Fundamentos de programación

C++

Más de 100 algoritmos codificados

EDITORIAL MACRO®

España - México - Colombia - Chile - Ecuador - Perú - Bolivia - Uruguay - Guatemala - Costa Rica



Fundamentos de programación C++
Autor: Ricardo Walter Marcelo Villalobos

© Derechos de autor registrados:
Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:
Empresa Editora Macro EIRL

Coordinadora de edición:
Cynthia Arestegui Baca

Diseño de portada:
Alejandro Marcas León

Corrección de estilo:
Milton A. Gonzales M.

Diagramación:
Katia Valverde Espinoza

Edición a cargo de:
© Empresa Editora Macro EIRL
Av. Paseo de la República N.° 5613 , Miraflores, Lima, Perú

Teléfono: (511) 748 0560
E-mail: proyecto@editorialmacro.com
Página web: www.editorialmacro.com

Primera edición: setiembre de 2008
Segunda edición: octubre de 2014
Tiraje: 1000 ejemplares

Impresión
Talleres gráficos de la Empresa Editora Macro EIRL
Jr. San Agustín N.° 612-624, Surquillo, Lima, Perú

ISBN N.° 978-612-304-235-6
Hecho el depósito legal en la Biblioteca Nacional del Perú N.° 2014-15111

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.

AUTOR

Ricardo Marcelo Villalobos

Profesional de sistemas y contabilidad, con más de diez años de experiencia en TI, ha participado como asesor y desarrollador en proyectos de *software* para diversas empresas privadas y públicas del país, como Minera del Hill, Aruntani, Verkaufen, MINSA, IPD; y transnacionales como Magna Rosseta Cerámica- MRC, en la cuales ha utilizado sus conocimientos de contabilidad y de ingeniería de *software* para realizar el análisis y diseños de *software* con RUP, UML y patrones de arquitectura. Asimismo, ha realizado diseños con lenguajes Java, .NET y PHP; también ha trabajado con base de datos Oracle, SQL Server, MySQL y PostgreSQL.

Asimismo, ha participado como expositor en diferentes universidades e institutos (Universidad Nacional de Ingeniería- CEPS-UNI, Universidad Nacional de Trujillo, Universidad César Vallejo de Trujillo, Universidad Nacional José Faustino Sánchez Carrión de Huacho, Instituto San Agustín, Instituto José Pardo, Instituto Manuel Seoane Corrales, Instituto La Reyna Mercedaria). Ha escrito libros, artículos y manuales de desarrollo de *software* (*Visual Basic Nivel III componentes*, *Oracle 10g*, manuales de VB.NET, ADO.NET, POO.NET, Access, Java POO, PHP Fundamentos, PHP POO).

En 2008 fue invitado por la Empresa Editora Macro para formar parte del *staff* de escritores, y salen a la luz cuatro obras relacionadas a los primeros pasos de la ingeniería de *software* (libros de fundamentos y más de 100 algoritmos con Visual Basic, Java, C++ y C#).

En la actualidad, difunde su experiencia como docente en la Universidad Nacional de Ingeniería (UNI-FIIS- CEPS-UNI) y el Instituto San Ignacio (ISIL); asimismo, realiza capacitaciones para empresas, como Telefónica del Perú, FAP, La Caja de Pensiones Militar Policial, ALPECO, Banco de Materiales, entre otros.

Agradecimiento

Es difícil dejar de mencionar a las personas que día a día fortalecen el conocimiento y la sabiduría de los demás. Me faltarían líneas en este libro para mencionar a todos, pero quiero agradecer, en primer lugar, a Dios y a mis padres.

También a personas muy especiales que, con su sabiduría y experiencia, han ayudado y permitido plasmar muchas de sus ideas en esta obra: Peggy Sánchez, Sergio Matsukawa, Gustavo Coronel, Gino Henostroza, Julio Flores, Joel Carrasco, Luis Zúñiga, Jesús Echevarria y todos mis alumnos y amigos en general.

Prólogo

Cómo no recordar las primeras clases de Algoritmo y la ilusión que todos tienen por aprender a programar. Esta obra plasma los primeros pasos que cualquier estudiante de la carrera de Ingeniería de sistemas, *software* e informática debe conocer para empezar a analizar, diseñar y codificar sus primeros algoritmos; y así pasar la barrera que todo programador debe dominar, que son las estructuras de control de flujo tales como if, switch (C++, Java y C#), select case (vb), while y for.

Este libro contiene nueve capítulos con más de cien algoritmos resueltos y otros ochenta propuestos; estoy seguro de que al concluir la lectura, el usuario formará parte del mundo de los desarrolladores de *software*. En el primer capítulo se desarrollan los conceptos generales de arquitectura de la PC, *hardware*, *software*, lenguajes de programación, metodología de algoritmos, diagramas de flujo, pseudocódigo, variables, constantes, instrucciones, entre otros.

El segundo apartado contiene diez algoritmos básicos para entender y resolver en forma simple los problemas de entrada, proceso (secuencial) y salida de los cálculos realizados. El tercer capítulo presenta quince algoritmos con la estructura más utilizadas en la solución de problemas, llamada if. En el cuarto capítulo se explica la forma más fácil de solucionar problemas sin el uso de if anidados y engorrosos. En el quinto capítulo se enseña a entender y dominar la estructura repetitiva, y a aplicar los conceptos de contador, acumulador, bucles, entre otros.

Debido a que muchas veces es más fácil resolver procesos repetitivos usando la estructura for, en el sexto apartado se encuentran quince problemas resueltos; aunque muchos de ellos pertenecen al capítulo anterior, esto servirá para analizar su simplicidad. En el sétimo apartado –tomando en cuenta que uno de los temas más utilizados en el manejo de colecciones de datos tiene que ver con los arreglos (*arrays*)– se explica el concepto y se resuelven problemas de arreglos, algoritmos de búsqueda y ordenación de datos. En el capítulo octavo, se explican y resuelven problemas con cadena de caracteres (texto). Finalmente, una de las mejores recomendaciones para resolver y reutilizar procesos es el concepto de divide y vencerás, por ello en el capítulo nueve se enseña cómo separar un problema en varias partes reutilizables.

ÍNDICE

Índice

Capítulo 1
Fundamentos de programación 13

1.1 Introducción 13
1.2 Computadora 14
1.3 Arquitectura de una computadora 14
1.4 Unidades de medida de almacenamiento 15
1.5 Sistemas de numeración 16
1.6 Conversión binario a decimal 16
1.7 Conversión decimal a binario 16
1.8 Representación de texto en el sistema binario 17
1.9 Representación binaria de datos no numéricos ni de texto 17
1.10 Los programas (*software*) 17
1.11 Lenguajes de programación 18
1.12 Traductores del lenguaje de programación 19
1.13 Ciclo de vida de un *software* 19
1.14 Algoritmo 20
1.14.1 Características que deben cumplir los algoritmos obligatoriamente 20
1.14.2 Características aconsejables para los algoritmos 21
1.14.3 Fases en la creación de algoritmos 21
1.14.4 Herramientas de un algoritmo 21
1.14.5 Instrucciones 23
1.15 Comentarios 24
1.16 Palabras reservadas 24
1.17 Identificadores 24
1.18 Variables 25
1.19 Constantes 26
1.20 Tipo de datos simples (primitivos) 26
1.21 Tipo de datos complejos (estructurados) 28
1.22 Operadores y expresiones 29
1.23 Control de flujo 32

Capítulo 2
Estructura secuencial 33

2.1 Estructura secuencial 33
Problema n.º 1 33
Problema n.º 2 34
Problema n.º 3 36
Problema n.º 4 37
Problema n.º 5 38
Problema n.º 6 40
Problema n.º 7 41
Problema n.º 8 43
Problema n.º 9 45
Problema n.º 10 46
2.2 Problemas propuestos 49

Capítulo 3
Estructura selectiva simple y doble 51

3.1 Introducción 51
3.2 Estructura selectiva simple 51
3.3 Estructura selectiva doble 52
3.4 Estructuras anidadas 52
Problema n.º 11 53
Problema n.º 12 55
Problema n.º 13 57
Problema n.º 14 59
Problema n.º 15 61
Problema n.º 16 63
Problema n.º 17 65
Problema n.º 18 66
Problema n.º 19 70
Problema n.º 20 71
Problema n.º 21 74
Problema n.º 22 76
Problema n.º 23 79
Problema n.º 24 81
Problema n.º 25 82
3.5 Problemas propuestos 85

Capítulo 4
Estructura selectiva múltiple 87

4.1 Introducción 87
4.2 Estructura selectiva múltiple 87
4.2.1 Estructura selectiva múltiple usando rangos 89
Problema n.º 26 89
Problema n.º 27 91
Problema n.º 28 93
Problema n.º 29 95
Problema n.º 30 97
Problema n.º 31 99
Problema n.º 32 101
Problema n.º 33 104
Problema n.º 34 106
Problema n.º 35 108
Problema n.º 36 112
Problema n.º 37 115
Problema n.º 38 117
Problema n.º 39 120
Problema n.º 40 123
4.3Problemas propuestos 129

Capítulo 5
Estructura repetitiva «Mientras» 131

5.1 Introducción 131
5.2 Contador 131
5.3 Acumulador 132
5.4 Salir del bucle 132
5.5 Continuar al inicio del bucle 132
5.6 Estructura repetitiva «Mientras» 133
5.7 Estructura repetitiva «Mientras» anidada 133
Problema n.º 41 134
Problema n.º 42 135
Problema n.º 43 137
Problema n.º 44 138
Problema n.º 45 140
Problema n.º 46 141
Problema n.º 47 143
Problema n.º 48 144

Problema n.º 49 146
Problema n.º 50 148
Problema n.º 51 149
Problema n.º 52 151
Problema n.º 53 152
Problema n.º 54 154
Problema n.º 55 156
5.8 Problemas propuestos 159

Capítulo 6
Estructura repetitiva «Para» 161

6.1 Introducción 161
6.2 Estructura repetitiva «Para» 161
6.3 Estructura repetitiva «Para» anidada 162
Problema n.º 56 162
Problema n.º 57 164
Problema n.º 58 165
Problema n.º 59 167
Problema n.º 60 168
Problema n.º 61 170
Problema n.º 62 171
Problema n.º 63 173
Problema n.º 64 176
Problema n.º 65 177
Problema n.º 66 179
Problema n.º 67 181
Problema n.º 68 182
Problema n.º 69 184
Problema n.º 70 186
6.4 Problemas propuestos 189

Capítulo 7
Estructuras de datos. Arreglos (vectores y matrices) 191

7.1 Introducción 191
7.2 *Arrays* (Arreglos) 192
7.3 Operaciones con *arrays* 192
7.4 Creación de *arrays* 193
7.5 Recorrido por los elementos del *array* 194

Problema n.º 71 195
Problema n.º 72 196
Problema n.º 73 198
Problema n.º 74 199
Problema n.º 75 201
Problema n.º 76 203
Problema n.º 77 205
Problema n.º 78 208
Problema n.º 79 209
Problema n.º 80 211
Problema n.º 81 213
Problema n.º 82 216
Problema n.º 83 218
Problema n.º 84 220
Problema n.º 85 223
7.6 Problemas propuestos 228

Capítulo 8
Cadenas de caracteres 229

8.1 Introducción 229
8.2 Juego de caracteres 229
8.3 Carácter (*char*) 230
8.4 Cadena de caracteres (*string*) 231
8.5 Operaciones con cadena 231
8.6 Concatenación 231
8.7 Comparación 232
8.8 Cálculo de longitud 232
8.9 Extracción de cadenas (subcadenas) 233
Problema n.º 86 234
Problema n.º 87 235
Problema n.º 88 237
Problema n.º 89 238
Problema n.º 90 241
Problema n.º 91 243
Problema n.º 92 244
Problema n.º 93 246
Problema n.º 94 247
Problema n.º 95 250
8.10 Problemas propuestos 253

Capítulo 9
SubAlgoritmos (procedimientos y funciones) 255

9.1 Introducción 255
9.2 Procedimientos 256
9.3 Funciones 256
9.4 Paso de parámetros 257
9.5 Parámetros por valor (entrada) 257
9.6 Parámetros por referencia (salida) 258
Problema n.º 96 259
Problema n.º 97 261
Problema n.º 98 263
Problema n.º 99 266
Problema n.º 100 269
Problema n.º 101 272
Problema n.º 102 275
9.7 Problemas propuestos 278

Capítulo 1

Fundamentos de programación

1.1 Introducción

En los primeros ciclos de toda carrera profesional relacionada a la ingeniería de sistemas, los estudiantes requieren entender, aprender y dominar los fundamentos de programación para resolver problemas que permitirán automatizar procesos usando la computadora.

Saber programar es la base de toda su carrera y, para conseguir este objetivo, he plasmado mi experiencia de docencia de mas de diez años en el campo de la Ingeniería de Sistemas. Sé que este libro le ayudará a resolver todas sus dudas y dominar las principales estructuras de programación.

Este libro contiene más de 100 algoritmos resueltos y codificados en el lenguaje de C++, el padre de los lenguajes de programación en la actualidad.



A continuación se describen los conceptos generales de los fundamentos de programación.

1.2 Computadora

Es un aparato electrónico que recibe datos (entrada), los procesa (instrucciones denominado programa) y devuelve información (salida), también conocido como ordenador o PC (Personal Computer). En la actualidad existe una variedad de computadoras para diferentes propósitos.

Servidores **Computadora personal** **Computadora portátil** **PDA**



1.3 Arquitectura de una computadora

Las computadoras tienen dos componentes principales que son el ***hardware*** y el ***software***, que trabajan en coordinación para llevar a cabo sus objetivos.

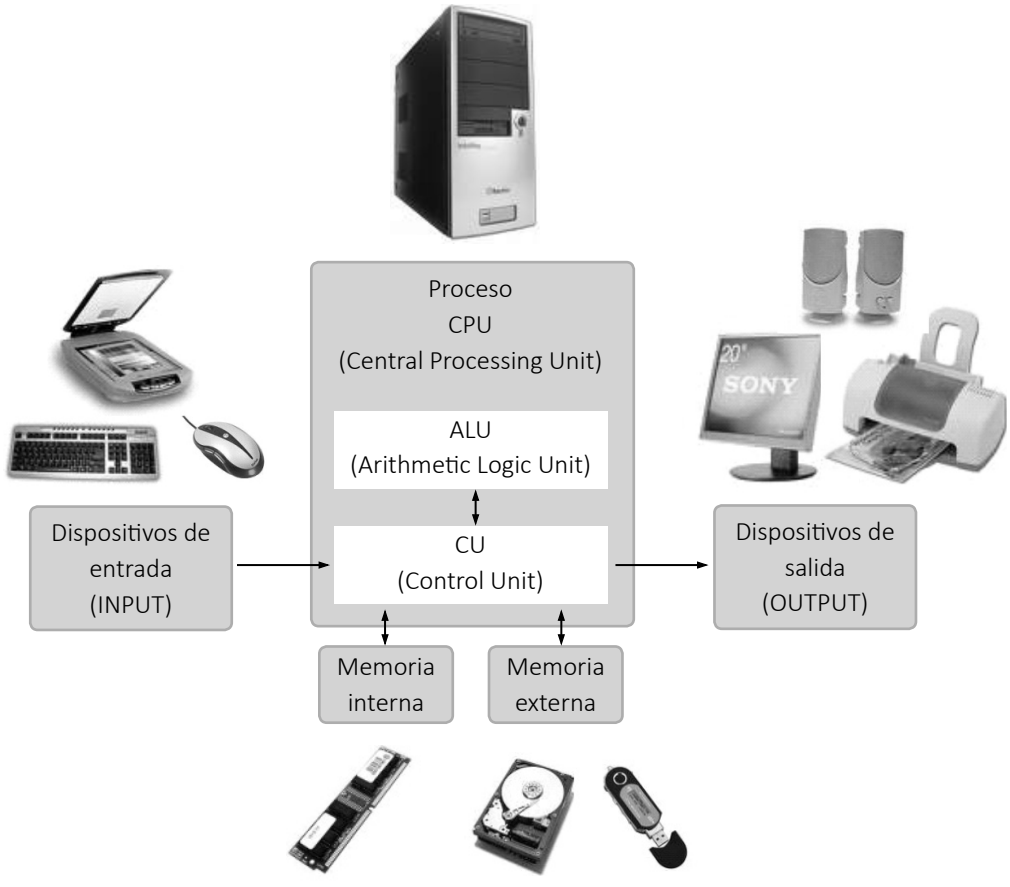
Hardware: *Hard* (duro) – *ware* (componente), representa la parte física de la computadora.



Software: *Soft* (blando) – *ware* (componente), representa la parte lógica de la computadora (los programas); estos se encuentran almacenados en los componentes físicos de la computadora, tales como memorias RAM, ROM, Discos Duros (*Hard Disk*), entre otros.



La siguiente figura muestra la arquitectura de computadora y sus principales componentes en coordinación.



1.4 Unidades de medida de almacenamiento

La memoria interna (RAM) y las memorias externas (disco duro) almacenan información. La información que se guarda y entiende la PC está en formato binario (0- 1).

BIT (BInary DigiT): El bit representa la unidad mínima de información que almacena una computadora.

BYTE: Está compuesto por 8 bit (01110011), entonces existe 2^8 = 256 combinaciones diferentes (tabla de código ASCII).

Por lo general, la información se representa por caracteres y cada carácter (número, letra, símbolo, etc.) es un byte. Para medir la información se utilizan múltiplos de bytes.

Byte	1 B		8 bits
Kilobyte	1 KB	2^{10} bytes	1024 bytes
Megabyte	1 MB	2^{20} bytes	1024 KB
Gigabyte	1 GB	2^{30} bytes	1024 MB
Terabyte	1 TB	2^{40} bytes	1024 GB

1.5 Sistemas de numeración

Todos los sistemas de numeración tienen una **base**, que es el número total de símbolos que utiliza el sistema. En el caso de la numeración decimal, la base es 10; en el sistema binario es 2.

El **Teorema Fundamental de la Numeración** permite saber el valor decimal que tiene cualquier número en cualquier base. Dicho teorema utiliza la fórmula:

$$\ldots + X_3 \cdot B^3 + X_2 \cdot B^2 + X_1 \cdot B^1 + X_0 \cdot B^0 + X_{-1} \cdot B^{-1} + X_{-2} \cdot B^{-2} + \ldots$$

Donde:

- **X_i:** Es el símbolo que se encuentra en la posición número **i** del número que se está convirtiendo. Teniendo en cuenta que la posición de las unidades es la posición 0 (la posición -1 sería la del primer decimal).
- **B:** Es la base del sistemas que se utiliza para representar al número.

Por ejemplo, si tenemos el número 153,6 utilizando el sistema octal (base ocho), el paso a decimal se sería:

$$1 \cdot 8^2 + 5 \cdot 8^1 + 3 \cdot 8^0 + 6 \cdot 8^{-1} = 64 + 40 + 3 + 6 \div 8 = 107{,}75$$

1.6 Conversión binario a decimal

El **Teorema Fundamental de la Numeración** se puede aplicar para saber el número decimal representado por un número escrito en binario. Así, para el número binario 10011011011 la conversión sería (los ceros se han ignorado):

$$1 \cdot 2^{10} + 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1243$$

1.7 Conversión decimal a binario

El método más utilizado consiste en ir haciendo divisiones sucesivas entre dos. Los restos son las cifras binarias. Por ejemplo, para pasar el 39:

```
39 ÷ 2 = 19 resto 1
19 ÷ 2 =  9 resto 1
 9 ÷ 2 =  4 resto 1
 4 ÷ 2 =  2 resto 0
 2 ÷ 2 =  1 resto 0
 1 ÷ 2 =  0 resto 1
```

Ahora las cifras binarias se toman al revés. Con lo cual, el número 100111 es el equivalente en binario de 39.

1.8 Representación de texto en el sistema binario

Puesto que una computadora no solo maneja números, habrán dígitos binarios que contengan información no traducible al sistema decimal. Todo depende de cómo se interprete esa traducción. Por ejemplo, en el caso del texto, lo que se hace es codificar cada carácter en una serie de números binarios. El código **ASCII** ha sido durante mucho tiempo el más utilizado, inicialmente era un código que utilizaba 7 bits para representar texto, lo que significaba que era capaz de codificar 127 caracteres. Por ejemplo, el número 65 (1000001 en binario) se utiliza para la **A** mayúscula.

Poco después apareció un problema: este código bastaba para los caracteres del inglés, pero no para otras lenguas. Entonces se añadió el octavo bit para representar otros 128 caracteres que son distintos, según idiomas (Europa Occidental usa unos códigos que no utiliza Europa Oriental).

Eso provoca que un código como el 190 signifique cosas diferentes si cambiamos de país. Por ello, cuando un ordenador necesita mostrar texto, tiene que saber qué juego de códigos debe de utilizar, lo cual supone un tremendo problema.

Una ampliación de este método de codificación es el código **UNICODE**, que puede utilizar hasta 4 bytes (32 bits), con lo que es capaz de codificar cualquier carácter en cualquier lengua del planeta, utilizando el mismo conjunto de códigos. Poco a poco se ha ido extendiendo cada vez más, pero la preponderancia histórica que ha tenido el código ASCII complica su popularidad.

1.9 Representación binaria de datos no numéricos ni de texto

En el caso de datos más complejos (imágenes, vídeo, audio) se necesita una codificación más compleja. Además, en estos datos no hay estándares, por lo que hay decenas de formas de codificar. En el caso, por ejemplo, de las imágenes, una forma básica de codificarlas en binario es la que graba cada **píxel** (cada punto distinguible en la imagen) mediante **tres bytes**: el primero graba el nivel de **rojo**; el segundo, el nivel de **azul;** y el tercero, el nivel de **verde**. Y así por cada píxel.

Por ejemplo, un punto en una imagen de color rojo puro:

11111111 00000000 00000000

Naturalmente, en una imagen no solo se graban los píxeles sino el tamaño de la imagen, el modelo de color, etc. De ahí que representar estos datos sea tan complejo para el ordenador (y tan complejo entenderlo para nosotros).

1.10 Los programas (*software*)

Los programas o *software* son un conjunto de instrucciones ordenadas para ejecutarse de forma rápida y precisa en una computadora. El *software* se divide en dos grupos: *software* de **sistema operativo** y ***software* de aplicaciones**.

El proceso de escribir un programa se denomina **programación**, y el conjunto de instrucciones que se utilizan para escribir un programa se llama **lenguaje de programación**.

1.11 Lenguajes de programación

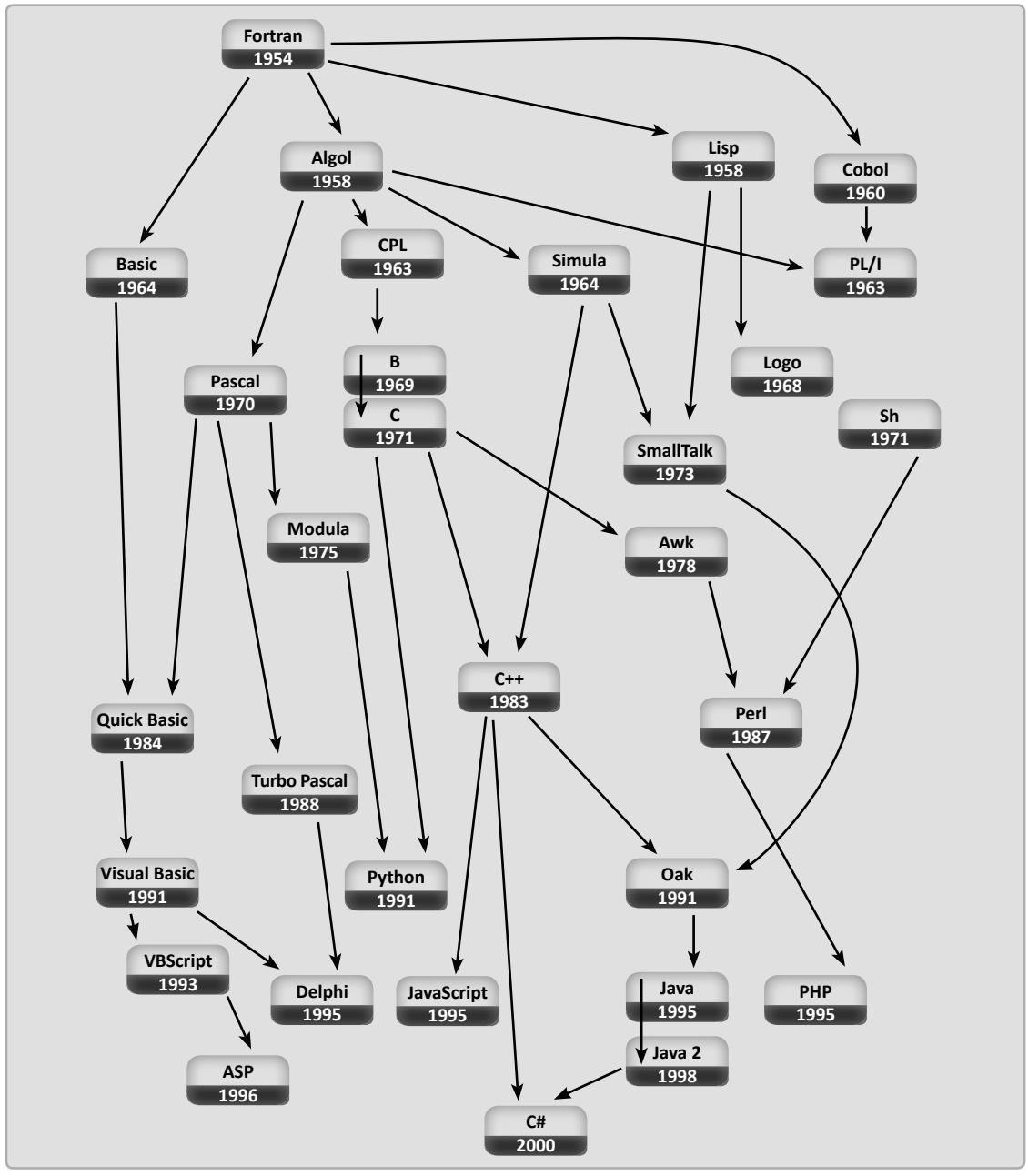
Sirve para escribir programas y permite la comunicación usuario (programador) versus máquina (PC).

Existen tres tipos de lenguajes de programación:

• **Lenguaje de máquina:** Programación binaria, difícil de programar y dependiente de la máquina.

• **Lenguaje de bajo nivel (ensamblador):** Usa símbolos nemotécnicos, necesita ser traducido al lenguaje de máquina y sigue siendo dependiente.

• **Lenguaje de alto nivel:** Cercano al lenguaje natural, tiempo de programación relativamente corto, es independiente de la máquina. A continuación se muestra un plano de la evolución de los lenguajes de programación de alto nivel.



1.12 Traductores del lenguaje de programación

Son programas que traducen los **códigos fuentes** (programas escritos en un lenguaje de alto nivel) a **código máquina**.

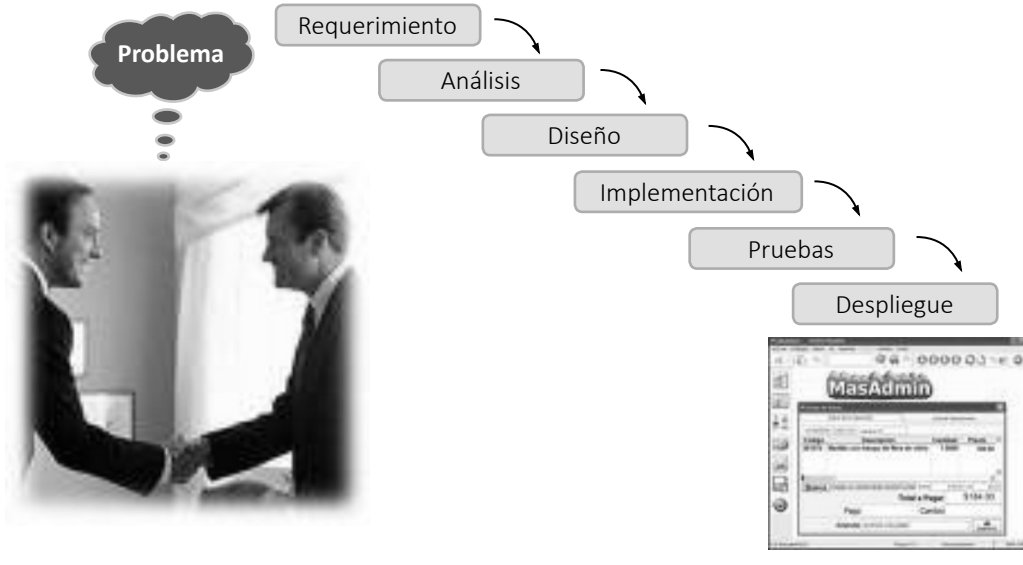
Los traductores se dividen en:

- **Intérpretes:** Traducción y ejecución secuencial (línea por línea), ejecución lenta.
- **Compiladores:** Traduce el código fuente a programa objeto (ejecutable código máquina). Ejecución rápida.

1.13 Ciclo de vida de un *software*

La construcción de un ***software,*** por más pequeño que sea, involucra las siguientes etapas:

- **Requerimiento:** Enunciado del problema a resolver.
- **Análisis:** ¿Qué? (Entender el problema – entrada – proceso – salida).
- **Diseño:** ¿Cómo? (Resolver el problema – algoritmo – diagrama de flujo – diseño de interfaz de usuario).
- **Implementación:** ¿Hacerlo? (Codificación / Programar).
- **Pruebas:** ¿Funciona? (Verificar / Comprobar).
- **Despliegue:** ¿Instalar? (Distribuir el programa).



1.14 Algoritmo

Método que describe la solución de un problema computacional mediante una serie de pasos precisos, definidos y finitos.

- **Preciso:** Indicar el orden de realización en cada paso.
- **Definido:** Al epetir los pasos n veces se obtiene el mismo resultado.
- **Finito:** Tiene un número determinado de pasos.

La solución de un algoritmo debe describir tres partes:

- **Entrada:** Datos que se necesitan para poder ejecutarse.
- **Proceso:** Acciones y cálculos a realizar.
- **Salida:** Resultado esperado.



La palabra algoritmo procede del matemático árabe Mohamed Ibn Al Kow Rizmi, quien escribió entre los años 800 y 825 su obra ***Quitad Al Mugabala***, donde recogió el sistema de numeración hindú y el concepto del cero. Fibonacci, tradujo la obra al latín y la llamó ***Algoritmi Dicit***.

El lenguaje algorítmico es aquel que implementa una solución teórica a un problema, indicando las operaciones a realizar y el orden en el que deben efectuarse. Por ejemplo, en el caso de que nos encontremos en casa con un foco malogrado de una lámpara, un posible algoritmo sería:

a. Comprobar si hay foco de repuesto.
b. En el caso de que haya, sustituir el foco anterior por el nuevo.
c. Si no hay foco de repuesto, bajar a comprar uno nuevo en la tienda y ponerlo en lugar del malogrado.

Los algoritmos son la base de la programación de ordenadores, ya que los **programas de ordenador** se pueden entender como algoritmos escritos en un código especial, entendible por un ordenador.

La desventaja del diseño de algoritmos radica en que no podemos escribir lo que deseemos; el lenguaje ha utilizar no debe dejar posibilidad de duda, debe recoger todas las posibilidades.

1.14.1 Características que deben de cumplir los algoritmos obligatoriamente

- **Un algoritmo debe resolver el problema para el que fue formulado**. Lógicamente, no sirve un algoritmo que no resuelve ese problema. En el caso de los programadores, a veces crean algoritmos que resuelven problemas diferentes al planteado.
- **Los algoritmos son independientes del lenguaje de programación**. Los algoritmos se escriben para poder ser utilizados en cualquier lenguaje de programación.
- **Los algoritmos deben ser precisos**. Los resultados de los cálculos deben ser exactos, de manera rigurosa. No es válido un algoritmo que sólo aproxime la solución.
- **Los algoritmos deben ser finitos**. Deben de finalizar en algún momento. No es un algoritmo válido aquel que produce situaciones en las que el algoritmo no termina.
- **Los algoritmos deben poder repetirse**. Deben permitir su ejecución las veces que haga falta. No son válidos los que tras ejecutarse una vez ya no pueden volver a hacerlo por la razón que sea.

1.14.2 Características aconsejables para los algoritmos

- **Validez:** Un algoritmo es válido si carece de errores. Un algoritmo puede resolver el problema para el que se planteó y, sin embargo, no ser válido debido a que posee errores.
- **Eficiencia:** Un algoritmo es eficiente si obtiene la solución al problema en poco tiempo. No lo es si tarda en obtener el resultado.
- **Óptimo:** Un algoritmo es óptimo si es el más eficiente posible y no contiene errores. La búsqueda de este algoritmo es el objetivo prioritario del programador. No siempre podemos garantizar que el algoritmo hallado sea el óptimo, a veces sí.

1.14.3 Fases en la creación de algoritmos

Hay tres fases en la elaboración de un algoritmo:

a. **Análisis:** En esta se determina cuál es exactamente el problema a resolver. Qué datos forman la entrada del algoritmo y cuáles deberán obtenerse como salida.

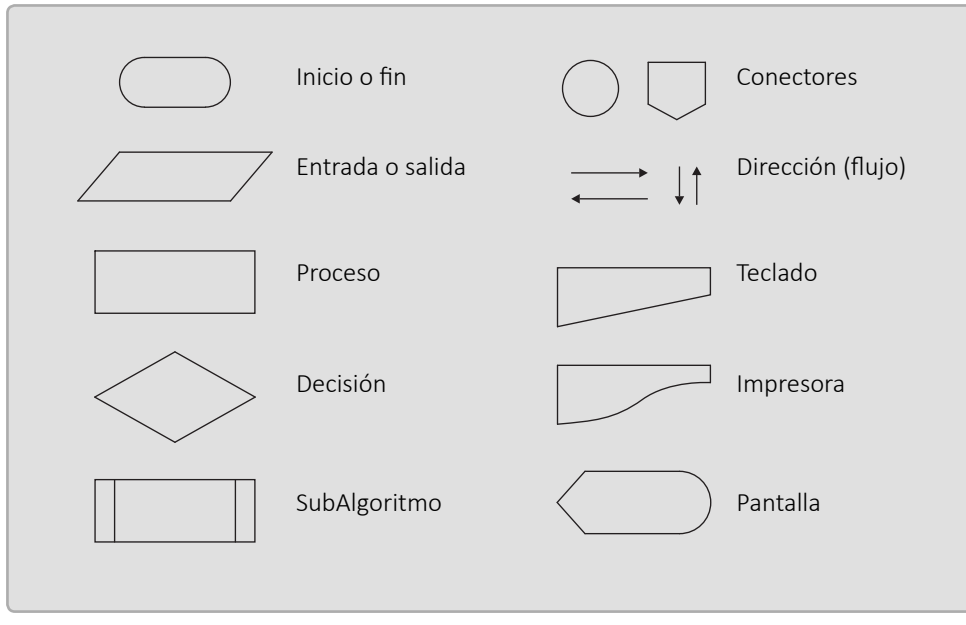
b. **Diseño:** Elaboración del algoritmo.

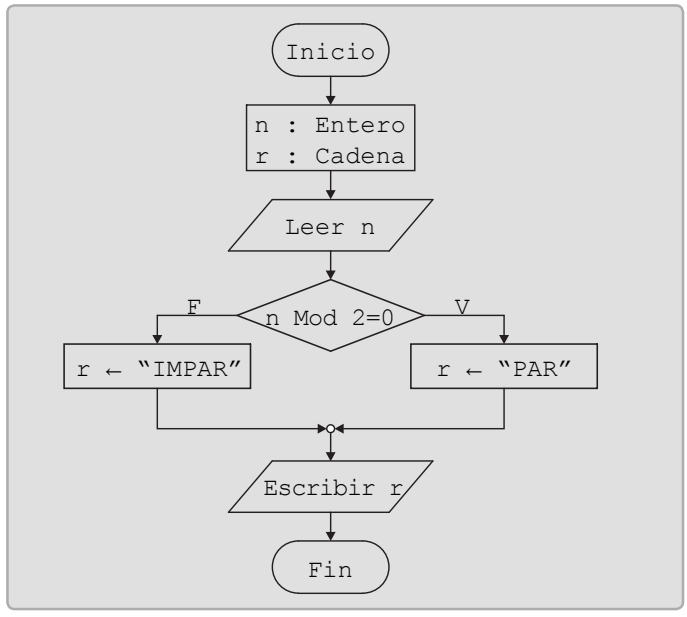
c. **Prueba:** Comprobación del resultado. Se observa si el algoritmo obtiene la salida esperada para todas las entradas.

1.14.4 Herramientas de un algoritmo

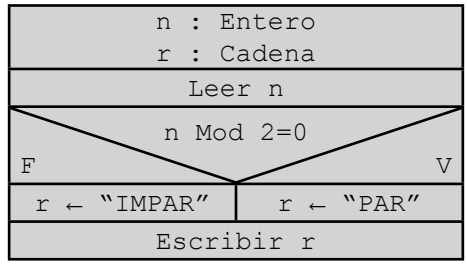
Para expresar la solución de un problema se pueden usar diferentes herramientas de programación, tales como diagrama de flujo (***flow chart***), diagrama N-S (Nassi Schneiderman), pseudocódigo.

• **Diagrama de flujo:** Es una representación gráfica que utiliza símbolos normalizados por ANSI, y expresa las sucesivas instrucciones que se deben seguir para resolver el problema. Estas instrucciones no dependen de la sintaxis de ningún lenguaje de programación, sino que deben servir fácilmente para su transformación (codificación) en un lenguaje de programación.





• **Diagrama de Nassi Scheneiderman (N-S):** Conocido también como el diagrama de Chapin, es como un diagrama de flujo pero sin flechas y con cajas continuas.



• **Pseudocódigo:** Permite expresar las instrucciones en un lenguaje común (ingles, español, etc.) para facilitar tanto la escritura como la lectura de la solución de un programa. No existen reglas para escribir pseudocódigo.

```
Inicio

   //Variables
   n : Entero
   r : Cadena

   //Entrada
   Leer n

   //Proceso
   Si n Mod 2 = 0 Entonces
       r ← "PAR"
   SiNo
       r ← "IMPAR"
   Fin Si

   //Salida
   Escribir r

Fin
```

1.14.5 Instrucciones

Son las acciones que debe realizar un algoritmo para resolver un problema. Las instrucciones más comunes son las siguientes:

A. Instrucción de inicio/ fin: Representa el inicio y fin de un algoritmo.

Diagrama de flujo	Pseudocódigo
Inicio	Inicio
Fin	Fin

B. Instrucción de asignación: Representa la asignación de un valor a una variable, se puede representar usando una flecha o el símbolo de igualdad, el cual es usado por muchos de los lenguajes de programación.

Diagrama de flujo	Pseudocódigo
N ← 10	N ← 10
N = 10	N = 10

C. Instrucción de lectura: Representa el ingreso de datos mediante un dispositivo de entrada, que muchas veces es representado por un símbolo de teclado.

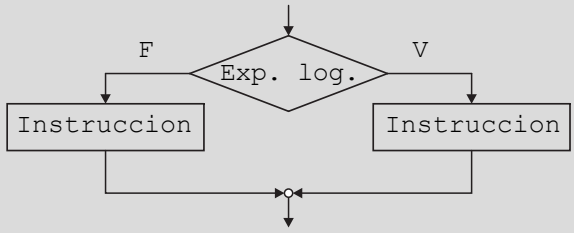
Diagrama de flujo	Pseudocódigo
Leer N	Leer N
Leer N	Leer N

D. Instrucción de escritura: Representa la salida de la información mediante un dispositivo de salida, puede ser representado por el símbolo de entrada/salida, por símbolo de pantalla o impresora.

Diagrama de flujo	Pseudocódigo
Escribir R	Escribir R
Escribir R	Escribir R
Escribir R	Escribir R

E. Instrucción de bifurcación: Cambian el flujo del programa según el resultado de una expresión lógica (condición).

Diagrama de flujo



Pseudocódigo

Si <Exp. Log> **Entonces**
<Instrucción>
SiNo
<Instrucción>
Fin Si

1.15 Comentarios

Permiten describir y explicar, además sirve como ayuda para recordar y entender las operaciones que se van a ejecutar. Los comentarios no son instrucciones, por lo tanto al ser traducido el código fuente a código binario (tiempo de compilación), los lenguajes de programación los ignoran. Dependiendo el lenguaje de programación los comentarios se escriben usando cierta simbología, en este libro usaremos el símbolo **//** en los pseudocódigos para colocar comentarios.

Ejemplo pseudocódigo

```
//Variables
N : Entero
```

C++

```
//Variables
int N;
```

1.16 Palabras reservadas

Son palabras usadas por el lenguaje de programación que no deben ser utilizadas como identificadores de variables, funciones, entre otros.

- **Algunas de las palabras reservadas de C++**

`short, int, float, double, if, for, switch ...`

1.17 Identificadores

Son los nombres que asignamos a las variables, constantes, funciones, objetos, entre otros; y no pueden coincidir con las palabras reservadas porque ocasionaría ambiguedad, y el compilador no lo entendería. Por lo general, los identificadores deben de cumplir las siguientes reglas:

- Deben comenzar por una letra. Evite usar **ñ** y tilde.
- No debe coincidir con palabras reservadas del lenguaje de programación que está utilizando.

Error de compilación C++

```
// Identificador de Variable es if
// y esta es palabra reservada
int if;
```

1.18 Variables

Representa un espacio de memoria RAM, el cual guarda un valor que servirá para algún proceso en particular; dicho valor puede ser modificado en cualquier momento. Las variables tienen, por lo general, un identificador (nombre) y, asignado, el tipo de dato que se esta utilizando; es decir, si almacena un número (entero), si es texto o alfanumérico (cadena), si es un valor verdadero o falso (lógico) llamado también booleano.

Ejemplo pseudocódigo

```
//Variables
N : Entero

C++
//Variables
int N;
```

Para asignarle un valor, usamos el operador de asignación, para algoritmos usaremos (←) o (=); este último es el más usado por los lenguajes de programación.

Ejemplo pseudocódigo

```
//Asignar un valor
N ← 10
//Cambiar su valor
N ←50

C++
//Asignar un valor
N = 10;
//Cambiar su valor
N = 50;
```

1.19 Constantes

Representa un espacio de memoria RAM, el cual guarda un valor que servirá para algún proceso en particular; dicho valor permanece fijo, es decir, no puede cambiarse en la ejecución del programa. Las constantes tienen, al igual que las variables, un identificador (nombre) y un tipo de dato.

Ejemplo pseudocódigo

```
//Constantes
PI ← 3.14159 : Real
//Error ya no puede modificarlo
PI ← 3.14

C++
//Constantes
const float PI = 3.14159F;
//Error ya no puede modificarlo
PI = 3.14;
```

1.20 Tipo de datos simples (primitivos)

Al declarar una variable, debemos indicar el tipo de dato que es permitido almacenar en dicha variable. Cada lenguaje de programación trabaja con una variedad de tipo de datos; por lo general, todos usan los llamados «primitivos», que son los siguientes:

A. Entero: Representan los números enteros (no almacena decimales).

Ejemplo pseudocódigo

```
//Crear la variable
//(identificador y tipo de dato)
N : Entero

//Asignar un valor
//(identificador, operador de asignación y valor)
N ← 15
```

En el lenguaje de C++ el tipo entero se puede trabajar con **short**, **int** y **long**; la diferencia está en que unos almacenan rangos de números diferentes, llamados también «entero corto» y «entero largo».

Ejemplo C++

```
//Entero corto
short N;
//Asignar un valor (error de desbordamiento)
//Sobrepaso su limite (rango)
```

```
N = 45000
//Entero largo
int N;
long N;
//Asignar un valor
N = 4500099;
```

B. Real: Representan los números reales (almacena decimales).

Ejemplo pseudocódigo

```
//Crear la variable
//(identificador y tipo de dato)
N : Real
//Asignar un valor
//(identificador, operador de asignación y valor)
N ← 15.75
```

En el lenguaje de Java, el tipo real se puede trabajar con **float** o **double**, la diferencia está en la cantidad de decimales que pueden almacenar, llamados también «precisión simple» y «precisión doble».

```
//Precisión simple
float N;

//Se redondea a 15.123457
N = 15.12345678;

//Precisión doble
double N;

//Lo almacena sin redondear 15.12345678
N = 15.12345678;
```

C. Carácter: Representa un carácter de cualquier tipo: texto, número, símbolo, etc. El valor se coloca entre comillas simples.

Ejemplo pseudocódigo

```
//Crear la variable
R : Caracter
//Asignar un valor
R ← 'A'
R ← '9'
R ← '*'
```

Ejemplo C++

```
'Crear la variable
char R;
'Asignar un valor
R = 'A';
R = '9';
R = '*';
```

E. Lógico: Representan los valores «verdadero» o «falso», conocidos también como boolean, no se colocan comillas simple ni dobles.

Ejemplo pseudocódigo

```
//Crear la variable
L : Logico
//Asignar un valor
L ← VERDADERO
L ← FALSO
```

En C++ se utiliza el tipo de dato llamado «boolean», para almacenar valores lógicos.

Ejemplo C++

```
'Crear la variable
bool L;
//Asignar un valor
L = true;
L = false;
```

1.21 Tipo de datos complejos (estructurados)

Son aquellos que están constituidos por tipos de datos simples y definen una estructura de datos, un ejemplo claro es el tipo cadena, que está compuesto por un conjunto de caracteres (tipo de dato carácter). Existe una variedad de tipo de datos complejos, el enfoque de este libro es algorítmico y solo tocaremos dos tipos de datos complejos: cadena y arreglos. Los libros que profundizan el tema se llaman libros de estructura de datos.

A. Cadena: Representa un conjunto de caracteres, internamente es un arreglo de caracteres; por lo general, se representa con comillas dobles.

Ejemplo pseudocódigo

```
//Crear la variable
R : Cadena
//Asignar un valor
R ← "ricardomarcelo@hotmail.com"
```