Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde

FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N° 3

Introducción a la Programación

Introducción a la Programación

Hasta aquí, se ha aprendido a resolver problemas de complejidad leve, planteando algoritmos en pseudocódigo. A partir de este momento, se codificarán estas soluciones algorítmicas empleando un lenguaje de programación interpretable por una computadora, creando un *programa*. De esta forma, será la propia computadora la que ejecute el algoritmo y realice las acciones que conforman la solución.

En esta unidad temática se abordarán los conceptos básicos relativos a la creación de programas. En primer lugar se hará una revisión de las etapas más importantes de la resolución de problemas. Luego, se explicará la forma en que se ejecutan (prueban) los programas. Como en general los programas no funcionan correctamente -la primera vez que se ejecutan- será necesario eliminar los errores, proceso que se denomina depuración de los programas.

Existe una gran cantidad de lenguajes de programación. En esta materia se abordará uno en particular: el ANSI/ISO C++. Sin embargo, en esta unidad se verán las características generales que distinguen a unos lenguajes de otros y las ventajas relativas. Finalmente se propondrán algunos consejos acerca de un proceso que nunca debe faltar en la programación: la documentación. La documentación incluye toda la información que se puede suministrar acerca del programa y que no constituye el código del programa en si mismo.
Se utilizará para los ejercicios un compilador C++, que permitirá afirmar los conceptos teóricos desarrollados y editar, compilar y ejecutar los primeros programas.

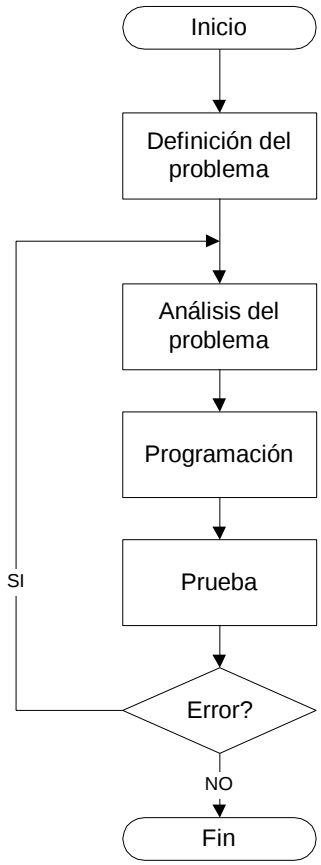
Revisión de Conceptos

Se estudió en la Unidad 1 el proceso de resolución de problemas computacionales, donde se distinguen las etapas siguientes.

- Definición del problema.
- Análisis del problema.
- Elección del método
- Codificación.
- Prueba.
- Depuración.
- Documentación.

Las etapas correspondientes a la **Codificación**, **Prueba** y **Depuración** constituyen el proceso de **Programación**, que se desarrollará a partir de aquí en la asignatura.

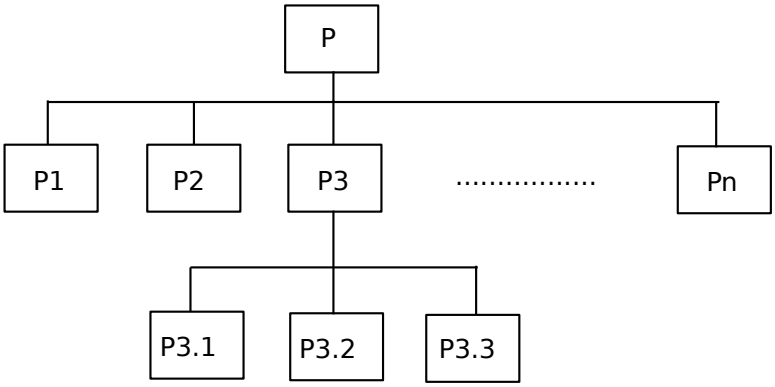
Diagrama de las Etapas de Resolución de Problemas



Nota: En este diagrama no se incluye el proceso de documentación ya que, como se verá luego, debe realizarse durante todas las etapas de resolución del problema.

Estrategia - Método Top Down

Diseñar una estrategia, consiste en dividir o descomponer el problema original en una sucesión de problemas más simples, de tamaño suficientemente pequeño como para que cada uno de ellos pueda ser comprendido en su totalidad. Esta técnica es conocida como *Top-Down* o de refinamientos sucesivos y, como se verá más adelante, se adapta perfectamente a la codificación de programas mediante un lenguaje modular y estructurado.



La estrategia nos define QUÉ hacer

Algoritmo

En esta etapa se plantea en base a la estrategia, el conjunto de acciones que permitirán resolver el problema, mediante pseudocódigo, diagrama de flujo, etc.

El algoritmo define CÓMO hacerlo

Programa

Un algoritmo codificado empleando un lenguaje de programación interpretable por una computadora constituye un programa.

Ejecución y prueba del programa

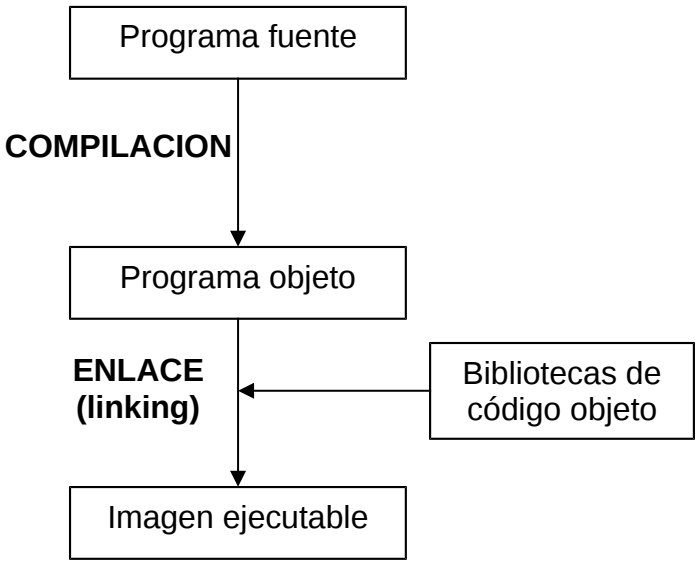
Para poder probar un *programa* escrito en un lenguaje de programación de Alto Nivel es necesario generar un código ejecutable. Este proceso puede efectuarse mediante la **COMPILACION** o mediante la **INTERPRETACIÓN** del código fuente que editó el programador.

Código Ejecutable

El Proceso de Compilación

El proceso de compilación es una traducción del código fuente a un código ejecutable por la computadora. El resultado de compilar un archivo o programa fuente es un nuevo archivo llamado imagen ejecutable.

Los archivos ejecutables pueden ser directamente utilizados por el usuario mediante una simple llamada desde el sistema operativo (por ejemplo un doble clic en un entorno gráfico tipo Windows). El archivo ejecutable ya no requiere del compilador ni del entorno que permitió su creación y puede ser utilizado en cualquier computadora (de plataforma compatible a la admitida por el compilador).



El Proceso de Interpretación

En este caso el intérprete procesa instrucción por instrucción, sin generar un código ejecutable completo. Cada vez que el usuario necesita ejecutar el programa deberá llamar al intérprete para que lo ejecute línea por línea.

Ventajas y Desventajas de Compiladores e Intérpretes

Ventajas

Compilación	**Interpretación**
Errores de sintaxis antes de la ejecución Velocidad de ejecución Protección del código.	Ejecución en una sola etapa Proceso interactivo de depuración

Desventajas

Compilación	**Interpretación**
Proceso en varias etapas y a menudo engorroso. (*) Depuración laboriosa	Errores de sintaxis detectados durante la ejecución. Baja velocidad de ejecución. Código abierto.

(*) NOTA: existen compiladores con entornos de desarrollo integrados (IDE) que simplifican esta tarea.

Depuración de Programas

Depuración o “debugging” significa eliminar los errores de un programa. En ciertos casos esa depuración es sencilla, pero a menudo constituye un proceso penoso. Esto depende de los tipos de errores de un programa.

Errores de un Programa

i) Errores en Tiempo de Compilación: son aquellos en los que se infringen las reglas que definen la estructura de las declaraciones y sentencias. Estos errores son denominados también errores de sintaxis. Los errores de sintaxis más comunes son: errores tipográficos, falta del punto y coma final y utilización de variables que no han sido declaradas previamente.
ii) Errores en Tiempo de Ejecución: los programas contienen estos errores cuando, a pesar de contar con sentencias sintácticamente válidas, se producen errores al ejecutar estas sentencias. Por ejemplo, un programa podría intentar utilizar un archivo que no existe en el disco o dividir un número por cero.
iii) Errores de Lógica: en muchos casos el programa arroja resultados incorrectos a pesar de que no posea errores de sintaxis y tampoco errores al ejecutarse. En general se trata de los casos en que el programa no realiza las tareas que originalmente se solicitaron en la definición del problema. Por ejemplo, puede que un problema requiera multiplicar dos números; por error, se diseña un algoritmo que los suma. Al probar el programa, se observaría que no hay errores de sintaxis, tampoco errores de ejecución y, sin embargo, al darle dos números al programa devolvería un valor inesperado.

Lenguajes de Programación

Los algoritmos se convierten en programas al ser codificados empleando lenguajes, cuyas instrucciones pueden ser procesadas por una computadora. Pero las computadoras procesan los programas de acuerdo al tipo de lenguaje utilizado.
Existen varias decenas de lenguajes de programación y a su vez muchas versiones de cada uno de ellos. En base a la similitud de estos lenguajes de programación respecto de nuestro lenguaje natural se los puede clasificar en 3 tipos: *Lenguajes de Máquina, Lenguajes de Bajo Nivel (Ensambladores) y Lenguajes de Alto Nivel.*

Lenguajes de Máquina

Los lenguajes de Máquina generan programas usando instrucciones que pueden ser resueltas directamente por el procesador de la computadora sin mediar traducciones.
Recuérdese que una computadora es un dispositivo electrónico que solo puede procesar dos estados de señales eléctricas: encendido y apagado; si se representan estas señales mediante un modelo matemático binario usando ceros y unos, es posible representar instrucciones que conformen un programa.
Por ejemplo para sumar dos números se puede escribir:

```
0110  1001  1010  1011
```

Este tipo de lenguaje tiene la ventaja de que sus programas pueden ser ejecutados directamente sin un proceso de traducción previo, lo cual implica una velocidad óptima del proceso. Como contrapartida, puede observarse que aún en problemas sencillos, el código es complejo de crear, carece de legibilidad, es muy complejo de depurar ante la presencia de errores, y tiene total dependencia del procesador de la computadora.
En respuesta a estos problemas se crearon lenguajes intermedios más cercanos al nivel del lenguaje natural que usan las personas para comunicarse.

Lenguajes de Bajo Nivel

Estos lenguajes pueden ser interpretados con más facilidad por una persona, pero la codificación continúa siendo una tarea compleja que requiere de una gran especialización por parte del programador. El lenguaje típico de bajo nivel es el ensamblador (Assembler Language), el cual está formado por sentencias nemotécnicas basadas en abreviaturas del inglés para representar acciones: ADD, MOV, SUB, DIV, etc. A modo de ejemplo, para sumar dos valores numéricos usando ensamblador, la acción sería:

```
ADD X, Y, SUMA
```

Lo cual se lee: *sumar el número almacenado en la posición de memoria **X** con el número de la posición **Y**. El resultado, ubicarlo en la posición de memoria representada por **SUMA**.*

La elaboración de soluciones a problemas grandes o complejos se hace muy engorrosa con este lenguaje. Además, está muy ligado al juego de instrucciones de la marca y modelo de cada microprocesador, lo cual hace que los programas sean poco portables. Su uso se limita al control de dispositivos electrónicos que requieren programas pequeños y sencillos, o partes de otros programas de computadoras.

SI bien su eficiencia es muy alta, un programa en ensamblador requiere de una traducción a código máquina para que la computadora ejecute sus instrucciones. El programa escrito en ensamblador es el programa fuente y el generado por la traducción es el programa objeto.

Lenguajes de Alto Nivel

Los lenguajes de alto nivel son los más populares entre los programadores, y su aparición permitió a la ingeniería del software abordar nuevos paradigmas y modelos, para resolver problemas de mayor complejidad. La formación de programadores es más rápida y su gran ventaja es la portabilidad: los programas son independientes del hardware.

Su denominación de *alto nivel* se debe a que su sintaxis es similar a la forma en que las personas se comunican y dan órdenes, usualmente en forma imperativa. Estos lenguajes están conformados por un conjunto de palabras y símbolos que tienen una relación directa con su significado: *`while, if, write, else, class, file, float, string, etc.`*

Los actuales lenguajes de alto nivel poseen sofisticados entornos de desarrollo que incluyen un amplio espectro de herramientas para la edición, compilación y depuración de los programas. La difusión de estos lenguajes, sumado a la accesibilidad de los productos de hardware ha revolucionado la industria del software en los últimos 15 años.

Como desventajas se debe mencionar el alto requerimiento de recursos de la computadora (memoria, espacio en disco, etc.) y el mayor tiempo de ejecución.

Existen numerosos lenguajes de alto nivel. Algunos de ellos: Fortran, Cobol, Basic, Pascal, Modula, C, C++, Java, SmallTalk, Vusal Basic, Visual C++, Delphi (Object Pascal), C++ Builder, Python, etc. Se describen sus características salientes y su origen en la lista siguiente

Algunos Lenguajes de Programación

- 1949: Primer *Assembler* (ShortCode).
- 1951: Grace Hooper escribe el primer compilador.
- 1957: FORTRAN: FORmula TRANslator. El lenguaje dominante en el ámbito científico. Velocidad de ejecución y tipos de datos de gran precisión. FORTAN IV, FORTRAN 77, FORTRAN 90, FORTRAN 95, FORTRAN 2000.
- 1958: LISP (John McCarthy, MIT): Nuevo paradigma de programación. Diseñado para inteligencia artificial.
- 1958: ALGOL: ALGOritmic Language. Propuesta de un comité científico internacional. para uso científico y académico. Primer lenguaje con gramática formal BNF (Bakcus-Naur Format). Recursividad. ALGOL68.
- 1959: COBOL: COmmon Business Oriented Language. Lenguaje orientado a problemas administrativos y contables, donde los formatos tienen suma importancia y no se requiere gran precisión numérica.
- 1964: BASIC (John Kemeny, Thomas Kurtz). Lenguaje intérprete de aplicación general. Muy sencillo de aprender.
- 1968: PASCAL (Niklaus Wirth, ETH). Paradigma de programación estructurada. Gran popularidad en ambientes académicos en los ´80 y parte de los ´90. Propone buenos hábitos de programación, fuertemente tipificado. Alocación dinámica de memoria. Las versiones de los 90 aceptan el modelo de objetos. Sucesores de Pascal: MODULA, MODULA-2.
- 1972: C (Dennis Ritchie, Bell Labs): Sucesor del lenguaje “B” . Estructuras de control similares a Pascal. Programas simples, rápidos y eficientes. Acepta aritmética de punteros, alocación dinámica de memoria. No es tan amigable como otros lenguajes.
- 1983: C++ (Bjarne Stroustroup): C con Clases. Extensión de C al modelo de objetos.
- 1987: Perl (Larry Wall): Lenguaje de scripts. Poderoso para el tratamiento de texto y programación en Internet.
- 1990: Python (G. van Rossum). Lenguaje de script creado para un sistema distribuido (Univ. Amsterdam). Descendiente de ABC. Posee influencias de C/ C++ y otros lenguajes.
- 1994: Java (Sun Labs). Lenguaje multiplataforma, compilación intermedia para *bytecode*, programación de aplicaciones para Internet.

Clasificación de los Lenguajes de Programación

Existen diferentes tipos de Lenguajes de Programación, pudiéndoselos clasificar según diferentes criterios. Un criterio de clasificación es según la metodología empleada para abordar el diseño del programa. Al momento de escoger entre uno u otro, deberá considerarse el tipo de problema que desea resolverse.

En esta materia, utilizaremos **Programación Estructurada**, basada principalmente en la utilización de estructuras de control secuenciales, condicionales e iterativas.

Otros tipos de lenguajes de programación, que utilizan diferentes técnicas para encarar el diseño de los programas son:

- Procedimentales
- Imperativos y Declarativos
- Orientados a objetos
- Orientados a eventos

Esta clasificación no es excluyente. Se pueden encontrar, por ejemplo, lenguajes orientados a objetos y manejados por eventos. De forma similar, algunos lenguajes son procedimentales y manejados por eventos. Algunos lenguajes, en particular, las versiones más recientes del lenguaje Pascal o C++ permiten programar en base a cualquiera de las clasificaciones anteriores e incluso combinarlas a todas en un solo programa. Sin embargo es conveniente utilizar de forma consistente un determinado modelo o metodología para abordar el problema.

Principales Ideas

- Para poder utilizar un programa la computadora debe interpretarlo o compilarlo previamente.
- Un compilador convierte el código fuente en una imagen ejecutable (por ejemplo, un archivo .exe).
- Los programas compilados son más rápidos de ejecutar que los interpretados y permiten ser utilizados sin necesidad de distribuir el código fuente entre los usuarios.
- Para que un programa funcione correctamente generalmente hay que corregir sus errores mediante el “debugging” o depuración.
- Existen tres tipos de errores: los de tiempo de compilación, los de tiempo de ejecución y los de lógica.
- Los lenguajes de programación pueden clasificarse en lenguaje máquina, de bajo nivel y de alto nivel, dependiendo de la cercanía de su sintaxis con el lenguaje natural de las personas. Más cerca de la máquina implica menor nivel, más cerca de los lenguajes humanos, mayor nivel.
- Los lenguajes de alto nivel se pueden clasificar según los paradigmas de programación que soportan: procedimentales, imperativos/declarativos, orientados a objetos y manejados por eventos. Es posible que algunos lenguajes combinen estos paradigmas.
- La documentación es muy importante y debe realizarse durante todo el proceso de resolución de un problema mediante computadoras. La documentación debe realizarse tanto en el programa (documentación interna) como en archivos o impresiones adicionales (documentación externa).