UNL FICH Facultad de Ingeniería y Ciencias Hídricas

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Ingeniería Informática

FUNDAMENTOS DE PROGRAMACIÓN

UNIDAD 5
INTRODUCCIÓN A LA PROGRAMACIÓN

2016

UNIDAD 5

Introducción a la Programación

Introducción

En las unidades anteriores se han resuelto numerosos problemas escribiendo algoritmos mediante un pseudolenguaje y/o diagramas de flujo. En esta unidad temática se abordarán los conceptos básicos relativos a la creación y codificación de programas utilizando un lenguaje de programación real. 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. Dado que en general los programas no funcionan correctamente -la primera vez que se ejecutan- será necesario analizar también los procedimientos para eliminar los errores, proceso que se denomina depuración de programas.

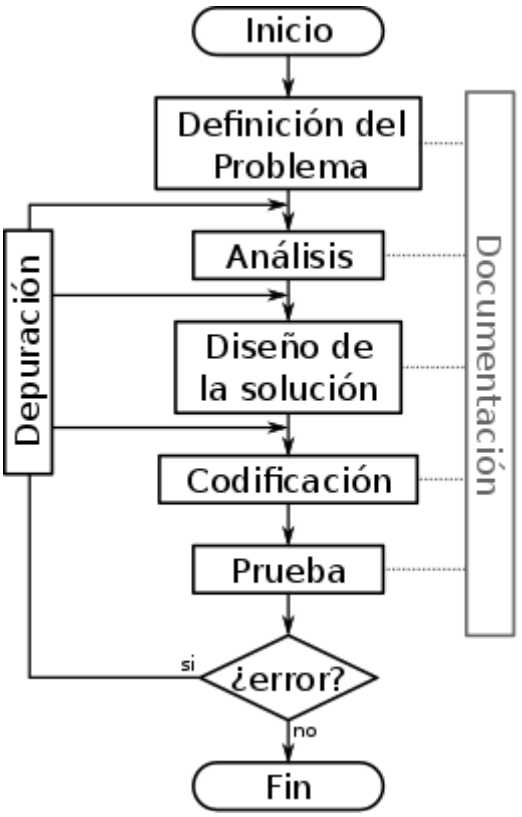
Un lenguaje de programación es un conjunto de reglas que definen cómo se debe describir la solución a un problema (en nuestro caso el algoritmo), para que esta pueda ser ejecutada por una máquina (la computadora). Existe una gran cantidad de lenguajes de programación. En esta unidad se verán las características generales que distinguen a unos lenguajes de otros y sus 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 su desarrollo, y que no constituye el código del programa en sí mismo.

Resolución de Problemas

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

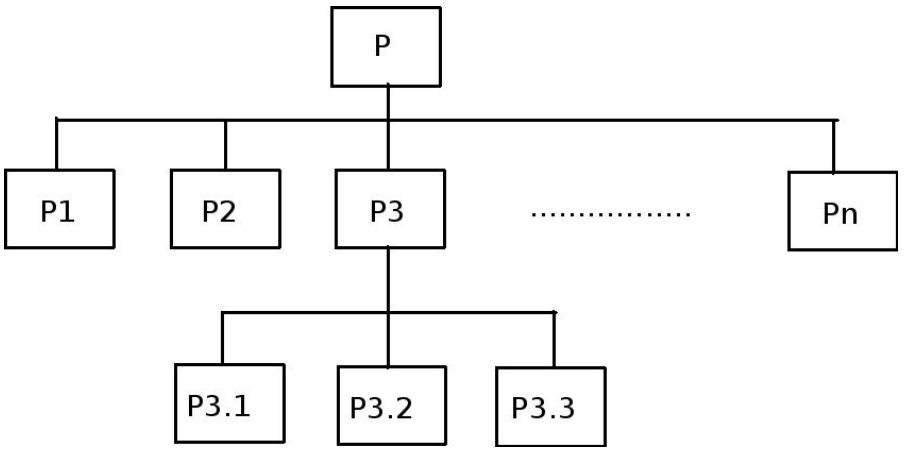
En las próximas unidades de esta asignatura nos centraremos en las etapas correspondientes a la **Codificación**, **Prueba** y **Depuración**.



Estrategia

"Saber controlar la complejidad es la esencia de la programación"
Brian Kernigan

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. La técnica presentada en la unidad 1 como *Top-Down* o de refinamientos sucesivos, como se verá más adelante, se adapta perfectamente a la codificación de programas mediante un lenguaje modular y estructurado. La idea central en esta estrategia consiste en identificar un problema complejo y dividirlo en subproblemas más simples de modo que la suma de los mismos sea equivalente al problema original. De esta forma, ya no es necesario resolver el problema original, sino que se pueden abordar cada uno de los subproblemas por separado, volviendo a aplicar la misma técnica para descomponerlos nuevamente si es necesario.



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.

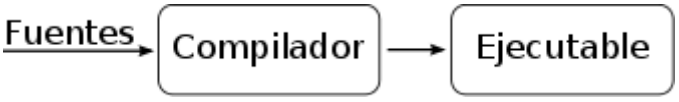
Ejecución y prueba del programa

Para poder probar un *programa* escrito en un lenguaje de programación de Alto Nivel es necesario generar un código ejecutable. Esto es, traducir el algoritmo escrito en un lenguaje de alto nivel a *"lenguaje de máquina"*, el lenguaje de más bajo nivel, que tiene correspondencia directa con las instrucciones que puede efectivamente ejecutar un microprocesador. Este proceso puede efectuarse mediante dos mecanismos diferentes que analizaremos a continuación: la **COMPILACIÓN** y la **INTERPRETACIÓN**.

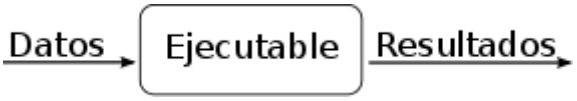
Compilación e Interpretación de Programas

El proceso de Compilación

El proceso de compilación es el proceso en el cual se realiza una traducción del código fuente a un código ejecutable por una computadora con un determinado sistema operativo (plataforma). Se denomina "compilador" al conjunto de herramientas encargadas de realizar dicha traducción. El resultado de compilar un archivo fuente es un nuevo archivo llamado *"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 compilación es generalmente irreversible, ya que cada instrucción de un lenguaje de alto nivel puede generar hasta miles de instrucciones en lenguaje de máquina, y además, existen en general muchas formas de traducir una misma instrucción, y es tarea del compilador utilizar la más adecuada en cada contexto.

Por esto, aunque para ejecutar un programa ya compilado no se requiera del código fuente ni de las herramientas de desarrollo, para modificarlo (por ejemplo, al encontrarle un error, o si se quieren nuevas funcionalidades) sí. Entonces, el programador debe distribuir el ejecutable a los usuarios, pero debe conservar el código fuente para poder modificarlo y recompilarlo si es necesario en el futuro.

El Proceso de Interpretación

En este caso el intérprete analiza cada instrucción del código fuente y realiza las acciones que la misma representa, sin generar en ningún momento un código ejecutable completo. Cada vez que el usuario necesita ejecutar el programa deberá llamar al intérprete para que lo ejecute.

Ventajas y Desventajas de Compilación vs. Interpretación

Compilación	**Interpretación**
– Proceso en varias etapas y a menudo engorroso[1]. + Detección de errores antes de la ejecución + Velocidad de ejecución + Protección del código – Depuración laboriosa	+ Ejecución en una sola etapa + Proceso interactivo de depuración – Errores de sintaxis detectados durante la ejecución. – Baja velocidad de ejecución. – Código abierto.

Depuración de Programas

El proceso de depuración o "debugging" es el proceso por el cual se identifican y eliminan los errores de un programa. En ciertos casos la depuración es sencilla, pero a menudo puede constituir un proceso complejo y tedioso. Esto depende de los tipos de errores de un programa, y de las herramientas de las que dispongamos en nuestro entorno de desarrollo para analizarlos[2]. La habilidad para analizar un programa y acotar o encontrar la causa de un error es una competencia fundamental en un buen programador.

Tipos de errores de un Programa

i) Errores en Tiempo de Compilación: son aquellos en los que se infringen las reglas del lenguaje, 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. En un lenguaje compilado, estos errores impiden que el programa comience a ejecutarse, ya que falla la traducción a código de máquina que realiza el compilador.

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 una variable

[1] Si bien es cierto que en general el proceso de compilación es más complejo que el de interpretación, hoy en día existen muchas herramientas para simplificarlo, y a lo largo de esta materia el alumno no se verá en la obligación de interactuar directamente con el compilador, sino que podrá utilizar un IDE que resuelva por completo este problema de forma transparente.

[2] En las subsiguientes unidades, cuando se estudie la programación en C++, se proporcionará material adicional para entender y aprender a utilizar las herramientas de depuración propias de los entornos de desarrollo para este lenguaje.

que durante una ejecución tome el valor cero. Estos errores impiden que el programa finalice su ejecución normalmente.

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; pero por error, se implementa un algoritmo que los suma. Al probar el programa, se observará que no hay errores de sintaxis, ni tampoco errores de ejecución; sin embargo, al ingresar dos números en el programa éste informará un resultado incorrecto.

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.
En la actualidad existen miles 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 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 del proceso óptima. 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 tipo de procesador de la computadora.
En respuesta a estos problemas se crearon lenguajes intermedios más abstractos, o más cercanos al lenguaje natural que usan las personas para comunicarse.

Lenguajes de Bajo Nivel

Estos lenguajes pueden ser interpretados con más facilidad por una persona que

un lenguaje de máquina, 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 conocido como “ensamblador” (Assembler Language), el cual está formado por sentencias nemotécnicas basadas en abreviaturas del inglés para representar instrucciones propias de un lenguaje de máquina: ADD, MOV, SUB, DIV, etc. Para sumar dos valores numéricos usando ensamblador:

```
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 estos lenguajes. Además, están muy ligados 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.

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.

Existen numerosos lenguajes de alto nivel. En general, los diferentes lenguajes de alto nivel proporcionan diferentes mecanismos de abstracción, dando lugar a distintas técnicas y paradigmas de programación. Algunos ejemplos de lenguajes populares son: Basic, Pascal, C, C++, C#, Java, Python, Go, etc.

En esta materia se abordará un lenguaje de programación de alto nivel en particular: C++. Se podrá utilizar para probar los ejemplos y resolver los ejercicios cualquier entorno de desarrollo para C++, que permita afirmar los conceptos teóricos desarrollados editando, compilando y ejecutando sus primeros programas.