Fundamentos de Programación

Diana Teresa Gómez Forero
Urbano Eliécer Gómez Prada
Jairo Bernardo Viola Villamizar

Escuela de Ingeniería Escuela de Ingeniería

Diana Teresa Gómez Forero

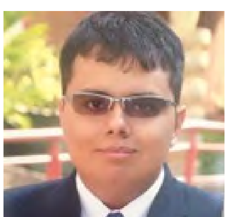
Ingeniera de Sistemas con Maestría en Ciencias de la Computación, y en Informática. Como docente de Lógica de Programación y de Análisis y Diseño de Algoritmos en la Universidad Pontificia Bolivariana, Seccional Bucaramanga, ha experimentado múltiples estrategias pedagógicas para facilitar el aprendizaje de la algoritmia, su aplicación en diferentes lenguajes de programación y, en general, para desarrollarla profesionalmente.

Contacto: diana.gomez@upb.edu.co

Urbano Eliécer Gómez Prada

Ingeniero de Sistemas y Magíster en Ingeniería en el área de Informática y Ciencias de la Computación. Docente de experiencia en diversas asignaturas de la línea de Ingeniería de Software, Pensamiento Sistémico, Modelado y Simulación y Tecnologías aplicadas a la Educación. Promueve el aprendizaje como un proceso de representación de situaciones nuevas a partir de esquemas conocidos en donde se amplía la cobertura y complejidad. Ha desarrollado proyectos de software de gestión organizacional para Pymes y sistemas de información y capacitación o simulación para acercar las TIC al sector agroindustrial.

Contacto: urbano.gomez@upb.edu.co



Jairo Bernardo Viola Villamizar

Ingeniero Electrónico, Magíster en Ingeniería Electrónica, docente de Lógica y Algoritmia y de Programación Estructurada en la Universidad Pontificia Bolivariana, Seccional Bucaramanga. Como docente de Ingeniería Electrónica, comprende la importancia del proceso de la programación como parte fundamental de la formación de un ingeniero electrónico y ha empleado diferentes técnicas para mejorar el aprendizaje de la programación en los estudiantes. Sus áreas de investigación son el control fraccionario, la robótica industrial, el procesamiento de imágenes y el reconocimiento de patrones.

Contacto: jairo.viola@upb.edu.co

Escuela de Ingeniería Escuela de Ingeniería

Fundamentos de Programación

Diana Teresa Gómez Forero
Urbano Eliécer Gómez Prada
Jairo Bernardo Viola Villamizar

005.1
G633

Gómez Forero, Diana Teresa, autor
Fundamentos de Programación / Diana Teresa Gómez Forero, Urbano Eliécer Gómez Prada, Jairo Bernardo Viola Villamizar -- Medellín: UPB. Seccional Bucaramanga, 2018.
128 p: 16.5 x 23.5 cm.
ISBN: 978-958-764-543-9

1. Algortitmos – 2. Informática – 3. Lenguajes de programación – I. Gómez Prada, Urbano Eliécer, autor – II. Viola Villamizar, Jairo Bernardo, autor

CO-MdUPB / spa / rda
SCDD 21 / Cutter-Sanborn

© Diana Teresa Gómez Forero
© Urbano Eliécer Gómez Prada
© Jairo Bernardo Viola Villamizar
© Editorial Universidad Pontificia Bolivariana
Vigilada Mineducación

Fundamentos de Programación
ISBN: 978-958-764-543-9
Escuela de Ingenierías
Facultad de Ingeniería de Sistemas e Informática
Dirección General de Investigaciones
Seccional Bucaramanga

Arzobispo de Medellín y Gran Canciller UPB: Mons. Ricardo Tobón Restrepo
Rector General: Pbro. Julio Jairo Ceballos Sepúlveda
Rector Seccional Bucaramanga: Presbítero Gustavo Méndez Paredes
Vicerrectora Académica Seccional Bucaramanga: Ana Fernanda Uribe Rodríguez
Decano de la Escuela de Ingenierías: Edwin Dugarte Peña
Director de la Facultad de Ingeniería de Sistemas e Informática: Diana Teresa Gómez Forero
Gestor Editorial Seccional Bucaramanga: Ginette Rocío Moreno Cañas
Editor: Juan Carlos Rodas Montoya
Coordinación de Producción: Ana Milena Gómez Correa
Corrección de Estilo: Romy Alexandra Lopez Muñetón
Diagramación e ilustración portada: Sissi Tamayo Chavarriaga

Dirección Editorial:
Editorial Universidad Pontificia Bolivariana, 2018
E-mail: editorial@upb.edu.co
www.upb.edu.co
Telefax: (57)(4) 354 4565
A.A. 56006 - Medellín - Colombia

Radicado: 1624-17-08-17

Prohibida la reproducción total o parcial, en cualquier medio o para cualquier propósito sin la autorización escrita de la Editorial Universidad Pontificia Bolivariana.

Contenido

Prefacio 19

Capítulo 1
El proceso de la programación 25
1.1 El proceso de la programación 26
1.1.1 Definición del problema 27
1.1.2 Análisis del problema 27
1.1.3 Diseño del programa 29
1.1.4 Codificación del algoritmo 32
1.1.5 Implementación del programa 36
1.1.6 Mantenimiento del programa 36

Capítulo 2
El enfoque algorítmico 39
2.1 Los algoritmos y su representación 40
2.1.1 Robi-Q, lenguaje para dibujar en cuadrículas 41
2.1.2 Elementos de un diagrama de flujo 46
2.2 Expresiones y operadores 47
2.2.1 Operadores aritméticos 48
2.2.2 Operadores relacionales 48
2.2.3 Operadores lógicos 48
2.2.4 Jerarquía de los operadores 50

Capítulo 3
Principios de la codificación 57
3.1 Tipos de datos 58
3.2 Variables 59
3.3 Lenguaje de programación 60
3.4 Lectura y escritura de datos 62
3.4.1 C++ 62

3.4.2 Java ... 66
3.4.3 Python ... 71

Capítulo 4
Situaciones elementales ... 75

Capítulo 5
Situaciones con estructuras de control ... 91
5.1 Estructura de control condicional If-Else ... 92
5.1.1 Combinación con operadores lógicos ... 101
5.1.2 Unión de estructuras de control ... 104
5.1.2.1 Estructura condicional anidada ... 104
5.1.2.2 Estructura condicional secuencial ... 108
5.1.3 Estructura switch case ... 116
5.2 Estructuras de programación repetitivas ... 128
5.2.1 Ciclo for ... 128
5.2.2 Estructura while ... 138
5.2.3 Estructura do-while ... 148
5.3 Arreglos ... 155
5.3.1 Tipos de arreglos por dimensiones ... 156
5.3.2 Partes de un arreglo ... 156
5.3.3 Tipos de arreglos por declaración ... 158
5.4 Funciones ... 174

Capítulo 6
Técnicas de validación y comprobación ... 187
6.1 Pruebas de escritorio a pseudocódigo y diagramas de flujo ... 188
6.2 Pruebas de seguimiento al código desde el IDE ... 191
6.2.1 C++ empleando Dev C++ ... 191
6.2.2 Java empleando NetBeans ... 192
6.2.3 Python empleando Python IDLE ... 194
6.3 Conformación de conjuntos de datos para pruebas ... 195
6.4 Manejo de excepciones ... 197
6.4.1 C++ ... 198
6.4.2 Java ... 200
6.4.3 Python ... 203

Capítulo 7
Paradigmas de programación ... 207
7.1 Programación estructurada ... 208
7.2 Programación funcional ... 209
7.3 Programación orientada a objetos ... 210
7.4 Programación orientada a eventos ... 214
7.5 Otros paradigmas de programación ... 217
7.5.1 Programación orientada a aspectos ... 217
7.5.2 Programación por procedimientos ... 218

Apéndice
Sistemas de numeración ... 219
- Sistema decimal ... 220
- Sistema binario ... 221
- Sistema hexadecimal ... 223
- Conversión entre sistemas de numeración ... 225
Sistema decimal a binario ... 225
Sistema binario a decimal ... 228
Sistema hexadecimal a binario ... 229
Sistema binario a hexadecimal ... 230
Sistema decimal a hexadecimal ... 231
Sistema hexadecimal a decimal ... 233

Referencias ... 235

Lista de figuras

Figura 1. Fases del proceso de programación 27
Figura 2 Dimensiones de una piscina 28
Figura 3. Pseudocódigo (a) y diagrama de flujo (b) para el algoritmo de las dimensiones de la piscina y el volumen de agua contenida 31
Figura 4. Lenguaje Q para la programación de Robi-Q 41
Figura 5. Programación de Robi-Q para el conjunto de instrucciones establecido 42
Figura 6. Trayectoria deseada 43
Figura 7. Expresiones 47
Figura 8. Condición con operador lógico && 49
Figura 9. Condición con operador lógico || 50
Figura 10. Tipos de datos 59
Figura 11. Diagrama de flujo para el ejemplo operaciones_básicas 77
Figura 12. Diagrama de flujo para el ejemplo nómina_básica 81
Figura 13. Diagrama de flujo para el ejemplo área_triángulo 84
Figura 14. Estructura condicional IF 93
Figura 15. Diagrama de flujo para el ejemplo par_impar 94
Figura 16. Diagrama de flujo para el ejemplo Impuestos 97
Figura 17. Diagrama de flujo para el ejemplo múltiplo_número 102
Figura 18. Unión de estructuras de control 104
Figura 19. Diagrama de flujo para el ejemplo notas_condicional_anidado 106
Figura 20. Estructura condicional secuencial 109
Figura 21. Diagrama de flujo para el ejemplo notas_condicionales_secuenciales 110
Figura 22. Distribución del presupuesto del hospital 115
Figura 23. Diagrama de flujo para el ejemplo cantidad_días_mes 119
Figura 24. Diagrama de flujo para el ejemplo Operaciones 122
Figura 25. Representación del ciclo for 129
Figura 26. Diagrama de flujo para el ejemplo Edades 131
Figura 27. Diagrama de flujo para el ejemplo serie_números_for 134
Figura 28. Ciclo while 139
Figura 29. Diagrama de flujo para el ejemplo Suma 141
Figura 30. Diagrama de flujo para el ejemplo serie_números_while 143
Figura 31. Estructura do-while 148
Figura 32. Diagrama de flujo para el ejemplo serie_números_do_while 150
Figura 33. Representación de un arreglo unidimensional o vector 157
Figura 34. Representación de un arreglo bidimensional o matriz ..158
Figura 35. Diagrama de flujo para el ejemplo separar_cadena 160
Figura 36. Matriz de calificaciones a) inicial y b) después de la sumatoria 161
Figura 37. Diagrama de flujo para el ejemplo notas_arreglos 162
Figura 38. Diagrama de flujo para el ejemplo ordenamiento_burbuja 165
Figura 39. Conversión de decimal a binario 168
Figura 40. Diagrama de flujo para el ejemplo decimal_a_binario 169
Figura 41. Invocación de una función en un programa 175
Figura 42. Diagrama de flujo para el ejemplo suma_funciones 176
Figura 43. Diagrama de flujo para el ejemplo operaciones_funciones 179
Figura 44. Diagrama de flujo y pseudocódigo para el ejemplo aprobación_estudiantes 189
Figura 45. Seguimiento de código en C++ empleando Dev-C++ 191
Figura 46. Punto de interrupción en Dev C++ 192
Figura 47. Prueba de escritorio de un programa en NetBeans 193
Figura 48. Ventana de depuración en NetBeans 193
Figura 49. Puntos de interrupción en Python 194
Figura 50. Programa para el manejo de excepciones en C++ 199
Figura 51. Excepción en C++ 200
Figura 52. Manejo de excepciones en Java 201

Figura 53. Excepción en Java 201
Figura 54. Excepción generalizada en Java....202
Figura 55. Mensaje de excepción generalizada en Java....202
Figura 56. Estructura de una excepción en Python....204
Figura 57. Excepción implementada en Python 204
Figura 58. Código en C++ empleando programación estructurada....209
Figura 59. Diagrama de clases....211
Figura 60. Definición de una clase en Java 212
Figura 61. Definición de getters y setters 213
Figura 62. Definición de un método....214
Figura 63. Creación de una interfaz gráfica en NetBeans....215
Figura 64. Edición de interfaz gráfica en NetBeans....215
Figura 65. Definición de la clase principal en NetBeans y ejecución de la ventana creada 216

Lista de tablas

Tabla 1. Conjunto de datos de prueba para el algoritmo de las dimensiones de la piscina y el volumen de agua contenida....31
Tabla 2. Codificación en C++, Java y Python del ejemplo de la piscina.... 33
Tabla 3. Símbolos para diagramas de flujo 46
Tabla 4. Operadores aritméticos 48
Tabla 5. Operadores relacionales.... 49
Tabla 6. Tabla de verdad para los operadores lógicos a) AND b) OR y c) NOT 50
Tabla 7. Jerarquía de operadores 51
Tabla 8. Ejemplo de representación de variables 60
Tabla 9. Tipos de datos 61
Tabla 10. Comandos iniciales en un lenguaje de programación 61
Tabla 11. Entradas y salidas de operaciones_básicas....77
Tabla 12. Solución al ejemplo operaciones_básicas empleando C++, Java y Python.... 78
Tabla 13. Entradas y salidas del programa 81
Tabla 14. Algoritmo en C++, Java y Python para el ejemplo nómina_básica empleando C++, Java y Python.... 82
Tabla 15. Entradas y salidas del programa área_triángulo.... 85
Tabla 16. Algoritmo en C++, Java y Python para el ejemplo área_triángulo 85
Tabla 17. Entradas y salidas para el ejemplo par_impar 94
Tabla 18. Codificación del ejemplo par_impar empleando C++, Java y Python.... 95
Tabla 19. Entradas y salidas para el ejemplo Impuestos.... 96
Tabla 20. Codificación empleando C++, Java y Python para el ejemplo Impuestos 97
Tabla 21. Codificación del ejemplo múltiplo_número empleando C++, Java y Python....101

Tabla 22. Codificación en C++, Java y Python para el ejemplo múltiplo_número....103
Tabla 23. Entradas y salidas del ejemplo notas_condicional_anidado....105
Tabla 24. Codificación en C++, Java y Python para el ejemplo notas_condicional_anidado....106
Tabla 25. Código en C++, Java y Python para el ejemplo notas_condicionales_secuenciales....111
Tabla 26. Entradas y salidas para el ejemplo cantidad_días_mes....119
Tabla 27. Código en C++ y Java para el ejemplo cantidad_días_mes....120
Tabla 28. Entradas y salidas para el ejemplo Operaciones....121
Tabla 29. Código en C++ y Java para el ejemplo Operaciones....122
Tabla 30. Requisitos de admisión a una carrera....125
Tabla 31. Entradas y salidas para el ejemplo Edades....130
Tabla 32. Código en C++, Java y Python para el ejemplo Edades....131
Tabla 33. Entradas y salidas para el ejemplo serie_números_for....133
Tabla 34. Código en C++, Java y Python para el ejemplo serie_números_for....134
Tabla 35. Entradas y salidas para el ejemplo Suma....140
Tabla 36. Código en C++, Java y Python para el ejemplo Suma....141
Tabla 37. Código en C++, Java y Python para el ejemplo serie_números_while....143
Tabla 38. Entradas y salidas para el ejemplo serie_números_do_while....149
Tabla 39. Código en C++ y Java para el ejemplo serie_números_do_while....150
Tabla 40. Entradas y salidas del algoritmo para el ejemplo separar_cadena....159
Tabla 41. Código en C++, Java y Python para el ejemplo separar_cadena....160
Tabla 42. Código en C++, Java y Python para el ejemplo notas_arreglos....162
Tabla 43. Código en C++, Java y Python para el ejemplo ordenamiento_burbuja....166
Tabla 44. Entradas y salidas para el ejemplo decimal_a_binario....169
Tabla 45. Código en C++, Java y Python para el ejemplo decimal_a_binario....170
Tabla 46. Entradas y salidas para el ejercicio generar_matriz....172
Tabla 47. Entradas y salidas para el ejemplo suma_funciones....176
Tabla 48. Código en C++, Java y Python para el ejemplo suma_funciones....177
Tabla 49. Entradas y salidas para el ejemplo operaciones_funciones....178
Tabla 50. Código en C++, Java y Python para el ejemplo operaciones_funciones....179
Tabla 51. Tabla general para pruebas de escritorio....188
Tabla 52. Prueba de escritorio para el ejemplo aprobación_estudiantes....190
Tabla 53. Entradas y salidas para el ejemplo operaciones_en_matemáticas....195

Lista de ejemplos

Ejemplo 1. Las dimensiones de la piscina y el volumen de agua contenida 28
Ejemplo 2. Robi-Q sigue los primeros comandos 42
Ejemplo 3. Operadores aritméticos 51
Ejemplo 4. Operadores relacionales 52
Ejemplo 5. Operadores lógicos 53
Ejemplo 6. Operaciones_básicas 76
Ejemplo 7. Nómina_básica 79
Ejemplo 8. Área_triángulo 83
Ejemplo 9. Par_impar 93
Ejemplo 10. Impuestos 96
Ejemplo 11. Múltiplo_número 101
Ejemplo 12. Notas_condicional_anidado 105
Ejemplo 13. Notas_condicionales_secuenciales 109
Ejemplo 14. Cantidad_días_mes 118
Ejemplo 15. Operaciones 121
Ejemplo 16. Edades 129
Ejemplo 17. Serie_números_for 133
Ejemplo 18. Suma 139
Ejemplo 19. Serie_números_while 142
Ejemplo 20. Serie_números_do_while 149
Ejemplo 21. Separar_cadena 158
Ejemplo 22. Notas_arreglos 161
Ejemplo 23. Ordenamiento_burbuja 164
Ejemplo 24. Decimal_a_binario 167
Ejemplo 25. Sumatoria_funciones 175
Ejemplo 26. Operaciones_funciones 178
Ejemplo 27. Aprobación_estudiantes 189

Lista de ejercicios

Ejercicio 1. Un nuevo diseño para Robi-Q 43
Ejercicio 2. Evaluación de expresiones 54
Ejercicio 3. Boletas de la feria 86
Ejercicio 4. Área del rectángulo 87
Ejercicio 5. Ley de Ohm 87
Ejercicio 6. Cajero automático 87
Ejercicio 7. Número mayor 98
Ejercicio 8. Persona mayor 99
Ejercicio 9. Mayor_tres_números 112
Ejercicio 10. Mayor_igual_dos_números 113
Ejercicio 11. Mayor_área_figuras_geométricas 113
Ejercicio 12. Edad_anios_meses_días 113
Ejercicio 13. Admisiones 124
Ejercicio 14. Día_del_anio 125
Ejercicio 15. Conversiones_masa 125
Ejercicio 16. Elecciones 125
Ejercicio 17. Fibonacci_for 135
Ejercicio 18. Impares_dsc 135
Ejercicio 19. Múltiplos 136
Ejercicio 20. Suma_armónica 136
Ejercicio 21. Divisores 136
Ejercicio 22. Fibonacci_while 144
Ejercicio 23. Factorial_while 145
Ejercicio 24. Números_pares 145
Ejercicio 25. División_entre_dos 145
Ejercicio 26. Repetir_hasta_múltiplo_4 146
Ejercicio 27. Promedio_notas 146
Ejercicio 28. Fibonacci_do_while 151
Ejercicio 29. Voltaje_repetitivo_do_while 152
Ejercicio 30. Voltaje_repetitivo_promedio_do_while 152
Ejercicio 31. Factorial_do_while 152

Ejercicio 32. Rebotes_pelota_do_while 153
Ejercicio 33. Total_compra_do_while 153
Ejercicio 34. Separar_pares_impares 171
Ejercicio 35. Generar_matriz 171
Ejercicio 36. Cadena 172
Ejercicio 37. Separar_múltiplos_5_3_resto 172
Ejercicio 38. Calculadora_funciones 183
Ejercicio 39. Carga_arreglo_funciones 183
Ejercicio 40. Suma_mayor_funciones 183
Ejercicio 41. Edad_persona_funciones 183

Lista de actividades

Actividad 1. Algoritmos en Robi-Q 43
Actividad 2. Operaciones y operadores 54
Actividad 3. Representación de los algoritmos 88
Actividad 4. Estructuras condicionales simples 99
Actividad 5. Condicionales anidados y secuenciales 114
Actividad 6. Switch case 126
Actividad 7. Ciclo for 136
Actividad 8. Ciclo while 146
Actividad 9. Ciclo do-while 153
Actividad 10. Arreglos computacionales 173
Actividad 11. Funciones 184

Prefacio

Propósito de este libro

Fundamentos de Programación es un libro pensado para estudiantes que quieran ingresar a la programación de computadores o fortalecer sus conocimientos acerca del tema. Los contenidos presentados en este libro surgen de temáticas y ejercicios desarrollados y validados en los cursos de Lógica de Programación, Lógica y Algoritmia, Ingeniería de Software, entre otras asignaturas de programación impartidas desde la Facultad de Ingeniería de Sistemas e Informática de la Universidad Pontificia Bolivariana, Seccional Bucaramanga.

Como objetivos fundamentales, este texto busca brindar a los estudiantes las herramientas suficientes para interpretar situaciones problema del mundo real o formal, diseñar sus soluciones computacionales a partir de algoritmos, implementar tales soluciones sobre lenguajes de programación como C++, Java y Python, y validar los resultados a partir de conjuntos de prueba.

Enfoque

Este texto está enfocado en la enseñanza de los conceptos básicos de programación estructurada para la solución de problemas del mundo real. Cada uno de los temas presentados se encuentra acompañado de una serie de ejemplos que, a su vez, cuentan con su análisis de entradas y salidas, con la representación del algoritmo empleando diagramas de flujo y con la codificación del algoritmo en los lenguajes C++, Java y Python.

Características de aprendizaje

Teniendo en cuenta que el proceso de aprendizaje de la programación requiere de una práctica constante por parte del estudiante, en este libro se plantean ejercicios de refuerzo de cada tema, así como actividades con ejercicios de mayor complejidad para el desarrollo de las habilidades de síntesis, análisis y codificación requeridas para dar una solución computacional.

Resumen de los capítulos

Los capítulos del texto se encuentran organizados comenzando por los fundamentos matemáticos de la programación, luego introducen las estructuras lógicas de control, arreglos y pruebas de los algoritmos empleando entornos de desarrollo.

Capítulo 1: El proceso de la programación

En este capítulo se presentan las fases del proceso de la programación empleadas para el desarrollo de un programa que corresponden a la definición del problema, análisis, diseño, codificación, implementación y mantenimiento. Estas etapas son ilustradas a través de un caso de estudio.

Capítulo 2: El enfoque algorítmico

Este capítulo presenta la definición formal de algoritmo, sus características, así como sus formas de representación empleando diagramas de flujo y seudocódigos. Además, se abordan los tipos de expresiones y operadores empleados para el desarrollo de un programa.

Capítulo 3: Codificación del algoritmo

En esta sección se presentan los tipos de datos utilizados para representar la información en un computador, el concepto de variable y de lenguaje de programación. De la misma forma, se relacionan las secuencias de código básicas para la lectura y escritura de datos en C++, Java y Python.

Capítulo 4: Situaciones elementales

Esta parte del texto ofrece un conjunto de ejemplos en los cuales se aplica el proceso de la programación para dar solución a un problema específico. Cada ejemplo cuenta con la descripción detallada de los pasos del proceso de la programación, así como con la codificación en C++, Java y Python.

Capítulo 5: Estructuras de control

Aquí se aborda el manejo de las estructuras de control presentes en el paradigma de la programación estructurada. Inicialmente se tratan las estructuras de selección para la formulación de preguntas y como elemento de control del flujo de un algoritmo. Luego se presentan las estructuras de control repetitivas que, como su nombre lo indica, permiten repetir un conjunto de instrucciones de acuerdo con un conjunto de condiciones establecidas. Posteriormente, se introduce al manejo de arreglos computacionales, los cuales facilitan el manejo de información dentro de un programa. Finalmente se presenta el concepto de función o subrutina que es útil para el desarrollo de algoritmos complejos con tareas altamente repetitivas.

Capítulo 6: Técnicas de validación y comprobación

El capítulo 6 presenta una introducción al manejo de los entornos de programación Dev C++, NetBeans y Python IDLE empleados para

la codificación en C++, Java y Python de los algoritmos presentados en este libro. Además, se establece la metodología para la realización de pruebas de escritorio de los algoritmos en cada uno de los entornos de desarrollo presentados. También se especifica el manejo de excepciones en C++, Java y Python.

Capítulo 7: Paradigmas de programación

Finalmente, se ofrece al lector una revisión general de los paradigmas de programación existentes como herramientas para abordar la solución de un problema computacional. Inicialmente se establecen los conceptos de programación estructurada, programación orientada a objetos, programación de eventos la cual se aplica para el desarrollo de interfaces gráficas entre otros paradigmas utilizados en desarrollos de software comercial.

Pedagogía

El presente texto brinda al estudiante y al docente una nueva forma de abordar el desarrollo de la programación guiada hacia el desarrollo de las competencias del estudiante, además de servir como herramienta de apoyo para el diseño de los cursos de programación de los docentes.

Los estudiantes podrán encontrar, inicialmente, un desarrollo conceptual de los temas apoyado por un conjunto de ejemplos. Como se puede observar a partir del capítulo 2, los ejemplos tienen su respectiva tabla de análisis de entradas y salidas de cada uno de los algoritmos, así como el diagrama de flujo del algoritmo representado con los símbolos ANSI. También, se presenta el código en C++, Java y Python para cada ejemplo, donde el código se muestra en los mismos colores usados en los entornos de desarrollos de cada lenguaje. Adicionalmente, al final de cada uno de los temas encontrará un conjunto de ejercicios de refuerzo. Es importante mencionar que el texto contenido en los diagramas de flujo, así como en los segmentos de código carece de tildes. Por ejemplo la palabra *número* podrá encontrarla como *numero*, esto obedece a que lenguajes como Python, Java, C++ provienen del idioma inglés, y sus compiladores no reconocen caracteres especiales como las vocales con tilde. Sin embargo, expresiones que se encuentren dentro de pares de comillas sí podrán llevar tildes pues estas no tienen el propósito de ser procesadas, sino que se muestran en la forma como fueron digitadas originalmente.

Para los docentes, el libro ofrece un conjunto de actividades al final de cada tema, las cuales muestran los objetivos de formación que se desean alcanzar en cada una. Estas actividades tienen un nivel de dificultad más alto que el de los ejercicios de refuerzo ya que tienen un objetivo de formación, lo que las hace ideales para el desarrollo de laboratorios o ejercicios de clase. Además, el docente contará con un portal web en el cual podrá encontrar una ampliación de los recursos presentados en este libro.

Los ejercicios y actividades se basan en la recopilación de la experiencia obtenida durante el desarrollo de los cursos de programación ofrecidos por la Facultad de Ingeniería de Sistemas e Informática.

Recursos para el docente

El docente puede acceder a un conjunto de recursos adicionales para el desarrollo de sus clases en el sitio web (http://sara.bucaramanga.upb.edu.co) el cual incluye los siguientes materiales:

- Solución de los ejercicios en las actividades propuestas (únicamente para docentes)
- Solución a los ejercicios de refuerzo de cada tema (docentes y estudiantes)
- Diapositivas de cada capítulo para apoyar la labor docente
- Un banco de ejercicios para el desarrollo de exámenes y pruebas para la asignatura (únicamente para docentes)

Software empleado

Para la construcción de los ejemplos presentados en el documento, así como para la implementación de los ejercicios y actividades, los autores han utilizado los siguientes entornos de programación de acuerdo con el lenguaje de programación:

Para desarrollar los programas en lenguaje C++ se utiliza el entorno de desarrollo integrado (IDE) Bloodshed Dev C++ versión 5.5.3. En el caso del lenguaje de programación Python, los programas han sido escritos en Python 3 empleando el IDE Python IDLE en su versión 3.3.1 en conjunto con las librerías Numpy para el manejo de arreglos.

Para construir programas en Java, se emplea el IDE Netbeans con el Java JDK 7u80. Todos los IDE mencionados anteriormente son soportados bajo el sistema operativo Windows en sus versiones XP, 7, 8, 8.1 y 10 y cuentan con licencia de código abierto para su uso sin restricciones en cualquier computador.

Los autores recomiendan a los lectores interesados en este documento emplear los entornos de desarrollo, librerías y APK relacionados para que de esta forma no tengan mayores inconvenientes al implementar los algoritmos presentados en el libro.

Contacto con los autores

Los autores han revisado una y otra vez los detalles técnicos de este libro. A pesar de esto, algunos aspectos pudieron ser omitidos. Si el lector descubre algún error técnico en el documento, puede contactar a los autores por medio de los correos electrónicos urbano.gomez@upb.edu.co, diana.gomez@upb.edu.co, jairo.viola@upb.edu.co. Las correcciones del libro se realizarán en la versión electrónica disponible en la página web http://sara.bucaramanga.upb.edu.co.

Capítulo 1
El proceso de la programación

Por un momento olvide los computadores y piense en tareas que usted ha desarrollado más de una vez, por ejemplo, la rutina que realiza diariamente desde el momento en que se despierta hasta el momento en que sale de su vivienda. Allí puede encontrar un algoritmo. Con algunas modificaciones el algoritmo de la situación planteada es el siguiente:

1. Apagar el despertador
2. Aceptar que es momento de levantarse
3. Realizar las actividades de higiene personal
4. Tomar el desayuno

Describa otras rutinas cotidianas:

1. Realizar una llamada desde su celular
2. Amarrarse los zapatos
3. Realizar la compra de cualquier producto en el supermercado

Al hacerlo, posiblemente identificó una secuencia de acciones, con un orden de principio a fin. También se cuestionó qué tan precisas fueron sus descripciones. Estas consideraciones también aplican cuando se piensa en algoritmos, y más específicamente cuando queremos ver representadas estas situaciones en entornos computacionales. Para ello debemos introducirnos en el proceso de la programación. Las siguientes secciones mostrarán las diferentes etapas que componen este proceso, conceptos que deben aplicarse dentro de cada etapa y recomendaciones para fluir cómodamente dentro de la programación.

1.1 El proceso de la programación

La programación de computadores es un proceso que se origina en la concepción de una idea y que continúa con su adecuada representación en estructuras lógicas que deben ser transformadas a comandos procesables por un lenguaje de programación, su ejecución computacional, la validación de los resultados y su puesta en

funcionamiento. Para recordar más fácilmente este proceso, será resumido a continuación y detallado a lo largo del capítulo.

Figura 1. Fases del proceso de programación

1.1.1 Definición del problema

Esta etapa consiste en poder describir e interpretar una determinada situación de la vida real, de algún sistema físico, comercial, matemático, etc. Pueden utilizarse para ello el lenguaje natural, las representaciones gráficas, las fórmulas, y cualquier otro elemento que aporte a la descripción del problema. El programador debe refinar sus habilidades de lectura, extraer los elementos esenciales y, eventualmente, reducir el problema de tal manera que tome solo aquellos elementos que puedan ser llevados a una solución computacional que responda al propósito de la situación planteada.

1.1.2 Análisis del problema

Esta fase requiere una clara definición que contemple exactamente lo que se espera que una solución computacional logre para satisfacer el objetivo expuesto en el problema. Para ello lo recomendable es descomponer la situación en tres aspectos:

- ¿Cuál es el resultado esperado? Es decir, ¿qué clase de respuesta debería obtener?, ¿cuántos valores conforman la respuesta esperada?
- ¿Qué entradas se requieren? Es decir, ¿qué clase de información o datos son requeridos para poder llegar a la respuesta esperada?

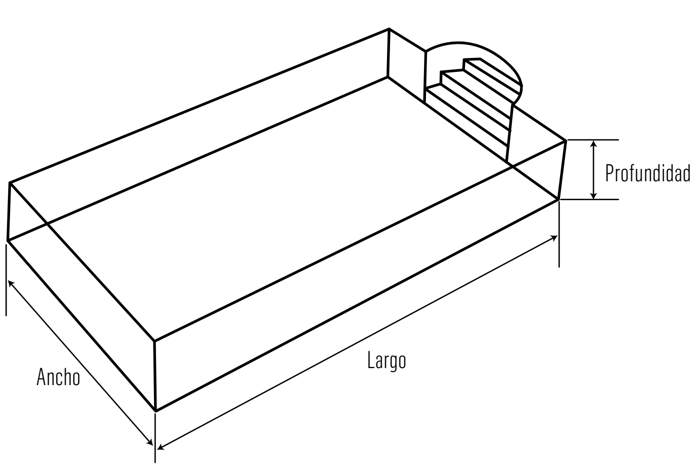
- ¿Qué proceso, método u operación puede conducir a la salida deseada a partir de los valores dados como entrada?

Para apreciar mejor las etapas de análisis y diseño del programa se propone el siguiente ejemplo.

Ejemplo 1.
Las dimensiones de la piscina y el volumen de agua contenida

Una piscina tiene las siguientes dimensiones: largo 25 metros, ancho 10 metros, y de profundidad tiene 1.5 metros, tal como lo muestra la figura 2. Se desea determinar el volumen de agua que se requiere para llenarla.

Figura 2. Dimensiones de una piscina



Definición del problema:

A partir de la figura 2, se puede observar que la piscina tiene una forma similar a un prisma rectangular del cual se debe calcular su volumen.

Análisis del problema:

El problema se descompone en las siguientes partes:

¿Cuál es el resultado esperado?

El enunciado solicita el volumen de agua para llenar la piscina. Por lo tanto, se debe calcular el volumen del prisma rectangular que representa a la piscina.

¿Qué entradas se requieren?

Se requiere conocer las dimensiones de esta piscina: largo, ancho y profundidad, esas dimensiones están dadas en el enunciado.

¿Qué proceso, método u operación puede conducir a la salida deseada a partir de los valores dados como entrada?

A partir de (1) se puede obtener el volumen de la piscina:

$$volumen=largo*ancho*profundidad \tag{1}$$

1.1.3 Diseño del programa

Una vez se tiene claridad sobre cuál es el resultado esperado, las entradas requeridas, y las operaciones que conducirán a la solución, el reto es ¿cómo organizar esta información para llevarla al computador? Para ello se organiza la secuencia de pasos que puede llevar a la solución, esta secuencia debe diseñarse estableciendo un principio, un fin y un orden sin ambigüedades. A este diseño se le conoce con la expresión “algoritmo”.

Los algoritmos pueden ser expresados con palabras (pseudocódigo) o de manera gráfica, con símbolos especiales interconectados según la secuencia correspondiente, los cuales son conocidos como diagramas de flujo.

Los algoritmos en cualquiera de sus dos formas son un mecanismo para que los programadores organicen sus ideas y las comuniquen. Cumplen la misma función que cumpliría un plano arquitectónico para comprender el diseño de un edificio, pero valga aclarar que aún no tienen efecto sobre el computador.

Continuando con el ejemplo de la piscina, el pseudocódigo y el diagrama de flujo para este algoritmo se muestran en la figura 3 (página siguiente). Como se observa, en ellos hay total equivalencia y con solo cinco pasos se resuelve el problema.

En ambos casos se marca un inicio, luego se realiza la captura o lectura de los valores de entrada requeridos (largo, ancho, profundidad). Conocidos estos valores, se realiza la operación para calcular el volumen a partir de (1) y una vez que el volumen es calculado, se muestra o imprime su valor, pues de esta forma el usuario visualizará el resultado esperado. Cumplido el objetivo de la situación propuesta se marca el final del algoritmo. Secciones posteriores se encargarán de ampliar detalles al proceso algorítmico.

Si bien el algoritmo ya fue definido, el diseño de la solución contiene otro paso que es “desarrollar la prueba de escritorio”. Esto significa que debe establecerse un conjunto de valores que se probarán sobre el algoritmo para confirmar si tal algoritmo conduce a resultados correctos o no. Para el caso de la piscina el primer conjunto de valores a probar son los valores dados en el enunciado del problema y que corresponden a: largo= 25 m, ancho = 10 m, profundidad = 1.5 m.

En adelante los conjuntos de datos de prueba serán ubicados en tablas, como se observa en la tabla 1 (página siguiente). En la parte izquierda de la tabla se ubican las entradas y sobre el lado derecho las salidas. Cada columna corresponde a un valor de entrada o de salida particular.

Figura 3. Pseudocódigo (a) y diagrama de flujo (b) para el algoritmo de las dimensiones de la piscina y el volumen de agua contenida

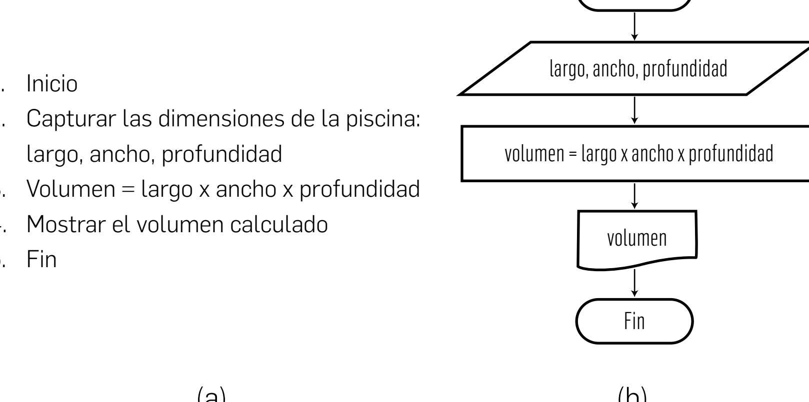


Tabla 1. Conjunto de datos de prueba para el algoritmo de las dimensiones de la piscina y el volumen de agua contenida

Entradas			Salidas
Largo	Ancho	Profundidad	Volumen
25	10	1.5	375

A partir de los valores presentados en la tabla 1, se toma el algoritmo en cualquiera de sus dos formas, realizando manualmente un seguimiento sobre papel, paso a paso, con los valores propuestos en el conjunto de prueba (esto explica el nombre de “prueba de escritorio”):

1. Dar inicio al algoritmo.
2. Capturar las dimensiones de la piscina. Este paso proyecta que en la solución computacional al usuario le aparecerá algún mensaje que solicite el largo de la piscina, a lo que el usuario responderá digitando el valor 25. Luego le será solicitado el ancho, que

el usuario digitará como 10, e igual le será solicitada la profundidad y el usuario digitará 1.5.

3. El cálculo del *volumen = largo x ancho x profundidad*, con los valores ingresados, arrojará 375.
4. Mostrar el volumen calculado, significa la visualización -supuestamente en la pantalla de un computador- del valor obtenido. Para los valores ingresados, el valor que se mostrará será 375.
5. Luego se da por concluido el proceso.

Si al realizar este seguimiento paso a paso, se obtienen los resultados esperados para el conjunto de prueba, significa que el algoritmo es correcto. Es conveniente crear varios conjuntos de prueba con valores convencionales e incluso con valores extremos o atípicos, y probarlos sobre el algoritmo. Únicamente así se comprobará qué tan robusto resulta el algoritmo propuesto, y se mejorará hasta que alcance un nivel de respuesta satisfactorio. En la medida en que adquiera más conocimientos sobre algoritmia, comprenderá que este sencillo algoritmo en realidad debería contener más elementos para que resuelva situaciones más reales. También conocerá más adelante que las pruebas de escritorio pueden realizarse utilizando algunas herramientas propias de cada entorno de programación. Este tema será abordado con mayor profundidad en el capítulo 3.

1.1.4 Codificación del algoritmo

En la fase de codificación, se transforma el pseudocódigo o el diagrama de flujo a un código de algún lenguaje de programación como C++, Java o Python (los lenguajes elegidos en este libro para su aprendizaje). Para esto se emplea un entorno de desarrollo integrado (en adelante IDE, por sus siglas en inglés), conjunto de herramientas software que permiten la escritura, edición y ejecución de programas. Estos IDE varían dependiendo del lenguaje de programación empleado.

Continuando con el ejemplo de la piscina, en la tabla 2 se presenta la codificación del algoritmo para el cálculo del volumen de una piscina rectangular empleando los lenguajes de programación C++, Java y Python. Como se puede observar cada uno de los pasos establecidos en el diagrama de flujo corresponden a un conjunto de instrucciones especiales dentro de cada lenguaje de programación. A lo largo del libro encontrará más explicaciones que le permitirán comprender el significado de los comandos y la forma como deben escribirse (sintaxis) según cada lenguaje de programación.

Tabla 2. Codificación en C++, Java y Python del ejemplo de la piscina

Código en C++
1 int largo;
2 int ancho;
3 int profundidad;
4 cout<<”Ingrese la longitud de la piscina”;
5 cin>>largo;
6 cout<<”Ingrese el ancho de la piscina”;
7 cin>>ancho;
8 cout<<”Ingrese la profundidad de la piscina”;
9 cin>>profundidad;
10 volumen=largo*ancho*profundidad
11 cout<<”El volumen de la piscina es”<< volumen;

Continúa

Código en Java

```
1  Scanner teclado = new Scanner(System.in);
2  int largo;
3  int ancho;
4  int profundidad;
5  System.out.print("Ingrese la longitud de la piscina");
6  largo = teclado.nextInt();
7  System.out.print("Ingrese ancho de la piscina");
8  ancho = teclado.nextInt();
9  System.out.print("Ingrese la profundidad de la pis-
   cina");
10 largo = teclado.nextInt();
11 volumen=largo*ancho*profundidad
12 System.out.print ("el volumen de la piscina es");
13 System.out.print (volumen);
```

Código en Python

```
1  largo=int(input("Ingrese la longitud de la piscina"));
2  ancho=int(input("Ingrese el ancho de la piscina"));
3  profundidad=int(input("Ingrese la profundidad de la
   piscina"))
4  volumen=alto*ancho*profundidad;
5  print("El volumen de la piscina es", volumen)
```

Los anteriores segmentos de código fuente, resultan fácilmente asimilables a estructuras escritas en lenguaje inglés (por ello se les conoce como lenguaje de alto nivel). Sin embargo, están escritas para ser comprendidas por humanos (particularmente por el programador), pero aún no son comprensibles para el computador. Para ello debe realizarse una transformación llamada compilación (que se hace con una orden o un simple clic sobre el IDE), y traduce el anterior segmento en lenguaje de alto nivel a lenguaje de máquina. Este paso es fundamental, ya que las computadoras son máquinas que operan con base en el sistema de numeración binaria (lenguaje conformado exclusivamente por unos y ceros). Para una mejor comprensión de cómo transformar los caracteres propios del lenguaje de alto nivel al código binario, puede explorar el apéndice A.

Si el proceso de compilación resulta exitoso, el IDE podrá ejecutar el programa que ahora está en lenguaje de máquina sobre los componentes físicos que conforman el computador. Durante la ejecución, los resultados del programa se mostrarán en la pantalla o monitor, de esta forma se puede comprobar si el programa realiza la tarea solicitada.

Es importante resaltar que, usualmente, esta etapa de codificación, compilación y ejecución, puede ser cíclica, pues se regresará una y otra vez corrigiendo errores hasta que se logre la ejecución. En capítulos posteriores logrará identificar algunos de los errores más frecuente, y cómo evitarlos.

Es una buena práctica que la solución desarrollada hasta este punto cuente con una documentación interna (dentro del código fuente) y externa (en documentos que soporten las fases anteriores y que detallen las funcionalidades que la solución ofrece). Esta documentación será de gran ayuda para el equipo de analistas, desarrolladores y de mantenimiento de las soluciones alcanzadas. En la medida que incorpore más conocimiento, comprenderá que existe un horizonte muy amplio al respecto.

1.1.5 Implantación del programa

La implantación se da cuando el programa ha sido probado exhaustivamente y realiza correctamente las funcionalidades esperadas. Entonces se instala para ponerlo en funcionamiento y al servicio de sus correspondientes usuarios. Esta etapa puede variar en complejidad dependiendo del tipo de situación.

Para el caso del ejemplo de la piscina, la situación es tan sencilla, que podría pensarse que con la ejecución funcional ya está listo. Pero en la vida real, esta situación podría ser apenas un elemento dentro de un software más complejo o podría requerir de todo un plan administrativo para que entre en funcionamiento. Esta etapa está por fuera del alcance de este libro y, por tanto, no será abordada.

1.1.6 Mantenimiento del programa

El entorno digital cambia con el paso del tiempo, esto hace que las necesidades cambien y que un programa se vuelva obsoleto, por ello es necesario realizarle continuas optimizaciones.

Para el ejemplo de la piscina, el programa generado podría requerir mantenimiento cuando se describa una nueva situación no contemplada inicialmente, por ejemplo, que dentro de la piscina se incluirá una fuente que ocupa cierto espacio, y por lo tanto la fórmula para calcular el volumen del agua requiere alguna modificación, esto solo a manera de comentario pues en este libro no se desarrollará la etapa de mantenimiento a los programas desarrollados.

Nota 1. Recomendaciones antes de iniciar

1. Hay programadores que omiten la formalidad de las etapas de definición, análisis y diseño, y saltan a la fase de codificación por considerar que el “verdadero” producto es el programa, lo cual es una práctica errónea ya que, al igual que una construcción requiere de un análisis del terreno y de un diseño que permita reconocer las posibles mejoras o alcances, un programa tiene muchas posibilidades de solución y seguir todos los pasos facilitará hallar la mejor de ellas.

2. La programación, a diferencia de otras materias, como podría ser la historia, requiere un estudio metódico y ordenado (en historia se puede estudiar la Edad Media sin tener grandes conocimientos de la Edad Antigua). Los temas de este libro exigen seguirlos en secuencia. Es importante no dejar temas sin entender y relacionar.

3. Es bueno tener paciencia cuando los problemas no se resuelven por completo, y es de fundamental importancia dedicar tiempo al análisis individual de los problemas. Para llegar a ser programador se debe recorrer un largo camino en el que cada tema es fundamental para conceptos futuros.

Capítulo 2
El enfoque algorítmico

2.1 Los algoritmos y su representación

Según Cairó (2009), un algoritmo es un "conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema". Un algoritmo debe cumplir con las siguientes características:

- **Finito:** debe tener un número determinado de pasos y obtener un resultado en un tiempo limitado.
- **Definido:** el orden en que se realicen los pasos no puede presentar ambigüedades, de tal forma que para un conjunto idéntico de datos de entrada se debe obtener siempre el mismo resultado.
- **Preciso:** la secuencia de pasos debe conducir a un resultado correcto.

Existen dos formas de representación de los algoritmos: el pseudocódigo y los diagramas de flujo. El pseudocódigo es un lenguaje intermedio entre nuestro lenguaje y el lenguaje de programación, representa la solución de un algoritmo de la forma más detallada posible, y a su vez lo más parecida posible al lenguaje que posteriormente se utilizará para la codificación en el computador.

El diagrama de flujo es la representación gráfica de un algoritmo. Su correcta construcción facilita la codificación de un programa en cualquier lenguaje de programación. La gran ventaja de utilizar pseudocódigo o diagramas de flujo es que permiten representar las ideas para solucionar un problema o situación sin comprometerse con algún lenguaje de programación en particular. La misma solución en pseudocódigo o en diagrama de flujo se constituye en el punto de partida para desarrollar un programa, bien sea en lenguaje C++, Java, Python, o cualquier otro lenguaje de programación existente.

2.1.1 Robi-Q, lenguaje para dibujar en cuadrículas

Code.org (2014) es una organización sin fines de lucro dedicada a ampliar la participación en la informática y que tiene la visión de que cada estudiante en cada escuela debe tener la oportunidad de aprender programación de computadoras. Esta organización ha creado una serie de estrategias didácticas para facilitar la comprensión de la lógica de la programación. Una de ellas invita a dibujar figuras simples sobre cuadrículas y para ello debe indicarse la secuencia de acciones. Los autores han adaptado esta idea, y la han personificado en un pequeño robot al cual nombraron Robi-Q.

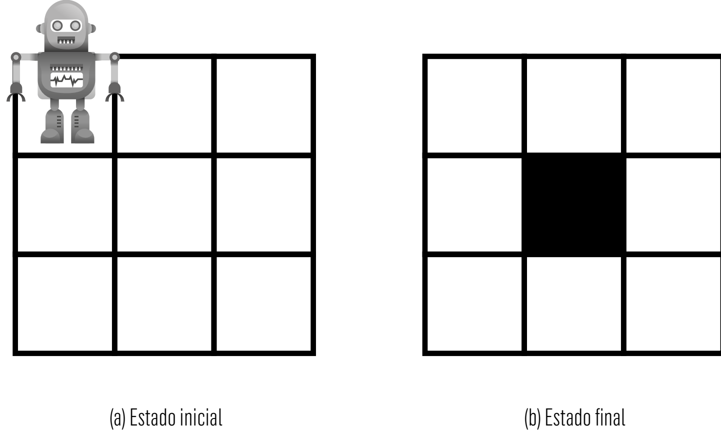
Este pequeño personaje conoce un limitado conjunto de comandos (lenguaje Q), que se muestra en la figura 4. Solo existen comandos para desplazarse a izquierda, a derecha, arriba, y abajo y un quinto símbolo para rellenar un cuadro. Robi-Q solo puede desplazarse por una cuadrícula inicialmente en blanco, y podrá rellenar los cuadros que le indiquen. De esta forma dibuja las figuras solicitadas.

Figura 4. Lenguaje Q para la programación de Robi-Q

Ejemplo 2.
Robi-Q sigue los primeros comandos

Originalmente Robi-Q se encuentra en la posición superior izquierda de la cuadrícula, como se muestra en la figura 5 (a), y le es dada la secuencia de órdenes: **v > O**

Figura 5. Programación de Robi-Q para el conjunto de instrucciones establecido



(a) Estado inicial (b) Estado final

El resultado de ejecutar la secuencia se aprecia en la figura 5 (b).

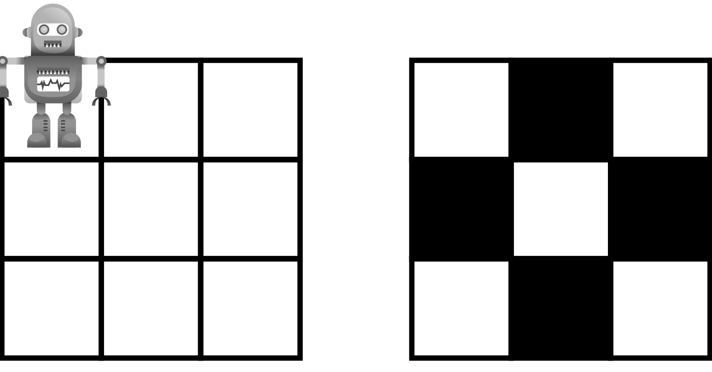
La simplicidad de este ejemplo intenta mostrar un algoritmo desde su mínima expresión. Se tiene un conjunto finito de acciones (solo tres comandos), que mantienen una secuencia, un principio, un fin y que llevaron a un resultado determinado. Todas las veces que esta secuencia se ejecute iniciando desde el mismo punto, se obtendrá el mismo resultado.

Ejercicio

Ejercicio 1. Un nuevo diseño para Robi-Q

Elabore el algoritmo para que Robi-Q transforme la superficie original (derecha) en la mostrada a la izquierda en la figura 6.

Figura 6. Trayectoria deseada



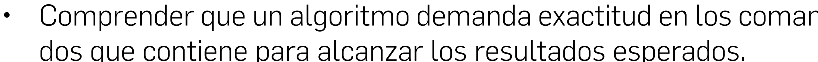
Actividad 1.
Algoritmos en Robi-Q

Objetivo general

Afianzar el concepto de algoritmo mediante la elaboración de figuras en cuadrículas siguiendo el microlenguaje Q.

Objetivos específicos

- Generar algoritmos en lenguaje Q a partir de una figura dada.
- Seguir comandos para construir soluciones.



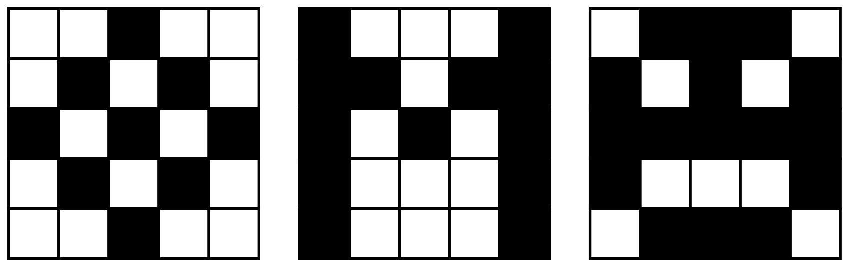
- Comprender que un algoritmo demanda exactitud en los comandos que contiene para alcanzar los resultados esperados.

Generalidades

Tomando como base el lenguaje Q y las figuras dadas, se le solicita al aprendiz escribir los algoritmos para lograr reproducir estos patrones. En otros ejemplos se le solicita interpretar un algoritmo dado en lenguaje Q, y seguirlo para reproducir la imagen que esta secuencia generará.

Ejercicios

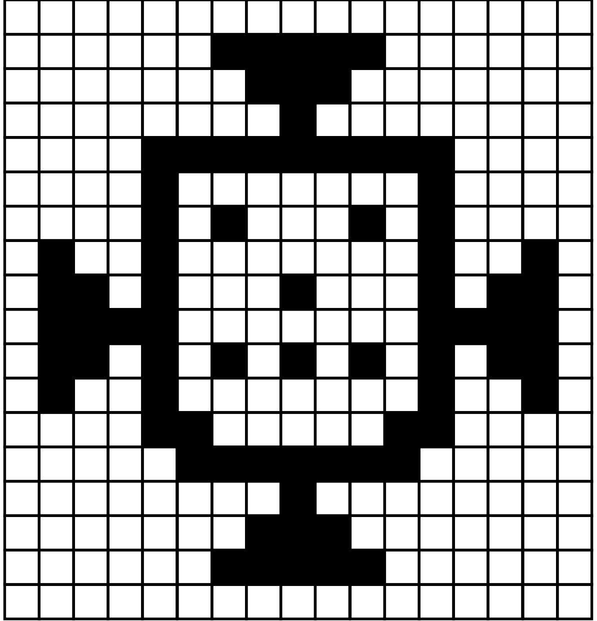
1. Escriba para cada una de las figuras siguientes, el algoritmo en lenguaje Q que permitirá obtener los diseños correspondientes. Este tipo de solicitudes se conoce como requerimiento, ya que especifica una necesidad que debe ser resuelta para un interesado.



2. Asuma que le ha sido dado el siguiente segmento de código en lenguaje Q. ¿Qué figura obtendrá?

```
v > > o > > o v v < o < < v o v > o v > o
```


3. Escriba el algoritmo en lenguaje Q que permitirá obtener la siguiente figura.



2.1.2 Elementos de un diagrama de flujo

Al igual que los mapas, que nos permiten elaborar imágenes mentales de la ubicación de lugares, los diagramas de flujo nos permiten representar y recordar la secuencia de pasos que nos conducirá a solucionar alguna situación específica. Y esta imagen mental prevalece por encima de los comandos dados en lenguajes particulares de programación, de ahí la importancia de diseñar algoritmos con esta metodología. En la tabla 3 se presentan los símbolos estándar utilizados para la elaboración de diagramas de flujo.

Tabla 3. Símbolos para diagramas de flujo

Nombre	Símbolo	Descripción
Terminal		Se usa al principio y al final de un programa, también para representar una interrupción.
Entrada		Representa la operación de lectura. Dentro de este símbolo se ponen las variables que están a la espera de recibir algún valor.
Salida		Representa la operación de escritura. Dentro de este símbolo se escriben los resultados o mensajes a visualizar.
Proceso		Aquí se ponen las expresiones que ejecutarán alguna acción o transformación de valores.
Decisión		Representa la toma de decisiones. Dentro se escriben expresiones condicionales cuya respuesta es verdadera o falsa, y dependiendo de este valor se continuará por diferente camino.
Conector		Conecta dos partes de un diagrama a través de un conector en la salida y otro conector en la entrada. Dentro de este símbolo se pone un número o letra que denota el segmento a continuar.
Indicador de dirección o Línea de flujo		Conecta los símbolos anteriores, indicando la secuencia en la que el algoritmo debe desarrollarse.

Es probable que la combinación de algunos símbolos, para efectos de representar algoritmos en un diagrama de flujo, obedezca a un estándar o patrón. A estas situaciones se le conoce como estructuras de control y serán descritas en el capítulo 5. Las diversas combinaciones en la forma de los símbolos podrán llevar a diferentes resultados.

2.2 Expresiones y operadores

Observe la Figura 7. Allí puede encontrar varias fórmulas, que pueden representar algunas operaciones matemáticas. En el mundo de la computación se usa más el término *expresión*.

Figura 7. Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis, tal y como se conocen en la notación tradicional de las matemáticas. Las expresiones son las que se encuentran al lado derecho del signo "=" (conocido como operador de asignación). Cada expresión es evaluada para calcular su respectivo valor, el cual es asignado a la variable que se encuentra al lado izquierdo de este símbolo. A continuación se detallará cada uno de los elementos que conforman las expresiones.

2.2.1 Operadores aritméticos

En la tabla 4 se presentan los operadores aritméticos que se emplean en la construcción de expresiones en programación. Ellos representan las operaciones básicas como la suma o el residuo de una división.

Tabla 4. Operadores aritméticos

Operador	Operación	Ejemplo	Resultado
+	Suma	125.78+62.50	188.28
-	Resta	65.30-32.33	32.97
*	Multiplicación	8.25*7	57.75
/	División	41744	3.75
\	División entera	17 \ 3	5
%	Módulo (residuo)	14% 2	0

2.2.2 Operadores relacionales

Los operadores relacionales solo son funcionales si se complementan con números, constantes o variables, el resultado del uso de los operadores relacionales es un valor «verdadero» o «falso». En la tabla 5 (página siguiente) se presentan los operadores relacionales que puede contener una expresión.

2.2.3 Operadores lógicos

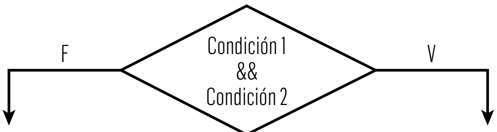
Los operadores lógicos son && y ||. Son empleados fundamentalmente en las estructuras condicionales (que se tratarán en el capítulo 5) para agrupar varias condiciones simples y convertirlas en compuestas. El operador && traducido se lee como "Y", es la conjunción en las proposiciones compuestas, es decir su valor es verdad

Tabla 5. Operadores relacionales

Operador	Operación	Ejemplo	Resultado
==	Igual que	'Hola' = 'Lola'	FALSO
!=	Diferente a	'a' != 'b'	VERDADERO
<	Menor que	7 < 5	FALSO
>	Mayor que	22 > 11	VERDADERO
<=	Menor o igual que	15 <= 22	VERDADERO
>=	Mayor o igual que	35 >= 20	VERDADERO

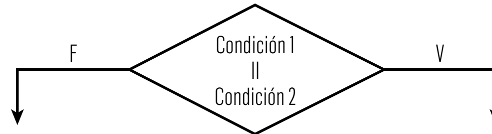
si la condición 1 y la condición 2 son verdaderas. Cuando se vinculan dos o más condiciones con el operador "&&", las dos condiciones deben ser verdaderas para que el resultado de la condición compuesta sea verdadero y continúe por la rama del verdadero de la estructura condicional. La utilización de operadores lógicos permite, en muchos casos, plantear algoritmos más cortos y comprensibles, su representación se muestra en la figura 8.

Figura 8. Condición con operador lógico &&



Por su parte, el operador || traducido se lo lee como "O". Si la condición 1 es verdadera o la condición 2 es verdadera, se ejecuta la rama del verdadero. Cuando se vinculan dos o más condiciones con el operador "O", basta con que una de las dos condiciones sea verdadera para que el resultado de la condición compuesta sea verdadero. La representación del operador || se muestra en la figura 9.

Figura 9. Condición con operador lógico ||



La unión de términos o variables con estos operadores entregarán algunos resultados que pueden ser resumidos en la tabla 6.

Tabla 6. Tabla de verdad para los operadores lógicos a) AND b) OR y c) NOT

DISYUNCIÓN: O, OR, ||, V		
OP 1	OP 2	RESULTADO
F	F	F
F	V	V
V	F	V
V	V	V

(a)

CONJUNCIÓN: Y, AND, &&, ^		
OP 1	OP 1	OP 1
F	F	F
F	V	F
V	F	F
V	V	V

(b)

NEGACIÓN: NO, NOT, !	
OP	RESULTADO
F	V
V	F

(c)

2.2.4 Jerarquía de los operadores

En la tabla 7 se presenta la jerarquía de los diferentes tipos de operadores de acuerdo con el orden de evaluación dentro de una expresión. Como se puede observar, las operaciones entre paréntesis presentan el máximo nivel de la jerarquía, mientras que las operaciones relacionales y lógicas se encuentran en la parte más baja de la jerarquía. Esto significa que las primeras operaciones a evaluar dentro de una expresión corresponden a los términos que se encuentren entre paréntesis, seguidas de los operadores aritméticos, relacionales y lógicos. En el caso particular de los operadores de

multiplicación, división, división parte entera y residuo, las operaciones se realizan recorriendo de izquierda a derecha la expresión en el orden que se encuentren dentro de la misma. A continuación, se presentan algunos ejemplos de evaluación de expresiones con diferentes tipos de operadores.

Tabla 7. Jerarquía de operadores

Operador	Símbolo	Jerarquía
Paréntesis	()	(Mayor)
Potenciación	**	
Multiplicación, división, división parte entera y residuo	*, /, \,	
Suma, resta	+, -	
Operadores relacionales	==,>,<,≥,≤,=!	
Operador lógico NOT	Not	
Operador lógico AND	And	
Operador lógico OR	Or	(Menor)

Ejemplo 3.
Operadores aritméticos

¿Cuál es el resultado de la expresión k=9+7*8-36/5?

Inicialmente se evalúan las operaciones de multiplicación y división. La multiplicación es la primera en ser evaluada ya que ocupa el primer lugar al recorrer la expresión de izquierda a derecha.

k=9+ 56-36 / 5

Luego se evalúa la división ya que es la segunda operación en aparecer al evaluar la expresión de izquierda a derecha

k=9+56-7.2

Finalmente, se realizan las operaciones de suma y resta dando como resultado final:

k=57.8

Ejemplo 4.
Operadores relacionales

Sea x=6 y b=7, ¿cuál es el resultado de la siguiente proposición?

$p=((x*0.5)+(b*3))\leq((x*3)\%B)$

El primer paso, de acuerdo con el orden de precedencia, es la evaluación de las expresiones entre paréntesis. Teniendo en cuenta que existen varias operaciones con paréntesis, las primeras en ser evaluadas serán las más sencillas a lado y lado de la expresión.

$(p=(x*0.5)+(b*3))\leq((x*3)\%B)$

Reemplazando los valores de x y b:

$p=(3+21)\leq(18\%B)$

Posteriormente, se evalúa la sumatoria a un lado de la expresión y el residuo de la división al otro y se obtiene:

$p=24\leq4$

Como se puede observar, el elemento restante en la operación corresponde al operador relacional *menor igual*, cuyos resultados posibles son falso o verdadero dependiendo del cumplimiento de la condición. En este caso el 24 no es menor o igual que 4, por lo tanto, el resultado final de la expresión será:

p=falso

Ejemplo 5.
Operadores lógicos

¿Cuál es el resultado de la siguiente expresión lógica donde a=1 , b=2, c=3?

$w=not[\ (\ (a>b)or(a<c))and\ (\ (a=c)or(a\geq b))\ ]$

El primer paso consiste en evaluar las operaciones entre paréntesis más simples de cada elemento de la expresión.

$w=not\ [\ (\ (a>b)or(a<c))and\ (\ (a=c)or(a\geq b))]$

Reemplazando ***a, b, c*** y evaluando las operaciones de relación se obtiene:

$w=not[(\ (1>2)or(1<3))and\ (\ (1=3)or(1\geq 2))]$
$w=not[(\ (F)or(T))and\ (\ (F)or(F))]$

Evaluando las operaciones lógicas OR

$w=not[(\ T)and\ (\ F)]$

Evaluando el operador lógico AND

$w=not[F]$

Finalmente evaluando la operación lógica NOT se obtiene el resultado final de la operación:

$w=T$

Ejercicio

Ejercicio 2. Evaluación de expresiones

Evalúe las siguientes expresiones donde
x=2, y=3, z=-4, w=6, *p=falso, q=verdadero, r=falso*

1. $x+z*y/w$
2. $14 \% w \% y$
3. $|r|(|p|!q || !r$ && $(3>4))$
4. $(x+z)*y/w$
5. $y\text{-a}bs(z)*w$

Actividad 2.
Operaciones y operadores

Objetivo general

Aplicar los conocimientos básicos de la lógica matemática necesarios para elaboración de los programas de computador.

Objetivos específicos

- Aplicar la jerarquía de los operadores en la solución de operaciones.
- Comprender las diferencias en condiciones de comparación unidas por operadores lógicos.
- Revisar los resultados de verdad al utilizar operadores relacionales.

Generalidades

La lógica matemática generalmente es considerada obvia por algunos aprendices de la programación y no es así, muchos errores en los programas se deben a su mal uso, esta actividad invita al aprendiz a generar los resultados para algunas situaciones planteadas.

Vocabulario

- Operadores relacionales
- Operadores lógicos
- Operadores matemáticos
- Operaciones

Ejercicios

Evalúe las siguientes expresiones empleando el orden de los operadores y obtenga el resultado correspondiente.

1. (5%3)-(((9*4)/8*2)+3)

2. ((5%3)-((9*4)/(8*2)))+3

3. 5%(3-(9*4)/8*(2+3)))

4. Sea x=6 = 6 y b=7, resuelva la siguiente expresión:
   $(x*0.5)+(b*3)>((x*3)\%b)$

5. Sea x=2 = 6 y b=3, resuelva la siguiente expresión:
   $((x*5)+(b*2))>((b*3) \% x)$

Capítulo 3
Principios de la codificación

3.1 Tipos de datos

En los programas de computador existen diferentes clases de información a capturar lo cual genera que existan diferentes tipos de datos que resultan adecuados para representar cada uno de los conceptos siguientes. Por ejemplo, el sueldo de un trabajador, la edad de una persona, el número de hijos, el estado civil, el vencimiento de un producto, el número de teléfono, la dirección o la temperatura en un sensor.

Un tipo de dato corresponde al atributo que indica al computador y al programador algo sobre la clase de dato que se va a procesar y el espacio que debe asignársele en la memoria RAM del computador. Por ejemplo:

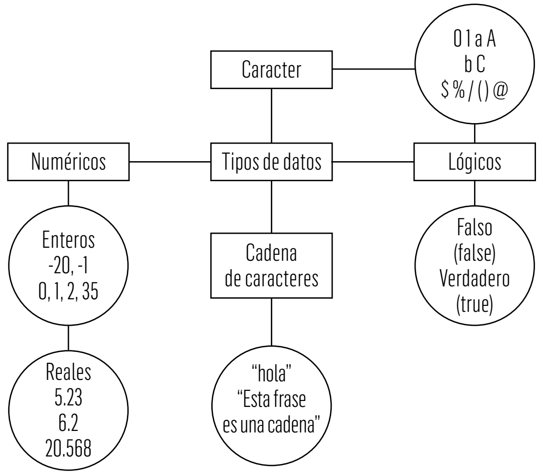
1. **Numéricos**: corresponde a los datos cuantificables:
   - Enteros: el salario de una persona corresponde a un valor entero.
   - Reales: los intereses ganados en algún movimiento pueden ser decimales ya que la tasa de interés también lo es.

2. **Carácter**: corresponde a valores que puede tomar cualquier símbolo del teclado, también son conocidos como alfanuméricos

3. **Lógicos**: corresponde a los valores que puede tomar una condición, es decir Verdadero (Sí) o Falso (No). Por ejemplo: ¿ya son las tres de la tarde? o 7 > 5

Una representación de los tipos de datos se muestra en la figura 10. Es importante resaltar que la información clasificada dentro de los tipos de datos es manejada internamente dentro del computador empleando los sistemas de numeración binario, decimal y hexadecimal. En el apéndice A se presentan los sistemas de numeración, su distribución, estructura y los diferentes métodos de conversión entre sistemas. Cabe resaltar que esta sección es opcional y será de ayuda para los programadores que deseen profundizar en el diseño y en la arquitectura de computadores.

Figura 10. Tipos de datos



3.2 Variables

Una variable es la representación de datos en el computador, un espacio en memoria RAM del computador cuyo nombre o identificador puede ser reconocido por el programa. El espacio contiene un dato o valor con el que se pueden hacer operaciones en el programa. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. En la tabla 8 se presentan algunos ejemplos de variables.

Atendiendo a los estándares de programación, y con el objetivo de comenzar a desarrollar buenas prácticas de programación en el lector, se sugiere utilizar nombres de variables comprensibles y que estén relacionados con la función de esta dentro del programa. Además, se recomienda emplear la notación de *camello* para nombrar las variables de un programa. Este tipo de notación indica que el nombre de las variables debe comenzar con una letra minúscula y si el nombre está compuesto por varias palabras, la primera letra

de cada nueva palabra se encontrara en mayúscula. En la tabla 8 se observa, por ejemplo, la variable anioNacimiento, la cual corresponde al año de nacimiento de un usuario. En este caso la primera letra de la palabra anio se encuentra en minúscula, mientras que la primera letra de la palabra nacimiento se encuentra en mayúscula. De la misma forma ocurre para la variable notaDefinitiva presente también en la tabla 8.

Tabla 8. Ejemplo de representación de variables

Nombre de la variable	Significado	Posibles valores
anioNacimiento	Año de nacimiento del usuario	1977, 1991, 1995
notaDefinitiva	Definitiva de la asignatura	4.4, 4.9, 2.9

Los tipos de datos presentados en la tabla 9 (página siguiente) definen una variable de acuerdo con las siguientes características:

- Identificador: posición de memoria.
- Constante: dato que no cambia durante la ejecución de un programa.
- Variable: objetos que cambian su valor durante la ejecución de un programa.

3.3 Lenguaje de programación

Según Saavedra (2010), un lenguaje de programación es un conjunto de comandos que puede ser utilizado para controlar el comportamiento de una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente. Es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras, un sistema de comunicación que

Tabla 9. Tipos de datos

Tipo básico	Tipo	Ejemplo	Tamaño en Bytes	Rango mínimo...máximo
Caracter	char	'C'	1	0...255
Entero	short	-15	2	-128...127
	int	1024	2	-32768...32767
	unsigned int	42325	2	0...65535
	long	262144	4	-2147483648...2147483647
Real	float	10.5	4	$3.4*10^{-38}...3.4*10^{38}$
	double	0.00045	8	$2.7*10^{-308}...2.7*10^{308}$
	long double	1,00E-08	8	Igual que double

posee una determinada estructura, contenido y uso. Algunas de las instrucciones que debe desarrollar un programa y que corresponden con los elementos de la tabla 3 (Símbolos para diagramas de flujo), son presentadas en la tabla 10.

Tabla 10. Comandos iniciales en un lenguaje de programación

Nombre	Símbolo	C++	Python	Java	Pseudocódigo
Entrada		`cin>>`	`input`	`showInputDialog` `teclado.nextInt`	Leer
Decisión		Pertenece a varias estructuras de control y de ello depende el comando que se debe utilizar, ya sea If, DoWhile o While			Si se cumple la condición De lo contrario
Salida		`cout<<`	`print`	`System.out.print`	Imprimir

3.4 Lectura y escritura de datos

Aprender la lectura y la escritura de datos de un lenguaje de programación es el paso clave para empezar a programar en él. Mediante la lectura de datos podemos hacer que el programa reciba datos necesarios que solo el usuario puede ingresar y así realizar los procesos que se tengan programados sobre ellos. La escritura de datos permite entregar salidas al usuario como resultado de aplicar operaciones sobre datos, mensajes a mostrar como ayudas, bienvenidas o también resultados u otros usos que se les quiera dar a las salidas.

A continuación, se presentará la forma de leer y escribir datos así como la estructura básica de un programa para cada uno de los tres lenguajes que se estudiarán en el libro (C++, Java y Python) en su modo más básico.

3.4.1 C++

Inicialmente, antes de definir las formas de entradas y salidas, se debe conocer la estrucutra general de un programa. A continuación, se muestra la estructura de un programa en C++ en el entorno de desarrollo visual studio.

```
//Librerias                                  1
#include <iostream>

//Variables                                  2
using namespace std;
int dato = 10;
float numero = 25.32;
string nombre = "pedro";

//Programa principal                         3
int main()
{
    cout<<"hola mundo";
}
```


Como se puede observar, un programa en C++ tiene tres secciones principales. La primera sección identificada con el número 1 corresponde a las librerías utilizadas en el programa. Una librería se puede entender como un conjunto de algoritmos ya desarrollados, los cuales pueden ser utilizados dentro del programa para realizar tareas específicas como mostrar un mensaje en la pantalla o capturar los datos que usará el programa. Es importante resaltar que en el caso de C++, definir las librerías es obligatorio para cualquier código que se realice. Para los ejemplos desarrollados en este libro, basta con utilizar la librería iostream, la cual gestiona los comandos de entradas y salidas del sistema como *cout* y *cin*.

La segunda sección del programa corresponde a la definición de las variables a utilizar. Este paso es necesario, ya que C++ debe conocer las variables del programa con el fin de asignar un espacio para ellas en la memoria del computador. En este ejemplo se define una variable entera llamada *dato,* una variable flotante llamada *número* y una variable tipo string o cadena de caracteres llamada nombre.

La tercera sección del programa establece el programa principal. En el caso de C++, este se encuentra demarcado por la función int main(), dentro de la cual se escribirán las instrucciones del programa. Durante el desarrollo de este documento, se sugiere a los lectores que empleen esta estructura para el desarrollo de sus programas, colocando los códigos de ejemplo presentados en este libro dentro de la sección denominada programa principal. En el sitio web del libro encontrará un tutorial para la creación del programa en cada lenguaje de programación.

Como se observa en el programa principal, el programa ejecutará el comando *cout*, el cual corresponde a la forma básica para mostrar un mensaje en la pantalla del computador. Dicho mensaje se encuentra entre comillas dobles y esta resaltado en color rojo, el cual corresponde a la frase hola mundo. A continuación se muestra el comando cout en conjunto con el resultado de la ejecución del programa y su salida en la pantalla del computador.

```
cout<<"Hola mundo";
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
Hola mundo
Presione una tecla para continuar . . .
```

Ahora si en lugar de mostrar un mensaje, se quiere visualizar una cantidad numérica, la sintaxis del comando *cout* será:

```
cout<<64;
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
64
Presione una tecla para continuar . . .
```

Si se quiere imprimir el mensaje “Hola”, el comando *cout* se escribe así:

```
cout<<"hola";
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
hola
Presione una tecla para continuar . . .
```

Usando *cout* también es posible imprimir el valor contenido en alguna variable. Por ejemplo, para imprimir la variable *nombre*, la cual tiene el valor de “Pedro”, la sintaxis de *cout* es:

```
cout<<nombre;
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
pedro
Presione una tecla para continuar . . .
```

En el caso de una variable numérica como son *número* y *dato* el procedimiento es el mismo y se imprimirán los valores 10 y 25.32 respectivamente.


```
cout<<dato;
cout<<numero;
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
10
25.32
Presione una tecla para continuar . . .
```

Para recibir un valor de teclado, C++ usa el comando *cin,* el cual recibe el flujo de datos de entrada y lo almacena en la variable que se haya especificado. Por ejemplo, si se quisiera recibir un nombre que ingrese el usuario por el teclado y almacenarlo en la variable *nombre*, se emplearía el siguiente código escrito dentro del programa principal:

```
cout<<"Ingrese el nombre del usuario";
cout<<numero;
cout<<"Su nombre es"<<nombre;
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
Ingrese el nombre del usuario luis
Su nombre es luis
Presione una tecla para continuar . . .
```

Pero si el valor que desea ingresar no es una cadena de caracteres sino un número entero, el código empleando el comando *cin* para ingresar un número por teclado y almacenarlo en la variable *dato* sería:

```
cout<<"Ingrese un numero";
cout<<dato;
cout<<"El numero ingresado es"<<dato<<".\n";
```

```
D:\visual studio 2013\hola mundo\Debug\hola mundo.exe
ingrese un numero 25
El numero ingresado es 25.
Presione una tecla para continuar . . .
```

Observe que en este caso, el comando *cout* utiliza un mayor número de elementos para mostrar el resultado del programa. En este caso se muestra un mensaje que dice “el número indicado es”. Luego empleando el operador << se coloca la variable *dato*.

Posteriormente, se utiliza el símbolo \n, el cual corresponde a una secuencia de escape para el salto de línea, lo cual es equivalente a pulsar la tecla enter al trabajar en un procesador de texto como Microsoft Word.

Estas secuencias de escape son constantes de caracteres que son de gran utilidad para aplicar formato al texto, haciéndolo más legible para el usuario del programa. Estas secuencias de escape son iguales para los tres lenguajes de programación del texto y se presentan a continuación:

\n → Imprime una nueva línea
\t → Aplica tabulación horizontal
\v → Aplica tabulación vertical
\\ → Imprime el símbolo \ (barra diagonal inversa)
\’ → Imprime el símbolo ‘ (comilla simple)
\” → Imprime el símbolo ‘’ (comilla doble)

3.4.2 Java

En el caso de Java, la estructura básica de un programa en el entorno de programación NetBeans se muestra a continuación:

Como se puede observar, la estructura de un programa en Java es similar a la estructura propuesta para un programa en C++. En primer lugar, se encuentran las librerías, las cuales cumplen la misma función que en el lenguaje C++. Particularmente, en Java se requiere definir la librería scanner para la gestión de entradas y salidas.

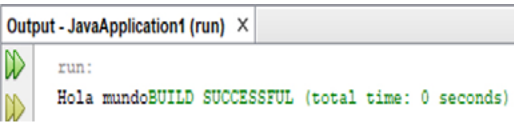
En segundo lugar, dado que Java es un lenguaje de programación orientado a objetos se requiere la definición de una clase principal para el programa. Dentro de ella se agrupará el programa principal, así como otros elementos que se deseen utilizar dentro del programa.

En tercer lugar, se encuentra el método principal del programa en Java, que funciona de la misma manera que la función int main() definida en lenguaje C++. Es decir, en esta sección se escribirán las instrucciones del programa así como se definirán las variables que usará el programa. Cabe resaltar que dentro del programa principal se encuentra la definición de la variable *teclado* utilizando la librería scanner. En este caso particular, y dado que Java es orientado a objetos, esta *variable* teclado será necesaria para capturar información desde el teclado.

Se recomienda a los lectores que deseen comenzar por Java como lenguaje de programación, seguir la estructura presentada rigurosamente para facilitar el desarrollo de los programas.

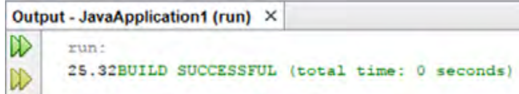
En este programa, se presenta inicialmente la impresión del mensaje "Hola mundo". Debido a que se está trabajando sobre la consola, para escribir datos en Java se usará el método *System.out.print().* Obsérvese que, aunque la consola es diferente a la utilizada por el lenguaje C++, cumple el mismo propósito, es decir, mostrar el resultado de la instrucción ejecutada.

```
System.out.print("Hola mundo");
```



Al igual que como se explicó para C++, la salida también nos permite mostrar valores de variables y mensajes. Por ejemplo, si queremos imprimir el valor de la variable *número:*

```
System.out.print(numero);
```



Ahora, si se desea imprimir la variable *nombre* , la cual tiene el valor de "Juan", la sintaxis de *System.out.print()* es:

```
System.out.print(nombre);
```


En el caso de una variable numérica como son *número* y *dato* el procedimiento es el mismo y se imprimirán los valores 10 y 25.32 respectivamente. Cabe resaltar que en este caso se ha incluido un salto de línea (\n) para mejorar la visualización de los resultados en la consola.

```
System.out.print(dato);
System.out.print("\n");
System.out.print(numero);
```

```
Output - JavaApplication1 (run)
run:
10
25.32BUILD SUCCESSFUL (total time: 0 seconds)
```

Para recibir datos por teclado, en Java se utiliza el objeto *teclado* creado a partir de la clase scanner. En este caso, para recibir un nombre tecleado por el usaurio y almacenarlo en la variable *nombre*, se emplearía el siguiente código escrito dentro del programa principal:

```
System.out.println("Ingrese un nombre");
nombre=teclado.nextLine();
System.out.println("Su nombre es"+nombre);
```

```
Output - JavaApplication1 (run)
run:
Ingrese un nombre
pepe
Su nombre es pepe
BUILD SUCCESSFUL (total time: 4 seconds)
```

Como se puede observar, empleando *teclado.nextLine(),* se toma la información digitada por teclado. En este caso particular, el comando *System.out.println* permite mostrar la información en la pantalla y además incorpora el salto de línea. Esto quiere decir que no es necesario incluir esta secuencia de escape de forma adicional.

Pero si el valor que desea ingresar no es una cadena de caracteres sino un número entero, el código debe modificarse empleando el comando *teclado.nextInt()* para ingresar un número entero por teclado y almacenarlo en la variable *número* como se muestra a continuación.

```
System.out.println("Ingrese un número");
dato=teclado.nextInt();
System.out.println("el numero ingresado es"+dato);
```

```
Output - JavaApplication1 (run)
run:
Ingrese un número
23
el numero ingresado es 23.0
BUILD SUCCESSFUL (total time: 3 seconds
```

Por otra parte, si el número que se desea capturar es flotante o doble el comando a utilizar es *teclado.nextDouble().* A continuación se muestra el código para capturar un número *double* por teclado y almacenarlo en la variable número.

```
System.out.println("Ingrese un número");
número=teclado.nextDouble();
System.out.println("el numero ingresado es"+numero);
```

```
Output - JavaApplication1 (run)
run:
Ingrese un número
45,981
el numero ingresado es 45.981
BUILD SUCCESSFUL (total time: 5 seconds)
```


3.4.3 Python

A diferencia de C++ y Java, el desarrollo de programas en Python no requiere de una estructura específica conformada por un método o función para definir el programa principal. En este caso basta con definir las instrucciones directamente sobre el editor de código.

En Python también existen varias funciones que permiten ingresar e imprimir datos, sin embargo, las más usadas son *input* (para lectura) y *print* (para escritura).

Cuando se quiere mostrar el valor de una variable, imprimir un número o una cadena de texto, use la función *print()* con los valores deseados dentro de los paréntesis y comillas dobles. A continuación se presenta la impresión del mensaje "hola mundo" y su resultado en la consola de Python. Al igual que con C++ y Java, el mensaje se imprime en la consola, la cual es diferente en Pyhton pero cumple el mismo propósito.

```
print("hola mundo")
```

```
>>>
hola mundo
>>>
```

De la misma forma que en C++ y Java, la salida también nos permite mostrar valores de variables y mensajes. Por ejemplo, si queremos imprimir el valor de la variable *número* en Python se realiza de la siguiente forma:

```
numero=10
print(numero)
```

```
>>>
10
>>>
```

Como se puede observar, en el código de Python inicialmente se define la variable *número* la cual tiene el valor de 10. Posteriormen-

te, el valor de la variable es mostrado en la consola empleando el comando print. Ahora, si la variable *número* almacena un valor de tipo flotante que corresponde a 25.32, la forma de mostrarlo en la pantalla es la siguiente:

```
numero=25.32            >>>
print(numero)           25.32
                        >>>
```

Como se puede ver, el proceso para imprimir un número decimal es el mismo que para un número entero, basta con definir el valor decimal en la variable para posteriormente imprimirlo en la consola.

Si se desea imprimir dos variables numéricas como son *número* con valor de 10 y *dato* con valor de 25.32 el procedimiento es el que se muestra en la siguiente imagen. A diferencia de C++, el comando *print* trae incluido el salto de línea (\n) para mejorar la visualización de los resultados en la consola. En este caso nuevamente se definen primero las variables *número* y *dato* y posteriormente se imprimen empleando el comando print.

```
numero=10               >>>
dato=25.32              10
print(numero)           25.32
print(dato)             >>>
```

Para recibir datos por teclado, en Python se utiliza el comando *input()*. En este caso, para recibir un nombre digitado por el usuario y almacenarlo en la variable *nombre,* se emplearía el siguiente código:


```
                                    >>>
nombre=input("ingrese su nombre")   Ingrese su nombre pedro
print(nombre)                       pedro
                                    >>>
```

Como se puede observar, empleando *input* se despliega en la consola el mensaje ingrese su nombre el cual está entre comillas y en color verde. Posteriormente se da un espacio para ingresar el valor de la variable *nombre* el cual corresponde a la palabra Pedro. Finalmente, el contenido de la variable nombre se muestra en la consola.

Sin embargo, cuando el valor que se desea capturar por teclado corresponde a un número entero almacenado en la variable *número*, la sintaxis del comando *input* cambia de la siguiente manera:

```
                                      >>>
numero=int(input("Ingrese un valor")) Ingrese un valor 10
print(numero)                         10
                                      >>>
```

Como se puede observar, al comando *input* se antepone la palabra *int,* la cual en este caso corresponde a la definición del tipo de dato que se desea capturar por teclado, el cual corresponde al valor entero 10. Ahora, si el número que se desea capturar es un número decimal, la sintaxis del comando *input* es:

```
                                        >>>
numero=float(input("Ingrese un valor")) Ingrese un valor 25.32
print(numero)                           25.32
                                        >>>
```

En este caso, el comando input va acompañado de la palabra *float*, con la cual se indica que el valor que se desea capturar es un número decimal correspondiente a 25.32. En el caso que se desee imprimir un mensaje más detallado, la sintaxis del comando print puede modificarse como:

```
                                              >>>
numero=float(input("Ingrese un valor")) Ingrese un valor 10
print("El numero ingresado es",numero) El numero ingresado es 10.0
                                              >>>
```

Como se puede observar, dentro del comando *print* se coloca el mensaje *el número ingresado es*, el cual esta entre comillas. Observe que el mensaje se encuentra separado de la variable número por medio de una coma, lo que indica que lo primero que se mostrará en la consola es el mensaje y posteriormente el valor de la variable número que corresponde a 10.

Capítulo 4
Situaciones elementales

Teniendo en cuenta que en este punto el lector ya ha realizado una revisión del documento y tiene una nueva perspectiva del proceso de la programación, a continuación se plantean los siguientes ejemplos aplicados donde se empleará el proceso de la programación para dar solución a problemas específicos usando los lenguajes de programación C++, Java y Python.

Ejemplo 6.
Operaciones_básicas

En matemáticas, las operaciones básicas que se pueden aplicar sobre un conjunto de números enteros son la suma (+), la resta (-), la multiplicación (*) y la división (/). Un programa sencillo para comenzar puede ser este: dados dos números ingresados por el usuario, hallar la suma, la resta, el producto y la división entre ellos. Después de esto mostrar en pantalla cada uno de los resultados.

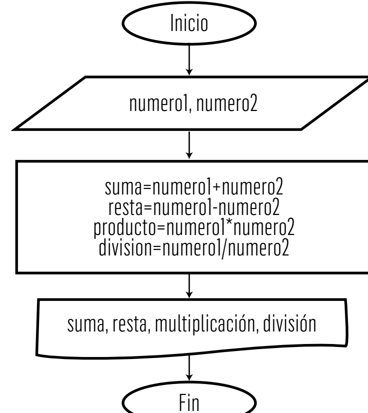
Análisis del problema

En el ejemplo se requieren dos datos de entrada: *numero1* y *numero2*. Observe la correspondencia de las entradas con el símbolo representado en la siguiente figura. Como datos de salida se necesitan las variables: *suma*, *resta*, *producto* y *division,* las cuales almacenarán el resultado de las operaciones desarrolladas. En total este programa requerirá de 6 variables, 2 de entrada y 4 de salida.

Diseño del programa

El siguiente paso es la construcción del diagrama de flujo, presentado en la figura 11, el cual identifica claramente los datos de entrada, las operaciones y los datos de salida. Como puede observarse, el primer paso consiste en ingresar los valores *numero1* y numero2 para posteriormente calcular las operaciones de suma, resta, multiplicación y división. Finalmente se muestra el resultado de dichas operaciones.

Figura 11. Diagrama de flujo para el ejemplo OperacionesBásicas



El siguiente paso corresponde a la prueba de escritorio presentada en la tabla 11. Aquí se pueden observar las variables de entrada con diferentes valores de prueba así como el valor esperado de las variables de salida.

Tabla 11. Entradas y salidas de operaciones_básicas

Entradas		Salidas*			
numero1	*numero2*	*suma*	*resta*	*producto*	*division*
20	12	32	8	240	1
8	40	48	-32	320	0
15	3	18	12	45	5

* Se debe revisar que el diagrama de flujo entregue las salidas

Codificación del algoritmo

El paso siguiente es la codificación. En la tabla 12 se presenta el código correspondiente al diagrama de flujo en C++, Java y Python que da solución al problema abordado. Como se puede observar, cada uno de los lenguajes de programación emplea sus propios comandos de entrada y salida para la codificación del algoritmo.

Tabla 12. Solución al ejemplo operaciones_básicas empleando C++, Java y Python

C++	Etapas
`int numero1;` `int numero2;`	Definir las variables
`cout<<” Ingrese el primer número”;` `cin>> numero1;`	Carga por teclado
`cout<<” Ingrese el segundo número”;` `cin>> numero2;`	Carga por teclado
`suma = numero1 + numero2;`	Operación
`resta = numero1 -numero2;`	Operación
`producto = numero1 * numero2;`	Operación
`division = numero1 / numero2;`	Operación
`cout<<suma;` `cout<<producto;`	Salida por pantalla

Java	Etapas

`Scanner teclado = new Scanner(System.in);`
`int numero1;`
`int numero2;`
	Definir las variables

`System.out.print(“Ingrese el primer número”);` `numero1 = teclado.nextInt();`	Carga por teclado

Continúa

Java	Etapas
`System.out.print(“”Ingrese el segundo número”);` `numero2 = teclado.nextInt();`	Carga por teclado
`suma = numero1 + numero2;`	Operación
`resta = numero1 – numero2;`	Operación
`producto = numero1 * numero2;`	Operación
`division = numero1 / numero2;`	Operación
`System.out.print (suma);` `System.out.print (producto);`	Salida por Pantalla

Python	Etapas
	Definir las variables
`numero1 = int(input(“Ingrese el primer numero”))`	Carga por teclado
`numero2 = int(input(“Ingrese el segundo numero”))`	Carga por teclado
`suma = numero1 + numero2`	Operación
`reta = numero1 - numero2`	Operación
`producto = numero1 * numero2`	Operación
`division = numero1 / numero2`	Operación
`print(suma)` `print(producto)`	Salida por Pantalla

Ejemplo 7.
Nómina_básica

En una fábrica de automóviles se necesita conocer cuál es el sueldo mensual que se paga a cada uno de sus operarios. Si el sueldo se calcula hallando el producto entre las horas de trabajo por mes y

el costo por hora trabajada, diseñe un programa que permita a la fábrica calcular el sueldo de cada operario.

Análisis del problema

En el ejemplo se necesitan dos datos de entrada: *horasTrabajo* y *costoHora*. La variable *horasTrabajo* almacena la cantidad de horas trabajadas por el operario. La variable *costoHora* almacena el precio de una hora de trabajo. Como salida del programa se tiene la variable *sueldo* que almacena el sueldo a abonar al operario.

La elección del nombre de una variable debe ser representativa. En el ejemplo el nombre *horasTrabajo* es lo suficientemente claro para dar una idea acabada sobre su contenido.

Es posible darle otros nombres adecuados, pero debe evitarse el uso de nombres como, por ejemplo, *hTr*. Lo que se busca es que al resolver un problema, dicho nombre recuerde el significado para el valor que almacena en el programa, es decir, cuando pase el tiempo y se deba realizar alguna actividad de mantenimiento, y por ende se lea el diagrama, sea fácilmente posible recordar y entender.

Diseño del programa

Definiendo las variables de entrada y salida del algoritmo, la figura 12 (página siguiente) presenta el diagrama de flujo del algoritmo. Como se puede observar, inicialmente se solicitan las horas trabajadas por el empleado, así como el valor de la hora trabajada. Luego se calcula el salario del trabajador como el producto entre las horas trabajadas y el valor de cada hora. Finalmente se muestra el resultado en la pantalla.

La prueba de escritorio del algoritmo se presenta en la tabla 13 donde el valor del salario final es mayor si el valor de las horas trabajadas es más alto al igual que el número de horas.

Figura 12. Diagrama de flujo para el ejemplo nómina_básica

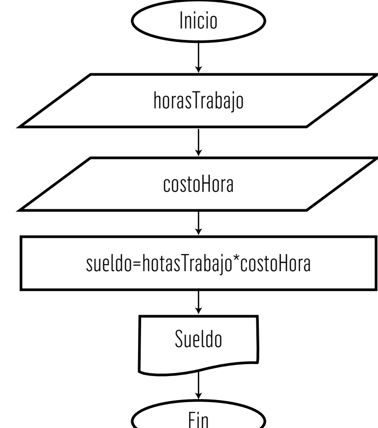


Tabla 13. Entradas y salidas del programa

Entradas		Salidas
horasTrabajo	*costoHora*	*sueldo*
20	25000	500000
15	18000	270000
11	23000	253000

Codificación del algoritmo

En la tabla 14 se presenta el algoritmo para el cálculo de la nómina de un empleado codificado en C++, Java y Python. Como indica la tabla, cada una de las instrucciones presentadas simboliza una de las etapas del algoritmo en cada lenguaje de programación.

Tabla 14. Algoritmo en C++, Java y Python para el ejemplo nómina_básica empleando C++, Java y Python

C++	Etapas

`int horasTrabajo;`
`int costoHora;`
`int sueldo;`
	Definir las variables

`cout<<" Ingrese las horas trabajadas";` `cin>> horasTrabajo`	Carga por teclado
`cout<<"Ingrese el costo de la hora";` `cin>>costoHora;`	Carga por teclado
`Sueldo = horasTrabajo * costoHora;` `cout<<sueldo;`	Operación
`cout<<sueldo;`	Salida por Pantalla

Java	Etapas

`Scanner teclado=new Scanner(System.in);`
`int horasTrabajo;`
`int costoHora;`
`int sueldo;`
	Definir las variables

`System.out.print("Ingrese las horas");` `horasTrabajo = teclado.nextInt();`	Carga por teclado
`System.out.print("Ingrese costo de hora");` `costoHora = teclado.nextInt();`	Carga por teclado
`Sueldo = horasTrabajo * costoHora;`	Operación
`System.out.print (Sueldo);`	Salida por pantalla

Python	Etapas
	Definir las variables
`horasTrabajo = int(input("Ingrese horas trabajadas"))`	Carga por teclado
`sueldo = int(input("Ingrese el costo de la hora"))`	Carga por teclado
`Sueldo = horasTrabajo * costoHora`	Operación
`print(sueldo)`	Salida por pantalla

Ejemplo 8.
Área_triángulo

El área de un triángulo rectángulo se calcula multiplicando la medida de su base por su altura y dividiendo el resultado entre 2. Desarrolle un programa que permita al usuario obtener el área de un triángulo rectángulo teniendo estas medidas.

Análisis del problema

Para el desarrollo de este programa, se requieren tres variables. Como variables de entrada se necesitan el valor de la base del triángulo denominada *base* y el valor de la altura del triángulo llamada *altura*. Como variable de salida se denomina la variable área la cual almacenará el valor del área del triángulo. Para calcular el área del triángulo se utiliza la expresión (2).

$$area = base * altura / 2 \quad (2)$$

Diseño del algoritmo

En este paso se construye el diagrama de flujo para el cálculo del área de un triángulo, el cual es presentado en la figura 13. Para esto, inicialmente se ingresan los valores de la base y la altura del triángulo. Posteriormente, se calcula el área empleando (2) a partir de los valores de la base y la altura. Finalmente se presenta el resultado en la pantalla.

Como siguiente paso se desarrolla la prueba de escritorio, la cual se muestra en la tabla 15. Nótese que para diferentes valores de la base y la altura se obtiene el correspondiente valor del área del triángulo.

Figura 13. Diagrama de flujo para el ejemplo área_triángulo

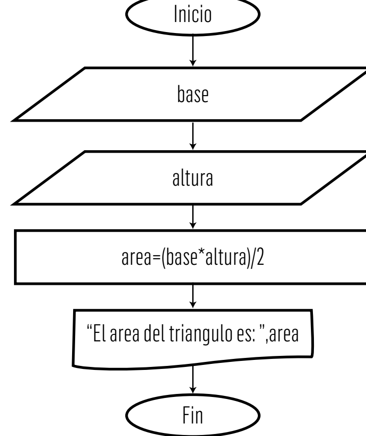


Tabla 15. Entradas y salidas del programa área_triángulo

Entradas		Salidas
base	*altura*	*area*
20	4	40
15	10	75
10	20	100

Codificación del algoritmo

En la tabla 16 se presenta la codificación del algoritmo en C++, Java y Python. Como se observa, cada una de las instrucciones corresponde a una sección específica del diagrama de flujo, representada de acuerdo con la sintaxis del programa.

Tabla 16. Algoritmo en C++, Java y Python para el ejemplo área_triángulo

C++	Etapas
`double base, altura, area;`	Definir las variables
`cout<<" Ingrese el valor de la base del triangulo: "; cin>>h;`	Carga por teclado
`cout<<" Ingrese el valor de la altura del triangulo: "; cin>>b;`	Carga por teclado
`area = (base*altura)/2;`	Operación
`cout<<" El área del rectángulo es:"<<-base;`	Salida por pantalla

Java	Etapas
`System.out.print("”Ingrese la base”);` `base = teclado.nextInt();`	Definir las variables
`System.out.print("”Ingrese la altura”);` `altura = teclado.nextInt();`	Carga por teclado
`area = (base*altura)/2`	Carga por teclado
`System.out.print (sueldo);`	Operación
`System.out.print("”Ingrese la base”);` `base = teclado.nextInt();`	Salida por pantalla

Python	Etapas
	Definir las variables
`base = int(input("Ingrese la base: "))`	Carga por teclado
`altura = int(input("Ingrese la altura: "))`	Carga por teclado
`area = (base*altura)/2`	Operación
`print("El área del rectángulo es: ", a)`	Salida por pantalla

Ejercicios

Ejercicio 3. boletas de la feria

En una feria se tiene instalada una montaña rusa y para subir a ella se requiere que las personas entreguen su boleta de manera ordenada. Cada boleta posee un número que la identifica del resto. Debido a que la boleta también se usa para los demás juegos de la feria, es requerido que las boletas se devuelvan a cada persona comenzando por la última que subió a la montaña rusa y terminando por la primera. Suponiendo que una mañana se atendieron únicamente a tres personas, idee la forma de saber los números de las boletas en el orden de salida. Construya un programa que pida al usuario los números de las tres boletas y los devuelva en el orden contrario a como se ingresaron.

Ejercicio 4. Área del rectángulo

El área de un rectángulo puede hallarse teniendo la medida de su base y de su altura mediante la fórmula:

*area=base*altura*

Construya un programa que reciba del usuario las medidas de la base y la altura y permita determinar el área del rectángulo correspondiente.

Ejercicio 5. Ley de Ohm

En un laboratorio de electrónica se necesita calcular el voltaje que atraviesa un circuito sobre el cual se está experimentado. Mediante algunos análisis que se le hacen, se ha establecido la resistencia y la intensidad que presenta la corriente que atraviesa el circuito. Teniendo la fórmula:

*voltaje=resistencia*intensidad*

construya un programa que, una vez ingresados los valores de la resistencia y la intensidad, permita al usuario calcular el voltaje del circuito.

Ejercicio 6. Cajero automático

Un cajero electrónico entrega el dinero a retirar en billetes de $50, $20 y $10. Construya un programa que calcule la cantidad de billetes de cada valor que el cajero debe entregar al usuario según el valor a retirar ingresado. Asuma que el usuario asigna siempre un valor a retirar permitido.

Actividad 3.
Representación de los algoritmos

Objetivo general

Construir la tabla de análisis de un requerimiento común en donde sea posible identificar las variables de entrada y de salida y el proceso que permitirá la transformación haciendo uso de algún conjunto de datos de ejemplo.

Objetivos específicos

- Identificar los datos de entrada y de salida.
- Asignar un nombre y tipo a cada una de las variables.
- Definir las operaciones que se deben ejecutar en el futuro programa.
- Representar el algoritmo empleando un diagrama de flujo.
- Codificar el algoritmo en C++, Java o Pyhton.

Generalidades

La clave del éxito en la programación radica en comprender las necesidades del cliente. En este nivel básico, se busca identificar los datos con que se cuenta al iniciar un programa y las operaciones que se le deben aplicar para obtener los resultados.

Vocabulario

- Variables de entrada y salida
- Operaciones matemáticas
- Diagrama de flujo
- Prueba de escritorio
- Codificación del algoritmo

Ejercicios

1. Una persona tiene un terreno campestre al cual desea construirle una valla que lo rodee. Para saber cuánto debe gastar en los materiales, la mejor forma de calcular la longitud de los alrededores del terreno es hallando el perímetro. Debido a que el terreno tiene forma cuadrada, el perímetro se calculará multiplicando la longitud de un lado por cuatro. Desarrolle un programa que reciba el valor del lado de un cuadrado y muestre por pantalla el perímetro del mismo.

2. Un colegio le pide que desarrolle un programa con el cual los niños puedan practicar dos operaciones básicas: la suma y la multiplicación. El programa deberá recibir cuatro números enteros. Realice la suma de los dos primeros y calcule el producto entre los dos restantes.

3. Los principales gastos mensuales de un padre de familia corresponden a educación, alimentación, salud y transporte. Él desea saber cuál es el total de gastos y el valor promedio de estos para un mes. Desarrolle un programa que tome el valor de los gastos y entregue la sumatoria y el promedio total de gastos mensuales de la familia.

4. En una tienda se venden frutas, vegetales y algunos otros productos alimenticios. Las cuentas para cada una de las ventas normalmente se hacen a mano, pero debido a que la tienda está empezado a recibir una cantidad mayor de clientes necesitan de un programa que agilice los cálculos y minimice errores humanos. Desarrolle un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

5. Dialogue con sus compañeros de programación sobre las siguientes preguntas:

- ¿Podemos enseñar a resolver un problema?
- ¿Podemos enseñar a analizar el mundo?
- ¿Podemos enseñar a pensar?

Capítulo 5
Situaciones con estructuras de control

Este capítulo presenta las diferentes estructuras de control con su respectiva semántica, gramática y sintaxis en diferentes lenguajes de programación, a medida que avance en la lectura y aplicación del texto irá adquiriendo la habilidad necesaria.

5.1 Estructura de control condicional *If-Else*

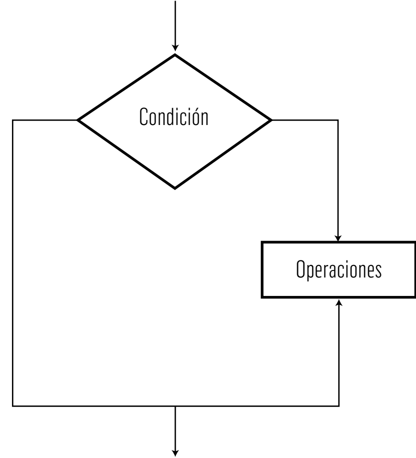
No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando es necesario tomar una decisión se deben agregar a los diseños las estructuras condicionales. En la vida diaria se presentan situaciones donde es necesario decidir, por ejemplo:

- ¿Elijo la carrera A o la carrera B?
- Para ir al trabajo, ¿elijo el camino A o el camino B?
- Al cursar una carrera, ¿elijo el horario de la mañana, de la tarde o de la noche?

Por ello, en un problema se combinan estructuras secuenciales con condicionales (y otras que se estudiarán más adelante). Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna. La representación gráfica que se debe agregar a los diagramas se presenta en la figura 14 (página siguiente).

Como se observa en la figura 14, el rombo representa la condición de la decisión y las dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda. Se trata de una estructura **condicional simple** porque por el camino del verdadero hay actividades y por el camino del falso no hay actividades. Por el flujo de datos de la opción del verdadero pueden existir varias operaciones, entradas y salidas, inclusive, más adelante, es posible que existan otras estructuras condicionales.

Figura 14. Estructura condicional IF



Ejemplo 9.
parImpar

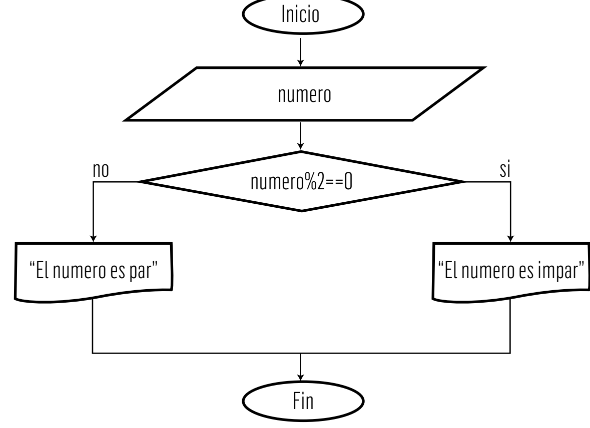
Se puede determinar que un número es par cuando al dividirlo por 2 su residuo es 0. En el caso en que el residuo fuera 1, se sabría que el número es impar. Diseñe un programa que reciba un número entero del usuario y determine si es par o no. (Consejo: acuérdese de la función del operador *mod*). La tabla 17 (página siguiente), presenta las variables del problema con algunos datos de prueba.

El siguiente paso es la construcción del diagrama de flujo presentado en la figura 15 (página siguiente), el cual identifica claramente los datos de entrada, las operaciones y los datos de salida. En el ejemplo hay un dato de entrada: *num,* el cual almacena el número ingresado y después de aplicar la operación correspondiente muestra el mensaje “Es par” o “Es impar”, dependiendo de cuál fue la entrada.

Tabla 17. Entradas y salidas para el ejemplo par_impar

Entradas	Tipo de dato	Valor	Salida
numero	entero	26	par
		1245	impar
		1	impar

Figura 15. Diagrama de flujo para el ejemplo par_impar



A partir del diagrama de flujo se realiza la codificación del algoritmo. En la tabla 18 se presenta el algoritmo codificado en los lenguajes de programación C++, Java y Python. Al codificar se lleva a cabo la definición de variables, la cual no está presente en el diagrama de flujo, en este caso sería: “int num;”.

Tabla 18. Codificación del ejemplo par_impar empleando C++, Java y Python

Código en C++

```
1 int numero;
2 cin>>numero;
3 4 if (numero%2==0)
5 {
6 cout<<”El numero es par”;
7 }
8 else
9 {
10 cout<<”El numero es impar”;
11}
```

Código en Python

```
1 numero=int(input(“Ingrese el numero”))
2 if numero%2==0:
3 print(“El numero es par”)
4 else:
5 print(“El numero es impar”)
```

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int numero;
3 numero=teclado.next.int();
4 if (numero%2==0)
5 {
6 System.out.print(“El numero es par”);
7 }
8 else
9 {
10 System.out.print(“El numero es impar”);
11}
```

Ejemplo 10.
impuestos

En una empresa, los trabajadores con un sueldo superior a $3000 deben pagar impuestos. Diseñe un programa que tome el sueldo de la persona y determine si debe o no abonar. En la tabla 19 se presentan las entradas y salidas del algoritmo.

Tabla 19. Entradas y salidas para el ejemplo Impuestos

Entradas	Tipo de dato	Valor	Salidas
salario	double	2000	No debe pagar impuestos
		4000	Sí debe pagar impuestos
		3500	Sí debe pagar impuestos

El siguiente paso es la construcción del diagrama de flujo que se muestra en la figura 16 y en el cual se identifican claramente los datos de entrada, las operaciones y los datos de salida. En el ejemplo hay un dato de entrada: *salario.* La variable *salario* almacena el sueldo de la persona; según el valor de esta variable se mostrará uno de los mensajes. La codificación del programa empleando C++, Java y Python se presenta en la tabla 20. Al codificar se lleva a cabo la definición de variables, la cual no está presente en el diagrama de flujo, en este caso sería: "double salario;". Para la prueba de escritorio, lea el diagrama de flujo y revise que los resultados concuerdan con los que se presentaron en la tabla de análisis. Las pruebas de escritorio no serán mostradas en este libro, pero le serán de gran ayuda a la hora de revisar sus ejercicios.

Figura 16. Diagrama de flujo para el ejemplo Impuestos

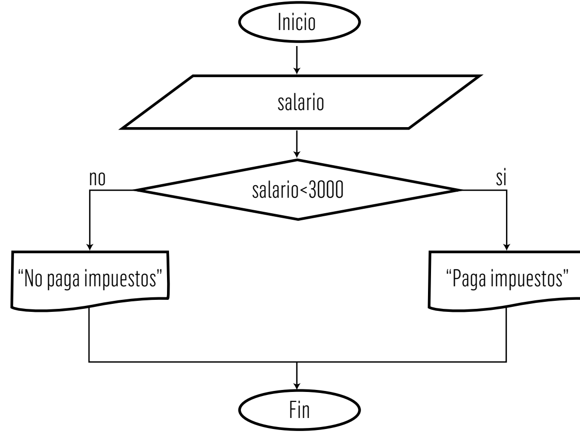


Tabla 20. Codificación empleando C++, Java y Python para el ejemplo Impuestos

Código en C++

```
1 int salario;
2 cin>>salario;
3 if (salario>3000)
4 {
5 cout<<”Paga impuestos”
6 }
7 else
8 {
9 cout<<”No paga impuestos”
10}
```

Código en Python
1 salario=int(input("Ingrese el sueldo:"))
2 if salario>3000:
3 print("Paga impuestos")
4 else:
5 print("No paga impuestos")

Código en Java
1 Scanner teclado=new Scanner(System.in);
2 int salario;
3 System.out.print("Ingrese el sueldo");
4 salario=teclado.next.int();
5 if (salario>3000)
6 {
7 System.out.print("Paga impuestos");
8 }
9 else
10{
11 System.out.print("No paga impuestos");
12}

Ejercicios

Ejercicio 7. Número mayor

En una oficina con más de 100 empleados desean saber, por cada pareja posible, cuál persona tiene un salario más alto que la otra. Se le pide diseñar un programa que pida por teclado dos números diferentes y determine cuál es el mayor entre los dos.

Ejercicio 8. Persona mayor

En una encuesta en la que participan varias personas se pide, entre varias cosas, el nombre y la edad de la persona. De estas personas se escogerán aleatoriamente dos, pero solo se tendrán en cuenta los datos de la mayor entre ambas. Construya un programa que solicite el nombre y edad de dos personas y determine el nombre de la persona con mayor edad.

Actividad 4.
Estructuras condicionales simples

Objetivo general

Diseñar diagramas de flujo con estructuras condicionales a partir de requerimientos básicos de programación siguiendo las tres primeras fases: análisis del problema, diseño del algoritmo y prueba de escritorio.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento que exija el uso de la estructura de control condicional If.
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.

Generalidades

Los diagramas de flujo describen situaciones de la vida, algunas de ellas presentan diferentes posibilidades dependiendo de los resultados que se van dando. En la programación de computadores comúnmente se presenta diferentes alternativas que deben ser evaluadas según ciertas condiciones acompañadas de operadores relacionales.

Vocabulario

- Requerimiento
- Estructura de control condicional If
- Operadores relacionales
- Prueba de escritorio

Ejercicios

1. En un centro de cambio de moneda necesitan agilizar sus operaciones mediante la compra de una aplicación que les permita hacer las operaciones de conversión de moneda de manera fiable y rápida. Para probar el software que usted les ofrece, primero le piden que les diseñe un programa que convierta únicamente entre pesos colombianos y euros.

2. Desarrolle un programa que convierta un valor en pesos a euros o que lleve a cabo la operación inversa, según decida el usuario al iniciar. Asuma que un euro cuesta 2600 pesos.

3. Realice un programa que lea por teclado dos números, si el primero es mayor al segundo informar su suma y diferencia, en caso contrario informar el producto y la división del primero respecto al segundo.

4. En una escuela, para la clase de biología, se hicieron tres evaluaciones. Para poder pasar al siguiente nivel, un estudiante debe obtener un promedio igual o mayor a 7. Escriba un programa que permita ingresar tres notas de un alumno con valores entre 1 y 10, y si el promedio de las mismas es mayor o igual a 7, el programa debe mostrar el mensaje "Promovido".

5.1.1 Combinación con operadores lógicos

La combinación de condiciones con operadores lógicos permite unir dos condiciones para colocarlas en una sola comprobación. Un ejemplo de ello es presentado a continuación.

Ejemplo 11.
Múltiplo_número

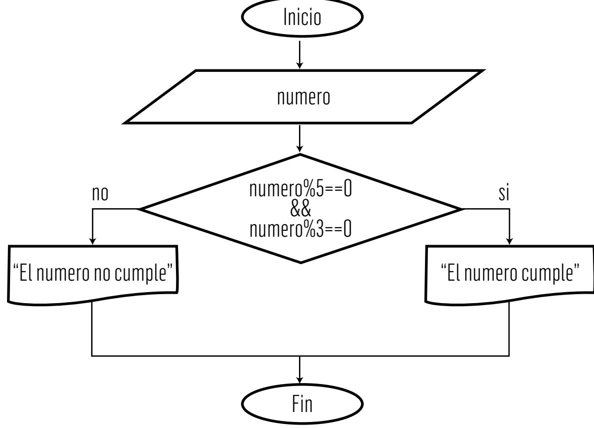
En una institución se necesita de un programa que verifique que un número es múltiplo de 15. Sin embargo, la máquina encargada de realizar el proceso solo puede aplicar el módulo a la entrada sobre un número de un dígito. Debido a esto, se decidió que, para hallar el resultado, a la entrada se le aplicarían módulo de 3 y módulo de 5, y si ambas respuestas eran correctas se sabría que el número es múltiplo de 15. Construya un programa que determine si el número ingresado por teclado es múltiplo de 3 y al mismo tiempo, múltiplo de 5. En la tabla 21 se presentan las entradas y salidas del algoritmo. Como se puede observar se establece la variable de entrada *numero.* Además, se establecen unos valores para *numero* para realizar la prueba de escritorio, con la respectiva salida.

Tabla 21. Codificación del ejemplo múltiplo_número empleando C++, Java y Python

Entradas	Tipo de dato	Valor	Salidas
numero	entero	15	El número cumple
		54	El número no cumple
		28	El número no cumple

El siguiente paso es la construcción del diagrama de flujo, presentado en la figura 17, el cual identifica claramente los datos de entrada, las operaciones y los datos de salida. En el ejemplo hay un dato de entrada: *numero*. La variable *numero* almacena el número ingresado por el usuario. Si el número es múltiplo de 3 y también múltiplo de cinco mostrará el mensaje “El número cumple”. De lo contrario mostrará “El número no cumple”.

Figura 17. Diagrama de flujo para el ejemplo múltiplo_número



La codificación del programa se presenta en la tabla 22. Al codificar se lleva a cabo la definición de variables, la cual no está presente en el diagrama de flujo, en este caso: “int *numero*;”

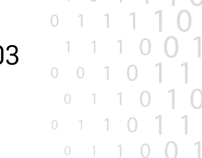


Tabla 22. Codificación en C++, Java y Python para el ejemplo múltiplo_número

Código en C++
1 int numero;
2 cout<<”Ingrese un numero”
3 cin>>numero;
4 if ((numero%5==0) && (numero%3==0))
5 {
6 cout<<”El numero cumple”
7 }
8 else
9 {
10 cout<<”El numero no cumple”
11}

Código en Python
1 numero=int(input(“Ingrese el numero:”))
2 if numero%5==0 and numero%3==0:
3 print(“El numero cumple”)
4 else:
5 print(“El numero no cumple”)

Código en Java
1 Scanner teclado=new Scanner(System.in);
2 int numero;
3 System.out.print(“Ingrese el numero”);
4 numero=teclado.next.int();
5 if ((numero%5==0) && (numero%3==0))
6 {
7 System.out.print(“El numero cumple”);
8 }
9 else
10{
11 System.out.print(“El numero no cumple”);
12}

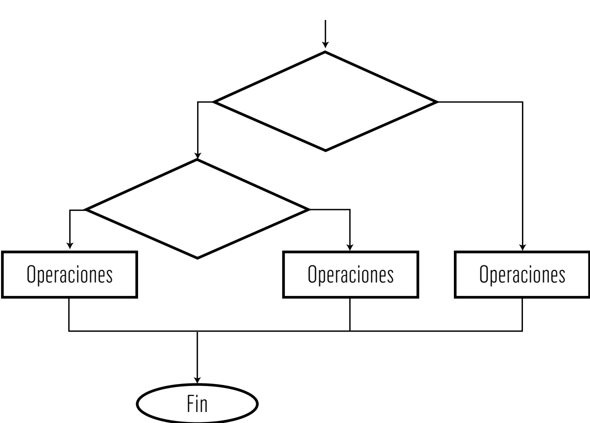
5.1.2 Unión de estructuras de control

La estructura condicional IF se puede encadenar para realizar varias comprobaciones seguidas en un mismo programa o agregar otras condiciones en caso de que en el flujo de datos positivo o negativo se deban realizar otras comprobaciones o unir dos condiciones para colocarlas en una sola comprobación. A estos recursos se les conoce con los nombres secuencial, anidado y combinación de condiciones por operadores lógicos, respectivamente. Un ejemplo de estructuras condicionales combinadas se presenta en la figura 18.

5.1.2.1 Estructura condicional anidada

Se denomina estructura condicional anidada cuando por la rama del verdadero o del falso de una estructura condicional hay otra estructura condicional. Por ejemplo, el diagrama de flujo presentado en la figura 18 contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primera estructura.

Figura 18. Unión de estructuras de control



Ejemplo 12.
Notas_condicional_anidado

En una escuela se tienen tres tipos de calificaciones. Si el promedio escolar es mayor o igual a 7, el estudiante es promovido; si está entre 4 o 7 es calificado como regular; si es menor a 4, el estudiante es reprobado. Diseñar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima el mensaje correspondiente:

- Si el promedio es mayor o igual que 7 mostrar “Promocionado”.
- Si el promedio es menor o igual que 4 y mayor que 7 mostrar “Regular”.
- Si el promedio es menor que 4 mostrar “Reprobado”.

El análisis de entradas y salidas se presenta en la tabla 23, donde se pueden observar las variables detectadas. Además, se han colocado unos datos que facilitarán posteriormente la prueba de escritorio.

Tabla 23. Entradas y salidas del ejemplo notas_condicional_anidado

Entradas			Salidas
nota1	*nota2*	*nota3*	*promedio*
7	10	9	8,67
4	4	3	3,67
9	1	6	5,33

El siguiente paso es la construcción del diagrama de flujo, presentado en la figura 19, donde se identifican claramente los datos de entrada, las operaciones y los datos de salida. En el ejemplo hay tres datos de entrada: *nota1, nota2* y *nota3*. Las variables *nota1*, *nota2* y *nota3* almacenan cada una de las notas ingresadas por el usuario. La variable *promedio* es el resultado de sumar las tres notas y di-

vidirlas por 3. Finalmente, la codificación del programa se presenta en la tabla 24.

Figura 19. Diagrama de flujo para el ejemplo notas_condicional_anidado

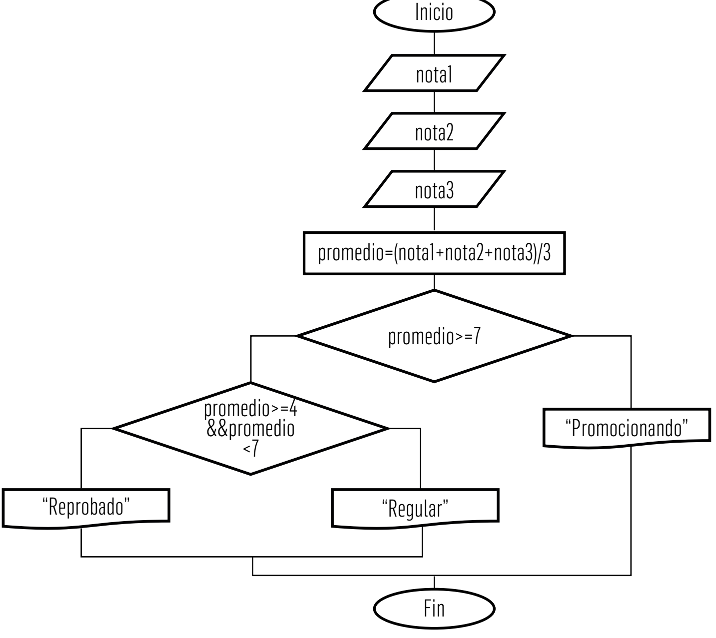


Tabla 24. Codificación en C++, Java y Python para el ejemplo notas_condicional_anidado

Código en C++
1 double nota1,nota2,nota3;
2 cout<<”Ingrese la primera nota”;
3 cin>>nota1;
4 cout<<”Ingrese la segunda nota”;
5 cin>>nota2;
6 cout<<”Ingrese la tercera nota”;

Continúa

Código en C++
7 cin>>nota1;
8 promedio=(nota1+nota2+nota3)/3
9 if (promedio>=7)
10 {
11 cout<<”Promocionado”;
12 }
13 else
14 {
15 if((promedio>=4) && (promedio<7))
16 {
17 cout<<”Regular”;
18 }
19 else{
20 cout<<”No habilitado”;
21 }
22 }

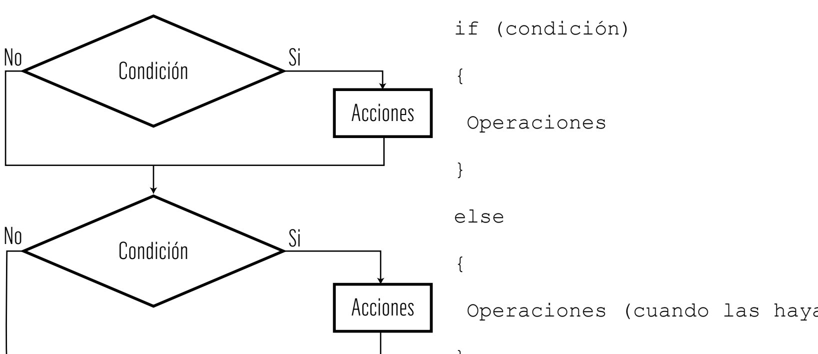
Código en Python
1 nota1=int(input(“Ingrese la primera nota”))
2 nota2=int(input(“Ingrese la segunda nota”))
3 nota3=int(input(“Ingrese la tercera nota”))
4 promedio=(nota1+nota2+nota3)/3
5 if promedio>=7:
6 print(“Promocionado”)
7 elif promedio>=4 && promedio<7:
8 print(“Regular”)
9 else:
10 print(“No habilitado”);

Código en Java
1 Scanner teclado=new Scanner(System.in);
2 double nota1,nota2,nota3;
3 System.out.print("Ingrese la primera nota");
4 nota1=teclado.next.int();
5 System.out.print("Ingrese la segunda nota");
6 nota2=teclado.next.int();
7 System.out.print("Ingrese la tercera nota");
8 nota3=teclado.next.int();
9 promedio=(nota1+nota2+nota3)/3
10 if (promedio>=7)
11 {
12 System.out.print("Promocionado");
13 }
14 else
15 {
16 if ((promedio>=4) && (promedio<7))
17 {
18 System.out.print("Regular");
19 }
20 else
21 {
22 System.out.print("No habilitado");
23 }
24 }

5.1.2.2 Estructura condicional secuencial

Se denomina estructura condicional secuencial cuando en un mismo diagrama de flujo se requieren varias estructuras y cada una va al terminar la otra. En la figura 20 se presenta el diagrama de flujo para este tipo de estructura condicional junto con su pseudocódigo.

Figura 20. Estructura condicional secuencial



Nota 2. IF anidados o secuenciales

Es común que se presenten estructuras condicionales anidadas aún más complejas, además es necesario tener presente que no siempre un mismo problema tiene estas tres formas de ser resuelto tal y como acaba de suceder. En la actividad 7 se proponen algunos ejercicios que puede revisar en la página web para conocer las soluciones.

Ejemplo 13.
Notas_condicionales_secuenciales

Este ejemplo presenta la solución del ejemplo 12 para la clasificación de un estudiante de acuerdo con sus notas, empleando estructuras condicionales secuenciales. El análisis de entradas y salidas ya se encuentra planteado en la tabla 23. La figura 21 muestra el diagrama de flujo en el que se emplean estructuras condicionales

secuenciales. Como se puede observar, una vez se evalúa una condición, se procede a la siguiente sin importar el resultado de la condición evaluada previamente. En la tabla 25 se encuentra la codificación en C++, Java y Python de este ejemplo.

Figura 21. Diagrama de flujo para el ejemplo notas_condicionales_secuenciales

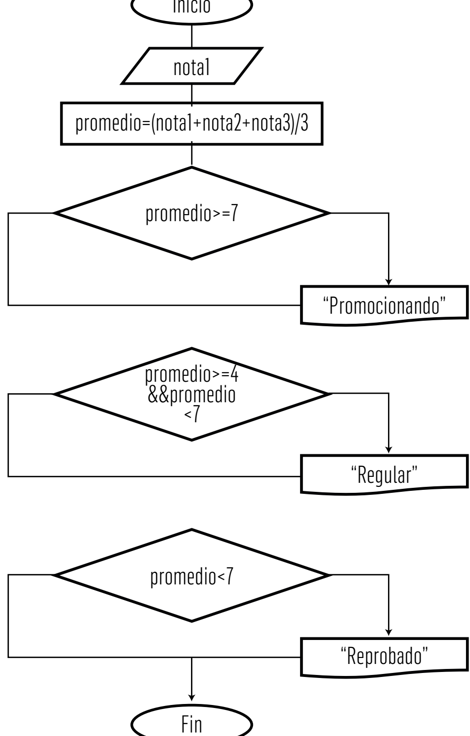


Tabla 25. Código en C++, Java y Python para el ejemplo notas_condicionales_secuenciales

Código en C++
1 double nota1,nota2,nota3;
2 cout<<"Ingrese la primera nota";
3 cin>>nota1;
4 cout<<"Ingrese la segunda nota";
5 cin>>nota2;
6 cout<<"Ingrese la tercera nota";
7 cin>>nota1;
8 promedio=(nota1+nota2+nota3)/3
9 if (promedio>=7)
10 {
11 cout<<"Promocionado";
12 }
13 if ((promedio>=4) && (promedio<7)) {
14 cout<<"Regular";
15 }
16 if (promedio>=4) {
17 cout<<"No habilitado";
18 }

Código en Python
1 nota1=int(input("Ingrese la primera nota"))
2 nota2=int(input("Ingrese la segunda nota"))
3 nota3=int(input("Ingrese la tercera nota"))
4 promedio=(nota1+nota2+nota3)/3
5 if promedio>=7:
6 print("Promocionado")
7 if promedio>=4 && promedio<7:
8 print("Regular")
9 if promedio<4:
10 print("No Habilitado")

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 double nota1,nota2,nota3;
3 System.out.print("Ingrese la primera nota");
4 nota1=teclado.next.int();
5 System.out.print("Ingrese la segunda nota");
6 nota2=teclado.next.int();
7 System.out.print("Ingrese la tercera nota");
8 nota3=teclado.next.int();
9 promedio=(nota1+nota2+nota3)/3
10 if (promedio>=7)
11 {
12 cout<<"Promocionado";
13 }
14 if ((promedio>=4) && (promedio<7))
15 {
16 cout<<"Regular";
17 }
18 if (promedio>=4)
19 {
20 cout<<"No habilitado";
21 }
```

Ejercicios

Ejercicio 9. Mayor_tres-números

A una tienda de mascotas llegan unas personas que desean comprar dos perros, pero preferirían que aquellos que les entregaran fueran los más jóvenes. Actualmente en la tienda tienen tres perros, así que solo tienen que hallar el mayor de ellos y descartarlo para

la venta. Construya un programa que reciba tres números enteros y calcule el mayor de ellos.

Ejercicio 10. Mayor_igual_dos_números

Recordemos un ejemplo anterior. En una oficina con más de 100 empleados desean saber por cada pareja posible cuál persona tiene un salario más alto que la otra. Se le pide diseñar un programa que pida por teclado dos números diferentes y determine cuál es el mayor entre los dos. El programa debe revisar la posibilidad de que sean iguales y, si es así, imprimir un mensaje notificándolo.

Ejercicio 11. Mayor_área_figuras-geométricas

El área de un triángulo rectángulo se calcula multiplicando la medida de su base por su altura y dividiendo el resultado entre 2. Desarrolle un programa que permita al usuario obtener el área de un triángulo rectángulo teniendo estas medidas. Implemente también el cálculo del área para un cuadrado a partir de la medida de uno de sus lados; y de un círculo a partir de su radio; además determine cuál figura tiene la mayor área (no es necesario validar si son iguales).

Ejercicio 12. Edad_anio_meses_días

En una base de datos de una empresa se tienen las fechas de nacimiento de cada uno de sus empleados. Se desea conocer la edad de cada uno de ellos en un momento específico. Construya un programa que permita determinar la edad en años, meses y días de una persona en una fecha ingresada por el usuario y teniendo la fecha de nacimiento (recibir por separado el año, el mes y el día de cada una de las fechas). Suponga que las fechas siempre son ingresadas correctamente.

Actividad 5.
Condicionales anidados y secuenciales

Objetivo general

Diseñar diagramas de flujo con estructuras condicionales anidadas y/o secuenciales con operadores lógicos para la combinación de condiciones a partir de requerimientos básicos de programación siguiendo las tres primeras fases: análisis del problema, diseño del algoritmo y prueba de escritorio.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento que exija el uso de la estructura de control condicional If anidada o secuencial.
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.

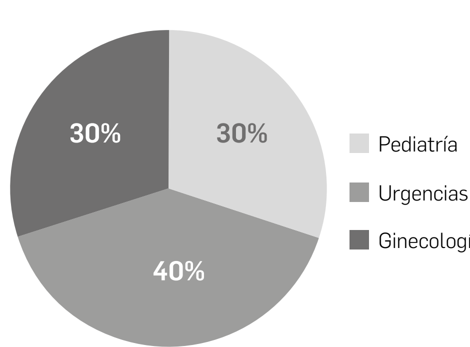
Vocabulario

- Requerimiento
- Estructura de control condicional If anidada y/o secuencial
- Operadores relacionales y lógicos

Ejercicios

1. El presupuesto de un hospital se distribuye como se muestra en la figura 22. Desarrolle un algoritmo que muestre cuánto le corresponde a cada sección, dado el presupuesto para el presente año.

Figura 22. Distribución del presupuesto del hospital



2. En una empresa se tienen los datos de las ganancias obtenidas en tres meses diferentes, los datos están expresados en millones de dólares. La empresa desea saber si durante esos tres meses experimentó un crecimiento en las ganancias o no. Desarrolle un algoritmo tal que, dados como datos de entrada tres números enteros, determine si los mismos están en orden creciente, decreciente o no muestran orden.

3. En un cierto país se pagan impuestos sobre cada artículo que se compre de acuerdo con la siguiente fórmula:

   $$precio_{final} = costo_{articulo} + impuesto$$

   El valor del impuesto varía así:

   a. Si el artículo tiene un costo hasta $20, entonces no se causan impuestos.
   b. Si el costo del artículo llega hasta $40 el impuesto corresponde al 10% del costo.
   c. Si el costo del artículo llega hasta $100 el impuesto corresponde al 20% del costo.
   d. Cualquier artículo cuyo costo supere los $100 pagará un impuesto del 30%.

Desarrolle un algoritmo que determine el precio a pagar por cualquier artículo de acuerdo con la legislación de este país.

1. Una empresa privada necesita que usted desarrolle una aplicación que capture un valor entre 1 y 12, y luego imprima a qué mes corresponde y cuántos días tiene dicho mes. Por ejemplo: si se captura el valor 3 debe responder “el mes corresponde a marzo y tiene 31 días”. A cualquier valor capturado por fuera de dicho rango debe responder como “valor inválido”.

2. La suma de los tres ángulos de un triángulo es de 180 grados. Desarrolle un algoritmo que lea valores correspondientes a tres ángulos y determine si ellos conforman o no un triángulo. Además, indique a qué tipo de triángulo corresponde:

   a. Rectángulo: si tiene un ángulo recto (90 grados)
   b. Obtusángulo: tiene un ángulo obtuso (mayor que 90 pero menor que 180 grados)
   c. Acutángulo: los tres ángulos son agudos (menores que 90 grados).

5.1.3 Estructura *switch case*

Con frecuencia es necesario que existan más de dos elecciones posibles en una condición. Por ejemplo, en unas elecciones en las que se puede escoger entre tres diferentes posibilidades o en diferentes operaciones que se pueden realizar según un número seleccionado. En la estructura *switch* se evaluará una expresión que podrá tomar ***n*** valores distintos y el flujo de datos tomará el camino que la selección indica.

En forma ocasional, un algoritmo tendrá una serie de decisiones en la que una variable o expresión se probará por separado contra cada uno de los valores constantes enteros que pueda asumir y se tomarán diferentes acciones. Para esta forma de toma de decisiones se proporciona una estructura de selección múltiple: *switch case*. La

estructura *switch* está formada de una serie de etiquetas *case*, y de un caso opcional *default*, como se presenta a continuación:

```
switch (variable){
case x:
operaciones1
break;
case y:
operaciones2
break;
default:
operacionesN
break;
}
```

La variable a evaluar en la sentencia *switch* debe ser de tipo *int* o *char*. La palabra reservada *switch* es seguida por el nombre de la variable entre paréntesis. Esto se conoce como la expresión de control. Después de esta expresión se abre una llave { para el inicio de los *cases.* El valor de esta expresión es comparado con cada una de las etiquetas *case.* Si ocurre una coincidencia, se ejecutará la sentencia correspondiente a dicho *case*, y de inmediato, mediante el enunciado *break*, se sale de la estructura *switch*. El enunciado *break* causa que el control del programa continúe con el primer enunciado que sigue después de la estructura *switch*.

Se utiliza el enunciado *break* porque de lo contrario los *cases* en un enunciado *switch* se ejecutarían juntos. Si en alguna parte de la estructura *break* no se utiliza este enunciado (*break*), entonces, cada vez que ocurre una coincidencia en la estructura se ejecutarían todos los enunciados de los *cases* restantes.

Si no existe coincidencia, el caso *default* es ejecutado y se imprime por lo general un mensaje de error. Después de terminar con la sentencia de la expresión *default* se cierra la llave } del *switch*. La expresión *default* es opcional. Cada *case* puede tener una o más sentencias. La estructura *switch* es diferente a todas las demás estructuras porque no se requieren llaves alrededor de varias sentencias dentro de un *case*. Se pueden utilizar varias etiquetas, lo que significa que el mismo conjunto de acciones ocurrirá para cualquiera de estos casos. Al igual que con la estructura *do while*, la estructura *switch case* solo está definida para los lenguajes de programación C++ y Java.

Ejemplo 14.
Cantidad_días_mes

Se desea saber la cantidad de días que hay en un mes específico del año. Para hacerlo, cada mes se representará con un número entre el 1 y el 12, correspondiente a su posición en el calendario. Desarrolle una aplicación que, según el número del mes ingresado por el usuario, imprima cuántos días tiene ese mes. Por ejemplo: si se captura el valor 3 debe responder "El mes tiene 31 días". Asuma que febrero tiene 28 días y que cualquier valor capturado se encuentra en el rango indicado. En la tabla 26 se presentan las entradas y salidas del algoritmo. Como se puede observar, para diferentes valores del mes se obtiene un resultado diferente. Por ejemplo, el mes 2 que es febrero tiene 28 días o el mes 11 que es noviembre tiene 31 días.

En la figura 23 se presenta el diagrama de flujo para el ejemplo 14. Como se puede observar, inicialmente se solicita el número del mes, el cual se almacena en la variable numero*Mes*. Luego, empleando un selector del tipo *switch case*, se establecen 3 caminos diferentes para encontrar el número de días del mes almacenados en la variable *dias*. El primer camino corresponde a los meses 1, 3, 5, 7, 8, 10, 12 que tienen 31 días. El segundo camino, se especifica únicamente

Tabla 26. Entradas y salidas para el ejemplo cantidad_días_mes

Entradas	Salidas
numeroMes	*Días*
4	30
1	31
2	28
11	31

al mes 2 que tiene 28 días. El tercer camino especifica los meses restantes que corresponden a aquellos con 30 días. Finalmente se imprime la cantidad de días que tiene el mes deseado para luego finalizar el algoritmo. La codificación en C++ y Java se presenta en la tabla 27.

Figura 23. Diagrama de flujo para el ejemplo cantidad_días_mes

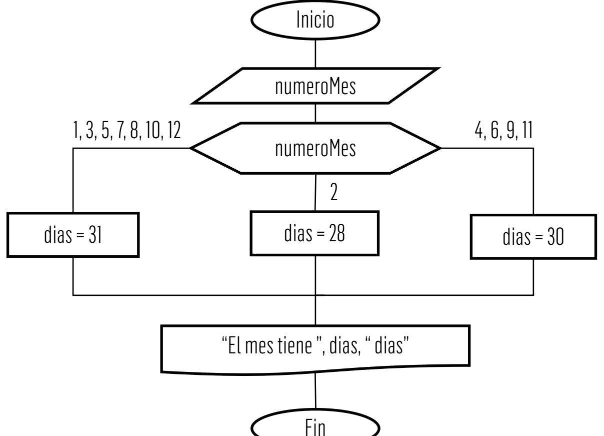


Tabla 27. Código en C++ y Java para el ejemplo cantidad_días_mes

Código en C++	Código en Java
1 int numeroMes, dias; 2 cout<<"Ingrese el numero del mes"; 3 cin>>numeroMes; 4 switch(numeroMes) 5 { 6 case 1: 7 case 3: 8 case 5: 9 case 8: 10 case 10: 11 case 12: 12 dias=31; 13 break; 14 case 2: 15 dias=28; 16 break; 17 default: //dias restantes 18 dias=30; 19 break; 20 } 21 cout<<"El mes tiene"<<-dias<<"dias";	1 Scanner teclado=new Scanner(System.in); 2 int numeroMes,dias; 3 System.out.print("Ingrese el numero del mes deseado"); 4 numeroMes=teclado.next.int(); 5 switch(numeroMes) 6 { 7 case 1: 8 case 3: 9 case 5: 10 case 8: 11 case 10: 12 case 12: 13 dias=31; 14 break; 15 case 2: 16 dias=28; 17 break; 18 default: //dias restantes 19 dias=30; 20 break; 21 } 22 System.out.print("El mes tiene"+dias+"dias");

Ejemplo 15.
Operaciones

Recuerde un ejemplo anterior. En matemáticas, las operaciones básicas que se pueden aplicar sobre un conjunto de números enteros son la suma (+), la resta (-), la multiplicación (*) y la división (%). Esta vez haga un programa que implemente todas las operaciones básicas y aplique una de ellas a un par de números. En la tabla 28 se presentan las entradas y salidas del algoritmo. En este caso la variable *opción*, servirá para seleccionar la operación deseada. En este ejemplo, si *opción* vale 1, corresponde a la suma de los dos números, si es 2 corresponde a la resta, 3 a la multiplicación y 4 a la división.

Tabla 28. Entradas y salidas para el ejemplo Operaciones

Entradas			Salidas
numero1	*numero2*	*Opcion*	*resultado*
2	2	4	1
43	11	1	54
5	15	2	-10
3	12	3	36

El diagrama de flujo del algoritmo se presenta en la figura 24. Como se puede ver, primero se piden las variables de entrada *numero1*, *numero2* y *opcion*. Posteriormente, se ingresa a la estructura *switch case*, la cual tiene cuatro caminos. Los tres primeros corresponden a las opciones de suma, resta y multiplicación que tienen los números 1, 2 y 3 respectivamente. La opción 4 representa la división. En esta última opción, se pone una limitación adicional, la cual verifica que la división no sea por cero. De ser así, imprimirá un mensaje de

error. De lo contrario realizará la división entre n1 y n2. El resultado de la operación seleccionada se almacena en la variable *resultado*. Finalmente, se imprime el resultado de la operación y se da fin al programa. La codificación del algoritmo se presenta en la tabla 29.

Figura 24. Diagrama de flujo para el ejemplo Operaciones

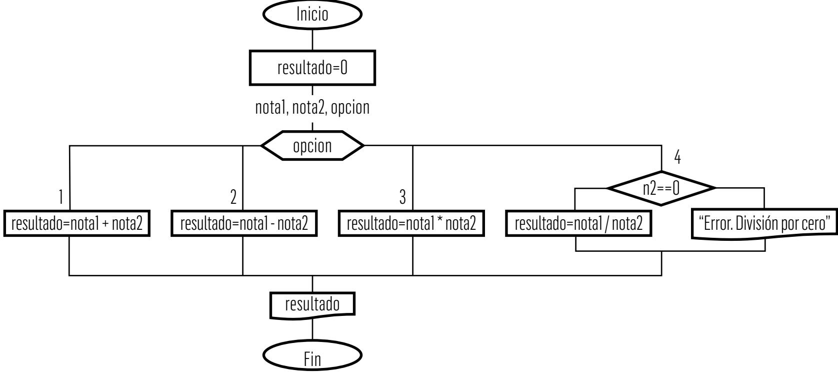


Tabla 29. Código en C++ y Java para el ejemplo Operaciones

Código en C++

```
1 int opcion;
2 double numero1,numero2,resultado=0;
3 cout<<"Ingrese dos numeros";
4 cin>>numero1;
5 cin>>numero2;
6 cout<<"1)suma\n2)resta\n
3)multiplicacion\n4)division\n";
7 cout<<"Ingrese la opcion:";
8 cin>>opcion;
9 switch(opcion){
10 case 1:
```

Continúa

Código en C++

```
11 resultado=numero1+numero2;
12 break;
13 case 2:
14 resultado=numero1-numero2;
15 break;
16 case 3:
17 resultado=numero1*numero2;
18 break;
19 case 1:
20 if (numero2==0){
21 cout<<"Error, division por cero"
22 }
23 else{
24 resultado=numero1/numero2;
25 }
26 break;
27 default:
28 cout<<"Operacion invalida";
29 break;
30 }
31 cout<<"El resultado es"<<resultado;
```

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int opcion;
3 double numero1,numero2,resultado=0;
4 System.out.print("Ingrese dos numeros");
5 numero1=teclado.next.double();
6 numero2=teclado.next.double();
7 System.out.print("1)suma\n2)resta\n
3)multiplicacion\n4)division\n");
8 System.out.print("Ingrese la opcion");
9 opcion=teclado.next.int();
10 switch(opcion){
```

Continúa

Código en Java

```
11 case 1:
12 resultado=numero1+numero2;
13 break;
14 case 2:
15 resultado=numero1-numero2;
16 break;
17 case 3:
18 resultado=numero1*numero2;
19 break;
20 case 1:
21 if (numero2==0){
22 cout<<"division por cero"}
23 else{
24 resultado=numero1/numero2;}
25 break;
26 default:
27 cout<<"Operacion invalida";
28 break; }
29 System.out.print("El resultado es"+resultado);
```

Ejercicios

Ejercicio 13. Admisiones

Un alumno que recién se ha graduado desea ingresar a una universidad que admite a sus estudiantes dependiendo de la nota promedio que haya obtenido en su último curso y de la profesión a la cual desee entrar. Desarrolle una aplicación en la cual el usuario ingrese la nota del estudiante y la profesión que desea elegir e imprima la palabra "Aceptado" en caso de que el promedio sea mayor a los valores presentados en la tabla 30.

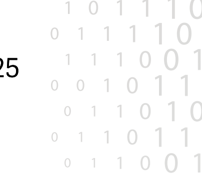


Tabla 30. Requisitos de admisión a una carrera

Carrera	Promedio
Ingenierías	mayor que 8
Comunicación Social	mayor que 6.5
Administración de Empresas	mayor que 7.5
Negocios Internacionales	mayor que 7

Ejercicio 14. Día_del_anio

Una empresa fabricante de software lo contrata a usted para diseñar una aplicación de un calendario. Uno de los requerimientos de la aplicación es que tenga un conteo de los días del año, por ejemplo, si es 1 de febrero la aplicación debe decir que es el día 32 del año. Diseñar un programa que le pida al usuario ingresar un día y un mes, ambos en números, y que calcule qué día del año es (tome el año como no bisiesto).

Ejercicio 15. Conversiones_masa

En una academia de física requieren de una aplicación que les permita convertir de una unidad a otra el peso de un objeto. Haga un programa que convierta un peso desde kilogramos a: hectogramos, decagramos, gramos, decigramos, centigramos y miligramos. Diseñe también un menú en consola que muestre cada una de las opciones disponibles.

Ejercicio 16. Elecciones

En un proceso electoral electrónico existen cinco opciones y cada elector debe oprimir la foto del candidato que se presenta en pan-

talla. Cada botón corresponde al número que se le asignó según sorteo. Usted debe diseñar un programa que permita establecer la cantidad de votos que lleva cada candidato después de que cada elector votó y que permita un proceso repetitivo para que puedan votar varios electores sin cerrar el programa. Para el conteo de los votos debe hacer uso de un *switch case*. Represente la acción de votar por ingresar el número del candidato.

Actividad 6.
Switch case

Objetivo general

Diseñar diagramas de flujo con la estructura *switch* a partir de requerimientos básicos de programación con actividades repetitivas que el aprendiz debe identificar siguiendo las fases de construcción de un programa.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento que exija el uso de la estructura de control *switch.*
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Estructura de control repetitiva *switch case*
- Actividades repetitivas

Ejercicios

1. Una empresa paga mensualmente a sus empleados un determinado valor en dólares según la categoría a la cual pertenezcan. Diseñe un programa que calcule la suma total pagada a un número indeterminado de empleados, y la cantidad de empleados que pertenecen a cada categoría. Las categorías se dividen de la siguiente forma. La categoría 1 paga 100 dólares semanales, la categoría 2 paga 130 dólares semanales. Para cualquier otra categoría paga 200 dólares semanales.

2. En una escuela desean llevar un conteo de los estudiantes según sus respectivas notas con el fin de conocer cuál es la proporción de estudiantes en comparación con su desempeño académico. Mediante el uso de la estructura *switch*, construya un programa que reciba la nota de un número indeterminado de estudiantes y al final muestre la cantidad de estudiantes clasificados según la nota en los siguientes grupos. El primer grupo está compuesto entre los estudiantes superiores a 0 y menores a 1. El segundo grupo los estudiantes iguales o superiores a 1 y menores a 2. El tercer grupo los estudiantes iguales o superiores a 2 y menores a 3. El cuarto grupo los estudiantes iguales o superiores a 3 y menores a 4. Finalmente, el quinto grupo abarca a los estudiantes iguales o superiores a 4.

3. A un puerto marítimo llegan distintas clases de embarcaciones que se identifican por su tamaño. Cada embarcación posee una plataforma distinta para el embarque y desembarque de mercancías o pasajeros, y para diferenciarlas se representa cada una con una letra diferente. Desarrolle un programa que reciba 5 tipos distintos de embarcaciones (A, B, C, D y E) y determine cuántas embarcaciones hay por cada tipo.

4. Realizar un programa que simule un cajero automático y permita a un usuario realizar un depósito, un retiro y consultar su saldo. Para el depósito, el usuario debe elegir entre: $50, $100, $200. Para el retiro, debe elegir entre: $50, $100, $500. El programa debe verificar que no se puede hacer un retiro superior al saldo del usuario.

5.2 Estructuras de programación repetitivas

Una de las propiedades más importantes de la computación está dada por la capacidad de una computadora de evaluar una gran cantidad de instrucciones repetitivas en periodos cortos de tiempo. Esto ha sido la base para los avances en áreas como la astronomía, el diseño mecánico, la industria automotriz, la biología y muchos otros campos con modelamientos matemáticos avanzados.

Sin embargo, muchos de los problemas de programación que se presentan en cada disciplina, requieren la repetición de un conjunto de instrucciones bajo un conjunto de condiciones establecidas.

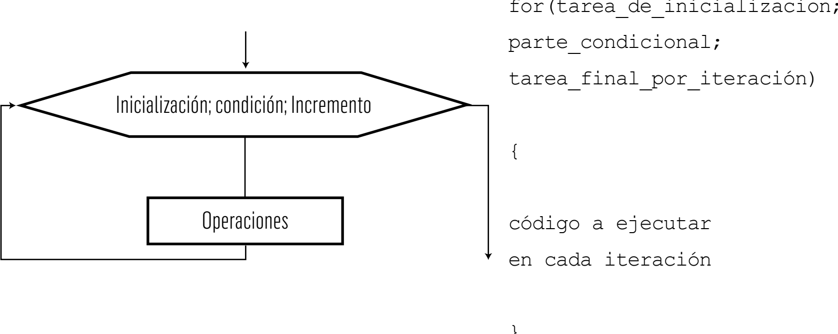
Para esto, la programación cuenta con un conjunto de estructuras o ciclos repetitivos, los cuales permiten reiterar un conjunto de instrucciones de forma rápida y efectiva. Generalmente, los lenguajes de programación estructurada cuentan con los ciclos repetitivos *for* y *while*. A continuación, se abordarán estos ciclos repetitivos, su uso, sus principales diferencias y cómo se codifican en los lenguajes de programación C++, Java y Python.

5.2.1 Ciclo *for*

En general, la estructura repetitiva *for* se usa en aquellas situaciones en las que es conocida la cantidad de veces que se desea ejecutar un bloque de instrucciones. Cada una de estas repeticiones se conoce como iteración. En la figura 25 se muestra la representación en diagrama de flujo del ciclo repetitivo *for* junto con su pseudocódigo.

Como se puede observar, el ciclo *for* tiene tres secciones en su forma más típica y básica. En la sección de "inicialización" se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. Este contador tendrá la función de realizar el conteo de repeticiones o iteraciones que tendrá el ciclo *for*. En la sec-

Figura 25. Representación del ciclo *for*



ción de "condición" se coloca la condición final de la variable contador. Esta condición deberá ser verdadera para que el ciclo continúe. Pero si la condición es falsa, el ciclo *for* se detendrá. En la sección de "incremento" se coloca una instrucción que permite modificar el valor de la variable que hace de contador, es decir, la tasa de aumento del contador. De esta forma se permite que alguna vez la condición sea falsa.

Cuando el ciclo *for* comienza, antes de dar la primera vuelta, la variable contador del ciclo toma el valor indicado en la sección de "inicialización". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo. Al finalizar el bloque de operaciones se ejecuta la instrucción de incremento del contador que se haya colocado en la sección "incremento". Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso. Cuando es conocida la cantidad de veces que se repite el bloque, es mucho más fácil emplear un ciclo *for.*

Ejemplo 16. Edades

En un estudio psicológico realizado a un grupo de personas, se tienen las edades de todos los integrantes y se desea saber cuál fue la mayor edad. Realice un programa que lea un número determinado

de edades y luego diga cuál es la mayor. En la tabla 31 se presenta el análisis de entradas y salidas del algoritmo para un conjunto de valores establecidos para la prueba de escritorio.

Tabla 31. Entradas y salidas para el ejemplo Edades

Entradas		Proceso	Salidas
cantidad	*edad*	*contador*	*mayor*
3		0	
	41	1	41
	15	2	41
	76	3	76

En la figura 26 se presenta el diagrama de flujo del algoritmo planteado. Como se puede observar, inicialmente se solicita al usuario la cantidad de edades que desea ingresar. Este resultado se almacena en la variable *cantidad*. De acuerdo con la tabla 31, el algoritmo requerirá el ingreso de tres edades. Posteriormente se inicializa la variable *mayor* que almacena el valor de la mayor edad ingresada. Esta inicializa en cero ya que no se ha ingresado ni evaluado ninguna edad. Posteriormente se inicializa el ciclo *for*, donde i corresponde a la variable contadora presente en la sección de inicialización que comienza en cero. Esta variable contadora será evaluada en la sección "Condición" y será verdadera mientras *cantidad* sea menor que tres. En la sección "incremento" se establece que *i* aumentará su valor en uno con cada iteración del ciclo *for*. Esto indica que el algoritmo tendrá tres repeticiones.

Una vez dentro del ciclo *for*, se solicita al usuario ingresar la primera edad, la cual se almacena en la variable *edad*. Luego se evalúa mediante una estructura condicional simple, si la edad es mayor que *mayor.* De ser verdad, *mayor* tomará el valor de la edad que se acaba de ingresar. Al finalizar la evaluación condicional, aumenta el valor de i de acuerdo con la sección de incremento y se repite el

proceso descrito anteriormente. Esto se realizará hasta que *i* sea mayor que 3. En ese momento, la condición del ciclo *for* será falsa y finalizará el programa imprimiendo el resultado de la mayor edad ingresada. La codificación del algoritmo en C++, Java y Python es presentada en la tabla 32.

Figura 26. Diagrama de flujo para el ejemplo Edades

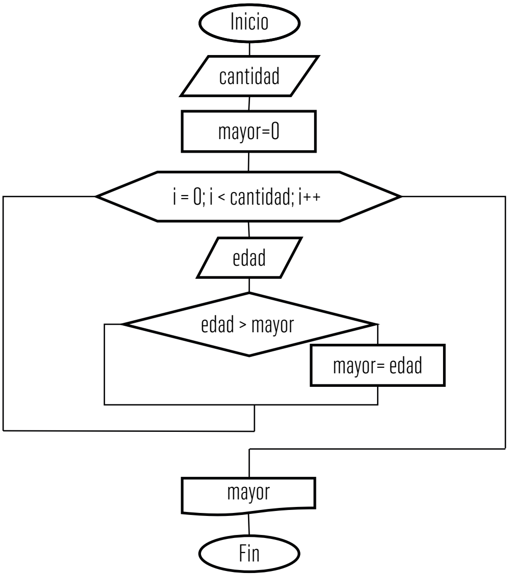
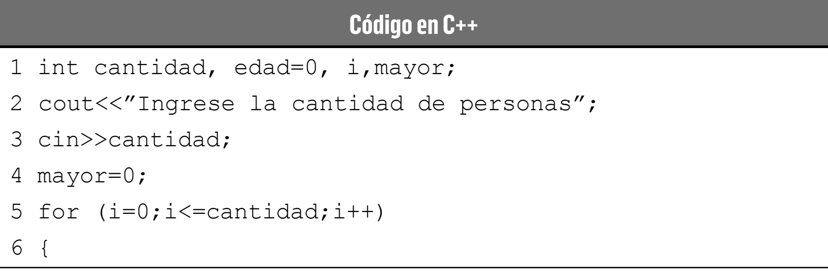


Tabla 32. Código en C++, Java y Python para el ejemplo Edades



Código en C++

```
1 int cantidad, edad=0, i,mayor;
2 cout<<"Ingrese la cantidad de personas";
3 cin>>cantidad;
4 mayor=0;
5 for (i=0;i<=cantidad;i++)
6 {
```

Continúa

Código en C++

```
7 cout<<"Ingrese la edad"<<i+1<<":";
8 cin>>edad;
9 if (edad>mayor) {
10 mayor=edad;
11 }
12 }
13 cout<<"La mayor edad es"<<mayor;
```

Código en Python

```
1 cantidad=int(input("Ingrese la cantidad de personas"))
2 mayor=0
3 for i in range(cantidad):
4 edad=int(input("Ingrese la edad"))
5 if edad>mayor:
6 mayor=edad
7 print("La mayor edad es ",mayor)
```

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int cantidad,edad,i,mayor=0;
3 System.out.print("Ingrese la cantidad de personas");
4 cantidad=teclado.next.int();
5 for (i=0;i<=cantidad;i++)
6 {
7 System.out.print("Ingrese la edad"+(i+1)+":");
8 edad=teclado.next.int();
9 if (edad>mayor)
10 {
11 mayor=edad;
12 }
13 }
14 cout<<"La mayor edad es"<<mayor;
```

Ejemplo 17.
Serie_números_for

En una institución preescolar se desea enseñar a los niños mediante herramientas tecnológicas. Para matemáticas, se ha propuesto conseguir un programa que muestre los números que existen hasta una determinada cifra.

Diseñe un programa que solicite un valor entero positivo y presente los números desde el uno hasta el valor ingresado de uno en uno. Por ejemplo, si es ingresado el 10 debe mostrar en la pantalla los números: 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10. En la tabla 33 se presenta el análisis de entradas y salidas del algoritmo para un conjunto de valores establecidos para la prueba de escritorio.

Tabla 33. Entradas y salidas para el ejemplo serie_números_for

Entrada	Salida
numero	valor
4	1
	2
	3
	4

En la figura 27 se presenta el diagrama de flujo del algoritmo planteado. Como se puede observar, inicialmente se solicita al usuario el número del cual se desean conocer todos sus antecesores y se almacena en *numero*. Luego se inicializa el ciclo *for*, estableciendo en la sección "Inicialización" a *valor* como variable contadora en cero. En la sección "condición", el límite para *valor* está dado por *n*. es decir el ciclo se repetirá hasta que *valor* sea mayor que *n*. En la sección "Incremento", *valor* aumentará una unidad con cada iteración del ciclo *for.*

Ya en el ciclo *for,* se imprime el valor actual de *valor*. Una vez lo imprime *valor* aumenta en uno como se especifica en la sección "incremento". Este proceso se repetirá, tal como se puede apreciar en la tabla 34 hasta que *valor* sea mayor que *numero*. La codificación del algoritmo en C++, Java y Python es presentada en la tabla 29.

Figura 27. Diagrama de flujo para el ejemplo serie_números_for

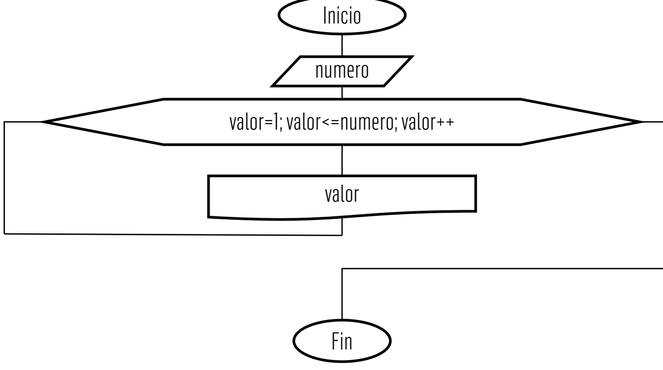


Tabla 34. Código en C++, Java y Python para el ejemplo serie_números_for

Código en C++

```
1 int i,numero, valor;
2 cout<<"Ingrese un numero";
3 cin>>numero;
4 for(valor=1;valor<=numero;valor++)
5 {
6 cout<<valor<<"\n";
7 }
```

Código en Python

```
1 numero=int(input("Ingrese el numero deseado"))
2 for valor in range(numero):
3 print(valor)
```

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int numero,valor;
3 System.out.print("Ingrese el numero deseado");
4 numero=teclado.next.int();
4 for(valor=1;valor<=numero;valor++)
5 {
6 System.out.print(valor+"\t");
7 }
```

Ejercicio 17. Fibonacci_for

Un hombre tiene una pareja de conejos y desea saber cuántas parejas poseerá por cada mes que pase. Se conoce que los conejos empiezan a reproducirse al segundo mes de haber nacido, y que una pareja da a luz a otra pareja en un mes. Esta relación es conocida en las matemáticas como *sucesión de Fibonacci*, que comienza con los números 1 y 1, y a partir de estos, cada término será la suma de los dos anteriores. Lo que significa que cada mes, las parejas de conejos que habrá serán: 1,1, 2, 3, 5, 8, 13, etc.

Diseñe un programa que permita al usuario conocer cuántas parejas de conejos hay por cada mes hasta llegar a 55 parejas.

Ejercicio 18. Impares_secuencia

Los números impares son aquellos que no son divisores de 2, es decir que si se divide uno de los números entre 2 al final el residuo será 1. Haga un programa que muestre los números impares existentes desde 50 hasta 1 (es decir, en forma descendente).

Ejercicio 19. Múltiplos

Los múltiplos de un número entero son aquellos resultantes de multiplicar aquel número por cada número entero positivo. Por ejemplo, los múltiplos de 3 serían: 3, 6, 9, 12, 15, 18, 21. Diseñe un programa que muestre al usuario los múltiplos de un número ingresado mientras que el múltiplo sea menor a 100.

Ejercicio 20. Suma_armónica

La suma armónica consiste en una serie de sumas que cumplen con la siguiente forma:

$$k=1+1/2+1/3+1/4+\cdots+1/n$$

Donde *n* es un número entero positivo que funciona como límite. Haga un programa que imprima la suma armónica con un límite *n* ingresado por el usuario. Tenga en cuenta que debido a que se usan fracciones, no puede usar variables enteras.

Ejercicio 21. Divisores

Diseñe un programa que genere los divisores de un número ingresado por el usuario. Tenga en cuenta que el divisor de un número es aquel entero que puede ser multiplicado por otro entero para producir este número. Por ejemplo, los divisores de 6 serían: 1, 2, 3 y 6.

Actividad 7.
Ciclo for

Objetivo general

Diseñar diagramas de flujo con la estructura *for* a partir de requerimientos básicos de programación con actividades repetitivas que el aprendiz debe identificar siguiendo las fases de construcción de un programa, puede incluir la estructura condicional If-Else.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento que requieran el uso de la estructura cíclica *for.*
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Estructura de control cíclica *for*
- Actividades repetitivas

Ejercicios

1. Una persona desea invertir dinero en un banco, el cual le otorga un 2% de interés mensual. ¿Cuál será la cantidad de tiempo para duplicar el dinero si todo el dinero lo reinvierte en el saldo?

2. Una empresa fabricadora de objetos de metal posee un lote con n piezas. Diseñar un programa que pida al usuario la cantidad de piezas y luego la longitud de cada una. Si la longitud se excede de 2 m o es inferior a 1 m, la pieza no será válida. Imprimir la cantidad de piezas validas del lote. Utilice una estructura *for* para realizar el proceso.

3. Un profesor de literatura hizo una evaluación para 10 estudiantes. Para aprobar el curso, el estudiante debe haber sacado un puntaje mayor o igual a 7. Escriba un programa que pida 10 notas y calcule cuántos estudiantes obtuvieron más de 7 (aprobaron) y

cuántos obtuvieron menos (reprobaron). Haga los cálculos usando una estructura *for.*

4. Una empresa paga sueldos a sus empleados que van desde $100 dólares a $500, dependiendo del trabajo al que estén asignados. Desarrollar un programa que lea el sueldo de n empleados (valide que esté entre $100 a $500) e imprima cuántos empleados cobran entre $100 y $300, y cuántos cobran más. Imprimir también el gasto total de la empresa en sueldos. Use una estructura *for* para el cálculo de los sueldos.

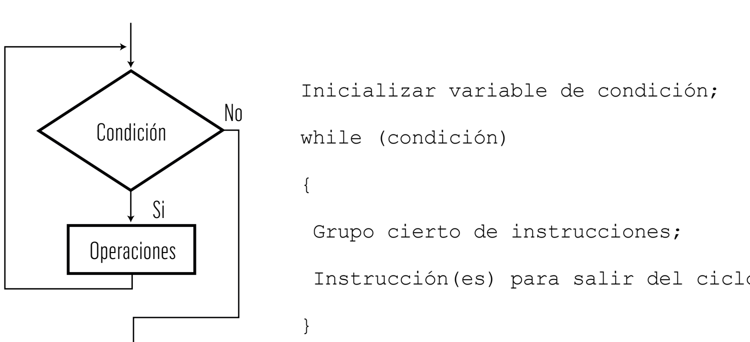
5.2.2 Estructura *while*

El ciclo *while* es una estructura repetitiva que permite ejecutar una instrucción o un conjunto de instrucciones varias veces mientras sea verdadera la pregunta inicial. Es decir, el ciclo *while* probará la condición antes de realizar cada repetición haciendo que se repitan o no las operaciones establecidas.

A diferencia del ciclo *for,* donde se conoce de antemano el número de iteraciones, el ciclo *while* ejecutará el cuerpo de instrucciones mientras una condición permanezca como verdadera. En el momento en que la condición se convierte en falsa, el ciclo *while* finalizará. Esto quiere decir que el número de iteraciones del programa dependerá del cumplimiento de la condición evaluada y no de un valor preestablecido.

En la figura 28 se muestra la representación en diagrama de flujo del ciclo *while* junto con su pseudocódigo. Como se puede observar, este ciclo utiliza un elemento de comparación como la estructura condicional IF. Sin embargo, una vez se ejecutan las operaciones, se observa una realimentación nuevamente hacia la condición del ciclo. Esto significa que mientras la condición en el elemento condicional sea verdadera, se repetirán las operaciones. Por lo tanto, es importante no confundir la representación gráfica de la estructura repetitiva *while* (mientras) con la estructura condicional IF (si).

Figura 28. Ciclo *while*



Para realizar la prueba de escritorio del ciclo *while*, en primer lugar se verifica la condición. Si esta resulta verdadera se ejecutan las operaciones indicadas por la rama del Verdadero. Cuando la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo, es decir, el bloque se repite MIENTRAS la condición sea Verdadera.

El flujo de la condición verdadera es graficado en la parte inferior de la condición y una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva. Tenga en cuenta que puede darse que la condición siempre retorna *verdadero*, en ese caso se estará en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación que hace que nunca finalice el programa.

Ejemplo 18.
Suma

Un micromercado requiere de un programa que calcule el costo total de la venta de un producto. Debido a que las ventas no son tan exigentes, solo requieren de un programa que calcule el total a partir de la cantidad de productos a llevar y el precio de cada uno de ellos. Realizar un programa que pida al usuario la cantidad de productos, que reciba el precio de cada uno y que luego imprima el costo total.

En la tabla 35 se establecen las entradas, salidas y variables de proceso del algoritmo. Inicialmente se establecen como entradas la cantidad de productos *cantidad* y el precio *precio* de cada uno de ellos. Como variables del proceso se tiene una variable contador *producto* que lleva el conteo de productos. Finalmente, como salida, la variable *valorTotal* lleva el valor acumulado de los productos.
El diagrama de flujo del algoritmo se presenta en la figura 29. Inicialmente, se solicita la cantidad de productos que se van a ingresar. Luego se inicializan las variables *producto* y *valorTotal* en cero.

Tabla 35. Entradas y salidas para el ejemplo Suma

Entradas		Proceso	Salidas
cantidad	*precio*	*productor*	*valorTotal*
3		0	0
	7.5	1	7.5
	2	2	9.5
	11.5	3	21

Posteriormente se ingresa al ciclo *while*, evaluando si el contador *producto* es menor que la cantidad de productos. De ser verdadero se pide el precio *precio* del primer producto y este se le adiciona a la variable *valorTotal*. Además, se aumenta en 1 el valor de *producto* y se reinicia el ciclo *while*. Se evalúa nuevamente la condición y continúa siendo verdadera ya que *producto* vale uno mientras que *cantidad* es tres. De esta forma se repite el proceso hasta que *producto* es igual a *cantidad.* En este punto se rompe el ciclo *while*, se imprime el resultado de la suma de productos y finaliza el programa. La implementación del algoritmo en C++, Java y Python se presenta en la tabla 36.

Figura 29. Diagrama de flujo para el ejemplo Suma

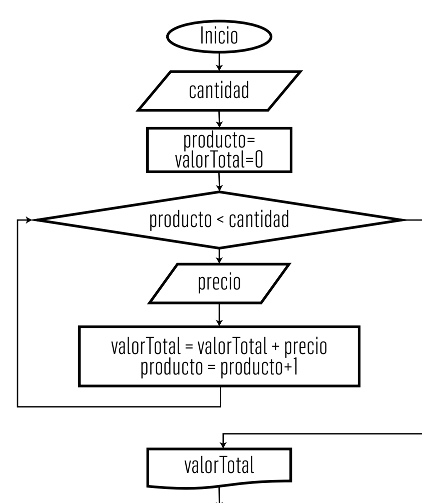


Tabla 36. Código en C++, Java y Python para el ejemplo Suma

Código en Python

```
1 cantidad=int(input("Ingrese la cantidad de numeros"))
2 producto=0; valorTotal=0
3 while producto<cantidad:
4 precio=int(input("Ingrese la cantidad de productos"))
5 valorTotal=valorTotal+precio
6 producto=producto+1
7 print("La suma es:",valorTotal)
```

Código en C++

```
1 int cantidad,producto;
2 double precio,valorTotal;
3 cout<<"Ingrese la cantidad de productos:";
4 cin>>cantidad;
```

Continú

Código en C++
5 producto=0;valorTotal=0;
6 while (producto<cantidad)
7 {
8 cout<<"Ingrese un numero";
9 cin>>precio;
10 valorTotal=valorTotal+precio;
11 producto=producto+1;
12 }
13 cout<<"La suma es:"<<valorTotal;

Código en Java
1 Scanner teclado=new Scanner(System.in);
2 int cantidad,producto;
3 double precio,valorTotal;
4 System.out.print("Ingrese la cantidad de productos");
5 cantidad=teclado.next.int();
6 precio=0;valorTotal=0;
7 while (producto<cantidad)
8 {
9 System.out.print("Ingrese un numero");
10 precio=teclado.next.int();
11 valorTotal=valorTotal+precio;
12 produco=producto+1;

Ejemplo 19.
Serie_números_while

Recordemos un ejemplo anterior. En una institución preescolar se desea enseñar a los niños mediante herramientas tecnológicas. Para matemáticas, se ha propuesto conseguir un programa que muestre los números que existen hasta determinada cifra. Diseñe un programa que solicite un valor entero positivo y presente los números desde el 1 hasta el valor ingresado de uno en uno. Por ejemplo, si es ingresado el 10 debe mostrar en la pantalla los números: 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10. Para este ejercicio debe utilizar la estructura *while*. En la tabla 28 se presentan las entradas y salidas del algoritmo, las cuales son las mismas para este ejemplo. En la figura 26 se presenta el diagrama de flujo empleando ciclo *while* para este ejemplo. A diferencia del ejemplo anterior, en este ejemplo se utiliza un ciclo *while* para evaluar la condición. Esto significa que para este ejemplo la cantidad de iteraciones serán la condición limitante para el ciclo repetitivo, las cuales aumentarán en cada iteración dentro del bloque de operaciones. En la tabla 32 se presenta la codificación en C++, Java y Python para este ejemplo.

Figura 30. Diagrama de flujo para el ejemplo serie_números_while

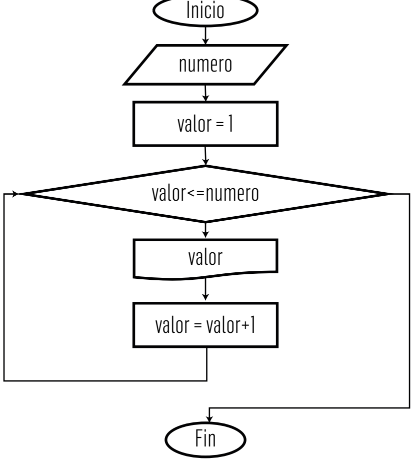
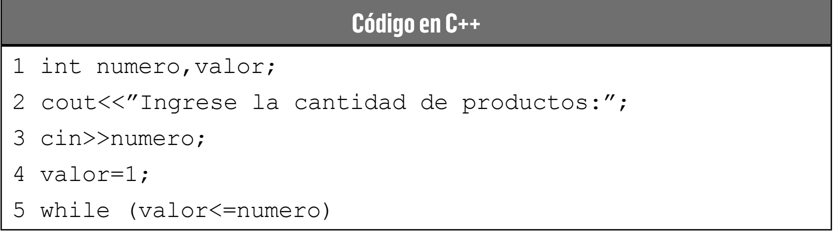


Tabla 37. Código en C++, Java y Python para el ejemplo serie_números_while



Código en C++
1 int numero,valor;
2 cout<<"Ingrese la cantidad de productos:";
3 cin>>numero;
4 valor=1;
5 while (valor<=numero)

Continúa

Código en C++

```
6 {
7 cout<<valor<<”\n”;
8 valor=valor+1;
9 }
```

Código en Python

```
1 numero=int(input(“Ingrese la cantidad de numeros”))
2 valor=1
3 while valor<=numero:
4 print(valor)
5 valor=valor+1
```

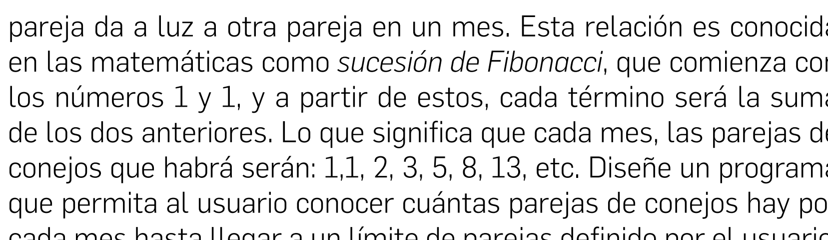
Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int numero,valor;
3 System.out.print(“Ingrese el numero deseado”);
4 numero=teclado.next.int();
5 valor=1;
6 while (valor<=numero)
7 {
8 System.out.print(valor+”\t”);
9 valor=valor+1;
10 }
```

Ejercicios

Ejercicio 22. Fibonacci_while

Un hombre tiene una pareja de conejos y desea saber cuántas parejas poseerá por cada mes que pase. Se conoce que los conejos empiezan a reproducirse al segundo mes de haber nacido, y que una



pareja da a luz a otra pareja en un mes. Esta relación es conocida en las matemáticas como *sucesión de Fibonacci*, que comienza con los números 1 y 1, y a partir de estos, cada término será la suma de los dos anteriores. Lo que significa que cada mes, las parejas de conejos que habrá serán: 1,1, 2, 3, 5, 8, 13, etc. Diseñe un programa que permita al usuario conocer cuántas parejas de conejos hay por cada mes hasta llegar a un límite de parejas definido por el usuario.

Ejercicio 23. Factorial_while

Cada temporada, un equipo cambia el tamaño de su plantilla debido a la transferencia de jugadores que hace con otros equipos. Dado que se aproxima la fecha para entregar la numeración de la plantilla, el técnico del equipo desea conocer de cuántas formas distintas pueden organizarse los jugadores. Esta forma de ordenar es conocida en las matemáticas como permutación, y para hallar el número total de permutaciones de un conjunto de elementos podemos usar el concepto de factorial. El factorial de un número consiste en el producto de todos los números enteros positivos menores o iguales a él. Por ejemplo, el factorial de 5 (representado como 5!) consiste en multiplicar 5 x 4 x 3 x 2 x 1, lo que daría como resultado 120. Haga un programa que permita calcular el factorial de un número asignado por el usuario usando la estructura *while.*

Ejercicio 24. Números_pares

Para un ejercicio matemático una persona desea conocer todos los números pares que hay hasta cierto límite. Haga un programa que muestre los números pares desde 1 hasta un límite ingresado por el usuario.

Ejercicio 25. División_entre_dos

Un número par es un número entero que es divisible por 2, es decir que si se divide entre 2 no puede haber ningún residuo. Teniendo un

número par ingresado por el usuario, decir cuántas veces se puede dividir este entre 2 hasta llegar a 1.

Ejercicio 26. Repetir_hasta_múltiplo_4

Desarrollar un programa que lea una serie de números y que se detenga únicamente hasta leer uno que sea múltiplo de 4. Al terminar de leerlos debe mostrar la suma de esos números, incluyendo el número con el cual terminó (es decir, el último en ser ingresado).

Ejercicio 27. Promedio_notas

En una institución académica se tienen las notas definitivas de cinco materias para 4 estudiantes. Se desea conocer el promedio que alcanzó cada uno de ellos. Haga un programa que, usando el ciclo *while*, lea cada una de las notas de los estudiantes e imprima el promedio de cada uno de ellos.

Actividad 8.
Ciclo while

Objetivo general

Diseñar diagramas de flujo con la estructura repetitiva *while* a partir de requerimientos básicos de programación con actividades repetitivas que el aprendiz debe identificar siguiendo las fases de construcción de un programa. Puede incluir la estructura condicional If-Else si así lo requiere el programa.

Objetivos

- Elaborar diagramas de flujo a partir de un requerimiento que requieran el uso de la estructura de control condicional *while.*

- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Estructura de control cíclica *while*
- Actividades repetitivas

Ejercicios

1. Una persona desea invertir dinero en un banco que le otorga un 2% de interés mensual. ¿Cuál será la cantidad de tiempo para duplicar el dinero si todo el dinero lo reinvierte en el saldo?

2. Un número primo es un número entero que posee únicamente como divisores a él mismo y al 1. Los números compuestos son lo contrario a los primos, son aquellos que poseen al menos un divisor diferente a él mismo y a 1. Diseñe un programa que reciba cualquier cantidad de números mayores que 1 (hasta que el usuario especifique que se detenga) y cada vez que lea el número imprima si es primo o compuesto.

3. Una empresa fabricadora de objetos de metal posee un lote con *n* piezas. Diseñar un programa que pida al usuario la cantidad de piezas y luego la longitud de cada una. Si la longitud se excede de 2 m o es inferior a 1 m, la pieza no será válida. Imprimir la cantidad de piezas validas del lote.

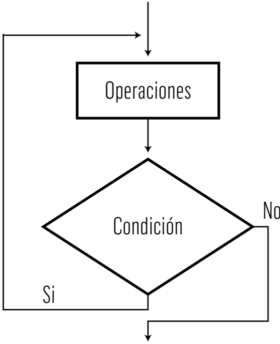
4. Un profesor de literatura hizo una evaluación para 10 estudiantes. Para aprobar el curso, el estudiante debe haber sacado un puntaje mayor o igual a 7. Escriba un programa que pida 10 notas y calcule cuántos estudiantes obtuvieron más de 7 (aprobaron) y cuántos obtuvieron menos (reprobaron).

5. Una empresa paga sueldos a sus empleados que van desde $100 dólares a $500, dependiendo del trabajo al que estén asignados. Desarrollar un programa que lea el sueldo de *n* empleados (valide que esté entre $100 a $500) e imprima cuántos empleados cobran entre $100 y $300, y cuántos cobran más. Imprimir también el gasto total de la empresa en sueldos.

5.2.3 Estructura *do-while*

Do-while es una estructura repetitiva que ejecuta al menos una vez su bloque repetitivo, a diferencia del *while* o del *for* que podían no ejecutar el bloque. En la figura 27 se muestra la representación en diagrama de flujo de la estructura *do-while*.

Figura 31. Estructura *do-while*



Como se puede observar, esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo. La condición de la estructura está abajo del bloque a repetir, a diferencia del *while* o del *for* que está en la parte superior. La siguiente figura muestra el diagrama de la estructura. El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera. Si la condición retorna Falso el ciclo se detiene. Todos los ciclos repiten por verdadero y cortan por falso. Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

Esta estructura repetitiva es exclusiva de los lenguajes de programación C y Java, por lo que, en la etapa de codificación, solo se presentaran estos dos lenguajes.

Ejemplo 20.
Serie_números_do_while

Recuerde los ejemplos 17 y 19. En una institución preescolar se desea enseñar a los niños mediante herramientas tecnológicas. Para matemáticas, se ha propuesto conseguir un programa que muestre los números que existen hasta determinada cifra. Diseñe un programa que solicite un valor entero positivo y presente los números desde el 1 hasta el valor ingresado de uno en uno. Por ejemplo, si es ingresado el 10 debe mostrar en la pantalla los números: 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10.

Las entradas y salidas del algoritmo se muestran en la tabla 38. Como se puede observar, los valores para la prueba de escritorio se establecen para n=4.

Tabla 38. Entradas y salidas para el ejemplo serie_números_do_while

Entradas	Salidas
numero	*valor*
4	1
	2
	3
	4

En la figura 32 se presenta el diagrama de flujo del algoritmo empleando la estructura *do-while.* A diferencia del ejemplo 23, la eva-

luación de la condición se encuentra al final del diagrama, después del bloque de operaciones. Es decir, en este caso al menos una vez ejecutará el bloque de operaciones. Para este algoritmo, primero se ingresa la cantidad de números mediante la variable *numero* y luego se establece *valor* en uno. Posteriormente se imprime el valor de *valor* para luego aumentarlo en uno. Después se evalúa si *valor* es menor que *numero*, de ser cierto, se realimenta el bucle, comenzando nuevamente en el punto donde se imprime *valor.* Una vez *valor* sea mayor que *numero* el programa finalizará. En la tabla 39 se presenta el Código en C++ y Java para el ejemplo 20. Cabe resaltar que solo se coloca el Código en C++ y Java porque esta estructura solo está definida para estos dos lenguajes de programación.

Figura 32. Diagrama de flujo para el ejemplo serie_números_do_while

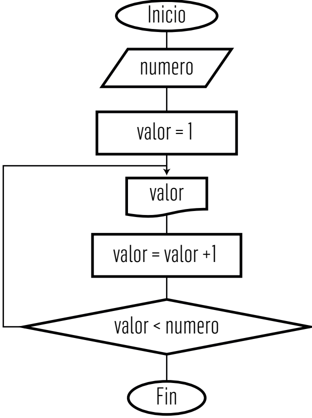


Tabla 39. Código en C++ y Java para el ejemplo serie_números_do_while

Código en C++

```
1 int valor=1;
2 cout<<"Ingrese un numero";
3 cin>>numero;
```

Continúa

Código en C++

```
4 do
5 {
6 cout<<numero<<"\n";
7 valor=numero+1;
8 }while(valor<numero);
```

Código en Java

```
1 Scanner teclado=new Scanner(System.in);
2 int valor=1;numero;
3 System.out.print("Ingrese el numero deseado");
4 numero=teclado.next.int();
5 valor=1;
6 do
7 {
8 System.out.print(valor+"\t");
9 valor=valor+1;
10 }while (valor<=numero)
```

Ejercicios

Ejercicio 28. Fibonacci_do_while

Un hombre tiene una pareja de conejos y desea saber cuántas parejas poseerá por cada mes que pase. Se conoce que los conejos empiezan a reproducirse al segundo mes de haber nacido, y que una pareja da a luz a otra pareja en un mes. Esta relación es conocida en las matemáticas como *sucesión de Fibonacci*, que comienza con los números 1 y 1, y a partir de estos, cada término será la suma de los dos anteriores. Lo que significa que cada mes, las parejas de conejos que habrá serán: 1,1, 2, 3, 5, 8, 13, etc. Diseñe un programa que permita al usuario conocer cuántas parejas de conejos hay por cada mes hasta llegar a 55 parejas.

Ejercicio 29. Voltaje_repetitivo_do_while

Recuerde el ejercicio 5**.** En un laboratorio de electrónica se necesita calcular el voltaje que atraviesa un circuito sobre el cual se está experimentado. Mediante algunos análisis que se le hacen, se ha establecido la resistencia y la intensidad que presenta la corriente que atraviesa el circuito. Teniendo la fórmula:

*voltaje=resistencia*intensidad*

construya un programa que, ingresados los valores de la resistencia y la intensidad, permita al usuario calcular el voltaje del circuito. En este caso se pide que el proceso se haga repetitivamente, permitiendo al usuario calcular cuantas veces quiera. El usuario debe poder decidir cuándo quiere que el proceso se detenga.

Ejercicio 30. Voltaje_repetitivo_promedio_do_while

Al ejercicio 29, agréguele la posibilidad de calcular el promedio de los voltajes recibidos en cada operación, es decir, el resultado se mostrará antes que el programa termine.

Ejercicio 31. Factorial_do_while

Cada temporada, un equipo cambia el tamaño de su plantilla debido a la transferencia de jugadores que hace con otros equipos. Debido a que se aproxima la fecha para entregar la numeración de la plantilla, el técnico del equipo desea conocer de cuántas formas distintas pueden organizarse los jugadores. Esta forma de ordenar es conocida en las matemáticas como permutación, y para hallar el número total de permutaciones de un conjunto de elementos podemos usar el concepto de factorial. El factorial de un número consiste en el producto de todos los números enteros positivos menores o iguales a él. Por ejemplo, el factorial de 5 (representado como 5!) consiste en multiplicar 5 x 4 x 3 x 2 x 1, lo que daría como resultado 120.

Haga un programa que permita calcular el factorial de un número asignado por el usuario usando la estructura *do-while*.

Ejercicio 32. Rebotes_pelota_do_while

Cuando se lanza un objeto al suelo el efecto de rebote lo impulsa nuevamente hacia el lado contrario. Sin embargo, la gravedad (además de factores como la masa y las propiedades del material del objeto) van reduciendo la fuerza del rebote hasta que el objeto queda prácticamente inmóvil en el suelo. Se conoce que, para una pelota de goma, el efecto de rebote ocasiona que cada vez que toca el suelo la pelota alcanza una altura que se va reduciendo el 10% por cada salto. Esto quiere decir que, si en un rebote la pelota alcanzó una altura de 75 cm, en el siguiente salto alcanzará 67.5 cm.

Diseñe un programa que permita determinar cuántos rebotes da la pelota si cae desde una altura ingresada por el usuario, hasta llegar a una altura menor a la quinta parte de la altura ingresada.

Ejercicio 33. Total_compra_do_while

En un supermercado cada cliente toma los productos que desea comprar y luego son registrados por un usuario en el computador, en donde reporta la cantidad de cada producto y el valor unitario. Haga un programa para calcular el total de una compra con un número indeterminado de ítems.

Actividad 9.
Ciclo do-while

Objetivo general

Diseñar diagramas de flujo con la estructura repetitiva *do-while* a partir de requerimientos básicos de programación con actividades

repetitivas que el aprendiz debe identificar siguiendo las fases de construcción de un programa. Puede incluir la estructura condicional If-Else si así lo requiere el ejercicio.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento que exija el uso de la estructura de control condicional *do-while.*
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Estructura de control cíclica *do-while*
- Actividades repetitivas

Ejercicios

1. En una institución donde se enseña matemáticas, es requerido un programa que les permita calcular el promedio de los números pares y de los impares de una lista de números. Desarrollar un programa que, dados unos números ingresados por el usuario, determine el promedio de pares, de impares y la cantidad que se ingresó de cada uno.

2. En un videojuego, la misión del jugador es llevar una nave por una tormenta de asteroides recibiendo el mínimo daño. Por cada impacto el jugador recibe un punto negativo, y si llega a alcanzar 1000 puntos es descalificado automáticamente. Después de que todos los jugadores han terminado, aquel con menos puntos negativos es el ganador. Desarrolle un algoritmo tal que de un listado de números (menores que 1000) ingresados determine cuál de ellos es el menor.

3. En una universidad, cada semestre los alumnos matriculan la cantidad de materias que ellos deseen, esto con el fin de que ellos puedan ajustar los horarios de estudio con los de trabajo. Al final del semestre, los promedios entre las notas de cada materia son calculados y aquellos con un promedio menor a 3 son citados a decanatura. Desarrolle un programa que reciba una cantidad de estudiantes y una cantidad de notas especificadas por el usuario, y que luego calcule el promedio de cada estudiante.

4. Un micromercado requiere imprimir la factura de cada venta. Desarrollar un programa que reciba la cantidad, el valor (sin impuesto) y el impuesto de cada artículo que se vende y al final imprima el subtotal (sin impuestos), la suma de los impuestos y el valor total a pagar (es decir, con impuestos).

5. Elabore un programa que imprima los múltiplos de un número ingresado por el usuario. El programa debe dejar de imprimir desde un límite, también ingresado por el usuario.

5.3 Arreglos

En programación un arreglo (llamado en inglés *array*) es un conjunto finito y ordenado de elementos homogéneos en una zona de almacenamiento contiguo, que contiene una serie de elementos del mismo tipo. Desde el punto de vista lógico se puede ver como un conjunto de elementos en una fila (o filas y columnas, si tuviera dos dimensiones).

Cuando se dice que un arreglo es ordenado, esto significa que todos los elementos pueden ser ubicados con subíndices, ya que se almacenan en la memoria central del computador en un orden adyacente de posiciones de memoria sucesivas. Cada elemento de un vector se puede procesar como si fuera una variable simple que ocupa una posición de memoria dada, de manera tal que cada elemento del vector es accesible directamente. Por su parte, la homogeneidad implica que todos los elementos del arreglo son datos del mismo tipo.

5.3.1 Tipos de arreglos por dimensiones

Los arreglos pueden ser:

- Arreglos unidimensionales (vectores): es un tipo de datos estructurado que está formado de una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales.
- Arreglos bidimensionales (matrices, tablas): es un tipo de dato estructurado, finito ordenado y homogéneo. El acceso a ellos también es en forma directa por medio de un par de índices. Los arreglos bidimensionales se usan para representar datos que pueden verse como una tabla con filas y columnas.
- Arreglos multidimensionales (tres dimensiones o más): también es un tipo de dato estructurado que está compuesto por n dimensiones. Para hacer referencia a cada componente del arreglo es necesario utilizar n índices, uno para cada dimensión.

5.3.2 Partes de un arreglo

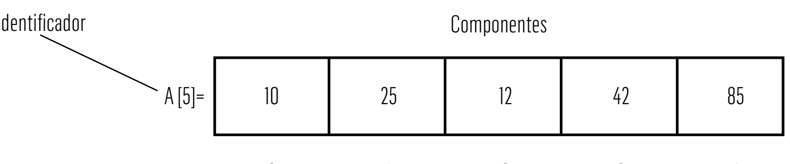
Un arreglo, en general, se compone de los siguientes elementos:

- Identificador: es el nombre que le asigna el usuario al arreglo.
- Los componentes: hacen referencia a los elementos que forman el arreglo, es decir, a los valores que se almacenan en cada una de las casillas del mismo.
- Los índices: permiten hacer referencia a los componentes del arreglo en forma individual, especifican cuántos elementos tendrá el arreglo y además, de qué modo podrán ser accedidos esos componentes.

En el caso de un vector o arreglo unidimensional, este consta de n elementos que pueden representarse en una sola dimensión gráfica como se muestra en la figura 33. Se puede observar que el vector tiene por identificador el nombre A y cuenta con 5 posiciones o componentes, en las cuales se almacenan los elementos 10, 25,12, 42 y

85. Además, cada uno de estos componentes cuenta con un índice que permite acceder a cada uno de los elementos que conforman el vector. Es importante resaltar que en la mayoría de los lenguajes de programación los índices de un arreglo comienzan desde el número cero, como es el caso del vector A. De esta forma, la sintaxis para acceder a cada elemento del vector será de la forma A[n], donde A corresponde al nombre del arreglo y n al índice del elemento a acceder. Por ejemplo, el índice 1 de A, se representa como A[1]=10 o el elemento A[3]=42.

Figura 33. Representación de un arreglo unidimensional o vector

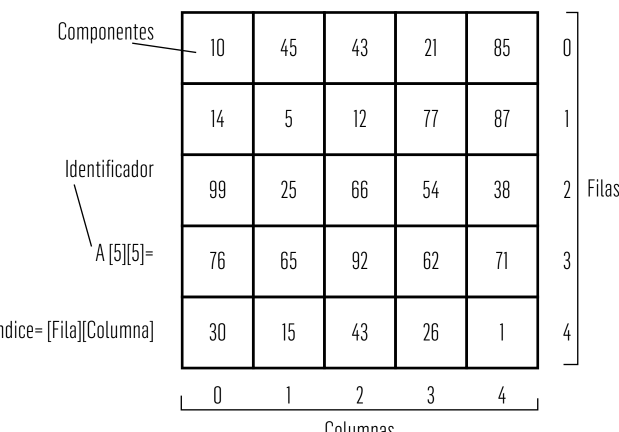


La figura 34 muestra la representación de un arreglo bidimensional o matriz. Se puede observar que el arreglo bidimensional cuenta, como su nombre lo dice, con dos dimensiones que se conocen como filas (horizontales) y columnas (verticales), lo que arroja una representación gráfica similar a una tabla o cuadrícula.

Para acceder a cada elemento de la matriz, se requiere un índice al igual que los vectores. Sin embargo, ya que la matriz tiene dos dimensiones el índice requerirá de la fila y la columna para identificar un elemento. Además, el identificador de la matriz será de la forma A[5][5], el cual indica el nombre del arreglo junto con su cantidad de filas y columnas respectivamente.

Por lo tanto, el componente A[3][2] corresponderá al número 25 mientras que el componente A[2][4] será el número 87. Al igual que con los vectores, el conteo de filas y columnas comenzará en cero.

Figura 34. Representación de un arreglo bidimensional o matriz



5.3.3 Tipos de arreglos por declaración

Los arreglos son declarados por el programador directamente en el código y pueden ser estáticos o dinámicos. Los arreglos declarados como estáticos son aquellos en los que el número de componentes se define al momento de su creación y permanece constante durante la ejecución del programa. Por otra parte, en los arreglos declarados como dinámicos el número de componentes puede ser cambiado durante la ejecución del programa, es decir, la memoria es asignada dinámicamente durante la ejecución del código. Este tipo de arreglos presenta algunos beneficios, como no necesitar una longitud constante en su declaración, evitando la existencia de espacios vacíos en la memoria.

Ejemplo 21.
Separar_cadena

Construya un programa que reciba como entrada una cadena de texto y como salida imprima la letra por línea de la cadena invertida. Por ejemplo, si se ingresa "Adiós", la salida sería la palabra "sóidA".

En la tabla 40 se presenta el análisis del problema. En este caso se toma como entrada una cadena de caracteres de 5 posiciones, las cuales corresponden a un arreglo unidimensional o vector. Los valores establecidos para la prueba de escritorio son para la cadena de caracteres "Adiós".

Tabla 40. Entradas y salidas del algoritmo para el ejemplo separar_cadena

Entradas	Proceso	Salidas
mensaje		
Adiós	4	"s"
	3	"ó"
	2	"i"
	1	"d"
	0	"A"

La figura 35 presenta el diagrama de flujo del algoritmo. Como se puede observar, la primera acción del programa solicita la cadena de caracteres para almacenarla en la variable *mensaje*. Posteriormente, emplea un ciclo *for* descendiente, cuya condición de inicialización corresponde a la longitud de la cadena de caracteres. La condición de parada se da cuando el contador *i* llegue a cero. Finalmente, la tasa de decremento es de una unidad. De esta forma en cada iteración del ciclo *for* se accede a cada componente del arreglo, desde el último elemento hasta el primero y se muestra en pantalla dicho caracter. En la tabla 41 se presentan los códigos en C++, Java y Python para este ejemplo.

Figura 35. Diagrama de flujo para el ejemplo separar_cadena

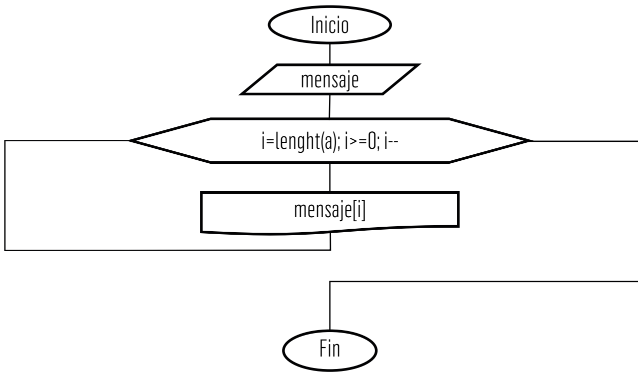


Tabla 41. Código en C++, Java y Python para el ejemplo separar_cadena

Código en C++

```
1 String mensaje;
2 cout<<"Ingrese una palabra: ";
3 cin>>mensaje;
4 for (int i = mensaje.length(); i >= 0; i--)
5 {
6 cout<<mensaje[i]<<"\n";
7 }
```

Código en Python

```
1 mensaje=input("Ingrese una palabra: ")
2 for i in range(len(mensaje), 0, -1):
3 print(mensaje[i - 1])
```

Código en Java

```
1 Scanner teclado = new Scanner(System.in);
2 String mensaje;
3 System.out.print("Ingrese una palabra: ");
```

Continúa

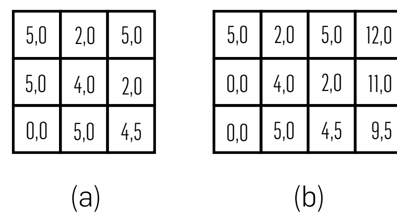
Código en Java

```
4 mensaje = teclado.next();
5 for (int i = mensaje.length(); i >= 0; i--)
6 {
7 System.out.print(mensaje[i] + "\n");
8 }
```

Ejemplo 22. Notas_arreglos

Una institución académica requiere de un programa que almacene las notas de sus estudiantes y determine la suma total de sus calificaciones individuales. Construya un programa que permita ingresar n notas para m estudiantes. Posterior a ello, el programa debe indicar la suma en la última columna. En la figura 36 se muestra un ejemplo de una matriz de calificaciones antes y después de la suma.

Figura 36. Matriz de calificaciones a) inicial y b) después de la sumatoria



(a)

5,0	2,0	5,0
5,0	4,0	2,0
0,0	5,0	4,5

(b)

5,0	2,0	5,0	12,0
0,0	4,0	2,0	11,0
0,0	5,0	4,5	9,5

En la figura 37 se presenta el diagrama de flujo para el ejemplo 22. Inicialmente se solicita la cantidad de estudiantes y de notas. De esta forma se define un conjunto de ciclos *for* anidados los cuales asignarán las notas en cada una de las posiciones de la matriz *calificaciones*. Finalmente, se imprimirá la columna correspondiente a la sumatoria de las notas de cada estudiante. En la tabla 42 se presenta el Código en C++, Java y Python para el algoritmo planteado.

Figura 37. Diagrama de flujo para el ejemplo notas_arreglos

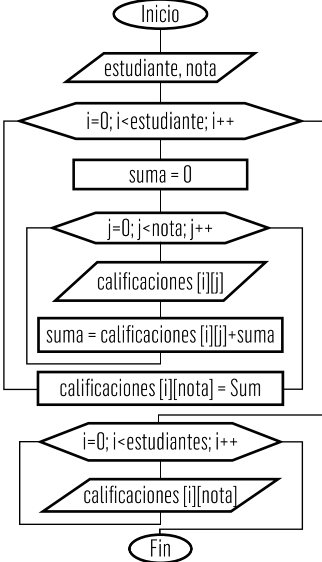


Tabla 42. Código en C++, Java y Python para el ejemplo notas_arreglos

Código en Python

```
1 estudiante= int(input("Ingrese la cantidad de estudian-
tes: "))
2 nota = int(input("Ingrese la cantidad de notas: "))
3 calificaciones = []
4 for i in range(estudiante):
5 calificaciones.append([])
6 suma = 0
7 print("Ingrese las notas: ")
8 for j in range(Not):
9 calificaciones[i].append(float())
10 suma = suma + calificaciones[i][j]
11 calificaciones[i].append(Sum)
12 for i in range(estudiante):
13 print(calificaciones[i][nota])
```


Código en C++

```
1 int estudiante, i, j, nota;
2 double Sum;
3 cout<<"Ingrese la cantidad de estudiantes: ";
4 cin>>estudiante;
5 cout<<"Ingrese la cantidad de notas: ";
6 cin>>nota;
7 double calificaciones[estudiante][nota];
8 for (i = 0; i < estudiante; i++)
9 {
10     suma = 0;
11     cout<<"Ingrese las notas:\n";
12     for (j = 0; j < nota; j++)
13     {
14            cin>>calificaciones[i][j];
15            suma= calificaciones[i][j]+suma;
16     }
17     calificaciones[i][nota] = suma;
18 }
19 for (i = 0; i < estudiante; i++)
20 {
21 cout<<calificaciones[i][nota];
22 }
```

Código en Java

```
1 Scanner teclado = new Scanner(System.in);
2 int estudiante, i, j, nota;
3 double suma;
4 System.out.print("Ingrese la cantidad de estudiantes:
");
5 estudiante = teclado.nextInt();
6 System.out.print("Ingrese la cantidad de notas: ");
7 nota = teclado.nextDouble();
8 double[][] calificaciones[estudiante][nota];
```

Continúa

Código en Java

```
9 for (i = 0; i < estudiante; i++)
10 {
11     suma = 0;
12     System.out.print(”Ingrese las notas:\n”);
13     for (j = 0; j < nota; j++)
14     {
15           calificaciones[i][j] = teclado.nextDouble();
16           suma= calificaciones[i][j]+suma;
17     }
18     calificaciones[i][nota] = suma;
19 }
20 for (i = 0; i < estudiante; i++)
21 {
22     System.out.print(calificaciones[i][nota]);
23 }
```

Ejemplo 23. Ordenamiento_burbuja

El ordenamiento de burbuja es un sencillo algoritmo que funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente e intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma en la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Debido a que solo usa comparaciones para operar elementos, se le considera un algoritmo de comparación y es el más sencillo de implementar.

Conociendo esto, una empresa desea implementar el método burbuja para ordenar una cantidad determinada de números enteros *cantidad* que se ingresan al sistema. Construya un programa

que permita hacer esta operación a los valores ingresados por el usuario. En la figura 38 se presenta el diagrama de flujo para el método de la burbuja aplicado a un vector o arreglo unidimensional llamado *numeros*.

Como primer paso, el algoritmo imprime el vector desordenado. Después, empleando dos ciclos *for* anidados, se realiza el ordenamiento del arreglo. Para esto el primer ciclo *for* con variable contadora *i* fija el elemento base para realizar la comparación. Posteriormente empleando el segundo ciclo *for* con variable contadora *j*, se recorre el resto del arreglo. De esta forma se realiza la comparación entre los elementos y empleando una variable temporal *Temporal* se cambian de posición los elementos en el vector. Finalmente, una vez se completa el ciclo *for* anidado, el algoritmo entrega el arreglo ordenado. El código del método de la burbuja en C++, Java y Python se presenta en la tabla 43.

Figura 38. Diagrama de flujo para el ejemplo ordenamiento_burbuja

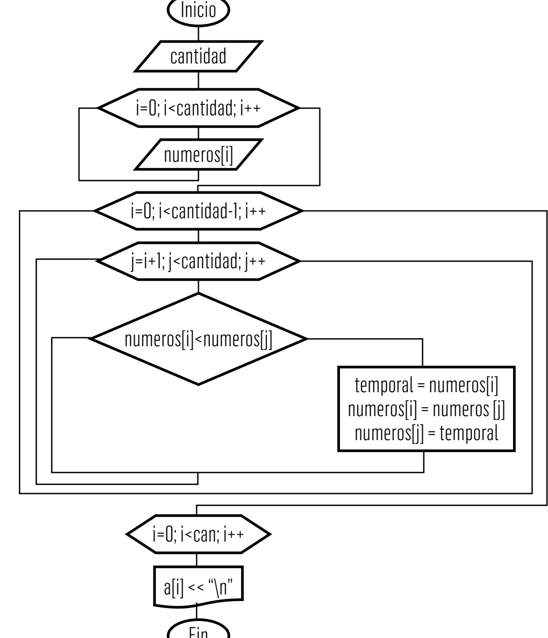


Tabla 43. Código en C++, Java y Python para el ejemplo ordenamiento_burbuja

Código en C++

```
1 int cantidad, i, j, temporal;
2 cout<<"Ingrese la cantidad de números: ";
3 cin>>cantidad;
4 int numeros[cantidad];
5 for (i = 0; i < cantidad; i++){
6 cout<<"Ingrese un número: ";
7 cin>>numeros[i];
8 }
9 for (i = 0; i < cantidad; i++){
10 for (j = i + 1; j < cantidad; j++) {
11 if (numeros[i] < numeros[j]){
12 temporal = numeros[i];
13 numeros[i] = numeros[j];
14 numeros[j] = temporal;
15 }
16 }
17 }
18 for (i = 0; i < cantidad; i++){
19 cout<<numeros[i]<<"\n";
20 }
```

Código en Python

```
1 cantidad = int(input("Ingrese la cantidad de números: "))
2 numeros = []
3 for i in range(cantidad):
4 numeros.append(int(input("Ingrese un número:")))
5 for i in range(cantidad):
6 for j in range(i + 1, cantidad):
7 if numeros[i] < numeros[j]:
8 temporal = numeros[i]
9 numeros[i] = numeros[j]
```

Continúa

Código en Python

```
10 numeros[j] = temporal
11 for i in range(cantidad):
12 print(numeros[i])
```

Código en Java

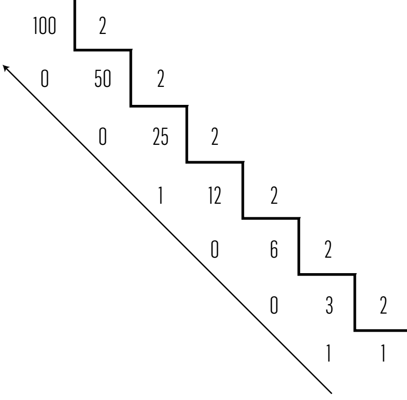
```
1 Scanner teclado = new Scanner(System.in);
2 int cantidad, i, j, temporal;
3 System.out.print("Ingrese la cantidad de números: ");
4 cantidad = teclado.nextInt();
5 int[] numeros[cantidad];
6 for (i = 0; i < cantidad; i++){
7 System.out.print("Ingrese un número: ");
8 numeros[i] = teclado.nextInt();
9 }
10 for (i = 0; i < cantidad; i++){
11 for (j = i + 1; j < cantidad; j++) {
12 if (numeros[i] < numeros[j]){
13 temporal = numeros[i];
14 numeros[i] = numeros[j];
15 numeros[j] = temporal;
16 }
17 }
18 }
19 for (i = 0; i < cantidad; i++){
20 System.out.print(numeros[i] + "\n");
21 }
```

Ejemplo 24. Decimal_binario

Un número binario es una secuencia conformada por unos (1) o ceros (0) que representan un número decimal y cuya aplicación es de gran importancia en las ciencias de la computación y la electrónica.

Para convertir un número de decimal a binario, el número debe dividirse entre 2 hasta que el dividendo sea menor que el divisor (2). El número binario resultante se obtiene cogiendo el resultado de la última división (con la que se finalizó) y añadirle los dígitos de cada uno de los residuos del resto de las divisiones en orden invertido. Un ejemplo de convertir el número decimal 100 a número binario puede verse en la figura 39. Como se puede apreciar, el resultado en binario sería **1100100**. Desarrolle un programa que reciba un número positivo en valor decimal e imprima el resultado de convertirlo a base binaria.

Figura 39. Conversión de decimal_binario



En la tabla 44 se presentan las entradas y salidas del algoritmo. En este caso, el valor asignado para la prueba de escritorio corresponde al número 7, el cual será convertido a su equivalente binario, almacenando en un vector los residuos de la división en base dos del número en base decimal.

El diagrama de flujo del algoritmo se presenta en la figura 40. Como se muestra en el diagrama, el primer paso es el ingreso del número decimal que se desea convertir a binario. Posteriormente, empleando un ciclo *do while* se realiza la división sucesiva en base dos del número a convertir. Los residuos son almacenados en el vector b[i], el cual incrementará hasta que el cociente de la división sea cero.

Posteriormente, empleando un ciclo *for* descendente, se imprimen los elementos del vector en orden descendente que corresponden al equivalente binario del número *n*. El código en C++, Java y Python se muestra en la tabla 45.

Tabla 44. Entradas y salidas para el ejemplo decimal_binario

Entradas	Proceso		
decimal	binario[ ]	i	j
7	-	0	-
3	1	1	-
1	1	2	-
0	1	3	-
0	1	3	2
0	1	3	1
0	1	3	0

Figura 40. Diagrama de flujo para el ejemplo decimal_binario

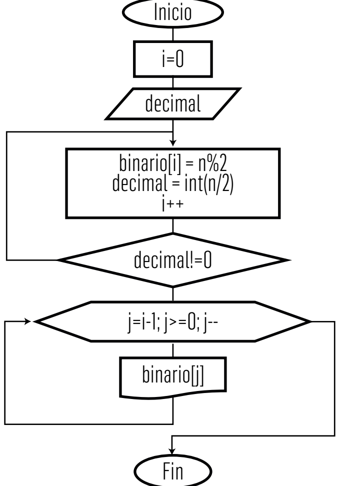


Tabla 45. Código en C++, Java y Python para el ejemplo decimal_binario

Código en C++

```
1 int i = 0, decimal, binario[99];
2 cout<<"Ingrese el número: ";
3 cin>>decimal;
4 do{
5 binario[i] = n % 2;
6 i++;
7 decimal = int( n / 2);
8 } while (decimal != 0);
9 for (int j = i - 1; j >= 0; j--){
10 cout<<binario[j];
11 }
```

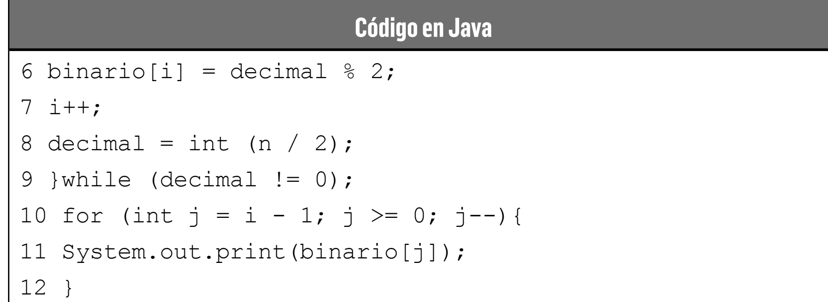
Código en Python

```
1 binario=[]
2 i=0
3 for i in range(99):
4 binario.append(0)
5 decimal=int(input("Ingrese el numero"))
6 while(decimal!=0):
7 binario[i] = decimal % 2;
8 i++;
9 decimal = int (n / 2);
10 for j in range(i-1,0,-1):
11 print(binario[i])
```

Código en Java

```
1 Scanner teclado = new Scanner(System.in);
2 int i = 0, decimal, binario[99];
3 System.out.print("Ingrese el número: ");
4 decimal = teclado.nextInt();
5 do{
```

Continúa



Código en Java

```
6 binario[i] = decimal % 2;
7 i++;
8 decimal = int (n / 2);
9 }while (decimal != 0);
10 for (int j = i - 1; j >= 0; j--){
11 System.out.print(binario[j]);
12 }
```

Ejercicios

Ejercicio 34. Separar_pares_impares

En un proceso industrial son generados productos para dos empresas distintas. Para poderlos producir secuencialmente para ambas empresas, los productos se etiquetan con números: a la primera empresa le corresponden los productos con número impar y a la segunda los productos con número par. Construya un programa que vaya recibiendo un listado de números de los productos y los separe en pares e impares, asuma por facilidad que va ingresando los números uno por uno.

Ejercicio 35. Generar_matriz

En un grupo de estudio para maratones de programación se presenta un problema en el cual únicamente le entregan la tabla de análisis y usted debe deducir de qué se trata el problema. Su objetivo es descubrir cuál proceso genera el arreglo bidimensional *matriz,* de acuerdo a las entradas *tamaño* e *inicio* y lograr reproducir el programa que lo genera como se muestra en la tabla 46.

Tabla 46. Entradas y salidas para el ejercicio generar_matriz

Entradas		Salidas				
tamano	inicio	matriz				
4	2	2	0	0	0	
		3	4	0	0	
		5	6	7	0	
		8	9	10	11	
5	10	10	0	0	0	0
		11	12	0	0	0
		13	14	15	0	0
		16	17	18	19	0
		20	21	22	23	24

Ejercicio 36. Cadena

Un procesador de texto permite agregar múltiples palabras a su diccionario. Para identificar una de otra, el procesador recibe cada una de las palabras separadas con comas, sin embargo, para almacenarlas individualmente luego debe separarlas cada una. Hacer un programa que reciba un listado de palabras que se ingresan separadas por comas y las separe con saltos de línea. El resultado debe quedar guardado en un vector y al final el programa debe mostrarlo.

Ejercicio 37. Separar_múltiplos

Como parte de un reto matemático, se tiene un grupo de números enteros y se necesita saber cuáles de ellos son múltiplos de 3 y de 5. Diseñe un programa que reciba una lista de *n* números enteros definida por el usuario y los clasifique en tres arreglos: una lista de números múltiplos de 3, otra lista de números múltiplos de 5 y otra lista con el resto de números. Tenga en cuenta que un número puede repetirse en las dos primeras listas. Al finalizar, el programa deberá imprimir cuatro listas: la lista con los números de entrada y las tres listas generadas.

Actividad 10.
Arreglos computacionales

Objetivo general

Diseñar programas que requieran el uso de arreglos a partir de requerimientos básicos de programación con estructuras de control que el aprendiz debe identificar siguiendo las fases de construcción de un programa.

Objetivos específicos

- Elaborar diagramas de flujo que requieren arreglos.
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Arreglos

Ejercicios

1. Para hallar los múltiplos de 3, basta con iniciar por este número e ir sumando de a tres, así tendríamos que los múltiplos son 3, 6, 9, 12, etc. Sin embargo, también existe otra forma que es, teniendo una lista de números, dividirlos por 3 y si el residuo es 0 el número es múltiplo de 3. A esto se le conoce como divisibilidad.

Desarrolle un programa que llene un arreglo con los números comprendidos entre 0 y 100 que son divisibles por 3.

2. En una encuesta llevada a cabo sobre una comunidad, se tomaron las edades de cinco personas. Desarrolle un programa que solicite cinco números enteros, los almacene en un arreglo y luego calcule la media aritmética de esos números.

3. En un informe de contabilidad se tiene una lista de valores. Los positivos representan las ganancias y los negativos las pérdidas. Se requiere de un programa que calcule la suma total de los elementos positivos y de los negativos, que informe la cantidad de elementos de ambos tipos, y además almacene cada uno de los elementos en un arreglo para evitar la pérdida de datos.

4. La matriz transpuesta de una matriz M(m,n) se obtiene intercambiando filas por columnas y viceversa. Escriba un programa que genere la matriz transpuesta de una matriz de 3 filas y 4 columnas ingresada por el usuario. El resultado se tiene que almacenar en una nueva matriz y, al final, el programa deberá imprimir la matriz de entrada y la nueva.

5. Una palabra es palíndroma cuando sigue siendo la misma al ser invertida, como por ejemplo la palabra *sometemos*. Un estudiante de español desea saber si ciertas palabras son palíndromas, por lo que desea obtener un programa que reciba una cadena y la escriba al revés.

5.4 Funciones

Una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la

misma, una vez que esta finalizó con su tarea, el control es devuelto al punto desde el cual la función fue llamada. En la figura 41 se presenta gráficamente la situación descrita anteriormente. En el programa principal, se llama al proceso A el cual realiza un conjunto de acciones. Posteriormente, se retorna al programa principal y se invoca en proceso B que realiza un conjunto de acciones. Finalmente retorna al programa principal y finaliza el programa.

Figura 41. Invocación de una función en un programa

Existen dos formas de que los programas principales invoquen a las funciones: llamadas de función por valor y por referencia. Cuando no se pasa un argumento utilizando una llamada por valor se hace una copia del argumento en la función llamada. Los cambios a la copia no afectan el valor de la variable original en el invocador, con esto se evita efectos secundarios que tanto minan el desarrollo de sistemas de software correctos y confiables.

Ejemplo 25. Sumatoria_funciones

Cada lenguaje de programación tiene una forma diferente de construir una función, sin embargo, son muy similares. Para familiarizar-

se con el concepto, construya un programa que reciba un número entero y mediante una función reemplace el valor de esta variable por la del número que le sigue. En la tabla 47 se presentan las entradas y salidas del algoritmo.

Tabla 47. Entradas y salidas para el ejemplo sumatoria_funciones

Entradas	Salidas
numero	*numero*
5	6
42	43

El diagrama de flujo para este ejemplo se presenta en la figura 42. Para este programa, inicialmente se solicita el número deseado. Después, ese número se pone dentro de una función llamada incremento, la cual se encuentra en la parte inferior del diagrama. En esta función, simplemente se aumenta en una unidad el número ingresado y este luego es retornado al programa principal donde se almacena en la variable *numero*. Finalmente, se imprime el resultado final del algoritmo. En la tabla 48 se presenta el código para este algoritmo en C++, Java y Python.

Figura 42. Diagrama de flujo para el ejemplo sumatoria_funciones

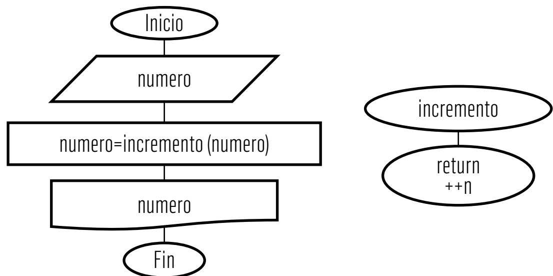


Tabla 48. Código en C++, Java y Python para el ejemplo sumatoria_funciones

Código en C++

```
1 int main() {
2 int numero;
3 cout<<”Ingrese un número: “;
4 cin>>numero;
5 numero = incremento(numero);
6 cout<<numero<<endl;
7 return 0;
8 }
9 int incremento (int n) {
10 return ++n;
11 }
```

Código en Python

```
1 def incremento (n):
2 return (n + 1)
3 numero = int(input(“Ingrese un número: “))
4 numero = incremento(numero)
5 print (numero)
```

Código en Java

```
1 public static void main(String[] args) {
2 Scanner teclado = new Scanner(System.in);
3 int numero;
4 System.out.print(“Ingrese un número: “);
5 numero = teclado.nextInt();
6 numero = incremento(numero);
7 System.out.print(numero + “\n”);
8 }
9 static int incremento (int n) {
10 return ++n;
11 }
```

Ejemplo 26.
Operaciones_funciones

Para una escuela necesitan un programa que permita hallar la suma de dos números, la raíz cuadrada de un número y el logaritmo de un número. Construya un programa que permita realizar estas diferentes operaciones matemáticas basándose en funciones y que pida al usuario la opción a escoger. En la tabla 49 se presentan las entradas y salidas del algoritmo.

Como se puede observar, dependiendo del valor asignado a la variable *opcion*, se realizará cada una de las operaciones deseadas. Por otra parte, la variable *seguir*, permite al usuario establecer si desea o no continuar con la ejecución del algoritmo. En la figura 43 se presenta el diagrama de flujo del algoritmo y las funciones requeridas dentro del mismo. Como se notará, se utiliza un ciclo *do while* para asegurar la repetición del algoritmo. Posteriormente, se solicita al usuario el ingreso de la opción deseada, luego la opción se procesa dentro de la cadena de condicionales y dependiendo de la opción seleccionada se ejecutará la función suma ingresando la palabra suma, raíz ingresando la palabra raiz o logaritmo ingresando la palabra logaritmo. En cada una de estas funciones se solicita el ingreso del número a procesar para luego entregar el resultado de acuerdo con el nombre de la función. Finalmente, se evalúa si se desea continuar con la ejecución del algoritmo. De ser afirmativo se ingresa la palabra 'si' y comenzará nuevamente el programa. De lo contrario el programa finalizará. En la tabla 50, se presenta el código en C++, Java y Python para este algoritmo.

Tabla 49. Entradas y salidas para el ejemplo operaciones_funciones

Entradas				Salidas		
opción	*seguir*	*dato1*	*dato2*	*raiz*	*logaritmo*	-
"suma"	si	56	12	-	-	68
"raiz"	si	-	-	121	-	11
"logaritmo"	no	-	-	-	100	2

Figura 43. Diagrama de flujo para el ejemplo operaciones_funciones

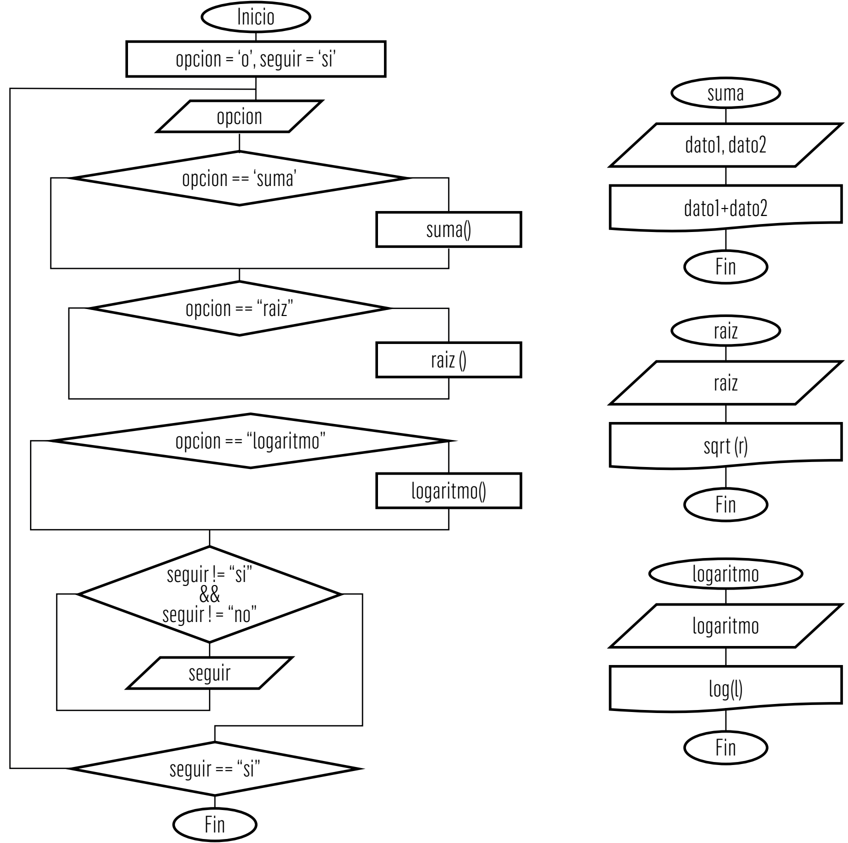


Tabla 50. Código en C++, Java y Python para el ejemplo operaciones_funciones

Código En Python
1 def suma():
2 print("Ingrese dos números:)
3 dato1 = float()
4 dato2 = float()
5 print(dato1+dato2)

Continúa

Código En Python

```
6
7 def raiz():
8 print("Ingrese un número:)
9 raiz = float()
10 print(math.sqrt(raiz))
11
12 def logaritmo():
13 print("Ingrese un número:)
14 logaritmo = float()
15 print(math.log10(l))
16 def main():
17 opcion = 'o'
18 seguir = 'si'
19 while seguir == 'si':
20 print("Digite s para sumar, r para hallar la raíz cua-
drada y l para logaritmo")
21 opcion = chr()
22 if opcion == 'suma':
23 suma()
24 if opcion == 'raiz':
25 raiz()
26 if opcion == 'logaritmo':
27 logaritmo();
28 print("Desea continuar (s/n)?")
29 seguir = chr()
```

Código en C++

```
1 int main() {
2 char opcion = 'o', seguir = 'si';
3 do{
4 cout<<"\nDigite s para sumar, r para hallar la raíz
cuadrada y l para logaritmo\n";
5 opcion = getch();
6 if (opcion == 'suma') suma();
```

Continúa

Código en C++

```
7 if (opcion == 'raiz') raiz();
8 if (opcion == 'logaritmo') logaritmo();
9 cout<<"\nDesea continuar (s/n)?\n";
10 seguir = getch();
11 while (seguir != 'si' && seguir != 'no') {
12 cout<<"\nDesea continuar (s/n)?\n";
13 seguir = getch();
14 }
15 } while (seguir == 'si');
16 }
17
18 void suma() {
19 float dato1, dato2;
20 cout<<"\nIngrese dos números:\n";
21 cin>>dato1;
22 cin>>dato2;
23 cout<<dato1+dato2;
24 }
25
26 void raiz() {
27 float raiz;
28 cout<<"\nIngrese un número:\n";
29 cin>>raiz;
30 cout<<sqrt(r);
31 }
32
33 void logaritmo() {
34 float logaritmo;
35 cout<<"Ingrese un número:\n";
36 cin>>logaritmo;
37 cout<<log(l);
38 }
```

Código En Java

```
1 public static void main(String[] args) {
2 Scanner teclado = new Scanner(System.in);
3 char opcion = 'o', seguir = 'si';
4 do{
5 System.out.print("\nDigite s para sumar, r para hallar
la raíz cuadrada y l para logaritmo\n");
6 opcion = teclado.nextChar();
7 if (opcion.equals('suma')) suma();
8 if (opcion.equals('raiz)') raiz();
9 if (opcion.equals('logaritmo')) logaritmo();
10 System.out.print("\nDesea continuar (si/no)?\n");
11 seguir = teclado.nextChar();
12 while(seguir.equals('si') == false && seguir.equal-
s('no') == false) {
13 System.out.print("\nDesea continuar (si/no)?\n");
14 seguir = teclado.nextChar();
15 }
16 } while (seguir.equals('si'));
17 }
18 static void suma() {
19 float dato1, dato2;
20 System.out.print("\nIngrese dos números:\n");
21 dato1 = teclado.nextFloat();
22 dato2 = teclado.nextFloat();
23 System.out.print(dato1+dato2);
24 }
25 static void raiz() {
26 float raiz;
27 System.out.print("\nIngrese un número:\n");
28 raiz = teclado.nextFloat();
29 System.out.print(Math.sqrt(r));
30 }
31 static void logaritmo() {
32 float logaritmo;
33 System.out.print("Ingrese un número:\n");
34 logaritmo= teclado.nextFloat();
35 System.out.print(Math.log(l));}
```

Ejercicios

Ejercicio 38. Calculadora_funciones

Una escuela desea tener un programa que simule una calculadora básica (suma, resta, multiplicación, división), pero también desean agregarle unas funciones extras. Construya un programa que permita realizar las operaciones básicas de una calculadora y además permita hallar la media de dos números y los divisores y el factorial de un número. La calculadora debe soportar números negativos, decimales y números grandes.

Ejercicio 39. Carga_arreglo_funciones

Un profesor de programación desea enseñar a sus estudiantes la forma en que los vectores almacenan los datos. Construya un programa que reciba 10 caracteres (letras, números o símbolos) llamando a una función y luego imprima en pantalla la posición del vector y el valor en esta. Ejemplo: "v[0] = a".

Ejercicio 40. Suma_mayor_funciones

Usando el concepto de funciones, desarrolle un programa que reciba dos números del usuario y calcule la suma y el mayor entre ellos. Use dos funciones diferentes para cada proceso.

Ejercicio 41. Edad_persona_funciones

En una base de datos de una empresa se tienen las fechas de nacimiento de cada uno de sus empleados. Se desea conocer la edad de cada uno en un momento específico. Construya un programa que permita determinar la edad en años, meses y días de una persona en una fecha ingresada por el usuario y teniendo la fecha de nacimiento

(recibir por separado el año, mes y día de cada una de las fechas). Suponga que las fechas siempre son ingresadas correctamente. Para este programa, implemente las siguientes tres funciones:

- **fechaValida:** comprueba si la fecha leída es correcta.
- **bisiesto:** comprueba si un año es bisiesto. La llama la función fecha_válida
- **calcularEdad:** recibe las dos fechas y devuelve la edad únicamente en años.

Actividad 11. Funciones

Objetivo general

Diseñar programas a partir de funciones que requieren el uso de estructuras de control que el aprendiz debe identificar siguiendo las fases de construcción de un programa.

Objetivos específicos

- Elaborar diagramas de flujo a partir de un requerimiento.
- Validar el diagrama de flujo diseñado a partir de conjunto de datos de entrada y salida.
- Hacer uso de funciones.
- Implementar en un lenguaje de programación de alto nivel.

Vocabulario

- Requerimiento
- Estructuras de control
- Funciones

Ejercicios

1. En un proceso industrial son generados productos para dos empresas distintas. Para poder generarlos secuencialmente para ambas empresas, los productos se etiquetan con números: a la primera empresa le corresponden los productos con número impar y a la segunda los productos con número par.

   Construya un programa que posea una función de nombre par, que tome un número entero como parámetro, y devuelva 1 si es par o devuelva 0 si es impar. Dependiendo de este resultado, debe mostrar un mensaje especificando que es par o impar.

2. Una empresa desea que usted desarrolle una aplicación para procesar cadenas de texto. Para este programa, desarrolle una función llamada *última*, que reciba una cadena de hasta 100 caracteres como parámetro, y devuelve el último carácter con una función y la cantidad de caracteres con otra.

3. Se tiene una matriz de 3x4 en la cual se encuentran los números enteros entre el 1 y el 9 (el resto está lleno de ceros). La matriz tiene la siguiente forma:

1	5	9
2	6	0
3	7	0
4	8	0

   Desarrolle un programa que use una función para mostrar la matriz de manera ordenada.

Capítulo 6

Técnicas de validación y comprobación

Este capítulo presenta las técnicas de validación frecuentes para evaluar y comprobar el funcionamiento de un algoritmo como las pruebas de escritorio o la validación de datos. Los temas por ver son:

- Pruebas de escritorio a pseudocódigo y diagramas de flujo.
- Pruebas de seguimiento al código desde el IDE.
- Conformación de conjuntos de datos para pruebas.
- Lógica para validación de datos de entrada.
- Manejo de excepciones.

6.1 Pruebas de escritorio a pseudocódigo y diagramas de flujo

Las pruebas de escritorio son aplicadas a los algoritmos diseñados, ya sea en pseudocódigo o diagramas de flujo, con el fin de verificar su correcto funcionamiento. Una prueba de escritorio es útil para determinar cuáles salidas producirá cierta entrada y es un método de fácil aplicación. Para aplicar una prueba de escritorio, debemos tener en cuenta todas las variables del sistema que se ven involucradas en las operaciones. La forma más común de representar una prueba de escritorio es por medio de una tabla como se presenta en la tabla 51.

Tabla 51. Tabla general para pruebas de escritorio

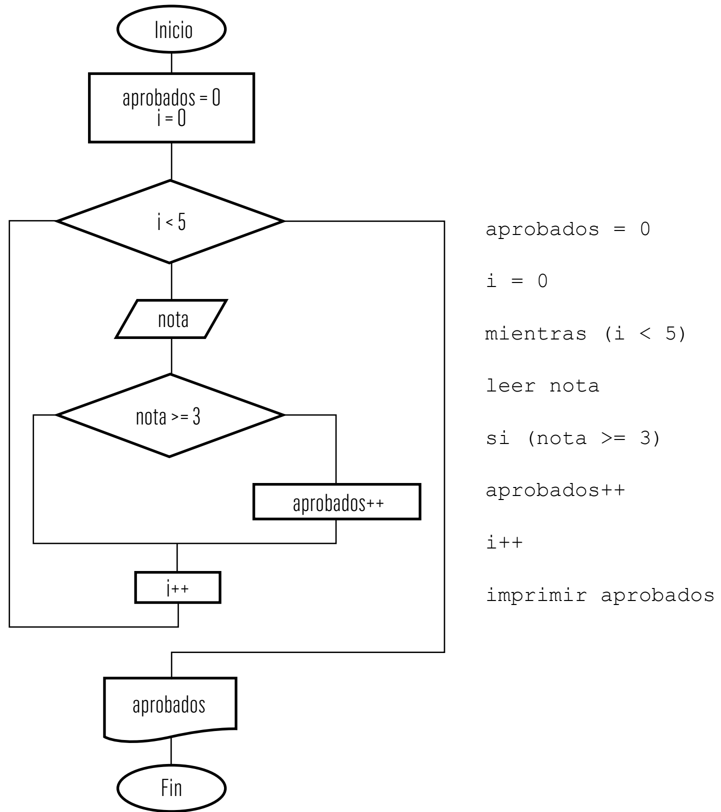
Instrucción	Entradas	Variables	Salidas

A continuación, se presenta el ejemplo 27, el cual, muestra la aplicación de una prueba de escritorio a un programa o algoritmo de forma general

Ejemplo 27.
Aprobación_estudiantes

Diseñe un programa que reciba la nota de 5 estudiantes y diga cuántos aprobaron (si una nota está entre 3 y 5). En la figura 44 se muestra el diagrama de flujo y el pseudocódigo del algoritmo.

Figura 44. Diagrama de flujo y pseudocódigo para el ejemplo aprobación_estudiantes



La prueba de escritorio para el ejemplo 27 se puede observar en la tabla 52. Como se puede ver, la prueba de escritorio se asemeja mucho a la tabla de análisis que se desarrollaba en los ejercicios anteriores del libro, pero más completa, ya que en este caso se contempla todo el desarrollo del algoritmo, instrucción por instrucción, con el objetivo de obtener un panorama general del funcionamiento del programa.

Tabla 52. Prueba de escritorio para el ejemplo aprobación_estudiantes

Instrucción	Entrada	Variables			Salida
		aprobados	i	nota	
aprobados = 0	-	0	-	-	-
i = 0	-	0	0	-	-
leer nota	3.4	0	0	3.4	-
aprobados++	-	1	0	3.4	-
i++	-	1	1	3.4	-
leer nota	2.5	1	1	2.5	-
i++	-	1	2	2.5	-
leer nota	1.9	1	2	1.9	-
i++	-	1	3	1.9	-
leer nota	5	1	3	5	-
aprobados++	-	2	3	5	-
i++	-	2	4	5	-
leer nota	4.2	2	4	4.2	-
aprobados++	-	3	4	4.2	-
i++	-	3	5	4.2	-
imprimir aprobados	-	3	5	4.2	3

6.2 Pruebas de seguimiento al código desde el IDE

6.2.1 C++ empleando Dev C++

Para activar la depuración en un proyecto de Dev-C++, el compilador que se usará para el lenguaje C++, debemos ir al menú *Proyecto* y hacer clic derecho en el nombre de nuestro proyecto. Allí se debe hacer clic en *Opciones de Proyecto*. En esta ventana se debe ir a la pestaña *Compilador* y en el submenú *Enlazador (linker)* asegurarse de que la opción *Generar Información de Depuración* está en *Yes*. Una vez activada esta opción, podremos usar el depurador para hacer un seguimiento al Código en C++ a cada uno de los puntos de interrupción que especifiquemos. Este procedimiento es presentado en la figura 45.

Figura 45. Seguimiento de código en C++ empleando Dev-C++

Para establecer un punto de interrupción debemos hacer clic al lado izquierdo del número de la línea en la cual deseamos que el programa se interrumpa como se muestra en la figura 46. Aquí hay dos puntos de interrupción establecidos: uno en el momento en el que se recibe el valor de *n* y otro en el momento en que se suma la variable *i*. Para comenzar la depuración podemos ir al menú *Depurar* y hacer clic en *Depurar,* o podemos presionar F8 en el teclado. La depuración comenzará y el programa se detendrá en el primer punto especificado. Desde allí podemos seguir el programa paso a paso con la tecla F7, lo cual nos puede ser muy útil en momentos en los que necesitamos saber en dónde existe un error, en qué lugar del código comienza el error o hacer otros tipos de seguimiento.

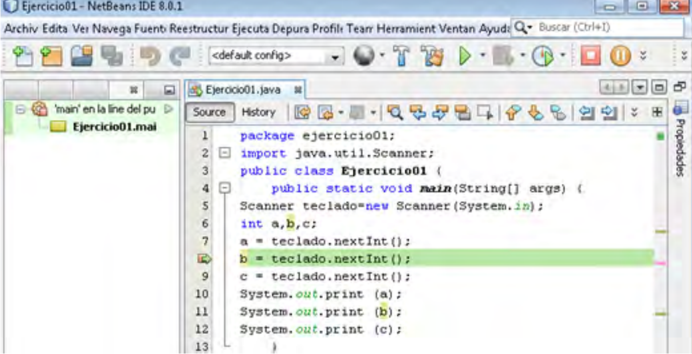
Figura 46. Punto de interrupción en Dev C++

```
14      cout<<"Ingrese un numero: ";
15      cin>>n;
16      while (i <= n)
17      {
18         if (n % i == 0) div++;
19         i++;
20      }
21      if (div == 2)
```

6.2.2 Java empleando NetBeans

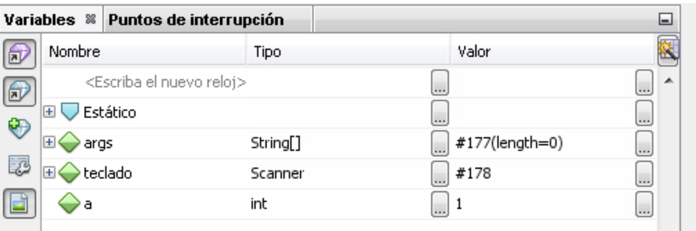
Las pruebas de seguimiento en Java en NetBeans se aplican de forma parecida a la que se hizo en Dev-C++. El procedimiento se muestra en la figura 47. Hacer clic en el menú *Depurar* y hacemos clic en *Debug Project*, o usamos las teclas Ctrl + F5. Una vez arranca el programa, continuará normalmente hasta detenerse en el punto de interrupción especificado. Los puntos de interrupción en NetBeans se definen haciendo doble clic en el número de la línea a la cual deseamos asignarlo.

Figura 47. Prueba de escritorio de un programa en NetBeans



En la ventana de depuración presentada en la figura 48, la cual normalmente se encuentra en la parte inferior de NetBeans, podemos observar también detalles de las variables que se han creado en el programa: el nombre, el tipo y su valor en el momento actual. Por ejemplo, la variable *a* es de tipo entero (*int*) y tiene almacenado el valor '1', el cual recibió por teclado en la línea 7 del código. Las variables *b* y *c* aún no se han creado, puesto que el programa se detuvo en la línea 8, antes de ejecutar estas instrucciones.

Figura 48. Ventana de depuración en NetBeans



Podemos continuar con el seguimiento línea por línea con la tecla F8 o paso por paso con la tecla F7. El seguimiento del código nos puede ser muy útil en momentos en los que necesitamos saber en

dónde existe un error, en qué lugar del código comienza el error o hacer otros tipos de seguimiento.

6.2.3 Python empleando Python IDLE

Las pruebas de seguimiento para código en Python pueden realizarse desde el *Shell* de Python. Para agregar un punto de interrupción debemos hacer clic derecho en la línea a la cual se le va a aplicar el seguimiento y hacer clic en la opción *Set Breakpoint.* Las líneas en las cuales se hayan establecido puntos de interrupción quedarán coloreadas de amarillo como se ve en la figura 49.

Figura 49. Puntos de interrupción en Python

```
File  Edit  Format  Run  Options  Window  Help
continuar = 1
while continuar == 1:
    i = 1
    div = 0
    n = int(input("Ingrese un numero: "))
    while i <= n:
        if n % i == 0:
            div = div + 1
        i = i + 1
    if div == 2:
        print("Es primo")
    else:
        print("Es compuesto")
    continuar = int(input("Ingrese 1 para continuar: "))
```

Para iniciar la depuración, vamos al *Shell* de Python y hacemos clic en la opción *Debugger* del menú *Debug*. Nos aparecerá una ventana con el título *Debug Control*. Después debemos iniciar el programa, lo cual hacemos desde el editor de código con la tecla F5 o la opción *Run Module* en el menú *Run*. Una vez iniciado el programa podremos ir paso por paso desde la ventana *Debug Control* con la opción *Step* o saltar directamente entre puntos de interrupción con la opción *Out*. Si queremos detener la ejecución del programa, usamos la opción *Quit*, y si queremos dejar de depurar simplemente cerramos la ventana de *Debug Control.*

Las pruebas de seguimiento son muy útiles cuando se necesita buscar el origen de un problema o cuando se quiere verificar el funcionamiento de determinada parte del código. Con un poco de práctica, este tipo de técnicas facilitan la programación eficiente y nos ayudan a desarrollar programas evitando los errores.

6.3 Conformación de conjuntos de datos para pruebas

Para la ejecución de pruebas en el desarrollo de programas es muy recomendable preparar un conjunto con datos de entrada que nos sirvan para validar el proceso y sus salidas en condiciones diferentes. Por esta razón, entre los datos de prueba debe haber también valores que puedan darse en condiciones extrañas o de los cuales se sospeche que el programa pueda generar errores.

Retomemos un ejemplo. En matemáticas, las operaciones básicas que se pueden aplicar sobre un conjunto de números enteros son la suma (+), la resta (-), la multiplicación (*) y la división (%). Esta vez haga un programa que implemente todas las operaciones básicas y aplique una de ellas a un par de números. En ese ejemplo, la tabla de análisis está dada por la tabla 53, la cual se presenta a continuación.

Tabla 53. Entradas y salidas para el ejemplo operaciones_en_matemáticas

Entradas		Salidas			
número1	número2	suma	resta	producto	división
20	12	32	8	240	1
8	40	48	-32	320	0
15	3	18	12	45	5

Cada valor de las variables que se encuentra en *Entradas* hace parte del conjunto de datos para pruebas. Cada línea de estas corresponde a las entradas para un caso de prueba. Por ejemplo, para el

caso de prueba de la primera línea, tenemos las entradas 20 y 12. Esto quiere decir que, en este caso, la variable *numero1* tendrá el valor de 20, la variable *numero2* sera 12. Mediante el proceso que llevará a cabo el programa (y que se puede observar en el diagrama de flujo correspondiente a este ejemplo) se obtendrá el valor de la(s) variable(s) de salida, que en este caso son *suma*=12, *resta*=8, *producto*=240 y *división*=1.

La importancia de este conjunto de datos se hace más evidente cuando realizamos programas más complejos, los cuales suelen tener una gran cantidad entradas o salidas. En estos casos, el uso de archivos que contengan el conjunto de valores de entrada es aplicado para facilitar el ingreso de estas en el programa.

Como ejemplo se expone un ejercicio presentado en el Uva Online Judge: **100- The 3n + 1 problem.** Para este ejercicio debe considerarse el siguiente pseudocódigo:

```
1. leer n
2. imprimir n
3. si n = 1 entonces PARAR
4.          si n es par entonces:
5.                    n = 3n + 1
6.          si no:
7.                    n = n / 2
8. GOTO 2
```

Este algoritmo termina su ejecución al imprimir el número 1. Por ejemplo, si *n* fuera 22, la salida para ese algoritmo sería la siguiente: 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1. La cantidad de números calculados es llamada *longitud de ciclo de n.* Entonces para este ejemplo, la longitud de ciclo de 22 es 16. El ejercicio consiste en que dados dos números *i* y *j*, se debe determinar la máxima longitud de ciclo para un número que se encuentre entre aquellos dos.

Este problema pertenece a una práctica para maratones de programación, en donde se requiere una gran cantidad de datos en el conjunto para la realización de pruebas sobre el algoritmo, y para el cual es recomendado ingresar estos valores en un archivo. Un ejemplo de un posible conjunto de datos de entrada y salida para este ejercicio es el siguiente, donde hay 22 líneas, es decir, 22 posibles casos con sus respectivas entradas y salidas.

Datos de entrada	Datos de salida
1 10	1 10 20
21 22	21 22 16
100 200	100 200 125
201 210	201 210 89
900 1000	900 1000 174
1 1000000	1 1000000 525
1000000 1	1000000 1 525
201 202	201 202 27
202 203	202 203 40
203 204	203 204 40
204 205	204 205 27
205 206	205 206 89
206 207	206 207 89
207 208	207 208 89
208 209	208 209 40
209 210	209 210 40
201 210	201 210 89
22 22	22 22 16
1 500000	1 500000 449
1 50000	1 50000 324
1 5000	1 5000 238
5000 1	5000 1 238

De esta forma, el programador puede darse cuenta de lo acertado que es el programa, evaluando las múltiples soluciones y detectando los posibles errores que pueda estar cometiendo en el proceso.

6.4 Manejo de excepciones

Las excepciones se dan cuando un compilador retorna un mensaje de error ocasionado por algún problema en el código del progra-

ma que se dio en el tiempo de ejecución, a diferencia de los errores de sintaxis, los cuales se dan por comandos mal escritos y que el compilador detecta antes de iniciar la ejecución del programa. Estas excepciones son mostradas por los compiladores con mensajes de advertencia después de que este detiene el programa, con el fin de que el programador encuentre el tipo de error que fue causado y el origen del problema en el código.

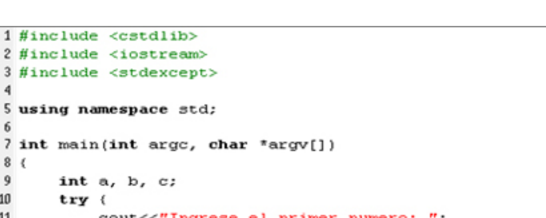
El manejo de excepciones es la técnica que permite al programador controlar este tipo de excepciones, con lo cual se puede definir una respuesta o acción que se dará después de que se produce tal error en el programa.

Según el lenguaje de programación escogido, la forma de manejar las excepciones va a variar en cuanto a sintaxis, pero suele ser similar en concepto y estructura. Para este libro se mostrará el manejo de excepciones en los tres lenguajes de programación que se vienen tratando: C++, Java y Python.

6.4.1 C++

En C++ existe una instrucción elemental para el manejo de excepciones: la instrucción *try-catch*. El bloque *try* lo usamos para encerrar la parte del código que nos puede arrojar una excepción, mientras que *catch* lo usaremos para definir las instrucciones que seguirá el programa en el caso de que la excepción especificada se cause. El tipo de excepción es reconocido por el programa a través de la expresión *throw*, la cual suele ser implícita (a menos que se la defina dentro del *try*, con lo cual pasaría a ser explícita). En el caso de que no haya un bloque *catch* con la excepción ocurrida, el programa se detendrá y no permitirá continuar con la ejecución, por lo que es recomendable también especificar un bloque *catch* generalizado. Un ejemplo de manejo de excepciones para un programa que calcula la división entre dos números se presenta en la figura 50.

Figura 50. Programa para el manejo de excepciones en C++



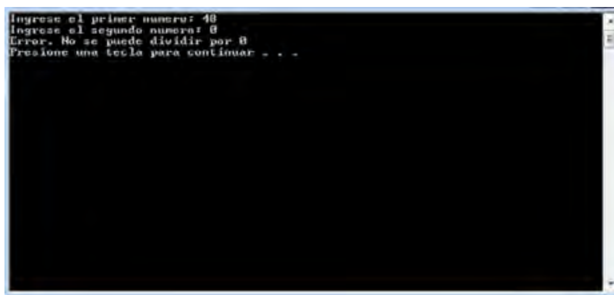
```
1 #include <cstdlib>
2 #include <iostream>
3 #include <stdexcept>
4
5 using namespace std;
6
7 int main(int argc, char *argv[])
8 {
9      int a, b, c;
10     try {
11          cout<<"Ingrese el primer numero: ";
12          cin>>a;
13          cout<<"Ingrese el segundo numero: ";
14          cin>>b;
15          if (b == 0)
16          {
17              throw "excepcion";
18          }
19          else
20          {
21              c = a / b;
22              cout<<"El resultado es: "<<c<<endl;
23          }
24     }
25     catch(...)
26     {
27          cout<<"Error. No se puede dividir por 0"<<endl;
28     }
29     system("PAUSE");
30     return EXIT_SUCCESS;
31 }
```

Si el segundo número fuera 0, la aplicación arrojaría un error, ya que la división por 0 no existe. En este caso se usa *throw* junto con cualquier parámetro, y luego en el *catch* para este ejemplo se emplea "...", con lo cual recibirá cualquier tipo de excepción que se encuentre. Al intentar dividir algún número por 0, el resultado de la excepción se presenta en la figura 51 (página siguiente).

La clase que controla las excepciones manejadas en C++ es "std::exception". De ella se derivan todas las demás excepciones que se pueden controlar. Algunas excepciones que son posibles de controlar mediante el manejo de excepciones en C++ son:

- **std::bad_alloc** (ocasionada cuando no se puede asignar el espacio de almacenamiento requerido)
- **std::bad_array_new_length** (cuando una matriz es declarada con un tamaño inválido)
- **std::bad_exception** (cuando se devuelve una excepción inválida)

Figura 51. Excepción en C++



- **std::bad_function_call** (originada cuando se llama a una función que está vacía, es decir, *null*)
- **std::ios_base::failure** (causada por errores de entrada o salida)
- **std::overflow_error** (cuando hay un desbordamiento causado por errores aritméticos)
- **std::regex_error** (error causado por alguna librería de expresiones regulares) **std::system_error** (error del sistema)
- **std::underflow_error** (cuando hay un subdesbordamiento causado por errores aritméticos)

Para poder usarlas, es necesario incluir la librería "stdexcept" agregando ***include <stdexcept>*** al comienzo del programa, en donde se definen las librerías.

6.4.2 Java

En Java, el manejo de excepciones también se hace mediante la estructura *try-catch.* Su uso es el mismo que en C++, usamos un bloque *try* en donde encerramos el código del que vamos a controlar las excepciones, y un bloque *catch* en el cual definiremos las acciones que ejecutará si llega a capturar la excepción especificada. Al final también se podría incluir un bloque *Finally*, en el cual se incluyen las acciones de finalización, que son instrucciones ejecutadas sin importar si hay una excepción o no, y que por lo general son de limpieza. Para el mismo ejemplo de manejos de excepciones que se hizo con C++, calcular la división entre dos números y validar la división por 0, se hace como se muestra en la figura 52.

Figura 52. Manejo de excepciones en Java

```
Scanner teclado = new Scanner(System.in);
int a, b, c;
try {
    System.out.println("Ingrese el primer numero: ");
    a = teclado.nextInt();
    System.out.println("Ingrese el segundo numero: ");
    b = teclado.nextInt();
    c = a / b;
    System.out.println("El resultado es: " + c + "\n");
} catch (ArithmeticException e) {
    System.out.println("Error. No se puede dividir por 0\n");
}
```

Si durante la división, ingresamos un valor que genere una operación aritmética inválida, como dividir por 0, el programa generará la excepción y se ejecutarán las instrucciones dentro del *catch*, en este caso se imprimirá el mensaje de error presentado en la figura 53.

Figura 53. Excepción en Java

```
Salida - ProgramaPrueba (run)
run:
Ingrese el primer numero:
5
Ingrese el segundo numero:
0
Error. No se puede dividir por 0
```

También es posible controlar excepciones de forma generalizada usando *Exception* dentro del *catch,* como muestra la figura 54.

Figura 54. Excepción generalizada en Java

```
Scanner teclado = new Scanner(System.in);
int a, b, c;
try {
    System.out.println("Ingrese el primer numero: ");
    a = teclado.nextInt();
    System.out.println("Ingrese el segundo numero: ");
    b = teclado.nextInt();
    c = a / b;
    System.out.println("El resultado es: " + c + "\n");
} catch (ArithmeticException e) {
    System.out.println("Error. No se puede dividir por 0\n");
}
catch (Exception e) {
    System.out.println("Error durante la ejecución del programa\n");
}
```

De esta forma, si se ingresa un caracter diferente de un número, al no ser un error de aritmética, el programa imprimirá el mensaje de la figura 55.

Figura 55. Mensaje de excepción generalizada en Java

```
Salida - ProgramaPrueba (run)
run:
Ingrese el primer numero:
dos
Error durante la ejecución del programa
```

En Java, las excepciones se encuentran en el paquete *java.lang,* el cual se encuentra importado por defecto en todos los programas de Java. Algunas de las expresiones comúnmente manejadas son:

- **ArithmeticException** (ocurre por un error aritmético como intentar dividir por 0)
- **ArrayStoreException** (cuando se intenta asignar a una matriz un valor de tipo incorrecto)

- **ClassNotFoundException** (cuando no existe la clase invocada)
- **IllegalArgumentException** (cuando se invoca un método con los argumentos inválidos)
- **IndexOutOfBoundsException** (cuando el índice está fuera del rango, por ejemplo, en una matriz)
- **InstantiationException** (ocurre cuando se intenta crear un objeto a partir de una interfaz o de una clase abstracta)
- **NegativeArraySizeException** (cuando se crea una matriz de tamaño negativo)
- **NoSuchMethodException** (si el método invocado no existe)
- **NullPointerException** (si se retorna *null* cuando se intenta llamar a un objeto)
- **NumberFormatException** (ocurre cuando un *String* que se intenta convertir a algún tipo numérico posee un formato inválido)
- **StringIndexOutOfBounds** (cuando el tamaño de la matriz definida excede el límite definido)
- **UnsupportedOperationException** (cuando la operación que se desea ejecutar no puede ser realizada)

6.4.3 Python

El manejo de excepciones en Python se realiza con la estructura *Try-Except,* una estructura que cumple la misma función que el *Try-Catch* visto en los otros lenguajes. Al igual que en Java, y a diferencia de C++, el *Try-Except* también puede incluir *Finally,* el cual incluye el código que se ejecutará sin importar si hay una excepción o no, y que por lo general es código de limpieza. La estructura básica de la estructura *Try-Except* en Python se presenta en la figura 56 (página siguiente).

Como ejemplo se realizará el mismo ejercicio planteado anteriormente, en el que se hará la división entre dos números enteros. El código para este ejemplo se presenta en la figura 57 (página siguiente).

Teniendo esto, cuando el usuario trate de dividir algún número por 0, se generará un error de tipo *ArithmeticError* y se mostrará el error definido en la estructura *except* ("Error. No se puede dividir por 0").

Figura 56. Estructura de una excepción en Python

```
try:
    código con posibles excepciones
except tipo de error:
    código que se ejecutará si ocurre la
    excepción especificada
finally:
    acciones de finalización
```

Figura 57. Excepción implementada en Python

```
try:
    a = int(input("Ingrese el primer numero: "))
    b = int(input("Ingrese el segundo numero: "))
    c = a / b
    print("El resultado es: ", c)
except ArithmeticError:
    print("Error. No se puede dividir por 0.")
```

Las excepciones en Python poseen una jerarquía en la cual la clase de la que se derivan todas es llamada *BaseException*. Hay tres clases de excepciones principales que se pueden encontrar: *StopIteration*, *StandardError* y *Warning*. Las excepciones que más suelen usarse se derivan de *StandardError*, de ellas se pueden mencionar entre las más importantes:

- **BufferError:** cuando una operación relacionada con *buffer* no puede ser realizada
- **ArithmeticError:** error en operaciones aritméticas como es la división por cero

- **AttributeError:** cuando una referencia o asignación a un atributo genera un error
- **EnvironmentError:** de esta se derivan los errores que tienen que ver con algo por fuera del entorno, como errores de lectura y escritura (IOError).
- **MemoryError:** cuando una operación excede el uso de memoria pero aún puede ser corregida
- **RuntimeError:** cuando ocurre un error que no entra en ninguna de las otras categorías.
- **SyntaxError:** cuando existe un error de sintaxis en el código
- **SystemError:** cuando existe un error interno del sistema pero no es necesario detener el programa
- **TypeError:** cuando una operación recibe un argumento del tipo incorrecto
- **ValueError:** cuando una operación recibe un argumento de tipo válido pero con un valor incorrecto

Capítulo 7
Paradigmas de programación

Un paradigma de programación define la forma en la cual se modelará un programa de computadora, pues determina el esqueleto del programa, el tipo de estructuras que se usarán y el enfoque que se aplicará para la solución de los problemas planteados. En este capítulo se verán los paradigmas de programación más aceptados y cómo se construye un programa teniendo en cuenta sus principios. Los temas que se verán en esta unidad son:

- Programación estructurada
- Programación orientada a objetos
- Programación orientada a eventos
- Otros paradigmas de programación

7.1 Programación estructurada

En los primeros años de la historia de la programación, cuando empezaban a ser implementados los primeros lenguajes de programación, apareció el que es probablemente el primer paradigma de programación: la programación estructurada. Este paradigma apareció con el fin de otorgar al programa una estructura organizada, en la cual el flujo de la información fuera fácilmente entendible a partir de la secuencia de instrucciones. Esto quiere decir que las instrucciones de un programa siempre se ejecutarían una tras otra.

Los programas desarrollados con este paradigma hacen uso de dos estructuras lógicas básicas: las estructuras de selección (if) y las estructuras de iteración (*while, for*), y tienen un orden secuencial. Las ventajas de este paradigma son facilitar la compresión, depuración y mantenimiento del código. Lenguajes de programación como Pascal, Ada y C han sido ampliamente usados para el desarrollo de programas que siguen este paradigma. La figura 58 es un ejemplo de programación estructurada en C++.

En este programa se puede ver que se sigue un orden secuencial, con cada instrucción ordenada según como se quiere ejecutar. Existen dos estructuras de iteración y dos estructuras de selección. El programa termina su ejecución una vez la estructura *while* más externa (la que posee la condición *continuar == 1*) termina todas sus iteraciones, dado que es la última instrucción que se ejecuta (fíjese en que el corchete que cierra la estructura es la última línea del código).

Figura 58. Código en C++ empleando programación estructurada

```
int n, continuar, i, div;
continuar = 1;
while (continuar == 1)
{
   i = 1;
   div = 0;
   cout<<"Ingrese un numero: ";
   cin>>n;
   while (i <= n)
   {
      if (n % i == 0) div++;
      i++;
   }
   if (div == 2)
   {
      cout<<"Es primo"<<endl;
   }
   else
   {
       cout<<"Es compuesto"<<endl;
   }
   cout<<"Ingrese 1 para continuar: ";
   cin>>continuar;
}
```

7.2 Programación funcional

La programación funcional es un paradigma basado en la programación declarativa y que busca la solución de problemas mediante el uso de funciones matemáticas que ignoran los cambios de estado o datos mutables. Estas funciones no tienen el mismo comportamiento que un método, proporcionan una relación entre los datos de entrada y de salida, por lo cual la calidad del resultado depende de las funciones definidas en el programa.

Dentro de un programa construido bajo este paradigma, se reconocen varios tipos de funciones: funciones puras, de primera clase y de orden superior. Las funciones de primera clase y de orden superior pueden tomar otra función como argumento o dar como resultado una función, lo cual es una práctica común en el desarrollo de programas bajo este paradigma. Las funciones puras (también conocidas como expresiones) reciben únicamente valores inmutables y pueden ser modificadas con el fin de optimizar el código.

La aplicación de lenguajes de programación funcional se ha dado más en ámbitos académicos, dada su formalidad y mayor complejidad. Lenguajes de programación como Hope y Rex son ampliamente usados en áreas académicas, mientras que lenguajes como Wolfram (Mathematica), Haskell y F# también han sido usados para aplicaciones industriales y comerciales.

7.3 Programación orientada a objetos

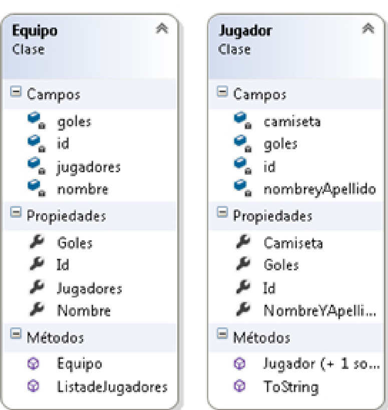
La programación orientada a objetos es un paradigma de programación basado en la programación estructurada y en la cual la estructura de un programa está constituida por estructuras de datos conocidas como “objetos”. Estos objetos se componen de dos elementos fundamentales: atributos y métodos. Los atributos son las características que posee un objeto, que se definen en variables y son los que lo definen. Mientras que los métodos son los comportamientos que tiene el objeto, es decir, las operaciones que puede realizar sobre sí mismo o sobre otros objetos.

En la programación orientada a objetos también se presenta el concepto de “clases”. Las clases son los moldes a partir de los cuales se crea un objeto. Un objeto es una **instanciación** de una clase; pueden existir múltiples objetos instanciados de una clase, los cuales tendrán los mismos atributos y métodos, pero al ser objetos diferentes no tendrán los mismos valores. Para entender mejor el concepto se presenta el siguiente ejemplo.

En la figura 59, se presenta un diagrama de clases, se pueden ver las clases existentes con sus respectivos atributos (**campos y propiedades**) y métodos. Las clases son abstracciones de un concepto que posee características y que puede tener operaciones. En el ejemplo existen dos clases: **Equipo**, abstracción de un equipo de fútbol, y **Jugador**, abstracción de un jugador. Usted puede notar que podrían crearse múltiples instanciaciones de una de las clases, puesto que son conceptos que pueden representar diferentes unidades (como dos jugadores diferentes) que comparten características (los jugadores poseen un número de camiseta, un nombre, etc.). Ambos campos y propiedades representan las características del objeto, sin embargo su declaración en el código es diferente. En el ejemplo se evidencia que los campos y las propiedades corresponden a los mismos conceptos.

El uso de propiedades suele darse cuando los campos son privados, es decir que la modificación de sus valores solo puede darse desde el objeto en sí. Las propiedades permiten modificar los valores de un campo evitando su modificación directa mediante el uso de dos métodos conocidos como *getters* y *setters*. Los *getters* retornan el valor que se encuentra en la variable, mientras que los *setters* asignan o modifican un valor a aquella variable.

Figura 59. Diagrama de clases



En el ejemplo existen cuatro características para Equipo: goles, id, jugadores y nombre. Para este ejemplo codificaremos en lenguaje Java. En la figura 60 se presenta la codificación de una clase, teniendo en cuenta los atributos establecidos.

Figura 60. Definición de una clase en Java

```
public class Jugador
    {
        private int id, goles, camiseta;
        private String nombreyApellido;

        public Jugador(int id, String nombreyApellido, int goles, int camiseta)
        {
            this.id = id;
            this.nombreyApellido = nombreyApellido;
            this.goles = goles;
            this.camiseta = camiseta;
        }
```

Los atributos se declaran en la parte superior, afuera de los métodos. Son privados por qué se quiere evitar su modificación directa desde otra parte fuera de la clase. El ID, la cantidad de goles y el número de camiseta son variables enteras, mientras que el nombre y el apellido se almacenará en una variable *String* puesto que es una cadena de texto. El primer método que se crea es el constructor de la clase. Este siempre debe ir puesto que es con el cual se instancia un objeto de la clase. El constructor debe ser público ya que el objeto tiene que crearse desde otra clase, que para este ejemplo será la clase *Main* (la clase principal del programa). Las variables en medio de los paréntesis del constructor son los parámetros (son el mismo concepto visto en el tema de Funciones).

Al construir una clase, los parámetros reciben valores, los cuales son asignados a las variables locales que se identifican con un *this.* Por ejemplo, para la clase Jugador se recibe un ID, y mediante la línea *this.id = id,* se almacena el valor recibido en la variable local *id,* que por ser privada no podría haber recibido el valor de manera

directa. Los *getters* y *setters* tienen una estructura parecida a la de un constructor. Como se mencionaba arriba, el *getter* retorna el valor de la variable, en este caso *id,* y el *setter* lo modifica. La figura 61 presenta la definición de un *getter* y un *setter* dentro de una clase.

Figura 61. Definición de *getters y setters*

```
public int getId()
{
    return id;
}

public void setId(int id)
{
    this.id = id;
}
```

En la clase Equipo, existen dos métodos: Equipo (el constructor de la clase) y ListadeJugadores. El método ListadeJugadores, definido en la figura 62, tiene la función de pedir los datos de 4 jugadores y luego imprimir la lista con la información de cada uno de ellos. Como puede ver, este método ejecuta operaciones propias de la clase Equipo, además en este ejemplo interactúa con la clase Jugadores, ya que durante el *for* crea cuatro objetos de Jugador con los valores ingresados por el usuario y luego usa el método *toString* de la clase Jugador para imprimir la información de cada objeto. La programación orientada a objetos actualmente es uno de los paradigmas de programación más empleados y es soportado por una gran cantidad de lenguajes de programación, como los usados en este libro: C++, Java y Python. La abstracción de conceptos en objetos y su facilidad de comprensión dada su estructura hacen que sea un paradigma importante para la programación.

Figura 62. Definición de un método

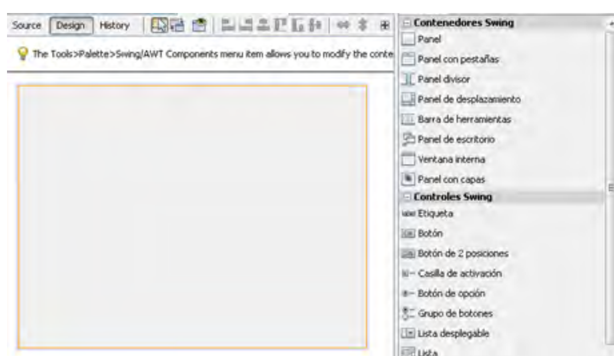
```java
public void ListadeJugadores() {
    Scanner teclado = new Scanner(System.in);
    Jugador[] equipodeJugadores = new Jugador[3];
    for (int i = 0; i < 3; i++) {
        System.out.println("Ingrese los datos del jugador No. " + (i + 1));
        System.out.println("Id: ");
        int jId = teclado.nextInt();
        System.out.print("Nombre y apellido: ");
        String jNombreyApellido = teclado.next();
        System.out.print("Goles: ");
        int jGoles = teclado.nextInt();
        System.out.print("No. Camiseta: ");
        int jCamiseta = teclado.nextInt();

        equipodeJugadores[i] = new Jugador(jId, jNombreyApellido, jGoles, jCamiseta);
    }
    for (int i = 0; i < 3; i++) {
        System.out.println(equipodeJugadores[i].toString());
    }
}
```

7.4 Programación orientada a eventos

La programación orientada a eventos es un paradigma en el cual la secuencia de un programa no sigue un flujo secuencial, sino que es dirigido por acciones conocidas como eventos, los cuales pueden ser externos (las acciones del usuario como oprimir una tecla) o pueden ser internos (hilos o temporizadores). Este paradigma de programación es usado ampliamente en aplicaciones con interfaz gráfica que puede incluir dispositivos móviles (Android) o aplicaciones web (JavaScript).

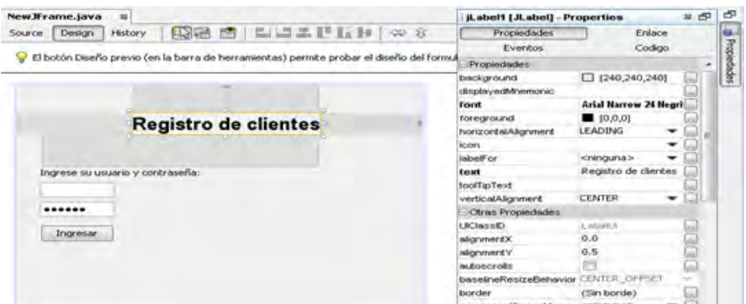
Usted puede crear aplicaciones con entorno gráfico en lenguaje Java por medio de NetBeans. Para ello, abra la aplicación y cree un nuevo proyecto. Al proyecto agregue un *Formulario JFrame* desde el mismo menú en el cual agregaría una clase. Los *JFrame* son clases pertenecientes a la biblioteca *Swing* con las cuales se crea una ventana en entorno gráfico a la cual se le pueden agregar otros componentes como botones, cajas de texto, tablas, entre otros. El resultado del proceso anterior se presenta en la figura 63.

Figura 63. Creación de una interfaz gráfica en NetBeans



Como se muestra en la figura 63, en las pestañas superiores usted podrá elegir si desea editar el código (*Source*) o si desea hacerlo de manera gráfica (*Design*). El panel de la derecha contiene los elementos que usted puede agregar a la ventana. Simplemente escoja uno y haga clic en la parte de la ventana en donde desea situarlo. Se recomienda colocar siempre un *Panel* en la ventana, en el cual colocar encima los componentes, pues esta práctica será de gran utilidad cuando llegue a desarrollar programas más complejos.

Figura 64. Edición de interfaz gráfica en NetBeans



Usted puede editar las propiedades de los objetos haciendo clic en la pestaña *Propiedades* en la parte derecha. Allí existe también una pestaña llamada *Eventos* en la cual podrá definir las acciones que realice el objeto una vez un disparador (*trigger*) ha sido activado. Por ejemplo, crear otra ventana cuando el botón *Ingresar* sea presionado (este evento se llama *actionPerformed*). Dado que los JFrame también son objetos, para poder visualizar la ventana que hemos creado primero debemos inicializarla. Para ello, nos ubicamos en el *main* de la clase principal de nuestro programa, la cual en NetBeans es creada con el nombre de nuestro proyecto. En este ejemplo la clase principal se llama *ProgramaPrueba.* Una vez en el *main,* debemos crear un nuevo objeto de nuestra clase JFrame creada y usamos el método *setVisible()* con el parámetro *true* para hacerla visible. Después de ejecutar el programa, usted podrá observar que se ha creado la ventana programada como se presenta en la figura 65.

Aunque la programación orientada a eventos no es el único paradigma que incluye diseño de interfaces gráficas, es tal vez el paradigma que mejor los aprovecha, el comportamiento del programa será determinado por las acciones que el usuario aplique sobre él.

Figura 65. Definición de la clase principal en NetBeans y ejecución de la ventana creada

```
public class ProgramaPrueba {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Ventana v = new Ventana();
        v.setVisible(true);
    }

}
```

Registro de Clientes

Ingrese su usuario y contraseña:

Ingresar

7.5 Otros paradigmas de programación

Existen muchos más paradigmas de programación, cada uno con una forma distinta de otorgar una estructura a un programa. Usted ya vio tres de los paradigmas más importantes: programación estructurada, orientada a objetos y orientada a eventos. A continuación, se le explicarán brevemente otros paradigmas de programación que también han sido reconocidos como importantes, aunque en un nivel menor a los ya explicados.

7.5.1 Programación orientada a aspectos

La programación orientada a aspectos es un paradigma reciente. Apareció a inicios de la década de los 90 por parte de Gregor Kiczales y su grupo de investigación (Reina, 2000). Este paradigma busca principalmente definir conceptos en unidades lo más específicas posibles y volverlas independientes entre ellas. Con esto se busca lograr un mínimo acoplamiento, lo cual facilita la reusabilidad del código y la unión de distintas partes siempre y cuando sean compatibles, además también se busca lograr un código entendible y sobre el cual se pueda aplicar depuración de forma fácil y sobre un elemento en concreto.

Existen complementos diseñados específicamente para la aplicación de la programación orientada a aspectos en lenguajes de programación ya existentes. Por ejemplo, las extensiones **AspectJ** y **AspectC++,** diseñadas para Java y C++ respectivamente.

Este paradigma de programación apenas se encuentra en crecimiento y no debe dejar de desarrollarse si quiere llegar a ser un paradigma importante. Existen ventajas en tener un programa con esta estructura: se puede llegar a un mayor nivel de modularidad, lo cual también facilitará la evolución del programa (se pueden modificar aspectos de forma fácil), lograr un código limpio, la imple-

mentación de requerimientos de forma separada y la reusabilidad en otros sistemas.

Sin embargo, también hay que considerar que existen falencias considerables que, según Asteasuain & Contreras (2002), se pueden resumir en: posibles choques entre el código funcional y de aspectos (que ocurren al violar el encapsulamiento para la implementación de los aspectos), choques entre los aspectos (dos aspectos que funcionan por separado, pero no al aplicarlos sobre un conjunto) y choques entre el código de aspectos y los mecanismos del lenguaje (como es la *anomalía de la herencia*).

7.5.2 Programación por procedimientos

La programación por procedimientos se deriva de la programación estructura y se basa en el concepto de “divide y vencerás”. Su principal característica es presentar una estructura basada en “procedimientos” (concepto similar a lo que en programación orientada a objetos llamaríamos “métodos”), con lo cual se evita la repetición de código mediante la reusabilidad. Entre los lenguajes más comunes al implementar este paradigma se encuentran C++, BASIC y Pascal.

La programación por procedimientos comparte la filosofía de la programación orientada a aspectos al buscar programas con mayor modularidad, sin embargo, una de las grandes desventajas de la programación por procedimientos en contraste con la programación orientada a aspectos es que no poseen abstracción de los conceptos que se presentan en el mundo real, lo cual dificulta su reutilización en otros sistemas y su efectividad en la solución de problemas reales.

Apéndice

Sistemas de numeración

Se llama sistema de numeración al conjunto de reglas que permiten representar conceptos matemáticos abstractos mediante una serie de símbolos llamados números. Con respecto a la base, los sistemas de numeración pueden dividirse como se muestra en la Figura A1.

Figura A1. Sistemas de numeración

Sistema decimal
* Posicional de base 10 * Se acoge a la regla de las potencias * Dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Sistema hex adecimal
* Posicional de base 16 * Se acoge a la regla de las potencias * Dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Sistema binario
* Posicional de base 2 * Se acoge a la regla de las potencias * Dígitos 0, 1

Sistema octal
* Posicional de base 8 * Se acoge a la regla de las potencias * Dígitos 0, 1, 2, 3, 4, 5, 6, 7

Existen otros sistemas de numeración tales como el sistema duodecimal (base 12) y el sistema sexagesimal (base 60) empleados en la antigüedad para el control de la producción agrícola los cuales actualmente se encuentran en desuso.

Sistema decimal

Es un sistema de numeración posicional que emplea 10 dígitos para la representación de cantidades numéricas, en esto caso, los dígitos correspondientes a los diez primeros números arábigos 0,1,2,3,4,5,6,7,8,9. Por esto se puede decir que el sistema de numeración decimal tiene una base numérica de 10 elementos.

Este sistema es el más usado en la actualidad, se caracteriza por ser básicamente posicional. Esto significa que cada dígito del número decimal corresponde a una potencia de 10 y la suma de estas potencias dará como resultado el número representado. Por ejemplo, sea el número $(145)_{10}$, este puede descomponerse en potencias de 10 como se muestra a continuación.

$(145)_{10}$			
Dígito independiente	1	4	5
Potencia de 10	10^2	10^1	10^0

Dependiendo del valor de la potencia, estas pueden representarse como unidades (10^0), decenas $(10)^1$, centenas $(10)^2$, unidades de mil $(10)^3$ o potencias de mayor denominación. Hay que tener en cuenta que las unidades siempre corresponden al último dígito a la derecha del número decimal representado. Al multiplicar cada potencia de 10 por su respectivo dígito independiente se obtiene:

Centenas	Decenas	Unidades
$1*10^2$	$4*10^1$	$5*10^0$
100	40	5

Finalmente, al sumar las unidades, decenas y centenas del número se obtiene el número representado en sistema decimal.

$$100 + 40 + 5 = (145)_{10}$$

Sistema binario

Es un sistema numérico posicional de base dos el cual emplea dos dígitos que corresponden a los números 0 y 1 para la representación de diferentes cantidades contables, es decir es un sistema de base

dos. Un número binario está conformado por la combinación de dos o más de los dígitos 0 y 1 como se presenta a continuación.

$$(10101)_2$$

Como se puede observar, el número está conformado por 5 dígitos con valor uno o cero. En el caso del sistema binario cada dígito se conoce como bit.

El sistema binario es el sistema de numeración empleado en todos los equipos electrónicos digitales desde un reloj despertador hasta un *Smartphone* o un computador personal. Esto se debe a que los equipos electrónicos solamente pueden reconocer dos estados posibles, es decir, el equipo puede estar encendido o apagado, lo que hace que no existan estados intermedios de operación.

Al igual que el sistema decimal el sistema binario es un sistema posicional. Esto significa que un número binario se conforma por la suma de un conjunto de potencias de 2 multiplicadas por cada dígito binario. En el caso del número $(10101)_2$, este se conforma por la suma de potencias establecida como:

$(10101)_2$	1	0	1	0	1
Potencia De 2	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
Bits*potencia	16	0	4	0	1

Finalmente, la suma de los productos individuales de cada bit su la potencia se obtiene el número binario en su representación decimal:

$$16+4+1=(21)_{10}$$

La asignación de las potencias se realiza de la misma forma que el sistema decimal, el último dígito a la derecha tendrá la potencia de menor valor y el último dígito a la izquierda la potencia más alta alcanzada por el número. Respectivamente, estos bits del número binario se conocen como el bit menos significativo y el bit más significativo.

Sistema hexadecimal

El sistema hexadecimal es un sistema de numeración posicional que emplea 16 símbolos para la representación de cantidades contables. Este sistema utiliza como símbolos los diez números del sistema decimal (del 0 al 9) y las seis primeras letras del alfabeto latino A, B, C, D, E y F las cuales corresponden a los números en sistema decimal 10, 11, 12, 13, 14 y 15 respectivamente. Su importancia radica en que permite codificar los números binarios para facilitar su manejo. Al igual que los sistemas de numeración decimal y binario, el sistema hexadecimal es un sistema de numeración posicional. Esto quiere decir que un número hexadecimal está formado por la suma de potencias de 16 multiplicadas por cada dígito hexadecimal. Un número hexadecimal puede ser:

$(4AF)_{16}$ Dígitos	4	A	F
Potencia de 16	$16^2=256$	$16^1=16$	$16^0=1$
dígito*potencia	1024	160	15

Finalmente, al sumar los productos individuales de cada dígito hexadecimal con su potencia se obtiene el número hexadecimal en su representación decimal:

$$1024+160+15=(1199)_{16}$$

Teniendo en cuenta que los sistemas de numeración binario, decimal y hexadecimal comparten elementos comunes, en la tabla 3 se presenta la relación entre los dígitos de los sistemas binario, decimal y hexadecimal para la representación de cantidades numéricas.

Tabla A1. Dígitos de los sistemas de numeración binario, decimal y hexadecimal

Binario	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Conversión entre sistemas de numeración

Los tres sistemas de numeración presentados anteriormente son los más utilizados por los programadores para representar los resultados obtenidos por un algoritmo implementado en un computador. Por lo tanto, se requiere conocer los métodos de conversión entre sistemas de numeración para asegurar una mejor interpretación de los resultados arrojados de un programa.

Sistema decimal a binario

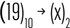
Para convertir datos del sistema decimal al sistema binario, en este libro se abordarán el método de los residuos y el método de las potencias.

Método de los residuos

El método de los residuos es un método general de conversión entre los diferentes sistemas de numeración. Este método propone dividir el número en la base del sistema al cual se desea convertir en un conjunto de divisiones sucesivas hasta que el cociente de la división sea cero. Finalmente, se tomarán los residuos de las operaciones de división sucesiva como los dígitos del número convertido al nuevo sistema de numeración. En el caso del sistema decimal a binario, se tomará el número decimal y se divide entre dos, que corresponde a la base del sistema binario. En el siguiente ejemplo se aplica el método de los residuos para convertir un número decimal a binario.

Ejemplo A1. Conversión de decimal a binario por el método de los residuos

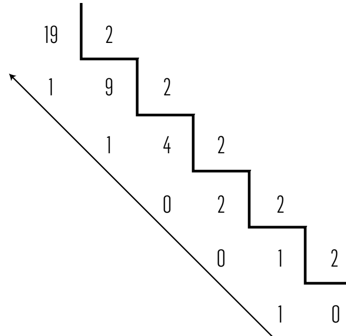
Convertir el siguiente número decimal a binario empleando el método de los residuos.



$$(19)_{10} \rightarrow (x)_2$$

Aplicando el método de los residuos se realiza el conjunto de divisiones sucesivas en la base del sistema de numeración hasta que el cociente de la división sea cero como se muestra en la figura A2.

Figura A2. Método de los residuos para la conversión entre sistemas de numeración



Una vez finalizada la división sucesiva entre dos, se toman los residuos en el sentido de la flecha. Tenga en cuenta que el bit menos significativo corresponderá al residuo de la primera división sucesiva y el bit más significativo al residuo de la última división. De acuerdo con esto, número $(19)_{10}$ en binario corresponderá a:

$$(10011)_2$$

Método de las potencias

El método de las potencias para la conversión entre sistemas de numeración, toma en cuenta que un número se puede formar por la suma de las potencias de la base en la cual desea ser representado.

En el caso de la conversión entre sistema decimal a sistema binario, el método de las potencias establece que se colocaran secuencialmente potencias de 2^n hasta que la ultima potencia sea mayor al número que se desea representar. Finalmente, desde la potencia más grande se asignará el dígito 1 o 0 para asegurar que al final la suma de potencias corresponda al número que se desea representar.

Ejemplo A2. Conversión de decimal a binario por el método de las potencias

Convertir el siguiente número decimal a binario empleando el método de las potencias.

$$(19)_{10}\text{-}(x)_2$$

Inicialmente se establece el conjunto de potencias 2^n para el número$(19)_{10}$. En este caso se establece únicamente hasta la potencia 2^5 ya que esta supera el número que se desea representar.

Potencia de 2	2^5	2^4	2^3	2^2	2^1	2^0
Valor de cada potencia	32	16	8	4	2	1

Una vez definidas las potencias a utilizar, se asigna el dígito 1 o 0 a cada potencia, para que de esta forma la suma del producto de cada potencia multiplicada por el dígito binario correspondiente sea equivalente al número que se desea representar.

Para llevar a cabo este paso de forma sencilla, se parte de la potencia más alta y se observa si al multiplicar el número por el dígito 1 se supera el número a representar. De esta forma se va acumulando el resultado y evaluando si cada potencia supera el valor deseado hasta llegar a la potencia más baja del conjunto. Esta operación puede representarse en forma de sumatoria de la siguiente forma:

$$(0*2^5)+(1*2^4)+(0*2^3)+(0*2^2)+(1*2^1)+(1*2^0)$$

$$(0*32)+(1*16)+(0*8)+(0*4)+(1*2)+(1*1)=(19)_{10}$$

Como se puede observar, en el caso de la potencia 2^5 que equivale al número 32, esta potencia supera el valor deseado a representar, por lo tanto, el dígito binario para la multiplicación será cero. Al pasar a la siguiente potencia de 2^4, esta equivale al número 16, el cual no supera el valor del número a representar, por lo tanto, se coloca el dígito binario 1. Después, se evalúa la potencia $2^3=8$. En este caso el acumulado de la sumatoria va en 16, si a 16 se le suma 8, supera el valor a representar, por lo tanto, esta potencia va con el dígito binario cero. De la misma manera se evalúa para 2^2, 2^1 y 2^0, dando como resultados los dígitos binarios 0, 0, 1 respectivamente.

Finalmente, los dígitos resultantes del proceso son 010011 y corresponderán al número en su representación binaria. Al igual que en aritmética básica el cero a la izquierda no se tiene en cuenta, por lo que el número binario que representa el número $(19)_{10}$ es:

$$(10011)_2$$

Sistema binario a decimal

Para realizar la conversión de sistema binario a decimal se utiliza el método de las potencias descrito en la sección anterior teniendo en cuenta que aquí ya se conocen los dígitos binarios y bastará con multiplicarlos por su potencia correspondiente. De esta manera, la suma de las potencias por el dígito binario arrojará el equivalente decimal del número binario.

Ejemplo A3. Conversión de binario a decimal

Dado el siguiente número binario obtener su representación en sistema decimal.

$$(100111)_2\text{-}(x)_{10}$$

Inicialmente se establecen las potencias que acompañarán a los dígitos binarios. Para esto se parte del bit menos significativo, que es el último bit a la derecha, al cual se le asigna la potencia 2^0 y así sucesivamente hasta llegar al bit más significativo del número binario correspondiente al último bit a la izquierda del número. Para este ejemplo las potencias se establecen de la siguiente manera:

Dígito	1	0	0	1	1	1
Potencia de 2	2^5	2^4	2^3	2^2	2^1	2^0
Valor de cada potencia	32	16	3	4	2	1

Dado que el número binario solo tiene 6 dígitos se asignan las potencias desde la 2^0 hasta la 2^5. Al multiplicar cada dígito por la potencia correspondiente, y sumando los resultados de cada producto, se obtiene el número en su representación en el sistema decimal. En forma de sumatoria, la conversión se representa de la siguiente manera:

$$(1*2^5)+(0*2^4)+(0*2^3)+(1*2^2)+(1*2^1)+(1*2^0)$$

$$(1*32)+(0*16)+(0*8)+(1*4)+(1*2)+(1*1)=(39)_{10}$$

Finalmente, el número $(100111)_2$ en su representación decimal es:

$$(39)_{10}$$

Sistema hexadecimal a binario

Como se puede observar en la tabla 3, los dígitos del sistema de numeración hexadecimal pueden ser representados por un conjunto de 4 bits o dígitos binarios también llamado *Nibble*. De esta forma, para convertir un número hexadecimal en binario basta con reemplazar cada dígito hexadecimal con su equivalente binario de 4 bits.

Ejemplo A4. Conversión de hexadecimal a binario

Convertir el siguiente número hexadecimal a su equivalente binario.

$$(4AF)_{16}\text{-}(x)_2$$

El número hexadecimal está representado por los dígitos 4, A y F, cuyos *Nibble* o representación binaria individual corresponden a 0100, 1010 y 1111 respectivamente. Reemplazando cada dígito hexadecimal por su *Nibble*, la representación en binario del número es:

$$(0100\ 1010\ 1111)_2$$

Sistema binario a hexadecimal

En este caso, el procedimiento de conversión requiere que se agrupen los dígitos binarios en grupos de 4 bits desde el bit menos significativo. De esta forma, al reemplazar los *Nibble* por su equivalente hexadecimal presentado en la tabla 3, se obtendrá el equivalente hexadecimal del número binario.

Ejemplo A5. Conversión de binario a hexadecimal

Convertir el siguiente número binario a hexadecimal.

$$(1001100)_2\text{-}(x)_{16}$$

Tomando como punto de partida el bit menos significativo, se agrupan los bits en grupos de cuatro dígitos:

$$(0100)\text{-}(1100)$$

Como se puede observar, para el *Nibble* de la derecha no hay problema cuenta con 4 bits, mientras que en el caso del segundo *Nibble* solo hay 3 bits en el número. En este caso y teniendo en cuenta que este es el último *Nibble* se agrega un cero a la izquierda para agrupar correctamente el número. De esta forma al buscar el equivalente de los *Nibble* en la tabla 3, el número en sistema hexadecimal estará representado como:

$$(4C)_{16}$$

Sistema decimal a hexadecimal

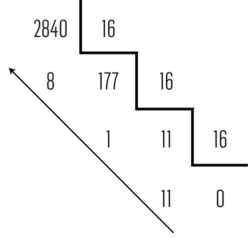
Para convertir números decimales a hexadecimales existen dos métodos. El método directo utilizando los residuos de la división sucesiva en 16 como se muestra en el ejemplo 6 o el método indirecto que primero transforma el número de decimal a binario y finalmente a hexadecimal presentado en el ejemplo 7.

Ejemplo A6. Conversión de decimal a hexadecimal por el método de los residuos

Convertir el siguiente número decimal en hexadecimal empleando el método de los residuos.

$$(2840)_{10}\rightarrow(x)_2$$

Al igual que en la conversión de números decimales a binario, se realiza una división sucesiva en la base del sistema de numeración objetivo. Teniendo en cuenta que es el sistema hexadecimal, se realiza la división sucesiva entre 16 como se muestra a continuación:



Como se puede observar, los residuos de la división corresponden a los números 11, 1 y 8. Con base en la tabla A1 se transforma cada dígito a su equivalente hexadecimal, el cual en el caso del número 11 es el dígito B y para los otros dígitos sigue siendo 1 y 8. Al igual que ocurre con el método de los residuos para la conversión entre decimal y binario, el residuo obtenido cuando el cociente de la división es cero será el primer dígito del número hexadecimal. Los dígitos restantes son ordenados en el sentido de la flecha como se observa en la operación. De esta forma, la representación hexadecimal del número $(2840)_{10}$ es:

$$(B18)_{16}$$

Ejemplo A7. Conversión decimal a hexadecimal por el método indirecto

Convertir el siguiente número decimal en hexadecimal empleando el método de conversión indirecta.

$$(2840)_{10} \rightarrow (x)_2$$

Este método de conversión transforma primero el número de decimal a binario para luego pasarlo de binario a hexadecimal. Aplicando el método de las potencias presentado anteriormente, la representación binaria del número está dada por:

$$(1*2^{11})+(0*2^{10})+(1*10^{9})+(1*10^{8})+(0*2^{7})+(0*2^{6})+(0*10^{5})+(1*10^{4})+(1*2^{3})+(0*2^{2})+(0*10^{1})+(0*10^{0})=(2840)_{10}$$

Una vez obtenida la representación binaria del número, se agrupa en conjuntos de 4 bits o *Nibble* partiendo desde el bit menos significativo. De esta forma la representación del número en hexadecimal es:

$$(B18)_{16}$$

Sistema hexadecimal a decimal

No existe un método directo de conversión entre hexadecimal a decimal, por lo tanto, solo puede aplicarse el método indirecto, es decir, pasar el número hexadecimal primero a binario y luego a decimal.

Ejemplo A8. Conversión hexadecimal a decimal por el método indirecto

Convertir el siguiente número decimal en hexadecimal empleando el método de conversión indirecta.

$$(2AB)_{10} \rightarrow (x)_2$$

Inicialmente, se transforma el número hexadecimal en binario, reemplazando cada dígito por su equivalente binario de 4 bits presentado en la tabla 3, obteniendo como resultado el número en su representación binaria.

$$(0010\ 1010\ 1011)_2$$

Aplicando el método de las potencias al número en binario, el equivalente en sistema decimal es:

$$(0*2^{11})+(0*2^{10})+(1*10^{9})+(0*10^{8})+(1*2^{7})+(0*2^{6})+(1*10^{5})+(0*10^{4})+(1*2^{3})+(0*2^{2})+(1*10^{1})+(1*10^{0})=(683)_{10}$$

Referencias

Atlantic International University. (s.f.) *Definición de programa*. Recuperado de: https://cursos.aiu.edu/Programacion%20de%20Computadoras/PDF/Tema%201.pdf

Asteasuain, F. y Contreras, B. (2002). *Programación orientada a aspectos, análisis del paradigma.* (Tesis de Licenciatura Departamento de Ciencias e Ingeniería de la Computación). Universidad Nacional del Sur. Recuperado de: http://www.angelfire.com/ri2/aspectos/Tesis/tesis.pdf

Cairó, O. (2006). *Fundamentos de programación, piensa en C.* México: Pearson,Prentice Hall.

Cortijo Bon, F. (s.f.). *Curso de C++ Builder. Tratamiento de excepciones*. Universidad de Granada. Recuperado de: http://elvex.ugr.es/decsai/builder/intro/6.html

El Mundo Informático. (2007). *Lenguajes de programación*. Recuperado de: http://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/

Fokker, J. (1996). *Programación funcional*. Universidad de Utretcht. Recuperado de: http://ima.udg.edu/~villaret/fp-sp.pdf

Martínez, F. y Quetglás, G. (2003). *Introducción a la programación estructurada en C*. Valencia: Universitat de Valencia. Servei de Publicacions.

Microsoft Developer Network. (s.f.). *Instrucciones try, throw y catch (C++)*. Recuperado de: https://msdn.microsoft.com/es-co/library/6dekhbbc.aspx

Pozo, S. (2003). *Manejo de excepciones. C++ con Clase*. Recuperado de: http://c.conclase.net/curso/?cap=043

Python Software Foundation. (2014). *Tutorial de Python. Errores y excepciones*. Recuperado de: http://docs.python.org.ar/tutorial/3/errors.html

Reina, A. (2000). *Visión general de la programación orientada a aspectos*. Recuperado de: https://www.lsi.us.es/docs/informes/aopv3.pdf

Rodríguez, C. (2012). *Paradigmas de programación*. Valladolid: Universidad de Valladolid. Obtenido de: http://www.infor.uva.es/~cvaca/asigs/docpar/intro.pdf

Rodríguez, J. (1992). *Tecnología de la programación. Excepciones en C++.* Valladolid: Universidad de Valladolid. Recuperado de: http://www.infor.uva.es/~jmrr/tgp/tgprecurso/excepcio2.htm

Tutorials Point. (s.f.). *Java–Built-in Exceptions*. Recuperado de: http://www.tutorialspoint.com/java/java_builtin_exceptions.htm

Universidad de Carabobo. (s.f.). *Programación basada en objetos.* Recuperado de: http://dptocomp.ing.uc.edu.ve/compAvanzada/lecturas/ProgBasaEnObj_1.pdf

Wikilibros. (2015). *Programación en C++/Excepciones*. Recuperado de: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Excepciones

SU OPINIÓN

Para la Editorial UPB es muy importante ofrecerle un excelente producto.
La información que nos suministre acerca de la calidad de nuestras publicaciones será muy valiosa en el proceso de mejoramiento que realizamos.
Para darnos su opinión, comuníquese a través de la línea (57)(4) 354 4565
o vía e-mail a editorial@upb.edu.co
Por favor adjunte datos como el título y la fecha de publicación, su nombre, e-mail y número telefónico.

Esta obra se publicó en archivo digital
en el mes de septiembre de 2018.

Fundamentos de Programación es un libro pensado para estudiantes que quieran ingresar a la programación de computadores o fortalecer sus conocimientos acerca del tema. Los contenidos presentados en este libro surgen de temáticas y ejercicios desarrollados y validados en los cursos de Lógica de Programación, Lógica y Algoritmia, Ingeniería de Software, entre otras asignaturas de programación impartidas desde la Facultad de Ingeniería de Sistemas e Informática de la Universidad Pontificia Bolivariana, Seccional Bucaramanga.

Como objetivos fundamentales, este texto busca brindar a los estudiantes las herramientas suficientes para interpretar situaciones problema del mundo real o formal, diseñar sus soluciones computacionales a partir de algoritmos, implementar tales soluciones sobre lenguajes de programación como C++, Java y Python, y validar los resultados a partir de conjuntos de prueba.

ISBN: 978-958-764-543-9