FUNDAMENTOS DE PROGRAMACION

Sarah Valentina Sánchez Torres

Ilse Dayana Alfonso Bonilla

1002

IED SAN JOSEMARIA ESCRIVA DE BALAGUER

Tecnología e Informática

Chía, Cundinamarca

2014

INTRODUCCION

Este trabajo tiene como objetivo, identificar los tipos de lenguajes de programación, para la realización de un proyecto que se ha venido planteando ya desde hace unos meses junto con el profesor del área de informática y es la creación de juegos, el ideal es que estos juegos sean constructivos y emitan un mensaje educativo para aquellos que ingresen a estos, para el desarrollo de este proyecto es necesaria esta investigación previa sobre la programación de los computadores y los lenguajes de programación.

Un lenguaje de programación no es más que un sistema estructurado y diseñado principalmente para que las máquinas y computadoras se entiendan entre sí y con nosotros, los humanos. Contiene un conjunto de acciones consecutivas que el ordenador debe ejecutar. Estos lenguajes de programación usan diferentes normas o bases y se utilizan para controlar cómo se comporta una máquina (por ejemplo, un ordenador), también pueden usarse para crear programas informáticos, etc.

Para finalizar Hoy en día utilizamos aplicaciones para todo, programas especializados en diferentes tareas, etc.…si queremos saber qué hay detrás de todo eso necesitamos nadar en el maravilloso e interesante mundo de los lenguajes de programación y así entender por qué cuando a un juego le doy al botón “play” puedo empezar a jugar, o sencillamente por qué puedo hacer una suma en una calculadora online.

RESUMEN

En este trabajo nos enfatizamos básicamente el ocho actividades en las cuales se toca el tema de programación y lenguaje de programación, iniciamos informándonos del tema por medio de unas definiciones, continuamos, identificando ejemplos que ayuden a visualizar con más claridad de que trata el tema, llegando a un punto intermedio en el cual tenemos unas bases de conocimiento y debemos empezar a emplear analogías e incluir ventajas y desventajas que permitan conocer fortalezas y debilidades del proyecto a tratar o de los sistemas informáticos que se utilizaran. Para finalizar entramos en la búsqueda de programas que nos permitan ejecutar nuestro proyecto, buscando con esto un satisfecho término del proceso de investigación.

La programación de computación en este caso juega un papel fundamental porque abre la posibilidad de habilitar los computadores con el objetivo de realizar, el proyecto, este proyecto será dirigido por el docente del área de informática, el cual con su experiencia y su largo recorrido por el mundo tecnológico nos enseñara como real un juego con la proyección de que este nos solo sea una fuente de conocimientos tecnológicos sino que también sirva para la retroalimentación del ser como tal.

Para concluir recordar a todos los lectores que este trabajo es específicamente investigativo, ella que es el trabajo inicial de este plan o idea de proyecto, esperamos que al final de este trabajo no solo hayamos brindado una ayuda para todos los que se documentaron por medio de este informe, sino que también podamos considerar este taller como la estructura o el esqueleto que sostendrá nuestra meta de un juego educativo.

PALABRAS CLAVE

Proceso: Un proceso es un conjunto de actividades mutuamente relacionadas o que, al interactuar, transforman elementos de entrada y los convierten en resultados. Un proceso puede informalmente entenderse como un programa en ejecución.

Programación de Computadores: La programación informática, a menudo acortada como programación, es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas computacionales. El código fuente es escrito en un lenguaje de programación. El propósito de la programación es crear programas que exhiban un comportamiento deseado.

Lenguaje de programación: Captura de la microcomputadora Commodore PET-32 mostrando un programa en el lenguaje de programación BASIC, bajo el emulador VICE en una distribución GNU/Linux. Un ejemplo de código fuente escrito en el lenguaje de programación Java, que imprimirá el mensaje "Hello World!" a la salida estándar cuando es compilado y ejecutado**.** Un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras.

Implementación: Código fuente de un programa escrito en ellenguaje de programación Java. La implementación de un lenguaje es la que provee una manera de que se ejecute un programa para una determinada combinación desoftware y hardware. Existen básicamente dos maneras de implementar un lenguaje: compilación e interpretación.

FUNDAMENTOS DE PROGRAMACIÒN

1. Definir los siguientes términos:
    - **Proceso:** Un proceso es un conjunto de actividades mutuamente relacionadas o que, al interactuar, transforman elementos de entrada y los convierten en resultados. Un proceso puede informalmente entenderse como un programa en ejecución. Formalmente un proceso es "Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados". Se denomina proceso a la consecución de determinados actos, acciones, sucesos o hechos que deben necesariamente sucederse para completar un fin específico. Todos estos pasos o instancias que componen un proceso deben ser organizados, coordinados y realizados de manera sistemática, de a uno por vez (secuencia alternativa) o pueden incluso superponerse las instancias (secuencia simultánea). Proceso es un término que, si bien podemos definir como lo hemos hecho de manera precisa, refiere a diferentes fines según sobre qué estemos hablando.
    - **Actividad:** La actividad es una faceta de la psicología. Mediatiza la vinculación del sujeto con el mundo real. La actividad es generadora del reflejo psíquico el cual, a su vez, mediatiza a la propia actividad. Siempre está ligada a cierta necesidad que provoca la búsqueda. Durante la realización de la actividad colectiva e individual tiene lugar el reflejo psíquico de la realidad y se forma la conciencia.

      La actividad humana consciente tendiente hacia una finalidad es la sustancia de la conciencia humana porque es un proceso objetivo tanto como todos los procesos de la naturaleza.

      En psicología se estudian la actividad objetal externa y la actividad interna donde esta última es secundaria porque se forma en proceso

de interiorización de la actividad objetal externa formando un plano interior de la conciencia. Este proceso de interiorización Lev Vygotski lo interpretaba como pasaje de la función psíquica superior desde el plano social externo al plano individual interno de su realización.

La actividad está estrechamente interrelacionado con los conceptos conciencia y de lo ideal.

- **Programar:** Un programa informático es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora. Sin programas, estas máquinas no pueden funcionar.1 2 Al conjunto general de programas, se le denomina software, que más genéricamente se refiere al equipamiento lógico o soporte lógico de una computadora digital. En informática, se los denomina comúnmente binarios, (propio en sistemas Unix, donde debido a la estructura de este último, los ficheros no necesitan hacer uso de extensiones; posteriormente, los presentaron como ficheros ejecutables, con extensión .exe, en los sistemas operativos de la familia Windows) debido a que una vez que han pasado por el proceso de compilación y han sido creados, las instrucciones que se escribieron en un determinado lenguaje de programación, han sido traducidas al único idioma que la máquina comprende, combinaciones de ceros y unos llamado código máquina. El mismo término, puede referirse tanto a un programa ejecutable, como a su código fuente, el cual es transformado en un binario una vez que es compilado. Generalmente el código fuente lo escriben profesionales conocidos como programadores. Este código se escribe en un lenguaje que sigue uno de los siguientes dos paradigmas: imperativo o declar. Un programa informático es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora. Sin programas, estas máquinas no pueden funcionar.1 2 Al conjunto general de programas, se le denomina software, que

más genéricamente se refiere al equipamiento lógico o soporte lógico de una computadora digital.

En informática, se los denomina comúnmente binarios, (propio en sistemas Unix, donde debido a la estructura de este último, los ficheros no necesitan hacer uso de extensiones; posteriormente, los presentaron como ficheros ejecutables, con extensión .exe, en los sistemas operativos de la familia Windows) debido a que una vez que han pasado por el proceso de compilación y han sido creados, las instrucciones que se escribieron en un determinado lenguaje de programación, han sido traducidas al único idioma que la máquina comprende, combinaciones de ceros y unos llamado código máquina. El mismo término, puede referirse tanto a un programa ejecutable, como a su código fuente, el cual es transformado en un binario una vez que es compilado.

Generalmente el código fuente lo escriben profesionales conocidos como programadores. Este código se escribe en un lenguaje que sigue uno de los siguientes dos paradigmas: imperativo o declarativo, y que posteriormente puede ser convertido en una imagen ejecutable a través de un programa-compilador. Cuando se pide que el programa sea ejecutado, el procesador ejecuta instrucción por instrucción.

De acuerdo a sus funciones, los programas informáticos se clasifican en software de sistema y software de aplicación. En los computadores actuales, al hecho de ejecutar varios programas de forma simultánea y eficiente, se le conoce como multitarea.

Programación de Computadores: La programación informática, a menudo acortada como programación, es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas computacionales. 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. 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.

- **Lenguaje Binario:** El sistema binario, llamado también sistema diádico en ciencias de la computación, es un sistema de numeración en el que los números se representan utilizando solamente las cifras cero y uno (0 y 1). Es uno de los que se utiliza en las computadoras, debido a que trabajan internamente con dos niveles de voltaje, por lo cual su sistema de numeración natural es el sistema binario (encendido, apagado 0). el sistema binario puede ser representado solo por dos dígitos. Un número binario puede ser representado por cualquier secuencia de bits (dígitos binarios), que suelen representar cualquier mecanismo capaz de usar dos estados mutuamente excluyentes. secuencias de símbolos podrían ser interpretadas como el mismo valor numérico binario:

```
1 0 1 0 0 1 1 0 1 0
| - | - - | | - | -
```

- x o x o o x x o x o
- y n y n n y y n y n

- Lenguaje Asambler: El lenguaje ensamblador, o assembler (assembly language en inglés), es un lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador. Esta representación es usualmente definida por el fabricante de hardware, y está basada en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portátiles.

  Un programa utilitario llamado ensamblador es usado para traducir sentencias del lenguaje ensamblador al código de máquina del computador objetivo. El ensamblador realiza una traducción más o menos isomorfa (un mapeo de uno a uno) desde las sentencias mnemónicas a las instrucciones y datos de máquina. Esto está en contraste con los lenguajes de alto nivel, en los cuales una sola declaración generalmente da lugar a muchas instrucciones de máquina.

2. ¿Cómo se clasifican los lenguajes de programación?

Un lenguaje de programación es un lenguaje inventado para controlar una máquina, (normalmente, un ordenador). Hay muchísimos, de toda clase de tipos y características, inventados para facilitar el abordaje de distintos

problemas, el mantenimiento del software, su reutilización, mejorar la productividad, etc.

Los lenguajes de programación se pueden clasificar según varios criterios. He encontrado doce en total: Nivel de abstracción, propósito, evolución histórica, manera de ejecutarse, manera de abordar la tarea a realizar, paradigma de programación, lugar de ejecución, concurrencia, interactividad, realización visual, determinismo y productividad.

Hay que tener en cuenta también, que en la práctica, la mayoría de lenguajes no pueden ser puramente clasificados en una categoría, pues surgen incorporando ideas de otros lenguajes y de otras filosofías de programación, pero no importa al establecer las clasificaciones, pues el auténtico objetivo de las mismas es mostrar los rangos, las posibilidades y tipos de lenguajes que hay.

1. Nivel de abstracción.

Según el nivel de abstracción, o sea, según el grado de cercanía a la máquina:

Lenguajes de bajo nivel: La programación se realiza teniendo muy en cuenta las características del procesador. Ejemplo: Lenguajes ensamblador.

Lenguajes de nivel medio: Permiten un mayor grado de abstracción pero al mismo tiempo mantienen algunas cualidades de los lenguajes de bajo nivel. Ejemplo: C puede realizar operaciones lógicas y de desplazamiento con bits, tratar todos los tipos de datos como lo que son en realidad a bajo nivel (números), etc.

Lenguajes de alto nivel: Más parecidos al lenguaje humano. Manejan conceptos, tipos de datos, etc., de una manera cercana al pensamiento humano ignorando (abstrayéndose) del funcionamiento de la máquina. Ejemplos: Java, Ruby.

Hay quien sólo considera lenguajes de bajo nivel y de alto nivel, (en ese caso, C es considerado de alto nivel).

2. Propósito.

Según el propósito, es decir, el tipo de problemas a tratar con ellos:

Lenguajes de propósito general: Aptos para todo tipo de tareas: Ejemplo: C.

Lenguajes de propósito específico: Hechos para un objetivo muy concreto. Ejemplo: Csound (para crear ficheros de audio).

Lenguajes de programación de sistemas: Diseñados para realizar sistemas operativos o drivers. Ejemplo: C.

Lenguajes de script: Para realizar tareas varias de control y auxiliares. Antiguamente eran los llamados lenguajes de procesamiento por lotes (batch) o JCL ("Job Control Languages"). Se subdividen en varias clases (de shell, de GUI, de programación web, etc.). Ejemplos: bash (shell), mIRC script, JavaScript (programación web).

3. Evolución histórica.

Con el paso del tiempo, se va incrementando el nivel de abstracción, pero en la práctica, los de una generación no terminan de sustituir a los de la anterior:

Lenguajes de primera generación (1GL): Código máquina.

Lenguajes de segunda generación (2GL): Lenguajes ensamblador.

Lenguajes de tercera generación (3GL): La mayoría de los lenguajes modernos, diseñados para facilitar la programación a los humanos. Ejemplos: C, Java.

Lenguajes de cuarta generación (4GL): Diseñados con un propósito concreto, o sea, para abordar un tipo concreto de problemas. Ejemplos: NATURAL, Matemática.

Lenguajes de quinta generación (5GL): La intención es que el programador establezca el qué problema ha de ser resuelto y las condiciones a reunir, y la máquina lo resuelve. Se usan en inteligencia artificial. Ejemplo: Prolog.

4. Manera de ejecutarse.

Según la manera de ejecutarse:

Lenguajes compilados: Un programa traductor traduce el código del programa (código fuente) en código máquina (código objeto). Otro programa, el enlazador, unirá los ficheros de código objeto del programa principal con los de las librerías para producir el programa ejecutable. Ejemplo: C.

Lenguajes interpretados: Un programa (intérprete), ejecuta las instrucciones del programa de manera directa. Ejemplo: Lisp.

También los hay mixtos, como Java, que primero pasan por una fase de compilación en la que el código fuente se transforma en "bytecode", y este "bytecode" puede ser ejecutado luego (interpretado) en ordenadores con distintas arquitecturas (procesadores) que tengan todos instalados la misma "máquina virtual" Java.

5. Manera de abordar la tarea a realizar.

Según la manera de abordar la tarea a realizar, pueden ser:

Lenguajes imperativos: Indican cómo hay que hacer la tarea, es decir, expresan los pasos a realizar. Ejemplo: C.

Lenguajes declarativos: Indican qué hay que hacer. Ejemplos: Lisp, Prolog. Otros ejemplos de lenguajes declarativos, pero que no son lenguajes de programación, son HTML (para describir páginas web) o SQL (para consultar bases de datos).

6. Paradigma de programación.

El paradigma de programación es el estilo de programación empleado. Algunos lenguajes soportan varios paradigmas, y otros sólo uno. Se puede decir que históricamente han ido apareciendo para facilitar la tarea de programar según el tipo de problema a abordar, o para facilitar el mantenimiento del software, o por otra cuestión similar, por lo que todos corresponden a lenguajes de alto nivel (o nivel medio), estando los

lenguajes ensambladores “atados” a la arquitectura de su procesador correspondiente. Los principales son:

Lenguajes de programación procedural: Divide el problema en partes más pequeñas, que serán realizadas por subprogramas (subrutinas, funciones, procedimientos), que se llaman unas a otras para ser ejecutadas. Ejemplos: C, Pascal.

Lenguajes de programación orientada a objetos: Crean un sistema de clases y objetos siguiendo el ejemplo del mundo real, en el que unos objetos realizan acciones y se comunican con otros objetos. Ejemplos: C++, Java.

Lenguajes de programación funcional: La tarea se realiza evaluando funciones, (como en Matemáticas), de manera recursiva. Ejemplo: Lisp.

Lenguajes de programación lógica: La tarea a realizar se expresa empleando lógica formal matemática. Expresa qué computar. Ejemplo: Prolog.

Hay muchos paradigmas de programación: Programación genérica, programación reflexiva, programación orientada a procesos, etc.

7. Lugar de ejecución.

En sistemas distribuidos, según dónde se ejecute:

Lenguajes de servidor: Se ejecutan en el servidor. Ejemplo: PHP es el más utilizado en servidores web.

Lenguajes de cliente: Se ejecutan en el cliente. Ejemplo: JavaScript en navegadores web.

8. Concurrencia.

Según admitan o no concurrencia de procesos, esto es, la ejecución simultánea de varios procesos lanzados por el programa:

Lenguajes concurrentes: Ejemplo: Ada.

Lenguajes no concurrentes: Ejemplo: C.

9. Interactividad.

Según la interactividad del programa con el usuario u otros programas:

Lenguajes orientados a sucesos: El flujo del programa es controlado por la interacción con el usuario o por mensajes de otros programas/sistema operativo, como editores de texto, interfaces gráficos de usuario (GUI) o kernels. Ejemplo: VisualBasic, lenguajes de programación declarativos.

Lenguajes no orientados a sucesos: El flujo del programa no depende de sucesos exteriores, sino que se conoce de antemano, siendo los procesos batch el ejemplo más claro (actualizaciones de bases de datos, colas de impresión de documentos, etc.). Ejemplos: Lenguajes de programación imperativos.

10. Realización visual.

Según la realización visual o no del programa:

Lenguajes de programación visual: El programa se realiza moviendo bloques de construcción de programas (objetos visuales) en un interfaz adecuado para ello. No confundir con entornos de programación visual, como Microsoft Visual Studio y sus lenguajes de programación textuales (como Visual C#). Ejemplo: Mindscript.

Lenguajes de programación textual: El código del programa se realiza escribiéndolo. Ejemplos: C, Java, Lisp.

11. Determinismo.

Según se pueda predecir o no el siguiente estado del programa a partir del estado actual:

Lenguajes deterministas: Ejemplos: Todos los anteriores.

Lenguajes probabilísticos o no deterministas: Sirven para explorar grandes espacios de búsqueda, (como gramáticas), y en la investigación teórica de hipercomputación. Ejemplo: mutt (generador de texto aleatorio).

12. Productividad.

Según se caractericen por tener virtudes útiles o productivas, u oscuras y enrevesadas:

Lenguajes útiles o productivos: Sus virtudes en cuanto a eficiencia, sencillez, claridad, productividad, etc., motiva que sean utilizados en

empresas, administraciones públicas y/o en la enseñanza. Ejemplos: Cualquier lenguaje de uso habitual (C, Java, C++, Lisp, Python, Ruby,…).

Lenguajes esotéricos o exóticos: Inventados con la intención de ser los más raros, oscuros, difíciles, simples y/o retorcidos de los lenguajes, para diversión y entretenimiento de frikis programadores. A veces exploran nuevas ideas en programación. Ejemplo: Brainfuck.

3. ¿Qué son los lenguajes de alto, medio y bajo nivel?, De tres ejemplos de cada uno.

Lenguaje del Alto Nivel: Un lenguaje de programación de alto nivel se caracteriza por expresar los algoritmos 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.

Fortran: Abreviatura de Fórmula Translator (traductor de fórmulas), fue definido alrededor del año 1954, y disponible para el público en 1957 en los Estados Unidos por la compañía IBM.1 Es el más antiguo de los lenguajes de alto nivel, pues antes de su aparición todos los programas se escribían en lenguaje ensamblador o en lenguaje máquina.

Es un lenguaje especializado en aplicaciones técnicas y científicas, caracterizándose por su potencia en los cálculos matemáticos, pero estando limitado en las aplicaciones de gestión, manejo de archivos, tratamiento de cadenas de caracteres y edición de informes.

A lo largo de su existencia han aparecido diferentes versiones, entre las que destaca la realizada en 1966 por ANSI (American National Standard Institute) en la que se definieron nuevas reglas del lenguaje y se efectuó la independencia del mismo con respecto a la "máquina", es decir, comenzó la transportabilidad del lenguaje. Esta versión se denominó FORTRAN IV o FORTRAN 66. En 1977, apareció una nueva versión más evolucionada que se llamó FORTRAN V o FORTRAN 77, esta versión está reflejada en el documento «ANSI X3.9-1978: Programming Language FORTRAN» y define dos niveles del lenguaje denominados FORTRAN 77 completo y FORTRAN 77 básico, siendo el segundo un subconjunto del primero. Esta última versión incluye además instrucciones para el manejo de cadenas de caracteres y de archivos, así como otras para la utilización de técnicas de programación estructurada. Estas características hacen que el lenguaje también sea válido para determinadas aplicaciones de gestión.

Cobol: Es el lenguaje más usado en las aplicaciones de gestión, creado en 1960 por un comité denominado CODASYL, patrocinado por el Departamento de Defensa de los Estados Unidos, a fin de disponer de un lenguaje universal para aplicaciones comerciales como expresa su nombre (COmmon Business Oriented Language). Entre sus características se pueden citar su parecido al lenguaje natural (inglés), es auto- documentado y tiene gran capacidad en el manejo de archivos, así como en la edición de informes escritos. Entre sus inconvenientes están sus rígidas reglas de formatos de escritura, la necesidad de describir todos los elementos al máximo detalle, la extensión excesiva en sus sentencias e incluso duplicación en algunos casos, la inexistencia de funciones matemáticas y, por último, su no adecuación a las técnicas de programación estructurada.

Pascal: Fue creado por el matemático suizo Niklaus Wirth en 1970, basándose en el lenguaje AL-GOL, en cuyo diseño había participado en los años 60. Su nombre proviene del filósofo y matemático francés del siglo xvii Blaise Pascal, que inventó la primera máquina de tipo mecánico para sumar.

Aunque en principio la idea del diseñador era proporcionar un lenguaje adecuado para la enseñanza de los conceptos y técnicas de programación, con el tiempo ha llegado a ser un lenguaje ampliamente utilizado en todo tipo de aplicaciones, poseyendo grandes facilidades para la programación de sistemas y diseño de gráficos.

Aporta los conceptos de tipo de datos, programación estructurada y diseño descendente, entre otros, además de haberse convertido en predecesor de otros lenguajes más modernos, como MODULA-2 y ADA.

Lenguaje de Medio Nivel: Lenguaje de medio nivel es un lenguaje de programación informática como el lenguaje C, que se encuentran entre los lenguajes de alto nivel y los lenguajes de bajo nivel.

Suelen ser clasificados muchas veces de alto nivel, pero permiten ciertos manejos de bajo nivel. Son precisos para ciertas aplicaciones como la creación de sistemas operativos, ya que permiten un manejo abstracto (independiente de la máquina, a diferencia del ensamblador), pero sin perder mucho del poder y eficiencia que tienen los lenguajes de bajo nivel.

Una característica distintiva, por ejemplo, que convierte a C en un lenguaje de medio nivel y al Pascal en un lenguaje de alto nivel es que en el primero es posible manejar las letras como si fueran números (en Pascal no), y por el contrario en Pascal es posible concatenar las cadenas de caracteres con el operador suma y copiarlas con la asignación (en C es el usuario el responsable de llamar a las funciones correspondientes).

Una de las características más peculiares del lenguaje de programación C; es el uso de "apuntadores", los cuales son muy útiles en la implementación de algoritmos como Listas ligadas, Tablas Hash y algoritmos de búsqueda y

ordenamiento que para otros lenguajes de programación (como Java por ejemplo) les suele ser un poco más complicado implementar.

BCPL: Es la sigla en inglés de *Basic Combined Programming Language* (Lenguaje de Programación Básico Combinado). Fue diseñado por Martin Richards de la Universidad en 1966 debido a las dificultades experimentadas con el lenguaje de programación CPL durante los años 60. El primer compilador implementado fue escrito en 1967 mientras Richards visitaba el MIT. El lenguaje fue descrito por primera vez en un proyecto presentado en una conferencia informática en 1969. Años después, Dennis Ritchie y Ken Thompson lo utilizaron como base para desarrollar B (que a su vez, más tarde daría lugar al popular lenguaje de programación C).

Es un lenguaje de programación ordenado, potente y muy fácil de adaptar a diferentes arquitecturas. Se popularizó en los programas de arranque de las computadoras (*bootstraps* en inglés) debido a sus compiladores simples y compactos, algunos con capacidad para correr en sólo 16 kilobytes. Inclusive algunos sistemas fueron escritos total o parcialmente en BCPL (TRIPOS y Amiga Kickstart entre otros).

La principal razón de la capacidad de adaptación a las diferentes arquitecturas es la estructura de su compilador, el que fue dividido en dos partes. La cara visible del mismo interpretaba el código fuente y generaba código máquina para una máquina virtual; la otra cara del compilador tomaba dicho código máquina y lo traducía al código necesario para la arquitectura deseada. No mucho después, este diseño de compiladores se hizo popular; pero el compilador de Richards fue el primero en definir una máquina virtual para este propósito. Algunos de los lenguajes que utilizan el mismo formato son Java y Pascal.

C: C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de

programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos. La primera estandarización del lenguaje C fue en ANSI, con el estándar X3.159-1989. El lenguaje que define este estándar fue conocido vulgarmente como ANSI C. Posteriormente, en 1990, fue ratificado como estándar ISO (ISO/IEC 9899:1990). La adopción de este estándar es muy amplia por lo que, si los programas creados lo siguen, el código es portátil entre plataformas y/o arquitecturas.

Visual Basic: Es un lenguaje de programación dirigido por eventos, desarrollado por Alan Cooper para Microsoft. Este lenguaje de programación es un dialecto de BASIC, con importantes agregados. Su primera versión fue presentada en 1991, con la intención de simplificar la programación utilizando un ambiente de desarrollo que facilitó en cierta medida la programación misma.La última versión fue la 6, liberada en 1998, para la que Microsoft extendió el soporte hasta marzo de 2008. En 2001 Microsoft propuso abandonar el desarrollo basado en la API Win32 y pasar a un framework o marco común de librerías, independiente de la versión del sistema operativo, .NET Framework, a través de Visual Basic .NET (y otros lenguajes como C Sharp (C#) de fácil transición de código entre ellos); fue el sucesor de Visual Basic 6. Aunque Visual Basic es de propósito general, también provee facilidades para el desarrollo de aplicaciones de bases de datos usando Data Access Objects, Remote Data Objects o ActiveX Data Objects. Visual Basic contiene un entorno de desarrollo integrado o IDE que

integra editor de textos para edición del código fuente, un depurador, un compilador (y enlazador) y un editor de interfaces gráficas o GUI.

Lenguaje de nivel bajo: Un lenguaje de programación de bajo nivel es aquel en el que sus instrucciones ejercen un control directo sobre el hardware y están condicionados por la estructura física de la computadora que lo soporta. El uso de la palabra bajo en su denominación no implica que el lenguaje sea inferior a un lenguaje de alto nivel, si no 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.

El Lenguaje Ensamblador: *Assembler* (*assembly language* en inglés), es un lenguaje de programación de bajo nivel para los computadores,microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código específico para cada arquitectura legible por un programador. Esta representación es usualmente definida por el fabricante de hardware, y está basada en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portátiles.

El Código Binario: Es el sistema numérico usado para la de representación de textos, o procesadores de instrucciones de computadora utilizando el sistema binario (sistema numérico de dos dígitos, o bit: el "0" (cerrado) y el "1" (abierto)). En informática y telecomunicaciones, el código binario se utiliza con variados métodos de codificación de datos, tales como cadenas de caracteres, o cadenas de bits. Estos métodos pueden ser de ancho fijo o ancho variable. Por ejemplo en el caso de un CD, las señales

que reflejarán el "láser" que rebotará en el CD y será recepcionado por un sensor de distinta forma indicando así, si es un cero o un uno.

En un código binario de ancho fijo, cada letra, dígito, u otros símbolos, están representados por una cadena de bits de la misma longitud, como un número binario que, por lo general, aparece en las tablas en notación octal, decimal o hexadecimal.

Según Anton Glaser, en su History of Binary and other Nondecimal Numeration, comenta que los primeros códigos binarios se utilizaron en el año 1932: C.E. Wynn-Williams ("Scale of Two"), posteriormente en 1938: Atanasoff-Berry Computer, y en 1939: Stibitz ("excess three") el código en Complex Computer.

Es frecuente también ver la palabra bit referida bien a la ausencia de señal, expresada con el dígito "0", o bien referida a la existencia de la misma, expresada con el dígito "1". El byte es un grupo de 8 bits, es decir en él tenemos 256 posibles estados binarios.

4. Hacer un cuadro comparativo que permita observar ventajas y desventajas de los lenguajes alto, medio y bajo.

Aspectos	Lenguaje de Alto Nivel	Lenguaje de Medio Nivel	Lenguaje de Bajo Nivel
Ventajas	Mayor adaptación al equipo. Posibilidad de obtener la máxima velocidad con mínimo uso de memoria. Se trabaja a nivel de instrucciones, es decir, su programación es al más fino detalle. Está orientado a la máquina.	Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas. Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado "no llevado al extremo" (permitiendo ciertas licencias de ruptura). Un sistema de tipos que impide operaciones sin sentido. Usa un lenguaje de preprocesado, el preprocesador de C, para tareas como definir macros e incluir múltiples archivos de código fuente. Acceso a memoria de bajo nivel mediante el uso de punteros. Interrupciones al procesador con uniones. Un conjunto reducido de palabras clave. Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros. Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo. Tipos de datos agregados (struct) que permiten que datos relacionados (como un empleado, que tiene un id, un nombre y un salario) se combinen y se manipulen como un todo (en una única variable "empleado"	La programación en un lenguaje de alto nivel tiene ciertas ventajas: Genera un código más sencillo y comprensible. Escribir un código válido para diversas máquinas y, posiblemente, sistemas operativos. Lograr independencia de la máquina, pudiendo utilizar un mismo programa en diferentes equipos con la única condición de disponer de un programa traductor o compilador, que lo suministra el fabricante, para obtener el programa ejecutable en lenguaje binario de la máquina que se trate. Además, no se necesita conocer el hardware específico de dicha máquina.
Desventajas	Imposibilidad de escribir código independiente de la máquina. Mayor dificultad en la programación y en la comprensión de los programas. El programador debe conocer más de un	Recolección de basura nativa, sin embargo se encuentran a tal efecto bibliotecas como la "libgc" desarrollada por Sun Microsystems, o el Recolector de basura de Boehm. Soporte para programación orientada a objetos, aunque la	Reducción de velocidad al ceder el trabajo de bajo nivel a la máquina. Algunos requieren que la máquina cliente posea una determinada plataforma.

	centenar de instrucciones. Es necesario conocer en detalle la arquitectura de la máquina.	implementación original de C++ fue un preprocesador que traducía código fuente de C++ a C. Encapsulación. Funciones anidadas, aunque GCC tiene esta característica como extensión. Polimorfismo en tiempo de código en forma de sobrecarga, sobrecarga de operadores y sólo dispone de un soporte rudimentario para la programación genérica. Soporte nativo para programación multihilo y redes de computadores.	

5. ¿Qué son las palabras reservadas en programación?

En los lenguajes informáticos, una palabra reservada es una palabra que tiene un significado gramatical especial para ese lenguaje y no puede ser utilizada como un identificador de objetos en códigos del mismo, como ser variables. Por ejemplo, en SQL, un usuario no puede ser llamado "group", porque la palabra group es usada para indicar que un identificador se refiere a un grupo, no a un usuario. Al tratarse de una palabra clave su uso queda restringido. Ocasionalmente la especificación de un lenguaje de programación puede tener palabras reservadas que están previstas para un posible uso en futuras versiones. En Java const y goto son palabras reservadas — no tienen significado en Java, pero tampoco pueden ser usadas como identificadores. Al reservar los términos pueden ser implementados en futuras versiones de Java, si se desea, sin que el código fuente más antiguo escrito en Java deje de funcionar.

Palabras reservadas e independencia del lenguaje

En la CLI de .NET, todos los lenguajes tienen que proporcionar un mecanismo para utilizar los identificadores públicos que son palabras reservadas en ese lenguaje. .|. Para ver por qué es necesario, supongamos que se define una clase enVB.NET como sigue:

```
Public Class this
End Class
```

Entonces, se compila esta clase en un ensamblado de .NET y se distribuye como parte de un conjunto de herramientas. Un programador de C#, que quiere definir una variable de tipo “this” encontraría un problema: “this” es una palabra reservada en C#. El siguiente fragmento en C# no compilará:

```
this x = new this();
```

Un tema similar aparece cuando se accede a miembros, sobrescribiendo métodos virtuales e identificando espacios de nombres. En C#, colocando

la arroba (@) antes del identificador, se forzará a ser considerado como un identificador en vez de una palabra reservada por el compilador. El signo arroba no es considerado parte del identificador.
@this x = new @this();

Por consistencia, esta utilización también se permite en configuraciones no-públicas como variables locales, nombres de parámetros y miembros privados.

Palabras reservadas en SQL

En SQL, son palabras reservadas todas las sentencias, cláusulas modificadoras, tipos de dato, y funciones propias delDBMS.Así, por ejemplo, no se pueden usar denominaciones tales como SELECT, GROUP, CONCAT, SUM, MAX o semejantes. En cada DBMS, además, existen conjuntos de denominaciones que le son propias, y que si pueden ser usadas en otros, lo que tiende a producir ciertos problemas al migrar proceso o sintaxis de uno a otro. Tal es el caso de SYSDATE, NOW, o BIGINT, el primero de los cuales es usual en Oracle, y los dos siguientes en MySQL. Pese a eso, todos los DBMS tienen, también, la posibilidad de usar estas palabras reservadas, siempre que se respete ciertas reglas sintácticas: Todas las palabras reservadas usadas para nombres de objetos creados en una base de datos deben estar encerrados entre caracteres específicamente usados para ello. En varios sistemas de bases de datos se utilizan para ello los acentos graves (`), que no tienen otro uso en el SQL.

Programación

La programación informática, a menudo acortada como programación, es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas computacionales. 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.

Historia

Para crear un programa, y que la computadora lo intérprete y ejecute las instrucciones escritas en él, debe usarse un lenguaje de programación. En sus inicios las computadoras interpretaban sólo instrucciones en un lenguaje específico, del más bajo nivel, conocido como código máquina, siendo éste excesivamente complicado para programar. De hecho sólo 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 letras 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 usa 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 un lenguaje de alto nivel, es necesario compilarlo, es decir, traducirlo a lenguaje máquina.1

Léxico y programación

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.

Programas y algoritmos

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación concreto) uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un diagrama de flujo, en forma de código como en pseudocódigo o un lenguaje de programación, en forma explicativa, etc.

Los programas suelen subdividirse en partes menores, llamadas módulos, de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa. Esta es una práctica muy utilizada y se conoce como "refino progresivo".

Según Niklaus Wirth, un programa está formado por los algoritmos y la estructura de datos.

Se han propuesto diversas técnicas de programación cuyo objetivo es mejorar tanto el proceso de creación de softwarecomo su mantenimiento. Entre ellas, se pueden mencionar las siguientes:

•programación declarativa
•programación estructurada
•programación modular
•programación orientada a objetos

Compilación

El programa escrito en un lenguaje de programación (fácilmente comprensible por el programador) 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 en lenguaje máquina, código esté directamente ejecutable por el procesador. A este proceso se le llama compilación.

Habitualmente la creación de un programa ejecutable (un típico.exe para Microsoft Windows o DOS) conlleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina). El segundo paso se llama enlazado en el cual se enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

Estos dos pasos se pueden hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico.obj para Microsoft Windows, DOS o para Unix); para enlazarlos en fases posteriores, o crear directamente el ejecutable; con lo que la fase de compilación se almacena sólo temporalmente. Un programa podría tener partes escritas en varios lenguajes, por ejemplo, Java, C, C++ y ensamblador, que se podrían compilar de forma independiente y luego enlazar juntas para formar un único módulo ejecutable.

Programación e ingeniería del software

Existe una tendencia a identificar el proceso de creación de un programa informático con la programación, que es cierta cuando se trata de

programas pequeños para uso personal, y que dista de la realidad cuando se trata de grandes proyectos.

El proceso de creación de software, desde el punto de vista de la ingeniería, incluye los siguientes pasos:

1. Reconocer la necesidad de un programa para solucionar un problema o identificar la posibilidad de automatización de una tarea.
2. Recoger los requisitos del programa. Debe quedar claro qué es lo que debe hacer el programa y para qué se necesita.
3. Realizar el análisis de los requisitos del programa. Debe quedar claro cómo debe realizar el programa las cosas que debe hacer. Las pruebas que comprueben la validez del programa se pueden especificar en esta fase.
4. Diseñar la arquitectura del programa. Se debe descomponer el programa en partes de complejidad abordable.
5. Implementar el programa. Consiste en realizar un diseño detallado, especificando completamente todo el funcionamiento del programa, tras lo cual la codificación (programación propiamente dicha) debería resultar inmediata.
6. Implantar (instalar) el programa. Consiste en poner el programa en funcionamiento junto con los componentes que pueda necesitar (bases de datos, redes de comunicaciones, etc.).

La ingeniería del software se centra en los pasos de planificación y diseño del programa, mientras que antiguamente (programación artesanal) la realización de un programa consistía casi únicamente en escribir el código, bajo sólo el conocimiento de los requisitos y con una modesta fase de análisis y diseño.

Referencias históricas

La primera programadora de computadoras conocida fue Ada Lovelace, hija de Anabella Milbanke Byron y Lord Byron. Anabella introdujo en las matemáticas a Ada quien, después de conocer a Charles Babbage, tradujo y amplió una descripción de su máquina analítica. Incluso, aunque Babbage nunca completó la construcción de cualquiera de sus máquinas, el trabajo

que Ada realizó con éstas le hizo ganarse el título de primera programadora de computadoras del mundo. El nombre del lenguaje de programación Ada fue escogido como homenaje a esta programadora.

Este proceso está aplicado a todos los métodos científicos que actualmente se practican.

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:

•Corrección. 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 desarrollarlo 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 así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; 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 programadores puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones de código. Otros, por diversión o para impedir un análisis cómodo a otros programadores, recurren al uso decódigo 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 al obtener la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico de 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 aquélla en la que se elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha desarrollado para sistemas GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

Ciclo de vida del software

Artículo principal: Proceso para el desarrollo de software

El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final. El propósito de este programa es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura de que los métodos utilizados son apropiados. Estos programas se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación. El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados. El ciclo de vida básico de un software consta de los siguientes procedimientos:

•Definición de objetivos: definir el resultado del proyecto y su papel en la estrategia global.

•Análisis de los requisitos y su viabilidad: recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.

•Diseño general: requisitos generales de la arquitectura de la aplicación.

•Diseño en detalle: definición precisa de cada subconjunto de la aplicación.

•Programación (programación e implementación): es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.
•Prueba de unidad: prueba individual de cada subconjunto de la aplicación para garantizar que se implementaron de acuerdo con las especificaciones.
•Integración: para garantizar que los diferentes módulos se integren con la aplicación. Éste es el propósito de la prueba de integración que está cuidadosamente documentada.
•Prueba beta (o validación), para garantizar que el software cumple con las especificaciones originales.
•Documentación: sirve para documentar información necesaria para los usuarios del software y para desarrollos futuros.
•Mantenimiento: para todos los procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del software (mantenimiento continuo).
El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores.

6. ¿Qué es la sintaxis de un programa o lenguaje de programación?

Sintaxis

La sintaxis es la parte de la gramática que estudia las reglas y principios que gobiernan la combinatoria de constituyentes sintácticos y la formación de unidades superiores a estos, como los sintagmas y oraciones gramaticales. La sintaxis, por tanto, estudia las formas en que se combinan las palabras, así como las relaciones sintagmáticas y paradigmáticas existentes entre ellas.

Lenguaje de programación

Captura de la microcomputadora Commodore PET-32 mostrando un programa en el lenguaje de programación BASIC, bajo el emulador VICE en una distribución GNU/Linux. Un ejemplo de código fuente escrito en el lenguaje de programación Java, que imprimirá el mensaje "Hello World!" a la salida estándar cuando es compilado y ejecutado

Un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras.

Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

Está formado por un conjunto de símbolos y reglas sintácticas ysemánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depurase compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:

•El desarrollo lógico del programa para resolver un problema en particular.

•Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).

•Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.

•Prueba y depuración del programa.

•Desarrollo de la documentación.

Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML (lenguaje para el marcado de páginas web que no es propiamente un

lenguaje de programación, sino un conjunto de instrucciones que permiten estructurar el contenido de los documentos).

Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

operación define tipos de datos para los cuales la operación es aplicable, con la implicación de que no es aplicable a otros tipos. Por ejemplo, "este texto entre comillas" es una cadena. En la mayoría de los lenguajes de programación, dividir un número por una cadena no tiene ningún significado. Por tanto, la mayoría de los lenguajes de programación modernos rechazaran cualquier intento de ejecutar dicha operación por parte de algún programa. En algunos lenguajes, estas operaciones sin significado son detectadas cuando el programa es compilado (validación de tipos "estática") y son rechazadas por el compilador, mientras en otros son detectadas cuando el programa es ejecutado (validación de tipos "dinámica") y se genera una excepción en tiempo de ejecución.

Un caso especial de lenguajes de tipo son los lenguajes de tipo sencillo. Estos son con frecuencia lenguajes de marcado o de scripts, como REXX o SGML, y solamente cuentan con un tipo de datos; comúnmente cadenas de caracteres que luego son usadas tanto para datos numéricos como simbólicos.

En contraste, un lenguaje sin tipos, como la mayoría de los lenguajes ensambladores, permiten que cualquier operación se aplique a cualquier dato, que por lo general se consideren secuencias de bits de varias longitudes. Lenguajes de alto nivelsin datos incluyen BCPL y algunas variedades de Forth.

En la práctica, aunque pocos lenguajes son considerados con tipo desde el punto de vista de la teoría de tipos (es decir, que verifican o rechazan todas las operaciones), la mayoría de los lenguajes modernos ofrecen algún grado de manejo de tipos. Si bien muchos Para que la computadora entienda nuestras instrucciones debe usarse un lenguaje específico conocido como código máquina, el cual la máquina comprende fácilmente, pero que lo hace excesivamente complicado para las personas. De hecho sólo consiste en cadenas extensas de números 0 y 1.

Para facilitar el trabajo, los primeros operadores de computadoras decidieron hacer un traductor para reemplazar los 0 y 1 por palabras o abstracción de palabras y letras provenientes del inglés; éste se conoce como lenguaje ensamblador. Por ejemplo, para sumar se usa la letra A de la palabra inglesa add (sumar). El lenguaje ensamblador sigue la misma estructura del lenguaje máquina, pero las letras y palabras son más fáciles de recordar y entender que los números.

La necesidad de recordar secuencias de programación para las acciones usuales llevó a denominarlas con nombres fáciles de memorizar y asociar: ADD (sumar), SUB (restar), MUL (multiplicar), CALL (ejecutar subrutina), etc. A esta secuencia de posiciones se le denominó "instrucciones", y a este conjunto de instrucciones se le llamó lenguaje ensamblador. Posteriormente aparecieron diferentes lenguajes de programación, los cuales reciben su denominación porque tienen una estructura sintáctica similar a los lenguajes escritos por los humanos, denominados también lenguajes de alto nivel.

La primera programadora de computadora conocida fue Ada Lovelace, hija de Anabella Milbanke Byron y Lord Byron. Anabella introdujo en las matemáticas a Ada quien, después de conocer a Charles Babbage, tradujo y amplió una descripción de su máquina analítica. Incluso aunque Babbage nunca completó la construcción de cualquiera de sus máquinas, el trabajo que Ada realizó con éstas le hizo ganarse el título de primera programadora

de computadoras del mundo. El nombre del lenguaje de programación Ada fue escogido como homenaje a esta programadora.

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

El primer manual para el lenguaje Fortran apareció en octubre de 1956, con el primer compilador Fortran entregado enabril de 1957. Esto era un compilador optimizado, porque los clientes eran reacios a usar un lenguaje 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.

En 1960, se creó COBOL, uno de los lenguajes usados aún en la actualidad, en informática de gestión.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método más eficiente para programarlas. Entonces, se crearon los lenguajes de alto nivel, como lo fue BASIC en las versiones introducidas en los microordenadores de la década de 1980. Mientras que una tarea tan sencilla como sumar dos números puede necesitar varias instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará una sola sentencia.

Elementos

Variables y vectores

Las variables podrían calificarse como contenedores de datos y por ello se diferencian según el tipo de dato que son capaces de almacenar. En la mayoría de lenguajes de programación se requiere especificar un tipo de variable concreto para guardar un dato concreto. Por ejemplo, en Java, si

deseamos guardar una cadena de texto deberemos especificar que la variable es del tipo String. Por otra parte, en lenguajes como el PHP este tipo de especificación de variables no es necesario. Además, existen variables compuestas por varias variables llamadas vectores. Un vector no es más que un conjunto de variables consecutivas en memoria y del mismo tipo guardadas dentro de una variable contenedora. A continuación, un listado con los tipos de variables y vectores más comunes:

• Variables tipo Char: Estas variables contienen un único carácter, es decir, una letra, un signo o un número.

• Variables tipo Int: Contienen un número entero.

• Variables tipo float: Contienen un número decimal.

• Variables tipo String: Contienen cadenas de texto, o lo que es lo mismo, es un vector con varias variables del tipo Char.

• Variables del tipo Boolean: Solo pueden contener un 0 o un 1. El cero es considerado para muchos lenguajes como el literal "False", mientras que el 1 se considera "True".

Condicionantes

Los condicionantes son estructuras de código que indican que, para que cierta parte del programa se ejecute, deben cumplirse ciertas premisas; por ejemplo: que dos valores sean iguales, que un valor exista, que un valor sea mayor que otro... Estos condicionantes por lo general solo se ejecutan una vez a lo largo del programa. Los condicionantes más conocidos y empleados en programación son:

• If: Indica una condición para que se ejecute una parte del programa.

• Else if: Siempre va precedido de un "If" e indica una condición para que se ejecute una parte del programa siempre que no cumpla la condición del if previo y si se cumpla con la que el "else if" especifique.

• Else: Siempre precedido de "If" y en ocasiones de "Else If". Indica que debe ejecutarse cuando no se cumplan las condiciones prévias.

Bucles[editar]

Los bucles son parientes cercanos de los condicionantes, pero ejecutan constantemente un código mientras se cumpla una determinada condición. Los más frecuentes son:

• For: Ejecuta un código mientras una variable se encuentre entre 2 determinados parámetros.

• While: Ejecuta un código mientras se cumpla la condición que solicita.

Hay que decir que a pesar de que existan distintos tipos de bucles, ambos son capaces de realizar exactamente las mismas funciones. El empleo de uno u otro depende, por lo general, del gusto del programador.

Funciones

Las funciones se crearon para evitar tener que repetir constantemente fragmentos de código. Una función podría considerarse como una variable que encierra código dentro de sí. Por lo tanto cuando accedemos a dicha variable (la función) en realidad lo que estamos es diciendo al programa que ejecute un determinado código predefinido anteriormente.

Todos los lenguajes de programación tienen algunos elementos de formación primitivos para la descripción de los datos y de los procesos o transformaciones aplicadas a estos datos (tal como la suma de dos números o la selección de un elemento que forma parte de una colección). Estos elementos primitivos son definidos por reglas sintácticas y semánticas que describen su estructura y significado respectivamente.

Sintaxis

Con frecuencia se resaltan los elementos de la sintaxis con colores diferentes para facilitar su lectura. Este ejemplo está escrito en Python.

A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos. Por otra parte, hay algunos lenguajes de programación que son más

gráficos en su naturaleza, utilizando relaciones visuales entre símbolos para especificar un programa.

La sintaxis de un lenguaje de programación describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto. El significado que se le da a una combinación de símbolos es manejado por su semántica (ya sea formal o como parte del código duro de la referencia de implementación). Dado que la mayoría de los lenguajes son textuales, este artículo trata de la sintaxis textual.

La sintaxis de los lenguajes de programación es definida generalmente utilizando una combinación de expresiones regulares (para la estructura léxica) y la Notación de Backus-Naur (para la estructura gramática). Este es un ejemplo de una gramática simple, tomada de Lisp:

expresión ::= átomo | lista
átomo ::= número | símbolo
número ::= [+-]? ['0'-'9']+
símbolo ::= ['A'-'Z'] ['a'-'z'].*
lista ::= '(' expresión* ')'

Con esta gramática se especifica lo siguiente:

• Una expresión puede ser un átomo o una lista;
• Un átomo puede ser un número o un símbolo;
• Un número es una secuencia continua de uno o más dígitos decimales, precedido opcionalmente por un signo más o un signo menos;
• Un símbolo es una letra seguida de cero o más caracteres (excluyendo espacios); y
• Una lista es un par de paréntesis que abren y cierran, con cero o más expresiones en medio.

Algunos ejemplos de secuencias bien formadas de acuerdo a esta gramática:

'12345', '()', '(a b c232 (1))'

No todos los programas sintácticamente correctos son semánticamente correctos. Muchos programas sintácticamente correctos tienen

inconsistencias con las reglas del lenguaje; y pueden (dependiendo de la especificación del lenguaje y la solidez de la implementación) resultar en un error de traducción o ejecución. En algunos casos, tales programas pueden exhibir un comportamiento indefinido. Además, incluso cuando un programa está bien definido dentro de un lenguaje, todavía puede tener un significado que no es el que la persona que lo escribió estaba tratando de construir.

Usando el lenguaje natural, por ejemplo, puede no ser posible asignarle significado a una oración gramaticalmente válida o la oración puede ser falsa:

•"Las ideas verdes y descoloridas duermen furiosamente" es una oración bien formada gramaticalmente pero no tiene significado comúnmente aceptado.

•"Juan es un soltero casado" también está bien formada gramaticalmente pero expresa un significado que no puede ser verdadero.

El siguiente fragmento en el lenguaje C es sintácticamente correcto, pero ejecuta una operación que no está definida semánticamente (dado que p es un apuntador nulo, las operaciones p->real y p->im no tienen ningún significado):

```
complex *p = NULL;
complex abs_p = sqrt (p->real * p->real + p->im * p->im);
```

Si la declaración de tipo de la primera línea fuera omitida, el programa dispararía un error de compilación, pues la variable "p" no estaría definida. Pero el programa sería sintácticamente correcto todavía, dado que las declaraciones de tipo proveen información semántica solamente.

La gramática necesaria para especificar un lenguaje de programación puede ser clasificada por su posición en la Jerarquía de Chomsky. La sintaxis de la mayoría de los lenguajes de programación puede ser especificada utilizando una gramática Tipo-2, es decir, son gramáticas libres de contexto. Algunos lenguajes, incluyendo a Perl y a Lisp, contienen construcciones que permiten la ejecución durante la fase de análisis. Los lenguajes que permiten construcciones que permiten al programador alterar

el comportamiento de un analizador hacen del análisis de la sintaxis un problema sin decisión única, y generalmente oscurecen la separación entre análisis y ejecución. En contraste con el sistema de macros de Lisp y los bloques BEGIN de Perl, que pueden tener cálculos generales, las macros de C son meros reemplazos de cadenas, y no requieren ejecución de código.

Semántica estática

La semántica estática define las restricciones sobre la estructura de los textos válidos que resulta imposible o muy difícil expresar mediante formalismos sintácticos estándar. Para los lenguajes compilados, la semántica estática básicamente incluye las reglas semánticas que se pueden verificar en el momento de compilar. Por ejemplo el chequeo de que cada identificador sea declarado antes de ser usado (en lenguajes que requieren tales declaraciones) o que las etiquetas en cada brazo de una estructura case sean distintas. Muchas restricciones importantes de este tipo, como la validación de que los identificadores sean usados en los contextos apropiados (por ejemplo no sumar un entero al nombre de una función), o que las llamadas a subrutinas tengan el número y tipo de parámetros adecuado, puede ser implementadas definiéndolas como reglas en una lógica conocida como sistema de tipos. Otras formas de análisis estáticos, como los análisis de flujo de datos, también pueden ser parte de la semántica estática. Otros lenguajes de programación como Java y C# tienen un análisis definido de asignaciones, una forma de análisis de flujo de datos, como parte de su semántica estática.

Sistema de tipos

Artículo principal: Sistema de tipos

Un sistema de tipos define la manera en la cual un lenguaje de programación clasifica los valores y expresiones en tipos, cómo pueden ser manipulados dichos tipos y cómo interactúan. El objetivo de un sistema de

tipos es verificar y normalmente poner en vigor un cierto nivel de exactitud en programas escritos en el lenguaje en cuestión, detectando ciertas operaciones inválidas. Cualquier sistema de tipos decidible tiene sus ventajas y desventajas: mientras por un lado rechaza muchos programas incorrectos, también prohíbe algunos programas correctos aunque poco comunes. Para poder minimizar esta desventaja, algunos lenguajes incluyen lagunas de tipos, conversiones explícitas no checadas que pueden ser usadas por el programador para permitir explícitamente una operación normalmente no permitida entre diferentes tipos. En la mayoría de los lenguajes con tipos, el sistema de tipos es usado solamente para checar los tipos de los programas, pero varios lenguajes, generalmente funcionales, llevan a cabo lo que se conoce como inferencia de tipos, que le quita al programador la tarea de especificar los tipos. Al diseño y estudio formal de los sistemas de tipos se le conoce como teoría de tipos.

Ç+os versus lenguajes no tipados

El texto que sigue es una traducción defectuosa o incompleta.

Si quieres colaborar con Wikipedia, busca el artículo original y mejora o finaliza esta traducción.

Puedes dar aviso al autor principal del artículo pegando el siguiente código en su página de discusión: {{subst:Aviso mal traducido|Lenguaje de programación}} ~~~~

Se dice que un lenguaje tiene tipos si la especificación de cada lenguaje de producción provee medios para brincarse o subvertir el sistema de tipos.

Tipos estáticos versus tipos dinámicos

El texto que sigue es una traducción defectuosa o incompleta.

Si quieres colaborar con Wikipedia, busca el artículo original y mejora o finaliza esta traducción.

Puedes dar aviso al autor principal del artículo pegando el siguiente código en su página de discusión: {{subst:Aviso mal traducido|Lenguaje de programación}} ~~~~

En lenguajes con tipos estáticos se determina el tipo de todas las expresiones antes de la ejecución del programa (típicamente al compilar). Por ejemplo, 1 y (2+2) son expresiones enteras; no pueden ser pasadas a una función que espera una cadena, ni pueden guardarse en una variable que está definida como fecha.

Los lenguajes con tipos estáticos pueden manejar tipos explícitos o tipos inferidos. En el primer caso, el programador debe escribir los tipos en determinadas posiciones textuales. En el segundo caso, el compilador infiere los tipos de las expresiones y las declaraciones de acuerdo al contexto. La mayoría de los lenguajes populares con tipos estáticos, tales como C++, C# y Java, manejan tipos explícitos. Inferencia total de los tipos suele asociarse con lenguajes menos populares, tales como Haskell y ML. Sin embargo, muchos lenguajes de tipos explícitos permiten inferencias parciales de tipo; tantoJava y C#, por ejemplo, infieren tipos en un número limitado de casos.

Los lenguajes con tipos dinámicos determinan la validez de los tipos involucrados en las operaciones durante la ejecución del programa. En otras palabras, los tipos están asociados con valores en ejecución en lugar de expresiones textuales. Como en el caso de lenguajes con tipos inferidos, los lenguajes con tipos dinámicos no requieren que el programador escriba los tipos de las expresiones. Entre otras cosas, esto permite que una misma variable se pueda asociar con valores de tipos distintos en diferentes momentos de la ejecución de un programa. Sin embargo, los errores de tipo no pueden ser detectados automáticamente hasta que se ejecuta el código, dificultando la depuración de los programas, no obstante, en lenguajes con tipos dinámicos se suele dejar de lado la depuración en favor de técnicas de desarrollo como por ejemploBDD y TDD. Ruby, Lisp, JavaScript y Python son lenguajes con tipos dinámicos.

Tipos débiles y tipos fuertes

Los lenguajes débilmente tipados permiten que un valor de un tipo pueda ser tratado como de otro tipo, por ejemplo una cadena puede ser operada como un número. Esto puede ser útil a veces, pero también puede permitir ciertos tipos de fallas que no pueden ser detectadas durante la compilación o a veces ni siquiera durante la ejecución.

Los lenguajes fuertemente tipados evitan que pase lo anterior. Cualquier intento de llevar a cabo una operación sobre el tipo equivocado dispara un error. A los lenguajes con tipos fuertes se les suele llamar de tipos seguros.

Lenguajes con tipos débiles como Perl y JavaScript permiten un gran número de conversiones de tipo implícitas. Por ejemplo en JavaScript la expresión 2 * x convierte implícitamente x a un número, y esta conversión es exitosa inclusive cuando x es null, undefined, un Array o una cadena de letras. Estas conversiones implícitas son útiles con frecuencia, pero también pueden ocultar errores de programación.

Las características de estáticos y fuertes son ahora generalmente consideradas conceptos ortogonales, pero su trato en diferentes textos varia. Algunos utilizan el término de tipos fuertes para referirse a tipos fuertemente estáticos o, para aumentar la confusión, simplemente como equivalencia de tipos estáticos. De tal manera que C ha sido llamado tanto lenguaje de tipos fuertes como lenguaje de tipos estáticos débiles.

Implementación

Código fuente de un programa escrito en ellenguaje de programación Java.

La implementación de un lenguaje es la que provee una manera de que se ejecute un programa para una determinada combinación desoftware y hardware. Existen básicamente dos maneras de implementar un lenguaje: compilación e interpretación.

•Compilación: es el proceso que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz interpretar. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente.

•Interpretación: es una asignación de significados a las fórmulas bien formadas de un lenguaje formal. Como los lenguajes formales pueden definirse en términos puramente sintácticos, sus fórmulas bien formadas pueden no ser más que cadenas de símbolos sin ningún significado. Una interpretación otorga significado a esas fórmulas.

Se puede también utilizar una alternativa para traducir lenguajes de alto nivel. En lugar de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la compilación para utilizarlo en una ejecución futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el procesamiento de los datos. El código objeto no se graba para utilizarlo posteriormente.

La siguiente vez que se utilice una instrucción, se la deberá interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo o bucle, cada instrucción del bucle tendrá que volver a ser interpretada en cada ejecución repetida del ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una compilación después de cada modificación del programa cuando se quiere agregar

funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una ejecución del código.
La mayoría de lenguajes de alto nivel permiten la programación multipropósito, aunque muchos de ellos fueron diseñados para permitir programación dedicada, como lo fue el Pascal con las matemáticas en su comienzo. También se han implementado lenguajes educativos infantiles como Logo mediante una serie de simples instrucciones. En la actualidad son muy populares algunos lenguajes especialmente indicados para aplicaciones web, como Perl, PHP, Ruby, Python oJavaScript.

Técnica

Libros sobre diversos lenguajes de programación.

Para escribir programas que proporcionen los mejores resultados, cabe tener en cuenta una serie de detalles.

•Corrección. 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 desarrollarlo 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 así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; 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 programadores puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones

de código. Otros, por diversión o para impedir 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 al obtener la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico de 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 elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha desarrollado para sistemas GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

Paradigmas

Los programas se pueden clasificar por el paradigma del lenguaje que se use para producirlos. Los principales paradigmas son: imperativos, declarativos y orientación a objetos.

Los programas que usan un lenguaje imperativo especifican un algoritmo, usan declaraciones, expresiones y sentencias.3Una declaración asocia un nombre de variable con un tipo de dato, por ejemplo: var x: integer;. Una expresión contiene un valor, por ejemplo: 2 + 2 contiene el valor 4. Finalmente, una sentencia debe asignar una expresión a una variable o usar el valor de una variable para alterar el flujo de un programa, por ejemplo: x := 2 + 2; if x == 4 then haz_algo();. Una crítica común en los lenguajes imperativos es el efecto de las sentencias de asignación sobre una clase de variables llamadas "no locales".

Los programas que usan un lenguaje declarativo especifican las propiedades que la salida debe conocer y no especifica cualquier detalle de implementación. Dos amplias categorías de lenguajes declarativos son los lenguajes funcionales y los lenguajes lógicos. Los lenguajes funcionales no permiten asignaciones de variables no locales, así, se hacen más fácil, por ejemplo, programas como funciones matemáticas.4 El principio detrás de los lenguajes lógicos es definir el problema que se quiere resolver (el objetivo) y dejar los detalles de la solución al sistema.5 El objetivo es definido dando una lista de sub-objetivos. Cada sub-objetivo también se define dando una lista de sus sub-objetivos, etc. Si al tratar de buscar una solución, una ruta de sub-objetivos falla, entonces tal sub-objetivo se descarta y sistemáticamente se prueba otra ruta.

La forma en la cual se programa puede ser por medio de texto o de forma visual. En la programación visual los elementos son manipulados gráficamente en vez de especificarse por medio de texto.

7. Que son los lenguajes de quinta generación?

Son aquellos lenguajes en donde se usa una serie de instrucciones con limitaciones o restricciones a la base general del programa. Esto quiere decir que en vez de algoritmos lógica es usada. Este tipo de programación se conoce en inglés como constrain programming o logic programming. Las raíces que dieron origen a este tipo de programación de debe a la creación y evolución de la Lisp Machine en los años 80’s. De aquí, podemos mencionar lenguajes que ya usaban una semántica donde se usaba una programación enfocada en la lógica, un ejemplo es el ICAD. En este tiempo la quinta generación de lenguajes aún era considerada como algo futurístico y se creía que en un futuro esta remplazaría todos los lenguajes de programación anteriores. Varios proyectos y pruebas se han hecho ante el caso. Cabe mencionar los esfuerzos realizados por diferentes países como Japón. La ideología era la de desarrollar e implementar una nueva era en programación y redes computacionales usando estas nuevas tecnologías. Desafortunadamente y conforme a como los programas

crecieron se dieron cuenta que el paso principal para generar “esa” instrucción específica que generaría “inteligencia artificial” aún necesitaba del cerebro humano. Algunos lenguajes de quinta generación que suelen emplearse en modelos de inteligencia artificial son Prolog, OPS5, Mercury, Haskell, Modula 3, Jess e incluso C#.

8. Buscar 5 programas de creación de juegos y hacer un cuadro comparativo de ellos que permita observar ventajas y desventajas.

Aspectos	Game Maker	Stencyl	Scratch	Pilas-engine	Alice
Ventajas	• No necesita de grandes conocimientos de programación. • Contiene un lenguaje de programación de scripts, llamado Lenguaje Game Maker (GML). • Permite a los usuarios personalizar aún más sus juegos y extender sus características. • Los usuarios de Game Maker tienen permitido distribuir e incluso vender sus creaciones, interesante.	• Intuitivo y desarrollado por el MIT (Instituto Tecnológico de Massachusetts) • Posibilidad de exportar el juego a plataformas web • Poca necesidad de conocimientos de programación para poder trabajar con él • Posibilidad de exportar el juego para sistemas operativos de los iPhones e iPads	• Facilita a los más pequeños a crear sus propias historias interactivas, animaciones, juegos, música y arte. • Está diseñado especialmente para edades entre los 8 y 16 años, pero es usado por personas de todas las edades. • Permite compartir los proyectos a través de internet, pudiendo ser descargados y	• Es una herramienta que permite aprender a programar. • Los usuarios van a poder descubrir el funcionamiento de las computadoras y aprender a darles órdenes usando un lenguaje de programación. • Pilas-engine tiene juegos para todas las edades: hay algunos sencillos y	• Su entorno innovador en programación 3D hace que el crear una animación, un juego interactivo o video sea algo fácil y motivador. • Los objetos pueden moverse, girar, cambiar color, reaccionar al ratón y mucho más. • Su interfaz

			utilizados por otras personas.	otros más complejos.	interactiva genera instrucciones al arrastrar y soltar elementos gráficos.
Desventajas	• Lento comparado con lenguajes de programacion profesionales (Unity en ejemplo) • Soporta 3D lowpoly o bastante sencillo, sin funciones como Shaders. • Marca de agua en tus juegos (Version Gratuita solamente) • Y como desventaja clara, no se puede crear la aplicación para Windows 7, si	• Los juegos que esta aplicación permite crear son exclusivamente en 2D • proporcionar puntos de partida funcionales de géneros comunes de juegos 2D. • Este programa no permite ninguna extensión disponible.	• Ralentización en base a múltiples objetos • Al cabo de un tiempo y dependiendo de la computadora se podrá ver una ralentización en el proyecto si se excede un cierto número de objetos que trabajan al mismo tiempo, por lo que es poco	• Que no posee un centro. Por que no hay un servidor al que los usuarios tengan que acceder. • Los datos van de una computadora a otro directamente, pasando entre pares iguales.	• En este programa se percibe una falta de diseño. • Falta una planeación mucho más creativa de la animación. • En este programa dentro de la falta de diseño se ve que se

	para el resto de las plataformas que soporta también GameMaker.		recomendable para proyectos grandes. • Involucran muchos elementos, quedan muy pesados, lo que hace que a veces el programa no responda con la rapidez deseada.		codifica sin diseñar.

CONCLUSIONES

Podemos concluir que los fundamentos de programación:

1. Son un programa que ha sido diseñado para el control de algunas máquinas como suele serlo el computador.
2. Existen diversos tipos de fundamentos de programación con todas las características posibles.
3. Estos fundamentos fueron creados con el objetivo de facilitar diferentes problemas dentro de estos podemos encontrar:
    - El mantenimiento del software
    - Mejora de la productividad
    - Reutilización del software
4. Las clasificaciones de estos lenguajes pueden identificarse mediante las siguiente clasificaciones:
    - Nivel de abstracción
    - Nivel de propósito
    - Nivel de evolución histórica
    - Manera de ejecutarse
    - Manera de abordar la tarea a realizar
    - Paradigma de programación
    - Lugar de ejecución, concurrencia, interactividad.
    - Realización visual, determinismo y productividad.

REFERENCIAS BIBLIOGRAFICAS.

1. Fundamentos de Programación. 2008
   http://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n

   Wikibooks.org-Colombia

2. Lenguaje de programación. 2006
   http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n
   Wikipedia.org-Colombia
3. Lenguajes de programación. 2007
   http://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/
   Wordpress.com- Colombia
4. Lenguajes de programación. 2002
   http://es.kioskea.net/contents/304-lenguajes-de-programacion
   Kioskea.net- Colombia