Hola!

Nos encontramos en una nueva semana de Introducción a la Informática, la número 3.

En esta semana trabajaremos en la unidad 2, conociendo conceptos de Lenguajes de Programación.

Los temas que trataremos son:

Unidad 2: Conceptos de Lenguajes de programación
2.1. Lenguajes de Programación.
2.2. Lenguaje de Máquina.
2.3. Lenguajes de alto y bajo nivel.
2.4. Intérpretes y Compiladores.
2.5. Paradigmas de Programación: imperativo, funcional, lógico y orientado a objetos.

Nuestro objetivo en esta semana será conocer estos conceptos

Para ello iremos realizando distintas lecturas en los libros de Brookshear y Beekman.

Lenguajes de Programación

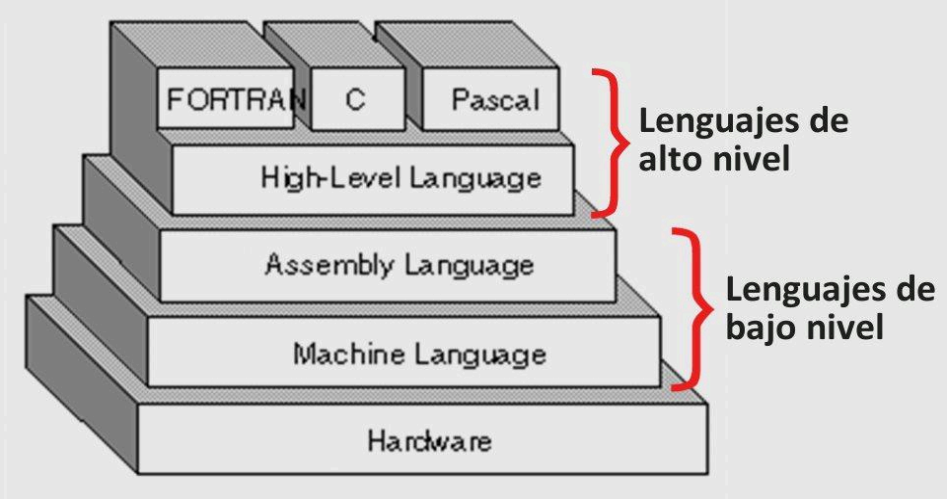
Los lenguajes de programación son herramientas que permiten construir aplicaciones personalizadas. Un lenguaje de programación es un conjunto de instrucciones junto con una serie de reglas que indiquen como pueden combinarse esas instrucciones para representar ideas más complejas. Cada instrucción tiene su propia sintaxis y semántica. La sintaxis hace referencia a la representación simbólica de la instrucción; la semántica hace referencia al significado de la instrucción. Cada lenguaje de programación contiene un conjunto finito y preciso de instrucciones utilizables para especificar la solución buscada. Estos lenguajes de programación tienen como objetivo lograr la comunicación entre el programador y las máquinas.

Lenguaje máquina

Como sabemos el hardware solo comprende lenguaje máquina. Este lenguaje es propio de cada máquina, si bien todos tienen instrucciones para efectuar las cuatro operaciones aritméticas básicas, para comparar pares de números, instrucciones para formar bucles, etc. cada uno de estos lenguajes máquina son lenguajes diferentes, y las máquinas basadas en uno de ellos no pueden entender los programas escritos en otro.

El lenguaje máquina, utilizado por los primeros programadores, emplea códigos numéricos para representar las distintas instrucciones, esto dificultaba demasiado la tarea de programar, era tedioso tener que identificar errores en extensas combinaciones de 0s y 1s.

Lenguajes de bajo y alto nivel



Con la invención del ensamblador (un lenguaje funcionalmente similar al lenguaje máquina, pero más sencillo de escribir, leer y comprender por las personas) el proceso de programación se hizo más sencillo. En este lenguaje, los programadores utilizan códigos alfabéticos que se corresponden con las instrucciones numéricas de la máquina. Por ejemplo, la sentencia ensamblador para restar podría ser SUB. Desde luego, SUB no significa nada para la computadora, la cual sólo responde ante comandos del tipo 10110111. Para establecer un lazo de unión entre el programador y la computadora, un programa llamado ensamblador traduce cada instrucción de este lenguaje en la sentencia máquina correspondiente.

Los lenguajes ensamblador y máquina son lenguajes de bajo nivel, los cuales conllevan procesos repetitivos, tediosos y muy propensos a los errores. El lenguaje ensamblador se continua utilizando para escribir partes de aplicaciones en las que la velocidad y la comunicación con el hardware es un factor crítico.

La programación con lenguajes de bajo nivel obliga al programador a pensar al nivel de la máquina y a incluir una enorme cantidad de detalles en cada programa. Esto conlleva procesos repetitivos, tediosos y muy propensos a los errores. Para complicar aún más las cosas, cualquier programa escrito en uno de estos lenguajes deben ser reescritos por completo antes de poder utilizarlos en una computadora con un lenguaje máquina diferente.

En busca de dar solución a las cuestiones antes mencionadas surgieron los lenguajes de alto nivel, que están a medio camino entre el lenguaje natural de los humanos y los lenguajes máquina, fueron desarrollados a principio de la década de los 50 para simplificar y perfilar el proceso de programación. Los lenguajes de alto nivel permiten que los programadores escriban programas usando una terminología y notación familiar en lugar de las enigmáticas instrucciones máquina.

Intérpretes y compiladores

Para que una computadora entienda un programa escrito en uno de estos lenguajes, es preciso convertirlo al idioma de las máquinas, es decir, a unos y ceros. Para convertir un programa en lenguaje máquina, necesitamos un software de traducción. Dicho programa puede ser un intérprete (un programa que traduce y transmite cada sentencia de forma individual, del mismo modo que en las Naciones Unidas se traduce un discurso del español al inglés), o un compilador (un programa que traduce el programa completo antes de pasarlo a la computadora, del mismo modo que un estudiante puede traducir una novela del inglés al español).

Los intérpretes y los compiladores traducen los programas de alto nivel en lenguaje máquina. Una vez interpretada o compilada, una sentencia de uno de estos lenguajes se transforma en varias instrucciones máquina. Un lenguaje de alto nivel oculta al programador la mayoría de los detalles oscuros de las operaciones máquina. Como resultado de ello, resulta más sencillo centrarse en la lógica básica del programa, es decir, en la idea principal. Además de ser más sencillos de escribir y depurar, los programas de alto nivel tienen la ventaja de poder transportarse de una máquina a otra. Un código escrito en un determinado lenguaje estándar puede ser compilado y ejecutado en cualquier computadora que disponga de ese compilador.

Paradigmas de programación

Los paradigmas de programación representan enfoques fundamentalmente distintos para obtener soluciones a los problemas y, por tanto, afectan al proceso completo de desarrollo software.

El paradigma imperativo, también conocido como paradigma procedimental, representa el enfoque tradicional del proceso de programación. Como su nombre sugiere, el paradigma imperativo define el proceso de programación como el desarrollo de una secuencia de comandos que, al ser ejecutados, manipula los datos para generar el resultado deseado. Por tanto, el paradigma imperativo nos dice que debemos enfocar el proceso de programación determinando un algoritmo para solucionar el problema que nos traemos entre manos y luego expresando dicho algoritmo como una secuencia de sentencias.

El paradigma orientado a objetos, que está asociado con el proceso de programación denominado programación orientada a objetos (OOP, *Object-Oriented Programming*). De acuerdo con este paradigma, un sistema software se ve conceptualmente como un conjunto de unidades, denominadas objetos, cada uno de las cuales es capaz de llevar a cabo las acciones que le afectan directamente, así como de solicitar acciones a otros objetos. De forma conjunta, estos objetos interactúan para resolver el problema que tengamos entre manos.

El paradigma orientada a aspectos (AOP, Aspect-oriented programming) tiene por objetivo proporcionar mecanismos que hacen posible separar los elementos que son transversales a todo el sistema. Permite capturar los diferentes intereses entrecruzados que componen una aplicación en entidades bien definidas, eliminando las dependencias inherentes entre cada uno de los módulos que la componen. Este paradigma tiene por objetivo encapsular aspectos transversales, que atraviesan a diferentes módulos y no pueden encapsularse limpiamente en el POO.

El paradigma declarativo pide al programador que describa el problema que hay que resolver, en lugar del algoritmo que hay que aplicar. Para ser más precisos, un sistema de programación declarativo aplica un algoritmo preestablecido para resolución de problemas de propósito general con el fin de solucionar los problemas que se le presenten. En un entorno de este tipo, la tarea del programador consiste en desarrollar un enunciado preciso del problema en lugar de describir un algoritmo para la resolución del problema.

El paradigma funcional. En este caso, un programa se ve como una entidad que acepta entradas y genera salidas. Los matemáticos denominan a tales entidades funciones, razón por la que esta técnica recibe el nombre de paradigma funcional. Bajo este paradigma, los programas se construyen conectando entidades predefinidas más pequeñas (funciones predefinidas), tal que las salidas de cada unidad se utilicen como entradas de otras unidades, de tal forma que al final se obtenga la relación entrada-salida global deseada. En resumen, el proceso de programación bajo el paradigma funcional consiste en construir funciones como conjuntos anidados de otras funciones más simples.

Lecturas

Debemos realizar las siguientes lecturas:

- **BROOKSHEAR, J. G. INTRODUCCIÓN A LA COMPUTACIÓN.** 11ra edición. PEARSON EDUCACIÓN, S.A., Madrid, 2012. Cap. 2 "2 **.2 Lenguaje máquina". Páginas 91 a 98.**

- **BROOKSHEAR, J. G. INTRODUCCIÓN A LA COMPUTACIÓN.** 11ra edición. PEARSON EDUCACIÓN, S.A., Madrid, 2012. Apéndice D "Lenguajes de programación de alto nivel **". Páginas 641 a 643.**
- **BEEKMAN, G. *INTRODUCCIÓN A LA INFORMÁTICA*.** *6ta edición*. PEARSON EDUCACIÓN, S.A., Madrid, 2005. **Cap. 14 "DISEÑO Y DESARROLLO DE SISTEMAS. Lenguajes de programación y metodologías". Páginas 520 a 524.**
- **BROOKSHEAR, J. G. INTRODUCCIÓN A LA COMPUTACIÓN.** 11ra edición. PEARSON EDUCACIÓN, S.A., Madrid, 2012. Cap. 6 "Paradigmas de programación **". Páginas 288 a 293.**

Materiales

Podemos revisar estos materiales didácticos producidos por la cátedra:

- U2_03_Conceptos de Lenguajes de Programación
- Programación orientada a aspectos
- Ejemplos sobre Paradigmas de Programación

Actividades

- Cuestionario de seguimiento Semana 3 (Cierra lunes 8-04-2024 a las 23:59)