

Liceo Compu-Market

Cátedra: Practica Supervisada

Catedrático: Erick Gonzáles

Pre-Laboratorio “La Programación”

Nombre: Pablo Gioanni Camey Medrano

Grado: 5.to Bachillerato en Computación con Oriert. Cient.

Clave “5”

Fecha de Entrega: 20/04/2017

Índice

|  |  |  |
| --- | --- | --- |
|  |  |  |
| No. | Descripción | Pág.# |
|  | Carátula…………………………………………………………………………………… | [1](#_top) |
|  | Índice……………………………………………………………………………………… | [2-3](#indice) |
|  | Introducción……………………………………………………………………………… | [4](#introduccion) |
| 1. | La Programación……………………………………………………………………… | [5](#PROGRAMACION) |
| 2. | ¿Qué es programación? …………………………………………………………… | [6-9](#queesprogra) |
| 3. | Historia de la programación……………………………………………………… | [9-10](#historiaprogra) |
| 4. | ¿Qué tipo de lenguaje utiliza una computadora? ……………………… | [10](#quelenguajeutilizalapc) |
| 5. | Léxico……………………………………………………………………………………… | [10](#lexico) |
| 6. | Algoritmo………………………………………………………………………………… | [10](#algoritmo) |
| 7. | Definición……………………………………………………………………………… | [11](#defalgo) |
| 8. | Medios de expresión……………………………………………………………… | [12](#medioex) |
| 9. | Diagrama de flujo……………………………………………………………………… | [12](#diagramaflu) |
| 10. | Pseudocódigo……………………………………………………………………………… | [13](#pseudocodigo) |
| 11. | Sistema Formal ………………………………………………………………… | [13](#sistemaformal) |
| 12. | Variables……………………………………………………………………………… | [14](#variable) |
| 13. | Tipos de datos……………………………………………………………………… | [15](#tiposdedatos) |
| 14. | Estructuras secuenciales……………………………………………………… | [15](#estructurasecuenciales) |
| 15. | Compilación……………………………………………………………………………… | [16](#compilacion) |
| 16. | Los objetivos de la programación……………………………………………… | [16-17](#objetivosdprogra) |
| 17. | Los tipos de lenguaje en programación……………………………………… | [18](#tiposdlenguaje) |
| 18. | Lenguaje de bajo nivel……………………………………………………………… | [18-19](#lenguajebajo) |
| 19. | Lenguaje intermedio……………………………………………………………… | [19-20](#lenguajeinter) |
| 20. | Lenguaje de alto nivel……………………………………………………………… | [21](#lenguajealto) |
| 21. | Lenguaje interpretado……………………………………………………………… | [21-24](#lenguajeinter) |
| 22. | Lenguaje compilado……………………………………………………………… | [24-25](#lenguajecompi) |
| 23. | Lenguaje de programación declarativo……………………………… | [25-26](#lenguajedprogradecla) |
| 24. | Programación lógica……………………………………………………………… | [27-30](#programacionlogica) |
| 25. | Programación funcional……………………………………………………… | [30-31](#programaciofuncional) |
| 26. | Programación en bases de datos……………………………………………… | [31-32](#progrmaciorientadabasededatos) |
| 27. | Modelos de bases de datos……………………………………………… | [32-36](#modelosdebasesdatos) |
| 28. | Lenguajes de programación imperativos……………………………… | [37](#lenguajesdeprograimperativos) |
| 29. | Lenguajes imperativos procedurales……………………………………… | [38](#lenguajesdeprograiprocedurales) |
| 30. | Diferencia entre lenguajes declarativos e imperativos……………… | [39](#difentrelenguajedeclaeimpera) |
| 31. | Lenguaje de programación orientada a objetos……………………… | [39-43](#progrmaciorientadabasededatos) |
| 32. | Ejemplos de lenguajes……………………………………………………………… | [44](#ejemplosdelenguajes) |
| 33. | Lenguajes de alto nivel……………………………………………………………… | [44](#ejemplosdelenguajes) |
| 34. | Visual Basic………………………………………………………………………………… | [44-47](#visualbasic) |
| 35. | Ada………………………………………………………………………………………. | [47-50](#ada) |
| 36. | Algol……………………………………………………………………………………… | [50-51](#algol) |
| 37. | Basic……………………………………………………………………………………… | [52-64](#basic) |
| 38. | C#………………………………………………………………………………………….. | [64-66](#cC) |
| 39. | Fortran……………………………………………………………………………………. | [66-69](#fortran) |
| 40. | Java………………………………………………………………………………………… | [70-72](#java) |
| 41. | Lips………………………………………………………………………………………….. | [72-80](#lips) |
| 42. | Modula 2…………………………………………………………………………………. | [81](#modula2) |
| 44. | Pascal………………………………………………………………………………………. | [81-85](#pascal) |
| 45. | PERL…………………………………………………………………………………………. | [86-93](#perl) |
| 46. | PHP………………………………………………………………………………………….. | [93-95](#perl) |
| 47. | PL/SQL…………………………………………………………………………………….. | [96](#plsql) |
| 48. | PYTHON…………………………………………………………………………………… | [97-99](#pyton) |
| 59. | Conclusiones Personales…………………………………………………………… | [100](#conclusion) |

Introducción

La programación es una base de algoritmos para la creación de nuevas funciones y programas. Así fue el inicio de este tema que ha hecho evoluciones en el mundo de la tecnología a métodos de codificaciones para innovar e inventar nuevas modalidades hacia mejorar y facilitar a quién le tenga interés al desglose de programas.

La Programación

¿Qué es programación?

La programación informática o programación algorítmica, acortada como programación, es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas de computadora. El código fuente es escrito en un lenguaje de programación. El propósito de la programación es crear programas que exhiban un comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas distintas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal. Programar no involucra necesariamente otras tareas tales como el análisis y diseño de la aplicación (pero sí el diseño del código), aunque sí suelen estar fusionadas en el desarrollo de pequeñas aplicaciones.

Programación es el proceso a través del cual un programa o aplicación informática es desarrollado.

Se conoce como programación en ciencias de la computación a los pasos que se abordan para crear el código fuente de un programa informático. De acuerdo con estos pasos, el código se escribe, se prueba y se perfecciona.

El software informático es aquel ejecutado por el hardware o dispositivos de una computadora, permitiendo que el usuario acceda a funciones y operaciones de todo tipo. Así, la programación es una de las actividades más determinantes en el desarrollo de sistemas eficientes, rápidos y amigables para todo tipo de usuarios.

Se conoce como programadores a aquello encargados de desarrollar este código con instrucciones para que el software se comporte de una u otra manera de acuerdo con las órdenes que reciba. El lenguaje de programación es, por otro lado, la serie de parámetros y códigos de los que el programador se vale para desarrollar software.

Historia de la programación

Gottfried Wilheml von Leibniz (1646-1716), quien aprendió matemáticas de forma autodidacta (método no aconsejable en programación) construyó una máquina similar a la de Pascal, aunque algo más compleja, podía dividir, multiplicar y resolver raíces cuadradas.

Pero quien realmente influyó en el diseño de los primeros computadores fue Charles Babbage (1793-1871). Con la colaboración de la hija de Lord Byron, Lady Ada Countess of Lovelace (1815-1852), a la que debe su nombre el lenguaje ADA creado por el DoD (Departamento de defensa de Estados Unidos) en los años 70. Babbage diseñó y construyó la "máquina diferencial" para el cálculo de polinomios. Más tarde diseñó la "máquina analítica" de propósito general, capaz de resolver cualquier operación matemática. Murió sin poder terminarla, debido al escepticismo de sus patrocinadores y a que la tecnología de la época no era lo suficientemente avanzada. Un equipo del Museo de las Ciencias de Londres, en 1991, consiguió construir la máquina analítica de Babbage, totalmente funcional, siguiendo sus dibujos y especificaciones.

Un hito importante en la historia de la informática fueron las tarjetas perforadas como medio para "alimentar" los computadores. Lady Ada Lovelace propuso la utilización de las tarjetas perforadas en la máquina de Babbage. Para que se enteren todos esos machistas desaprensivos, el primer programador/a fue una mujer. En 1880 el censo en Estados Unidos tardó más de 7 años en realizarse. Es obvio que los datos no eran muy actualizados. Un asistente de la oficina del censo llamado Herman Hollerit (1860-1929) desarrolló un sistema para automatizar la pesada tarea del censo. Mediante tarjetas perforadas y un sistema de circuitos eléctricos, capaz de leer unas 60 tarjetas por minuto realizó el censo de 1890 en 3 años ahorrando tiempo y dinero. Más tarde fundó la Tabulating Machine Company y en 1924 tras alguna que otra fusión nació la Internacional Bussines Machines, IBM.

Las computadoras de hoy en día se sustentan en la lógica matemática basada en un sistema binario. Dicho sistema se implementa sobre dispositivos electrónicos que permiten, o no, pasar la corriente, con lo que se consiguen los 2 estados binarios: 0 y 1. A mediados del siglo XX, cuando se empezaron a construir las primeras computadoras digitales, se utilizaban tubos de vacío para implementar los 2 estados binarios, pero ¿cómo aparecieron estos conceptos? Alan Mathison Turing (1912-1954) diseñó una calculadora universal para resolver cualquier problema, la "máquina de Turing". Tuvo mucha influencia en el desarrollo de la lógica matemática. En 1937 hizo una de sus primeras contribuciones a la lógica matemática y en 1943 plasmó sus ideas en una computadora que utilizaba tubos de vacío. George Boole (1815-1864) también contribuyó al algebra binaria y a los sistemas de circuitos de computadora, de hecho, en su honor fue bautizada el álgebra booleana.

La primera computadora digital electrónica patentada fue obra de John Vincent Atanasoff (1903-1995). Conocedor de los inventos de Pascal y Babbage, y ayudado por Clifford Berry (1918-1963), construyó el Atanasoff Berry Computer (ABC). El ABC se desarrolló entre 1937 y 1942. Consistía en una calculadora electrónica que utilizaba tubos de vacío y estaba basada en el sistema binario (sistema numérico en el que se combinan los valores verdadero y falso, o 0 y 1).

Entre 1939 y 1944, Howard Aiken (1900-1973) de la universidad de Harvard en colaboración con IBM desarrolló el Mark 1. Era una computadora electromecánica de 16 metros de largo y más de dos de alto. Tenía 700.000 elementos móviles y varios centenares de kilómetros de cables. Podía realizar las cuatro operaciones básicas y trabajar con información almacenada en forma de tablas.

Por desgracia, los avances tecnológicos suelen producirse gracias a los militares que se aprovechan de la ciencia para perfeccionar sus armas. En la Moore School de la Universidad de Pensilvania se estaba trabajando en un proyecto militar para realizar unas tablas de tiro para armas balísticas. Los cálculos eran enormes y se tardaban semanas en realizarlos.

Parece ser que John W. Mauchly (1907-1980), quien dirigía el departamento de física del Ursine College de Filadelfia vivió en casa de Atanasoff durante cuatro días a partir del 13 de junio de 1941, lo que seguramente aprovechó para conocer las ideas de Atanasoff.

Junto a John Presper Eckert (1919-1995), Mauchly desarrolló una computadora electrónica completamente operacional a gran escala, para acelerar los complicados cálculos del proyecto militar de la universidad Moore. Se terminó en 1946 y se llamó Electronic Numerical Integrator And Computer (ENIAC). El ENIAC tenía 18.000 tubos electrónicos integrados en un volumen de 84 metros cúbicos. Pesaba unas 30 toneladas y consumía alrededor de 100.000 vatios. Su capacidad de cálculo era de 5.000 operaciones por segundo, aunque tenía que programarse manualmente conectándola a 3 tableros que contenían más de 6000 interruptores. Cargar un programa podía ser una tarea de varios días. El calor dispado por semejante monstruo debía ser importante, y se necesitaba una instalación de aire acondicionado. En definitiva, un ordenador portátil... más o menos.

Puede que no os suene, pero quien conozca de "los entresijos de la informática" seguro que considera importante nombrar a Johann Ludwig Von Neumann (1903-1957), genio de las matemáticas, quien tuvo el honor de asistir a las clases de Albert Einstein en la universidad de Berlín. Autor de trabajos de lógica simbólica, matemática pura y aplicada, física y tecnología, publicó un artículo acerca del almacenamiento de los programas, en 1945. Proponía que los programas se guardaran en memoria al igual que los datos, en forma binaria. Esto tuvo como consecuencia el aumento de velocidad de los cálculos y la ausencia de errores producidos por fallos mecánicos al programar la máquina mediante cables.

En cuanto a la aparición de los lenguajes de programación, el archiconocido COBOL, que tantos problemas causó con el "efecto 2000", fue el primer lenguaje en el que no había que programar directamente en código binario, y fue Grace Murray Hoper en 1952, una oficial de la Marina de Estados Unidos desarrolló el primer compilador, un programa que puede traducir enunciados parecidos al inglés en un código binario comprensible para la maquina llamado COBOL (COmmon Business-Oriented Languaje).

A partir de ahí, los avances han sido vertiginosos:

* La utilización del transistor en las computadoras en 1958, sustituyendo los tubos de vacío
* La aparición del circuito integrado de mano de Jack Kilby, también en 1958
* La miniaturización de un circuito electrónico en un chip de silicio en 1961
* El primer microprocesador, el 4004 de Intel, en 1971
* Gary Kildall crea el sistema operativo CP/M en 1973
* IBM comercializa el primer PC en 1980

Recordando a los primeros tiempos del ENIAC, con enormes computadores, en 1998 se terminó el proyecto Blue Pacific. La "maquinita" tiene la nada despreciable cantidad de 5856 procesadores que en conjunto tienen una velocidad de 3'9 teraflops, 2'6 Terabytes de memoria, ocupa 2400 metros cuadrados y tiene un peso de 47 toneladas. Se utiliza para la simulación de explosiones nucleares, y "ha salido" por unos 13000 millones de pesetas... baratito.

Hay muchos más personajes que intervienen en la historia y que han realizado grandes aportaciones, pero no es cuestión de extenderse.

¿Qué tipo de lenguaje utiliza la computadora?

Para crear un programa, y que la computadora lo interprete y ejecute las instrucciones escritas en él, debe escribirse en un lenguaje de programación. En sus inicios las computadoras interpretaban solo instrucciones en un lenguaje específico, del más bajo nivel, conocido como código máquina, siendo éste excesivamente complicado para programar. De hecho, solo consiste en cadenas de números 1 y 0 (sistema binario).

Para facilitar el trabajo de programación, los primeros científicos, que trabajaban en el área, decidieron reemplazar las instrucciones, secuencias de unos y ceros, por palabras o abreviaturas provenientes del inglés; las codificaron y crearon así un lenguaje de mayor nivel, que se conoce como Assembly o lenguaje ensamblador. Por ejemplo, para sumar se podría usar la letra A de la palabra inglesa add (sumar).

En realidad, escribir en lenguaje ensamblador es básicamente lo mismo que hacerlo en lenguaje máquina, pero las letras y palabras son bastante más fáciles de recordar y entender que secuencias de números binarios. A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de alto nivel. Mientras que una tarea tan trivial como multiplicar dos números puede necesitar un conjunto de instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará con solo una. Una vez que se termina de escribir un programa, sea en ensamblador o en algunos lenguajes de alto nivel, es necesario compilarlo, es decir, traducirlo completo a lenguaje máquina.1 Eventualmente será necesaria otra fase denominada comúnmente link o enlace, durante la cual se anexan al código, generado durante la compilación, los recursos necesarios de alguna biblioteca. En algunos lenguajes de programación, puede no ser requerido el proceso de compilación y enlace, ya que pueden trabajar en modo intérprete. Esta modalidad de trabajo es equivalente, pero se realiza instrucción por instrucción, a medida que es ejecutado en programación.

Léxico

La programación se rige por reglas y un conjunto más o menos reducido de órdenes, expresiones, instrucciones y comandos que tienden a asemejarse a una lengua natural acotada (en inglés); y que además tienen la particularidad de una reducida ambigüedad. Cuanto menos ambiguo es un lenguaje de programación, se dice, es más potente. Bajo esta premisa, y en el extremo, el lenguaje más potente existente es el binario, con ambigüedad nula (lo cual lleva a pensar así del lenguaje ensamblador).

En los lenguajes de programación de alto nivel se distinguen diversos elementos entre los que se incluyen el léxico propio del lenguaje y las reglas semánticas y sintácticas.

Algoritmo

En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. Los algoritmos son el objeto de estudio de la algoritmia.

Definición

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un cálculo o un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida). Sin embargo, cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular.

A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos. Esto fue realizado por Alonzo Church en 1936 con el concepto de "calculabilidad efectiva" basada en su cálculo lambda y por Alan Turing basándose en la máquina de Turing. Los dos enfoques son equivalentes, en el sentido en que se pueden resolver exactamente los mismos problemas con ambos enfoques. Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o gráficas mientras que, en general, los algoritmos funcionan sobre una vasta cantidad de estructuras de datos.

En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:

Tiempo secuencial: Un algoritmo funciona en tiempo discretizado –paso a paso–, definiendo así una secuencia de estados computacionales por cada entrada válida.

Estado abstracto: Cada estado computacional puede ser descrito formalmente utilizando una estructura de primer orden y cada algoritmo es independiente de su implementación (los algoritmos son objetos abstractos) de manera que en un algoritmo las estructuras de primer orden son invariantes bajo isomorfismo.

Exploración acotada: La transición de un estado al siguiente queda completamente determinada por una descripción fija y finita; es decir, entre cada estado y el siguiente solamente se puede tomar en cuenta una cantidad fija y limitada de términos del estado actual.

Medios de expresión

Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros.

La descripción de un algoritmo usualmente se hace en tres niveles:

1. Descripción de alto nivel. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.
2. Descripción formal. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
3. Implementación. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.

Diagrama de flujo

Los diagramas de flujo son descripciones gráficas de algoritmos; usan símbolos conectados con flechas para indicar la secuencia de instrucciones y están regidos por ISO.

Los diagramas de flujo son usados para representar algoritmos pequeños, ya que abarcan mucho espacio y su construcción es laboriosa. Por su facilidad de lectura son usados como introducción a los algoritmos, descripción de un lenguaje y descripción de procesos a personas ajenas a la computación.

Pseudocódigo

El pseudocódigo (falso lenguaje, el prefijo pseudo significa falso) es una descripción de alto nivel de un algoritmo que emplea una mezcla de lenguaje natural con algunas convenciones sintácticas propias de lenguajes de programación, como asignaciones, ciclos y condicionales, aunque no está regido por ningún estándar. Es utilizado para describir algoritmos en libros y publicaciones científicas, y como producto intermedio durante el desarrollo de un algoritmo, como los diagramas de flujo, aunque presentan una ventaja importante sobre estos, y es que los algoritmos descritos en pseudocódigo requieren menos espacio para representar instrucciones complejas.

El pseudocódigo está pensado para facilitar a las personas el entendimiento de un algoritmo, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación. Programadores diferentes suelen utilizar convenciones distintas, que pueden estar basadas en la sintaxis de lenguajes de programación concretos. Sin embargo, el pseudocódigo, en general, es comprensible sin necesidad de conocer o utilizar un entorno de programación específico, y es a la vez suficientemente estructurado para que su implementación se pueda hacer directamente a partir de él.

Así el pseudocódigo cumple con las funciones antes mencionadas para representar algo abstracto los protocolos son los lenguajes para la programación. Busque fuentes más precisas para tener mayor comprensión del tema.

Sistema formal

Un sistema formal es un tipo de sistema lógico-deductivo constituido por un lenguaje formal, una gramática formal que restringe cuales son las expresiones correctamente formadas de dicho lenguaje y las reglas de inferencia y un conjunto de axiomas que permite encontrar las proposiciones derivables de dichos axiomas. Los sistemas formales también han encontrado aplicación dentro de la informática, la teoría de la información, y la estadística, para proporcionar una definición rigurosa del concepto de demostración. La noción de sistema formal corresponde a una formalización rigurosa y completa del concepto de sistema axiomático, los cuales pueden ser expresados en lenguaje formal o en lenguaje natural formalizado.

En la Teoría de la demostración, las demostraciones formales pueden expresarse en el lenguaje de los sistemas formales, consistentes en axiomas y reglas de inferencia. Los teoremas pueden ser obtenidos por medio de demostraciones formales. Este punto de vista de las matemáticas ha sido denominado formalista; aunque en muchas ocasiones este término conlleva una acepción peyorativa.

Variables

En programación, una variable está formada por un espacio en el sistema de almacenaje (memoria principal de un ordenador) y un nombre simbólico (un identificador) que está asociado a dicho espacio. Ese espacio contiene una cantidad de información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa.

En computación una variable puede ser utilizada en un proceso repetitivo: puede asignársele un valor en un sitio, ser luego utilizada en otro, más adelante reasignársele un nuevo valor para más tarde utilizarla de la misma manera. Procedimientos de este tipo son conocidos con el nombre de iteración. En programación de computadoras, a las variables, frecuentemente se le asignan nombres largos para hacerlos relativamente descriptivas para su uso, mientras que las variables en matemáticas a menudo tienen nombres escuetos, formados por uno o dos caracteres para hacer breve en su transcripción y manipulación.

Tipos de datos

Debido a que las variables contienen o apuntan a valores de tipos determinados, las operaciones sobre las mismas y el dominio de sus propios valores están determinadas por el tipo de datos en cuestión. Algunos tipos de datos usados:

Tipo de dato lógico.

Tipo de dato entero.

Tipo de dato de coma flotante (real, con decimales).

Tipo de dato carácter.

Tipo de dato cadena.

Ámbito

Local: Cuando la misma sólo es accesible desde un único procedimiento hijo, no pudiendo ser leída o modificada desde otro procedimiento hermano o desde el propio procedimiento padre. Es posible declarar variables en bloques de condición, bucles, etc de tal modo que sólo pueda accederse a ellas en el propio bloque.

Global: Cuando la misma es accesible tanto desde rutinas o macros de la aplicación, como en todos los procedimientos y funciones de la misma.

Estructura Secuencial

La estructura secuencial es aquella en la que una acción sigue a otra en secuencia. Las operaciones se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. La asignación de esto consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:

1. Simples: Consiste en pasar un valor constante a una variable (a ← 15).
2. Contador: Consiste en usarla como un verificador del número de veces que se realiza un proceso (a ← a + 1).
3. Acumulador: Consiste en usarla como un sumador en un proceso (a ← a + b).
4. De trabajo: Donde puede recibir el resultado de una operación matemática que involucre muchas variables (a ← c + b\*1/2).

Compilación

El programa escrito en un lenguaje de programación de alto nivel es llamado programa fuente y no se puede ejecutar directamente en una computadora. La opción más común es compilar el programa obteniendo un módulo objeto, aunque también puede ejecutarse en forma más directa a través de un intérprete informático.

El código fuente del programa se debe someter a un proceso de traducción para convertirlo a lenguaje máquina o bien a un código intermedio, generando así un módulo denominado "objeto". A este proceso se le llama compilación.

Objetivos de la programación

La programación debe perseguir la obtención de programas de calidad. Para ello se establece una serie de factores que determinan la calidad de un programa. Algunos de los factores de calidad más importantes son los siguientes:

Correctitud. Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente qué debe hacer el programa antes de su desarrollo y, una vez acabado, compararlo con lo que realmente hace.

Claridad. Es muy importante que el programa sea lo más claro y legible posible, para facilitar tanto su desarrollo como su posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo de programación. De esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones de código; una práctica común es realizar aclaraciones en el código fuente utilizando líneas de comentarios. Contrariamente, algunos por diversión o para impedirles un análisis cómodo a otros programadores, recurren al uso de código ofuscado.

Eficiencia. Se trata de que el programa, además de realizar aquello para lo que fue creado (es decir, que sea correcto), lo haga gestionando de la mejor forma posible los recursos que utiliza.

Normalmente, al hablar de eficiencia de un programa, se suele hacer referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración para mejorar la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico en la red que genera, etc.).

Portabilidad. Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquella en la que se desarrolló. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha elaborado para el sistema GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

Tipos de lenguaje en la programación

Lenguaje de bajo nivel

Un lenguaje de programación de características bajo nivel es aquel en el que sus instrucciones ejercen un control directo sobre el hardware y están condicionados por la estructura física de las computadoras que lo soportan. El uso de la palabra bajo en su denominación no implica que el lenguaje sea menos potente que un lenguaje de alto nivel, sino que se refiere a la reducida abstracción entre el lenguaje y el hardware. Por ejemplo, se utiliza este tipo de lenguajes para programar tareas críticas de los sistemas operativos, de aplicaciones en tiempo real o controladores de dispositivos.

Dicho lenguaje es muy simple o nada complicado, pero estructurar programas a ese nivel es muy difícil. Dado que este lenguaje viene dado por las especificaciones técnicas del hardware, no permite una abstracción fuera de lo estipulado para el microprocesador de un ordenador. Consecuentemente, es fácilmente trasladado a lenguaje de máquina.

Las estructuras de los lenguajes son como sigue:

1. Código Binario - Es el lenguaje básico, sólo admite todo (1) o nada (0). Todo sistema informático está basado en este código, ya que el 1 (todo, SÍ) quiere decir que se permite el paso de la electricidad y el 0 (nada, NO) no lo permite. Sería la forma en la que están almacenados los programas, sea en memoria, sea en dispositivos de almacenamiento. De esta forma son recibidas y ejecutadas cada una de las instrucciones por la CPU del ordenador.
2. Lenguaje Máquina - Las invocaciones a memoria, como los procesos aritméticos lógicos son posiciones literales de conmutadores físicos del hardware en su representación booleana. Estos lenguajes son literales de tareas.
3. Lenguajes ensambladores - También denominados nemotécnicos o nemónicos, no son ya programas ejecutables directamente por el ordenador, sino textos de código fuente que necesitan de alguna herramienta para su conversión a lenguaje máquina, son los programas llamados ensambladores. Sus instrucciones suelen ser una denominación abreviada de la instrucción máquina que simbolizan, y tienen una correspondencia casi directa a las instrucciones máquina que representan. El código resultante de la ejecución del programa ensamblador generaría un código binario ejecutable.

En este tipo de lenguajes se trabaja a nivel de instrucciones, es decir, su programación es al más fino detalle, además, está completamente orientado a la máquina.

* Adaptación - Máxima entre programación y aprovechamiento del recurso de la máquina.
* Velocidad - Máxima al contar con un acceso directo a los recursos, sin capas intermedias.
* Portabilidad - Mínima por estar restringido a las especificaciones del fabricante.
* Abstracción - Mínima por depender completamente de la técnica del hardware.
* Uso - Requiere de la máxima atención y de una organización estructurada sobre la base de los planos del hardware y del objetivo del software.
* isomorfismo - Conceptualización de los datos de la información, que se complementaran.

Lenguaje intermedio

En ciencias de la computación, un lenguaje intermedio es el lenguaje de una máquina abstracta diseñada para ayudar a realizar el análisis de un programa informático. El término proviene de su uso en compiladores, donde el código fuente de un programa es traducido a un modo más apropiado para transformaciones de mejora de código antes de generar el código objeto o código máquina para una máquina determinada.

El diseño del lenguaje intermedio difiere típicamente del lenguaje de máquina de tres maneras fundamentales:

* Cada instrucción representa exactamente una operación fundamental; por ejemplo, los modos de direccionamiento "shift-add" (desplazar y añadir) comunes en microprocesadores no están presentes.
* La información de la estructura de control puede no estar incluida en el juego de instrucciones.
* El número de registros disponibles puede ser grande, incluso ilimitado.

Un formato popular para lenguajes intermedios es el de código de tres direcciones.

El término también es usado para referirse a lenguajes usados como intermedios por algunos lenguajes de alto nivel que no crean código objeto o código máquina por sí mismos, sino que solamente el lenguaje intermedio. Este lenguaje intermedio es enviado a un compilador para tal lenguaje, que genera el código objeto o código máquina finalizado.

Aunque no está específicamente diseñado como un lenguaje intermedio, la naturaleza de C como una abstracción del lenguaje ensamblador y su uso como lenguaje de sistema en Unix-like y otros sistemas operativos lo han convertido en un lenguaje intermedio popular: Eiffel, Sather, Esterel, algunos dialectos de Lisp (Lush, Gambit, Haskell, Glasgow Haskell Compiler), Squeak, Cython, Seed7, Vala y otros usan C como un lenguaje intermedio. Variantes de C han sido diseñadas para proveer a C de características como un lenguaje ensamblador portable, incluyendo C-- o el lenguaje intermedio de C.

Lenguaje de alto nivel

Un lenguaje de programación de alto nivel se caracteriza por expresar el algoritmo de una manera adecuada a la capacidad cognitiva humana, en lugar de la capacidad ejecutora de las máquinas.

En los primeros lenguajes, la limitación era que se orientaban a un área específica y sus instrucciones requerían de una sintaxis predefinida. Se clasifican como lenguajes procedimentales o lenguajes de bajo nivel. Otra limitación de estos es que se requiere de ciertos conocimientos de programación para realizar las secuencias de instrucciones lógicas. Los lenguajes de alto nivel se crearon para que el usuario común pudiese solucionar un problema de procesamiento de datos de una manera más fácil y rápida.

Por esta razón, a finales de los años 1950 surgió un nuevo tipo de lenguajes de programación que evitaba estos inconvenientes, a costa de ceder un poco en las ventajas. Estos lenguajes se llaman de tercera generación o de nivel alto, en contraposición a los de bajo nivel o de nivel próximo a la máquina.

Lenguajes interpretados

En ciencias de la computación, intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

Usando un intérprete, un solo archivo fuente puede producir resultados iguales incluso en sistemas sumamente diferentes (ejemplo. una PC y una PlayStation 4). Usando un compilador, un solo archivo fuente puede producir resultados iguales solo si es compilado a distintos ejecutables específicos a cada sistema.

Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración (lo que se traduce, por ejemplo, en una mayor facilidad para reemplazar partes enteras del programa o añadir módulos completamente nuevos), y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete (lo que se conoce comúnmente como máquina virtual).

Para mejorar el desempeño, algunas implementaciones de programación de lenguajes de programación pueden interpretar o compilar el código fuente original en una más compacta forma intermedia y después traducir eso al código de máquina (ej. Perl, Python, MATLAB, y Ruby). Algunos aceptan los archivos fuente guardados en esta representación intermedia (ej. Python, UCSD Pascal y Java).

En la actualidad, uno de los entornos más comunes de uso de los intérpretes es en los navegadores web, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma.

Eficiencia

La desventaja principal de los interpretadores es que cuando se interpreta un programa, típicamente corre más lentamente que si hubiera sido compilado. La diferencia en velocidades puede ser minúscula o grande; a menudo un orden de magnitud y a veces más. Generalmente toma más tiempo correr un programa bajo un interpretador que correr el código compilado, pero puede tomar menos tiempo para interpretarlo que el tiempo total requerido para compilarlo y ejecutarlo. Esto es especialmente importante si se está haciendo y probando un código prototipo cuando un ciclo de editar, interpretar y depurar del interpretador, a menudo puede ser mucho más corto que el ciclo de editar, compilar, ejecutar y depurar del compilador.

La interpretación de código es más lenta que la ejecución de código compilado porque el interpretador debe analizar cada sentencia en el programa cada vez que es ejecutada y entonces realizar la acción deseada, mientras que el código compilado solo realiza la acción dentro de un determinado contexto fijo por la compilación. Este análisis en tiempo de ejecución se conoce como "sobrecarga interpretativa". En un interpretador, el acceso a las variables es también más lento porque el mapeo de identificadores hacia las localizaciones de almacenamiento debe hacerse repetidamente en tiempo de ejecución en vez de en el tiempo de compilación. Hay varios compromisos entre la velocidad de desarrollo al usar un interpretador y la velocidad de ejecución al usar un compilador. Algunos sistemas (ej., algunos LISPs) permiten al código interpretado y al compilado llamarse el uno al otro y compartir variables. Esto significa que una vez que una rutina ha sido probada y depurada bajo el interpretador puede ser compilada y por lo tanto beneficiarse de una ejecución más rápida mientras que otras rutinas están siendo desarrolladas. Muchos interpretadores no ejecutan el código fuente tal y como está sino que lo convierten en una forma interna más compacta. Por ejemplo, algunos interpretadores BASIC reemplazan palabras clave (keywords) con tokens de un simple byte que pueden ser usados para encontrar la instrucción en una tabla de saltos.

Un interpretador puede bien usar el mismo analizador lexicográfico y el analizador sintáctico (parser) que el compilador y entonces interpretar el árbol de sintaxis abstracta resultante.

En el espectro entre la interpretación y la compilación, otro acercamiento está transformando el código fuente en un árbol de sintaxis abstracta optimizado (AST), y después procediendo a ejecutar el programa siguiendo esta estructura arborescente. En este acercamiento cada sentencia necesita ser analizada (parsed) solo una vez. Como una ventaja sobre el bytecode, el AST mantiene la estructura y las relaciones globales del programa entre las sentencias (que se pierden en una representación de bytecode), y proporciona una representación más compacta.

Así, el AST se ha propuesto como un mejor formato intermedio para los compiladores justo a tiempo que el bytecode. También, permite realizar un mejor análisis durante tiempo de ejecución. Un interpretador Java basado en AST ha demostrado ser más rápido que un interpretador similar basado en bytecode, gracias a las más poderosas optimizaciones permitidas al tener la estructura completa del programa, así como tipos de datos de alto nivel, disponibles durante la ejecución.

Compilación

Para desdibujar más la distinción entre los interpretadores, los interpretadores de bytecode y la compilación, está la compilación justo a tiempo (o JIT), una técnica en la cual la representación intermedia es compilada a código de máquina nativo en tiempo de ejecución. Esto confiere la eficiencia de ejecutar el código nativo, al costo de tiempo de inicio y de un uso creciente de la memoria cuando el bytecode o el AST es compilado por primera vez. La optimización adaptativa es una técnica complementaria en la cual el interpretador hace un análisis de desempeño del programa que está corriendo (profiling) y compila sus partes más frecuentemente ejecutadas a código nativo. Ambas técnicas tienen algunas décadas, apareciendo en lenguajes tales como Smalltalk en la década de 1980.

En años recientes, la compilación justo a tiempo ha ganado la atención de la mayoría de los implementadores de lenguajes de programación, con Java, Python, y el Microsoft .NET Framework todos ahora incluyendo JITs.

Lenguaje compilado

Un lenguaje compilado es un lenguaje de programación cuyas implementaciones son normalmente compiladores (traductores que generan código de máquina a partir del código fuente) y no intérpretes (ejecutores paso a paso del código fuente, donde no se lleva a cabo una traducción en la pre ejecución).

El término es un tanto vago. En principio, cualquier lenguaje puede ser implementado con un compilador o un intérprete. Sin embargo, es cada vez más frecuente una combinación de ambas soluciones: un compilador puede traducir el código fuente en alguna forma intermedia (muchas veces llamado Bytecode), que luego se pasa a un intérprete que lo ejecuta.

Ventajas y desventajas

Los programas compilados a código nativo en tiempo de compilación tienden a ser más rápidos que los traducidos en tiempo de ejecución, debido a la sobrecarga del proceso de traducción. Sin embargo, las nuevas tecnologías como la compilación en tiempo de ejecución, y mejoras generales en el proceso de traducción están empezando a reducir esta brecha. En algún punto intermedio, tiende a ser más eficiente la solución mixta usando bytecode.

Los lenguajes de programación de bajo nivel son típicamente compilados, en especial cuando la eficiencia es la principal preocupación, en lugar de soporte de plataformas cruzadas. Para los lenguajes de bajo nivel, hay más correspondencias uno a uno entre el código programado y las operaciones de hardware realizadas por el código máquina, lo que hace que sea más fácil para los programadores controlar más finamente la CPU y uso de memoria.

Con un poco de esfuerzo siempre es posible escribir compiladores incluso para los lenguajes tradicionalmente interpretados. Por ejemplo, Common Lisp puede ser compilado a Java bytecode, que es interpretado por la máquina virtual de Java; a código C, que se compila a código máquina nativo; o es compilado directamente a código nativo.

Los lenguajes de programación que soportan múltiples objetivos de compilación ofrecen un mayor control para que el desarrollador elija la velocidad de ejecución o la compatibilidad entre plataformas.

Lenguaje de programación declarativo

Se les conoce como lenguajes declarativos en ciencias computacionales a aquellos lenguajes de programación en los cuales se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando, por ejemplo: Obtener los nombres de todos los empleados que tengan más de 32 años. Eso se puede lograr con un lenguaje declarativo como SQL.

La programación declarativa es una forma de programación que implica la descripción de un problema dado en lugar de proveer una solución para dicho problema, dejando la interpretación de los pasos específicos para llegar a dicha solución a un intérprete no especificado. La programación declarativa adopta, por lo tanto, un enfoque diferente al de la programación imperativa tradicional.

En otras palabras, la programación declarativa provee el "qué", pero deja el "cómo" liberado a la implementación particular del intérprete. Por lo tanto, se puede ver que la programación declarativa tiene dos fases bien diferenciadas, la declaración y la interpretación.

Es importante señalar que, a pesar de hacer referencia a intérprete, no hay que limitarse a "lenguajes interpretados" en el sentido habitual del término, sino que también se puede estar trabajando con "lenguajes compilados".

Características

Los lenguajes declarativos están orientados a buscar la solución del problema, sin preocuparse por la forma de llegar a ello; es decir, el programador debe concentrarse en la lógica del algoritmo, más que en el control de la secuencia.

* Los programas están formados por un conjunto de definiciones o ecuaciones, las cuales describen lo que debe ser calculado, no en sí la forma de hacerlo.
* Las variables sólo pueden tener asignado un solo valor a lo largo de la ejecución del programa, lo cual implica que no puede existir asignación destructiva. Debido a esto, cobra especial importancia el uso del anidamiento y la recursividad.
* Las listas representan la estructura fundamental de datos.
* El orden de la ejecución no resulta importante debido a que no existen efectos colaterales; es decir, que al calcular un valor, resulta imposible afectar el cálculo de otros y con esto se puede afirmar que cualquier secuencia de ejecución deberá conducir al mismo resultado.
* Las expresiones o definiciones pueden ser usadas como valores y por lo tanto se pueden tratar como argumentos de otras definiciones.
* El control de la ejecución no es responsabilidad del programador.

Desventajas

La principal desventaja de la programación declarativa es que no puede resolver cualquier problema dado, sino que está restringida al subconjunto de problemas para los que el intérprete o compilador fue diseñado.

Otra desventaja de la programación declarativa está relacionada con la eficiencia. Dado que es necesaria una fase de interpretación extra, en la cual se deben evaluar todas las consecuencias de todas las declaraciones realizadas, el proceso es relativamente más lento que en la programación imperativa, en que los cambios de estado del sistema están dados por instrucciones particulares y no por un conjunto de condiciones arbitrariamente grande.

Ventajas

A pesar de lo anterior existen algunas ventajas en el uso de la programación declarativa. Entre las ventajas se destaca que la solución de un problema se puede realizar con un nivel de abstracción considerablemente alto, sin entrar en detalles de implementación irrelevantes, lo que hace a las soluciones más fácil de entender por las personas. La resolución de problemas complejos es resuelta por el intérprete a partir de la declaración de las condiciones dadas.

La programación declarativa es muy usada en la resolución de problemas relacionados con inteligencia artificial, bases de datos, configuración, y comunicación entre procesos; sin embargo, ningún leguaje declarativo se aproxima en popularidad a los lenguajes imperativos.

Programación lógica

La idea fundamental de la programación lógica consiste en emplear la lógica como lenguaje de programación.

La lógica no es imperativa porque no sirve para indicar cómo resolver un problema (órdenes). La lógica es declarativa porque sirve para especificar qué problema resolver (condiciones).

En la programación lógica, se especifican las condiciones que satisfacen las soluciones, se deducen las soluciones a partir de las condiciones y el énfasis de todo está en qué problema resolver. El problema se describe especificando qué caracteriza a sus posibles soluciones.

La programación lógica, junto con la funcional, forma parte de lo que se conoce como programación declarativa. En los lenguajes tradicionales, la programación consiste en indicar cómo resolver un problema mediante sentencias; en la programación lógica, se trabaja de forma descriptiva, estableciendo relaciones entre entidades, indicando no cómo, sino qué hacer. Se establece entonces que la idea esencial de la programación lógica es: algoritmos = lógica + control.

Es decir, un algoritmo se construye especificando conocimiento en un lenguaje formal (lógica de primer orden), y el problema se resuelve mediante un mecanismo de inferencia (control) que actúa sobre aquél.

Al hacer un recorrido por la programación lógica, aparece como uno de sus lenguajes más representativos, Prolog, que es un clásico de la inteligencia artificial y que se aplica de múltiples formas en el desarrollo de software comercial.

Motivación

Históricamente, los ordenadores se han programado utilizando lenguajes muy cercanos a las peculiaridades de la propia máquina: operaciones aritméticas simples, instrucciones de acceso a memoria, etc. Un programa escrito de esta manera puede ocultar totalmente su propósito a la comprensión de un ser humano, incluso uno entrenado. Hoy día, estos lenguajes pertenecientes al paradigma de la Programación imperativa han evolucionado de manera que ya no son tan crípticos.

En cambio, la lógica matemática es la manera más sencilla, para el intelecto humano, de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas. De ahí que el concepto de "programación lógica" resulte atractivo en diversos campos donde la programación tradicional es un fracaso.

Campos de aplicación

La programación lógica encuentra su hábitat natural en aplicaciones de inteligencia artificial o relacionada:

* Sistemas expertos, donde un sistema de información imita las recomendaciones de un experto sobre algún dominio de conocimiento.
* Demostración automática de teoremas, donde un programa genera nuevos teoremas sobre una teoría existente.
* Reconocimiento de lenguaje natural, donde un programa es capaz de comprender (con limitaciones) la información contenida en una expresión lingüística humana.

La programación lógica también se utiliza en aplicaciones más "mundanas" pero de manera muy limitada, ya que la programación tradicional es más adecuada a tareas de propósito general.

Fundamentos

La mayoría de los lenguajes de programación lógica se basan en la teoría lógica de primer orden, aunque también incorporan algunos comportamientos de orden superior como la lógica difusa. En este sentido, destacan los lenguajes funcionales, ya que se basan en el cálculo lambda, que es la única teoría lógica de orden superior que es demostradamente computable (hasta el momento).

Concepto

Un concepto importante de programación lógica es la descomposición de programas en sus componentes lógicos y sus componentes de control. Con lenguajes de programación lógica de bajo nivel, estos componentes determinan la solución del problema, por eso los componentes de control pueden variar para proporcionar alternancia de ejecución de un programa lógico. Estos conceptos son capturados con el eslogan

Algoritmo= lógica + control

donde "lógica" representa un programa lógico y "control" diferentes estrategias de demostración del teorema.

Programación funcional

La programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas. El objetivo de la programación funcional es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo (pues la programación funcional es declarativa), sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus sub expresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al permitir conceptos tomados de los lenguajes imperativos, como las secuencias de instrucciones o la asignación de variables. En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje híbrido.

Programación a base de datos

Las bases de datos son programas que administran información y hacen más ordenada la información, aparte de hacer la fácil de buscar y por supuesto de encontrar.

Las características de las bases de datos pueden ser ventajosas o desventajosas: pueden ayudar a almacenar, organizar, recuperar, comunicar y manejar información en formas que serían imposibles sin las computadoras, pero también afecta de alguna manera ya que existen enormes cantidades de información en bases de datos de las que no se tiene control del acceso.

Las bases de datos tienen muchos usos: facilitan el almacenamiento de grandes cantidades de información; permiten la recuperación rápida y flexible de información, con ellas se puede organizar y reorganizar la información, así como imprimirla o distribuirla en formas diversas.

Es claro que los lenguajes orientados a bases de datos son declarativos y no imperativos, pues el problema es "qué" se quiere hacer o "qué" se necesita buscar y encontrar en la base de datos, y no se enfatiza el "cómo".

Una base de datos también se puede definir como un banco de datos o conjunto de datos que pertenecen al mismo contexto, almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

En la actualidad, y gracias al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos.

Los sistemas gestores de bases de datos (SGBD) permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

Tipos de bases de datos

Las bases de datos pueden clasificarse de varias maneras, de acuerdo al criterio elegido para su clasificación, que puede ser según la variabilidad de los datos almacenados o según el contenido.

Clasificación de bases de datos según la variabilidad:

Esta clasificación depende de cómo los datos cambian o varían dentro de la base de datos, o si se mantienen estáticos sin sufrir modificaciones o cambios. Dentro de esta clasificación se tiene lo siguiente:

Bases de datos estáticas: Éstas son bases de datos de sólo lectura, utilizadas primordialmente para almacenar datos históricos que posteriormente se pueden utilizar para estudiar el comportamiento de un conjunto de datos a través del tiempo, realizar proyecciones y tomar decisiones.

Bases de datos dinámicas: Éstas son bases de datos donde la información almacenada se modifica con el tiempo, permitiendo operaciones como actualización y adición de datos, además de las operaciones fundamentales de consulta. Un ejemplo de esto puede ser la base de datos utilizada en un sistema de información de una tienda de abarrotes, una farmacia, un videoclub, etc.

Clasificación de bases de datos según contenido

Esta clasificación basa su enfoque en el contenido o lo que almacena la base de datos, de esta manera:

Bases de datos bibliográficas: Solo contienen un "representante" de la fuente primaria, que permite localizarla. Un registro típico de una base de datos bibliográfica contiene información sobre el autor, fecha de publicación, editorial, título, edición, de una determinada publicación, etc. Puede contener un resumen o extracto de la publicación original, pero nunca el texto completo, porque si no se estaría en presencia de una base de datos a texto completo (o de fuentes primarias). Como su nombre lo indica, el contenido son cifras o números. Por ejemplo, una colección de resultados de análisis de laboratorio, entre otras.

Bases de datos de textos completos: Almacenan las fuentes primarias, como, por ejemplo, todo el contenido de todas las ediciones de una colección de revistas científicas.

Directorios: Un ejemplo son las guías telefónicas en formato electrónico.

Modelos de bases de datos

Además de la clasificación por la función de las bases de datos, éstas también se pueden clasificar de acuerdo a su modelo de administración de datos.

Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.

Algunos modelos con frecuencia utilizados en las bases de datos son:

* Bases de datos jerárquicas: Éstas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas. Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.
* Bases de datos de red: Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico). Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.
* Base de datos racional: Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información. El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales. Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización (reglas fundamentales para el buen funcionamiento de una base de datos relacional) de una base de datos. Durante los años '80 (1980-1989) la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión. La normalización de una base de datos relacional consiste en las siguientes reglas:

1. Primera Forma Normal (1FN): se eliminan los grupos repetitivos. Los atributos han de ser atómicos. Gráficamente las celdas de la tabla contienen solo un valor, en cada uno de los atributos sólo se puede incluir un dato, aunque sea compuesto, pero no se puede incluir una lista de datos. Se trata de que cada atributo guarde la menor cantidad de información posible. Para eliminar los grupos repetitivos se pone cada uno de ellos en una tabla aparte, esa nueva tabla hereda la clave primaria de la relación en la que se encontraban, se crea una clave foránea para la nueva tabla.
2. Segunda Forma Normal (2FN): dependencia completa. Está en 2FN si está en 1FN y si sus atributos no principales dependen de forma completa de la clave principal. Toda relación que tenga como clave sólo un atributo está en 2FN.
3. Tercera Forma Normal (3FN): dependencia transitiva. Está en segunda forma normal y todo atributo no primo es implicado por la clave primaria en una secuencia no transitiva.
4. Forma Normal de Boyce-Codd: una tabla está en FNBC, sí y sólo sí las únicas dependencias funcionales elementales son aquellas en las que la Clave primaria determina un atributo.
5. Cuarta Forma Normal (4FN): está en forma normal de Boyce-Codd y se eliminan las dependencias multivariadas y se generan todas las relaciones externas con otras tablas u otras bases de datos.
6. Quinta Forma Normal (5FN): está en cuarta forma normal y toda dependencia-join viene implicada por claves candidatas.

* Bases de datos orientadas a objetos: este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (estado y comportamiento). Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos: encapsulación (propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos), herencia (propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases y polimorfismo (propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos. En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos.
* Base de datos de documentales: Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes.
* Base de datos deductivas: un sistema de base de datos deductivas, es un sistema de base de datos, pero con la diferencia de que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos.
* También las bases de datos deductivas son llamadas base de datos lógica, a raíz de que se basan en lógica matemática.

Lenguajes de programación imperativos:

En ciencias de la computación se llama lenguajes imperativos a aquellos en los cuales se le ordena a la computadora cómo realizar una tarea siguiendo una serie de pasos o instrucciones, por ejemplo:

Paso 1, solicitar número.

Paso 2, multiplicar número por dos.

Paso 3, imprimir resultado de la operación.

Paso 4, etc,

El proceso anterior se puede realizar con un lenguaje imperativo como por ejemplo BASIC, C, C++, Java, Clipper, Dbase, C#, PHP, Perl, etc.

Dentro de la programación imperativa, se tiene un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

Los lenguajes imperativos se basan en comandos u órdenes que se le dan a la computadora para que haga algo, con el fin de organizar o cambiar valores en ciertas partes de la memoria.

La ejecución de estos comandos se realiza, en la mayor parte de ellos, secuencialmente, es decir, hasta que un comando no ha sido ejecutado no se lee el siguiente.

Según el dominio, o mejor dicho con el propósito que se utiliza el programa, se puede hablar de lenguajes de dominio específico y de dominio general.

Lenguajes imperativos procedurales:

En los lenguajes tradicionales o procedurales, es la aplicación quien controla qué porciones de código se ejecuta, y la secuencia en que este se ejecuta. La ejecución de la aplicación se inicia con la primera línea de código, y sigue una ruta predefinida a través de la aplicación, llamando procedimientos según sea necesario.

Los lenguajes procedurales están fundamentados en la utilización de variables para almacenar valores y en la realización de operaciones con los datos almacenados. Algunos ejemplos son: FORTRAN, PASCAL, C, ADA, ALGOL,…

En este tipo de lenguajes, la arquitectura consta de una secuencia de celdas, llamadas memoria, en las cuales se pueden guardar en forma codificada, lo mismo datos que instrucciones; y de un procesador, el cual es capaz de ejecutar de manera secuencial una serie de operaciones, principalmente aritméticas y booleanas, llamadas comandos. En general, un lenguaje procedural ofrece al programador conceptos que se traducen de forma natural al modelo de la máquina. El programador tiene que traducir la solución abstracta del problema a términos muy primitivos, cercanos a la máquina.

Con un lenguaje procedural el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. Así se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados. Las sentencias de un lenguaje procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual.

Diferencia entre lenguajes declarativos e imperativos

En los lenguajes declarativos las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

Los lenguajes imperativos describen paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado un programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar un problema.

Lenguajes de programación orientadas a objetos

En la Programación Orientada a Objetos (POO u OOP según siglas en inglés) se definen los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de la misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

Dada esta propiedad de conjunto de una clase de objetos, que al contar con una serie de atributos definitorios, requiere de unos métodos para poder tratarlos (lo que hace que ambos conceptos están íntimamente entrelazados), el programador debe pensar indistintamente en ambos términos, ya que no debe nunca separar o dar mayor importancia a los atributos a favor de los métodos, ni viceversa. Hacerlo puede llevar al programador a seguir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen esa información por otro (llegando a una programación estructurada camuflada en un lenguaje de programación orientada a objetos).

Esto difiere de los lenguajes imperativos tradicionales, en los que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. Los programadores de lenguajes imperativos escriben funciones y después les pasan los datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos y después envían mensajes a los objetos diciendo que realicen esos métodos por sí mismos.

Un objeto se puede definir como un grupo de procedimientos que comparten un estado. Se define al conjunto de datos como "estado", y "métodos" como el conjunto de procedimientos que pueden alterar ese estado. Un programa orientado a objetos es un método de implementación en el que los programas están organizados como colecciones de objetos, donde cada uno es una instancia de alguna clase, y donde todas las clases son miembros de una jerarquía de clases conectadas por relaciones de herencia. Este tipo de lenguajes son muy recientes en comparación a los primeros lenguajes de programación que aparecieron.

Conceptos

La programación orientada a objetos es una nueva forma de programar que trata de encontrar la solución a problemas de una forma que ofrece muchas ventajas y facilidades que no se tenían anteriormente. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entro ellos destacan los siguientes:

* Objeto: entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad ("métodos"). Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa).
* Clase: definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas. Una clase es una colección de objetos similares o la implementación, declaración o definición de un tipo de objeto. Cada vez que se construye un objeto de una clase se crea una instancia de esa clase. Por ejemplo en Visual Basic, se tiene la clase Form, y se pueden crear instancias de esa clase al tener Form1, Form2, etc. Así se está creando una instancia de la clase Form.
* Método: algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en lar propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
* Evento: un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente.
* Mensaje: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
* Propiedad o atributo: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto, y cuyo valor puede ser alterado por la ejecución de algún método.
* Estado interno: es una propiedad invisible de los objetos, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos).

Características:

Las características más importantes de la programación orientada a objetos son las siguientes:

* Abstracción: Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, unas variedades de técnicas son requeridas para ampliar una abstracción.
* Encapsulamiento: también llamado "ocultación de la información". Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos. Esta característica o propiedad permite por tanto ejecutar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
* Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Por ejemplo en Visual Basic, el polimorfismo se da al tener diferentes tipos de objetos (Form, Label, etc.)
* Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto pertenece a más de una clase se dice que hay herencia múltiple; esta característica no está soportada por algunos lenguajes (como Java). Con esta propiedad, los objetos heredan comportamientos dentro de una jerarquía de clases.

Ejemplos de lenguajes:

Lenguajes de alto nivel

Visual basic:

Visual Basic (VB) es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el framework .NET. Su introducción resultó muy controvertida, ya que debido a cambios significativos en el lenguaje VB.NET no es retro compatible con Visual Basic, pero el manejo de las instrucciones es similar a versiones anteriores de Visual Basic, facilitando así el desarrollo de aplicaciones más avanzadas con herramientas modernas. Para mantener eficacia en el desarrollo de las aplicaciones. La gran mayoría de programadores de VB.NET utilizan el entorno de desarrollo integrado Microsoft Visual Studio en alguna de sus versiones (desde el primer Visual Studio .NET hasta Visual Studio .NET 2015, que es la última versión de Visual Studio para la plataforma .NET), aunque existen otras alternativas, como SharpDevelop (que además es libre).

Visual 2008:

Para esta versión se añadieron varias novedades, incluyendo:

Soporte para LINQ

Expresiones lambda

Literales XML

Visual Basic 2010 ofrece soporte para entorno de ejecución dinámica.

VB 2010 forma parte de Microsoft Silverlight. Visual Basic es un lenguaje de programación que permite realizar diferentes programas de registro anidados.

Relación con Bisual Basic:

Si Visual Basic .NET debe considerarse una mera versión de Visual Basic, o si debe considerarse como un nuevo lenguaje de programación es un tema que ha traído mucha discusión, y que aún la trae.

La sintaxis básica es prácticamente la misma entre VB y VB.NET, con la excepción de los añadidos para soportar nuevas características como el control estructurado de excepciones, la programación orientada a objetos, o los Genéricos.

Las diferencias entre VB y VB.NET son profundas, sobre todo en cuanto a metodología de programación y bibliotecas, pero ambos lenguajes siguen manteniendo un gran parecido, cosa que facilita notablemente el paso de VB a VB.NET.

Visual studio.net 2002

Visual Studio .NET se publicó en 2002 y fue la primera versión de Visual Studio en introducir el framework .NET. Esta versión de Visual Studio introdujo, junto con el Framework .NET tres nuevos lenguajes de programación, Visual C#, VB.NET y Visual J#.

En esta primera versión de Visual Studio .NET se podían programar aplicaciones Windows.Forms (aplicaciones de escritorio) y aplicaciones ASP.NET (Aplicaciones Web).

Visual studio.net 2003

Visual Studio .NET 2003 se publicó en 2003 fue una actualización menor de Visual Studio .NET, básicamente propiciada por la introducción de la versión 1.1 del Framework .NET.

En esta versión se añadió por primera vez la posibilidad de programar para dispositivos móviles usando .NET, ya fuera usando el Compact Framework, o ASP.NET.

Añade soporte de 64-bit (x86-64: AMD64 e Intel 64, e IA-64: Itanium)

Ediciones: Express, Standard, ready bost, Professional, Tools for Office, y 5 ediciones Visual Studio Team System (Architects, Software Developers, Testers, y Database Professionals)

La versión interna de Visual Studio 2005 es la 8.0, mientras que el formato del archivo es la 9.0.

Visual studio.net 2008

El IDE de Visual Studio 2008 permite trabajar contra 3 .NET frameworks diferentes:

* .NET Framework 2.0
* .NET Framework 3.0
* .NET Framework 3.5

También es muy fácil de usar gracias al desarrollo de hardware.

Además, integra el framework ASP.NET AJAX para el desarrollo de AJAX.

Visual studio.net 2010

El 12 de abril del 2010, Microsoft publica Visual Studio 2010, nombre clave Dev10, y .NET Framework versión 4. Compatible con Visual Basic.net, con una interfaz rediseñada, más sencilla y con soporte para diseño de aplicaciones en Windows 7.

Id alternativos a visual studio:

Para desarrollar en VB.NET existen algunas alternativas a Visual Studio, quizás la más notable sea SharpDevelop.

SharpDevelop es un entorno de programación integrado que permite programar en C# y en VB.NET.

Este es un entorno publicado bajo licencia LGPL, lo que implica que es libre y que disponemos del código fuente.

MonoDevelop es una implementación de SharpDevelop para programar usando Mono, una implementación libre de .NET que funciona en distintos sistemas operativos.

ADA:

Ada es un lenguaje de programación orientado a objetos y fuertemente tipado de forma estática que fue diseñado por Jean Ichbiah de CII Honeywell Bull por encargo del Departamento de Defensa de los Estados Unidos. Es un lenguaje multipropósito, orientado a objetos y concurrente, pudiendo llegar desde la facilidad de Pascal hasta la flexibilidad de C++.

Fue diseñado con la seguridad en mente y con una filosofía orientada a la reducción de errores comunes y difíciles de descubrir. Para ello se basa en un tipado muy fuerte y en chequeos en tiempo de ejecución (desactivables en beneficio del rendimiento). La sincronización de tareas se realiza mediante la primitiva rendezvous.

Ada se usa principalmente en entornos en los que se necesita una gran seguridad y fiabilidad como la defensa, la aeronáutica (Boeing o Airbus), la gestión del tráfico aéreo (como Indra en España) y la industria aeroespacial entre otros.

Historia:

El lenguaje fue diseñado bajo encargo del Departamento de Defensa de los Estados Unidos (DoD). Durante los años 1970, este departamento tenía proyectos en una infinidad de lenguajes y estaba gastando mucho dinero en software. Para solucionarlo se buscó un lenguaje único que cumpliese unas ciertas normas recogidas en el documento Steelman. Después de un estudio de los lenguajes existentes en la época se decidió que ninguno las cumplía totalmente, por lo que se hizo un concurso público al que se presentaron cuatro equipos, cuyas propuestas se nombraron con un color: Rojo (Intermetrics), Verde (CII Honeywell Bull), Azul (SofTEch) y Amarillo (SRI International). Finalmente, en mayo de 1979 se seleccionó la propuesta Verde diseñada por Jean Ichbiah de CII Honeywell Bull, y se le dio el nombre de Ada. Esta propuesta era un sucesor de un lenguaje anterior de este equipo llamado LIS y desarrollado durante los años 1970.

El nombre se eligió en conmemoración de lady Augusta Ada Byron (1815-1852) Condesa de Lovelace, hija del poeta Lord George Byron, a quien se considera la primera programadora de la Historia, por su colaboración y relación con Charles Babbage, creador de la máquina analítica.

El lenguaje se convirtió en un estándar de ANSI en 1983 (ANSI/MIL-STD 1815) y un estándar ISO en 1987 (ISO-8652:1987).

El DoD y los ministerios equivalentes de varios países de la OTAN exigían el uso de este lenguaje en los proyectos que contrataban (el Ada mandate). La obligatoriedad en el caso de Estados Unidos terminó en 1997, cuando el DoD comenzó a usar productos COTS (commercial off the shelf).

Compiladores:

Un compilador de Ada muy usado es GNAT, originalmente desarrollado por la Universidad de Nueva York bajo patrocinio del DoD. Está basado en la tecnología de GCC y es software libre. Actualmente está mantenido por AdaCore (antes llamada Ada Core Technologies), empresa que ofrece soporte y servicios sobre el compilador.

Características:

La sintaxis, inspirada en Pascal, es bastante legible incluso para personas que no conozcan el lenguaje. Es un lenguaje que no escatima en la longitud de las palabras clave, en la filosofía de que un programa se escribe una vez, se modifica decenas de veces y se lee miles de veces (legibilidad es más importante que rapidez de escritura).

Es indiferente el uso de mayúsculas y minúsculas en los identificadores y palabras claves, es decir es un lenguaje case-insensitive.

En Ada, todo el programa es un único procedimiento, que puede contener subprogramas (procedimientos o funciones).

Cada sentencia se cierra con end qué cerramos. Es un modo de evitar errores y facilitar la lectura. No es necesario hacerlo en el caso de subprogramas, aunque todos los manuales lo aconsejan y casi todos los programadores de Ada lo hacen.

El operador de asignación es:=, el de igualdad =. A los programadores de C y similares les puede confundir este rasgo inspirado en Pascal.

La sintaxis de atributos predefinidos es Objeto'Atributo (o Tipo'Atributo) (nota: esto sólo aplica a atributos predefinidos por el lenguaje, ya que no es el concepto de atributo típico de OOP).

Se distingue entre "procedimientos" (subrutinas que no devuelven ningún valor, pero pueden modificar sus parámetros) y "funciones" (subrutinas que devuelven un valor y no modifican los parámetros). Muchos lenguajes de programación no hacen esta distinción. Las funciones de Ada favorecen la seguridad al reducir los posibles efectos colaterales, pues no pueden tener parámetros in out.

Tipos de datos:

* Integer: un número entero.
* Float: un número decimal.
* Character: una letra o símbolo del teclado (también puede ser un número o varios, pero no se pueden hacer operaciones entre ellos).
* String: consiste en una cadena de caracteres.
* Array: un array o vector es una variable que agrupa varios elementos de cualquiera de los tipos previamente descritos.

Ejemplo de uso de Array:

type T\_vector is array(1..10) of integer; -- estamos definiendo la variable T\_vector como un tipo de dato que almacenará 10 elementos de tipo entero.

Vector1:T\_vector; -- Estamos declarando una variable llamada "Vector1" del tipo "T\_vector" previamente definido.

Para acceder a cada uno de los huecos se accederia de la siguiente manera:

Vector1(4):= 5; --El hueco 4 de la variable Vector1 tiene el valor de 5.

ALGOL:

Se denomina ALGOL (o Algol) a un lenguaje de programación. La voz es un acrónimo de las palabras inglesas Algorithmic Language (lenguaje algorítmico).

Fue muy popular en las universidades durante los años 60, pero no llegó a cuajar como lenguaje de utilización comercial.

Sin embargo, Algol influyó profundamente en varios lenguajes posteriores que sí alcanzaron gran difusión, como Pascal, C y Ada.

Hacia 1965 dos corrientes se distinguieron sobre el tema de un sucesor para Algol. Como resultado se definieron los lenguajes Algol W que es un lenguaje minimalista, rápidamente implementado y distribuido y, por otra parte, Algol 68 que para la época está en la frontera entre un lenguaje para programar en él y un lenguaje para investigar sobre él.

ALGOL W:

Lenguaje elaborado diseñado por Niklaus Wirth y Tony Hoare a partir de los trabajos del grupo ALGOL de la IFIP. Se trata de un lenguaje conciso, simple de implementar, que evita todos los defectos conocidos del lenguaje Algol e incluye sus propias características adicionales.

Sin embargo, el grupo Algol no lo adoptó como sucesor de Algol prefiriendo en su lugar al que terminó siendo Algol 68. Algol W fue utilizado por gran cantidad de usuarios y sembró el camino para el nacimiento del lenguaje Pascal.

Entre las características del lenguaje se destacan: Aritmética de doble precisión, números complejos, Strings y estructuras de datos dinámicas, evaluación por valor, pasaje de parámetros por valor, valor resultado o resultado.

ALGOL 68:

La definición del lenguaje fue presentada en la reunión del comité ALGOL de la IFIP en 1965. Luego de varios años de revisión del diseño se llegó a una versión definitiva en 1968. Al principal autor es Adriaan van Wijngaarden.

Los objetivos principales de ALGOL 68 son el permitir comunicar algoritmos, el permitir una eficiente ejecución de los mismos en diferentes arquitecturas y el de servir como herramienta para la enseñanza.

Una característica interesante de ALGOL 68 es que su semántica fue definida formalmente antes de ser implementado en base al formalismo llamado gramáticas de dos niveles.

BASIC:

En la programación de computadoras, siglas de Beginner's All-purpose Symbolic Instruction Code1 (Código simbólico de instrucciones de propósito general para principiantes en español), es una familia de lenguajes de programación de alto nivel. El BASIC original, el Dartmouth BASIC, fue diseñado en 1964 por John George Kemeny y Thomas Eugene Kurtz en el Dartmouth College en Nuevo Hampshire, Estados Unidos, como un medio para facilitar la programación en ordenadores a estudiantes (y profesores) que no fueran de ciencias.

De aquella, casi todo el uso de los ordenadores requería codificar software hecho a medida, con lo cual quedaba restringida a personas con formación como científicos y matemáticos. BASIC originalmente fue desarrollado como una herramienta de enseñanza. El lenguaje y sus variantes llegaron a estar ampliamente disponibles en los microcomputadores a finales de los años 1970 y en los años 1980. BASIC sigue siendo popular a día de hoy en un puñado de dialectos altamente modificados, y en nuevos lenguajes, influenciados por BASIC tales como Microsoft Visual Basic o Gambas en GNU/Linux. En el año 2006, el 59% de los desarrolladores para la plataforma .NET usaban Visual Basic .NET como su único lenguaje.

Es importante aclarar que una de las grandes desventajas en comparación con lenguajes como COBOL, RPG es que no maneja como parte integrada el acceso a archivos directos o RANDOM. Se tenía que apoyar en una aplicación externa para poder acceder de manera directa un registro de algún archivo.

Antecedentes:

A principios de la década de 1960, las computadoras eran sumamente caras y se utilizaban únicamente para propósitos especiales, ejecutando "una única tarea" a la vez. Sin embargo, durante el transcurso de esa década, los precios bajaron al punto que incluso algunas pequeñas empresas ya podían costearlas. La velocidad de proceso de las máquinas se incrementó al grado que a menudo quedaban demasiado tiempo ociosas, porque no había suficientes tareas para ellas. Todo ello fue debido a la rápida evolución del hardware. Los lenguajes de programación de aquellos tiempos estaban diseñados para propósitos específicos, como las máquinas en las que eran ejecutados; por ejemplo, para desarrollar aplicaciones cálculo o procesamiento de fórmulas se diseñó el lenguaje FORTRAN, en tanto que para la programación en administración o gestión de información se desarrolló específicamente el COBOL.

A fin de incrementar el rendimiento y amortizar mejor los costos (por reducción del tiempo ocioso del procesador), y siendo que ya la velocidad de las máquinas comenzó a permitirlo, se propuso la idea de ejecutar más de una tarea "simultáneamente".

Fue así que surgió el concepto de sistema de tiempo compartido, que comenzó a ganar mucha popularidad. En sistemas de ese tipo, el tiempo de procesamiento del procesador central se dividía, y a cada usuario se le otorgaba secuencial y cíclicamente una pequeña porción o "cuota" de tiempo de proceso. Las máquinas eran lo suficientemente rápidas como para provocar en los usuarios la ilusión de que disponían de la funcionalidad de la máquina todo el tiempo para ellos ("seudo-simultaneidad" de procesos). Esa distribución del tiempo de cómputo entre los usuarios redujo considerablemente el costo de la computación, ya que una sola máquina podía ser compartida por numerosos usuarios.

Inicios de BASIC

El lenguaje BASIC fue inventado en 1964 por John George Kemeny (1926-1992) y Thomas Eugene Kurtz (1928-) en el Dartmouth College. En años subsiguientes, mientras que otros dialectos de BASIC aparecían, el BASIC original de Kemeny y Kurtz fue conocido como Dartmouth BASIC.

BASIC fue diseñado para permitir a los estudiantes escribir programas usando terminales de un computador en tiempo compartido. Estaba pensado para reducir notablemente la complejidad de los otros lenguajes del momento, con uno diseñado específicamente para la clase de usuarios que los sistemas de tiempo compartido permitían: un usuario más sencillo, fuera del área de las ciencias de la computación, a quien no le interesaba tanto la velocidad, solo el hecho de ser capaz de programar y usar la máquina sin demasiadas complicaciones. Los diseñadores del lenguaje también querían que permaneciera en el dominio público, lo cual contribuyó a que se diseminara rápidamente.

Los ocho principios que rigieron el diseño de BASIC fueron:

1. Ser fácil de usar para los principiantes.
2. Ser un lenguaje de propósito general (no orientado).
3. Permitir a los expertos añadir características avanzadas, conservando simple el lenguaje para los principiantes.
4. Ser interactivo en todos los casos.
5. Proveer mensajes de errores claros y amigables.
6. Responder rápido en los programas pequeños en general.
7. No requerir un conocimiento del hardware de la computadora.
8. Proteger al usuario del sistema operativo.

El lenguaje fue en parte basado en FORTRAN II y otra parte en Algol 60, con adiciones para hacerlo apropiado en sistemas de tiempo compartido y con elementos que facilitaran la operación aritmética de matrices. BASIC fue implementado por primera vez para la mainframe GE-265, máquina que soportaba múltiples terminales.

En 1968 Edsger Dijkstra publicó una carta con una famosa crítica3 en la que consideraba que los lenguajes de programación que usaban sentencias GOTO para estructurar un programa eran nocivos para la productividad del programador, y para la calidad del código resultante. En este artículo no se menciona a ningún lenguaje de programación en particular; únicamente se indica que el uso excesivo de GOTO en lenguajes de alto nivel es algo no recomendable, y proporciona las razones técnicas por las que esto es así. Pero sí se observó, desde sus inicios, una marcada tendencia de los programadores a utilizar excesivamente el GOTO en BASIC, máxime en los noveles; hecho éste que fue decreciendo con la incorporación al lenguaje de otros recursos, tales como subrutinas parametradas, y posteriormente con la aparición de técnicas de programación estructurada.

Contrario a la creencia popular, BASIC inicialmente no era un lenguaje interpretado sino compilado. Casi inmediatamente después de su lanzamiento, los profesionales de computación comenzaron a alegar que BASIC era muy lento y simple. Tal argumento, hasta no hace mucho, fue un tema recurrente en la industria de las computadoras.

Aun así, BASIC se extendió hacia muchas máquinas y plataformas, y se popularizó moderadamente en las minicomputadoras como las de la serie DEC PDP y la Data General Nova. En estos casos, el lenguaje era implementado como intérprete, en vez de un compilador, o alternativamente, en ambas formas de funcionamiento.

Crecimiento

Sin embargo, fue con la introducción de la microcomputadora Altair 8800 en 1975 que BASIC se extendió ampliamente. La mayoría de los lenguajes de programación eran demasiado grandes para ser albergados por las pequeñas memorias de que disponían la mayor parte de las máquinas en esa época; y con el lento almacenamiento que permitía la cinta de papel, y más tarde la cinta de audiocasete (los discos magnéticos aún no existían), y la falta de editores de texto adecuados, un lenguaje pequeño como BASIC resultaba una buena opción. Uno de los primeros en aparecer fue Tiny BASIC, una implementación simple de BASIC escrita originalmente por el Dr. Li-Chen Wang, y portada más tarde a la máquina Altair por Dennis Allison, a petición de Bob Albrecht (quien después fundó Dr. Dobb's Journal (DDJ)). El diseño de Tiny BASIC y el código fuente completo fue publicado en DDJ en 1976.

En 1975 Microsoft (entonces formado por dos personas: Bill Gates y Paul Allen) lanzó el Altair BASIC. Luego comenzaron a aparecer bajo licencia versiones para otras plataformas, y millones de copias y variantes pronto estarían en uso. BASIC se convirtió en uno de los lenguajes estándar en el Apple II. En 1979 Microsoft estaba negociando con varios vendedores de microcomputadores, incluyendo IBM, para licenciar un intérprete de BASIC para sus ordenadores. Se incluyó una versión en las memorias ROM de los PC IBM para los equipos sin discos. Los equipos que disponían de una unidad de disquete, el BASIC era iniciado automáticamente siempre que no se colocara ningún disquete de arranque como sistema operativo.

Las nuevas compañías intentaban seguir los pasos del éxito de Altair: IMSAI, North Star, y Apple, creando la revolución de la computadora casera.

BASIC se convirtió en una característica estándar para casi todas las computadoras hogareñas; la mayoría venía con un sistema operativo básico e intérprete de BASIC, todo alojado en una ROM (algo hecho por primera vez en la Commodore PET en 1977). Pronto habría muchos millones de computadores alrededor del mundo ejecutando BASIC, un número mucho más grande que el de todos los usuarios de otros lenguajes juntos. Muchos programas, especialmente los de la Apple II e IBM PC, dependían de la presencia del intérprete de BASIC de Microsoft y no podían ejecutarse sin este; por lo que Microsoft usó la licencia de copyright en los intérpretes de BASIC para influir en las negociaciones con los vendedores de computadores.

El BASIC fue también el lenguaje preinstalado en los computadores hogareños europeos de la década de los 80 como el ZX Spectrum (Sinclair BASIC), Amstrad CPC (Locomotive BASIC), MSX (MSX BASIC), el Commodore 64 y 128 (Basic 2.0, Basic 7.0, Simons' Basic), los Commodore Amiga (AmigaBASIC) o la familia Atari de 8 bits (Atari BASIC) y en los computadores hogareños japoneses NEC PC-8801 y NEC PC-9801 (N88-BASIC), haciendo muchas veces la función de intérprete y sistema operativo primitivo, ya que venían implementados ambos en ROM. Texas Instruments incorporó su propia versión en sus microcomputadoras, tal como la TI99/4A, y también con una versión extendida en una ROM externa o cartuchos (TI-Basic y TI Extended Basic).

Desenlace

En este período se crearon versiones de BASIC nuevas y más poderosas. Microsoft vendió varias versiones de BASIC para MS-DOS/PC-DOS, incluyendo BASICA, GW-BASIC (una versión compatible con BASICA que no necesitaba la ROM de IBM), y Quick BASIC. El fabricante de Turbo Pascal, Borland, publicó Turbo BASIC 1.0 en 1985 (versiones sucesoras aún se venden bajo el nombre de PowerBASIC por otra compañía). Aparecieron varias extensiones de BASIC para computadores caseros, típicamente con capacidad para gráficos, sonido, y comandos DOS, así como facilidades para Programación estructurada. Hubo lenguajes que usaron la sintaxis de BASIC como base para otros sistemas totalmente diferentes, por ejemplo, GRASS.

Sin embargo, a finales de la década de 1980 las computadoras nuevas eran mucho más complejas, e incluían características (como la Interfaz gráfica de usuario) que hacían a BASIC menos apropiado para programarlas. Al mismo tiempo las computadoras progresaban de ser interés para aficionados a herramientas usadas principalmente para ejecutar aplicaciones escritas por otros, y la programación en sí se fue haciendo menos importante para una creciente mayoría de usuarios. BASIC comenzó a desvanecerse, aunque numerosas versiones aún estaban disponibles.

Una de las más poderosas fue el denominado Locomotive BASIC 2 diseñado para el entorno gráfico GEM. Esta nueva versión del lenguaje permitía crear aplicaciones con interfaces gráficas dotadas de ventanas, menús y diferentes tipos de gráficos estadísticos.

La suerte de BASIC dio un giro nuevamente con la introducción de Visual Basic de Microsoft. Si bien este lenguaje utiliza prácticamente todas las palabras clave (sentencias, estructuras de control y funciones intrínsecas) y forma de manejo y tipo de datos que versiones BASIC anteriores (DOS); VB es abismalmente más potente y evolucionado; y se ha convertido en uno de los lenguajes más utilizados en la plataforma Windows; se estima que entre el 70 y el 80% del total de aplicaciones comerciales son programadas en VB. A partir de 2002, y con la introducción de la plataforma .NET Framework de Microsoft, Visual Basic comienza a utilizar el paradigma "orientado a objetos", aumentando la potencia del lenguaje y haciéndolo multiplataforma. Visual Basic for Applications (VBA) fue añadido a Microsoft Excel 5.0 en 1993 y al resto de la línea de productos de Microsoft Office en 1997. Windows 98 incluyó un intérprete de VBScript. La versión más reciente de Visual Basic es llamada VB.NET. Por otra parte, también existe la variante OpenOffice.org Basic menos poderosa pero similar a VBA de Microsoft.

Síntaxis

La sintaxis mínima de BASIC solo necesita los comandos LET, INPUT, PRINT, IF y GOTO. Un intérprete que ejecuta programas con esta sintaxis mínima no necesita una pila.

Algunas de las primeras implementaciones eran así de simples. Si se le agrega una pila, se pueden agregar también ciclos FOR anidados y el comando GOSUB. Un intérprete de BASIC con estas características necesita que el código tenga números de línea.

Los números de línea fueron un aspecto muy distintivo del BASIC clásico. Sin embargo, el uso de números de línea tiene la desventaja de requerir que el programador estime cuántas líneas ocupará la parte del programa que escribe. Este requerimiento se cumple generalmente incrementando los números de línea en un intervalo regular, como 10, pero esto lleva a problemas a la hora que el código después agregado exceda el espacio disponible entre las líneas originales. Para aliviar este problema de los primeros intérpretes de BASIC, los usuarios expertos pronto escribieron sus propios programas utilitarios para renumerar sus programas, después del ingreso inicial. Más tarde aparecieron intérpretes de BASIC que incluían un comando específico RENUMBER, el que permitía renumerar rápidamente (y las veces que se quisiera) todo el código nuevamente, con cualquier intervalo entre líneas indicado y a partir de un número entero dado; eliminando así el principal problema de la numeración de líneas obligatoria.

En los dialectos modernos de BASIC MIUN ya no es necesario incluir números de línea (aunque son permitidos), y la mayoría (o todos) han añadido control de flujo estructurado y los constructores de declaración de datos similares a los de otros lenguajes, tales como C y Pascal:

do

loop

while

until

exit

on... goto

Casi todos los dialectos de BASIC incluyen el comando REM (remark), que posteriormente fue sustituido por el símbolo ´ (apóstrofo o comilla simple). Es un comando no ejecutable, se utiliza a los fines de incluir líneas y notas aclaratorias en el código fuente, para la mejor comprensión (y documentación) del programa.

Variantes recientes como Visual Basic han introducido algunas características orientadas a objetos, y hasta herencia en la última versión. La administración de memoria es más fácil que con muchos otros lenguajes de programación procedurales por el uso de un Recolector de basura (y a costas de la velocidad de ejecución).

Procedimiento

BASIC no tiene una biblioteca externa estándar como otros lenguajes como C. En cambio, el intérprete (o compilador) contiene una biblioteca incorporada de procedimientos intrínsecos. Estos procedimientos incluyen la mayoría de las herramientas que un programador necesita para aprender a programar y escribir aplicaciones sencillas, así como funciones para realizar cálculos matemáticos, manejar cadenas, entrada desde la consola, gráficos y manipulación de archivos.

Viejos dialectos de BASIC no permitían a los programadores escribir sus propios procedimientos. Los programadores en cambio debían escribir sus programas con un gran número de enunciados GOTO para hacer las derivaciones de flujo y retorno del programa. Esto podía producir un código fuente muy confuso (la mayoría de las veces era así), comúnmente conocido como Código espagueti; el cual era sumamente difícil de mantener, mucho menos por programadores ajenos al desarrollo del software.

Con la inclusión posterior de enunciados GOSUB (Go-Subroutine) se ramificaba el programa a especies de subrutinas, sin parámetros o variables locales. Ellas proveen una forma de implementar una suerte de procedimientos (realmente no lo son, sencillamente es un "salto y retorno") y estructurar más el programa, evitando bastante la utilización de la dañina sentencia GOTO.

La mayoría de las versiones de BASIC más modernas, como Microsoft QuickBASIC (1985-1988) y BASIC PDS (Profesional Developmen System - 1990) añadieron soporte completo para subrutinas, funciones y programación estructurada. Esta es otra área donde BASIC difiere de muchos lenguajes de programación. Sin embargo, la primitiva GOSUB se ha mantenido hasta las versiones actuales, por razones compatibilidad.

BASIC, como Pascal, hace una distinción entre un procedimiento que no devuelve un valor (llamado subrutina) y un procedimiento que lo hace (llamado función). Muchos otros lenguajes (como C) no hacen esa distinción y consideran todo como una función (algunas que devuelven un valor “void” [vacío]).

Mientras que las funciones que devuelven un valor son una adición relativamente reciente a los dialectos de BASIC, muchos de los primeros sistemas soportaban la definición de funciones matemáticas en línea, con DEF FN (“DEFine FunctioN” [DEFinir FuncióN]). El Dartmouth BASIC original también soportaba funciones al estilo de Algol, así como subrutinas desde sus primeros tiempos.

Tipos de datos

BASIC es reconocido por tener muy buenas funciones para manipular cadenas de caracteres. Los primeros dialectos ya tenían un juego de funciones fundamentales (LEFT$, MID$, RIGHT$) para extraer y/o reemplazar subcadenas fácilmente. Como las cadenas son utilizadas en aplicaciones diarias, esta era una ventaja considerable sobre otros lenguajes al momento de su introducción.

El Dartmouth BASIC original soportaba únicamente datos de tipo numérico y cadenas. No había un tipo entero. Todas las variables numéricas eran de coma flotante. Las cadenas eran de tamaño dinámico. Soportaba arreglos de ambos, números y cadenas, en una o dos dimensiones.

Cada dialecto moderno de BASIC posee al menos los tipos de datos numérico y cadena. Estos tipos de datos se pueden distinguir usando un posfijo: los identificadores de cadenas terminan con $ (signo dólar, ejemplo la variable NOMBRE$), mientras que los numéricos sencillamente no llevan posfijo; a menos que se requiera indicar y forzar explícitamente qué tipo de numérico es, por ejemplo, A% es entero, A! es real simple precisión y A# es real doble precisión.

En BASIC las variables no necesitan forzosamente ser declaradas antes de usarse, excepto los arreglos de más de 10 elementos; aunque versiones BASIC relativamente modernas poseen la opción (considerada buena práctica de programación) para obligar al programador a declarar toda variable antes de su uso (una directiva como OPTION EXPLICIT). La declaración de variables en BASIC se hace usando la palabra clave DIM.

Muchos dialectos también soportan tipos numéricos adicionales, como enteros de 16 y 32 bits (simple y long, respectivamente), además de sus números de coma flotante. Adicionalmente algunos permiten la utilización de tipos de datos definidos por el usuario, similar a los "records" de Pascal, o las "structs" de C.

Versiones modernas de BASIC (como VBA) soportan una gran variedad de tipos de datos primitivos (o intrínsecos), además de los definidos por el usuario.

La mayoría de los dialectos de BASIC soporta arreglos en todos sus tipos de datos; es común también el soporte para arreglos mulidimensionales

Dependiendo del dialecto de BASIC y del uso del enunciado OPTION BASE, el primer índice de los arreglos que se declaren será 1, por defecto es cero.

En los ejemplos anteriores, si no se declara previamente "OPTION BASE 1", el primero es un arreglo en dos dimensiones de enteros 16 bits, con índices que van desde 0 hasta 100 (matriz de 101 x 101 elementos); en tanto que el segundo es un arreglo de enteros en una sola dimensión, de 0 a 30 (vector de 31 elementos). Observar que las dos formas de declaración de enteros son equivalentes, explicitándolo o con el uso del posfijo %. Análogamente para cadenas o string de caracteres, que en este caso además son de longitud variable (dinámicas, por defecto).

Disponibilidad y variantes:

Dependiendo del dialecto de BASIC y del uso del enunciado OPTION BASE, el primer índice de los arreglos que se declaren será , por defecto es cero.

En los ejemplos anteriores, si no se declara previamente "OPTION BASE 1", el primero es un arreglo en dos dimensiones de enteros 16 bits, con índices que van desde 0 hasta 100 (matriz de 101 x 101 elementos); en tanto que el segundo es un arreglo de enteros en una sola dimensión, de 0 a 30 (vector de 31 elementos). Observar que las dos formas de declaración de enteros son equivalentes, explicitándolo o con el uso del posfijo %. Análogamente para cadenas o string de caracteres, que en este caso además son de longitud variable (dinámicas, por defecto).

Disponibilidad y variantes del lenguaje

BASIC está disponible para casi todas las plataformas y sistemas operativos existentes. Una implementación gratuita que cumple con estándares y es multiplataforma es Bywater BASIC (bwBASIC).

El intérprete está escrito en C y viene bajo la licencia GNU. Está diseñado para interfaz de texto o consola (no gráfica), no incluye soporte para crear interfaces gráficas de usuario (GUI's, Graphical User Interface). Hay un BASIC gratuito que, si incluye soporte para GUI, es similar a Visual Basic y se ejecuta en Windows y GNU/Linux, es Phoenix Object BASIC.

Las versiones de intérpretes/compiladores más conocidos son la línea de productos Quick BASIC y QBASIC, este último es solo intérprete, ambos son de Microsoft. En la actualidad lo es el moderno Visual BASIC, que Microsoft ha tratado de mantener al menos mínimamente compatible con incluso las primeras versiones de sus BASIC (en realidad es escasamente compatible), si bien existe FreeBASIC que es un compilador libre, compatible en sintaxis con QBASIC/QuickBASIC.

Otras versiones comerciales incluyen PowerBASIC de PowerBASIC, PureBasic de Fantaisie Software, así como TrueBASIC de TrueBASIC, que cumple con los últimos estándares oficiales de BASIC. (True BASIC Inc. fue fundada por los creadores originales de Dartmouth BASIC.)

REALbasic es una variante disponible para Mac OS Classic, Mac OS X, Microsoft Windows y GNU/Linux, comercializada por los actuales propietarios de Rapid-Q, otra implementación inicialmente libre de BASIC actualmente abandonada. Una versión de un dialecto simple de BASIC para la parrot virtual machine, muestra cómo se implementa un intérprete de BASIC en un lenguaje similar al ensamblador. SmallBASIC es un dialecto que ejecuta en muchas plataformas (Win32, DOS, GNU/Linux y PalmOS) y viene bajo la licencia GNU (GPL).

Existen muchas implementaciones de BASIC freeware o GNU, como BCX, YaBasic, HBasic, XBasic, Gambas o Just BASIC, entre otras.

***C#***

C#1 (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.2

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Historia

Durante el desarrollo de la plataforma .NET, las bibliotecas de clases fueron escritas originalmente usando un sistema de código gestionado llamado Simple Managed C (SMC). En enero de 1999, Anders Hejlsberg formó un equipo con la misión de desarrollar un nuevo lenguaje de programación llamado Cool (Lenguaje C orientado a objetos). Este nombre tuvo que ser cambiado debido a problemas de marca, pasando a llamarse C#.3 La biblioteca de clases de la plataforma .NET fue migrada entonces al nuevo lenguaje.

Hejlsberg lideró el proyecto de desarrollo de C#. Anteriormente, ya había participado en el desarrollo de otros lenguajes como Turbo Pascal, Delphi y J++.

C# contiene veinte categorías generales de tipos de datos integrados: tipos de valor y tipos de referencia. El término tipo de valor indica que esos tipos contienen directamente sus valores.

Los tipos de coma flotante pueden representar números con componentes fraccionales. Existen dos clases de tipos de coma flotante: float y double. El tipo double es el más utilizado porque muchas funciones matemáticas de la biblioteca de clases de C# usan valores double. Quizá, el tipo de coma flotante más interesante de C# es decimal, dirigido al uso de cálculos monetarios. La aritmética de coma flotante normal está sujeta a una variedad de errores de redondeo cuando se aplica a valores decimales. El tipo decimal elimina estos errores y puede representar hasta 28 lugares decimales.

Los caracteres en C# no tienen un tamaño de 8 bits como en muchos otros lenguajes de programación, sino que usan un tamaño de 16 bits. Este tipo de dato se llama char y utiliza la codificación Unicode. No existen conversiones automáticas de tipo entero a char.

Para los tipos de datos lógicos no existen conversiones automáticas de tipo entero a bool.

Literales:

En ocasiones, resulta más sencillo usar un sistema numérico en base 16 en lugar de 10, para tal caso C# permite especificar números enteros en formato hexadecimal, y se define anteponiendo 0x, por ejemplo: 0xFF, que equivale a 255 en decimal.

C# tiene caracteres denominados secuencias de escape para facilitar la escritura con el teclado de símbolos que carecen de representación visual.

C#, al igual que C++, define un tipo de cadena de caracteres. Dentro de la cadena de caracteres se pueden usar secuencias de escape. Una cadena de caracteres puede iniciarse con el símbolo @ seguido por una cadena entre comillas ("), en tal caso, las secuencias de escape no tienen efecto, y además la cadena puede ocupar dos o más líneas.

Fortran

Fortran (previamente FORTRAN,1 contracción del inglés The IBM Mathematical Formula Translating System) es un lenguaje de programación de alto nivel de propósito general,2 procedimental3 e imperativo, que está especialmente adaptado al cálculo numérico y a la computación científica. Desarrollado originalmente por IBM en 1957 para el equipo IBM 704, y usado para aplicaciones científicas y de ingeniería, el FORTRAN vino a dominar esta área de la programación desde el principio y ha estado en uso continuo por más de medio siglo en áreas de cómputo intensivo tales como la predicción numérica del tiempo, análisis de elementos finitos, dinámica de fluidos computacional (CFD), física computacional y química computacional. Es uno de los lenguajes más populares en el área de la computación de alto rendimiento y es el lenguaje usado para programas que evalúan el desempeño (benchmark) y el ranking de los supercomputadores más rápidos del mundo.4

El FORTRAN abarca un linaje de versiones, cada una de las cuales evolucionó para añadir extensiones al lenguaje mientras que usualmente retenía compatibilidad con las versiones previas. Versiones sucesivas han añadido soporte para procesamiento de datos basados en caracteres (FORTRAN 77), programación de arreglos, programación modular y programación orientada a objetos (Fortran 90/95), y programación genérica (Fortran 2003).

Historia:

A finales de 1953, John W. Backus sometió una propuesta a sus superiores en IBM para desarrollar una alternativa más práctica al lenguaje ensamblador para programar el computador central IBM 704. El histórico equipo FORTRAN de Backus consistió en los programadores Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt y David Sayre.5

A mediados de 1954 fue terminada una especificación del borrador para el IBM Mathematical Formula Translating System. El primer manual de FORTRAN apareció en octubre de 1956, porque los clientes eran reacios a usar un lenguaje de programación de alto nivel a menos que su compilador pudiera generar código cuyo desempeño fuera comparable al de un código hecho a mano en lenguaje ensamblador.

Mientras que la comunidad era escéptica en que este nuevo a mano, este redujo por un factor de 20 al número de sentencias de programación necesarias para operar una máquina, y rápidamente ganó aceptación. Durante una entrevista en 1979 con Think, la revista de los empleados de IBM, el creador, John Backus, dijo, "Mucho de mi trabajo ha venido de ser perezoso. No me gustaba escribir programas, y por eso, cuando estaba trabajando en el IBM 701 escribiendo programas para computar trayectorias de misiles, comencé el trabajo sobre un sistema de programación para hacer más fácil escribir programas".6

El lenguaje fue ampliamente adoptado por los científicos para escribir programas numéricamente intensivos, que incentivó a los escritores de compiladores a producir compiladores que pudieran generar un código más rápido y más eficiente. La inclusión en el lenguaje de un tipo de datos y de la aritmética de números complejos amplió la gama de aplicaciones para las cuales el lenguaje se adaptaba especialmente e hizo al FORTRAN especialmente adecuado para aplicaciones técnicas tales como la ingeniería eléctrica.

Hacia 1960, las versiones de FORTRAN estaban disponibles para los computadores IBM 709, 650, 1620, y 7090. Significativamente, la cada vez mayor popularidad del FORTRAN estimuló a fabricantes de computadores de la competencia a proporcionar compiladores FORTRAN para sus máquinas, así que por 1963 existían más de 40 compiladores FORTRAN. Por estas razones, el FORTRAN es considerado ser el primer lenguaje de programación ampliamente usado soportado a través de una variedad de arquitecturas de computador.

El desarrollo del FORTRAN fue paralelo a la temprana evolución de la tecnología del compilador. De hecho, muchos avances en la teoría y el diseño de compiladores fueron motivados específicamente por la necesidad de generar código eficiente para los programas en FORTRAN.

Versiones:

* FORTRAN II
* FORTRAN IV
* FORTRAN 66
* FORTRAN 77
* Fortran 90
* Fortran 95
* Fortran 2003
* Fortran 2008

Principales:

El lenguaje fue diseñado teniendo en cuenta que los programas serían escritos en tarjetas perforadas de 80 columnas. Así, por ejemplo, las líneas debían ser numeradas y la única alteración posible en el orden de ejecución era producida con la instrucción goto. Estas características han evolucionado de versión en versión. Las actuales contienen subprogramas, recursión y una variada gama de estructuras de control.

Ventajas e incovenientes de su sintaxis:

Lo que fue la primera tentativa de proyección de un lenguaje de programación de alto nivel, tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y errores en la escritura de solo un carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes. Algunas de las versiones anteriores no poseían facilidades que son consideradas muy útiles, tal como la asignación dinámica de memoria.

Se debe tener en cuenta que la sintaxis de Fortran fue orientada para el uso en trabajos numéricos y científicos. Muchas de sus deficiencias han sido abordadas en revisiones recientes del lenguaje. Por ejemplo, Fortran 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos. Esto no solo mejora mucho la lectura del programa, sino que además aporta información útil al compilador.

Por estas razones Fortran no es casi usado fuera de los campos científicos y del análisis numérico, pero permanece como el lenguaje preferido para desarrollar aplicaciones de computación numérica de alto rendimiento.

Java:

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.1 2

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales, y librerías de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Historia:

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en el año 1991. El equipo (Green Team), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente Oak (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse Green tras descubrir que Oak era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a Java.

Es frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus diseñadores: James Gosling, Arthur Van Hoff, y Andy Bechtolsheim. Otros abogan por el siguiente acrónimo, Just Another Vague Acronym ("sólo otro acrónimo ambiguo más"). La hipótesis que más fuerza tiene es la de que Java debe su nombre a un tipo de café disponible en la cafetería cercana, de ahí que el icono de java sea una taza de café caliente. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el número mágico) de los archivos.class que genera el compilador, son en hexadecimal, 0xCAFEBABE. A pesar de todas estas teorías, el nombre fue sacado al parecer de una lista aleatoria de palabras.3

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gaga, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, de que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. [1] Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era Write Once, Run Anywhere (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.4

Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa Java Specification Requests (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la Java Language Specification (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

LIPS

El Lisp (o LISP) es una familia de lenguajes de programación de computadora de tipo multiparadigma con una larga historia y un uso intensivo de paréntesis en su sintaxis.

Especificado originalmente en 1958 por John McCarthy y sus colaboradores en el Instituto Tecnológico de Massachusetts, el Lisp es el segundo lenguajes de programación de alto nivel de mayor antigüedad entre los que continúan teniendo un uso extendido en la actualidad; únicamente FORTRAN es anterior.

Al igual que el FORTRAN, el Lisp ha cambiado mucho desde sus comienzos, y han existido un gran número de dialectos en su historia.

Hoy, los dialectos Lisp de propósito general más ampliamente conocidos son el Common Lisp y el Scheme.

El Lisp fue creado originalmente como una notación matemática práctica para los programas de computadora, basada en el cálculo lambda de Alonzo Church. Se convirtió rápidamente en el lenguaje de programación favorito en la investigación de la inteligencia artificial (AI). Como lenguajes de programación precursor, el Lisp fue pionero en muchas ideas en ciencias de la computación, incluyendo las estructuras de datos de árbol, el manejo de almacenamiento automático, tipos dinámicos, y el compilador auto contenido.

El nombre LISP deriva del "LISt Processing" (Proceso de LIStas). Las listas encadenadas son una de las estructuras de datos importantes del Lisp, y el código fuente del Lisp en sí mismo está compuesto de listas. Como resultado, los programas de Lisp pueden manipular el código fuente como una estructura de datos, dando lugar a los macro sistemas que permiten a los programadores crear una nueva sintaxis de lenguajes de programación de dominio específico empotrados en el Lisp.

La intercambiabilidad del código y los datos también da a Lisp su instantáneamente reconocible sintaxis. Todo el código del programa es escrito como expresiones S, o listas entre paréntesis. Una llamada de función o una forma sintáctica es escrita como una lista, con la función o el nombre del operador en primer lugar, y los argumentos a continuación; por ejemplo, una función f que toma tres argumentos puede ser llamada usando (f x y z).

Historia:

El Lisp fue inventado por John McCarthy en 1958 mientras estaba en el Instituto Tecnológico de Massachusetts (MIT). McCarthy publicó su diseño en 1960 en un artículo de Communications of the ACM titulado "funciones recursivas de expresiones simbólicas y su cómputo por la máquina, parte I"1 (la "parte II" nunca fue publicada). Allí mostró que con algunos operadores simples y una notación para las funciones, uno puede construir un lenguaje turing completo para los algoritmos.

Desde 1955 ó 1956, el Information Processing Language fue el primer lenguaje de AI, y ya había incluido muchos de los conceptos, tales como proceso por lista y recursión, que vinieron a ser usados en el Lisp.

La notación original de McCarthy usaba "expresiones M" en corchetes que serían traducidas a expresiones S. Como un ejemplo, la expresión M car[cons[A,B]] es equivalente a la expresión S (car (cons A B)). Una vez el Lisp fue implementado, los programadores rápidamente eligieron usar expresiones S, y las expresiones M fueron abandonadas. las expresiones M emergieron otra vez con los intentos efímeros del MLISP2 de Horace Enea y el CGOL de Vaughan Pratt.

El Lisp fue implementado primero por Steve Russel en un computador IBM 704. Russell había leído el artículo de McCarthy, y se dio cuenta (para la sorpresa de McCarthy) que la función eval del Lisp podía ser implementada en código de máquina. El resultado fue un intérprete de Lisp funcional que podía ser usado para correr programas Lisp, o más correctamente, "evaluar expresiones Lisp".

Dos rutinas de lenguaje ensamblador para el IBM 704 se convirtieron en las operaciones primitivas para descomponer listas: car (contenido del registro de dirección) y cdr (contenido del registro del decremento). Los dialectos del Lisp todavía usan el car y cdr (pronunciado /ˈkɑr/ y /ˈkʊdər/) para las operaciones que retornan el primer elemento y el resto de la lista respectivamente.

El primer compilador completo de Lisp, escrito en Lisp, fue implementado en 1962 por Tim Hart y Mike Levin en el MIT.3 Este compilador introdujo el modelo Lisp de compilación incremental, en el cual las funciones compiladas e interpretadas pueden entremezclarse libremente. El lenguaje en los memos de Hart y Levin es mucho más cercano al estilo moderno del Lisp que el anterior código de McCarthy.

Genealogía:

Sobre su historia de cincuenta años, el Lisp ha producido muchas variaciones en el tema base de un lenguaje de expresión S. Por otra parte, cada dialecto dado puede tener varias implementaciones, por ejemplo, hay más de una docena de implementaciones del Common Lisp.

Las diferencias entre los dialectos pueden ser muy visibles, por ejemplo, el Common Lisp y el Scheme usan diferentes palabras clave para definir funciones. Dentro de un dialecto que está estandarizado, sin embargo, las implementaciones conformadas soportan el mismo lenguaje base, pero con diferentes extensiones y bibliotecas.

Relación con inteligencia artificial:

Desde su inicio, el Lisp estaba estrechamente relacionado con la comunidad de investigación de la inteligencia artificial, especialmente en sistemas PDP-10.4 El Lisp fue usado como la implementación del lenguaje de programación Micro Planner que fue la fundación para el famoso sistema de AI SHRDLU. En los años 1970, a medida que la investigación del AI engendró descendientes comerciales, el desempeño de los sistemas Lisp existentes se convirtió en un problema creciente.

El Lisp era un sistema difícil de implementar con las técnicas de compilador y hardware común de los años 1970. Las rutinas de recolección de basura, desarrolladas por el entonces estudiante graduado del MIT, Daniel Edwards, hicieron práctico correr Lisp en sistemas de computación de propósito general, pero la eficacia todavía seguía siendo un problema. Esto llevó a la creación de las máquinas Lisp: hardware dedicado para correr ambientes y programas Lisp. Avances tanto en el hardware de computadora como en la tecnología de compiladores pronto hicieron obsoletas a las máquinas de Lisp, en detrimento del mercado del Lisp.

Durante los años 1980 y 1990, fue hecho un gran esfuerzo para unificar los numerosos dialectos del Lisp en un solo lenguaje (más notablemente, InterLisp, Maclisp, ZetaLisp, MetaLisp, y Franz Lisp). El nuevo lenguaje, Common Lisp, fue esencialmente un subconjunto compatible de los dialectos que reemplazó. En 1994, la ANSI publicó el estándar del Common Lisp, "ANSI X3.226-1994 Information Technology Programming Language Common Lisp". En aquel momento el mercado mundial para el Lisp era mucho más pequeño de lo que es hoy.

Habiendo declinado algo en los años 1990, Lisp experimentó un nuevo crecimiento de interés. La mayor parte de la nueva actividad está enfocada alrededor de las implementaciones de fuente abierta del Common Lisp, e incluye el desarrollo aplicaciones y de nuevas bibliotecas portátiles. Este interés puede ser medido en parte por las ventas de la versión impresa de Practical Common Lisp (Common Lisp Práctico) de Peter Seibel, un tutorial para nuevos programadores Lisp publicado en 2004,5 que estuvo brevemente en Amazon.com como el segundo libro de programación más popular. El libro es accesible en línea sin costo.6

Muchos nuevos programadores Lisp fueron inspirados por escritores como Paul Graham y Eric S. Raymond luchando por un lenguaje que otros consideran anticuado. Los nuevos programadores de Lisp frecuentemente describen el lenguaje como una experiencia que abre los ojos y una aclamación de ser substancialmente más productivo que otros lenguajes.7 Este aumento de conciencia puede ser contrastado con el "invierno de la inteligencia artificial" y el breve crecimiento del Lisp a mediados de los 1990.8

En su encuesta de las implementaciones del Common Lisp, Dan Weinreb lista once implementaciones activamente mantenidas. Scieneer Common Lisp es una nueva implementación comercial que bifurcó (fork) del CMUCL con un primer lanzamiento en 2002.

La comunidad del código libre ha creado la nueva infraestructura de soporte: Cliki es un Wiki que recoge la información relacionada con el Common Lisp, el Common Lisp directory lista recursos, el #lisp es un canal popular de IRC (con soporte por un Bot escrito en Lisp), lisppaste soporta la distribución y el intercambio y comentario de retazos de código (snippets), el Planet Lisp recoge el contenido de varios blogs relacionados con el Lisp, en el LispForum el usuario discute tópicos sobre Lisp, Lispjobs es un servicio para anunciar ofertas de trabajo y hay un nuevo servicio de noticias semanales (Weekly Lisp News).

Han sido celebrados los 50 años del Lisp (1958-2008) en LISP50@OOPSLA. Hay varias reuniones de usuario locales regulares (Boston, Vancouver, Hamburg,…), Reuniones Lisp (European Common Lisp Meeting, European Lisp Symposium) y una International Lisp Conference.

La comunidad Scheme mantiene activamente más de veinte implementaciones. Se han desarrollado en los últimos años varias significativas nuevas implementaciones (Chicken, Gauche, Ikarus, Larceny, Ypsilon). El estándar de Scheme Revised5 Report on the Algorithmic Language Scheme fue ampliamente aceptado en la comunidad del Scheme. El proceso Scheme Requests for Implementation ha creado muchas bibliotecas y extensiones casi estándares para el Scheme. Las comunidades de usuario de implementaciones individuales del Scheme continúan creciendo. En 2003 un nuevo proceso de estandarización del lenguaje fue comenzada y condujo al estándar R6RS del Scheme en 2007. El uso académico del Scheme para enseñar ciencias de la computación parece haber declinado algo. Algunas universidades ya no están usando Scheme en sus cursos preliminares de ciencias de la computación.

Hay también algunos nuevos dialectos Lisp. Notablemente: Newlisp (un lenguaje de scripting), Arc (desarrollado por Paul Graham) y recientemente Clojure (desarrollado por Rich Hickey) y NU para la programación con Cocoa de Apple.

Dialectos:

Los dos principales dialectos del Lisp usados para la programación de propósitos generales hoy en día son Common Lisp y Scheme. Estos lenguajes representan opciones de diseño significativamente diferentes.

El Common Lisp, descendiente principalmente de MacLisp, Interlisp, y Lisp Machine Lisp, es un superconjunto ampliado de los primeros dialectos del Lisp, con un estándar de lenguaje grande incluyendo muchos tipos de datos y formas sintácticas incorporados, así como un sistema del objeto. El Scheme es un diseño más minimalista, con un mucho más pequeño conjunto de características estándar pero con ciertas características de implementación (tales como optimización de llamada de cola y continuación completa) no encontradas necesariamente en Common Lisp. El Common Lisp también tomó prestadas ciertas características de Scheme tales como ámbito de léxico y clausura léxica.

El Scheme, es un dialecto del lenguaje Lisp con ámbito estático y cola recursiva auténtica inventado por Guy Lewis Steele Jr. y Gerald Jay Sussman. Fue diseñado para tener una semántica excepcionalmente clara y simple y pocas maneras diferentes de formar expresiones. Una amplia variedad de paradigmas programados encuentran una expresión conveniente en Scheme, incluyendo los estilos imperativo, funcional, y paso de mensajes. El Scheme continúa evolucionando con una serie de los estándares (Revisedn Report on the Algorithmic Language Scheme) y una serie de Scheme Requests for Implementation.

Además, los dialectos del Lisp son usados como lenguajes de scripting en un número de aplicaciones, con los más conocidos siendo el Emacs Lisp en el editor de Emacs, Visual Lisp en AutoCAD, Nyquist en Audacity.

Innovaciones:

El Lisp fue el primer lenguaje de programación homoicónico: la representación primaria del código del programa es el mismo tipo de estructura de la lista que también es usada para las principales estructuras de datos. Como resultado, las funciones del Lisp pueden ser manipuladas, alteradas o aún creadas dentro de un programa Lisp sin un extensivo análisis sintáctico (parsing) o manipulación de código de máquina binario. Esto generalmente es considerado una de las ventajas primarias del lenguaje con respecto a su poder expresivo, y hace al lenguaje favorable a la evaluación metacircular.

La ubicua estructura IF THEN ELSE, ahora admitida como un elemento esencial de cualquier lenguaje de programación, fue inventada por McCarthy para el uso en el Lisp, donde vio su primera apariencia en una forma más general (la estructura cond). Fue heredada por el ALGOL, que la popularizó.

El Lisp influyó profundamente a Alan Kay, el líder de investigación del Smalltalk, y entonces a su vez Lisp fue influenciado por Smalltalk, adoptando las características de la programación orientada a objetos (clases, instancias, etc.) a finales de los años 1970.

En gran parte debido a sus requerimientos de recursos con respecto al temprano hardware computacional (incluyendo los primeros microprocesadores), el Lisp no se hizo tan popular fuera de la comunidad de AI, como lo fueron el FORTRAN y el descendiente del lenguaje ALGOL, el lenguaje C. Lenguajes más nuevos como Java y Python han incorporado algunas versiones limitadas de algunas de las características del Lisp, pero no pueden necesariamente brindar la coherencia y la sinergia de los conceptos completos encontrados en el Lisp. Debido a su conveniencia para aplicaciones mal definidas, complejas, y dinámicas, el Lisp están disfrutando actualmente de un cierto resurgimiento del interés popular.

Algunas de las funciones predefinidas de Lisp tienen símbolos familiares (+ para la suma, \* para el producto), pero otras son más exóticas, especialmente dos que sirven precisamente para manipular listas, descomponiéndolas en sus componentes. Sus nombres ("car" y "cdr") son un poco extraños, reliquias de tiempos pasados y de la estructura de los ordenadores de segunda generación, "car" devuelve la cabeza de una lista y "cdr" su cola o resto.

Lisp sigue una filosofía de tratamiento no-destructivo de los parámetros, de modo que la mayoría de las funciones devuelven una lista resultado de efectuar alguna transformación sobre la que recibieron, pero sin alterar esta última.

Uno de los motivos por los que Lisp es especialmente adecuado para la IA es el hecho de que el código y los datos tengan el mismo tratamiento (como listas); esto hace especialmente sencillo escribir programas capaces de escribir otros programas según las circunstancias.

Lisp fue uno de los primeros lenguajes de programación en incluir manejo de excepciones con las primitivas catch y throw.

Derivado del Lisp es el lenguaje de programación Logo. Sin entrar en detalles, podría decirse que Logo es Lisp sin paréntesis y con operadores aritméticos infijos.

MODULA-2

Modula-2 es un lenguaje de programación cuyo autor es Niklaus Wirth, autor también del lenguaje Pascal.

Como novedad respecto a este último lenguaje, introduce el concepto de módulo, y de encapsulación. Del código contenido en un módulo, sólo se facilita una interfaz pública denominada módulo de definición, permaneciendo el resto oculto (encapsulado) para un desarrollador ajeno, lo que facilita el mantenimiento de dichas estructuras de programación a la vez que imposibilita la modificación de las mismas a cualquiera que no posea el código de su módulo de implementación.

Este concepto de módulo constituye el antecedente de las clases u objetos que se observan en el concepto moderno de Programación Orientada a Objetos (POO); sin embargo, la incapacidad de declarar múltiples instancias de los módulos, así como la ausencia de todo tipo de herencia, impiden afirmar que Modula-2 sea un lenguaje orientado a objetos propiamente dicho.

Modula-2 se utiliza principalmente en las universidades por su excelente adaptación a la enseñanza de lenguajes estructurados, pero en el ámbito laboral su difusión ha sido escasa frente al predominio de C, lo que ha contribuido a crear un distanciamiento entre universidad y mercado laboral.

Existen compiladores de Modula-2 como Mocka o GNU Modula-2 (para GNU/Linux y resto de sistemas soportados por GCC), FST (para MS-DOS) o Native XDS (para Windows) entre otros.

PASCAL:

Pascal es un lenguaje creado por el profesor suizo Niklaus Wirth entre los años 1968 y 1969 y publicado en 1970. Su objetivo era crear un lenguaje que facilitara el aprendizaje de programación a sus alumnos, utilizando la programación estructurada y estructuración de datos. Sin embargo con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo.

Pascal se caracteriza por ser un lenguaje de programación estructurado fuertemente tipado. Esto implica que:

El código está dividido en porciones fácilmente legibles llamadas funciones o procedimientos. De esta forma Pascal facilita la utilización de la programación estructurada en oposición al antiguo estilo de programación monolítica.

El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.

Historia:

Pascal, llamado así en honor del matemático y filósofo francés Blaise Pascal, fue desarrollado por Niklaus Wirth.

Antes de su trabajo en Pascal, Wirth había desarrollado Euler y ALGOL W y más tarde pasó a desarrollar los lenguajes Modula-2 y Oberon, al estilo de Pascal.

Inicialmente, Pascal era en gran parte, pero no exclusivamente, destinado a enseñar a los estudiantes la programación estructurada. Una generación de estudiantes usa Pascal como lenguaje de introducción de cursos de graduación. Las variantes de Pascal también se han usado para todo, desde proyectos de investigación a juegos de PC y sistemas embebidos. Existen nuevos compiladores de Pascal que son ampliamente utilizados.

Pascal fue el lenguaje primario de alto nivel utilizado para el desarrollo en el Apple Lisa, y en los primeros años del Macintosh. Algunas partes del sistema operativo original del Macintosh se traducen a mano al lenguaje ensamblador del Motorola 68000 de sus fuentes Pascal. El sistema de Tipopuesta TeX de Donald E. Knuth fue escrito en WEB, el sistema de Programación literaria original, basado en DEC PDP-10 Pascal, mientras que aplicaciones como Total Commander, Skype y Macromedia Captivate fueron escritos en Delphi (Object Pascal).

Object Pascal (Embarcadero Delphi) todavía se utiliza para el desarrollo de aplicaciones Windows, pero también tiene la capacidad de compilación cruzada del mismo código para Mac, iOS y Android. Otra versión multiplataforma llamado Free Pascal, con el IDE Lazarus, es popular entre los usuarios de Linux, ya que también ofrece desarrollo escribe una vez, compila para todos (write once, compile anywhere). CodeTyphon es una distribución de Lazarus con más paquetes preinstalados y compiladores cruzados.

Características:

Otra diferencia importante es que, en Pascal, el tipo de una variable se fija en su definición; las asignaciones a variables de valores de tipo incompatible no están autorizadas (en C, en cambio, el compilador hace el mejor esfuerzo para dar una interpretación a casi todo tipo de asignaciones).

Esto previene errores comunes donde variables son usadas incorrectamente porque el tipo es desconocido; y también evita la necesidad de notación húngara, que vienen a ser prefijos que se añaden a los nombres de las variables y que indican su tipo.

Implementa:

Las primeras versiones del compilador de Pascal, entre ellas la más distribuida fue UCSD Pascal, traducían el lenguaje en código para una máquina virtual llamada máquina-P. La gran ventaja de este enfoque es que para tener un compilador de Pascal en una nueva arquitectura de máquina solo hacía falta reimplementar la máquina-P. Como consecuencia de esto, solo una pequeña parte del intérprete tenía que ser reescrita hacia muchas arquitecturas.

En los años 1980, Anders Hejlsberg escribió el compilador Blue Label Pascal para el Nascom-2. Más tarde fue a trabajar para Borland y reescribió su compilador que se convirtió en Turbo Pascal para el IBM PC. Este nuevo compilador se vendió por 49 dólares, un precio orientado a la distribución masiva.

El económico compilador de Borland tuvo una larga influencia en la comunidad de Pascal que comenzó a utilizarlo principalmente en el IBM PC. En busca de un lenguaje estructurado muchos aficionados al PC reemplazaron el BASIC por este producto. Dado que Turbo Pascal sólo estaba disponible para una arquitectura, traducía directamente hacia el código máquina del Intel 8088, logrando construir programas que se ejecutaban mucho más rápidamente que los producidos en los esquemas interpretados.

Durante los años 1990, estuvo disponible la tecnología para construir compiladores que pudieran producir código para diferentes arquitecturas de hardware. Esto permitió que los compiladores de Pascal tradujeran directamente al código de la arquitectura en que corre.

Con Turbo Pascal versión 5.5, Borland agregó programación orientada a objetos a Pascal.

Sin embargo, Borland después decidió mejorar esa extensión del lenguaje introduciendo su producto Delphi, diseñado a partir de estándar Object Pascal, propuesto por Apple Inc. como base. Borland también lo llamó Object Pascal en las primeras versiones, pero cambió el nombre a 'lenguaje de programación Delphi' en sus últimas versiones.

Librerías:

Las librerías contienen procedimientos y funciones. El objetivo es poner esas funciones a disposición de ser usadas.

Cuando planeemos usar librerías, nos hemos de remitir a la sección USES de nuestro programa.

PERL:

Perl es un lenguaje de programación diseñado por Larry Wall en 1987. Perl toma características del lenguaje C, del lenguaje interpretado bourne shell (sh), AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación.

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

Historia

Larry Wall comenzó a trabajar en Perl en 1987 mientras trabajaba como programador en Unisys3 y anunció la versión 1.0 en el grupo de noticias comp.sources.misc el 18 de diciembre de 1987. El lenguaje se expandió rápidamente en los siguientes años. Perl 2, publicado en 1988, aportó un mejor motor de expresiones regulares. Perl 3, publicado en 1989, añadió soporte para datos binarios.

Hasta 1991 la única documentación de Perl era una simple (y cada vez más larga) página de manual Unix. En 1991 se publicó Programming Perl (el libro del camello) y se convirtió en la referencia de facto del lenguaje. Al mismo tiempo, el número de versión de Perl saltó a 4, no por marcar un gran cambio en el lenguaje, sino por identificar a la versión que estaba documentada en el libro.

Perl 4 trajo consigo una serie de lanzamientos de mantenimiento, culminando en Perl 4.036 en 1993. En este punto, Larry Wall abandonó Perl 4 para comenzar a trabajar en Perl 5. Perl 4 se quedaría en esa versión hasta hoy.

El desarrollo de Perl 5 continuó en 1994. La lista de correo perl5-porters se estableció en mayo de 1994 para coordinar el trabajo de adaptación de Perl 5 a diferentes plataformas. Es el primer foro para desarrollo, mantenimiento y adaptación de Perl 5.

Perl 5 fue publicado el 17 de octubre de 1994. Fue casi una completa reescritura del intérprete y añadió muchas nuevas características al lenguaje, incluyendo objetos, referencias, paquetes y módulos. A destacar, los módulos proveen de un mecanismo para extender el lenguaje sin modificar el intérprete. Esto permitió estabilizar su núcleo principal, además de permitir a los programadores de Perl añadirle nuevas características.

El 26 de octubre de 1995, se creó el Comprehensive Perl Archive Network (CPAN). CPAN es una colección de sitios web que almacenan y distribuyen fuentes en Perl, binarios, documentación, scripts y módulos. Originalmente, cada sitio CPAN debía ser accedido a través de su propio URL; hoy en día, http://www.cpan.org redirecciona automáticamente a uno de los cientos de repositorios espejo de CPAN.

En 2008, Perl 5 continúa siendo mantenido. Características importantes y algunas construcciones esenciales han sido añadidas, incluyendo soporte Unicode, Hilos (threads), un soporte importante para la programación orientada a objetos y otras mejoras

Características:

La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.

Perl también toma características de la programación shell. Todas las variables son marcadas con un Sigilo precedente (Sigil, en inglés). Los sigilos identifican inequívocamente los nombres de las variables, permitiendo a Perl tener una rica sintaxis. Notablemente, los sigilos permiten interpolar variables directamente dentro de las cadenas de caracteres (string). Como en los shell, Perl tiene muchas funciones integradas para tareas comunes y para acceder a los recursos del sistema.

Perl toma las listas del Lisp, hash (memoria asociativa) del AWK y expresiones regulares de la sed. Todo esto simplifica y facilita todas las formas del análisis sintáctico, manejo de texto y tareas de gestión de datos.

En Perl 5, se añadieron características para soportar estructuras de datos complejas, funciones de primer orden (p. e. clausuras como valores) y un modelo de programación orientada a objetos. Estos incluyen referencias, paquetes y una ejecución de métodos basada en clases y la introducción de variables de ámbito léxico, que hizo más fácil escribir código robusto (junto con el pragma strict). Una característica principal introducida en Perl 5 fue la habilidad de empaquetar código reutilizable como módulos. Larry Wall indicó más adelante que "la intención del sistema de módulos de Perl 5 era apoyar el crecimiento de la cultura Perl en vez del núcleo de Perl".

Todas las versiones de Perl hacen el tipificado automático de datos y la gestión de memoria. El intérprete conoce el tipo y requerimientos de almacenamiento de cada objeto en el programa; reserva y libera espacio para ellos según sea necesario. Las conversiones legales de tipo se hacen de forma automática en tiempo de ejecución; las conversiones ilegales son consideradas errores fatales.

Diseño:

El diseño de Perl puede ser entendido como una respuesta a tres amplias tendencias de la industria informática: rebaja de los costes en el hardware, aumento de los costes laborales y las mejoras en la tecnología de compiladores. Anteriormente, muchos lenguajes de ordenador como Fortran y C, fueron diseñados para hacer un uso eficiente de un hardware caro. En contraste, Perl es diseñado para hacer un uso eficiente de los costosos programadores de ordenador.

Perl tiene muchas características que facilitan la tarea del programador a costa de unos requerimientos de CPU y memoria mayores. Estas incluyen gestión de memoria automática; tipo de dato dinámico; strings, listas y hashes; expresiones regulares; introspección y una función eval().

Larry Wall fue adiestrado como lingüista y el diseño de Perl ha sido muy aleccionado con principios lingüísticos. Ejemplos incluyen la Codificación Huffman (las construcciones más comunes deben ser las más cortas), buena distribución (la información importante debe ir primero) y una larga colección de primitivas del lenguaje. Perl favorece las construcciones del lenguaje, tan naturales, como para los humanos son la lectura y la escritura, incluso si eso hace más complicado al intérprete Perl.

La sintaxis de Perl refleja la idea de que "cosas que son diferentes deben parecer diferentes". Por ejemplo, escalares, arrays y hashes tienen diferente sigilo. Índices de array y claves hash usan diferentes clases de paréntesis. Strings y expresiones regulares tienen diferentes delimitadores estándar.

Esta aproximación puede contrastarse con lenguajes como Lisp, donde la misma construcción S-expresión y sintaxis básica se usa para muchos y variados propósitos.

Perl tiene características que soportan una variedad de paradigmas de programación, como la imperativa, funcional y la orientada a objetos. Al mismo tiempo, Perl no obliga a seguir ningún paradigma en particular, ni obliga al programador a elegir alguna de ellas.

Implementación:

Perl está implementado como un intérprete, escrito en C, junto con una gran colección de módulos, escritos en Perl y C. La distribución fuente tiene, en 2005, 12 MB cuando se empaqueta y comprime en un fichero tar. El intérprete tiene 150.000 líneas de código C y se compila en un ejecutable de 1 MB en las arquitecturas de hardware más típicas. De forma alternativa, el intérprete puede ser compilado como una biblioteca y ser embebida en otros programas. Hay cerca de 500 módulos en la distribución, sumando 200.000 líneas de Perl y unas 350.000 líneas adicionales de código C. Mucho del código C en los módulos consiste en tablas de codificación de caracteres.

El intérprete tiene una arquitectura orientada a objetos. Todos los elementos del lenguaje Perl —escalares, listas, hashes, referencias a código, manejadores de archivo— están representados en el intérprete como estructuras C. Las operaciones sobre estas estructuras están definidas como una numerosa colección de macros, typedef y funciones; esto constituye la API C de Perl. La API Perl puede ser desconcertante para el no iniciado, pero sus puntos de entrada siguen un esquema de nombres coherente, que ayuda a los que quieran utilizarla.

La ejecución de un programa Perl se puede dividir, generosamente, en dos fases: tiempo de compilación y tiempo de ejecución.7 En tiempo de compilación el intérprete parsea el texto del programa en un árbol sintáctico. En tiempo de ejecución, ejecuta el programa siguiendo el árbol. El texto es parseado sólo una vez y el árbol sintáctico es optimizado antes de ser ejecutado, para que la fase de ejecución sea relativamente eficiente. Las optimizaciones del árbol sintáctico en tiempo de compilación incluyen simplificación de expresiones constantes, propagación del contexto y optimización en trozos sueltos de código. Sin embargo, las fases de compilación y ejecución pueden anidarse: un bloque BEGIN se ejecuta en tiempo de compilación, mientras que una función eval inicia una compilación durante una ejecución. Ambas operaciones están implícitas en otras -de forma notable, la cláusula use que carga bibliotecas, conocidas en Perl como módulos, implica un bloque BEGIN.

Perl es un lenguaje dinámico y tiene una gramática sensible al contexto que puede quedar afectada por el código ejecutado durante una fase de ejecución intermedia. Por eso Perl no puede ser parseado mediante una aplicación directa de analizadores sintácticos/parseadores Lex/Yacc. En cambio, el intérprete implementa su propio analizador léxico, que coordinado con un parseador modificado GNU bison resuelve las ambigüedades del lenguaje. Se ha dicho que "sólo perl puede parsear Perl", queriendo decir que sólo el intérprete Perl (perl) puede parsear el lenguaje Perl (Perl). La razón de esto se atestigua por las persistentes imperfecciones de otros programas que emprenden la tarea de parsear Perl, como los analizadores de código y los auto-indentadores, que tienen que vérselas no sólo con las muchas formas de expresar inequívocamente construcciones sintácticas, sino también con el hecho de que también Perl no puede, en general, ser parseado sin antes ser ejecutado.

El mantenimiento del intérprete Perl, a lo largo de los años, se ha vuelto cada vez más difícil. El núcleo ha estado en continuo desarrollo desde 1994. El código ha sido optimizado en rendimiento a expensas de la simplicidad, claridad y unas interfaces internas más fuertes. Nuevas características han sido añadidas, manteniendo todavía, compatibilidad virtualmente completa hacia atrás con las primeras versiones. El tamaño y la complejidad del intérprete son

No hay una especificación o estándar escrito para el lenguaje Perl y no hay planes de crear uno para la versión actual de Perl. Siempre ha existido sólo una implementación del intérprete. Este intérprete, junto con los test funcionales, forman la especificación de facto del lenguaje.

**PHP**

PHP es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

Fue creado originalmente por Rasmus Lerdorf en el año 1995. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP.Este lenguaje forma parte del software libre publicado bajo la licencia PHP, que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término PHP.

Cronología

PHP es un acrónimo recursivo que significa PHP Hypertext Preprocessor (inicialmente PHP Tools, o, Personal Home Page Tools). Fue creado originalmente por Rasmus Lerdorf; sin embargo, la implementación principal de PHP es producida ahora por The PHP Group y sirve como el estándar de facto para PHP, al no haber una especificación formal. Publicado con la PHP License, la Free Software Foundation considera esta licencia como software libre.

Visión general

PHP puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. El lenguaje PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores. El número de sitios basados en PHP se ha visto reducido progresivamente en los últimos años, con la aparición de nuevas tecnologías como Node.JS, Golang, ASP.NET, etc. El sitio web de Wikipedia está desarrollado en PHP. Es también el módulo Apache más popular entre las computadoras que utilizan Apache como servidor web.

El gran parecido que posee PHP con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones.

Aunque todo en su diseño está orientado a facilitar la creación de sitios webs, es posible crear aplicaciones con una interfaz gráfica para el usuario, utilizando alguna extensión como puede ser PHP-Qt, PHP-GTK, WxPHP, WinBinder, Roadsend PHP, Phalanger, Phc o HiP Hop VM. También puede ser usado desde la línea de comandos, de la misma manera como Perl o Python pueden hacerlo; a esta versión de PHP se la llama PHP-CLI (Command Line Interface).

Cuando el cliente hace una petición al servidor para que le envíe una página web, el servidor ejecuta el intérprete de PHP. Éste procesa el script solicitado que generará el contenido de manera dinámica (por ejemplo obteniendo información de una base de datos). El resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente.

Mediante extensiones es también posible la generación de archivos PDF,8 Flash, así como imágenes en diferentes formatos.

Permite la conexión a diferentes tipos de servidores de bases de datos tanto SQL como NoSQL tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird, SQLite o MongoDB.9

PHP también tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos, tales como Unix (y de ese tipo, como Linux o Mac OS X) y Microsoft Windows, y puede interactuar con los servidores de web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

PHP es una alternativa a las tecnologías de Microsoft ASP y ASP.NET (que utiliza C# y Visual Basic .NET como lenguajes), a ColdFusion de la empresa Adobe, a JSP/Java, CGI/Perl y a Node.js/Javascript. Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia GNU, existe además un entorno de desarrollo integrado comercial llamado Zend Studio. CodeGear (la división de lenguajes de programación de Borland) ha sacado al mercado un entorno de desarrollo integrado para PHP, denominado 'Delphi for PHP. También existen al menos un par de módulos para Eclipse, uno de los entornos más populares.

Historia

Fue originalmente diseñado en Perl, con base en la escritura de un grupo de CGI binarios escritos en el lenguaje C por el programador danés-canadiense Rasmus Lerdorf en el año 1994 para mostrar su currículum vítae y guardar ciertos datos, como la cantidad de tráfico que su página web recibía. El 8 de junio de 1995 fue publicado "Personal Home Page Tools" después de que Lerdorf lo combinara con su propio Form Interpreter para crear PHP/FI.

Dos programadores israelíes del Technion, Zeev Suraski y Andi Gutmans, reescribieron el analizador sintáctico (parser, en inglés) en 1997 y crearon la base del PHP3, y cambiaron el nombre del lenguaje por PHP: Hypertext Preprocessor.2 Inmediatamente comenzaron experimentaciones públicas de PHP3, y se publicó oficialmente en junio de 1998. Para 1999, Suraski y Gutmans reescribieron el código de PHP, y produjeron lo que hoy se conoce como motor Zend. También fundaron Zend Technologies en Ramat Gan, Israel.2

En mayo del 2000, PHP 4 se lanzó bajo el poder del motor Zend 1.0. El 13 de julio del 2007 se anunció la suspensión del soporte y desarrollo de la versión 4 de PHP,11 y, a pesar de lo anunciado, se ha liberado una nueva versión con mejoras de seguridad, la 4.4.8, publicada el 13 de enero del 2008.

El 13 de julio del 2004, se lanzó PHP 5, utilizando el motor Zend Engine 2.0 (o Zend Engine 2).2 Incluye todas las ventajas que provee el nuevo Zend Engine 2, como:

Mejor soporte para la programación orientada a objetos, que en versiones anteriores era extremadamente rudimentario.

Mejoras de rendimiento

Mejor soporte para MySQL con extensión completamente reescrita

* Mejor soporte a XML (XPath, DOM, etc.)
* Soporte nativo para SQLite
* Soporte integrado para SOAP
* Iteradores de datos
* Manejo de excepciones
* Mejoras con la implementación con Oracle

PL/1:

PL/1, acrónimo de Programming Language 1 (Lenguaje de Programación 1), fue propuesto por IBM hacia 1970 para responder simultáneamente a las necesidades de las aplicaciones científicas y comerciales, disponible en las novedosas plataformas de utilidad general IBM 360 y más adelante IBM 370.

Este lenguaje tenía muchas de las características que más adelante adoptaría el lenguaje C y algunas de C++. Por desgracia, IBM registra el nombre del lenguaje como forma de mantener control sobre su desarrollo, lo que disuadió a otras empresas de dar ese nombre a sus implementaciones. No siendo posible encontrar un único lenguaje para diversas plataformas, los potenciales usuarios del lenguaje prefirieron no adoptarlo a pesar de sus múltiples innovaciones, que incluían multiprocesamiento, recursión, estructuras de control modernas, facilidades para la puesta a punto, asignación dinámica de espacio para estructuras de datos, procedimientos genéricos, etc.

Sin embargo, dentro de los usuarios de IBM, el lenguaje se utilizó con bastante intensidad, y el proyecto Multics utilizó PL/1 como lenguaje de desarrollo para su sistema de operación.

PL/1 fue probablemente el primer lenguaje comercial cuyo compilador estaba escrito en el lenguaje que compilaba.

PL/SQL

PL/SQL (Procedural Language/Structured Query Language) es un lenguaje de programación incrustado en Oracle.

PL/SQL soportará todas las consultas, ya que la manipulación de datos que se usa es la misma que en SQL, incluyendo nuevas características:

* El manejo de variables.
* Estructuras modulares.
* Estructuras de control de flujo y toma de decisiones.
* Control de excepciones.

El lenguaje PL/SQL está incorporado en:

* Servidor de la base de datos.
* Herramientas de Oracle (Forms, Reports, ...).

En un entorno de base de datos los programadores pueden construir bloques PL/SQL para utilizarlos como procedimientos o funciones, o bien pueden escribir estos bloques como parte de scripts SQL\*Plus.

Los programas o paquetes de PL/SQL se pueden almacenar en la base de datos como otro objeto, y todos los usuarios que estén autorizados tienen acceso a estos paquetes. Los programas se ejecutan en el servidor para ahorrar recursos a los clientes.

PYTHON:

Para otros usos de este término, véase Pitón.

Python

Python-logo-notext.svg

Desarrollador(es)

Python Software Foundation

http://www.python.org/

Información general

Extensiones comunes .py, .pyc, .pyd, .pyo, .pyw

Paradigma Multiparadigma: orientado a objetos, imperativo, funcional, reflexivo

Apareció en 1991

Diseñado por Guido van Rossum

Última versión estable 3.6.0 / 2.7.11 (Principio de 2017/

5 de diciembre de 2015 (1 año y 4 meses))

Sistema de tipos Fuertemente tipado, dinámico

Implementaciones CPython, IronPython, Jython, Python for S60, PyPy, Pygame, ActivePython, Unladen Swallow

Dialectos Stackless Python, RPython

Influido por ABC, ALGOL 68, C, Haskell, Icon, Lisp, Modula-3, Perl, Smalltalk, Java

Ha influido a Boo, Cobra, D, Falcon, Genie, Groovy, Ruby, JavaScript, Cython, Go

Sistema operativo Multiplataforma

Licencia Python Software Foundation License

[editar datos en Wikidata]

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,1 que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Historia:

Python fue creado a finales de los ochenta2 por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.3

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.4

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for Life, BDFL).

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources.5 En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como “una de las mayores unidades de programación de Python”.2 El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else.3 En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la programación funcional: lambda, reduce, filter y map. Van Rossum explicó que “hace 12 años, Python adquirió lambda, reduce(), filter() y map(), cortesía de un hacker informático de Lisp que las extrañaba y que envió parches”.6 El donante fue Amrit Prem; no se hace ninguna mención específica de cualquier herencia de Lisp en las notas de lanzamiento.

La última versión liberada proveniente de CWI fue Python 1.2. En 1995, van Rossum continuó su trabajo en Python en la Corporation for National Research Initiatives (CNRI) en Reston, Virginia, donde lanzó varias versiones del software.

Durante su estancia en CNRI, van Rossum lanzó la iniciativa Computer Programming for Everybody (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de 'alfabetización' básico en lenguajes de programación, similar a la alfabetización básica en inglés y habilidades matemáticas necesarias por muchos trabajadores. Python tuvo un papel crucial en este proceso: debido a su orientación hacia una sintaxis limpia, ya era idóneo, y las metas de CP4E presentaban similitudes con su predecesor, ABC. El proyecto fue patrocinado por DARPA.7 En el año 2007, el proyecto CP4E está inactivo, y mientras Python intenta ser fácil de aprender y no muy arcano en su sintaxis y semántica, alcanzando a los no-programadores, no es una preocupación activa.8

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo.9 Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations.

Python 2.0 tomó una característica mayor del lenguaje de programación funcional Haskell: listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de recolección de basura capaz de recolectar referencias cíclicas.9

Posterior a este doble lanzamiento, y después que van Rossum dejó CNRI para trabajar con desarrolladores de software comercial, quedó claro que la opción de usar Python con software disponible bajo GNU GPL era muy deseable. La licencia usada entonces, la Python License, incluía una cláusula estipulando que la licencia estaba gobernada por el estado de Virginia, por lo que, bajo la óptica de los abogados de Free Software Foundation (FSF), se hacía incompatible con GPL. CNRI y FSF se relacionaron para cambiar la licencia de software libre de Python para hacerla compatible con GPL. En el año 2001, van Rossum fue premiado con FSF Award for the Advancement of Free Software.

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de bugs, y con una nueva licencia compatible con GPL.1

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: Python Software Foundation License. Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a Python Software Foundation (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la Apache Software Foundation.1 Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de static scoping (del cual Scheme es el originador).10

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un modelo de objetos de Python puro y consistente.11 También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.12

Conclusiones Personales

La programación es una fase completa para su uso en computadoras, que a conforme que va pasando el tiempo, desde la primera vez que una maquina proyectaba en una pantalla “Hola Mundo” hasta estos tiempos que cada vez se busca mejorar y renovar la programación en distintas formas.