Eduardo Del Portillo
IP y CFT Acuario Data **edelportillo@ipcc.cl**

Lenguajes de Programación

Historia de los lenguajes; Los lenguajes de programación cierran el abismo entre las computadoras, que sólo trabajan con números binarios, y los humanos, que preferimos utilizar palabras y otros sistemas de numeración.

Mediante los programas se indica a la computadora qué tarea debe realizar y como efectuarla, pero para ello es preciso introducir estas ordenes en un lenguaje que el sistema pueda entender. En principio, el ordenador sólo entiende las instrucciones en código máquina, es decir, el especifico de la computadora. Sin embargo, a partir de éstos se elaboran los llamados lenguajes de alto y bajo nivel.

Generaciones de los lenguajes

LENGUAJES DE BAJO NIVEL:

Utilizan códigos muy cercanos a los de la máquina, lo que hace posible la elaboración de programas muy potentes y rápidos, pero son de difícil aprendizaje.

LENGUAJES DE ALTO NIVEL:

Por el contrario, son de uso mucho más fácil, ya que en ellos un solo comando o instrucción puede equivaler a millares es código máquina. El programador escribe su programa en alguno de estos lenguajes mediante secuencias de instrucciones. Antes de ejecutar el programa la computadora lo traduce a código máquina de una sola vez (lenguajes compiladores) o interpretándolo instrucción por instrucción (lenguajes intérpretes).

Ejemplos de lenguajes de alto nivel: Pascal, Cobol, Basic, Fortran, C++ Un Programa de computadora, es una colección de instrucciones que, al ser ejecutadas por el CPU de una máquina, llevan a cabo una tarea ó función específica. Este conjunto de instrucciones que forman los programas son almacenados en archivos denomina dos archivos ejecutables puesto que, al teclear su nombre (o hacer clic sobre el icono que los identifica) logras que la computadora los cargue y corra, o ejecute las instrucciones del archivo. El contenido de un archivo ejecutable no puede ser entendido por el usuario, ya que no está hecho para que la gente lo lea, sino para que la computadora sea quien lo lea.

Los archivos de programas ejecutables contienen el código máquina, que la CPU identifica como sus instrucciones. Son lo que conocemos como Programas Objeto. Dado que sería muy difícil que los programadores crearan programas directamente en código de máquina, usan lenguajes más fáciles de leer, escribir y entender para la gente.

El programador teclea instrucciones en un editor, que es un programa parecido a un simple procesador de palabras, estas instrucciones son almacenadas en archivos denominados programas fuentes (código fuente). Si los programadores necesitan hacer cambios al programa posteriormente vuelven a correr el editor y cargan el programa fuente para modificarlo.

El proceso de conversión de programas fuente a programas objeto se realiza mediante un programa denominado compilador. El compilador toma un programa fuente y lo traduce a programa objeto y almacena este último en otro archivo.

PROGRAMA FUENTE:

Es el programa escrito en alguno de los lenguajes y que no ha sido traducido al lenguaje de la maquina, es decir el programa que no está en código de máquina y que por lo tanto no puede ser ejecutable.

PROGRAMA OBJETO:

Es aquel programa que se encuentra en lenguaje máquina y que ya es ejecutable por esta.

Programación Orientada a Objetos:

La programación orientada a objetos no es un concepto nuevo, sus inicios y técnicas de programación se iniciaron a principios de los 70. Se puede definir programación orientada a objetos (OOPS) como una técnica de programación que utiliza objetos como bloque esencial de construcción. La OOPS, es un tipo de programación más cercana al razonamiento humano. La OOPS surge como una solución a la programación de grandes programas, y para solventar el mantenimiento de dichas aplicaciones, ya que en la programación estructura el más mínimo cambio supone la modificación de muchas funciones relacionadas, en cambio con la OOPS solo es cuestión de añadir o modificar métodos de una clase o mejor, crear una nueva clase a partir de otra (Herencia). Dos lenguajes destacan sobre el resto para programar de esta forma, Smalltalk y C++.

Concepto de Objeto:

Desde un punto de vista general un Objeto es una estructura de datos de mayor o menor complejidad con las funciones que procesan estos datos. Dicho de otra forma, sería Datos más un Código que procesa estos datos. A los datos se les denomina miembros dato y a las funciones miembro o miembro funciones. Los datos están ocultos y sólo se puede acceder a ellos mediante las funciones miembro.

Clases:

Las Clases son como plantillas o modelos que describen como se construyen ciertos tipos de Objeto. Cada vez que se construye un Objeto de una Clase, se crea una instancia de esa Clase("instance"). Una Clase es una colección de Objetos similares y un Objeto es una instancia de una Clase. Se puede definir una Clase como un modelo que se utiliza para describir uno o más Objetos del mismo tipo.

Herencia:

Una característica muy importante de los Objetos y las Clases es la Herencia, una propiedad que permite construir nuevos Objetos (Clases) a partir de unos ya existentes. Esto permite crear "Sub-Clases" denominadas Clases Derivadas que comparten las propiedades de la Clase de la cual derivan (Clase base). Las Clases derivadas heredan código y datos de la clase base, asimismo incorporan su propio código y datos especiales. Se puede decir que la herencia permite definir nuevas Clases a partir de las Clases ya existentes.

Polimorfismo:

En un sentido literal, Polimorfismo significa la cualidad de tener más de una forma. En el contexto de POO, el Polimorfismo se refiere al hecho de que una simple operación puede tener diferente comportamiento en diferentes objetos. En otras palabras, diferentes objetos reaccionan al mismo mensaje de modo diferente. Los primeros lenguajes de POO fueron interpretados, de forma que el Polimorfismo se contemplaba en tiempo de ejecución. Por ejemplo, en C++, al ser un lenguaje compilado, el Polimorfismo se admite tanto en tiempo de ejecución como en tiempo de compilación Decimos entonces que:

El tema de la Programación Orientada a Objetos (Object Oriented Programming O-O-P) sigue siendo para el que escribe un territorio inquietante, interesante y en gran medida desconocido, como parece ser también para la gran mayoría de los que están en el campo de la programación. Sin tratar de excluir a aquellos que han afrontado este desarrollo desde el punto de vista académico y formal (maestrías y doctorados) el tema se antoja difícil para los no iniciados. Con este breve artículo se dirigiré en particular a la gran base de programadores prácticos que andan en búsqueda de mejores herramientas de desarrollo de programas, que faciliten el trabajo de los usuarios y a la vez disminuyan la gran cantidad de considerandos que aparecen al empeñarse en un proyecto de cómputo.

Posteriormente aparece el QuickBasic, que es muy familiar con el BASIC (ley del menor esfuerzo). Ofrece estructuras de datos (tipos y registros complejos), además de estructuras de instrucciones en procedimientos y módulos; editor "inteligente" que revisa la sintaxis y ejecución de las instrucciones mientras se edita el programa, generación de ejecutable una vez terminado (.EXE), existencia de bibliotecas externas y enlace con módulos objeto generados en otro lenguaje.

Pero la necesidad de estar en la ola de moda es más fuerte que el sentido común. Las aplicaciones en Windows siempre han despertado la envidia de los programadores, al hacer ver sus programas pálidos e insulsos por comparación. Solución: programar en Windows.

Originalmente programar en Windows representaba un largo y tedioso camino para dominar las complejas herramientas de desarrollo. Sólo recientemente han aparecido desarrolladores de aplicaciones para Windows que le permiten al programador pintar sus ventanas y realizar los enlaces entre los objetos
con programación tradicional, evitando en gran medida involucrarse con los conceptos complicados de los objetos.

Los principales conceptos que se manejan en la Programación Orientada a Objetos son: 1. encapsulado, 2. herencia y 3. Polimorfismo.

Según esto, la encapsulación es la creación de módulos autosuficientes que contienen los datos y las funciones que manipulan dichos datos. Se aplica la idea de la caja negra y un letrero de "prohibido mirar adentro". Los objetos se comunican entre sí intercambiando mensajes. De esta manera, para armar aplicaciones se utilizan los objetos cuyo funcionamiento está perfectamente definido a través de los mensajes que es capaz de recibir o mandar. Todo lo que un objeto puede hacer está representado por su interfase de mensajes. Para crear objetos, el programador puede recurrir a
diversos lenguajes como el C++, el Smalltalk, el Visual Objects y otros. Si se desea solamente utilizar los objetos y enlazarlos en una aplicación por medio de la programación tradicional se puede recurrir al Visual Basic, al CA-Realizer, al Power Builder, etc.

El concepto de herencia pareció sencillo de entender una vez que capta otro concepto de O-O-P: las clases. En O-O-P se acostumbra agrupar a los objetos en clases. Esto es muy común en la vida diaria. Todos nosotros tendemos a clasificar los objetos comunes por clases. Manejamos la clase mueble, la clase mascota, la clase alimento, etc. Obviamente en el campo de la programación esta clasificación es más estricta. ¿Cuál es el sentido de las clases? Fundamentalmente evitar definir los objetos desde cero y facilitar su rehuso. Si se trabaja con clases, al querer definir un nuevo objeto, se parte de alguna clase definida anteriormente, con lo que el objeto en cuestión hereda las características de los objetos de su clase.

Imaginemos que creamos una clase "aves" y describimos las características de las aves (plumas, pico, nacen de huevo, etc.). Más adelante necesitamos una clase "pingüino". Como pertenece a "aves" no requerimos volver a declarar lo descrito sino marcamos que "pingüino" es

una subclase de "aves" con lo que "pingüino" hereda todas sus características. A continuación sólo declaramos los detalles que determinan lo que distingue a "pingüino" de "aves". Asimismo podemos declarar "emperador" como una subclase de "pingüino", con lo que "emperador" heredará todas las características de las superclases "pingüino" y "aves" más las características que nosotros declaremos en particular para "emperador". En un programa (imaginario por supuesto) se puede utilizar estas clases (aves, pingüino y emperador). El hecho de colocar a un individuo en particular en estas clases es lo que se llama objeto y se dice que es una instancia de una clase. Así, si se coloco a Fredy (un pingüino emperador) en un programa, se dice que el objeto Fredy es una instancia de la clase emperador. Fredy aparecerá en el programa con todas las características (herencia) de aves, de pingüino y de emperador.

Por otra parte, entender el concepto de Polimorfismo implicó un buen número de horas de indagación y búsqueda de ejemplos. Espero que éste resulte claro: supóngase que declaramos un objeto llamado Suma. Este objeto requiere dos parámetros (o datos) como mensaje para operar. En la programación tradicional tendríamos que definir el tipo de datos que le enviamos, como por ejemplo dos números enteros, dos números reales, etc. En O-O-P el tipo de dato se conoce hasta que se ejecuta el programa.

COMPILADOR:

Es un programa que traduce un lenguaje de alto nivel al lenguaje máquina. Un programa compilado indica que ha sido traducido y está listo para ser ejecutado. La ejecución de los programas compilados es más rápida que la de los interpretados, ya que el interprete debe traducir mientras está en la fase de ejecución (saca todos los errores). Un compilador es un programa que traduce el programa fuente (conjunto de instrucciones de un lenguaje de alto nivel, por ejemplo BASIC o Pascal) a programa objeto (instrucciones en lenguaje máquina que la computadora puede interpretar y ejecutar). Se requiere un compilador para cada lenguaje de programación. Un compilador efectúa la traducción, no ejecuta el programa. Una vez compilado el programa, el resultado en forma de programa objeto será directamente ejecutable. Presentan la ventaja considerable frente a los intérpretes de la velocidad de ejecución, por lo que su uso será mejor en aquellos programas probados en los que no se esperan cambios y que deban ejecutarse muchas veces. En caso de que se opte por un interpretador se debe considerar que el intérprete resida siempre en memoria.

INTERPRETE:

Traductor de lenguajes de programación de alto nivel, los interpretes ejecutan un programa línea por línea. El programa siempre permanece en su forma original(programa fuente) y el interprete proporciona la traducción al momento de ejecutar cada una de las instrucciones. Un intérprete es un programa que procesa los programas escritos en un lenguaje
de alto nivel, sin embargo, está diseñado de modo que no existe independencia entre la etapa de traducción y la etapa de ejecución. Un intérprete traduce cada instrucción o sentencia del programa escrito a un lenguaje máquina e inmediatamente se ejecuta. Encuentran su mayor ventaja en la interacción con el usuario, al facilitar el desarrollo y puesta a punto de programas, ya que los errores son fáciles de detectar y sobre todo de corregir.

LENGUAJE MÁQUINA:

Lenguaje original de la computadora, un programa debe estar escrito en el lenguaje de la máquina para poder ser ejecutado. Este es generado por software y no por el programador. El programador escribe en un lenguaje de programación, el cual es traducido al lenguaje de máquina mediante interpretes y compiladores.

Case: (Computer-Aided Software Engineering o Computer- Aided Systems Engineering) Ingeniería de Software Asistida por Computadora o Ingeniería de Sistemas Asistida por computadora Software que se utiliza en una cualquiera o en todas las fases del desarrollo de un sistema de información, incluyendo análisis, diseño y programación. Por ejemplo, los diccionarios de datos y herramientas de diagramación ayudan en las fases de análisis y diseño, mientras que los generadores de aplicaciones aceleran la fase de programación.

Las herramientas CASE proporcionan métodos automáticos para diseñar y documentar las técnicas tradicionales de programación estructurada. La meta última de CASE es proveer un lenguaje para describir el sistema completo, que sea suficiente para generar todos los programas necesarios.