Профессор
Игорь Н. **Бекман**

# КОМПЬЮТЕРНЫЕ НАУКИ

Курс лекций

## Лекция 8. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Содержание



Согласно определению Б.Л Ворфа: «Язык формирует наш способ мышления и определяет то, о чём мы можем мыслить». Для компьютера важнее не просто язык, а язык программирования, под которым понимают формальную знаковую систему, предназначенную для записи программ, задающих алгоритм в форме, понятной для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Со времени создания первых программируемых машин человечество придумало уже более восьми с половиной тысяч языков программирования. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

В данной лекции рассматривается понятие языка программирования, приводятся примеры различных языков программирования и трансляторов. Основное внимание уделено машинно-ориентированным языкам.

Прогресс компьютерных технологий определил процесс появления новых разнообразных знаковых систем для записи алгоритмов – языков программирования. Оснащенный набор вычислительных формул дополнительной информации, превращает данный набор в алгоритм. Язык программирования служит двум связанным между собой целям: он даёт программисту аппарат для задания действий, которые должны быть выполнены, и формирует концепции, которыми пользуется программист, размышляя о том, что делать. Первой цели идеально отвечает язык, который настолько «близок к машине» что всеми основными машинными аспектами можно легко и просто оперировать достаточно очевидным для программиста образом. Второй цели идеально отвечает язык, который настолько «близок к решаемой задаче», чтобы концепции ее решения можно было выражать прямо и коротко.

Связь между языком, на котором мы думаем/программируем, и задачами и решениями, которые мы можем представлять в своем воображении, очень близка. По этой причине ограничивать свойства языка только целями исключения ошибок программиста в лучшем случае опасно. Как и в случае с естественными языками, есть огромная польза быть, по крайней мере, двуязычным. Язык предоставляет программисту набор концептуальных инструментов, если они не отвечают задаче, то их просто игнорируют.

Конкретный компьютер способен работать с программами, написанными на его родном машинном языке. Существует почти столько же разных машинных языков, сколько и компьютеров, но все они суть разновидности одной идей простые операции производятся со скоростью молнии на двоичных числах.

Процесс работы компьютера заключается в выполнении программы, то есть набора вполне определённых команд во вполне определённом порядке. Машинный вид команды, состоящий из нулей и единиц, указывает, какое именно действие должен выполнить центральный процессор. Значит, чтобы задать компьютеру последовательность действий, которые он должен выполнить, нужно задать последовательность двоичных кодов соответствующих команд. Программы в машинных кодах состоят из тысячи команд. Писать такие программы – занятие сложное и утомительное. Программист должен помнить комбинацию нулей и единиц двоичного кода каждой программы, а также двоичные коды адресов данных, используемых при её выполнении. Гораздо проще написать программу на каком-нибудь языке, более близком к естественному человеческому языку, а работу по переводу этой программы в машинные коды поручить компьютеру. Так возникли языки, предназначенные специально для написания программ, - языки программирования.

Всё множество языков программирования можно разделить на две группы: языки низкого уровня и языки высокого уровня.

Большинство программистов пользуются для составления программ языками высокого уровня. Как и обычный человеческий язык, такой язык имеет свой алфавит – множество символов, используемых в языке. Из этих символов составляются так называемые ключевые слова языка. Каждое из ключевых слов выполняет свою функцию, так же как в привычном нам языке нам языке слова, составленные из букв алфавита данного языка, могут выполнять функции разных частей речи. Ключевые слова связываются друг с другом в предложения по определённым синтаксическим правилам языка. Каждое предложение определяет некоторую последовательность действий, которые должен выполнить компьютер. Язык высокого уровня выполняет роль посредника между человеком и компьютером, позволяя человеку общаться с компьютером более привычным для человека способом. Часто такой язык помогает выбрать правильный метод решения задачи. Перед тем как писать программу на языке высокого уровня, программист должен составить алгоритм решения задачи, то есть пошаговый план действий, который нужно выполнить для решения этой задачи. Поэтому языки, требующие предварительного составления алгоритма, часто называют алгоритмическими языками.

## 1. ОПРЕДЕЛЕНИЕ ПОНЯТИЙ

Прежде всего дадим определения некоторых важных понятий, о которых пойдёт речь в этой лекции.

**Алгоритм** - точное предписание исполнителю совершить определенную последовательность действий для достижения поставленной цели за конечное число шагов.

**Язык программирования** (Алгоритмический язык) - искусственный (формальный) язык, предназначенный для записи алгоритмов. Язык программирования задается своим описанием и реализуется в виде специальной программы: компилятора или интерпретатора.

**Программа** - последовательность машинных команд, предназначенная для достижения конкретного результата.

**Программа** - согласно ГОСТ 19781-90 - данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

**Декларативный язык программирования** - язык программирования высокого уровня, построенный: на описании данных; и на описании искомого результата. Декларативные языки подразделяются на функциональные и логические языки.

**Исходный код** - текст программы на алгоритмическом языке. В компьютере исходный текст либо непосредственно выполняется интерпретатором, либо предварительно переводится компилятором в стандартный загрузочный код, способный многократно исполняться в определенной вычислительной среде.

**Макроязык** - программное средство, позволяющее пользователю задавать в тексте условные эквиваленты определенных его фрагментов. Макроязык состоит из: макроопределений, создаваемых пользователем; и из программ-препроцессоров, переводящих макроопределения в результирующие тексты.

**Машинный язык** - язык программирования, элементами которого являются команды компьютера, характеризующиеся: количеством операндов в команде; назначением информации, задаваемой в операндах; набором операций, которые может выполнить компьютер и др.

**Проблемно-ориентированный** язык программирования - язык программирования, управляющие структуры и/или структуры данных которого отражают особенности класса решаемых задач.

Конструкции машинного языка интерпретируются непосредственно аппаратурой.

**Процедурно-ориентированный язык программирования** - язык программирования высокого уровня, в основу которого положен принцип описания (последовательности) действий, позволяющей решить поставленную задачу. Обычно процедурно-ориентированные языки задают программы как совокупности процедур или подпрограмм.

**Рекурсивное построение** - в языках программирования - поименованная конструкция, использующая в своей структуре обращение к самой себе.

**Семантика** - в программировании - система правил истолкования отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка.
**Синтаксис** - набор правил построения фраз алгоритмического языка, позволяющий определить, осмысленные предложения в этом языке.
**Система программирования** - программная система, предназначенная для разработки программ на конкретном языке программирования. Система программирования предоставляет пользователю специальные средства разработки программ: транслятор, (специальный) редактор текстов программ, библиотеки стандартных подпрограмм, программную документацию, отладчик и др.
**Тип данных** - характеристика набора данных, которая определяет: диапазон возможных значений данных из набора; допустимые операции, которые можно выполнять над этими значениями; способ хранения этих значений в памяти. Различают: простые типы данных: целые, действительные числа и др.; - составные типы данных: массивы, файлы и др.
**Язык ассемблера** - согласно ГОСТ 19781-90 - язык программирования; символьная форма машинного языка с рядом возможностей, характерных для языка высокого уровня. Обычно язык ассемблера включает макросредства.
**Язык высокого уровня** - согласно ГОСТ 19781-90 - язык программирования, понятия и структура которого удобны для восприятия человеком.
**Языки высокого уровня** отражают потребности программиста, но не возможности системы обработки данных.
**Язык описания технических средств** - язык моделирования, разработки и тестирования устройств, предназначенных для обработки дискретных сигналов.

Итак, **язык программирования** - формализованный язык, предназначенный для описания программ и алгоритмов решения задач на ЭВМ. Языки программирования являются искусственными. В них синтаксис и семантика строго определены. Поэтому они не допускают свободного толкования выражения, что характерно для естественного языка.

Языки программирования разделяются на две основные категории языки высокого уровня и языки низкого уровня :

**Язык высокого уровня** - язык программирования, средства которого обеспечивают описание задачи в наглядном, легко воспринимаемом виде, удобном для программиста. Он не зависит от внутренних машинных кодов ЭВМ любого типа, поэтому программы, написанные на языках высокого уровня, требуют перевода в машинные коды программами транслятора либо интерпретатора. К языкам высокого уровня относят Фортран, ПЛ/1 , Бейсик, Паскаль, Си, Ада и др.

**Язык низкого уровня**, - язык программирования, предназначенный для определенного типа ЭВМ и отражающий его внутренний машинный код.

Различают также следующие виды языков программирования:
**Алгоритмический язык** - совокупность символов, соглашений и правил, используемых для однозначного описания алгоритмов и обычно являющаяся часть языка программирования;
**Неалгоритмический язык** - язык программирования, тексты которого не содержат указаний на порядок выполнения операций и служат лишь исходным материалом для синтеза алгоритма решения задачи;
**Формальный язык** - язык программирования, построенный по правилам некоторого логического исчисления или формальной грамматики, представляющей собой систему правил построения в заданном алфавите конечных знаковых последовательностей, множество которых образует формальный язык;
**Исходный язык** - язык программирования, на котором написана программа, в отличие от машинного языка, на котором программы выполняются компьютером. Исходные языки классифицируются на языки высокого уровня и языки низкого уровня .
**Машинный** (абсолютный) язык, язык ЭВМ - язык программирования, предназначенный для представления программ в форме, обеспечивающей возможность их выполнения техническими средствами;
**Машинозависимый** (машинно-ориентированный) язык, машинозависимый язык программирования - язык программирования, учитывающий структуру и характеристики ЭВМ определенного типа или конкретной ЭВМ;
**Машиннонезависимый язык** - язык программирования, структура и средства которого не связаны ни с какой конкретной ЭВМ и позволяют выполнять составленные на нем программы на любой ЭВМ, снабженной трансляторами с этого языка;
**Символический язык**, язык символического кодирования - язык программирования, ориентированный на конкретные ЭВМ и основанный на кодировании машинных операций при помощи определенного набора символов;
**Гибридный** (комбинированный) язык - язык программирования, использующий также средства другого языка;
**Графический язык** - язык, предназначенный для написания программ машинной графики и пользования ими.
**Базовый язык** – машинный язык , общий для семейства ЭВМ;
**Язык программирования в системе управления базами данных** (СУБД) с автономным языком.
**Общий язык** - Машинный язык, общий для группы ЭВМ и используемых ими внешних устройств;
**Эталонный язык** - язык, являющийся основой для всех его конкретных версий, являющихся вариантами адаптации эталонного языка к определенным условиям применения и назначения;

**Язык ассемблера**, ассемблер - универсальный язык программирования , относящийся к категории языков низкого уровня, структура которого определяется форматами команд, данными машинного языка и архитектурой ЭВМ. Используется программистами в тех случаях, когда невозможно применение языка высокого уровня или требуются эффективные программы в машинных кодах.
**Декларативный** (непроцедурный язык - язык программирования, который позволяет задавать связи и отношения между объектами и величинами, но не определяет последовательность выполнения действий (например, языки Пролог, *QBE*);
**Императивный** (процедурный) язык - язык программирования, который позволяет в явной форме (при помощи задания выполняемых операторов) определять действия и порядок (последовательность) их выполнения;
**Язык функционального программирования**, функциональный язык - декларативный язык программирования, основанный на понятии функций, которые задают зависимость, но не определяют порядок вычислений.
**Специализированный язык** - язык программирования, ориентированный на решение определенного круга задач;
**Язык описания страниц** [*PDL - Page Description Language*] - специализированный язык, предназначенный для печатающих устройств. Предусматривает возможность использования изображений в формате, независимом от параметров устройства отображения. Наиболее известным языком такого типа является *PostScript*.
**Автономный язык** - специализированный язык высокого уровня, в замкнутых СУБД (СУБД с автономным языком);
**Язык конструирования интерактивных технологий** - с СУБД - язык, предназначенный для описания технологических процессов обработки данных с учетом разделения характера операций по их типам, а также обеспечения диалога с администратором системы;
**Язык манипулирования данными**, ЯМД [*DML - Data Manipulation Language*] - в СУБД - язык, предназначенный для обращения к базе данных и выполнения поиска, чтения и модификации ее записей;
**Язык обработки списков** - специализированный язык, предназначенный для описания процессов обработки данных, представленных в виде списков объектов;
**Язык описания данных** [*DDL - Data Description Language*] - язык, предназначенный для описания концептуальной схемы базы данных;
**Язык описания хранения данных** [*DSDL - Data Storage Description Language*] - язык, предназначенный для описания физической структуры ( схемы ) базы данных;
**Язык описания страниц** - система для кодировки документов, которая позволяет точно описать ее внешний вид после подготовки к выводу на печать или на дисплей. Примером использования такого языка служит *PDF* (*Portable Document Format*), разработанный *Adobe* для хранения и представления изображений страниц.
**Язык представления знаний** [*KRL - Knowledge Representation Language*] - декларативный или декларативно-процедурный язык, предназначенный для представления знаний в памяти ЭВМ (например, языки Лисп и Пролог );
**Язык публикаций** - язык, используемый для публикации алгоритмов и программ;
**Язык спецификаций** - декларативный язык для задания спецификаций программ ;
**Проблемно-ориентированный язык** - язык программирования, предназначенный для решения определенного класса задач (проблем);
**Процедурный** (процедурно-ориентированный) язык - проблемно-ориентированный язык, который облегчает выражение процедуры, как точного алгоритма;
**Язык реального времени** - язык, используемый для программирования задач, в которых критическим является время реакции ЭВМ на сигналы, требующие от неё немедленных действий (например, язык Ада);
**Язык управления пакетом** - набор команд, директив, квалификаторов и правил их использования для управления пакетной обработкой данных;
**Язык управления заданиями** - язык, на котором записывается последовательность команд, управляющих выполнением задания. В отличие от обычных языков программирования, в которых объектами описания являются элементы, связанные с решением отдельной задачи, в языках управления заданиями преобразуемыми объектами являются целые программы и выходные потоки данных, обработанных этими программами.
**Общесетевой командный язык** [*CNCL - Common Network - Command language*] - стандартный в рамках вычислительной сети язык диалогового (интерактивного) поиска данных, предназначенный для унификации работы пользователей с неоднородными базами данных, управляемых различными СУБД ;
**Системный язык** - язык общения оператора ЭВМ с вычислительной системой, представляющий собой совокупность команд оператора и сообщений системы;
**Язык общего назначения**, универсальный язык - язык программирования, ориентированный на решение задач практически из любой области и объединяющий на единой методической основе наиболее существенные свойства и средства современных машино- и проблемноориентированных языков программирования (например, язык ассемблера , ПЛ/1 и др.);
**Язык ориентированный на пользователя** - слабоформализованный язык программирования, близкий к естественному языку ;
**Язык меню** - язык диалога пользователя с системой, основанный на использовании меню

# 2. ИСТОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

По мере развития вычислительной техники возникали разные методики программирования. На каждом этапе создавался новый подход, который помогал программистам с растущим усложнением программ.

На первых компьютерах программирование велось непосредственно в машинных кодах, а основным носителем информации были перфокарты и перфоленты. Программисты обязаны были знать архитектуру машины досконально. Программы были достаточно простыми, что обуславливалось, во-первых, весьма ограниченными возможностями этих машин, и, во-вторых, большой сложностью разработки и, главное, отладки программ непосредственно на машинном языке. Вместе с тем такой способ разработки давал программисту просто невероятную власть над системой. Становилось возможным использование таких хитроумных алгоритмов и способов организации программ, какие и не снились современным разработчикам. Например, применялся самомодифицирующийся код. Знание двоичного представления команд позволяло не хранить некоторые данные отдельно, а встраивать их в код как команды.

Первые программы заключались в установке ключевых переключателей на передней панели вычислительного устройства. Очевидно, таким способом можно было составить только небольшие программы. С развитием компьютерной техники появился машинный язык, с помощью которого программист мог задавать команды, оперируя с ячейками памяти, полностью используя возможности машины. Однако использование большинства компьютеров на уровне машинного языка затруднительно, особенно это касается ввода-вывода. Поэтому от его использования пришлось отказаться. Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающие наличие и типы ошибок, которые надо анализировать. «Слова» на машинном языке называются инструкции, каждая из которых представляет собой одно элементарное действие для центрального процессора, такое, например, как считывание информации из ячейки памяти.

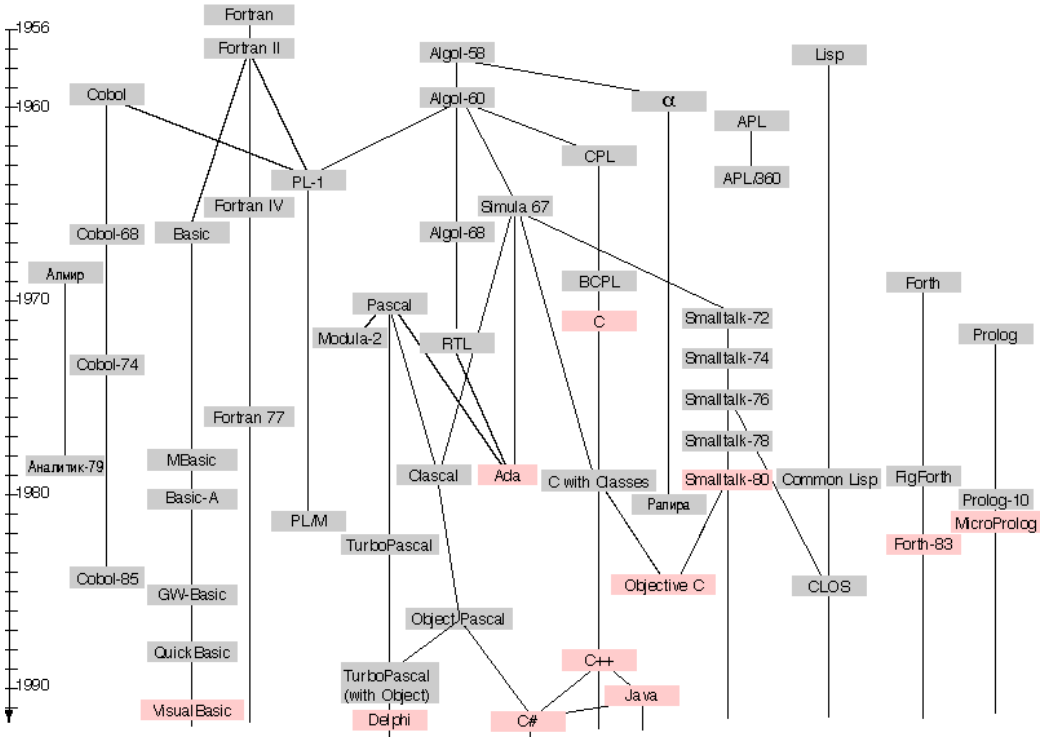


**Рис. 1.** К истории языков программирования.

Каждая модель процессора имеет свой собственный набор машинных команд, хотя большинство из них совпадает. Если Процессор А полностью понимает язык Процессора Б, то процессор А совместим с Процессором Б. Процессор Б будет назваться не совместимым с Процессором А если А имеет команды, не распознаваемые Процессором Б. В случае, когда нужно иметь эффективную программу, вместо машинных языков используются близкие к ним машинно-ориентированные языки - ассемблеры. Люди используют мнемонические команды взамен машинных команд. Но даже работа с ассемблером достаточно сложна и требует специальной подготовки. Например, для процессора *Zilog Z*80 машинная команда 00000101

предписывает процессору уменьшить на единицу свой регистр *B*. На языке ассемблера это же будет записано как *DEC B*.

Следующий шаг был сделан в 1954, когда был создан первый язык высокого уровня -Фортран (*FORTRAN - FORmula TRANslator*). Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека, с помощью них, можно писать программы до нескольких тысяч строк длиной. Однако легко понимаемый в коротких программах, этот язык становился нечитаемым и трудно управляемым, когда дело касалось больших программ. Решение этой проблемы пришло после изобретения языков структурного программирования (*structured programming language*), таких как Алгол (1958), Паскаль (1970), Си (1972). Структурное программирование предполагает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций безусловного перехода (GOTO), автономные подпрограммы, поддержка рекурсии и локальных переменных. Суть такого подхода заключается в возможности разбиения программы на составляющие элементы.

Также создавались функциональные (аппликативные) языки (Пример: *Lisp - LISt Processing*, 1958) и логические языки (пример: *Prolog - PROgramming in LOGic*, 1972). Хотя структурное программирование, при его использовании, дало выдающиеся результаты, даже оно оказывалось несостоятельным тогда, когда программа достигала определенной длины. Для того чтобы написать более сложную (и длинную) программу, нужен был новый подход к программированию. В итоге в конце 1970-х и начале 1980-х были разработаны принципы объекто-ориентированного программирования. ООП сочетает лучшие принципы структурного программирования с новыми мощными концепциями, базовые из которых называются инкапсуляцией, полиморфизмом и наследованием. Примером объектно-ориентированных языков являются: *Object Pascal, C++, Java* и др. ООП позволяет оптимально организовывать программы, разбивая проблему на составные части, и работая с каждой по отдельности. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче.

## 3. ВИДЫ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Понятие **компьютерный язык** (*computer language*) относится к языкам, ассоциируемым с компьютерной техникой. Чаще всего, этот термин соответствует понятию языка программирования, однако это соответствие не является вполне однозначным. Так, например, языки размётки (такие как *HTML*) не являются языками программирования, однако определённо относятся к компьютерным языкам. Компьютерный язык, как и любой другой язык, появляется, когда требуется передать информацию из одного источника другому. Языки программирования способствуют обмену информацией между программистами и компьютерами, языки разметки текста определяют понятную для людей и компьютеров структуру документов и т. п. Нередко понятие компьютерный язык также отождествляют со сленгом, распространённым среди людей, так или иначе общающихся с компьютерами.

**Язы́к программи́рования -** формальная знаковая система, предназначенная для записи программ, задающих алгоритм в форме, понятной для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

**Знаковая система - совокупность условных знаков и правил их взаимосвязи.**

Со времени создания первых программируемых машин человечество придумало около десяти тысяч языков программирования. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие известны миллионам людей. Создатели языков по-разному толкуют понятие *язык программирования*. К наиболее распространённым утверждениям, признаваемым большинством разработчиков, относятся следующие:

*Функция:* язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.

*Задача:* язык программирования отличается от естественных языков тем, что предназначен для передачи команд и данных от человека компьютеру, в то время, как естественные языки используются для общения людей между собой. В принципе, можно обобщить определение «языков программирования» - это способ передачи команд, приказов, чёткого руководства к действию; тогда как человеческие языки служат также для обмена информацией.

*Исполнение:* язык программирования может использовать специальные конструкции для определения и манипулирования структурами и управления процессом вычислений.

Язык программирования может быть представлен в виде набора спецификаций, определяющих его синтаксис и семантику.

Для многих широко распространённых 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 и решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

Современные цифровые компьютеры обычно являются двоичными и данные хранят в двоичном (бинарном) коде (хотя возможны реализации и в других системах счисления). Эти данные как правило отражают информацию из реального мира (имена, банковские счета, измерения и др.), представляющую высокоуровневые концепции.

Особая система, по которой данные организуются в программе, - это *система типов* языка программирования; разработка и изучение систем типов известна под названием теория типов. Языки могут быть классифицированы как системы со *статической типизацией* и языки с *динамической типизацией*. Статически-типизированные языки могут быть в дальнейшем подразделены на языки с *обязательной декларацией*, где каждая переменная и объявление функции имеет обязательное объявление типа, и языки с *выводимыми типами*. Иногда динамически-типизированные языки называются латентно-типизированными. Системы типов в языках высокого уровня позволяют определять сложные, составные типы, так называемые структуры данных. Как правило, структурные типы данных образуются как декартово произведение базовых (атомарных) типов и ранее определённых составных типов. Основные структуры данных (списки, очереди, хеш-таблицы, двоичные деревья и пары) часто представлены особыми синтаксическими конструкциями в языках высокого уровня. Такие данные структурируются автоматически.


**Рис. 2.** Классификация языков программирования

Существует несколько подходов к определению семантики языков программирования. Наиболее широко распространены разновидности следующих трёх: операционного, денотационного (математического) и деривационного (аксиоматического). При описании семантики в рамках операционного подхода обычно исполнение конструкций языка программирования интерпретируется с помощью некоторой воображаемой (абстрактной) ЭВМ.

Деривационная семантика описывает последствия выполнения конструкций языка с помощью языка логики и задания пред- и постусловий. Денотационная семантика оперирует понятиями, типичными для математики - множества, соответствия, а также суждения, утверждения и др. Язык программирования строится в соответствии с той или иной базовой моделью вычислений и парадигмой программирования.

Несмотря на то, что большинство языков ориентировано на императивную модель вычислений, задаваемую фон-неймановской архитектурой ЭВМ, существуют и другие подходы. Можно упомянуть языки со стековой вычислительной моделью (*Forth, Factor, Postscript* и др.), а также функциональное (Лисп, *Haskell, ML* и др.) и логическое программирование (Пролог) и язык Рефал, основанный на модели

вычислений, введённой советским математиком А.А.Марковым -младшим. В настоящее время также активно развиваются проблемно-ориентированные, декларативные и визуальные языки программирования.

Языки программирования могут быть разделены на компилируемые и интепретируемые.
Программа на компилируемом языке при помощи специальной программы компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполнимый модуль, который может быть запущен на выполнение как отдельная программа. Другими словами, компилятор переводит исходный текст программы с языка программирования высокого уровня в двоичные коды инструкций процессора. Если программа написана на интерпретируемом языке, то интерпретатор непосредственно выполняет (интерпретирует) исходный текст без предварительного перевода. При этом программа остаётся на исходном языке и не может быть запущена без интерпретатора. Можно сказать, что процессор компьютера - это интерпретатор машинного кода. Кратко говоря, компилятор переводит исходный текст программы на машинный язык сразу и целиком, создавая при этом отдельную исполняемую программу, а интерпретатор выполняет исходный текст прямо во время исполнения программы.

Разделение на компилируемые и интерпретируемые языки является несколько условным. Так, для любого традиционно компилируемого языка, как, например, Паскаль, можно написать интерпретатор. Кроме того, большинство современных «чистых» интерпретаторов не исполняют конструкции языка непосредственно, а компилируют их в некоторое высокоуровневое промежуточное представление (например, с разыменованием переменных и раскрытием макросов). Для любого интерпретируемого языка можно создать компилятор - например, язык Лисп, изначально интерпретируемый, может компилироваться без каких бы то ни было ограничений. Создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Как правило, скомпилированные программы выполняются быстрее и не требуют для выполнения дополнительных программ, так как уже переведены на машинный язык. Вместе с тем, при каждом изменении текста программы требуется её перекомпиляция, что создаёт трудности при разработке. Кроме того, скомпилированная программа может выполняться только на том же типе компьютеров и, как правило, под той же операционной системой, на которую был рассчитан компилятор. Чтобы создать исполняемый файл для машины другого типа, требуется новая компиляция.

Интерпретируемые языки обладают некоторыми специфическими дополнительными возможностями, кроме того, программы на них можно запускать сразу же после изменения, что облегчает разработку. Программа на интерпретируемом языке может быть зачастую запущена на разных типах машин и операционных систем без дополнительных усилий. Однако интерпретируемые программы выполняются заметно медленнее, чем компилируемые, кроме того, они не могут выполняться без дополнительной программы-интерпретатора. Некоторые языки, например, *Java* и *C#*, находятся между компилируемыми и интерпретируемыми. А именно, программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код. Далее байт-код выполняется виртуальной машиной. Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (*Just-in-time compilation*, *JIT*). Для *Java* байт-код исполняется виртуальной машиной *Java* (*Java Virtual Machine, JVM*), для *C# - Common Language Runtime*. Подобный подход в некотором смысле позволяет использовать плюсы как интерпретаторов, так и компиляторов. Следует упомянуть также оригинальный язык Форт (*Forth*) имеющий и интерпретатор и компилятор.

Современные языки программирования рассчитаны на использование ASCII, то есть доступность всех *графических* символов ASCII является необходимым и достаточным условием для записи любых конструкций языка. *Управляющие* символы *ASCII* используются ограниченно: допускаются только возврат каретки *CR*, перевод строки *LF* и горизонтальная табуляция *HT* (иногда также вертикальная табуляция *VT* и переход к следующей странице *FF*).

Ранние языки, возникшие в эпоху 6-битных символов, использовали более ограниченный набор. Например, алфавит Фортрана включает 49 символов (включая пробел): *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 = + - * / () . , $ ' :*Заметным исключением является язык *APL*, в котором используется очень много специальных символов. Использование символов за пределами *ASCI*I (например, символов *KOI8-R* или символов Юникода) зависит от реализации: иногда они разрешаются только в комментариях и символьных/строковых константах, а иногда и в идентификаторах. В СССР существовали языки, где все ключевые слова писались русскими буквами, но большу́ю популярность подобные языки не завоевали

Расширение набора используемых символов сдерживается тем, что многие проекты по разработке программного обеспечения являются международными. Очень сложно было бы работать с кодом, где имена одних переменных записаны русскими буквами, других - арабскими, а третьих - китайскими иероглифами. Вместе с тем, для работы с текстовыми данными языки программирования нового поколения (*Delphi* 2006, *C#, Java*) поддерживают *Unicode*.

Классы языков программирования включают: функциональные, процедурные (императивные, стековые, векторные, аспектно-ориентированные, декларативные, динамические, учебные, описания интерфейсов, прототипные, объектно-ориентированные, рефлексивные, логические, параллельного программирования, скриптовые (сценарные), эзотерические, с русским синтаксисом.

# 4. КОНКРЕТНЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

**Ада** [*Ada*]

Язык программирования высокого уровня, ориентированный на применение в системах реального времени и предназначенный для автоматизации задач управления процессами и/или устройствами, например, в бортовых (корабельных, авиационных и др.) ЭВМ. Разработан по инициативе Министерства Обороны США около 1980 на основе языка Паскаль для решения задач управления космическими полётами. Назван в честь английского математика Ады Августы Байрон (Лавлейс, дочь поэта Байрона, 1815-1851). Язык Ада - наиболее универсальный среди созданных языков. Однако трансляторы с этого языка пока не получили достаточного распространения. Язык замечателен тем, что очень много ошибок может быть выявлено на этапе компиляции. Кроме того, поддерживаются многие аспекты программирования, которые часто отдаются на откуп операционной системе (параллелизм, обработка исключений). В 1995 принят стандарт языка Ада95, который развивает предыдущую версию, добавляя в неё объекно- ориентированность и исправляя некоторые неточности. Оба этих языка не получили широкого распространения вне военных и прочих крупномасштабных проектов (авиация, железнодорожные перевозки, обеспечение жизнедеятельности космонавтов). Основной причиной является сложность освоения языка и достаточно громоздкий синтаксис (значительно более громоздкий, чем Поскалбь). Ада - структурный, модульный, объектно-ориентированный язык программирования, содержащий высокоуровневые средства программирования параллельных процессов. Синтаксис Ады унаследован от языков типа Алгол или Паскаль, но расширен, а также сделан более строгим и логичным. Ада - язык со строгой типизацией, в нём исключена работа с объектами, не имеющими типов, а автоматические преобразования типов сведены к абсолютному минимуму. Разработка программного обеспечения на Аде в целом обходится на 60% дешевле, а разработанная программа имеет в 9 раз меньше дефектов, чем при использовании языка Си.

**Алгол** [*ALGOL - ALGOrithmic Language*]

Язык программирования высокого уровня, ориентированный на описание алгоритмов решения вычислительных задач. Был создан в 1958 специалистами западно-европейских стран (команда во главе с Петером Науром) для научных исследований. Этот язык дал начало целому семейству Алгол-подобных языков (важнейший представитель - Паскаль). Язык Алгол — общепризнанный язык для публикации алгоритмов решения научных задач, построен на четких и полных определениях. Для Алгола характерны строгие, но негибкие структуры данных и программ. Алгол труден для реализации на большинстве ЭВМ, поэтому используются неполные варианты языка или его расширения. Версия этого языка Алгол-60 была принята Международной конференцией в Париже (1960) и широко использовалась на ЭВМ 2-го поколения. Версия Алгол-68, разработанная группой специалистов Международной федерации по обработке информации (ИФИП) в 1968, получила статус международного универсального языка программирования, ориентированного на решение не только вычислительных, но и информационных задач. Она не нашла столь широкого практического применения, как первая версия, но была весьма популярна в кругах теоретиков. Язык был достаточно интересен, так как обладал многими уникальными на так момент характеристиками. Алгол относится к языкам высокого уровня (*high-level language*) и позволяет легко переводить алгебраические формулы в программные команды. Этот язык так же, как и Фортран, предназначался для решения научно-технических задач. Кроме того, этот язык применялся как средство обучения основам программирования – искусства составления программ. Алгол был популярен в Европе, в том числе СССР, в то время как сравнимый с ним Фортран был распространен в США и Канаде. Хотя в настоящее время Алгол практически не используется, он послужил основой или оказал существенное влияние на разработку более современных языков, например, Ада, Паскаль и др. Даже когда язык Алгол почти перестал

использоваться для программирования, он ещё оставался официальным языком для публикации алгоритмов.

**АПЛ**
Язык АПЛ применяется для обработки структурных данных (векторов, матриц) и использует иероглифическую запись программных текстов. Из-за большого числа иероглифов (около 100) его иногда называют китайским Бейсиком.

**Ассемблер** [*assembly language*, или *assembler* – собирать, компоновать]
Язык Ассемблера – это символическое представление машинного языка – язык низкого уровня. Этот язык так же, как и Фортран, предназначался для решения научно-технических задач. Кроме того, этот язык применялся как средство обучения основам программирования – искусства составления программ; он облегчает процесс программирования по сравнению с программированием в машинных кодах. Этот язык так же, как и Фортран, предназначался для решения научно-технических задач. Кроме того, этот язык применялся как средство обучения основам программирования – искусства составления программ. В языке ассемблера используются символьные обозначения команд, которые легко понятны и быстро запоминаются. Вместо последовательности двоичных кодов команд записываются их символьные обозначения, а вместо двоичных адресов данных, используемых при выполнении команды, - символьные имена этих данных, выбранные программистом. Иногда язык ассемблера называют мнемокодом или автокодом. Программисту не обязательно употреблять настоящие адреса ячеек памяти с размещенными в них данными, участвующими в операции, и вычисляемые результаты, а также адреса тех команд, к которым программа не обращается. Некоторые задачи, например, обмен с нестандартными устройствами обработки данных сложных структур невозможно решить с помощью языков программирования высокого уровня. Это под силу ассемблеру. Ассемблер - машинный язык, и программист реализующий какую-либо задачу на языках высокого уровня, с помощью Ассемблера может определить осмыслено ли решение данной задачи, с точки зрения использования ЭВМ. Умея разобраться в распечатке языка ассемблера, даёт возможность облегчить поиск ошибок в программах, т.к. некоторые языки являются компиляторами.

**Бейсик** [*BASIC - Beginner's All-purpose Symbolic Instruction Code* - Многоцелевой Символический Обучающий Код для Начинающих]
Язык программирования высокого уровня - многоцелевой язык символических инструкций для начинающих). Язык задумывался в первую очередь как средство обучения и как первый изучаемый язык программирования. Он предполагался легко интерпретируемым и компилируемым. Разработан в 1963 в Дартмутском колледже Т. Куртом и Д. Кемени. Язык был основан частично на Фортран II и частично на Алгол-60, с добавлениями, делающими его удобным для работы в режиме разделения времени и, позднее, обработки текста и матричной арифметики. Первоначально предназначался для обучения программированию. Отличается простотой, легко усваивается начинающими программистами благодаря наличию упрощенных конструкций языка Фортран и встроенных математических функций, алгоритмов и операторов. В языке встроены удобные функции для работы с экраном дисплея, клавиатурой, внешними накопителями, принтером, каналами связи. Это позволяет относиться к Бейсику как к продолжению аппаратуры ПЭВМ. Системы Бейсика работают в режиме интерпретации, что способствует сокращению характерного цикла в работе программиста: составление программы - пробное исполнение - исправление ошибок - повторное исполнение. Существует множество различных версий Бейсика, которые не полностью совместимы друг с другом. Некоторые реализации Бейсика включают средства обработки данных и наборов данных. Большинство версий Бейсика используют интерпретатор, который преобразует его компоненты в машинный код и позволяет запускать программы без промежуточной трансляции. Некоторые более совершенные версии Бейсика позволяют использовать для этой цели трансляторы. На *IBM PC* широко используются *Quick Basic* фирмы *Microsoft, Turbo Basic* фирмы *Borland* и *Power Basic* (усовершенствованная версия *Turbo Basic*, распространяемая фирмой *Spectra Publishing* ). В начале 1999 фирма *Microsoft* выпустила версию языка *Visual Basic* 6.0 (*VB* 6.0), предназначенного для создания многокомпонентных программных приложений для систем уровня предприятий. Бейсик наряду с Несмотря на высказывания снобов - сторонников языков Си и Паскаля, Бейсик считается деловым языком, снабженным мощными средствами решения специфических задач, которые обычно большинство пользователей решают при помощи небольших компьютеров, а именно: работая с файлами и выводя текстовое и графическое изображение на экране дисплея. Несмотря на отдельные недостатки Бейсика, никто не будет отрицать, что Кемени и Куртс достигли основной цели: сделать программирование

доступнее для большего числа людей. Более того, с появлением транслятора *QuickBasic* фирмы *Microsoft* разработчики получили возможность строить на Бейсике приложения из раздельно откомпилированных модулей, некоторые из которых могут быть написаны на других языках. Теперь, как и в случае других ведущих языков программирования, разработчик имеет выбор из нескольких промышленных библиотек подпрограмм, которые содержат готовые решения для распространенных задач программирования.

**Кобол** [*COBOL - COmmon Buisiness-Oriented Language* – общий язык, ориентированный на бизнес]
Язык программирования высокого уровня, разработанный в 1960 ассоциацией КАДАСИЛ для решения коммерческих и экономических задач. Разработчиком первого единого стандарта Кобола являлась Грейс Хоппер (бабушка Кобола). Отличается развитыми средствами работы с файлами. Поскольку команды программ, написанных на этом языке, активно используют обычную английскую лексику и синтаксис, Кобол рассматривается как один из самых простых языков программирования. В настоящее время используется для решения экономических, информационных и других задач. На Коболе написаны тысячи прикладных коммерческих систем. Отличительной особенностью языка является возможность эффективной работы с большими массивами данных, что характерно именно коммерческих приложений. Популярность Кобола столь высока, что даже сейчас, при всех его недостатках (по структуре и замыслу Кобол во многом напоминает Фортран) появляются новые его диалекты и реализации. Так недавно появилась реализация Кобола, совместимая с *Microsoft .NET*, что потребовало, вероятно, внесения в язык некоторых черт объектно-ориентированного языка. Кобол обычно критикуется за многословность и громоздкость, поскольку одной из целей создателей языка было максимально приблизить конструкции к английскому языку. (До сих пор Кобол считается языком программирования, на котором было написано больше всего строк кода). В то же время, Кобол имел прекрасные для своего времени средства для работы со структурами данных и файлами, что обеспечило ему долгую жизнь в бизнес приложениях, по крайней мере, в США.

**Лекс**
генератор программ лексического анализа. Лексический анализ – это распознавание лексем во входном потоке символов. Предположим, что задано некоторое конечное множество слов (лексем) в некотором языке и некоторое входное слово. Необходимо установить, какой элемент множества (если он существует) совпадает с данным входным словом. Обычно лексический анализ выполняется так называемым лексическим анализатором. Лексический анализатор – это программа. Лексический анализ применяется во многих случаях, например, для построения пакетного редактора или в качестве распознавателя директив в диалоговой программе и т.д. Однако, наиболее важное применение лексического анализатора – использование его в компиляторе. Здесь лексический анализатор выполняет функцию программы ввода данных. Лексический анализатор выполняет первую стадию компиляции – читает строки компилируемой программы, выделяет лексемы и передает их на дальнейшие стадии компиляции (грамматический разбор, кодогенерацию и т.д.). Лексический анализатор распознает тип каждой лексемы и соответствующим образом помечает ее. Например, при компиляции Си-программы могут быть выделены следующие типы лексем: число, идентификатор, оператор, ограничитель и т.д. Лексический анализатор должен не только выделить лексему, но и выполнить некоторые преобразования. Например, если лексема – число, то его необходимо перевести во внутреннюю (двоичную) форму записи как число с плавающей или фиксированной запятой. А если лексема – идентификатор, то его необходимо разместить в таблице, чтобы в дальнейшем обращаться к нему не по имени, а по адресу в таблице. Хотя лексический анализ по своей идее прост, тем не менее, эта фаза работы компилятора часто занимает больше времени, чем любая другая. Частично это происходит из-за необходимости просматривать и анализировать исходный текст символ за символом. Иногда даже бывает необходимо вернуть прочитанный символ во входной поток с тем, чтобы повторить просмотр и анализ.

**Лисп** [*LISP - LISt Information Symbol Processing Processing* – обработка списков]
Алгоритмический язык, разработанный в 1962 Дж. Маккарти и предназначенный для манипулирования перечнями элементов данных (скорее для работы со строками символов, нежели для работы с числами). Используется преимущественно в университетских лабораториях США для решения задач, связанных с искусственным интеллектом. В настоящее время Лисп успешно применяется в экспертных системах, системах аналитических вычислений и т.п. Обширность области возможных приложений Лиспа вызвала появление множества различных диалектов Лиспа. Это легко объяснимо: применение Лиспа для понимания

естественного языка требует определенного набора базисных функций, отличных, например, от используемого в задачах медицинской диагностики. Существование множества различных диалектов Лиспа привело к созданию в начале 80-х гг. *Common LISP* Комитета, который должен был выбрать наиболее подходящий диалект Лиспа и предложить его в качестве основного. Этот диалект, выбранный Комитетом в 1985, получил название *Common LISP*. В дальнейшем он был принят в университетах США, а также многими разработчиками систем искусственного интеллекта, в качестве основного диалекта языка Лисп. В Европе для работ по искусственному интеллекту предпочитают использовать Пролог.

Функциональные языки Лисп, Пролог и СНОБОЛ применяются для разработки систем искусственного интеллекта. Эти языки ориентированы на обработку символьной информации, требуют больших массивов данных и стали применяться в ПЭВМ в связи с появлением дешевой полупроводниковой памяти, позволяющей довести объем ОЗУ до нескольких мегабайт. Языки этого класса относятся к языкам представления знаний. Язык Лисп применяется для программирования интеллектуальных задач - общение на естественном языке, доказательство теорем, принятие решений и т. п.

Язык программирования Лисп существенно отличается от других языков программирования, таких, как Паскаль, Си и т.п. Работа с символами и работа с числами как с основными элементами требует разных способов мышления. Первоначально Лисп был задуман как теоретическое средство для рекурсивных построений, а сегодня он превратился в мощное средство, обеспечивающее программиста разнообразной поддержкой, позволяющей ему быстро строить прототипы весьма и весьма серьезных систем. Математическая ясность и предельная четкость Лиспа – это еще не всё. Главное – Лисп позволяет сформулировать и запомнить «идиомы», столь характерные для проектов по искусственному интеллекту.

Лисп основан на представлении программы системой линейных списков символов, которые притом являются основной структурой данных языка. Лисп считается вторым после Фортрана старейшим высокоуровневым языком программирования. Этот язык широко используется для обработки символьной информации и применяется для создания программного обеспечения, имитирующего деятельность человеческого мозга. Любая программа на Лиспе состоит из последовательности выражений (форм). Результат работы программы состоит в вычислении этих выражений. Все выражения записываются в виде списков - одной из основных структур Лиспа, поэтому они могут легко быть созданы посредством самого языка. Это позволяет создавать программы, изменяющие другие программы или макросы, позволяющие существенно расширить возможности языка. Основной смысл Лисп-программы «жизнь» в символьном пространстве: перемещение, творчество, запоминание, создание новых миров и т.д. Лисп как метафора мозга, символ, метафора сигнала.

**ЛОГО** [*LOGO* от греч. *logos* - слово]
Язык программирования высокого уровня - диалоговый процедурный язык, реализованный на принципе интерпретации и работающий со списками, текстами, графическими средствами и т. д. Создан в Массачусетском технологическом институте в 1970 для целей обучения математическим понятиям. Используется также в школах и пользователями ПЭВМ при написании программ для создания чертежей на экране монитора и управления перьевым графопостроителем. Язык перспективен для обучения, создания электронных игрушек и т. д. Относится к классу проблемно-ориентированных языков (как и Лого, *CPSS*, Форт и Смолток).

**Модула-2**
развитие Паскаля - обладает лучшими средствами для обработки больших программных комплексов и позволяет более эффективно использовать особенности аппаратуры. Этот язык призван заполнить ниши между Паскалем и СИ. Возможно скоро Модула-2 станет наиболее популярным среди всех языков программирования.

**Паскаль** [*PASCAL - Program Applique a la Selection et la Compilation Automatique de la Litterature*]
Процедурно-ориентированный язык программирования высокого уровня, разработан в 1970 швейцарцем Н. Виртом, первоначально для обучения программированию в университетах. Назван в честь французского математика XVII века Блеза Паскаля. В своей начальной версии Паскаль имел довольно ограниченные возможности, поскольку предназначался для учебных целей, однако последующие его доработки позволили сделать его хорошим универсальным языком, широко используемым для написания больших и сложных программ. Существует ряд версий языка (например, *ETH Pascal, USD Pascal, Turbo Pascal*) и систем программирования на этом языке для разных типов ЭВМ. Для *IBM PC* наиболее популярной является система *Turbo Pascal фирмы Borland* (США). Язык Паскаль является одним из наиболее популярных

языков программирования и применяется для разработки системных и прикладных программ, в частности, для персональных ЭВМ. Язык Паскаль создан вначале исключительно для учебных целей и изящно реализовал большинство идей структурного программирования. Достоинства языка оказались столь значительными, что он приобрел огромную популярность для самых различных приложений. В частности, компилятор *Turbo Pascal*, снабженный интерактивным редактором, позволяет создавать достаточно сложное программное обеспечение - системы управления базами данных, графические пакеты и т. д. Язык замечателен тем, что это первый широко распространенный язык для структурного программирования. Впервые оператор безусловного перехода перестал играть основополагающую роль при управлении порядком выполнения операторов. В этом языке также внедрена строгая проверка типов, что позволило выявлять многие ошибки на этапе компиляции. Особенностями языка являются строгая типизация и наличие средств структурного программирования. По мнению Н.Вирта, язык должен способствовать дисциплинированию программирования, поэтому, наряду со строгой типизацией, в Паскале сведены к минимуму возможные синтаксические неоднозначности, а сам синтаксис интуитивно понятен даже при первом знакомстве с языком. Язык Паскаль учит не только тому, как правильно написать программу, но и тому, как правильно разработать метод решения задачи, подобрать способы представления и организации данных, используемых в задаче.

**Пролог** [*PROLOG - PROgramming in LOGic* - ПРОграммирование на языке ЛОГики]
Язык программирования высокого уровня декларативного типа, предназначенный для разработки систем и программ искусственного интеллекта. Относится к категории языков пятого поколения. Разработан в 1971 в университете г. Марсель (Франция), относится к числу широко используемых и постоянно развиваемых языков. Считается языком будущего. В основе этого языка лежат законы математической логики. Последняя его версия *Prolog* 6.0. Язык Пролог приобрёл в последние годы большую популярность в связи с японским проектом создания вычислительных систем пятого поколения. Он предназначен для создания широкого класса систем искусственного интеллекта, в том числе и персональных экспертных систем. Пролог - язык, предназначенный для поиска решений. Это декларативный язык, то есть формальная постановка задачи может быть использована для её решения. Пролог определяет логические отношения в задаче, как отличные от пошагового решения этой задачи. Центральной частью Пролога являются средства логического вывода, которые решают запросы, используя заданное множество фактов и правил, к которым обращаются как к утверждениям. Пролог также не имеет деления переменных на типы и может динамически добавлять правила и факты к средствам вывода. Таким образом, это гибкий язык, и он более пригоден для объектно-ориентированного расширения, чем язык со строго заданными типами, например, Паскаль. Пролог++ представляет собой дополнение к стандартному Прологу. Все свойства языка по-прежнему доступны программистам. Следовательно, Пролог++ можно отнести к группе гибридных языков, представителями которой считаются *Object Pascal* и *C++*. Расширение Пролог++ поддерживает все свойства, присущие обычно объектно-ориентированным языкам: концепции объектов и классов, единичное и многократное наследование, разбиение на подклассы и передачу сообщений. Поддерживаются также некоторые усовершенствованные свойства, существующие в таких языках, как *C++* и *Smalltalk*, включая общие и частные методы. Интересным свойством является поддержка в языке программирования с управлением данными. Эта техника, которая может быть еще названа программированием, «управляемым событиями», используется в большинстве языков объектно-ориентированного программирования, особенно в тех, которые разработаны для машин с интерфейсом, управляемым «мышью». Объектно-ориентированная программа реагирует на события, которые определяют поток управления. В Прологе++ программирование с управлением данными достигается при помощи концепции демонов. Демон представляет собой объект, методы которого вызываются в случае определенных событий и могут быть таким образом использованы для поддержки программирования с управлением данными. Сам язык основан на концепции передачи сообщений. Программа на Прологе++ строится вокруг множества объектов Пролога++, которые обмениваются сообщениями. В этом смысле Пролог++ ближе к чистому объектно-ориентированному языку, такому, как *Smalltalk*, чем *C++* или *Object Pascal*. Определения объектов строятся исходя из вызовов *Open_Object* [имя_объекта] и *Close_Object* [имя_объекта], а методы определяются практически так же, как в других объектно-ориентированных языках. Для задания наследования можно явным образом указать, какой метод какого объекта должен наследоваться, что является необходимым для многократного наследования. Как и язык Лисп, Пролог применяется, в основном, при проведении исследований в области программной имитации деятельности мозга человека. В отличие от описанных выше языков, этот язык не является алгоритмическим. Он относится к так называемым дескриптивным (*descriptive* – описательный) –

описательным языкам. Дескриптивный язык не требует от программиста разработки всех этапов выполнения задачи. Вместо этого, в соответствии с правилами такого языка, программист должен описать базу данных, соответствующую решаемой задаче, и набор вопросов, на которые нужно получить ответы, используя данные из этой базы.

**Рефал**
разработан в России (1966) ИПМ АН СССР. Этот язык прост и удобен для описания манипуляций над произвольными текстовыми объектами. Рефал применяется при разработке трансляторов с алгоритмических языков как универсальных и проблемно – ориентированных, так и автокодов. Рефал имеет такие важные сферы применения, как машинное выполнение громоздких аналитических выкладок в теоретической физике и прикладной математике; проектирование информационных систем, осуществляющих нетривиальную логическую обработку информации; машинное доказательство теорем; моделирование целенаправленного поведения; разработка диалоговых обучающих систем; исследования в области искусственного интеллекта и т.п. Программирование на Рефале имеет специфику, связанную, прежде всего, с тем, что Рефал является языком функционального типа в отличие от обычных операторных языков типа Алгол, Фортран и т.д. Если программа на операторных языках – совокупность приказов-операторов, то программа на Рефале представляет собой по существу описание связей и отношений между определенными понятиями. Вследствие того, что в Рефале программист сам определяет структуру обрабатываемой информации, эффективность программы существенно зависит от удачного или неудачного выбора этой структуры. Для задания структур в Рефале используются скобки, а специфика всех реализаций языка такова, что использование скобок резко повышает эффективность выполнения программы. Это достигается с помощью адресного соединения скобок. Определенной спецификой обладают и переменные типа «выражения» – имеется в виду их способность удлиняться при отождествлении.

**Си** [*C* – буква английского алфавита]
Многоцелевой язык программирования высокого уровня общего назначения, разработанный Д. Ритчи и Керниганом (1970) на базе языка BCPL. Машино-ориентированный язык используется на миниЭВМ и ПЭВМ. В нём объединяются достоинства низкоуровневых возможностей ассемблеров и мощных выразительных средств языков программирования высокого уровня. Является базовым языком операционной системы *Unix* и *MS DOS*, однако применяется и вне этих систем, для написания быстродействующих и эффективных программных продуктов, включая и операционные системы. С часто называют «переносимым ассемблером», имея в виду то, что он позволяет работать с данными практически так же эффективно, как на ассемблере, предоставляя при этом структурированные управляющие конструкции и абстракции высокого уровня (структуры и массивы). Именно с этим связана его огромная популярность и поныне. И именно это является его ахиллесовой пятой. Компилятор *C* очень слабо контролирует типы, поэтому очень легко написать внешне совершенно правильную, но логически ошибочную программу. Для *IBM PC* имеется ряд популярных версий языка Си, в том числе - *Turbo C* (фирмы *Borland*), *Microsoft C* и *Quick C* (фирмы *Microsoft*), а также *Zortech C* (фирмы *Symantec*). Многие из указанных версий обеспечивают также работу с Си и Си. Си хорошо известен своей эффективностью, экономичностью, и переносимостью. Указанные преимущества Си обеспечивают хорошее качество разработки почти любого вида программного продукта. Использование Си в качестве инструментального языка позволяет получать быстрые и компактные программы. Во многих случаях программы, написанные на Си, сравнимы по скорости с программами, написанными на языке ассемблера. При этом они имеют лучшую наглядность и их более просто сопровождать. Си сочетает эффективность и мощность в относительно малом по размеру языке. Си – это замечательный язык, и хотя некоторым он не нравится, но все же большинство программистов его любят. На Си вы можете создавать программы, которые делают все, что вы пожелаете. Нет другого такого языка, который бы так же стимулировал к программированию. Создается впечатление, что остальные языки программирования воздвигают искусственные препятствия для творчества, а Си – нет. Использование этого языка позволяет сократить затраты времени на создание работающих программ. Си позволяет программировать быстро, эффективно и предсказуемо. Си позволяет использовать все возможности вашей ЭВМ. Этот язык создан программистом для использования другими программистами, чего о других языках программирования сказать нельзя. Си обеспечивает полный набор операторов структурного программирования. Многие операции Си соответствуют машинным командам, и поэтому допускают прямую трансляцию в машинный код. Разнообразие операций позволяет выбирать их различные наборы для минимизации результирующего кода. Си поддерживает указатели на переменные и

функции. Указатель на объект программы соответствует машинному адресу этого объекта. Посредством разумного использования указателей можно создавать эффективно-выполняемые программы, так как указатели позволяют ссылаться на объекты тем же самым путем, как это делает машина. Си поддерживает арифметику указателей, и тем самым позволяет осуществлять непосредственный доступ и манипуляции с адресами памяти. В своем составе Си содержит препроцессор, который обрабатывает текстовые файлы перед компиляцией. Среди его наиболее полезных приложений при написании программ на Си являются: определение программных констант, замена вызовов функций аналогичными, но более быстрыми макросами, условная компиляция. Препроцессор не ограничен процессированием только исходных текстовых файлов Си, он может быть использован для любого текстового файла. Си-гибкий язык, позволяющий принимать в конкретных ситуациях самые разные решения. Тем не менее, Си налагает незначительные ограничения в таких, например, действиях, как преобразование типов. В языке Си есть ряд недостатков. Ведь от них не защищен не один проект, в том числе проект разработки и выполнения программ, на языке Си: Язык Си предъявляет достаточно высокие требования к квалификации использующего его программиста. При изучении Си желательно иметь представление о структуре и работе компьютера. Уровень старшинства некоторых операторов не является общепринятым, некоторые синтаксические конструкции могли бы быть лучше. Тем не менее, Си – чрезвычайно эффективный и выразительный язык, пригодный для широкого класса задач. Сегодня практически все основные операционные системы были написаны на Си или *C*++. Си имеется в наличии на большинстве компьютеров. Он не зависит от аппаратной части. В конце 70-х годов Си превратился в то, что мы называем «традиционный Си». В 1983 Американским комитетом национальных стандартов в области компьютеров и обработки информации был учреждён единый стандарт этого языка. Этот язык имеет богатые средства, позволяет писать гибкие программы, использующие все возможности современных персональных компьютеров.

**Си++** [*C*++] - Язык программирования высокого уровня, созданный Б. Страуструпом (1986) на базе языка Си (в язык *C* были добавлены объектно-ориентированные черты, взятые из *Simula*, и исправлены некоторые ошибки и неудачные решения языка). *C*++ является расширенной версией языка *C*, реализующей принципы объектно-ориентированного программирования. Используется для создания сложных программ. Для *IBM PC* наиболее популярной является система *Turbo C*++ фирмы *Borland* (США). Язык стал основой для разработки современных больших и сложных проектов. У него имеются, однако же, и слабые стороны, вытекающие из требований эффективности. Си++ - универсальный язык программирования, задуманный так, чтобы сделать программирование более приятным для серьезного программиста. Помимо возможностей, которые дает Си, Си++ предоставляет гибкие и эффективные средства определения новых типов. Используя определения новых типов, точно отвечающих концепциям приложения, программист может разделять разрабатываемую программу на легко поддающиеся контролю части. Такой метод построения программ часто называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. Такие объекты просты и надежны в использовании в тех ситуациях, когда их тип нельзя установить на стадии компиляции. Программирование с применением таких объектов часто называют объектно-ориентированным. При правильном использовании этот метод дает более короткие, проще понимаемые и легче контролируемые программы. Изначально Си++ был разработан, чтобы автору и его друзьям не приходилось программировать на ассемблере, Си или других современных языках высокого уровня. Основным его предназначением было сделать написание хороших программ более простым и приятным для отдельного программиста. Плана разработки Си++ на бумаге никогда не было. Проект, документация и реализация двигались одновременно. В качестве базового языка для Си++ был выбран Си, потому что он: многоцелевой, лаконичный и относительно низкого уровня: отвечает большинству задач системного программирования: идет везде и на всем: пригоден в среде программирования *UNIX*. Различие между Си и Си++ состоит в степени внимания, уделяемого типам и структурам. Си выразителен и снисходителен. Си++ еще более выразителен, но чтобы достичь этой выразительности, программист должен уделить больше внимания типам объектов. Когда известны типы объектов, компилятор может правильно обрабатывать выражения, тогда как в противном случае программисту пришлось бы задавать действия с мучительными подробностями. Знание типов объектов также позволяет компилятору обнаруживать ошибки, которые в противном случае остались бы до тестирования. Си++ позволяет разумным образом структурировать большие программы таким образом, чтобы для одного человека не было непомерным справляться с программами в 25 000 строк. Существенным критерием при разработке языка была простота. Там, где возникал выбор между упрощением руководства

по языку и другой документации и упрощением компилятора, выбиралось первое. Огромное значение также предавалось совместимости с Си, это помешало удалить синтаксис Си. В Си++ нет типов данных высокого уровня и нет первичных операций высокого уровня. В нем нет, например, матричного типа с операцией обращения или типа строка с операцией конкатенации. Если пользователю понадобятся подобные типы, их можно определить в самом языке. Си++ проектировался для использования в довольно традиционной среде компиляции и выполнения, среде программирования на Си в системе UNIX. Средства обработки особых ситуаций и параллельного программирования, требующие нетривиальной загрузки и поддержки в процессе выполнения, не были включены в Си++. Вследствие этого реализация Си++ очень легко переносима.

***C#* (*C Sharp*)** – «Си Шарп»: объектно-ориентированный яык программирования, о разработке которого в 2000 объявила фирма Microsoft. По своему характеру он напоминает языки *C++* и *Java* и предназначен для разработчиков программ, использующих языки *C* и *C++* для того, чтобы они могли более эффективно создавать Интернет-приложения. Указывается, что *C#* будет тесно интегрирован с языком *XML*.

**Смолток**
Язык Смолток предназначен для решения нечисловых задач при построении систем искусственного интеллекта. В языке Форт применены структурное программирование и очень компактный машинный код. Отличительными особенностями проблемно (объектно)-ориентированных языков (Смолток, Форт, Модула и Ада) модульность построения процедур, абстракцию данных, динамическую связку программ (позволяет отказаться от перекомпилирования всей программы при внесении изменений в отдельные модули) и использование механизма наследования иерархического типа. К недостаткам относится некоторая замедленность выполнения программ из-за их динамической связи и сложность трансляторов.

**Форт [***FOURTH* – четвёртый]
стал применяться в задачах управления различными системами после того, как его автор Чарльз Мур (1970) написал на нём программу, предназначенную для управления радиотелескопом Аризонской обсерватории. Ряд свойств, а именно интерактивность, гибкость и простота разработки делают Форт весьма привлекательным и эффективным языком в прикладных исследованиях и при создании инструментальных средств. Очевидными областями применения этого языка являются встраиваемые системы управления. Также находит применение при программировании компьютеров под управлением различных операционных систем.

**Фортран** [*FORTRAN - FORmula TRANslation* –преобразование формул]
Язык программирования высокого уровня, компилятор. Разработан фирмой *IBM* (Д. Букс, 1956) для описания алгоритмов решения вычислительных задач. Относится к категории процедурно-ориентированных языков. Наиболее распространенными версиями этого языка являются Фортран IV, Фортран 77 и Фортран 90. Используется на всех классах ЭВМ. Последняя его версия также применяется на ЭВМ с параллельной архитектурой. Язык Фортран наиболее эффективен при численных расчетах, прост по структуре и удобен при выполнении программ. Несмотря на свои недостатки, этот язык получил большое распространение при разработке прикладных программ для решения научных задач. Среди причин долголетия Фортрана (а он один из самых распространенных языков в мире), можно отметить простую структуру, как самого Фортрана, так и предназначенных для него трансляторов. Программа на Фортране записывается в последовательности предложений или операторов, и оформляется по определенным стандартам. Эти стандарты накладывают ограничения, в частности, на форму записи и расположения частей оператора в строке бланка для записи операторов. Программа, записанная на Фортране, представляет собой один или несколько сегментов (подпрограмм) из операторов. Сегмент, управляющий работой всей программы в целом, называется основной программой. Фортран был задуман для использования в сфере научных и инженерно-технических вычислений. Однако на этом языке легко описываются задачи с разветвленной логикой (моделирование производственных процессов, решение игровых ситуаций и т.д.), некоторые экономические задачи и особенно задачи редактирования (составление таблиц, сводок, ведомостей и т.д.). Модификация языка Фортран, появившиеся в 1958 (Фортран II) содержала понятие подпрограммы и общих переменных для обеспечения связи между сегментами. К 1962 относится появление языка, известного под именем Фортран IV и ставшего наиболее употребительным в настоящее время. К этому же времени относится и начало деятельности комиссии при Американской

Ассоциации Стандартов (*ASA*), которая выработала к 1966 году два стандарта – языки Фортран и базисный (основной) Фортран (*Basic FORTRAN*). Фортран заложен в основу *Basic* – диалогового языка.

**Ява**
создан 1995 году в корпорации *Sun Microsystems* К. Арнольдом и Д. Гослингом. Он наследовал синтаксис *C* и *C++* и был избавлен от некоторых неприятных черт последнего. Отличительной особенностью языка является компиляция в код некоей абстрактной машины, для которой затем пишется эмулятор (*Java Virtual Machine*) для реальных систем. Кроме того, в *Java* нет указателей и множественного наследования, что сильно повышает надежность программирования.

***AppleScript***
Машинозависимый (ориентирован на работу с ПЭВМ типа Макинтош фиры *Apple*) близкий к естественному английскому язык программирования, предназначенный для автоматизации повторяющихся задач, преимущественно связанных с процессами компьютерной графики (в том числе - обработки результатов сканирования, ввода изображений, цветоделения, составления каталогов, передачи печатных документов в *World Wide Web* и др.).

**APL** (*Application Programming Language*).
язык для описания математической обработки данных (1957). Особенность - использование математических символов и очень мощный синтаксис, позволяющий производить множество нетривиальных операций прямо над сложными объектами, не прибегая к разбиению их на компоненты. Широкому применению помешало использование нестандартных символов как элементов синтаксиса.

**Clipper**
Язык высокого уровня и система программирования, предназначенные для разработки программ для ПЭВМ, преимущественно - систем управления большими объемами данных. Разработчик языка фирма *Nantucket* (США). Начало работ по их созданию связано с разработкой компилятора для *dBase* (1984, год основания фирмы *Nantucket* Б. Ребеллом и Б. Расселом). Первые программные продукты *Clipper* - *ClipperWinter'84* (1985), *ClipperWinter*'85 (1986), *McMax* (версия для ПК *Macintosh* - 1986) и *ClipperSummer*'87 (1987). В 1990 выпущена версия языка *Clipper* 5.0, получившая широкое распространение в России. Она реализует концепцию открытой архитектуры и представляет собой язык, компилятор и систему разработки программ, включающую набор команд и функций, препроцессор, компоновщик, набор утилит.

**dBASE**
Язык программирования высокого уровня, предназначенный для создания пакетов прикладных программ, связанных с манипулированием большими объемами данных (*Xbase*). Первая версия 1980, в 1994 выпущена версия *dBASE* 5.0 для *Windows.*

**FoxPro**
Объектно-ориентированный язык, предназначенный для создания пакетов прикладных программ, в том числе для современных операционных систем, например - версия этого языка *FoxPro for Windows.*

**SGML** [*Standardized General Markup Language* -Стандартизованный обобщенный язык разметки] Разработка языка была вызвана необходимостью создания средств описания документов и правил их построения. Для задания структуры документа используются специальные метки – «теги», которые отделяют друг от друга элементы документа и файлы определения типа документа (*Document Type Definition – DTD*), выполняющие функции грамматики и определяющие структуру и содержание каждого элемента в документе. Принят *ISO* в качестве стандарта в 80-е годы. Сложность этого языка помешала ему лечь в основу первой спецификации для *Web – HTML*, который стал производным от *SGML*.

**html, html** [*HyperText Markup Language* - Язык разметки гипертекста]
разработан в исследовательском центре *CERN* в 1992. Он является производным от *SGML. html* устанавливает формат гипермедийных документов, в сети *WWW*. *HTML*-документы представляют собой *ASCII* -файлы, доступные для редактирования в любом текстовом редакторе. Отличием от обычного

текстового файла является наличие в *HTML*-документах специальных команд - тэгов, которые указывают правила форматирования документа.

**PL/1**
В 1964 *IBM* создала язык *PL*/1, который был призван заменить *Cobol* и *Fortran* в большинстве приложений. Язык обладает исключительным богатством синтаксических конструкций. В нём впервые появилась обработка исключительных ситуаций и поддержка параллелизма. Надо заметить, что синтаксическая структура языка была крайне сложной. Пробелы уже использовались как синтаксические разделители, но ключевые слова не были зарезервированы. В частности, следующая строка - это вполне нормальный оператор на *PL*/1: *IF ELSE=THEN THEN THEN*; *ELSE ELSE* В силу таких особенностей разработка компилятора для *PL*/1 была исключительно сложным делом. Язык так и не стал популярен вне мира *IBM*. Как язык программирования *PL/M* значительно уступает конкурирующим с ним языкам Паскаль и Модула-2.

**Snobol и Icon**
В 1962 году появился язык *Snobol* (а в 1974 - его преемник Icon), предназначенный для обработки строк. Синтаксис *Icon* напоминает *C* и *Pascal* одновременно. Отличие заключается в наличии мощных встроенных функций работы со строками и связанная с этими функциями особая семантика. Современным аналогом *Icon* и *Snobol* является *Perl* - язык обработки строк и текстов, в который добавлены некоторые объектно-ориентированные возможности. Считается очень практичным языком, однако ему недостает элегантности.

**SETL**
В 1969 году был создан язык *SETL* - язык для описания операций над множествами. Основной структурой данных в языке является множество, а операции аналогичны математическим операциям над множествами. Полезен при написании программ, имеющих дело со сложными абстрактными объектами.

**Скриптовые языки**
В связи развитием Интернет-технологий и внедрением высокопроизводительных компьютеров получили распространение так называемые скриптовые языки. Они ориентировались на использование в качестве внутренних управляющих языков во всякого рода сложных системах. Затем они вышли за пределы сферы своего изначального применения и используются ныне в совсем иных областях. Характерными особенностями данных языков являются, во-первых, их интерпретируемость (компиляция либо невозможна, либо нежелательна), во-вторых, простота синтаксиса, а в-третьих, легкая расширяемость. Таким образом, они идеально подходят для использования в часто изменяемых программах, небольших программах или в случаях, когда для выполнения операторов языка затрачивается время, несопоставимое со временем их разбора.

**JavaScript**
Язык создан в компании *Netscape Communications* в качестве языка для описания сложного поведения веб-страниц. Интерпретируется браузером во время отображения веб-страницы. По синтаксису схож с *Java* и с *C/C++*. Имеет возможность использовать встроенную в браузер объектную функциональность, однако подлинно объектно-ориентированным языком не является.

**VBScript**
Язык создан *Microsoft* в качестве альтернативы *JavaScript*. Имеет схожую область применения. Синтаксически схож с языком *Visual Basic*. Так же, как и *JacaScript*, исполняется браузером при отображении веб-страниц и имеет ту же степень объектно- ориентированности.

**Perl**
Язык создавался в помощь системному администратору операционной системы *Unix* для обработки различного рода текстов и выделения нужной информации. Развился до мощного средства работы с текстами. Является интерпретируемым языком и реализован практически на всех существующих платформах. Применяется при обработке текстов, а также для динамической генерации веб-страниц на веб-серверах.

**Python**
Интерпретируемый объектно-ориентированный язык программирования. По структуре и области применения близок к *Perl,* однако менее распространен и более строг и логичен. Имеются реализации для большинства существующих платформ.

**Simula**
Первым объектно-ориентрованным языком был язык *Simula* (1967). Этот язык был предназначен для моделирования различных объектов и процессов, и объектно-ориентированные черты появились в нем именно для описания свойств модельных объектов.

**Smalltalk**
объектно-ориентированный язык создан в 1972. Язык предназначался для сложных графических проектов. В нём классы и объекты - единственные конструкции программирования. Большим недостатком *Smalltalk* являются большие требования к памяти и низкая производительность полученных программ. Это связано с не очень удачной реализацией объектно-ориентированных особенностей. Популярность языков *C++* и *Ada* 95 связана именно с тем, что объектно- ориентированность реализована без существенного снижения производительности.

**Eiffel**
Существует язык с очень хорошей реализацией объектно- ориентированности, не являющийся надстройкой ни над каким другим языком. Это язык *Eiffel* (1986). Являясь чистым языком объектно-ориентированного программирования, он, кроме того, повышает надежность программы путем использования «контрольных утверждений».
Большинство компьютерных архитектур и языков программирования ориентированы на последовательное выполнение операторов программы. В настоящее время, существуют программно-аппаратные комплексы, позволяющие организовать параллельное выполнение различных частей одного и того же вычислительного процесса. Для программирования таких систем необходима специальная поддержка со стороны средств программирования, в частности, языков программирования. Некоторые языки общего назначения содержат в себе элементы поддержки параллелизма, однако же программирование истинно параллельных систем требует подчас специальных приемов.

**Occam**
создан в 1982 и предназначен для программирования транспьютеров - многопроцессорных систем распределенной обработки данных. Он описывает взаимодействие параллельных процессов в виде каналов - способов передачи информации от одного процесса к другому. В языке последовательный и параллельный порядки выполнение операторов равноправны, и их необходимо явно указывать ключевыми словами *PAR* и *SEQ*.

**Linda**
В 1985 была предложена модель параллельных вычислений *Linda*. Основной её задачей является организация взаимодействия между параллельно выполняющимися процессами. Это достигается за счет использования глобальной кортежной области. Процесс может поместить туда кортеж с данными (то есть совокупность нескольких, возможно разнородных, данных), а другой процесс может ожидать появления в кортежной области некоторого кортежа и, после его появления, прочитать кортеж с возможным последующим его удалением. Заметим, что процесс может, например, поместить кортеж в область и завершиться, а другой процесс может через некоторое время воспользоваться этим кортежем. Так обеспечивается возможность асинхронного взаимодействия. Очевидно, что при помощи такой модели может быть сэмулировано и синхронное взаимодействие. *Linda* - модель параллельных вычислений, она может быть добавлена в любой язык программирования. Существуют достаточно эффективные реализации Linda, обходящие проблему существования глобальной кортежной области с потенциально неограниченным объемом памяти.
Все языки, о которых шла речь ранее, имеют одно общее свойство: они императивны. Это означает, что программы на них, в конечном итоге, представляют собой пошаговое описание решения той или иной задачи. Можно попытаться описывать лишь постановку проблемы, а решать задачу поручить компилятору. Существует два основных подхода, развивающие эту идею: функциональное и логическое программирование.
Основная идея, лежащая в основе функционального программирования, - представление программы в виде математических функций (т.е. функций, значение которых определяется лишь их аргументами, а не контекстом

выполнения). Оператор присваивания в таких языках не используется (или, как минимум, его использование не поощряется). Императивные возможности имеются, но их применение обставлено серьезными ограничениями. Существуют языки с ленивой и с энергичной семантикой. Различие заключается в том, что в языках с энергичной семантикой вычисления производятся в том же месте, где они описаны, а в случае ленивой семантики вычисление производится только тогда, когда оно действительно необходимо. Первые языки имеют более эффективную реализацию, в то время как вторые - лучшую семантику.

Из языков с энергичной семантикой упомянем *ML* и два его современных диалекта - *Standard ML (SML)* и *CaML*. Последний имеет объектно-ориентированного потомка - *Objective CaML (O'CaML)*. Среди языков с ленивой семантикой наиболее распространены два: *Haskell* и его более простой диалект *Clean*.

Программы на языках логического программирования выражены как формулы математической логики, а компилятор пытается получить следствия из них. Родоначальником большинства языков логического программирования является язык *Prolog* (1971). У него есть ряд потомков - *Parlog* (1983, ориентирован на параллельные вычисления), *Delta Prolog* и др.

## 5. ТРАНСЛЯТОРЫ

Можно писать программы непосредственно на машинном языке, хотя это и сложно. На заре компьютеризации (в начале 1950-х г.г.), машинный язык был единственным языком. Для спасения программистов от сурового машинного языка программирования, были созданы языки высокого уровня (т.е. немашинные языки), которые стали своеобразным связующим мостом между человеком и машинным языком компьютера. Языки высокого уровня работают через трансляционные программы, которые вводят "исходный код", и заставляет компьютер выполнять соответствующие команды, которые даются на машинном языке. Существует два основных вида трансляторов: интерпретаторы, которые сканируют и проверяют исходный код в один шаг, и компиляторы, которые сканируют исходный код для производства текста программы на машинном языке, которая затем выполняется отдельно.

Создать язык, удобный для написания программ, недостаточно. Для каждого языка нужен свой переводчик. Такими переводчиками являются специальные программы-трансляторы. Транслятор – это программа, предназначенная для перевода программы, написанной на одном языке программирования, в программу на другом языке программирования. Процесс перевода называется трансляцией. Тексты исходной и результирующей программ находятся в памяти компьютера.

Примером транслятора является компилятор. Компилятор – это программа, предназначенная для перевода программы, написанной на каком-либо языке, в программу в машинных кодах. Процесс такого перевода называется компиляцией. Компилятор создаёт законченный результат – программу в машинных кодах. Затем эта программа выполняется. Откомпилированный вариант исходной программы можно сохранить на диске. Для повторного выполнения исходной программы компилятор уже не нужен. Достаточно загрузить с диска в память компьютера откомпилированный в предыдущий раз вариант и выполнить его.

Существует другой способ сочетания процессов трансляции и выполнения программы. Он называется интерпретацией. Суть процесса интерпретации состоит в следующем. Вначале переводится в машинные коды, а затем выполняется первая строка программы. Когда выполнение первой строки окончено, начинается перевод второй строки, которая затем выполняется и так далее. Управляет этим процессом программа-интерпретатор. Интерпретатор – это программа, предназначенная для построчных трансляции и выполнения исходной программы. Такой процесс называется интерпретацией. В процесс трансляции входит проверка исходной программы на соответствие правилам используемого в ней языка. Если в программе обнаружены ошибки, транслятор вводит сообщение о них на устройство вывода (обычно, на экран дисплея). Интерпретатор сообщает о найденных им ошибках после трансляции каждой строки программы. Это значительно облегчает процесс поиска и исправления ошибок в программе, однако существенно увеличивает время трансляции. Компилятор транслирует программу намного быстрее, чем интерпретатор, но сообщает о найденных им ошибках после завершения компиляции всей программы. Найти и исправить ошибки в этом случае труднее. Поэтому интерпретаторы рассчитаны, в основном, на языки, предназначенные для обучения программированию, и используются начинающими программистами. Большинство современных языков предназначены для разработки сложных пакетов программ и рассчитаны на компиляцию.

## 5.1 Интерпретаторы

Одно, часто упоминаемое преимущество интерпретаторной реализации состоит в том, что она допускает «непосредственный режим». Непосредственный режим позволяет вам задавать компьютеру задачу вроде *PRINT* 3.14159*3/2.1 и возвращает вам ответ, как только вы нажмете клавишу *ENTER* (это позволяет использовать компьютер стоимостью 2000 долларов в качестве калькулятора стоимостью 10 долларов). Кроме того, интерпретаторы имеют специальные атрибуты, которые упрощают отладку. Можно, например, прервать обработку интерпретаторной программы, отобразить содержимое определенных переменных, бегло просмотреть программу, а затем продолжить исполнение. Больше всего программистам нравится в интерпретаторах возможность получения быстрого ответа. Здесь нет необходимости в компилировании, так как интерпретатор всегда готов для вмешательства в вашу программу. Введите *RUN* и результат вашего самого последнего изменения оказывается на экране.

Однако интерпретаторные языки имеют недостатки. Необходимо, например, иметь копию интерпретатора в памяти все время, тогда как многие возможности интерпретатора, а следовательно и его возможности могут не быть необходимыми для исполнения конкретной программы. Слабо различимым недостатком интерпретаторов является то, что они имеют тенденцию отбивать охоту к хорошему стилю программирования. Поскольку комментарии и другие формализуемые детали занимают значительное место программной памяти, люди стремятся ими не пользоваться. Дьявол менее яростен, чем программист, работающий на интерпретаторном Бейсике, пытающийся получить программу в 120К в памяти емкостью 60К. но хуже всего то, что интерпретаторы тихоходны. Ими затрачивается слишком много времени на разгадывание того, что делать, вместо того чтобы заниматься действительно делом.

При исполнении программных операторов, интерпретатор должен сначала сканировать каждый оператор с целью прочтения его содержимого (что этот человек просит меня сделать?), а затем выполнить запрошенную операцию. Операторы в циклах сканируются излишне много. Очевидно, что если вам удалось каким-то образом отделить фазу сканирования/понимания от фазы исполнения вы имели бы более быструю программу. И это как раз то, для чего существуют компиляторы.

## 5.2. Компиляторы

Компилятор-это транслятор текста на машинный язык, который считывает исходный текст. Он оценивает его в соответствии с синтаксической конструкцией языка и переводит на машинный язык. Другими словами, компилятор не исполняет программы, он их строит. Интерпретаторы невозможно отделить от программ, которые ими прогоняются, компиляторы делают свое дело и уходят со сцены. При работе с компилирующим языком, таким как Турбо-Бейсик, вы придете к необходимости мыслить о ваших программах в признаках двух главных фаз их жизни: периода компилирования и периода прогона. Большинство программ будут прогоняться в четыре - десять раз быстрее их интерпретаторных эквивалентов. Если вы поработаете над улучшением, то сможете достичь 100-кратного повышения быстродействия. Оборотная сторона монеты состоит в том, что программы, расходующие большую часть времени на возню с файлами на дисках или ожидание ввода, не смогут продемонстрировать какое-то впечатляющее увеличение скорости.

Иногда один и тот же язык может использовать и компилятор, и интерпретатор. К числу таких языков относится, например, Бейсик. Как правило, программы-компиляторы и интерпретаторы называются так же, как и языки, для перевода с которых они предназначены. Слова Паскаль, Ада, Си могут относиться как к названиям языков, так и к названиям соответствующих программ.

# 6. МАШИННО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ

Машинно – ориентированные языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно – ориентированные языки позволяют использовать все возможности и особенности Машинно – зависимых языков: высокое качество создаваемых программ (компактность и скорость выполнения); возможность использования конкретных аппаратных ресурсов; предсказуемость объектного кода и заказов памяти; для составления эффективных программ необходимо знать систему команд и особенности функционирования данной ЭВМ; трудоемкость процесса составления программ ( особенно на машинных языках и ЯСК), плохо защищенного от появления ошибок; низкая скорость программирования; невозможность непосредственного использования программ, составленных на этих языках, на ЭВМ других типов.

Машинно-ориентированные языки по степени автоматического программирования подразделяются на классы.

## 6.1 Машинный язык

Компьютер имеет свой определенный Машинный язык (далее МЯ), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому МЯ является командным. Однако, некоторые семейства ЭВМ (например, *ЕС ЭВМ, IBM*/370/ и др.) имеют единый МЯ для ЭВМ разной мощности. В команде любого из них сообщается информация о местонахождении операндов и типе выполняемой операции.

Языки Символического Кодирования (далее ЯСК), так же, как и МЯ, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в ЯСК заменены на символы (идентификаторы), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ. Использование символических адресов – первый шаг к созданию ЯСК. Команды ЭВМ вместо истинных (физических) адресов содержат символические адреса. По результатам составленной программы определяется требуемое количество ячеек для хранения исходных промежуточных и результирующих значений. Назначение адресов, выполняемое отдельно от составления программы в символических адресах, может проводиться менее квалифицированным программистом или специальной программой, что в значительной степени облегчает труд программиста.

Есть также языки, включающие в себя все возможности ЯСК, посредством расширенного введения макрокоманд - они называются Автокоды. В различных программах встречаются некоторые достаточно часто использующиеся командные последовательности, которые соответствуют определенным процедурам преобразования информации. Эффективная реализация таких процедур обеспечивается оформлением их в виде специальных макрокоманд и включением последних в язык программирования, доступный программисту. Макрокоманды переводятся в машинные команды двумя путями – расстановкой и генерированием. В постановочной системе содержатся «остовы» - серии команд, реализующих требуемую функцию, обозначенную макрокомандой. Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в «остов» программы, превращая её в реальную машинную программу. В системе с генерацией имеются специальные программы, анализирующие макрокоманду, которые определяют, какую функцию необходимо выполнить и формируют необходимую последовательность команд, реализующих данную функцию. Обе указанных системы используют трансляторы с ЯСК и набор макрокоманд, которые также являются операторами автокода. Развитые автокоды получили название Ассемблеры. Сервисные программы и пр. составлены на языках типа Ассемблер.

Язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ на более сжатую форму - называется Макрос (средство замены). В основном, Макрос предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется макропроцессором. На макропроцессор поступает макроопределяющий и исходный текст. Реакция макропроцессора на вызов-выдача выходного текста. Макрос одинаково может работать, как с программами, так и с данными.

## 6.2. Машинно – независимые языки

Машинно – независимые языки – средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и ВС. Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка(задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на МЯ. Т.о., командные последовательности (процедуры, подпрограммы), часто используемые в машинных программах, представлены в высокоуровневых языках отдельными операторами. Программист получил возможность не расписывать в деталях вычислительный процесс на уровне машинных команд, а сосредоточиться на основных особенностях алгоритма.

С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач, ими стали проблемно – ориентированные языки. Эти языки, ориентированные на решение определенных проблем, должны

обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме.

Проблемных языков очень много, например: Фортран, Алгол – языки, созданные для решения математических задач; Simula, Слэнг - для моделирования; Лисп, Снобол – для работы со списочными структурами.

Универсальные языки были созданы для широкого круга задач: коммерческих, научных, моделирования и т.д. Первый универсальный язык был разработан фирмой IBM, ставший в последовательности языков Пл/1. Второй по мощности универсальный язык называется Алгол-68. Он позволяет работать с символами, разрядами, числами с фиксированной и плавающей запятой. Пл/1 имеет развитую систему операторов для управления форматами, для работы с полями переменной длины, с данными организованными в сложные структуры, и для эффективного использования каналов связи. Язык учитывает включенные во многие машины возможности прерывания и имеет соответствующие операторы. Предусмотрена возможность параллельного выполнение участков программ. Программы в Пл/1 компилируются с помощью автоматических процедур. Язык использует многие свойства Фортрана, Алгола, Кобола. Однако он допускает не только динамическое, но и управляемое и статистическое распределения памяти.

Появление новых технических возможностей поставило задачу перед системными программистами – создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ их назвали диалоговыми языками. Эти работы велись в двух направлениях. Создавались специальные управляющие языки для обеспечения оперативного воздействия на прохождение задач, которые составлялись на любых раннее неразработанных (не диалоговых) языках. Разрабатывались также языки, которые кроме целей управления обеспечивали бы описание алгоритмов решения задач. Необходимость обеспечения оперативного взаимодействия с пользователем потребовала сохранения в памяти ЭВМ копии исходной программы даже после получения объектной программы в машинных кодах. При внесении изменений в программу с использованием диалогового языка система программирования с помощью специальных таблиц устанавливает взаимосвязь структур исходной и объектной программ. Это позволяет осуществить требуемые редакционные изменения в объектной программе.

Одним из примеров диалоговых языков является Бэйсик, который использует обозначения подобные обычным математическим выражениям. Многие операторы являются упрощенными вариантами операторов языка Фортран. Поэтому этот язык позволяет решать достаточно широкий круг задач.

Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по фиксированным алгоритмам (табличные языки и генераторы отчетов), и языков связи с операционными системами. Позволяя чётко описывать как задачу, так и необходимые для её решения действия, таблицы решений дают возможность в наглядной форме определить, какие условия должны быть выполнены прежде чем переходить к какому-либо действию. Одна таблица решений, описывающая некоторую ситуацию, содержит все возможные блок-схемы реализаций алгоритмов решения. Табличные методы легко осваиваются специалистами любых профессий. Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе.

При использовании САПР приходится не только решать задачи вычислительного характера и обработки данных, но и автоматизировать описание объектов, процессы ввода, вывода и редактирования данных, ввода графических изображений, схем, чертежей и т. п. Для этой цели служат языки проектирования. Классификация языков проектирования приведена на **Рис. 3.**

Языки проектирования делят на: входные, выходные, сопровождения, промежуточные и внутренние. Входные языки служат для задания исходной информации об объектах и целях проектирования. Во входных языках можно выделить две части: непроцедурную, служащую для описания структур объектов, и процедурную, предназначенную для описания заданий на выполнение проектных операций.


**Рис. 3.** Классификация языков проектирования

Языки сопровождения служат для непосредственного общения пользователя с ЭВМ и применяются для корректировки и редактирования данных при выполнении проектных процедур. В диалоговых режимах работы с ЭВМ средства языков входного, выходного и сопровождения тесно связаны и объединяются под названием диалогового языка. Современные диалоговые языки широко используют средства машинной графики (графический диалог). Диалог с ЭВМ может быть пассивным, когда инициатор диалога - система и от пользователя требуются только простые ответы, и активным при двусторонней инициативе диалога. Наиболее распространенная форма пассивного диалога - это система встроенных, в том числе иерархических, директивных меню. Недиалоговые системы языков сопровождения ориентированы на пакетный режим работы ЭВМ.

Промежуточные языки используются для описания информации в системах поэтапной трансляции исходных программ. Введение таких языков облегчает адаптацию программных комплексов САПР к новым входным языкам, т.е. делает комплекс открытым по отношению к новым составляющим лингвистического обеспечения.

Внутренние языки устанавливают единую форму представления данных (текстовой и графической информации) в памяти ЭВМ по подсистемам САПР. Принимаются определенные соглашения об интерфейсах отдельных программ, что делает САПР открытой по отношению к новым элементам программного обеспечения.

В качестве примера современного языка проектирования можно указать язык *VHDL* (*VHSIC - hardware description language*) - язык описания аппаратуры на базе сверхвысокоскоростных интегральных схем. Этот язык принят в качестве стандарта как инструментальное средство автоматизации проектирования СБИС, ориентированное на методологию нисходящего проектирования. Он является достаточно универсальным, чтобы охватить все аспекты проектирования изделий в области цифровой электроники.

До середины 60-х компьютеры были слишком дорогими машинами, использовавшимися только для особых задач, и выполнявшими только одну задачу за раз (т. н. пакетная обработка). Языки программирования этой эры, как и компьютеры на которых они использовались, были разработаны для специфичных задач, таких как научные вычисления. Поскольку машины были дорогими и лишь одна задача выполнялась за раз, то и машинное время было дорого – поэтому скорость выполнения программы стояла на первом месте. Однако в течение 60-х цена на компьютеры стала падать так, что даже небольшие компании могли их себе позволить; скорость компьютеров всё увеличивалась и наступило время, когда они стали часто простаивать без задач. Чтобы этого не происходило, стали вводить системы с разделением времени (*time-sharing*). В таких системах процессорное время «нарезалось», и все пользователи поочерёдно получали короткие отрезки этого времени. Машины были достаточно быстрыми для того, чтобы в результате каждый пользователь за терминалом чувствовал себя так, будто работает с системой в одиночку. Машина же, в свою очередь, простаивала меньше, поскольку выполняла не одну, а сразу много задач. Разделение времени радикально снижало стоимость машинного времени, поскольку одна машина могла совместно использоваться сотнями пользователей. В этих условиях - когда мощность стала дешева и доступна - создатели языков программирования все больше стали задумываться об удобстве написания программ, а не только скорости их выполнения. «Мелкие» (атомарные) операции, выполняемые

непосредственно устройствами машины, объединили в более «крупные», высокоуровневые операции и целые конструкции, с которыми человеку куда проще и удобнее работать.

В последние десятилетия в программировании возник и получил существенное развитие объектно-ориентированный подход. Это метод программирования, имитирующий реальную картину мира: информация, используемая для решения задачи, представляется в виде множества взаимодействующих объектов. Каждый из объектов имеет свои свойства и способы поведения. Взаимодействие объектов осуществляется при помощи передачи сообщений: каждый объект может получать сообщения от других объектов, запоминать информацию и обрабатывать её определённым способом и, в свою очередь, посылать сообщения. Так же, как и в реальном мире, объекты хранят свои свойства и поведение вместе, наследуя часть из них от родительских объектов. Объектно-ориентированная идеология используется во всех современных программных продуктах, включая операционные системы. Первый объектно-ориентированный язык *Simula*-67 был создан как средство моделирования работы различных приборов и механизмов. Большинство современных языков программирования – объектно-ориентированные. Среди них последние версии языка *Turbo-Pascal, C++*, *Ada* и другие. В настоящее время широко используются системы визуального программирования *Visual Basic, Visual C++*, *Delphi* и другие. Они позволяют создавать сложные прикладные пакеты, обладающие простым и удобным пользовательским интерфейсом.

В настоящее время языки развиваются в сторону все большей и большей абстракции. И это сопровождается падением эффективности. Повышение уровня абстракции влечет за собой повышение уровня надежности программирования. С низкой эффективностью можно бороться путем создания более быстрых компьютеров. Если требования к памяти слишком высоки, можно увеличить её объем. Это, конечно, требует времени и средств, но это решаемо. А вот с ошибками в программах можно бороться только одним способом: их надо исправлять. А еще лучше - не совершать. А еще лучше максимально затруднить их совершение. И именно на это направлены все исследования в области языков программирования. А с потерей эффективности придется смириться. Оптимисты ожидают, что скоро появятся языки, умеющие принимать, обрабатывать и передавать информации в виде мысли, слова, звука или жеста - языки программирования высочайшего уровня.

Мечтать не вредно!