**Классификация языков программирования**

Сегодня можно насчитать более 2 тыс. различных языков программирования и их модификаций, однако лишь отдельные получили широкое признание.

Все языки программирования можно условно классифицировать по некоторым основным признакам:

1. По степени зависимости от аппаратных средств: языки низкого уровня и языки высокого уровня;
2. По принципам программирования: структурированные, модульные и объектно-ориентированные;
3. По ориентации на класс задач: универсальные, специализированные.

Языки программирования низкого уровня (машинно-ориентированные) — языки, в которых команды и данные учитывают архитектуру компьютера. Такие языки ориентированы на конкретный тип компьютера и учитывают его аппаратные особенности. Программы, которые представлены совокупностью 0 и 1, называют машинными или машинным кодом. Совокупность 0 и 1 указывает, какую именно действие следует выполнить процессору.

Языки программирования высокого уровня (машинно-независимые) — языки, на которых программы могут использоваться на компьютерах различных типов и которые более доступны человеку, чем языки низкого уровня. Первым языком высокого уровня, который получил широкое признание среди программистов мира, был Fortran. Формально, сюда можно отнести все известные языки программирования.

*Структурированные (процедурные) языки программирования* получили свое название потому что выдерживают четкую структуризацию программ. Они основаны на описании последовательной смены состояния компьютера, то есть значения ячеек памяти, состояния процессора и других устройств. Они манипулируют данными в пошаговом режиме, используя пошаговые инструкции. Структурированные языки полностью удовлетворяют потребности разработки небольших программ и программ средней сложности. (Пример, Pascal).

Парадигма *модульного программирования* предполагает построение программы из отдельных программных блоков особого вида - модулей. Модуль позволяет разбить текст программы на несколько физических файлов, компилируемых раздельно. Это дает возможность автономно отлаживать и тестировать модули, а также во многом снимает проблему внесения в готовый код непреднамеренных и несанкционированных изменений. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и др. программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней. Например, модуль Nunpy, Tkinter и др.

В конце XX в. была презентована новая методика программирования, получила название *объектно-ориентированного программирования* (ООП). То есть начали развиваться языки, содержащие конструкции, позволяющие определять объекты, принадлежащие к классам и имеющие средства для работы с абстрактными типами данных. К таким языкам относятся C ++, Java, C #, Python и др. Сегодня языки ООП практически вытеснили с рынка профессионального программирования структурированные языки.

**Система программирования.**

Для удобной разработки программ существуют специальные средства их создания, — системы (среды) программирования, которые обеспечивают весь цикл работы с программой — от ее разработки до выполнения и получения необходимых результатов.

Система программирования — это комплекс программных средств, предназначенных для автоматизации процесса подготовки и выполнения программ пользователя.

Назначение и состав систем программирования:

1. Редактор текста;
2. Язык программирования;
3. Библиотека подпрограмм;
4. Редактор связей (компоновщик);
5. Транслятор;
6. Отладчик;

*Редактор исходного кода*

Вводим текст разработанной программы, которую называют исходным кодом, в компьютер и храним в памяти. Для этого система программирования имеет редактор текста, который обеспечивает ввод и редактирование исходного кода.

*Компиляция и интерпретация*

После введения программы и исправления ошибок, которые могли произойти во время ввода, осуществляется преобразование программы с языка программирования высокого уровня в двоичный код. Такое преобразование осуществляется с помощью транслятора программ.

Различают два типа трансляторов: компиляторы и интерпретаторы.

В процессе интерпретации исходных текстов программ каждая команда (инструкция) последовательно превращается в двоичный код и сразу выполняется — на экране высвечивается результат ее выполнения. После завершения одной команды выполняется следующая и так далее до последней команды. Но результат преобразования не сохраняется, и каждый запуск программы начинается сначала.

В процессе компиляции осуществляется преобразование всего текста программного кода в двоичный код. Полученную после компиляции программу называют объектным модулем. Такая программа еще не готова к выполнению. Исходный код обычно содержит ссылки на другие модули (подпрограммы), которые содержатся в библиотеке подпрограмм (например, модуль вычисления квадратного корня). Таким образом, к программному модулю нужно добавить коды необходимых подпрограмм, чтобы подготовить программу для исполнения.

Компилируемая программы выполняются быстрее интерпретируемых. Режим интерпретации нуждается в дополнительной основной памяти, поскольку интерпретатор должен все время храниться вместе с кодом. Но интерпретация в работе удобнее. Особенно для программистов, которые только начинают работать с системами программирования, так контролируется результат каждой команды.

*Компоновка*

После компиляции компоновщик (редактор связей) «склеивает» отдельные двоичные модули в единую программу, которая называется исполняемой программой. После компиляции программа представлена двоичными символами 1 и 0 и готова к исполнению на компьютере.

*Отладка и тестирование*

Полученная программа, даже если она выполняется, не гарантирует, что нет логических ошибок. Она может выполняться, но результат исполнения может быть неправильным. Поэтому нужно провести тестирование (испытания) программы на

предмет выявления и устранения в ней логических ошибок. Тестирование — достаточно ответственный этап. В крупных IT-компаниях над разработкой программ, которые называют проектами, работают десятки и даже сотни программистов разных направлений. Одни из них разрабатывают проекты, другие занимаются тестированием программ, экономическим обоснованием и тому подобное. На этом этапе применяется отладчик программ, который позволяет пошагово анализировать программу. Отладчик позволяет выполнять трассировку программы, устанавливать и удалять контрольные точки в программах, условия приостановления выполнения программы и тому подобное.

**Этапы решения задачи на ЭВМ.**

Работа по решению любой задачи с использованием компьютера включает в себя следующие шесть этапов:

1. Постановка задачи.
2. Формализация задачи.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Непосредственно к программированию из этого списка относятся п. 3 - 5.

**Данные и величины.**

Данные – это совокупность величин, с которыми работает компьютер. По отношению к программе различают исходные, окончательные (результаты) и промежуточные данные, которые получают в процессе вычислений.

Любая величина имеет три основных свойства: имя, значение и тип. На уровне команд процессора величина идентифицируется адресом ячейки памяти, в которой она хранится.

Любые константы и переменные занимают ячейку памяти, а значения этих величин определяются двоичным кодом в этой ячейке.

В каждом языке программирования существует своя концепция и своя система типов данных. Однако в любой язык входит минимально необходимый набор основных типов данных: целые, вещественные, логические и символьные. С типом величины связаны три ее свойства: множество допустимых значений, множество допустимых операций, форма внутреннего представления.

**Типизация в языках программирования**

Неявная типизация означает, что при объявлении переменной не нужно указывать её тип (Python), при явной – это делать необходимо (Java, C++).

Раздаточный материал № 2

Java: int a = 1;
Python: a = 1

Также языки бывают с динамической (Python) и статической (C, C#, Java) типизацией. В первом случае тип переменной определяется непосредственно при выполнении программы, во втором – на этапе компиляции.

Сильная типизация (Python, Java) не позволяет производить операции в выражениях с данными различных типов, слабая (C и C++) – позволяет. В языках с сильной типизацией нельзя складывать, например, строки и числа, нужно все приводить к одному типу.

Вывод: язык Python относится к языкам с неявной сильной динамической типизацией.