Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»
(ВолгГТУ)
Кафедра иностранных языков

Семестровая работа

по английскому языку

Тема: «My Speciality. Program Languages»

Выполнил: студент группы
САПР-1.1п Чечеткин И. А.

Проверила:
Бойкова Т. А.

Краткая рецензия: _____

_____

_____

_____

_____

Оценка работы _____ баллов

Волгоград, 2016

Оглавление

ОРИГИНАЛ

1 DEFINITION

A programming language is a formal constructed language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

The earliest programming languages preceded the invention of the digital computer and were used to direct the behavior of machines such as Jacquard looms and player pianos. Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year. Many programming languages require computation to be specified in an imperative form, while other languages utilize other forms of program specification such as the declarative form.

The description of a programming language is usually split into the two components of syntax and semantics. Some languages are defined by a specification document, while other languages have a dominant implementation that is treated as a reference.

2 HISTORY

2.1 Early developments

The first programming languages designed to communicate instructions to a computer were written in the 1950s. An early high-level programming language to be designed for a computer was Plankalkül, developed for the German Z3 by Konrad Zuse between 1943 and 1945. However, it was not implemented until 1998 and 2000.

John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer. Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process much slower than running the equivalent machine code.

At the University of Manchester, Alick Glennie developed Autocode in the early 1950s. A programming language, it used a compiler to automatically convert the language into machine code. The first code and compiler was developed in 1952

for the Mark 1 computer at the University of Manchester and is considered to be the first compiled high-level programming language.

The second autocode was developed for the Mark 1 by R. A. Brooker in 1954 and was called the "Mark 1 Autocode". Brooker also developed an autocode for the Ferranti Mercury in the 1950s in conjunction with the University of Manchester. The version for the EDSAC 2 was devised by D. F. Hartley of University of Cambridge Mathematical Laboratory in 1961. Known as EDSAC 2 Autocode, it was a straight development from Mercury Autocode adapted for local circumstances, and was noted for its object code optimisation and source-language diagnostics which were advanced for the time. A contemporary but separate thread of development, Atlas Autocode was developed for the University of Manchester Atlas 1 machine.

Another early programming language was devised by Grace Hopper in the US, called FLOW-MATIC. It was developed for the UNIVAC I at Remington Rand during the period from 1955 until 1959. Hopper found that business data processing customers were uncomfortable with mathematical notation, and in early 1955, she and her team wrote a specification for an English programming language and implemented a prototype. The FLOW-MATIC compiler became publicly available in early 1958 and was substantially complete in 1959. Flow-Matic was a major influence in the design of COBOL, since only it and its direct descendent AIMACO were in actual use at the time. The language Fortran was developed at IBM in the mid '50s, and became the first widely used high-level general purpose programming language.

2.2 Refinement

The period from the 1960s to the late 1970s brought the development of the major language paradigms now in use:

- APL introduced array programming and influenced functional programming.
- ALGOL refined both structured procedural programming and the discipline of language specification.
- In the 1960s, Simula was the first language designed to support object-oriented programming; in the mid-1970s, Smalltalk followed with the first "purely" object-oriented language.

- C was developed between 1969 and 1973 as a system programming language, and remains popular.
- Prolog, designed in 1972, was the first logic programming language.
- In 1978, ML built a polymorphic type system on top of Lisp, pioneering statically typed functional programming languages.

The 1960s and 1970s also saw considerable debate over the merits of structured programming, and whether programming languages should be designed to support it.

2.3 Consolidation and growth

The 1980s were years of relative consolidation. C++ combined object-oriented and systems programming. The United States government standardized Ada, a systems programming language derived from Pascal and intended for use by defense contractors. In Japan and elsewhere, vast sums were spent investigating so-called "fifth generation" languages that incorporated logic programming constructs. The functional languages community moved to standardize ML and Lisp. Rather than inventing new paradigms, all of these movements elaborated upon the ideas invented in the previous decade.

The rapid growth of the Internet in the mid-1990s created opportunities for new languages. Perl, originally a Unix scripting tool first released in 1987, became common in dynamic websites. Java came to be used for server-side programming, and bytecode virtual machines became popular again in commercial settings with their promise of "Write once, run anywhere". These developments were not fundamentally novel, rather they were refinements to existing languages and paradigms, and largely based on the C family of programming languages.

Programming language evolution continues, in both industry and research.

3 ELEMENTS

All programming languages have some primitive building blocks for the description of data and the processes or transformations applied to them. These primitives are defined by syntactic and semantic rules which describe their structure and meaning respectively.

3.1 Syntax

A programming language's surface form is known as its syntax. Most programming languages are purely textual; they use sequences of text including words, numbers, and punctuation, much like written natural languages. On the other hand, there are some programming languages which are more graphical in nature, using visual relationships between symbols to specify a program.

The syntax of a language describes the possible combinations of symbols that form a syntactically correct program. The meaning given to a combination of symbols is handled by semantics.

3.2 Semantics

3.2.1 Static semantics

The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms. For compiled languages, static semantics essentially include those semantic rules that can be checked at compile time. Examples include checking that every identifier is declared before it is used or that the labels on the arms of a case statement are distinct. Many important restrictions of this type, like checking that identifiers are used in the appropriate context, or that subroutine calls have the appropriate number and type of arguments, can be enforced by defining them as rules in a logic called a type system. Other forms of static analyses like data flow analysis may also be part of static semantics. Newer programming languages like Java and C# have definite assignment analysis, a form of data flow analysis, as part of their static semantics.

3.2.2 Dynamic semantics

Once data has been specified, the machine must be instructed to perform operations on the data. For example, the semantics may define the strategy by which expressions are evaluated to values, or the manner in which control structures conditionally execute statements. The dynamic semantics of a language defines how and when the various constructs of a language should produce a program behavior. There are many ways of defining execution semantics. Natural language is often used to specify the execution semantics of languages commonly used in

practice. A significant amount of academic research went into formal semantics of programming languages, which allow execution semantics to be specified in a formal manner. Results from this field of research have seen limited application to programming language design and implementation outside academia.

3.2.3 Type system

A type system defines how a programming language classifies values and expressions into types, how it can manipulate those types and how they interact. The goal of a type system is to verify and usually enforce a certain level of correctness in programs written in that language by detecting certain incorrect operations. Any decidable type system involves a trade-off: while it rejects many incorrect programs, it can also prohibit some correct, albeit unusual programs. In order to bypass this downside, a number of languages have type loopholes, usually unchecked casts that may be used by the programmer to explicitly allow a normally disallowed operation between different types. In most typed languages, the type system is used only to type check programs, but a number of languages, usually functional ones, infer types, relieving the programmer from the need to write type annotations. The formal design and study of type systems is known as type theory.

4 DESIGN AND IMPLEMENTATION

Programming languages share properties with natural languages related to their purpose as vehicles for communication, having a syntactic form separate from its semantics, and showing language families of related languages branching one from another. But as artificial constructs, they also differ in fundamental ways from languages that have evolved through usage. A significant difference is that a programming language can be fully described and studied in its entirety, since it has a precise and finite definition. By contrast, natural languages have changing meanings given by their users in different communities. While constructed languages are also artificial languages designed from the ground up with a specific purpose, they lack the precise and complete semantic definition that a programming language has.

Many programming languages have been designed from scratch, altered to meet new needs, and combined with other languages. Many have eventually fallen into disuse. Although there have been attempts to design one “universal” programming language that serves all purposes, all of them have failed to be generally accepted

as filling this role. The need for diverse programming languages arises from the diversity of contexts in which languages are used:

- Programs range from tiny scripts written by individual hobbyists to huge systems written by hundreds of programmers.
- Programmers range in expertise from novices who need simplicity above all else, to experts who may be comfortable with considerable complexity.
- Programs must balance speed, size, and simplicity on systems ranging from microcontrollers to supercomputers.
- Programs may be written once and not change for generations, or they may undergo continual modification.
- Programmers may simply differ in their tastes: they may be accustomed to discussing problems and expressing them in a particular language.

One common trend in the development of programming languages has been to add more ability to solve problems using a higher level of abstraction. The earliest programming languages were tied very closely to the underlying hardware of the computer. As new programming languages have developed, features have been added that let programmers express ideas that are more remote from simple translation into underlying hardware instructions. Because programmers are less tied to the complexity of the computer, their programs can do more computing with less effort from the programmer. This lets them write more functionality per time unit.

A language's designers and users must construct a number of artifacts that govern and enable the practice of programming. The most important of these artifacts are the language specification and implementation.

4.1 Specification

The specification of a programming language is an artifact that the language users and the implementors can use to agree upon whether a piece of source code is a valid program in that language, and if so what its behavior shall be.

A programming language specification can take several forms, including the following:

- An explicit definition of the syntax, static semantics, and execution semantics of the language. While syntax is commonly specified using a formal grammar, semantic definitions may be written in natural language, or a formal semantics.
- A description of the behavior of a translator for the language. The syntax and semantics of the language have to be inferred from this description, which may be written in natural or a formal language.
- A reference or model implementation, sometimes written in the language being specified. The syntax and semantics of the language are explicit in the behavior of the reference implementation.

4.2 Implementation

An implementation of a programming language provides a way to write programs in that language and execute them on one or more configurations of hardware and software. There are, broadly, two approaches to programming language implementation: compilation and interpretation. It is generally possible to implement a language using either technique.

The output of a compiler may be executed by hardware or a program called an interpreter. In some implementations that make use of the interpreter approach there is no distinct boundary between compiling and interpreting. For instance, some implementations of BASIC compile and then execute the source a line at a time.

Programs that are executed directly on the hardware usually run several orders of magnitude faster than those that are interpreted in software.

One technique for improving the performance of interpreted programs is just-in-time compilation. Here the virtual machine, just before execution, translates the blocks of bytecode which are going to be used to machine code, for direct execution on the hardware.

5 USAGE

Thousands of different programming languages have been created, mainly in the computing field.

Programming languages differ from most other forms of human expression in that they require a greater degree of precision and completeness. When using a natural language to communicate with other people, human authors and speakers can

be ambiguous and make small errors, and still expect their intent to be understood. However, figuratively speaking, computers "do exactly what they are told to do", and cannot "understand" what code the programmer intended to write. The combination of the language definition, a program, and the program's inputs must fully specify the external behavior that occurs when the program is executed, within the domain of control of that program. On the other hand, ideas about an algorithm can be communicated to humans without the precision required for execution by using pseudocode, which interleaves natural language with code written in a programming language.

A programming language provides a structured mechanism for defining pieces of data, and the operations or transformations that may be carried out automatically on that data. A programmer uses the abstractions present in the language to represent the concepts involved in a computation. These concepts are represented as a collection of the simplest elements available. Programming is the process by which programmers combine these primitives to compose new programs, or adapt existing ones to new uses or a changing environment.

Programs for a computer might be executed in a batch process without human interaction, or a user might type commands in an interactive session of an interpreter. In this case the "commands" are simply programs, whose execution is chained together. When a language can run its commands through an interpreter, without compiling, it is called a scripting language.

In 2013 the ten most popular programming languages were: C, Java, PHP, JavaScript, C++, Python, Shell, Ruby, and Objective-C and C#.

ПЕРЕВОД

1 ОПРЕДЕЛЕНИЕ

Язык программирования (ЯП) – это формально построенный язык, предназначенный для передачи команд машине, в частности, компьютеру. Языки программирования могут быть использованы для создания программ, контролирующих поведение машины, или для выражения алгоритмов.

Первые языки программирования предшествовали изобретению цифровых машин и использовались, чтобы направить поведение различных машин, таких как жаккардовских ткацких станков и фортепиано. Тысячи различных языков программирования были созданы, в основном, в компьютерной области, и по сей день создаются многие языки программирования. Многие ЯП требуют, чтобы вычисление было указано в необходимом формате, в то время как другие языки используют другие формы спецификации программы, например, декларативная форма.

Описание языка программирования, как правило, делится на две составляющие: синтаксис и семантика. Некоторые языки имеют спецификационные документы, в то время как другие языки имеют доминирующую реализацию, которая рассматривается в качестве ссылки.

2 ИСТОРИЯ

2.1 Ранние разработки

Первые языки программирования, предназначенные для обращения к компьютеру, были написаны в 1950-е годы. Первый язык программирования высокого уровня для рассчетов на компьютере был назван Планкалкюль, разработанный для немецкого Z3 Конрадом Цузе между 1943 и 1945, однако, не был реализован до 1998 года и 2000 года.

Short Code Джона Мочли, предложенный в 1949 году, был одним из первых языков программирования высокого уровня, когда-либо разработанным для ЭВМ. В отличие от машинного кода, выражения на Short Code изображали математические выражения в понятной форме. Тем не менее, программа должна была быть переведена на машинный код каждый раз при запуске, что делает процесс гораздо более медленным, чем запуск эквивалентного машинного кода.

В начале 1950-х годов в Манчестерском университете Алик Гленни разработал AutoCode – язык программирования, использовавший компилятор для автоматического преобразования языка в машинный код. Первый код на этом языке и компилятор были разработаны в 1952 году для компьютера Марк 1 в Университете Манчестера и считается первым компилирующимся языком программирования высокого уровня.

Второй AutoCode был разработан для компьютера Марк 1 Р. А. Брукером в 1954 году и назывался "Марк-1 AutoCode". Брукер также разработал AutoCode для Ferranti Mercury в 1950-х годах совместно с Университетом Манчестера. Версия для EDSAC 2 была разработан Д. Ф. Хартли из Кембриджского университета математических лабораторий в 1961 году. Он стал известен как EDSAC 2 AutoCode, это был прямой наследник Mercury AutoCode, приспособленный для местных нужд, однако его оптимизация объектного кода и исходного языка диагностики были продвинуты для того времени. Современный, но отдельный поток развития, Atlas AutoCode был разработан в Университете Манчестера для машины Атлас 1.

Еще одним из первых языков программирования был FLOW-MATIC, разработанный Грейс Хоппер в США. Он был разработан для UNIVAC I в Remington Rand в период с 1955 до 1959 года. Хоппер обнаружила, что обработка бизнес-данных клиентов неудобна с математической нотации, а в начале 1955 года, она и ее команда написала спецификацию для языка программирования на английском и реализовала прототип. Компилятор FLOW-MATIC стал публично доступен в начале 1958 года и был фактически закончен в 1959 году. FLOW-Matic оказал большое влияние на разработку языка COBOL, так как только он и его прямой потомок AIMACO были в фактическом использовании в то время. Язык Фортран был разработан в середине 50-х годов компанией IBM. Он стал первым широко используемым языком высокого уровня общего назначения.

2.2 Усовершенствование

В период с 1960-х до конца 1970-х годов были разработаны основные языковые парадигмы, используемые до сих пор:

- APL представила программирование с использованием массивов и зачатки функционального программирования;

- язык ALGOL включил в себя и структурированное процедурное программирование, и дисциплину спецификации языка программирования;
- в 1960-х годах был разработан язык Simula, спроектированный для поддержки объектно-ориентированного программирования; в 1970-х годах был спроектирован «чисто» объектно-ориентированный язык Smalltalk;
- между 1969 и 1973 годами был разработан язык C, остающийся популярным и в наше время;
- язык Prolog, разработанный в 1972 году, стал первым языком логического программирования;
- в 1978 году язык ML обладал полиморфической системой типизации на основе Lisp, став первооткрывателем области статически типизированных функциональных языков программирования;

В 1960-е и 1970-е годы также шли значительные споры по поводу достоинств структурного программирования, и должны ли разрабатываться языки программирования для его поддержки.

2.3 Консолидация и рост

1980-е годы были годами относительной консолидации. C++ включил в себя ООП и системное программирование. Правительство Соединенных Штатов стандартизировало язык Ада, язык программирования систем, полученный из Паскаля, и предназначенный для использования у оборонных подрядчиков. В Японии и в других местах, огромные суммы были потрачены на исследование так называемых языков «пятого поколения», которые включали логические программные конструкции. Община функциональных языков стала стандартизировать ML и Lisp. Вместо изобретения новых парадигм, все эти шаги разработаны на идеях из предыдущего десятилетия.

Стремительный рост интернета в середине 1990-х годов открыл возможности для новых ЯП. Perl, изначально используемый для создания сценариев в Unix, стал широко использоваться для создания динамических веб-сайтов. Java стал использоваться для серверного программирования, и его байт-кодовые виртуальные машины стали весьма популярны в коммерческой среде с их лозунгом: «написано однажды, работает везде». Эти события не были принципи-

ально новыми, они лишь стали уточнением существующих языков и парадигм, и в основном базируются на семье языков C.

Эволюция языков программирования продолжила и продолжает свой рост и в промышленности, и в научных исследованиях.

3 ЭЛЕМЕНТЫ

Все ЯП имеют некоторые примитивные строительные элементы, используемые для описания данных и процессов или преобразований над ними. Эти элементы определяются синтаксическими и семантическими правилами, которые описывают их структуру и значение соответственно.

3.1 Синтаксис

«Поверхность» языка программирования известна как синтаксис. Большинство языков программирования являются чисто текстовыми: они используют последовательности блоков текста, в том числе слов, цифр и знаков препинания. Также существуют и письменные естественные языки и графические по природе, использующие визуальные отношения между символами для указания программы.

Синтаксис языка описывает возможные комбинации символов, которые образуют синтаксически правильную программу. Значение, придаваемое комбинациям символов обрабатывается правилами семантики.

3.2 Семантика

3.2.1 Статическая семантика

Статическая семантика определяет ограничения на структуру действительных текстов, которые трудно или невозможно выразить в стандартных синтаксических формализмах. Для компилируемых языков, статическая семантика включает, в основном, те семантические правила, которые могут быть проверены во время компиляции. Примерами могут являться проверка на объявление указателей до их использования, или что подписи на ветках развилки различны. Многие важные ограничения этого типа, как проверка, что указатели используются в соответствующем контексте, или что подпрограммы вызываются с соответствующими количеством и типами аргументов, могут

быть обеспечены путем определения их как правила в логике, называемой системой типов. Другие формы статических анализов, например, анализ потока данных, может также быть частью статической семантики. Новые языки программирования, такие как Java и C# имеют определенный анализ присвоения и формы анализа потока данных, как часть их статической семантики.

3.2.2 Динамическая семантика

После того, как данные были указаны, машине должно быть поручено выполнять операции над данными. Например, семантика может определить стратегию, с помощью которой вычисляются выражения для значений, или манеру, в которой управляющие структуры условно выполняют инструкции. Динамическая семантика языка определяет, как и когда различные конструкции языка должны изменять поведение программы. Есть много способов определения семантики исполнения. Естественный язык часто используется, чтобы задать выполнение семантики языков, широко используемых в практике. Значительное количество научных исследований вошли в формальную семантику языков программирования, позволяющую семантике исполнения быть указанной в формальной манере. Результаты исследований этой области имели ограниченное применение в конструкции языков программирования и в реализациях за пределами научных кругов.

3.2.3 Система типизации

Система типизации определяет, как язык программирования классифицирует значения и выражения по типам, как он может манипулировать этими типами, и как они взаимодействуют друг с другом. Целью системы типизации является проверка и, как правило, соблюдение определенного уровня правильности в программах, написанных на этом языке, обнаружение некоторыж неправильных операций. Любой разрешенный тип системы включает в себя компромисс: в то время как он отвергает много неверных программ, он также может запретить некоторые правильные, хотя и необычные, программы. Для того, чтобы обойти этот недостаток, в числе языков есть типовые лазейки, как правило, непроверяемые слепки, которые могут быть использованы программистом для явного разрешения нормально запрещенной операцию

между различными типами. В большинстве типизированных языков, система типизации используется только для проверки программ, но есть несколько языков, обычно функциональных, выводящих типы, освобождая программиста от необходимости писать аннотации типов. Официальная разработка и исследование систем типизации известны как теория типизации.

4 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

Языки программирования разделяют некоторые свойства с естественными языками, связанные с их назначением в качестве транспорта для общения, имея синтаксическую форму, отделенную от его семантики, и показывая языковые семьи родственных языков, ветвящихся друг от друга. Но, как искусственные конструкции, они также отличаются коренным образом от языков, которые развились через использование. Значительная разница в том, что язык программирования может быть полностью описан и изучен в полном объеме, так как он имеет точное и конечное определение. Напротив, естественные языки изменяют значения, данные им своими пользователями в различных сообществах. Построенные языки являются также искусственными языками, разработанными с нуля с определенной целью, они имеют точное и полное семантическое определение, что и язык программирования.

Многие языки программирования были разработаны с нуля, и изменены, чтобы удовлетворить новые потребности, а также комбинируются с другими языками. Многие из них, в конце концов, выходят из употребления. Хотя были попытки разработать единый «универсальный» язык программирования, который бы служил всем целям, все они не смогли быть общепринятыми, как заполняющие эту роль. Необходимость в различных языках программирования возникает из разнообразия условий, в которых они используются:

- программы варьируются от крошечных скриптов, написанных любителями, до огромных систем, над которыми работают сотни программистов;
- различная подготовка программистов: есть новички, для которых простота – это главное, а есть эксперты, которые могут комфортно работать с ЯП значительной сложности;
- программы должны балансировать скорость, размер и простоту на системах, начиная от микроконтроллеров и заканчивая суперкомпьютерами;
- программа может не изменятся на протяжении десятков лет, а может находится в непрерывном изменении;

- программисты могут просто отличатся в своих вкусах: они могут быть приучены к обсуждению проблем и их выражения на конкретном ЯП.

Одна общая тенденция в развитии языков программирования состоит в том, чтобы добавить больше возможностей для решения проблем, используя более высокий уровень абстракции. Первые языки программирования были очень тесно связаны с аппаратной частью компьютера. При разработке новых языков программирования были добавлены функции, которые позволяют программистам выражать идеи, являющиеся более удаленными от простого перевода в основные инструкции аппаратного обеспечения. Из-за этого программисты меньше связаны со сложностями на компьютере, их программы могут сделать больше вычислений с меньшим усилием от программиста. Это позволяет им писать больше функциональности в единицу времени.

Разработчики и пользователи языка могут построить ряд артифактов, которые регулируют и позволяют практику программирования. Наиболее важными из них являются спецификация и реализация языка.

4.1 Спецификация

Спецификация языка программирования является артефактом, который пользователи языка и разработчики могут использовать, чтобы согласовать, является ли кусок исходного кода действующей программой на этом языке, и, если да, то каким его поведение должно быть.

Спецификация языка программирования может принимать различные формы, в том числе следующие:

- четкое определение синтаксиса, статической семантики, и исполнительной семантики языка. В то время как синтаксис обычно определяются с помощью формальной грамматики, семантические определения могут быть написаны на естественном языке или с использованием формальной семантикм;

- описание поведения переводчика для языка. Синтаксис и семантика языка должны быть выведены из этого описания, которое может быть записано на естественном или формальном языке;

- ссылка или модель реализации, иногда написанные на уточняемом языке. Синтаксис и семантика языка являются четкими в поведении данной эталонной реализации.

4.2 Реализация

Реализация языка программирования дает возможность писать программы на этом языке, а также их выполнение на одной или нескольких конфигурациях аппаратного и программного обеспечения. Есть, в целом, два подхода к реализации языка программирования: компиляция и интерпретация. Это, как правило, можно реализовать, используя язык либо технику.

Выход компилятора может быть выполнен с помощью аппаратных средств или же программы под названием интерпретатора. В некоторых реализациях, которые используют подход интерпретатора, нет четкой границы между компиляцией и интерпретацией. Например, в некоторых реализациях программы BASIC компилируются, а затем выполняется по строчке из источника.

Программы, которые выполняются непосредственно на аппаратном уровне, как правило, работают на несколько порядков быстрее, чем те, которые интерпретируются в программном обеспечении.

Один из способов повышения производительности интерпретируемых программ является just-in-time компиляция. Виртуальная машина, как раз перед исполнением, переводит блоки байт-кода, которые будут использоваться для машинного кода, для непосредственного исполнения на аппаратном уровне.

5 ИСПОЛЬЗОВАНИЕ

Тысячи различных языков программирования были созданы, главным образом, в области вычислений.

Языки программирования отличаются от большинства других форм человеческого самовыражения тем, что они требуют большей степени точности и полноты. При использовании естественного языка для общения с другими людьми авторы и ораторы могут быть неоднозначными и делать небольшие ошибки, и продолжают ожидать, то что поймут их намерения. Тем не менее, образно говоря, компьютеры «делать то, что им сказали сделать», и не могут «понять», какой код программист собирался написать. Сочетание определения языка, программы и входов программы должны в полной мере указать внешнее поведение программы, что происходит, когда программа выполняется, в области контроля этой программы. С другой стороны, идеи об алгоритме может быть доведена до людей без точности, требуемой для выполнения с помощью псевдокода, который перемежает естественный язык с кодом, написанного на языке программирования.

Язык программирования обеспечивает структурированный механизм для определения частей данных, а также операции или преобразования, которые могут быть осуществлены автоматически над этими данными. Программист использует абстракции, присутствующие в языке для представления концепций в процессе вычисления. Эти концепции представлены в коллекции простейших доступных элементов. Программирование – это процесс, посредством которого программисты объединяют эти примитивы, создавая новые программы, или адаптируя существующие в новых областях применения или в изменяющейся окружающей среде.

Программы для компьютера могут быть выполнены в периодическом процессе без вмешательства человека, или пользователь может вводить команды в интерактивной сессии интерпретатора. В этом случае «команды» являются просто программами, исполнение которых соединено друг с другом. Когда язык может запускать свои команды через интерпретатор без компиляции, то такой ЯП называют скриптовым языком.

В 2013 году десятка самых популярных языков программирования состояла из: C, Java, PHP, JavaScript, C++, Python, Shell, Ruby, Objective-C и C#.

СЛОВА

Abstraction	Абстракция
Albeit	Тем не менее
Algorithm	Алгоритм
Analysis	Анализ
Argument	Аргумент
Artificial	Искусственный
Assignment	Присвоение
Attempt	Попытка
Behavior	Поведение
Boundary	Граничный
Broadly	Широко
Bytecode	Байт-код
Case	Случай
Chain	Цепочка
Combined	Комбинированный
Command	Команда
Communicate	Связываться
Compiler	Компилятор
Completeness	Полноценность
Consolidation	Консолидация
Contractor	Подрядчик
Customer	Закащчик
Decade	Десятилетие
Defense	Оборона
Definition	Определение
Descendent	Убывание
Description	Описание
Design	Разработка
Development	Разработка
Direction	Направление
Disuse	Неиспользование
Dominant	Доминирование

Elaborate	Разработка
Element	Элемент
Evolve	Развивать
Execute	Выполнять
Expression	Выражение
Follow	Следовать
Generation	Поколение
Hardware	Оборудование
Implementation	Реализация
Include	Включать в себя
Industry	Промышленность
Influence	Влияние
Interactive	Интерактивный
Interpreter	Интерпретатор
Introduction	Введение
Invent	Изобретать
Just-in-time	Точно в срок
Label	Этикетка
Language	Язык
Loom	Ткацкий станок
Loophole	Лазейка
Magnitude	Величина
Major	Основной
Merit	Заслуги
Notation	Обозначение
Novel	Новый
Occure	Появляться
Operation	Операция
Oriented	Ориентированный
Paradigm	Парадигма
Popular	Популярный
Precision	Точность
Primitive	Примитив
Processed	Обработанный

Prohibit	Запретить
Promise	Обещание
Prototype	Прототип
Pseudocode	Псевдокод
Purely	Чисто
Rapid	Быстрый
Refinements	Усовершенствования
Relationship	Отношение
Represent	Представлять
Require	Требовать
Research	Исследование
Script	Скрипт
Semantics	Семантика
Server-side	Серверный
Simple	Простой
Software	Программное обеспечение
Speaker	Оратор
Specification	Спецификация
Standardize	Стандартизация
Statement	Заявление
Support	Поддержка
Surface	Поверхность
Symbol	Символ
Syntax	Синтаксис
Transformation	Изменение
Translator	Переводчик
Treat	Относиться
Type	Тип
Understandable	Понятный
Utilize	Утилизация
Vast	Подавляющий
Vehicle	Транспорт
Virtual	Виртуальный
Visual	Визуальный