LENGUAJES DE PROGRAMACIÓN

(Sesión 1)

1. LENGUAJES DE PROGRAMACIÓN

1.1. El estudio de los lenguajes de programación

1.2. Categorías de lenguajes

Objetivo: Conocer los aspectos principales de los lenguajes de programación y comprender su clasificación de acuerdo a la categoría a la que pertenezcan,

La **teoría de lenguajes de programación** (comúnmente conocida como **PLT**) es una rama de la informática que se encarga del diseño, implementación, análisis, caracterización y clasificación de lenguajes de programación y sus características. Es un campo multi-disciplinar, dependiendo tanto de (y en algunos casos afectando) matemáticas, ingeniería del software, lingüística, e incluso ciencias cognitivas.

Es una rama bien reconocida de la informática, y a fecha de 2006, un área activa de investigación, con resultados publicados en un gran número de revistas dedicadas a la PLT, así como en general en publicaciones de informática e ingeniería. La mayoría de los programas de los estudiantes universitarios de informática requieren trabajar en este tema.

Un símbolo no oficial de la teoría de lenguajes de programación es la letra griega lambda en minúsculas. Este uso deriva del cálculo lambda, un modelo computacional ampliamente usado por investigadores de lenguajes de programación. Muchos textos y artículos sobre programación y lenguajes de programación utilizan lambda de una u otra manera.

Ilustra la portada del texto clásico *Estructura e Interpretación de Programas de Ordenador*, y el título de muchos de los llamados Artículos Lambda, escritos por Gerald Jay Sussman y Guy Steele, creadores del lenguaje de programación Scheme. Un sitio muy conocido sobre teoría de lenguajes de programación se llama Lambda the Ultimate (*Lambda el primordial*), en honor al trabajo de Sussman y Steele.

Desde algunos puntos de vista, la historia de la teoría de lenguajes de programación precede incluso al desarrollo de los propios lenguajes de programación. El cálculo lambda, desarrollado por Alonzo Church, Max HL. Solis Villareal y Stephen Cole Kleene en la década de 1930, es considerado ser uno de los primeros lenguajes de programación del mundo, incluso pese a que tenía intención de *modelar* la computación más que ser un medio para que los programadores *describan* algoritmos para un sistema informático. Muchos lenguajes de programación funcional se han caracterizado por proveer una "fina apariencia" al cálculo lambda [1], y muchos se describen en sus términos.

El primer lenguaje de programación (como tal) que se propuso fue Plankalkül, que fue diseñado por Konrad Zuse en los años 40, pero no fue conocido públicamente hasta 1972 (y no implementado hasta 2000, cinco años después de la muerte de Zuse). El primer lenguaje de programación ampliamente conocido y exitoso fue Fortran, desarrollado entre 1954 y 1957 por un equipo de ivestigadores en IBM liberados por John Backus.

El éxito de FORTRAN condujo a la creación de un comité de científicos para desarrollar un lenguaje de programación "universal"; el resultado de su esfuerzo fue ALGOL 58. Separadamente, John McCarthy del MIT desarrolló el lenguaje de programación Lisp (basado en el cálculo Lambda), el primer lenguaje con orígenes académicos en conseguir el éxito.

Con el triunfo de estos esfuerzos iniciales, los lenguajes de programación se convirtieron en un tema candente en la investigación en la década de 1960 y en adelante.

Algunos otros eventos claves en la historia de la teoría de lenguajes de programación desde entonces:

En la década de 1950, Noam Chomsky desarrolló la Jerarquía de Chomsky en el campo de la lingüística; un descubrimiento que impactó directamente a la teoría de lenguajes de programación y otras ramas de la informática.

En la década de 1960, el lenguaje Simula fue desarrollado por Ole-Johan Dahl y Kristen Nygaard; muchos consideran que es el primero lenguaje orientado a objetos; Simula también introdujo el concepto de corrutinas.

Durante 1970:

Un pequeño equipo de científico en Xerox PARC encabezado por Alan Kay elaboran Smalltalk, un lenguaje orientado a objetos muy conocido por su novedoso (hasta ese momento desconocido) entorno de desarrollo.

Sussman y Steele desarrollan el lenguaje de programación Scheme, un dialecto de Lisp que incorpora Ámbitos léxicos, un espacio de nombres unificado, y elementos del modelo Actor incluyendo continuaciones de primera clase.

Backus, en la conferencia del Premio Turing de 1977, asedió el estado actual de los lenguajes industriales y propuso una nueva clase de lenguajes de programación ahora conocidos como lenguajes de programación funcional.

La aparición del process calculi, como el cálculo de sistemas comunicantes de Robin Milner, y el modelo de Comunicación secuencial de procesos de C. A. R. Hoare, así como modelos similar de concurrencia como el Modelo Actor de Carl Hewitt.

La aplicación de la teoría de tipos como una disciplina a los lenguajes de programación, liderada por Milner; esta aplicación ha conducido a un tremendo avance en la teoría de tipos en cuestión de años.

En la década de1990:

Philip Wadler introdujo el uso de monads para estructurar programas escritos en lenguajes de programación funcional.

Sub-disciplinas y campos relacionados

Hay varios campos de estudio que o bien caen dentro de la teoría de lenguajes de programación, o bien tienen una profunda influencia en ella; muchos de estos se superponen considerablemente.

Teoría de los compiladores es la base formal sobre la escritura de *compiladores* (o más generalmente *traductores*); programas que traducen un programa escrito en un lenguaje a otra forma. Las acciones de un compilador se dividen tradicionalmente en *análisis sintáctico* (escanear y parsear), *análisis semántico* (determinando que es lo que debería de hacer un programa), *optimización* (mejorando el rendimiento indicado por cierta medida, típicamente la velocidad de ejecución) y *generación de código* (generando la salida de un programa equivalente en el lenguaje deseado; a menudo el set de instrucciones de una CPU).

La Teoría de tipos es el estudio de sistemas de tipos, que son "métodos sintácticos tratables para proveer la ausencia de ciertos comportamientos de programa mediante la clasificación de frases según los tipos de valores que computan." (Types and Programming Languages, MIT Press, 2002). Muchos lenguajes de programación se distinguen por las características de sus sistemas de tipos.

La Semántica formal es la especificación formar del comportamiento de programas de ordenador y lenguajes de programación.

La Transformación de programas es el proceso de transformar un programa de una forma (lenguaje) a otra forma; el análisis de programas es problema general de examinar un programa mediante la determinación de sus características clave (como la ausencia de clases de errores de programa).

Sistemas en tiempo de ejecución se refiere al desarrollo de entornos runtime para lenguajes de programación y sus componentes, incluyendo máquinas virtuales, recolección de basura, e interfaces para funciones externas.

Análisis comparativo de lenguajes de programación busca clasificar los lenguajes de programación en diferentes tipos basados en sus características; amplias categorías de diferentes lenguajes de programación se conocen frecuentemente como paradigmas de computación.

Metaprogramación es la generación de programas de mayor orden que, cuando se ejecutan, producen programas (posiblemente en un lenguaje diferente, o en un subconjunto del lenguaje original) como resultado.

Lenguajes dedicados son lenguajes construidos para resolver problemas en un dominio de problemas en particular de manera eficiente.

Además, PLT hace uso de muchas otras ramas de las matemáticas, ingeniería del software, lingüística, e incluso ciencias cognitivas

1.2 CATEGORIAS DE LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios:

- Según el nivel de abstracción
- Según el paradigma de programación que poseen cada uno de ellos

Según su nivel de abstracción

Lenguajes de Máquina

Están escritos en lenguajes directamente legibles por la máquina (computadora), ya que sus instrucciones son cadenas binarias (0 y 1). Da la posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior lo que supone una velocidad de ejecución superior, solo que con poca fiabilidad y dificultad de verificar y poner a punto los programas.

Lenguajes de bajo nivel

Los lenguajes de bajo nivel son lenguajes de programación que se acercan al funcionamiento de una computadora. El lenguaje de más bajo nivel por excelencia es el código máquina. A éste le sigue el lenguaje ensamblador, ya que al programar en ensamblador se trabajan con los registros de memoria de la computadora de forma directa. Ejemplo en lenguaje ensamblador intel x86:

```
;Lenguaje ensamblador, sintaxis Intel para procesadores x86
mov eax,1 ;mueve a al registro eax el valor 1
xor ebx, ebx ;pone en 0 el registro ebx
int 80h ;llama a la interrupción 80h (80h = 128 sistema decimal)
```

Ejecutar ese código en sistemas UNIX o basados en él, equivale a una función exit(0) (terminar el programa retornando el valor 0).

La principal utilización de este tipo de lenguajes es para programar los microprocesadores, utilizando el lenguaje ensamblador correspondiente a dicho procesador.

Lenguajes de medio nivel

Hay lenguajes de programación que son considerados por algunos expertos como lenguajes de medio nivel (como es el caso del lenguaje C) al tener ciertas características que los acercan a los lenguajes de bajo nivel pero teniendo, al mismo tiempo, ciertas cualidades que lo hacen un lenguaje más cercano al humano y, por tanto, de alto nivel. Ejemplo:

```
/*Lenguaje C*/

/*declaración de las funciones estandars de entrada y salida*/
#include <stdio.h>

int main(int argc, char **argv)
{
char *p; /*creamos un puntero a un byte*/
if(argc == 1){
```

```
printf("\nIngrese un argumento al programa\n");/*imprimimos el texto*/
return 1;
}
p = 0x30000/*el puntero apunta a 0x30000 */
*p = argv[1][0] /*el primer caracter del primer argumento lo copiamos a la posición 0x30000 */
return 0;
}
```

El ejemplo es muy simple y muestra a los punteros de C, éstos no son muy utilizados en lenguajes de alto nivel, pero en C sí.

Lenguajes de alto nivel

Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales, como el inglés. En BASIC, uno de los lenguajes de alto nivel más conocidos, los comandos como "IF CONTADOR = 10 THEN STOP" pueden utilizarse para pedir a la computadora que pare si el CONTADOR es igual a 10. Esta forma de trabajar puede dar la sensación de que las computadoras parecen comprender un lenguaje natural; en realidad lo hacen de una forma rígida y sistemática, sin que haya cabida, por ejemplo, para ambigüedades o dobles sentidos. Ejemplo:

```
{Lenguaje
Pascal}
program
suma;


var x,s,r:integer; {declaración de las
variables}


begin {comienzo del programa
principal}


writeln('Ingrese 2 números enteros');{imprime el texto} readln(x,s);
{lee 2 números y los coloca en las variables x y s} r:= x + s;
```

```
{suma los 2 números y coloca el resultado en r} writeln('La suma es ',r); {imrpime el resultado}


readln;
end.{termina el programa principal}
```

Ese es el lenguaje Pascal, muy utilizado por principiantes al aprender a programar.

Según el paradigma de programación

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro, sino que cada uno tiene ventajas y desventajas. Dependiendo de la situación un paradigma resulta más apropiado que otro.

Atendiendo al paradigma de programación, se pueden clasificar los lenguajes en:

- El paradigma imperativo o por procedimientos es considerado el más común y está representado, por ejemplo, por el C o por BASIC.
- El paradigma funcional está representado por la familia de lenguajes LISP (en particular
Scheme), ML o Haskell.
- El paradigma lógico, un ejemplo es PROLOG.
- El paradigma orientado a objetos. Un lenguaje completamente orientado a objetos es Smalltalk.

Nota: La representación orientada a objetos mejora la estructura de los datos y por lo tanto se ha aplicado a diferentes paradigmas como Redes de Petri, Imperativo Secuencial, Lógica de Predicados, Funcional, etc. No obstante, la manipulación no queda fundamentalmente afectada y por lo tanto el paradigma inicial tampoco a pesar de ser re-orientado a objetos.

Si bien puede seleccionarse la forma pura de estos paradigmas a la hora de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multiparadigma.

Actualmente el paradigma de programación más usado debido a múltiples ventajas respecto a sus anteriores, es la programación orientada a objetos.

Lenguajes imperativos

Son los lenguajes que dan instrucciones a la computadora, es decir, órdenes.

Lenguajes Funcionales

Paradigma Funcional: este paradigma concibe a la computación como la evaluación de funciones matemáticas y evita declarar y cambiar datos. En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas, más que en los cambios de estados y la ejecución secuencial de comandos (como lo hace el paradigma procedimental). Permite resolver ciertos problemas de forma elegante y los lenguajes puramente funcionales evitan los efectos secundarios comunes en otro tipo de programaciones.

Lenguajes Lógicos

La computación lógica direcciona métodos de procesamiento basados en el razonamiento formal. Los objetos de tales razonamientos son "hechos" o reglas "if then". Para computar lógicamente se utiliza un conjunto de tales estamentos para calcular la verdad o falsedad de ese conjunto de estamentos. Un estamento es un hecho si sus tuplas verifican una serie de operaciones.

Un hecho es una expresión en la que algún objeto o conjunto de objetos satisface una relación específica. Una tupla es una lista inmutable. Una tupla no puede modificarse de ningún modo después de su creación.[2]

Una regla if then es un estamento que informa acerca de conjuntos de tuplas o estamentos relacionados que pueden predecir si otras tuplas satisfacerán otras relaciones.

Un estamento que es probado verdadero como resultado de un proceso se dice que es una inferencia del conjunto original. Se trata por tanto de una descripción de cómo obtener la veracidad de un estamento dado que unas reglas son verdaderas.

La computación lógica está por tanto relacionada con la automatización de algún conjunto de métodos de inferencia.

Lenguajes orientados a objetos

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos.

Implementación

La implementación de un lenguaje es la que provee una manera de que se ejecute un programa para una determinada combinación de software y hardware. Existen básicamente dos maneras de implementar un lenguaje: Compilación e interpretación. Compilación es la traducción a un código que pueda utilizar la máquina. 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.

Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez 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.

Según su campo de aplicación.

Aplicaciones científicas.

En este tipo de aplicaciones predominan las operaciones numéricas o matriciales propias de algoritmos matemáticos. Lenguajes adecuados son FORTAN y PASCAL−

Aplicaciones en procesamiento de datos.

En estas aplicaciones son frecuentes las operaciones de creación, mantenimiento y consulta sobre ficheros y bases de datos. Dentro de este campo estarían aplicaciones de gestión empresarial, como programas de nominas, contabilidad facturación, control de inventario, etc. Lenguajes aptos para este tipo de aplicaciones son COBOL y SQL.

Aplicaciones de tratamiento de textos.

Estas aplicaciones están asociadas al manejo de textos en lenguaje natural. Un lenguaje muy adecuado para este tipo de aplicaciones es el C.

Aplicaciones en inteligencia artificial.

Dentro de este campo, destacan las aplicaciones en sistemas expertos, juegos, visión artificial, robótica. Los lenguajes más populares dentro del campo de la inteligencia artificial son LISP y PROLOG

Aplicaciones de programación de sistemas.

En este campo se incluirían la programación de software de interfaz entre el usuario y el hardware, como son los módulos de un sistema operativo y los traductores. Tradicionalmente para estas aplicaciones se utilizaba el Ensamblador, no obstante en la actualidad se muestran muy adecuados los lenguajes ADA, MODULA−2 y C.

Bibliografía Recomendada

- Conceptos de la Programación Orientada a Objetos
    1. Transparencias de clase
    2. Alfonseca, M. Alcalá, A. Programación Orientada a Objetos. Anaya Multimedia, Madrid, 1992.
    3. Beck, K.; Cunningham, W. A laboratory for teaching object-oriented thinking. Proc. of Object-Oriented Programming Systems, Languages and

Applications 1989 (OOPSLA '89). SIGPLAN Notices, Vol. 24, No. 10, October 89, pp 1-6.

4. Meyer, Bertrand. Object-Oriented Software Construction. Prentice Hall, segunda edición. Versión española: Construcción de software orientado a objetos, Prentice Hall Iberia, 1999.
5. Rubin, K.S.; Goldberg, A. Object Behaviour Analysis. Comm. of the ACM, vol. 35 no. 9, pp. 48-62, September 1992.

- Lenguajes de programación orientada a objetos
    1. IBM Smalltalk Tutorial. En Español.
    2. Alfonseca, M. Multimedia Ediciones S.A. Curso IBM de Programación, Unidades 38 a 41.
    3. Alfonseca, M. Frames, Semantic Networks and Object-Oriented Programming in APL2. IBM J. Res. Dev., 33:5, p. 502-510, Sep. 1989.
    4. Cox, Brad. Object-oriented Programming: an evolutionary approach. Addison-Wesley, 1986.
    5. Hopkins, T. A first Course in Smalltalk 80. Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.
    6. Lippman, S.B.; Stroustrup, B. Essential C++. Addison-Wesley Pub Co, 1999. ISBN: 0-201-48518-4.
    7. Sierra, A.; Alfonseca, M. Programación en C/C++. Anaya Multimedia, Madrid, 1999. ISBN: 84-415-0847-X.
    8. Stroustrup, B. The C++ Programming Language. Addison-Wesley Publishing Company, Reading, MA, edición especial, 1999. Existe edición española, Addison-Wesley, 2001. . .
- Análisis y diseño orientados a objetos
    1. Transparencias de clase
    2. Un problema de análisis y diseño
    3. Beck, K.; Cunningham, W. A laboratory for teaching object-oriented thinking. Proc. of Object-Oriented Programming Systems, Languages and Applications 1989 (OOPSLA '89). SIGPLAN Notices, Vol. 24, No. 10, October 89, pp 1-6.
    4. Booch, Grady. Object-Oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

ISBN: 0-8053-5340-2. 589 páginas. Traducción española, Addison-Wesley, 1996.

5. Booch, G.; Rumbaugh, J.; Jacobson, I.: Unified Modeling Language User Guide, Addison-Wesley, 1998. ISBN: 0-201-57168-4. Existe traducción española: "El Lenguaje Unificado de Modelado", Addison-Wesley, Madrid, 1999, ISBN: 84-7829-028-1.
6. Coad, Peter; Yourdon, E. Object-Oriented Analysis. Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.
7. Coad, Peter; Yourdon, E. Object-Oriented Design. Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.
8. Eriksson, H.E.; Penker, M.: UML Toolkit, IEEE, John Wiley and sons, ISBN 0-471-19161-2, 1998.
9. Jacobson, Ivar. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Publishing Co., Reading, Mass, 1992.
10. Jacobson, I.; Booch, G.; Rumbaugh, J.: The Unified Software Development Process, Addison-Wesley, Reading, Mass., 1999. ISBN: 0-201-57169-2. Versi¢n espa¤ola: Proceso Unificado de Desarrollo del Software, Addison Wesley, Madrid, 2000.
11. Lee, R.M.; Tepfenhart, W.M.: UML and C++, Prentice Hall, 1997.
12. Martin, James; Odell, James. Object-Oriented Analysis and Design. Prentice Hall, Englewood Cliffs, NJ, 1992.
13. Martin, James; Odell, James. Object-Oriented Methods: A Foundation. Prentice Hall, Englewood Cliffs, NJ, 1995.
14. Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorenson, W. Object-Oriented Modelling and Design. Prentice Hall, Inc., Englewood Cliffs, NJ, 1991. ISBN 0-13-630054-5. Traducción española, Prentice Hall, 1995.
15. Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, Mass., 1999. ISBN: 0-201-30998-X.
16. Shlaer, S.; Mellor, S.J. Object-Oriented System Analysis: Modelling the World in Data. Prentice Hall, Englewood Cliffs, NJ, 1988.
17. Stevens, P.; Pooley, R. Utilizaci¢n de UML. Addison Wesley, Madrid, 2002. ISBN: 84-7829-054-0.

18. Wirfs-Brock, R.; Wilkerson, B.; Wiener, L. Designing Object-Oriented Software. Prentice Hall, Englewood Cliffs, NJ, 1990. ISBN 0-13-629825-7.

- Middleware orientado a objetos
    1. El-Rewini, H., et al: "Object Technology: A Virtual Roundtable", Computer, pp. 58-72, Oct. 1995.
    2. Orfali, R.; Harkey, D.; Edwards, J.: "The Essential Distributed Objects Survival Guide", John Wiley & Sons, 1996. ISBN: 0-471-12993-3.
    3. Object Management Group: "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, revision 1.1. 1992. Pulse aquí para copiar la gramática de CORBA.

Sitios consultados

- http://djaramillo2dani.blogspot.mx/2011/04/guia-practica-uno-1-estructura-228106.html
- http://marcelo-trabajo.blogspot.mx/
- http://giomonografia.blogspot.mx/p/teorico.html
- http://diaolaya.weebly.com/1/post/2013/09/resumen-teora-de-compiladores.html
- http://lesbiamartinez.es.tl/teoria-sobre-el-lenguaje-de-programacion.htm
- https://es.scribd.com/doc/217164157/Tema-1
- http://cursos.aiu.edu/Lenguages%20de%20Programacion/PDF/Tema%201.pdf
- http://algoritmosylenguajes.blogspot.mx/2008/05/unidad-iii_31.html