Presentación:

Historia de la programación

· Jessica Rivero Espinosa (100025022) ·

· Inteligencia en Redes de Comunicaciones ·

· Ingeniería de Telecomunicación ·

· 5º Curso ·

La computadora fue inventada para facilitar el trabajo intelectual. Si el hombre tiene algún problema, el diseñador define el algoritmo que resuelve el problema, el programador lo codifica en un lenguaje de programación, el cual la computadora es capaz de "entender", luego la computadora ejecuta el algoritmo expresado como programa en el lenguaje de programación en cuestión, y entrega al hombre la respuesta. Los lenguajes de programación son el medio de comunicación entre el hombre y la máquina, por lo tanto son una forma de representación del conocimiento.

Representación de conocimiento

Representación del conocimiento es escribir en un lenguaje descripciones del mundo. Una de las ambiciones es poder llegar a representar el "sentido común".
En general una representación debe:

- Ser capaz de expresar el conocimiento que deseamos expresar.
- Tener capacidad para resolver problemas.
- Dar simplicidad para acceder al conocimiento y facilidad de entendimiento.

Por lo tanto un lenguaje de representación tiene que ser expresivo, conciso, no ambiguo, y efectivo, pues es el que determina todas las características previas.

Tipos de lenguajes de programación

Los tipos más importantes de lenguajes de programación son:

- ***Lenguajes Imperativos***

Su origen es la propia arquitectura de von Neumann, que consta de una secuencia de celdas (memoria) en las cuales se pueden guardar datos e instrucciones, y de un procesador capaz de ejecutar de manera secuencial una serie de operaciones (ó comandos) principalmente aritméticas y booleanas. En general, un lenguaje imperativo ofrece al programador conceptos que se traducen de forma natural al modelo de la máquina.
Ejemplos: FORTRAN, Algol, Pascal, C, Modula-2, Ada.

El programador tiene que traducir la solución abstracta del problema a términos muy primitivos, cercanos a la máquina, por lo que los programas son más "comprensibles" para la máquina que para el hombre. Esto es una desventaja para nosotros que hace que sea sumamente complicado construir código en lenguaje imperativo. Lo bueno de este lenguaje es que es tan cercano al lenguaje de la máquina que la eficiencia en la ejecución es altísima.

- ***Lenguajes Funcionales***

Los matemáticos resuelven problemas usando el concepto de función, que convierte datos en resultados. Sabiendo cómo evaluar una función, usando la computadora, podríamos resolver automáticamente muchos problemas. Este fue el pensamiento que llevó a la creación de los lenguajes de programación funcionales. Además se aprovechó la posibilidad que tienen las funciones para manipular datos simbólicos, y no solamente numéricos, y la propiedad de las funciones que les permite componer, creando de esta manera, la oportunidad para resolver problemas complejos a partir de las soluciones a

otros más sencillos. También se incluyó la posibilidad de definir funciones recursivamente.

Un lenguaje funcional ofrece conceptos que son muy entendibles y relativamente fáciles de manejar. El lenguaje funcional más antiguo y popular es LISP, diseñado por McCarthy en la segunda mitad de los años 50. Se usa principalmente en Inteligencia Artificial. En los 80 se añadió a los lenguajes funcionales la tipificación y algunos conceptos modernos de modularización y polimorfismo, un ejemplo es el lenguaje ML.

Programar en un lenguaje funcional significa construir funciones a partir de las ya existentes. Por lo tanto es importante conocer y comprender bien las funciones que conforman la base del lenguaje, así como las que ya fueron definidas previamente. De esta manera se pueden ir construyendo aplicaciones cada vez más complejas. La desventaja es que está alejado del modelo de la máquina de von Neumann y, por lo tanto, la eficiencia de ejecución de los intérpretes de lenguajes funcionales es peor que la ejecución de los programas imperativos precompilados.

- ***Lenguajes Lógicos***

Otra forma de razonar para resolver problemas en matemáticas se fundamenta en la lógica de primer orden. El conocimiento básico de las matemáticas se puede representar en la lógica en forma de axiomas, a los cuales se añaden reglas formales para deducir cosas verdaderas (teoremas). Gracias al trabajo de algunos matemáticos, de finales de siglo pasado y principios de éste, se encontró la manera de automatizar computacionalmente el razonamiento lógico -particularmente para un subconjunto significativo de la lógica de primer orden- que permitió que la lógica matemática diera origen a otro tipo de lenguajes de programación, conocidos como lenguajes lógicos. También se conoce a estos lenguajes, y a los funcionales, como lenguajes declarativos, porque para solucionar un problema el programador solo tiene que describirlo con axiomas y reglas de deducción en el caso de la programación lógica y con funciones en el caso de la programación funcional.

En los lenguajes lógicos se utiliza el formalismo de la lógica para representar el conocimiento sobre un problema y para hacer preguntas que se vuelven teoremas si se demuestra que se pueden deducir a partir del conocimiento dado en forma de axiomas y de las reglas de deducción estipuladas. Así se encuentran soluciones a problemas formulados como preguntas. Con base en la información expresada dentro de la lógica de primer orden, se formulan las preguntas sobre el dominio del problema y el intérprete del lenguaje lógico trata de encontrar la respuesta automáticamente. El conocimiento sobre el problema se expresa en forma de predicados (axiomas) que establecen relaciones sobre los símbolos que representan los datos del dominio del problema.

El PROLOG surgió a principio de los 70 y es el primer lenguaje lógico. Las aplicaciones en la Inteligencia Artificial lo mantienen vivo y útil.

En el caso de la programación lógica, el trabajo del programador es la buena descripción del problema en forma de hechos y reglas. A partir de ésta se pueden encontrar muchas soluciones dependiendo de como se formulen las preguntas (metas), que tienen sentido para el problema. Si el programa está bien definido, el sistema encuentra automáticamente las respuestas a las preguntas formuladas. En este caso ya no es necesario definir el algoritmo de solución, como en la programación imperativa,

lo fundamental aquí es expresar bien el conocimiento sobre el problema. En programación lógica, al igual que en programación funcional, el programa, en este caso los hechos y las reglas, están muy alejados del modelo von Neumann que posee la máquina en la que tienen que ser interpretados; por lo que la eficiencia de la ejecución es inferior a la de un programa equivalente en lenguaje imperativo. Sin embargo, para cierto tipo de problemas, la formulación del programa mismo puede ser mucho más sencilla y natural.

- ***Lenguajes Orientados a Objetos***

A mediados de los años 60 se empezó a usar las computadoras para la simulación de problemas del mundo real. Pero el mundo real está lleno de objetos, en la mayoría de los casos complejos, los cuales difícilmente se traducen a los tipos de datos primitivos de los lenguajes imperativos. Así surgió el concepto de *objeto* y sus colecciones (*clases de objetos*), que permitieron introducir abstracciones de datos a los lenguajes de programación. La posibilidad de reutilización del código y sus indispensables modificaciones, se reflejaron en la idea de las jerarquías de *herencia de clases*. También surgió el concepto de polimorfismo introducido vía procedimientos virtuales. Todos estos conceptos (que hoy identificamos como conceptos del modelo de objetos) fueron presentados en el lenguaje Simula 67, desde el año 1967, aunque este lenguaje estaba enfocado a aplicaciones de simulación discreta.

Fue en los años 80 cuando surgieron lenguajes de programación con conceptos de objetos encabezada por Smalltalk, C++, Eiffel, Modula-3, Ada 95 y terminando con Java.

El modelo de objetos, y los lenguajes que lo usan, parecen facilitar la construcción de sistemas o programas en forma modular. Los objetos ayudan a expresar programas en términos de abstracciones del mundo real, lo que aumenta su comprensión. La clase ofrece cierto tipo de modularización que facilita las modificaciones al sistema. La reutilización de clases previamente probadas en distintos sistemas también es otro punto a favor. Sin embargo, el modelo de objetos, a la hora de ser interpretado en la arquitectura von Neumann conlleva un excesivo manejo dinámico de memoria debido a la constante creación de objetos, así como a una carga de código fuerte causada por la constante invocación de métodos. Por lo tanto los programas en lenguajes orientados a objetos son ineficientes, en tiempo y memoria, contra los programas equivalentes en lenguajes imperativos, aunque les ganan en la comprensión de código.

- ***Lenguajes Concurrentes, Paralelos y Distribuidos***

El origen de los conceptos para el manejo de concurrencia, paralelismo y distribución está en el deseo de aprovechar al máximo la arquitectura von Neumann y sus modalidades reflejadas en conexiones paralelas y distribuidas.
Esto fue un tema importante sobre todo cuando las computadoras eran caras y escasas; el sistema operativo tenía que ofrecer la ejecución concurrente y segura de programas de varios usuarios, que desde distintos terminales utilizaban un solo procesador, y así surgió la necesidad de introducir algunos conceptos de *programación concurrente* para programar los sistemas operativos.

Cuando los procesadores cambiaron de tamaño y de precio, se abrió la posibilidad de contar con varios procesadores en una máquina y ofrecer el *procesamiento en paralelo* (procesar varios programas al mismo tiempo). Esto dio el impulso a la creación de

lenguajes que permitían expresar el paralelismo. Finalmente, llegaron las redes de computadoras, que también ofrecen la posibilidad de ejecución en paralelo, pero con procesadores distantes, lo cual conocemos como la *programación distribuida*. Históricamente encontramos soluciones conceptuales y mecanismos (semáforos, regiones críticas, monitores, envío de mensajes, llamadas a procedimientos remotos (RPC)) que se incluyeron después en lenguajes de programación como Concurrent Pascal o Modula (Basados en monitores), Ada ó SR (Basada en RendezVous (Tipo de RPC)), ALGOL 68(Semáforos), OCCAM (Envío de mensajes), Java...

Otros tipos de lenguajes de programación son:

Procedural Language, *Declarative Language*, *Applicative Language*, *Definitional Language*, *Single Assignment Language*, *Dataflow Language*, *Constraint Language*, *Lenguaje de cuarta generación(4GL)*, *Query Language*, *Specification Language*, *Assembly Language*, *Intermediate Language*, *Metalenguajes*.