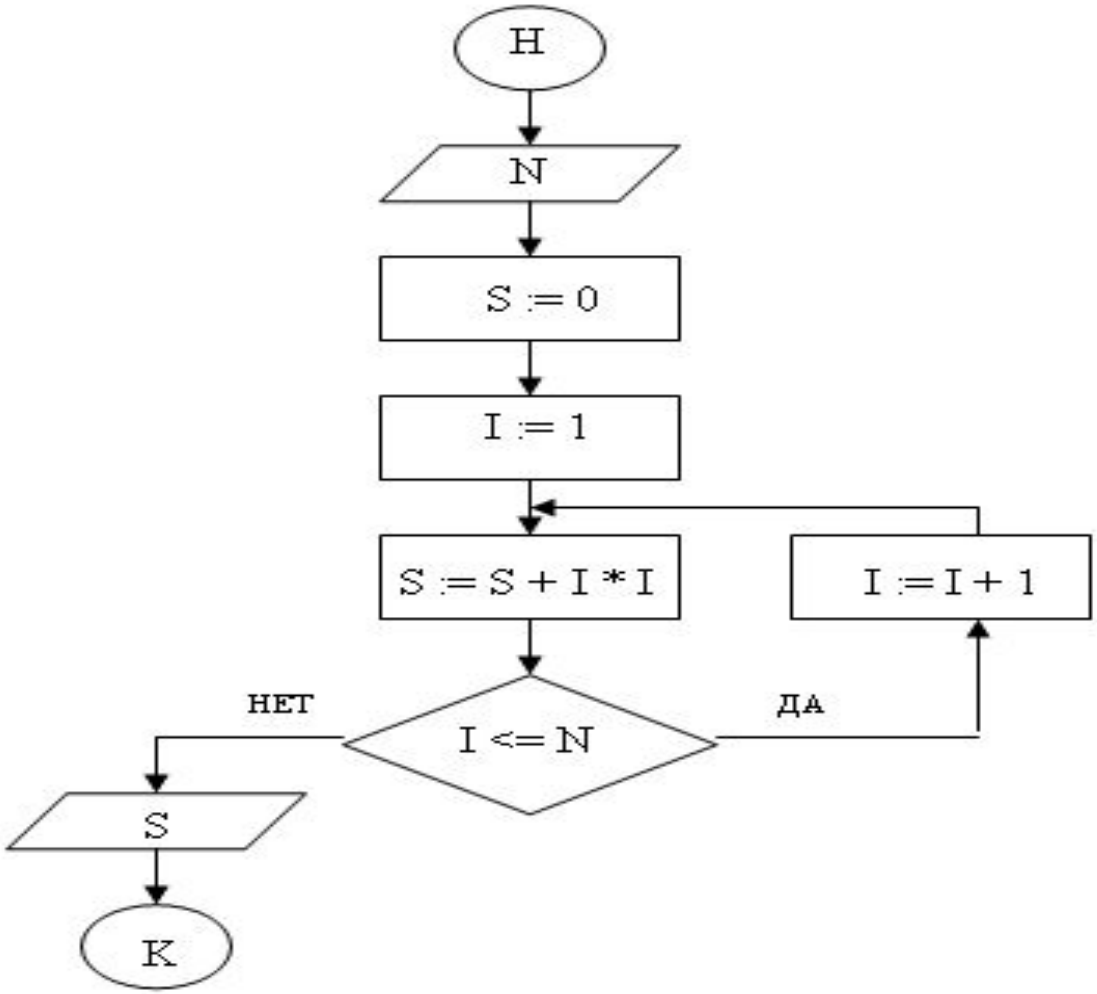
FOR I:=N1 **TO** N2 **DO <действия>**, где:

I – параметр цикла или счетчик;

N1 и **N2** – начальное и конечное значение параметра цикла;

Действия – оператор или группа операторов, заключенных в операторные скобки.

Пример. Задан ряд целых чисел 1 ... N. Найти сумму квадратов этих чисел.



Функциональные языки программирования

Основная идея: процесс вычисления трактуется как вычисление значений ***функций*** в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Противопоставляется парадигме процедурного (императивного) программирования, которая описывает процесс вычислений как последовательное изменение состояний.

При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например как ***список***.

Функциональное	**Процедурное (императивное)**
Предполагает обходиться вычислением результатов *функций* от *исходных данных* и *результатов других функций*. Не предполагает явного хранения состояния программы, соответственно, не предполагает изменяемость этого состояния . В функциональном языке при вызове функции с одними и теми же аргументами всегда получим одинаковый результат: выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках кэшировать результаты функций и вызывать их в порядке, не определяемом алгоритмом.	Базовой концепцией является *переменная*, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма. Функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных и менять состояние внешних переменных. В императивном программировании при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, можно получить разные данные на выходе из-за влияния на функцию состояния переменных.

Аспектно-ориентированное программирование (АОП)

Данная парадигма программирования основана на идее разделения функциональности для улучшения разбиения программы на ***модули***.

Указанные выше парадигмы программирования предоставляют определённые способы для разделения и выделения функциональности: ***функции***, ***классы***, ***объекты***, но некоторую функциональность с помощью предложенных методов невозможно выделить в отдельные сущности.

Такую функциональность называют ***сквозной функциональностью***.

Все языки АОП предоставляют средства для выделения сквозной функциональности в отдельную сущность.

Примером языка АОП является аспектно-ориентированное расширение для языка ***Java – AspectJ***, разработанное в 2001 г.

Основные понятия АОП:

- ***аспект*** — модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя ***совет*** в ***точках соединения***, определённых некоторым ***срезом***.
- ***совет*** — средство оформления кода, которое должно быть вызвано из точки соединения. Совет может быть выполнен *до*, *после* или *вместо* точки соединения.
- ***точка соединения*** — точка в выполняемой программе, где следует применить совет.
- ***срез*** — набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в ***AspectJ*** применяются ***Java-сигнатуры***) и позволяют их повторное использование с помощью переименования и комбинирования.
- ***внедрение*** — изменение структуры класса и (или) изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (***metaobject protocol***, ***MOP***).

Области применения языков программирования

- научные вычисления (языки ***C++***, ***Fortran***, ***Java***);
- системное программирование (языки ***C++, Java***);
- обработка информации (языки ***C++, COBOL***, ***Java***);
- искусственный интеллект (***LISP***, ***Prolog***);
- издательская деятельность (***Postscript***, ***TeX, LaTEX***);
- удаленная обработка информации (***Perl***, ***PHP***, ***Java***, ***C++***);
- описание документов (***HTML***, ***XML***).

Факторы, определяющие развитие языков программирования

- наличие среды программирования, поддерживающей разработку приложений на конкретном языке программирования;
- удобство сопровождения и тестирования программ;
- стоимость разработки с применением конкретного языка программирования;
- четкость и ортогональность конструкций языка;
- применение объектно-ориентированного подхода.

Стандартизация языков программирования

1. Американский национальный институт стандартов **ANSI** (**American National Standards Institute**).
2. Институт инженеров по электротехнике и электронике **IEEE** (**Institute of Electrical and Electronic Engineers**).
3. Организация международных стандартов **ISO** (**International Organization for Standardization**).