

Algoritmos y diagramas de flujo con Raptor

Edgar Danilo Domínguez Vera
Mayra Deyanira Flores Guerrero
Oscar Rangel Aguilar



Buenos Aires • Bogotá • Ciudad de México • Santiago de Chile

Director Editorial:

Marcelo Grillo Giannetto
mgrillo@alfaomega.com.mx

Jefe de Edición:

Francisco Javier Rodríguez Cruz
jrodriguez@alfaomega.com.mx

Datos catalográficos

Domínguez, Edgar; Flores, Mayra; Rangel, Oscar
Algoritmos y diagramas de flujo con Raptor
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 9786075380872

Algoritmos y diagramas de flujo con Raptor

Edgar Danilo Domínguez Vera; Mayra Deyanira Flores Guerrero; Oscar Rangel Aguilar

Derechos reservados © Alfaomega Grupo Editor, S.A. de C.V., México

Primera edición: Alfaomega Grupo Editor, México, septiembre 2017

© 2018 Alfaomega Grupo Editor, S.A. de C.V.

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720. Ciudad de México

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: <http://www.alfaomega.com.mx>

E-mail: atencionalcliente@alfaomega.com.mx

ISBN: 9786075380872

Derechos reservados:

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en todo el mundo.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México – Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443, Oficina 24, Santiago, Chile
Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegagroup.com.ar

Acerca de los autores

Edgar Danilo Domínguez Vera obtuvo, en el 2014, el Doctorado en Filosofía con especialidad en Administración en la Facultad de Contaduría Pública y Administración (FACPyA) de la Universidad Autónoma de Nuevo León (UANL). En 1999 cursó la Maestría en Ciencias de la Administración con especialidad en Sistemas en la Facultad de Ingeniería Mecánica y Eléctrica (FIME) de la UANL. En 1990 egresó de esta misma facultad de la carrera de Ingeniero Administrador de Sistemas (IAS). En el 2005 ocupó la subgerencia de Informática del Comité Administrador del Programa Federal de Construcción de Escuelas.

Mayra Deyanira Flores Guerrero obtuvo, en el 2013, el Doctorado en Educación. Desde el 2007 cuenta con la especialidad en Todóloga en Mecatrónica. En el 2005 cursó la Maestría en Ciencias de la Administración con especialidad en Relaciones Industriales en la Facultad de Ingeniería Mecánica y Eléctrica en la UANL. En el 2002 egresó de esta misma facultad de la carrera de Ingeniero Administrador de Sistemas (IAS). En el 2002 inició su carrera docente en la Facultad de Ingeniería Mecánica y Eléctrica, contribuyó con el desarrollo de los planes de estudio de las carreras de Ingeniero Administrador de Sistemas, Mecatrónica e Ingeniero en Tecnología de Software. En esta misma institución educativa fue Jefa de Academia de Biodispositivos. En la actualidad se desempeña como Jefa de Departamento de Sistemas y es docente en la Facultad de Ingeniería Mecánica y Eléctrica en la Facultad de Ciencias Biológicas de la UANL.

Oscar Rangel Aguilar obtuvo, en el 2015, el Doctorado en Educación en el Instituto José Martí de Monterrey. En el 2005 cursó la Maestría en Ciencias de la Administración con especialidad en Relaciones Industriales en la Facultad de Ingeniería Mecánica y Eléctrica (FIME) de la UANL. En 1991 egresó de esta misma facultad de la carrera de Ingeniero Administrador de Sistemas (IAS). Desde el año 2000 a la fecha ha contribuido a la formación integral de desarrolladores de software en los planes de estudio de las carreras de Ingeniero Administrador de Sistemas e Ingeniero en Tecnología de Software en la FIME-UANL. En esta misma dependencia fue Jefe

de Academia de Software de Base y en la actualidad es Jefe de la Academia de Arquitectura de Computadoras y Redes. Combina su carrera de docente e investigador con su actividad como instructor de la especialidad de Informática en el Centro de capacitación para el Trabajo Industrial núm. 125 (CECATI 125).

Si usted quiere contactar a alguno de los autores, puede hacerlo en los siguientes correos electrónicos, respectivamente: danilo.uanl71@gmail.com, mayradey@hotmail.com y oscar130@hotmail.com.

Dedicatoria

Dedico esta obra a la familia que he dejado de atender por dedicarme a su realización. Al amor de mi vida, Laura Maricela, y al adolescente que tengo el gusto de cuidar y educar, Edgar Belisario.

Edgar Danilo Domínguez Vera

A mis padres, Víctor Flores Gutiérrez y María de Jesús Guerrero Martínez, por estar siempre conmigo en cada etapa de mi vida, por su apoyo total, comprensión y por cuidar con todo su amor y paciencia a Mayita. También agradezco a Dios por tenerlos a mi lado. Asimismo, dedico este libro a mi hija Mayita, pues sin que yo le pida permiso me ha dado de su tiempo, es la personita más importante de mi vida y ha sido siempre mi fuente de inspiración para seguir adelante. Te amo princesa, MAB.

Mayra Deyanira Flores Guerrero

Dedico esta obra a mi amada esposa, María Salome Guevara Montoya, quien es el motor de mi vida y me apoya en cada proyecto que realizo. A mi madre, Socorro Aguilar de Rangel, y a la memoria de mi padre, Jesús Rangel Salas. A mis compañeros autores: Mayra Deyanira Flores Guerrero y Edgar Danilo Domínguez Vera, y a sus familias. A mis compañeros maestros de la Universidad Autónoma de Nuevo León, por los que están y por los que se adelantaron.

Oscar Rangel Aguilar

Agradecimientos

Agradezco a la vida por las interesantes oportunidades que me ha brindado, las cuales me han llenado de experiencias enriquecedoras. Tener mucha experiencia no significa que sepa más que los demás, sino que se ha tenido más oportunidades de equivocarse y sin lugar a duda lo he hecho. Lo interesante de equivocarse es que para reconocer los errores el ser humano tiene que sacar a relucir sus valores para poder seguir adelante. También agradezco a mis compañeros coautores por las enormes aportaciones en cada uno de los capítulos, pues hicieron una revisión minuciosa de cada texto, ejemplo y problema. Saber trabajar en equipo es una virtud que valoro mucho en ellos. Agradezco a mis compañeros maestros que ayudaron en la revisión técnica: Capítulo 1, M.C. Claudia Elisa Luna Mata; Capítulo 2, M.T. Valentín Belisario Domínguez Vera; Capítulo 3, Ing. Pablo Eusebio De León Cepeda; Capítulo 4, M.C. Arturo Del Ángel Ramírez; Capítulo 5, Dr. Francisco Eugenio López Guerrero; Capítulo 6, Dra. Irma Leticia Garza González, y Capítulo 7, Ing. Agustín Cortes Coss e Ing. Misael Antonio Guevara Correa.

Edgar Danilo Domínguez Vera

A Dios, que ha sido mi guía en el transcurso de mi vida y por seguir fortaleciendo mi corazón e iluminar mi mente para que fluyan las ideas, pues esta acción es la responsable de que conociera a personas que han contribuido en mi desarrollo. Al Dr. Edgar Danilo Domínguez Vera, quien me ha brindado la oportunidad de superarme profesionalmente, compartiendo mis conocimientos y experiencias, excelente profesor y amigo, mi admiración. Al Dr. Oscar Rangel Aguilar, mi amigo, quien me ha compartido su experiencia, aprendiendo de él por su apoyo y amistad. A todos los compañeros de la Coordinación de Administración y Sistemas que apoyaron en la realización del libro, por su tiempo, experiencia y opinión, muchas gracias. A mis hermanos Víctor, Griselda, Elisa y Adrián por su apoyo incondicional y estar al pendiente de Mayita y de mí.

Mayra Deyanira Flores Guerrero

A Dios, por permitirme vivir, disfrutar el día a día y terminar este libro.

Oscar Rangel Aguilar

Mensaje del editor

Una de las convicciones fundamentales de Alfaomega es que los conocimientos son esenciales en el desempeño profesional, ya que sin ellos es imposible adquirir las habilidades para competir laboralmente. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos, y de acuerdo con esto Alfaomega publica obras actualizadas, con alto rigor científico y técnico, y escritas por los especialistas del área respectiva más destacados.

Consciente del alto nivel competitivo que debe adquirir el estudiante durante su formación profesional, Alfaomega aporta un fondo editorial que se destaca por sus lineamientos pedagógicos, los cuales coadyuvan a desarrollar las competencias requeridas en cada profesión específica.

De acuerdo con esta misión, con el fin de facilitar la comprensión y apropiación del contenido de esta obra, cada capítulo inicia con una introducción en la que se plantean los antecedentes y una descripción de la estructura lógica de los temas expuestos; asimismo, a lo largo de la exposición se presentan ejemplos desarrollados con todo detalle y cada capítulo concluye con ejercicios de autoevaluación y problemas propuestos.

Además de la estructura pedagógica con que están diseñados nuestros libros, Alfaomega hace uso de los medios impresos tradicionales en combinación con las Tecnologías de la Información y las Comunicaciones (TIC) para facilitar el aprendizaje. Correspondiente a este concepto de edición, todas nuestras obras tienen su complemento en una página web en donde el alumno y el profesor encontrarán un software relacionado con temas específicos de la obra.

Los libros de Alfaomega están diseñados para ser utilizados en los procesos de enseñanza-aprendizaje, y pueden ser usados como textos en diversos cursos o como apoyo para reforzar el desarrollo profesional; de esta forma, Alfaomega espera contribuir a la formación y al desarrollo de profesionales exitosos para beneficio de la sociedad, y espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Contenido

Prólogo	XV
Introducción	XVII

Capítulo 1. Conceptos básicos 1

1.1. Introducción.....	2
1.2. Unidades funcionales de la computadora.....	2
1.3. Definición de algoritmo computacional.....	6
1.4. Acciones computacionales básicas	6
1.5. Código ASCII.....	8
1.6. Proceso del software.....	10
1.7. Enfoque de sistemas aplicado al diseño de algoritmos computacionales	13
1.8. Métodos para el diseño de algoritmos	15
1.9. Expresiones matemáticas	16
1.10. Reglas de prioridad para escribir expresiones matemáticas	17
1.11. Reglas de calidad para dar nombre a identificadores de variables y constantes	19
1.12. Reglas de calidad para escribir expresiones matemáticas	21
1.13. Ejercicios de autoevaluación	22
1.14. Problemas propuestos.....	27

Capítulo 2. Diseño básico de algoritmos 29

2.1. Introducción.....	30
2.2. Raptor y sus símbolos para ejecutar las acciones computacionales básicas.....	33
2.3. Símbolo Output	34
2.4. Símbolo Assignment	37
2.4.1. Asignación a un identificador constante.....	39
2.5. Operadores matemáticos básicos.....	43
2.6. Símbolo Input	47
2.7. Documentación interna en los diagramas.....	51

2.8. Desarrollo de algoritmos básicos	53
2.9. Ejercicios de autoevaluación	61
2.10. Problemas propuestos.	63

Capítulo 3. Diseño de algoritmos con la estructura selectiva 65

3.1. Introducción.	66
3.2. Operadores relacionales y lógicos.	66
3.3. Símbolo Selection	68
3.4. Algoritmos con la estructura selectiva doble.	69
3.5. Deficiencias en los algoritmos.	77
3.6. Anidamiento de la estructura selectiva	88
3.7. Ejercicios de autoevaluación	96
3.8. Problemas propuestos.	98

Capítulo 4. Diseño de algoritmos con la estructura repetitiva 101

4.1. Introducción.	102
4.2. Ciclo controlado por contador.	103
4.2.2 Contadores y acumuladores	109
4.3. Ciclo controlado por centinela.	117
4.4. Ciclo para validación de datos	118
4.5. Ciclos para el manejo de un conjunto de datos similares	124
4.6. Sucesiones numéricas.	128
4.7. Ejercicios de autoevaluación	132
4.8. Problemas propuestos.	133

Capítulo 5. Diseño de algoritmos por módulos 139

5.1. Introducción.	140
5.2. Diseño de algoritmos por módulos sin paso de parámetros	141
5.2.1 Aplicación del enfoque de sistema al diseño modular de algoritmos	146
5.2.2 Refinación del enfoque de sistemas en el diseño modular de algoritmos	153

5.3. Diseño de algoritmos por módulos con paso de parámetros	158
5.4. Ejercicios de autoevaluación	198
5.5. Problemas propuestos.	199

Capítulo 6. Diseño de algoritmos con arreglos de memoria. 201

6.1. Introducción.	202
6.2. Introducción a los arreglos de memoria unidimensionales	203
6.3. Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y sin paso de parámetros .	211
6.4. Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y con paso de parámetros	216
6.5. Introducción a los arreglos de memoria bidimensionales.	227
6.6. Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y sin paso de parámetros . .	238
6.7. Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y con paso de parámetros .	242
6.8. Proceso de datos con corte de control.	254
6.9. Ejercicios de autoevaluación	262
6.10. Problemas propuestos: arreglos de memoria unidimensionales y bidimensionales	263

Capítulo 7. Diseño de algoritmos con archivos de datos . 267

7.1. Introducción.	268
7.2. Almacenamiento de datos en archivos digitales	268
7.3. Lectura de datos desde archivos digitales	273
7.4. Agregar datos a un archivo digital.	279
7.5. Tratamiento de información.	283
7.6. Tratamiento de información con corte de control	297
7.7. Ejercicios de autoevaluación	301
7.8. Problemas propuestos: tratamiento básico de información.	303
7.9. Problemas propuestos: corte de control	307

Bibliografía. 311

Índice analítico. 313

Prólogo

Desde 1994, *The Chaos Report* evidencia los retos que la industria del software enfrenta. Muchos de ellos podrían realizarse si las personas que están en un programa de estudios, cuyo objetivo es formar profesionistas del software, se adentrarán a conocer la disciplina de la Ingeniería del Software.

Es común que las personas que desean cursar el nivel superior evadan las carreras de las Ciencias de la Ingeniería, debido a que implican el estudio y comprensión de un marco teórico con cierto grado de dificultad técnica. Además, en los programas de estudio las materias específicas de la Ingeniería del Software se cursan hasta la mitad de la carrera. De manera que cuando egresan ya han pasado al menos un par de periodos escolares, lo que contribuye al olvido de las bases teóricas.

En un principio, algunos profesores se conformaban con que los estudiantes aprendieran a programar e hicieron a un lado las reglas de calidad para generar un código estandarizado. Ingenuamente, consideraron que cuando estos profesionistas se incorporaran al mercado laboral, se ajustarían sin replicar a los estándares de calidad. Cuán lejos estaban de esa realidad. Se les enseñó que programar es tan flexible y amplio como el arte; por ello, después se les dificultó cumplir con los límites que requieren los modelos de calidad como el *Capability Maturity Model Integration* (CMMI), el *Personal Process Software* (PSP) o el *Team Process Software* (TSP), entre otros.

Las investigaciones muestran una renuencia de algunos profesionistas del software para ajustarse a esos estándares de calidad, quizá porque desconocen la relación que hay entre los costos de desarrollo del software y el adecuado apego a esos estándares. Para ellos sólo importa entregar un software funcional lo más pronto posible al cliente o usuario y no consideran los costos de desarrollo y mantenimiento a corto, mediano y largo plazo.




Consideramos indispensable, desde los primeros cursos de programación, enseñar a los estudiantes el cumplimiento de estándares de calidad. En este libro, el lector no solo aprenderá una herramienta automática de diseño de algoritmos, sino que también se le guiará para que adopte ciertas reglas de calidad que posteriormente podrá trasladar a la codificación de un algoritmo.

Las reglas de calidad se orientan a generar un código estandarizado y utiliza las cuatro notaciones ampliamente usadas: Húngara, Camello, Pascal y Guión Bajo. Aunque en este texto no se enseña a codificar, sí se muestra cómo pueden aplicarse estas notaciones cuando el programador tiene que dar nombre a los diversos identificadores, desde variables, constantes, módulos y escritura de fórmulas aritméticas, etcétera.

El software de Raptor ayuda a desarrollar la habilidad creativa de algoritmos computacionales orientados a la solución de problemas; además, es un software con el que el lector puede probar la eficacia del algoritmo creado. El brinco para convertir el algoritmo en un programa es más fácil y con menos incertidumbre.

En este libro, al programador novato se le lleva de la mano para que fácilmente adquiera las competencias para el desarrollo de un software. Y al programador intermedio se le consolida en la programación por módulos con pasos de parámetros, en el almacenamiento de datos en memoria externa, en el proceso de datos en archivos digitales y en la presentación de reportes para la toma de decisiones utilizando el corte de control.

No queremos dejar pasar la oportunidad de agradecer a Alfaomega Grupo Editor su apoyo en la publicación de esta obra. La persona que realizó la edición hizo un excelente trabajo.

Mucho nos gustaría recibir sus comentarios con respecto a este libro en los siguientes puntos de contacto: Mensajería instantánea:  81-1301-1461. Correo electrónico: danilo.uanl71@gmail.com. Redes sociales:  @danilouanl y  /danilo.dominguez.

Edgar Danilo Domínguez Vera
Mayra Deyanira Flores Guerrero
Oscar Rangel Aguilar

Introducción

Pertenecemos a la generación que conocemos la vida antes y después de que la tecnología de información y comunicación fuera parte integral de nuestras actividades cotidianas y laborales.

Aprender a hacer programas computacionales no debe ser una habilidad exclusiva de los desarrolladores de software, sino de todo profesional que desea resolver problemas mediante el proceso de datos de toda clase de sistemas. Antes de que una persona aprenda a programar, es recomendable que conozca los métodos para el diseño de algoritmos computacionales, de entre los cuales destaca el diagrama de flujo.

La metodología de diagramas de flujo cuenta con un conjunto de símbolos gráficos que representan las acciones computacionales, entre las más comunes son la lectura de datos, la impresión de resultados, el proceso de datos mediante la ejecución de operaciones aritméticas, la ejecución de acciones mediante selección alternativa y la ejecución repetitiva de acciones computacionales, entre otras.

Los símbolos gráficos se conjugan con una secuencia lógica que permiten solucionar problemas mediante la realización de cálculos matemáticos repetitivos que pueden procesar grandes cantidades de datos. Al diagrama de flujo que resuelve un problema se le llama algoritmo computacional y a la escritura o creación de algoritmos se le llama diseño de algoritmos. Los algoritmos se pueden transformar en un programa computacional usando un software que interpreta y ejecuta la sintaxis de un lenguaje de programación.

Raptor es un software o herramienta que permite diseñar algoritmos computacionales. Dado que una vez que se escribe un algoritmo en Raptor, éste puede interpretarse y ejecutarse permitiendo conocer su eficacia para resolver un problema; entonces, puede decirse que Raptor es un lenguaje gráfico de programación.

Anteriormente, cuando no existían herramientas como el Raptor, el programador solo podía probar la eficacia de un algoritmo utilizando la prueba de escritorio que era un proceso manual y tedioso. Al final de cuentas, casi siempre quedaba la incertidumbre de su eficacia hasta que ese algoritmo se convertía en programa y se hacían las pruebas pertinentes.

Un curso como Raptor, antes de que los estudiantes conozcan algún lenguaje de programación, es altamente recomendable porque les permite visualizar de manera gráfica como se construye un programa computacional. De hecho, se puede decir que un diagrama de flujo es la representación gráfica de un programa.

Después de más de 25 años de contribuir a la formación integral de desarrolladores de software, no nos queda duda de que Raptor es una herramienta que facilita el aprendizaje a quienes inician en el mundo de la programación de computadoras.

Con este libro se pretende contribuir a uno de los retos de la industria del software, que es medir e incrementar la productividad de los desarrolladores. Se han tomado las actividades que pertenecen a la metodología de Proceso de Software Personal (PSP) desarrollada por Watt S. Humphrey; además del marco de trabajo propuesto por Roger S. Pressman para el proceso de software, que son: comunicación, planeación, modelado, construcción y despliegue. En este libro se han agregado las reglas de calidad tomando como referencia las sugerencias mayormente hechas por Deitel.

Nota: A lo largo de este libro, el lector observará el uso de muchas palabras que deberían ir acentuadas, pero que no lo están. Hacemos la aclaración que no es un error ortográfico, lo que sucede es que en programación se usan palabras como identificadores a las cuales por motivos de mantenibilidad del software se recomienda no acentuar. Agradecemos su comprensión por esta situación.

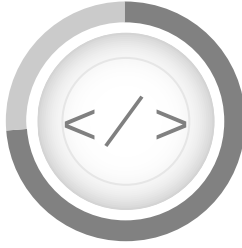
Plataforma de contenidos interactivos

Para tener acceso al material de la plataforma de contenidos interactivos de este libro, siga los siguientes pasos:

Ir a la página: <http://libroweb.alfaomega.com.mx>

Ir a la sección Catálogo y seleccionar la imagen de la portada del libro, al dar doble clic sobre ella, tendrá acceso al material descargable.

Nota: Se recomienda respaldar los archivos descargados de la página web en un soporte físico.



Capítulo 1

Conceptos básicos

- 1.1 Introducción
- 1.2 Unidades funcionales de la computadora
- 1.3 Definición de algoritmo computacional
- 1.4 Acciones computacionales básicas
- 1.5 Código ASCII
- 1.6 Proceso del software
- 1.7 Enfoque de sistemas aplicado al diseño de algoritmos computacionales
- 1.8 Métodos para el diseño de algoritmos
- 1.9 Expresiones matemáticas
- 1.10 Reglas de prioridad para escribir expresiones matemáticas
- 1.11 Reglas de calidad para dar nombre a identificadores de variables y constantes
- 1.12 Reglas de calidad para escribir expresiones matemáticas
- 1.13 Ejercicios de autoevaluación
- 1.14 Problemas propuestos

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Entender los conceptos básicos de la computación y la informática que le permitan diseñar algoritmos computacionales.
- Comprender las principales acciones computacionales que se utilizan para diseñar algoritmos computacionales.
- Aplicar el enfoque de sistemas a una metodología para el diseño de algoritmos computacionales.
- Transformar una fórmula matemática en una fórmula computacional.
- Otorgar nombres a los identificadores de un algoritmo computacional siguiendo reglas de calidad.



1.1 Introducción

Parece ser que no hay actividad humana que esté exenta de verse involucrada con las Tecnologías de la Información y la Comunicación (TIC). Comprender los conceptos y principios básicos de la informática y computación es necesario para aplicarlos en un ambiente laboral, pues ahora ya está reconocido como un conocimiento básico que permite la equidad e inclusión social.

Combatir el analfabetismo y la brecha digitales es una prioridad para quienes buscan mejores niveles de competitividad y productividad, pues el ejercicio pleno de los derechos humanos se ve limitado cuando una persona desconoce los conceptos básicos de la informática y computación.

No sólo las personas que se dedicarán al desarrollo de software deben contar con estos conocimientos básicos, sino cualquier individuo y más los profesionistas que se insertarán a la población económicamente activa. Los medios de comunicación especializados pronostican la necesidad de contar con técnicos y profesionistas altamente capacitados en las tecnologías de información y comunicación. Hago votos para que este texto sea una puerta inicial para entrar al mundo de las ciencias de la computación.



1.2 Unidades funcionales de la computadora

Entendemos por **tratamiento de información** el conjunto de acciones que permiten el procesamiento de datos, mediante las operaciones de lectura, escritura, copia, transmisión, ordenación, clasificación, comparación, archivo, cálculo, traducción, análisis y síntesis. La **informática** es la ciencia que se ocupa de estudiar el tratamiento de información por medio de la utilización de dispositivos electrónicos diseñados para ejecutar las acciones del procesamiento de datos.

A todo dispositivo electrónico diseñado para ejecutar las operaciones de tratamiento de información se le puede llamar computadora. Hay una amplia gama de dispositivos

electrónicos que cumplen con esas características y que la costumbre popular suele denominar con otro nombre, como es el caso de las tabletas electrónicas, los teléfonos inteligentes, las laptops, etc. Varias de esas denominaciones están ligadas a una marca registrada, como es el caso de las *ipad* o el *iphone* con la marca de la empresa Apple. Todos esos dispositivos cumplen con las características de una computadora.

La computadora es un dispositivo electrónico o hardware que se utiliza para el procesamiento de datos, con el objetivo de transformarlos en información que sea útil en la toma de decisiones. Además, está diseñada para presentar al usuario elementos que faciliten la aprehensión o asimilación de la información para una mejor toma de decisiones; tales elementos se presentan en forma de cálculos, reportes, resúmenes, síntesis, tablas, gráficas, imágenes, etcétera.

Por su rapidez, eficacia y confiabilidad, las computadoras, aunadas a las tecnologías de la información y la comunicación, son parte de la columna vertebral de prácticamente todas las actividades que realiza el ser humano. Sin embargo, requieren de programas computacionales o software para que realicen acciones útiles en apoyo a dichas actividades.

De manera conceptual, la computadora cuenta con seis unidades funcionales:

1. Unidad central de proceso, que a su vez se subdivide en la unidad aritmético-lógica y la unidad de control.
2. Unidad de memoria.
3. Unidad de entrada.
4. Unidad de salida.
5. Unidad de almacenamiento secundario.
6. Bus del sistema (véase figura 1.1).

Para que las unidades funcionales de la computadora puedan trabajar, se requiere de un conjunto de programas o software que se denomina sistema operativo.

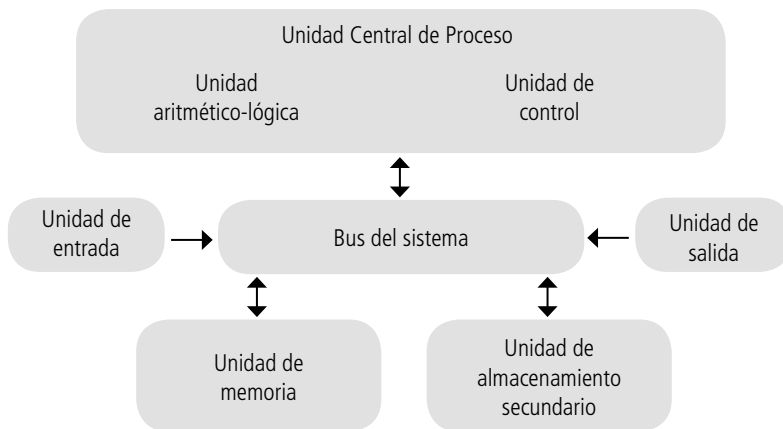


Figura 1.1 Unidades funcionales de la computadora.

Para comprender mejor la diferencia entre hardware (computadora) y software, tomemos como referencia la idea religiosa más conocida en nuestra cultura occidental: el cuerpo y el alma espiritual. Podemos decir que el cuerpo de una persona es el hardware y el alma espiritual es el software. El hardware es tangible por los sentidos humanos y el software es intangible.

Diversos autores han propuesto clasificar el software en varias categorías, tomando como referencia sus características y funcionalidades. Por el momento, consideraremos las dos más conocidas, que son el software de base, también llamado software de sistemas, y el software de aplicación. El sistema operativo está clasificado como software de base, que consiste en un conjunto de programas caracterizado por su gran interacción con el hardware, entre ellos el sector inicio o arranque; lo que posibilita que la computadora pueda ser utilizada por una persona. El software de aplicación es un programa o conjunto de programas que permiten realizar las operaciones del tratamiento de información, desde el procesamiento de datos, el procesamiento de transacciones y la generación de información para la toma de decisiones.

Para que una persona sea capaz de crear software es conveniente que conozca cómo trabaja cada una de las unidades funcionales de la computadora.

1. **Unidad central de proceso.** La unidad central de proceso o *Central Processing Unit* (CPU) se subdivide en la unidad aritmético-lógica o *Arithmetic-Logic Unit* (ALU) y la unidad de control o *Control Unit* (CU).
 - a) **Unidad aritmético-lógica** ejecuta operaciones matemáticas y lógicas para procesar los datos y obtener información. La suma, resta, multiplicación y división, así como *and*, *or* y *not*, son algunas de las operaciones que esta unidad puede realizar.
 - b) **Unidad de control** es la encargada de administrar los recursos o unidades con los que cuenta la computadora. Esta unidad interpreta las instrucciones generadas por un programa e inicia las operaciones apropiadas para llevarlas a cabo.
2. **Unidad de memoria.** La unidad de memoria de una computadora requiere de un suministro de energía permanente para funcionar, almacenar, conservar y acceder a información digital. La unidad de memoria de la computadora se subdivide en la memoria de acceso aleatorio o *Random Access Memory* (RAM) y la memoria de sólo lectura o *Read-Only Memory* (ROM).
 - a) La **RAM** también es conocida como memoria primaria, principal o interna; está construida por circuitos integrados. Su capacidad de almacenamiento y conservación está limitada en cuanto a espacio y a su conexión permanente a una fuente de energía. Cuando esa fuente se suspende, toda la información que estaba allí almacenada se pierde, por eso se dice que es volátil. Debido a estas limitaciones, es común que los usuarios usen métodos alternos de almacenamiento en un formato con características más seguras y permanentes.

- b) La **ROM** sólo permite la lectura de la información y no su modificación. La información digital almacenada en la ROM no se pierde cuando se suspende la fuente de energía. Sin embargo, requiere de la fuente de la energía para funcionar y acceder a los datos. Las tareas encargadas a los programas grabados en la ROM son para la gestión del proceso de arranque de la computadora, el chequeo inicial del sistema, la carga del sistema operativo y las diversas rutinas de control de dispositivos de entrada/salida. Estos programas o utilidades forman parte del Sistema Básico de Entrada y Salida o *Basic Input Output System* (BIOS).
3. **Unidad de entrada.** Todo dispositivo electrónico que permite la introducción de datos a la unidad de memoria de una computadora se le conoce como unidad de entrada. Dicha introducción de datos se hace mediante puertos o dispositivos diseñados con ese objetivo, los más comunes son el teclado, el *mouse* o ratón, digitalizador de imágenes o *scanner*, lector de tarjetas electrónicas, lector de código de barras, monitor sensible al tacto o *touch screen monitor*, etcétera.
4. **Unidad de salida.** Todo dispositivo electrónico que permite presentar al usuario el estado de los datos que se encuentran en la unidad memoria de la computadora es una unidad de salida. El monitor tradicional o la impresora son algunos de los que podemos mencionar.
5. **Unidad de almacenamiento secundario.** Todo medio electrónico que permite el almacenamiento y conservación de video, voz, datos, sonidos, texto, etc., en formato digital, sin que requiera estar permanentemente conectado a una fuente de energía, se le denomina **memoria auxiliar, externa o periférica**. A esta categoría pertenecen los discos duros, las cintas magnéticas, los discos externos, las tarjetas de memoria flash, entre otros; a los cuales también se les conoce como **soporte o medios de almacenamiento**. Estos medios tienen diversas capacidades de almacenamiento y costo, por lo que se usa uno u otro dependiendo de la cantidad de información que se va a almacenar.
6. **Bus del sistema.** Las cinco unidades funcionales explicadas anteriormente se pueden “comunicar” entre sí gracias al bus del sistema. Físicamente es un conjunto de cables y se encuentra separado en tres canales, que son el bus de datos, el bus de direcciones y el bus de control.
- a) **Bus de datos.** Es bidireccional y es el canal por medio del cual se conducen los datos entre la CPU y los demás dispositivos (memorias, puertos y otros).
- b) **Bus de direcciones.** Es un canal unidireccional por medio del cual la CPU envía las direcciones de memoria para ubicar información en los dispositivos de memoria, puertos u otros dispositivos de la computadora.

- c) **Bus de control.** Al igual que el bus de direcciones, es unidireccional y se utiliza para efectuar la lectura y escritura en las memorias y puertos de entrada-salida. En general, este bus lo emplea la CPU para controlar el flujo de los datos y las direcciones de forma organizada.

Conforme la tecnología ha ido avanzando, dispositivos que tradicionalmente eran sólo de salida, como el monitor, ahora también pueden ser de entrada, como el monitor sensible al tacto. De manera que este tipo de dispositivos ahora son de entrada-salida. Obvia decir que la mayor parte de los discos digitales o tarjetas de memoria flash también desempeñan el papel doble de ser medios de entrada y salida. Esto depende de la capacidad del medio o dispositivo para introducir datos a la memoria de la computadora (entrada) y almacenar información digital (salida).



1.3 Definición de algoritmo computacional

La **ingeniería de software** es la disciplina que aglutina métodos y herramientas que coadyuvan a la construcción de software con calidad. Al conjunto de actividades que permiten desarrollar software, *Roger S. Pressman* le llama **proceso de software**. Un algoritmo computacional puede convertirse en un programa computacional o software, utilizando métodos y herramientas de esta disciplina.

Las personas profesionales que se dedican a la construcción de programas computacionales se les llama **ingeniero de software** o más coloquialmente **programador**. Para ser un ingeniero de software eficaz es esencial aprender a resolver problemas de modo estructurado y sistemático, estableciendo una secuencia lógica llamada **algoritmo computacional**.

Así como el arquitecto o ingeniero civil utiliza planos para plasmar gráficamente la propuesta para la construcción de una obra civil o edificación, el ingeniero de software escribe la solución de un problema redactando las acciones computacionales o dibujando su representación gráfica, a los cuales se les denomina algoritmo computacional. Además, es conveniente estimar los recursos económicos, el tiempo, personal y materiales requeridos que serán utilizados durante la construcción del software.

Hay que distinguir entre algoritmo matemático y algoritmo computacional. El primero es un conjunto de pasos o procedimientos que deben seguirse para resolver un problema aritmético. Los algoritmos matemáticos que el ser humano empieza a conocer en los primeros años de su vida son la suma, la resta, la multiplicación y la división. El segundo es un conjunto de acciones computacionales que deben ejecutarse con una secuencia lógica para resolver un problema mediante operaciones propias del tratamiento de información.



1.4 Acciones computacionales básicas

Saber diseñar algoritmos computacionales es esencial en la construcción de software porque representa la creatividad y la habilidad para solucionar un problema, mediante

las operaciones del tratamiento de información. Un algoritmo computacional permite obtener los datos que se desean conocer a partir de datos previamente conocidos; a esto se le conoce como transformar datos en información.

Los algoritmos computacionales son independientes tanto del lenguaje de programación que se utilice para convertirlo en software, como de la computadora que los ejecuta. El **lenguaje de programación** es un medio que permite concretizar el algoritmo, y la **computadora** es el dispositivo que permite ejecutar las acciones computacionales establecidas en el algoritmo.

Si un algoritmo computacional se transforma en un programa, utilizando varios lenguajes de programación, entonces tendremos varios programas, diferentes entre sí, que hacen lo mismo, pero el algoritmo es el mismo.

En la creación de algoritmos computacionales no hay reglas estrictas porque no se trata de aplicar una metodología rígida e inmutable. Por el contrario, se aplica un método flexible y adaptable. Es inútil tratar de crear algoritmos por memorización, pero contar con ejemplos es de mucha ayuda para desarrollar en las personas la lógica computacional. La **creación de algoritmos** no es una ciencia exacta, aquí no hay soluciones únicas, un mismo problema puede ser resuelto por diferentes algoritmos.

Mucho tiempo se debatió si la programación de computadoras era arte o ciencia. Actualmente, hay un consentimiento generalizado de que es ambas cosas. Es arte porque requiere de la creatividad volitiva de las personas. Es ciencia porque trata de entender la realidad mediante el análisis de causas y efectos, y procura desarrollar una solución con el uso de técnicas, métodos y herramientas ya existentes, bajo un enfoque de sistemas.

Las acciones computacionales básicas para construir algoritmos computacionales son:

1. La **lectura de datos** es la asignación o inserción de datos en la memoria RAM de la computadora. La diferencia entre asignación e inserción es la utilización o no de un dispositivo de entrada, como el teclado, para colocar el dato en la memoria RAM.
2. La **impresión de resultados** es permitir que el usuario observe, en el monitor de la computadora o en papel, el estado o valor de los datos que están almacenados en la memoria RAM.
3. El **proceso de datos** es la ejecución de operaciones aritméticas que permiten obtener o calcular datos desconocidos a partir de datos previamente conocidos.
4. La **selección de alternativas** de acciones computacionales es la posibilidad de configurar diferentes opciones mutuamente excluyentes, que se pueden elegir a partir de condiciones matemáticas; éstas se plantean con operadores lógicos y relacionales.
5. El **ciclo** es la posibilidad de configurar una opción repetitiva y una opción diferida; estas opciones no son mutuamente excluyentes. La opción repetitiva consiste en que una o más acciones computacionales puedan ejecutarse en la

misma secuencia y de manera iterativa, bajo un ambiente controlado, para que de manera eventual deje de dar vueltas, impidiendo generar ciclos infinitos. La ejecución o no de la opción repetitiva se plantea a partir de condiciones matemáticas con el uso de operadores lógicos y relacionales. La opción diferida consiste en una o más acciones computacionales cuya ejecución se posterga para después de que el ciclo haya llegado a su fin.

La sintaxis de un lenguaje de programación es el conjunto de reglas que permiten escribir software. Un algoritmo computacional se transforma en software haciendo uso de la sintaxis; a esta actividad se le llama **codificación**. Por tanto, podemos afirmar que un algoritmo codificado es un programa computacional o software.

La **programación de computadoras** es la habilidad intelectual de analizar e interpretar un problema; llevar ese problema a un nivel de abstracción tal que permita desarrollar una solución mediante un algoritmo computacional, que posteriormente pueda convertirse en un código que sea interpretable y ejecutable por una computadora.



1.5 Código ASCII

A cada símbolo que forma parte de los alfabetos que permiten la comunicación entre las personas se le llama **carácter**. Estos símbolos o caracteres se combinan entre sí para conformar palabras que tienen una carga social y un significado en cada cultura o comunidad. Por tanto, una palabra es un conjunto de caracteres que tiene un significado denotativo y connotativo. Los significados procuran definirse mediante instituciones especializadas, como la Real Academia Española. Un conjunto de palabras y símbolos de expresión permite la existencia de un idioma, lengua o dialecto.

Los **números** son el idioma de los equipos de Tecnología de la Información y la Comunicación. Para que estos dispositivos electrónicos puedan comunicarse entre sí es necesario que los caracteres se conviertan en sus representaciones numéricas.

En la década de 1960, la necesidad de estandarización llevó a la creación del Código Estándar Americano para el Intercambio de Información (*ASCII o American Standard Code for Information Interchange*). La tabla del ASCII contiene 128 caracteres con su correspondiente número asignado. El código ASCII permite que los equipos TIC puedan comunicarse entre sí.

La tabla 1.1 muestra únicamente los caracteres imprimibles del ASCII. Se le sugiere buscar en Internet la tabla de caracteres ASCII de control no imprimibles y la tabla de caracteres ASCII extendido. Si durante la edición de un texto desea insertar un símbolo que no esté disponible en su teclado, insértelo mediante el código ASCII de la siguiente manera: mantenga presionada la tecla Alt, mientras escribe el número decimal equivalente.

Por ejemplo, para insertar el símbolo de arroba @, mantenga presionada la tecla Alt mientras escribe 64 en el teclado numérico, según se muestra en la tabla 1.1. Tenga la precaución de usar el teclado numérico para escribir los números, no el teclado

normal. Asegúrese de que la tecla Bloq núm o *Caps lock* esté activada, si es que su teclado lo requiere para escribir los números con el teclado numérico.

Tabla 1.1 Caracteres ASCII imprimibles

Número decimal	Carácter	Número decimal	Carácter	Número decimal	Carácter
32	espacio	65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
35	#	68	D	100	d
36	\$	69	E	101	e
37	%	70	F	102	f
38	&	71	G	103	g
39	'	72	H	104	h
40	(73	I	105	i
41)	74	J	106	j
42	*	75	K	107	k
43	+	76	L	108	l
44	,	77	M	109	m
45	-	78	N	110	n
46	.	79	O	111	o
47	/	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	:	91	[123	{
59	;	92	\	124	
60	<	93]	125	}
61	=	94	^	126	~
62	>	95	_	127	SUPR
63	?	96	`		
64	@				

Ya son varias las ocasiones en que se ha hecho referencia al concepto de procesamiento de datos, pero en ningún momento hemos dado la definición de lo que es el dato.

Con toda intención, se ha dejado la definición formal de lo que debe entenderse por dato hasta que el lector tenga conocimiento del papel que desempeña el código ASCII en la disciplina de la informática.

Un **dato** es un carácter o conjunto de caracteres que tienen un significado dentro de un contexto en particular. Desde el punto de vista de la informática, los datos, según sus características, pueden ser **numéricos**, **alfabéticos** o **alfanuméricos**. Los datos son numéricos cuando los caracteres que los componen son dígitos, permitiendo solamente un punto para señalar que hay decimales. Los datos son alfabéticos cuando los caracteres que los componen son exclusivamente letras del alfabeto. Los datos son alfanuméricos cuando los caracteres que los componen son una combinación de cualquier carácter disponible en el código ASCII.

Para que un dato sea considerado como tal es necesario que tenga alguna relación con alguna entidad o esté dentro del contexto de algún tema o situación, de lo contrario se vuelve un dato que no tiene significado o está vacío semánticamente hablando.

Los datos que el tratamiento de información estudia son aquellos que tienen la capacidad de procesarse para generar información a partir de datos conocidos. Una herramienta que permite el procesamiento de datos es la computadora y una metodología para automatizar dicho procesamiento es la programación de computadoras mediante algoritmos computacionales.



1.6 Proceso del software

Para Roger S. Pressman, el **proceso del software** es un conjunto de actividades, acciones y tareas que deben realizarse para crear, construir o desarrollar software. Consta de cinco actividades, que son: comunicación, planeación, modelado, construcción y despliegue.

Una actividad busca lograr un objetivo general y se realiza sin importar las características estructurales del producto o software que se construirá, como son el dominio de la aplicación, el tamaño del proyecto, la complejidad del esfuerzo o grado de rigor que se aplicará para construir el software. Una **acción** es un conjunto de tareas que permiten producir un producto de trabajo importante que forma parte de las características estructurales del software que se va a construir. Una tarea se centra en un objetivo pequeño, pero bien definido, que produce un resultado tangible y que ayuda a construir o definir los productos de trabajo de cada actividad.

El objetivo de crear software es resolver problemas o permitir hacer una actividad que sea útil para alguien, mediante el uso de un equipo computacional. En este libro, la mayoría de los problemas a resolver por la creación de software implica la transformación de datos conocidos en información que se desea conocer. Lo primero que necesitamos analizar es si los datos que conocemos son suficientes para generar los que se desean conocer. Por lo general, la transformación de los datos se logra con fórmulas matemáticas que permiten procesar los datos conocidos para obtener la información desconocida. Esto permite decir que la información está conformada por datos procesados.

Las computadoras no tienen la capacidad del ser humano del pensamiento o razonamiento; para que realicen un trabajo útil debemos proporcionar una serie de instrucciones que en su conjunto forman un algoritmo computacional.

Las actividades del proceso del software aquí propuesto son una combinación de las ideas de Roger S. Pressman y Watts Humphrey. Para desarrollar software y resolver un problema por programación de computadoras realice las siguientes actividades.

1. **Comunicación.** El objetivo y principal producto de trabajo que se genera con esta actividad es la definición del alcance del software, en la que se determinan las características y funcionalidades que tendrá o debe tener, esto de mutuo acuerdo entre el cliente o usuario y equipo de desarrollo constituido por ingenieros de software. El **cliente o usuario** son las personas que están interesadas en que se construya un software.

De esta manera, toda acción que permite reunir información que ayude a definir el alcance del software es parte de la actividad de comunicación. El conjunto de acciones a realizar que forman parte de la actividad de comunicación dependen de las características estructurales del producto o software que se construirá. Por lo que el conjunto de acciones puede variar de un proyecto a otro.

La identificación, definición y delimitación del problema, así como la definición de las estrategias que se utilizarán para reunir la información, el diagnóstico y contexto de la persona u organización que usará el software son tan solo algunos ejemplos de las acciones que pueden formar parte de la comunicación.

La aplicación de entrevistas y cuestionarios a los clientes o usuarios son ejemplos de tareas que forman parte de la acción de identificación, definición y delimitación del problema. La tarea de investigación de la cultura organizacional es otro ejemplo que forma parte de la acción que permite hacer un diagnóstico de una organización.

La identificación, definición y delimitación del problema, y su eventual solución, se logran teniendo comunicación con las personas que desean resolverlo. Es indispensable entender exactamente en qué consiste y el resultado al cual se quiere llegar. Es fundamental conocer y delimitar el problema.

La comunicación permite establecer los requerimientos del problema, entenderlo claramente, así como determinar con precisión los requisitos para su solución. Se debe descartar todo lo que no es importante y llegar a la raíz del mismo, y si después de esto el problema no queda totalmente definido y delimitado, se debe pedir más información a las personas que quieren resolverlo. En este caso es útil hacer un análisis mediante un enfoque de sistemas, el cual permite identificar los datos conocidos, los datos que se desea conocer y cómo pueden procesarse para conocerlos.

2. **Planeación.** Las acciones de esta actividad se encaminan a generar los productos de trabajo, como las estimaciones del proyecto, el seguimiento y control del proyecto, la administración de los riesgos, las estrategias de aseguramiento de la calidad, el control de la configuración del software, el

presupuesto económico y el tiempo que se invertirá en construir el software. La planeación también incluye determinar la lista y disposición de los requerimientos humanos, materiales y tecnológicos, la agenda pormenorizada de las acciones y tareas con su respectiva ventana de tiempo. La mayoría de los problemas se pueden resolver de más de una forma, por lo que es necesario pensar y planear diferentes alternativas que permitan resolverlo.

En la planeación deben definirse los productos de trabajo, las acciones y tareas a realizar. Un producto de trabajo es aquel elemento que forma parte de la estructura arquitectónica del software, por ejemplo, el alcance del software o el diseño del mismo. Una acción es el conjunto de tareas que producen un producto de trabajo importante. Una tarea se centra en un objetivo pequeño, pero bien definido, que produce un resultado tangible como las pruebas del software.

En esta etapa se deben considerar las restricciones del problema, como pueden ser los valores válidos que pueden tomar los datos conocidos, es decir, cuál es su dominio; por ejemplo, el dominio numérico para la calificación de un examen sólo puede ser de cero a cien, cualquier otro valor no es válido. Asimismo, se debe identificar el tipo de datos o su forma, es decir, si son numéricos, alfabéticos o alfanuméricos.

Durante la planeación se debe estimar la longitud final del software que se va a construir, y el tiempo, costo y esfuerzo que se requerirán para concluirlo. En esta etapa hay que obtener una serie de datos de prueba que serán alimentados al software, y se compararán los datos esperados con los observados.

3. **Diseño.** Consiste en crear o desarrollar una lista de pasos o acciones computacionales a seguir para la solución del problema, lo cual permitirá tener un modelo llamado algoritmo computacional. El diseño suele ser una representación gráfica del software que se va a construir. Se puede implementar mediante dos opciones: el diagrama de flujo y el pseudocódigo; en ambos casos, se refiere a la secuencia lógica en la que se deben ejecutar las acciones computacionales que permiten resolver el problema. Escribir el algoritmo es la parte más difícil del proceso de solución de problemas.
4. **Revisión del diseño.** Una vez que se tenga el algoritmo computacional hay que verificar que sea correcto y que resuelva el problema en la forma que se intenta. Diseñar algoritmos en una herramienta automática como Raptor es útil porque permite revisar y probar la eficacia del diseño.
5. **Codificación.** Consiste en transformar el algoritmo computacional en un programa computacional. Esto se hace utilizando una herramienta o software especial que permite la escritura de código, pues cuenta con un editor de textos y un conjunto de reglas que son parte de su sintaxis. También, se requiere conocer un lenguaje de programación y su sintaxis, ya que cada acción

computacional del algoritmo se convierte a una o varias líneas de código en el lenguaje seleccionado.

6. **Revisión de la codificación.** Todos los lenguajes de programación tienen sus propias reglas, las cuales deben respetarse para no generar errores; éstos se pueden clasificar en errores de sintaxis, de ejecución y de lógica.
7. **Compilación.** Una vez que se ha terminado de codificar el algoritmo, el programa computacional se debe compilar. La **compilación** es el proceso por medio del cual se verifica que el programa esté escrito, respetando las reglas o sintaxis del lenguaje. Cuando esto no sucede así, se deben corregir los errores hasta que desaparezcan por completo. A éstos se les llama errores de sintaxis. La mayoría del software creado para permitir la codificación de programas cuenta con una opción de compilación o verificación automatizada de la sintaxis. Cuando un código no tiene errores de sintaxis, el compilador genera un programa ejecutable.
8. **Prueba.** También llamada **prueba unitaria**. En esta etapa el programa se ejecuta para saber si funciona correctamente. Durante la prueba todavía pueden aparecer errores de ejecución o de lógica, los cuales deben ser corregidos. Se debe tener a la mano un grupo de valores de prueba que generan resultados esperados para comparar éstos con los resultados que arroja el programa.

En la actualidad hay herramientas que generan de manera automática el código a partir de un algoritmo computacional. Sin embargo, ya sea que el código se obtenga de manera manual o automática, se debe verificar que los códigos tengan las características de un buen programa, que son: que sea fácil de entender, fácil de modificar y que arroje los resultados correctos. Si logramos lo anterior, iremos en el mismo sentido de los objetivos de la ingeniería de software.



1.7 Enfoque de sistemas aplicado al diseño de algoritmos computacionales

La aplicación del enfoque de sistemas en el diseño de algoritmos computacionales implica identificar:

1. Las entradas del problema, es decir, los datos o elementos previamente conocidos.
2. Las salidas del problema, es decir, datos o elementos que se desea conocer.
3. El proceso es la utilización de los elementos de entradas del problema para producir una salida o resultado.

El enfoque de sistemas está representado gráficamente en la figura 1.2.



Figura 1.2 El enfoque de sistemas.

El diseño de algoritmos computacionales puede representar gráficamente las tres etapas del enfoque de sistemas:

1. **Entradas del problema.** Se refiere a la introducción de datos conocidos a la computadora, mediante asignación o inserción. Éstos se almacenan en una porción de la memoria RAM. Recuerde que la RAM sólo permite el almacenamiento y conservación de los datos, mientras recibe el suministro de una fuente de energía (véase figura 1.3, donde se representa una computadora que está “pensando”). Esta etapa del enfoque de sistemas coincide con la acción computacional que se conoce como lectura de datos.
2. **Proceso.** Implica la utilización de los datos de entrada para generar los datos de salida, para lo cual generalmente se utiliza alguna fórmula aritmética, a fin de calcular los datos que se desea conocer. Esta etapa del enfoque de sistemas coincide con la acción computacional que se conoce como proceso de datos.
3. **Salidas del problema.** Los datos de salida y, probablemente, algunos o todos los datos de entrada, se imprimirán en papel o se presentarán en el monitor de la computadora para que el usuario pueda utilizarlos a manera de información en la toma de decisiones. Esta etapa del enfoque de sistemas coincide con la acción computacional que se conoce como impresión de resultados.



Figura 1.3 La memoria RAM.



1.8 Métodos para el diseño de algoritmos

Como se mencionó anteriormente, para construir un programa computacional se parte de un algoritmo, en él se establece la secuencia lógica de acciones computacionales que se deben ejecutar para la solución del problema relacionado con el tratamiento de información. Los dos métodos más comunes para el diseño de algoritmos computacionales son: diagramas de flujo y pseudocódigo.

Un **diagrama de flujo** es un conjunto de símbolos que representan acciones computacionales y que se disponen en una secuencia lógica para que puedan ejecutar un algoritmo computacional, el cual es la representación gráfica de un programa computacional.

Para construir un diagrama de flujo se utiliza una serie de símbolos estándares mundialmente utilizados y desarrollados por organizaciones como el ANSI (*American National Standards Institute*) o la ISO (*International Organization for Standardization*). Los símbolos representan una acción computacional u operación del tratamiento de información.

Para la construcción manual de diagramas de flujo es conveniente tomar en cuenta las siguientes reglas generales:

1. Utilice símbolos estandarizados.
2. Desarrolle el diagrama de flujo, de tal forma que se lea de arriba abajo y de izquierda a derecha siempre que sea posible. No cruce líneas de flujo. Use puntas de flechas para indicar dirección.
3. Mantenga el diagrama de flujo claro, legible y simple. Deje suficiente espacio entre los distintos símbolos. Si la solución de un problema es larga y compleja, divídala en varios diagramas de flujo.
4. Escriba mensajes sencillos para describir los distintos pasos a lo largo del diagrama de flujo.
5. Escriba en forma clara y legible los símbolos.

Raptor es la herramienta automática o software que se utilizará en este libro para crear algoritmos computacionales con el método de los diagramas de flujo. Tiene como ventaja que toma en cuenta las recomendaciones anteriores, por lo que el diseñador de algoritmos puede concentrarse en la solución del problema y no en los errores manuales de diseño.

Hay que aclarar que es posible que los símbolos que utiliza Raptor sean diferentes a los que se pueden encontrar en Internet, aunque en esencia representan la misma acción computacional. Se le sugiere al lector que compare los símbolos de Raptor con los encontrados en Internet y procure que las diferencias no interfieran en la comprensión de las acciones computacionales. En este caso, podemos decir que lo importante no son los símbolos, sino la acción computacional que representan.

Por otro lado, el método de pseudocódigo se basa en el establecimiento de un conjunto de palabras clave que se refieren a las acciones computacionales y que

permiten describir un algoritmo computacional. Posteriormente, esta descripción se ajusta a la sintaxis de un lenguaje de programación para convertirlo en software o programa computacional. Este método está fuera del alcance de este libro, pero si desea profundizar en este tema puede descargar un software llamado **PSeint** en la siguiente liga <http://pseint.sourceforge.net/>.

Cabe hacer una precisión, no hay que confundir el diseño de algoritmos con el diseño de software; este último utiliza métodos más complejos para cumplir con su objetivo, entre los cuales podemos mencionar el diagrama de flujo de datos, el diccionario de datos, el español estructurado, la normalización de bases de datos, el modelo entidad-relación y el Lenguaje de Modelo Unificado o *Unified Model Language* (UML).



1.9 Expresiones matemáticas

La ciencia matemática estudia las propiedades y relaciones entre entidades abstractas. Esta ciencia aplica un procesamiento simbólico de las entidades que son su objeto de estudio, por lo que es necesario otorgarles, a cada una de ellas, un identificador para distinguirlas unas de otras y no confundirlas.

Los elementos que forman parte de una expresión o fórmula matemática se pueden clasificar en **operandos** y **operadores**. Los primeros son las variables independientes y las eventuales constantes que la fórmula pueda considerar. Cada operando hace referencia a una entidad que es estudiada por las matemáticas, por lo que se le debe asignar un identificador. Los segundos son los símbolos o caracteres que representan una operación aritmética, como la suma, resta, multiplicación y división. La variable dependiente no se considera operando porque no es parte del cálculo matemático, pero sí es una entidad, por lo que también se le asigna un identificador.

Por otro lado, tomando como referencia el enfoque de sistemas, las acciones computacionales que permiten la creación de algoritmos básicos son: lectura de datos, proceso de datos e impresión de resultados.

La acción computacional de **lectura de datos** implica la inserción o asignación de valores a un identificador, el cual se almacena en una celda de la memoria RAM. La inserción se hace utilizando un dispositivo periférico como el teclado y la realiza manualmente el usuario del programa; también, se puede hacer empleando un medio de almacenamiento de datos secundario. La asignación se hace usando una instrucción computacional. Por lo general, los identificadores que forman parte de la lectura de datos son las variables independientes o las constantes de una fórmula matemática, por lo que se les considera identificadores de entrada.

La acción computacional de **proceso de datos** implica la transformación de los identificadores de entrada en datos de salida. Esto se hace con la ejecución de los operadores aritméticos, para finalmente asignar el resultado a la variable dependiente de una fórmula matemática, que es el identificador de salida.

La tabla 1.2 permite visualizar mejor la combinación de la clasificación matemática de los elementos de una expresión aritmética y el enfoque de sistemas.

Tabla 1.2 Visualización de fórmulas matemáticas bajo el enfoque de sistemas

Fórmula matemática	Identificadores de entrada (variables independientes o constantes)	Identificador de salida (variable dependiente)
$C = A + B$	A, B	C
$X = \frac{A}{B}$	A, B	X
$Z = x^y$	X, Y	Z
$A = \pi r^2$	π, r	A
Esta visualización es desde el punto de vista de la ciencia matemática.		

Considere que la ingeniería de software no necesariamente clasifica los elementos de una fórmula matemática en identificadores de entrada e identificadores de salida. Para esta ingeniería todos son identificadores.

Antes de explicar la acción computacional de **impresión de resultados** es conveniente aclarar que la visualización de los elementos de una fórmula aritmética desde la ciencia matemática es más rígida que la visualización desde la ciencia computacional; esta última es más flexible porque considera que los identificadores de entrada pueden ser parte de los datos de salida o impresión de resultados.

La acción computacional de impresión de resultados implica mostrar, en el monitor o imprimir en papel, mensajes y/o el valor de los identificadores que están almacenados en la memoria RAM; en este caso, pueden ser tanto las variables independientes como las variables dependientes y las constantes de una fórmula matemática.

1.10 Reglas de prioridad para escribir expresiones matemáticas

Anteriormente, señalamos que los operadores matemáticos son los caracteres que representan una operación aritmética como la suma, resta, multiplicación, división, etc. Éstos se pueden clasificar en: **unarios**, éstos sólo requieren de un operando para funcionar, y **binarios**, que requieren de dos operandos. La tabla 1.3 sólo muestra los operadores binarios más comunes. Por el momento, no mostraremos los operadores unarios.

Tabla 1.3 Operadores matemáticos binarios

Operador	Acción
+	Suma
−	Resta
*	Multiplicación
/	División (cociente)
%	División entera (residuo)
^	Potencia

Para que las expresiones matemáticas puedan usarse en el diseño de algoritmos y programación de computadoras es necesario transformarlas en fórmulas computacionales, para lo cual se deben seguir ciertas reglas de prioridad para la evaluación de expresiones aritméticas que se presentan a continuación.

1. Se le llama **subexpresión** a una parte de la fórmula completa. Todas las subexpresiones deben ser evaluadas por separado.
2. Para alterar los niveles de prioridad se pueden usar paréntesis. Las subexpresiones que se anotan entre paréntesis tienen prioridad por encima de todas las demás.
3. Si hay varias subexpresiones se evalúan de izquierda a derecha.
4. Si hay subexpresiones que están dentro de otras subexpresiones se evalúa de la más interna a la más externa.
5. La regla de prioridad de los operadores dice que los operadores en la misma subexpresión son evaluados en el siguiente orden:
 - a) Los operadores unarios, doble más o doble menos.
 - b) Los operadores binarios de multiplicación y división.
 - c) Los operadores binarios de suma y resta.
6. La regla de la asociatividad dice que los operadores unarios en la misma subexpresión y en el mismo nivel de prioridad, tales como ++ y --, son evaluados de derecha a izquierda (asociatividad a la derecha). Por otro lado, los mismos operadores binarios en la misma subexpresión y en el mismo nivel de prioridad, tales como + y −, son evaluados de izquierda a derecha (asociatividad a la izquierda).

En la tabla 1.4 podrá visualizar fórmulas expresadas de manera matemática y su transformación, respetando las reglas de prioridad.

Tabla 1.4 Ejemplos de conversión de fórmula matemática a fórmula computacional

Fórmula matemática	Fórmula computacional
$D = A + B + C$	$D = A + B + C$
$D = A + BC$	$D = A + B * C$
$C = \frac{A}{B}$	$C = A / B$
$D = \frac{A}{B + C}$	$D = A / (B + C)$
$D = \frac{A + B}{C}$	$D = (A + B) / C$
$d = b^2 - 4ac$	$d = b^2 - 4 * a * c$
$M = \frac{3X^2}{Y - 3}$	$M = 3 * X^2 / (Y - 3)$
$m = \frac{y - b}{x - c}$	$m = (y - b) / (x - c)$
$K = \frac{3X}{1 + \sqrt{X}}$	$K = 3 * X / (1 + \text{sqrt}(X))$



11.11 Reglas de calidad para dar nombre a identificadores de variables y constantes

Todo software, durante su vida útil, debe recibir mantenimiento para que funcione adecuadamente. Por ello, una de las preocupaciones de la ingeniería de software es disminuir el costo de mantenimiento porque suele ser alto en tiempo y costo.

En el presente libro iremos presentando reglas de calidad en la escritura de algoritmos que tienen como objetivo generar un código estandarizado que facilita el mantenimiento del software. La estandarización del código tiene su base teórica y práctica en el proceso de software personal propuesto por Humphrey.

Recuerde que a los elementos de las expresiones matemáticas, variables y constantes, que se utilizarán en un algoritmo computacional se les debe otorgar un identificador o nombre que permita distinguirlos unos de otros. Los identificadores pueden ser de dos tipos: variables y constantes, según si el valor puede o no cambiar dentro de un algoritmo computacional. Para dar nombre a identificadores de variables se proponen las siguientes reglas de calidad.

1. Las variables deben tener un nombre significativo, es decir, que con el puro nombre se pueda deducir a qué se refiere.
2. Evite abreviaciones o variables de una sola letra.
3. El primer carácter del nombre de la variable debe ser una letra en mayúscula, las demás en minúscula. Cuando dos palabras describan mejor una variable, la primera letra de la segunda palabra deberá ser mayúscula.
4. Si el nombre de la variable requiere un número, escríbalo continuo a las letras. Dele preferencia a usar el número al final del nombre.
5. No utilice acentos en las palabras.

La tabla 1.5 muestra algunos ejemplos de nombres de variables, aplicando las reglas de calidad.

Tabla 1.5 Ejemplos aceptados para nombres de identificadores de variables

Matricula	ValorEntero2	PagoNeto1
Nombre	ContadorPares	PagoNeto2
Salon	PagoBruto	Salario1Parcial
ValorEntero1	ApellidoPaterno	Salario2Parcial

Las reglas de calidad para dar nombres a identificadores de constantes son las siguientes:

1. Los nombres de las constantes deben ser significativos, es decir, que con el puro nombre se pueda deducir a qué se refiere.
2. Evite abreviaciones o constantes de una sola letra.
3. Todos los caracteres alfabéticos que forman parte del nombre de la constante deben ser letras mayúsculas.
4. Si el nombre de la constante requiere un número, escríbalo continuo a las letras. Dele preferencia a usar el número al final del nombre.
5. Cuando sea necesario que el nombre de la constante tenga más de una palabra, éstas se separarán utilizando un guión bajo.
6. No utilice acentos en las palabras.

La tabla 1.6 muestra algunos ejemplos de nombres de constantes, aplicando las reglas de calidad.

Tabla 1.6 Ejemplos aceptados para nombres de identificadores de constantes

PI	VELOCIDAD_LUZ	ERROR_ESTADISTICO
GRAVEDAD	GRAVEDAD_METRICA1	GRAVEDAD1_METRICA
VALOR_MAXIMO	GRAVEDAD_METRICA2	GRAVEDAD2_METRICA
VALOR_MINIMO	GRAVEDAD_ANGLOSAJON	IMPUESTO_VALOR_AGREGADO
METRO	KILOGRAMO	LITRO

En los ejemplos de expresiones matemáticas mostrados en las tablas 1.2 y 1.4 es notorio que los nombres de los identificadores no cumplen con las reglas de calidad mostradas en este apartado. Un ejemplo de cómo debería quedar la fórmula para calcular el discriminante de una ecuación cuadrática ($d = b^2 - 4ac$) si cumpliéramos con estas reglas se muestra a continuación:

$$\text{Discriminante} = \text{CoeficienteLineal}^2 - 4 * \text{CoeficienteCuadratico} * \text{TerminoIndependiente}$$



1.12 Reglas de calidad para escribir expresiones matemáticas

Autores como Deitel y Humphrey emiten diversas recomendaciones para escribir expresiones matemáticas en las codificaciones de programas; sin embargo, por no ser de aplicación directa a los algoritmos, sólo presentaremos algunas de ellas.

El objetivo de la primera y segunda reglas, que son las que más vamos a usar, es evitar que las expresiones matemáticas luzcan amontonadas y sea difícil distinguir entre operadores y operandos, lo cual dificulta el mantenimiento del software.

1. Coloque espacios a cada lado de un operador binario.
2. Cuando utilice paréntesis deje un espacio en blanco antes y después de los operandos.
3. Cuando realice divisiones con expresiones cuyo denominador pueda ser CERO, o que el radicando de una raíz cuadrada pueda ser negativo, haga una prueba explícita de este caso y manéjela de manera apropiada a su programa (tal como la impresión de un mensaje de error), en lugar de permitir que ocurra un error de ejecución o fatal.
4. Otorgue un valor inicial a las variables contadoras y acumuladoras para disminuir la probabilidad de resultados incorrectos y evitar errores de ejecución o de lógica.



1.13 Ejercicios de autoevaluación

Las respuestas correctas a los siguientes ejercicios están contenidas en la tabla 1.7.

- 1.1 ¿Cómo se denomina el conjunto de acciones que permiten el procesamiento de datos, mediante las operaciones de lectura, escritura, copia, transmisión, ordenación, clasificación, comparación, archivo, cálculo, traducción, análisis y síntesis?
- 1.2 ¿Cómo se llama la ciencia que se ocupa de estudiar el tratamiento de información, mediante la utilización de dispositivos electrónicos diseñados para ejecutar las acciones que permiten el procesamiento de datos?
- 1.3 ¿Cómo se llama el dispositivo electrónico que permite ejecutar las operaciones de tratamiento de información y transformar datos en información?
- 1.4 ¿Cuál es la actividad humana por la cual se transforman datos en información?
- 1.5 ¿Cómo se denomina el software que se caracteriza por su gran interacción con el hardware, entre ellos, el sector de arranque y que permite a las unidades funcionales de la computadora iniciar y realizar sus acciones?
- 1.6 ¿Cómo se denomina el software que permite realizar las operaciones del tratamiento de información?
- 1.7 ¿Cómo se denomina la unidad funcional de la computadora que se subdivide en la unidad aritmético-lógica y la unidad de control?
- 1.8 ¿Cómo se denomina la unidad funcional de la computadora que es parte de la CPU y es la encargada de administrar los recursos o unidades con los que cuenta la computadora y que también interpreta las instrucciones generadas por un programa e inicia las operaciones apropiadas para llevarlas a cabo?
- 1.9 ¿Cómo se denomina la unidad funcional de la computadora que requiere de un suministro de energía permanente para almacenar, conservar y acceder a información digital?
- 1.10 ¿Cómo se denomina el dispositivo electrónico que requiere estar conectado a una fuente de energía para almacenar datos, conservar y acceder a información digital y que está compuesto por circuitos integrados? Se dice que es un medio de almacenamiento volátil porque cuando se desconecta de la fuente de energía todos los datos ahí almacenados se pierden.
- 1.11 ¿Cómo se denomina el dispositivo electrónico que requiere una fuente de energía para funcionar y acceder a información digital,

pero que cuando se desconecta de la fuente de energía no pierde la información ahí almacenada?

- 1.12 ¿Cómo se denomina la unidad funcional de la computadora que permite la introducción de datos a la unidad de memoria de una computadora?
- 1.13 ¿Cómo se denomina la unidad funcional de la computadora que permite presentar al usuario el estado de los datos que se encuentran en la unidad memoria de la computadora?
- 1.14 ¿Cómo se denomina la unidad funcional de la computadora que permite el almacenamiento de datos digitales sin que requiera estar permanentemente conectada a una fuente de energía?
- 1.15 ¿Cómo se denomina la unidad funcional de la computadora que permite a las otras cinco unidades funcionales de la computadora comunicarse entre sí y que físicamente es un conjunto de cables, además de que se encuentra separada en tres canales?
- 1.16 ¿Cómo se denomina la disciplina que aglutina métodos y herramientas que coadyuvan a la construcción de software con calidad?
- 1.17 ¿Cómo se denomina el conjunto de actividades, acciones y tareas que permiten desarrollar el software?
- 1.18 ¿Cómo se denomina el conjunto de pasos o procedimientos que deben seguirse para resolver un problema aritmético?
- 1.19 ¿Cómo se denomina el conjunto de acciones computacionales que deben ejecutarse con una secuencia lógica para resolver un problema, mediante operaciones propias del tratamiento de información?
- 1.20 ¿Cómo se denomina la acción computacional que permite asignar o insertar datos en la memoria RAM de la computadora?
- 1.21 ¿Cómo se denomina la acción computacional que permite al usuario observar, en el monitor de la computadora o en papel, el estado o valor de los datos que están almacenados en la memoria RAM?
- 1.22 ¿Cómo se denomina la acción computacional que permite obtener o calcular datos desconocidos a partir de datos previamente conocidos?
- 1.23 ¿Cómo se denomina la acción computacional que permite configurar diferentes opciones mutuamente excluyentes, que se pueden elegir a partir de condiciones matemáticas y se plantean con operadores lógicos y relacionales?
- 1.24 ¿Cómo se denomina la acción computacional que permite configurar una opción repetitiva y una opción diferida, que no son mutuamente excluyentes?

- 1.25 ¿Cómo se denomina la opción del ciclo que consiste en que una o más acciones computacionales puedan ejecutarse en la misma secuencia y de manera iterativa, bajo un ambiente controlado para que eventualmente deje de dar vueltas, impidiendo generar ciclos infinitos?
- 1.26 ¿Cómo se denomina la opción del ciclo que consiste en una o más acciones computacionales, cuya ejecución se posterga para después de que el ciclo haya llegado a su fin?
- 1.27 ¿Cómo se llama el conjunto de reglas que permiten escribir software en un lenguaje específico?
- 1.28 ¿Cuál es la actividad que permite transformar un algoritmo computacional en un programa computacional?
- 1.29 ¿Cómo se denomina la habilidad intelectual de analizar e interpretar un problema y llevar ese problema a un nivel de abstracción, tal que permita desarrollar una solución, mediante un algoritmo computacional, que después pueda convertirse en un código que sea interpretable y ejecutable por una computadora?
- 1.30 ¿Cuál es el nombre de los símbolos que forman parte de los alfabetos que permiten la comunicación entre las personas?
- 1.31 ¿Qué actividad del proceso de software permite generar la definición del alcance del software?
- 1.32 ¿Cuál es el principal producto de trabajo de la actividad de comunicación del proceso de software que permite establecer las características y funcionalidades que tendrá el software que se va a construir?
- 1.33 ¿Cómo se denomina la actividad del proceso de software que consiste en generar los productos de trabajo como las estimaciones del proyecto, el seguimiento y control del proyecto, la administración de los riesgos, las estrategias de aseguramiento de la calidad, el control de la configuración del software, el presupuesto económico y el tiempo que se invertirá en construir el software, así como la lista y disposición de los requerimientos del ser humano, materiales y tecnológicos, la agenda pormenorizada de las acciones y tareas con su respectiva ventana de tiempo?
- 1.34 ¿Cuál es la actividad del proceso de software que consiste en crear o desarrollar una lista de pasos o acciones computacionales a seguir para la solución de un problema, lo cual permitirá tener un modelo llamado algoritmo computacional y suele ser una representación gráfica del software que se va a construir?
- 1.35 ¿Qué actividad del proceso de software consiste en verificar que el diseño sea correcto y resuelva el problema en la forma que se intenta?

- 1.36 ¿Cuál es la actividad del proceso de software que consiste en revisar que el software creado cumpla con las reglas de sintaxis del lenguaje de programación seleccionado?
- 1.37 ¿Cómo se denomina la actividad del proceso de software que consiste en la revisión automatizada que determina que el software creado cumpla con las reglas de sintaxis del lenguaje de programación seleccionado, y que una vez verificado produce un programa ejecutable?
- 1.38 ¿Cuáles son los errores del código que no cumplen con las reglas de escritura del lenguaje seleccionado?
- 1.39 ¿Qué actividad del proceso de software consiste en ejecutar el programa para saber si funciona correctamente?
- 1.40 ¿Cómo se denomina el enfoque de comprensión de los problemas, en el cual se visualizan tres fases: *a)* los datos de entrada o datos conocidos, *b)* el proceso de datos y *c)* los datos de salida o datos que se desea conocer?
- 1.41 ¿Cómo se llama la fase del enfoque de sistemas en que se visualizan los datos conocidos del problema?
- 1.42 ¿Cuál es la fase del enfoque de sistemas en que se visualiza cómo se transformarán los datos conocidos para obtener los datos que se desea conocer?
- 1.43 ¿Cómo se denomina la fase del enfoque de sistemas en que se visualizan los resultados que produce el procesamiento de datos?
- 1.44 ¿Cómo se llama el conjunto de símbolos que representan acciones computacionales y que se disponen en una secuencia lógica para que puedan ejecutar un algoritmo computacional?
- 1.45 ¿Cómo se nombra el conjunto de palabras clave que se refieren a las acciones computacionales y que permiten describir un algoritmo computacional?
- 1.46 ¿Cuál es la ciencia que estudia las propiedades y relaciones entre entidades abstractas y que aplica procesamiento simbólico de las entidades que son su objeto de estudio, por lo que es necesario otorgarles, a cada una de ellas, un identificador para distinguir las unas de otras y no confundirlas?
- 1.47 La(s) variable(s) independiente(s) de una fórmula matemática se consideran como parte de los datos de entrada. ¿Falso o verdadero?
- 1.48 La variable dependiente de una fórmula matemática se considera como dato de salida. ¿Falso o verdadero?
- 1.49 Desde el punto de vista matemático, ¿cuál es el nombre general que reciben la variable dependiente, las variables independientes y las constantes de una fórmula matemática?

- 1.50** Desde el punto de vista matemático, ¿cuál es el nombre general que reciben las variables independientes y las constantes de una fórmula matemática?
- 1.51** Desde el punto de vista matemático, ¿cuál es el nombre general que reciben los símbolos o caracteres que representan una operación aritmética?
- 1.52** ¿Cómo se denomina la acción que permite agregar datos a la memoria RAM y que se lleva a cabo utilizando un dispositivo periférico como el teclado?
- 1.53** ¿Cómo se denomina la acción que permite agregar datos a la memoria RAM, utilizando una instrucción computacional?
- 1.54** ¿Cómo se denominan los operadores aritméticos que requieren de dos operandos para funcionar?

Tabla 1.7 Respuestas a los ejercicios de autoevaluación

1.1 Tratamiento de información	1.2 Informática	1.3 Computadora	1.4 La toma de decisiones
1.5 Sistema operativo	1.6 Software de aplicación	1.7 La unidad central de proceso	1.8 La unidad de control
1.9 La unidad de memoria	1.10 Memoria RAM	1.11 Memoria ROM	1.12 Unidad de entrada
1.13 Unidad de salida	1.14 Unidad de almacenamiento secundario	1.15 El bus del sistema	1.16 Ingeniería de software
1.17 Proceso de software	1.18 Algoritmo matemático	1.19 Algoritmo computacional	1.20 Lectura de datos
1.21 Impresión de resultados	1.22 Proceso de datos	1.23 Selección de alternativas	1.24 El ciclo
1.25 Opción repetitiva	1.26 Opción diferida	1.27 Sintaxis	1.28 Codificación
1.29 Programación de computadoras	1.30 Carácter	1.31 Comunicación	1.32 Alcance del software
1.33 Planeación	1.34 Diseño	1.35 Revisión del diseño	1.36 Revisión de la codificación
1.37 Compilación	1.38 Error de sintaxis	1.39 Prueba	1.40 Enfoque de sistemas
1.41 Las entradas del problema	1.42 El Proceso de datos	1.43 Salidas del problema	1.44 Diagrama de flujo
1.45 Pseudocódigo	1.46 Matemáticas	1.47 Verdadero	1.48 Verdadero
1.49 Identificadores	1.50 Operandos	1.51 Operadores	1.52 Inserción
1.53 Asignación	1.54 Binarios		



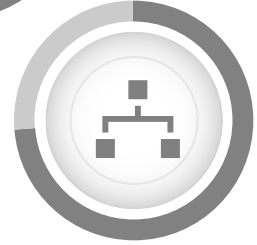
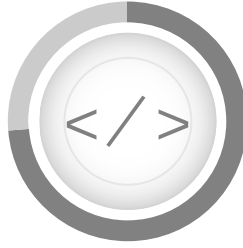
1.14 Problemas propuestos

Las personas que se dedican al desarrollo de software deben tener habilidades analíticas para describir un proceso en la solución de problemas. Tanto la mejora de un pensamiento estructurado que le permita acomodar ideas en un orden lógico para alcanzar un objetivo, como el avance de las competencias de comunicación oral y escrita que le permita transmitir con claridad un mensaje, son habilidades que un ingeniero debe tener para construir software con calidad.

Utilice la descripción narrativa y realice los siguientes problemas.

- 1.1 Lea el cuento infantil de Caperucita Roja y escriba una versión alterna considerando que usted es el lobo feroz. Escriba máximo una cuartilla (hoja tamaño carta) en fuente arial, tamaño 12 puntos. Aplique las reglas de ortografía, redacción y de párrafo.
- 1.2 Use elementos del enfoque de sistemas, explique el sistema educativo y el sistema jurídico de nuestro país. Escriba por lo menos media cuartilla para cada uno de ellos.
- 1.3 Tome en cuenta las reglas de calidad para dar nombre a identificadores y escriba los nombres de los elementos que componen las fórmulas que se enlistan a continuación.
 - a) Área de un círculo.
 - b) Una ecuación cuadrática.
 - c) Teorema de Pitágoras para calcular la longitud de la hipotenusa de un triángulo rectángulo.
 - d) Velocidad final de un cuerpo en caída libre.
- 1.4 Tome en cuenta el ejercicio anterior y las reglas de prioridad para escribir expresiones aritméticas. Convierta las fórmulas matemáticas en fórmulas computacionales.
- 1.5 Escriba tres versiones, cada una con 10, 15 y 20 pasos, de cómo cambiar la llanta de un automóvil.
- 1.6 Describa el algoritmo para sumar dos valores numéricos positivos, el primero con tres dígitos y el segundo con dos dígitos. No describa ejemplos específicos. Explique el algoritmo de forma generalizada. Considere que los números más a la izquierda de ambas cifras deben ser diferentes a cero. Haga lo mismo describiendo la operación de restar, donde la cifra con tres dígitos es el minuendo y la de dos dígitos es el sustraendo.

- 1.7 Haga lo mismo que en el punto anterior, pero ahora con la multiplicación y la división.
- 1.8 Conoce el espacio total en un disco duro y el espacio ocupado. Escriba la fórmula que permita calcular el porcentaje de espacio disponible.
- 1.9 Un maestro desea determinar el porcentaje de aprobados y reprobados en un grupo; sólo sabe cuántos alumnos han aprobado y cuántos han reprobado. Escriba la fórmula que permite calcular ambos porcentajes.
- 1.10 Una maestra midió la altura de Juan al principio y final del año escolar. Escriba la fórmula que permite calcular el porcentaje de crecimiento de Juan.
- 1.11 Asista a una tienda de conveniencia y describa las características y funcionalidades del sistema de procesamiento de transacciones que se utiliza como punto de venta.



Capítulo 2

Diseño básico de algoritmos

- 2.1 Introducción
- 2.2 Raptor y sus símbolos para ejecutar las acciones computacionales básicas
- 2.3 Símbolo Output
- 2.4 Símbolo Assignment
- 2.5 Operadores matemáticos básicos
- 2.6 Símbolo Input
- 2.7 Documentación interna en los diagramas
- 2.8 Desarrollo de algoritmos básicos
- 2.9 Ejercicios de autoevaluación
- 2.10 Problemas propuestos

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Diseñar algoritmos computacionales básicos con la metodología de diagramas de flujo, utilizando la herramienta de Raptor.
- Utilizar identificadores y comprender su función dentro de un algoritmo computacional.
- Comprender cómo agregar datos a la memoria RAM de la computadora, utilizando la acción computacional de lectura de datos.
- Ejecutar las operaciones aritméticas básicas, utilizando la acción computacional de procesamiento de datos.
- Permitir que valores que están almacenados en la memoria RAM puedan ser observados, en el monitor, por el usuario, utilizando la acción computacional de impresión de resultados.



2.1 Introducción

La automatización de las actividades humanas ha permitido aumentar la competitividad y productividad de las empresas y personas. El desarrollo de software no ha sido ajeno a esta tendencia. La automatización de las actividades del proceso de software, mediante las herramientas de Ingeniería de Software Asistida por Computadora o *ComputerAided Software Engineering* (CASE), ha permitido reducir el costo económico, tiempo y esfuerzo de desarrollo.

El software de Raptor es una herramienta automática puntual que tiene características de *lower CASE*, pues permite crear, de manera gráfica, algoritmos computacionales con la metodología de los diagramas de flujo y la generación de código. Uno de los beneficios de Raptor es que se puede revisar el funcionamiento del algoritmo porque ejecuta las acciones computacionales establecidas en el mismo.

Raptor es un lenguaje gráfico de programación que está diseñado para ayudar a sus usuarios a visualizar tanto la creación como la ejecución de las acciones computacionales que forman parte de un algoritmo. Reduce al mínimo las reglas de sintaxis de los lenguajes de programación tradicionales, las cuales suelen confundir a los novatos cuando están iniciándose en el mundo del desarrollo de software.

Los algoritmos hechos en Raptor permiten que la ejecución se pueda rastrear a través del diagrama de flujo. Gracias a esta posibilidad, a los principiantes se les facilita el aprendizaje y el desarrollo de la lógica algorítmica; este aprendizaje es más lento cuando usan un lenguaje tradicional o cuando escriben diagramas de flujo sin una herramienta automática como ésta.

En la figura 2.1 podemos apreciar la interfaz principal de Raptor con una ventana sobrepuesta llamada *About Raptor*, en la que destaca la figura de un dinosaurio completo y de perfil. En ella se nos indica que la palabra raptor está formada por las iniciales de *Rapid Algorithmic Prototyping Tool for Ordered Reasoning*, así como por los nombres de los creadores, la versión del software, la fecha de actualización,

la página electrónica de donde se puede descargar de manera gratuita y otros datos. Esta ventana se puede observar seleccionando la opción *About* del menú *Help*. Para obtener el software visite la dirección electrónica <http://raptor.martincarlisle.com/>.

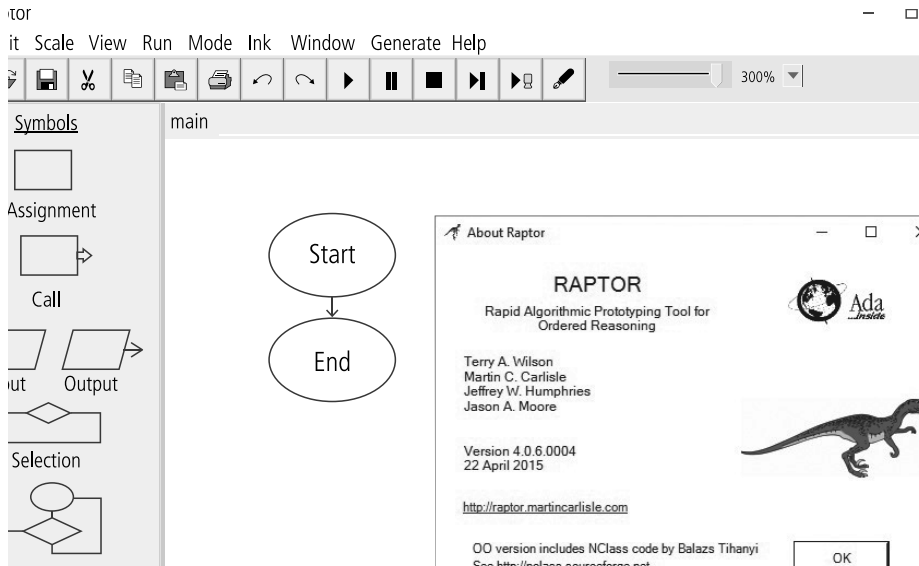


Figura 2.1 Interfaz principal de Raptor.

En la parte superior de la interfaz principal se pueden ver los menús y la barra de herramientas de los que dispone Raptor (véase figura 2.2). Los menús son *File*, *Edit*, *Scale*, *View*, *Run*, *Mode*, *Ink*, *Window*, *Generate* y *Help*. Debajo de los menus están los botones comunes de *New*, *Open*, *Save*, *Cut*, *Copy*, *Paste*, *Print*, *Undo*, *Redo*, *Execute to Completion*, *Pause*, *Stop/Reset*, *Step to Next Shape*, *Test against Server*, *Toggle ink*. Enseguida de los botones está el control con desplazamiento horizontal de la velocidad de ejecución de los algoritmos y el control del tamaño de visualización de los símbolos. En este caso, el control de velocidad está en el máximo y la visualización al 300%. Sin embargo, queremos destacar el botón de *Execute to Completion* (mejor conocido como *Play*), porque lo vamos a utilizar con frecuencia (nombres de las teclas en español: Nuevo, Abrir, Guardar, Cortar, Copiar, Pegar, Imprimir, Deshacer, Rehacer, Ejecutar hasta terminar, Pausa, Detener/Reestablecer, Paso hasta la siguiente figura, Probar contra el servidor y Poner tinta, respectivamente).



Figura 2.2 Menús y barra de herramientas de Raptor.

En la parte central de la interfaz principal se aprecia la pestaña de *main* con los globos elipsoidales de Start y End (Inicio y Fin) (véase figura 2.3).

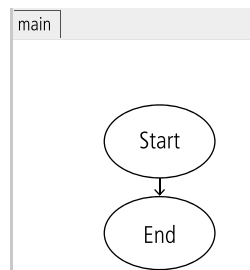


Figura 2.3 Parte central de la interfaz principal de Raptor.

En el costado izquierdo de la interfaz principal está la paleta de Symbols donde aparecen los bloques de diagrama de flujo Assignment, Call, Input, Output, Selection y Loop. Exactamente debajo de la paleta de Symbols hay un recuadro de exhibición donde aparecerán los valores que tomen los identificadores que estén en la memoria RAM (véase figura 2.4).

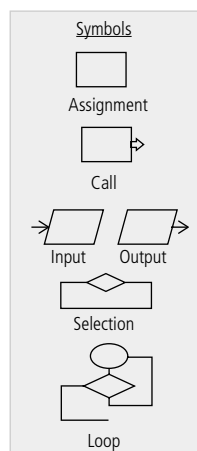


Figura 2.4 Paleta de símbolos y recuadro de exhibición de identificadores.

Además, Raptor cuenta con la interfaz secundaria que es una ventana alterna titulada *MasterConsole*, ésta se puede ocultar o sobreponer a la interfaz principal. Esta interfaz secundaria cuenta con los menús de *Font*, *Font Size*, *Edit* y *Help*. En la parte inferior tiene una caja de texto y el botón *Clear* (véase figura 2.5).

La ventana *MasterConsole* es más que un recuadro de exhibición de los valores de los identificadores que están en memoria RAM. En esta ventana es posible mostrar dichos valores acompañados de mensajes que permiten contextualizarlos en relación con una situación en particular.

MasterConsole debe activarse continuamente para poder ver los resultados calculados por los algoritmos. Una forma de activarse es presionando con el apuntador del ratón el botón de *Execute to Completion*; una segunda forma es seleccionando la opción *Execute to Completion* del menú *Run*; una tercera forma es activándola con el ratón.

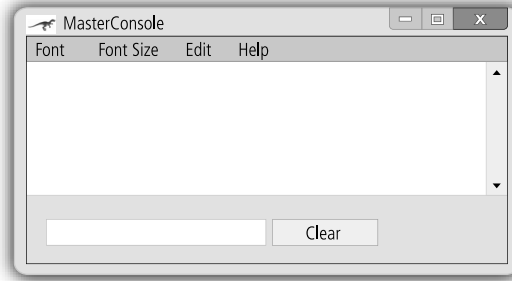


Figura 2.5 Ventana de *MasterConsole*.



2.2 Raptor y sus símbolos para ejecutar las acciones computacionales básicas

Para ejecutar las acciones computacionales básicas propias del tratamiento de información como la lectura de datos, el proceso de datos y la impresión de resultados, Raptor utiliza los símbolos Input, Assignment y Output, respectivamente.

El símbolo Input es la representación gráfica de la acción computacional de lectura de datos: un paralelogramo que tiene una flecha que “entra” en la esquina superior izquierda (véase figura 2.6) y equivale a la fase de las entradas del problema del enfoque de sistemas. Este símbolo permite la captura de datos a la memoria RAM, utilizando el teclado de la computadora. Los valores capturados se almacenan en celdas de memoria RAM que se conocen como variables o constantes.

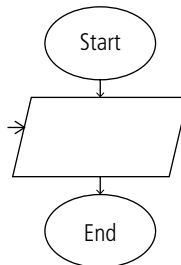


Figura 2.6 Símbolo Input (lectura de datos).

El símbolo Assignment es la representación gráfica de la acción computacional de un proceso de datos: un rectángulo que permite la asignación de un valor a un

identificador y la ejecución de operaciones matemáticas. Los valores asignados se almacenan en una celda de la memoria RAM y equivale a la fase de proceso del enfoque de sistemas (véase figura 2.7).

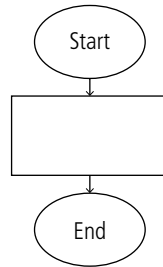


Figura 2.7 Símbolo Assignment (proceso de datos).

El símbolo Output es la representación gráfica de la acción computacional de impresión de resultados: un paralelogramo que tiene una flecha que “sale” de la esquina inferior derecha (véase figura 2.8), equivale a la fase de las salidas del problema del enfoque de sistemas. Este símbolo permite visualizar, en la ventana de *MasterConsole*, los mensajes y el valor de los identificadores que están en la memoria RAM.

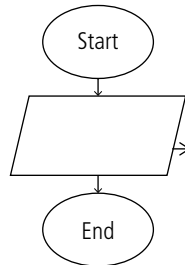


Figura 2.8 Símbolo Output (impresión de resultados).



2.3 Símbolo Output

Raptor utiliza el símbolo Output para representar la acción computacional de impresión de resultados. Output permite mostrar al usuario los valores de los identificadores que están en la memoria RAM. A estos valores se les puede agregar un mensaje de contextualización y ambos son observables en la ventana de *MasterConsole*. Los mensajes de contextualización ayudan a disminuir en el usuario la incertidumbre que genera mostrar un valor suelto sin referencia alguna.


La impresión de resultados se puede clasificar en exhibición de datos, mensaje simple y mensaje compuesto. Primero, la exhibición de datos es cuando se muestra el valor de algún identificador, pero no se incluye mensaje de contextualización. Segundo,

el mensaje simple es cuando se imprime un mensaje de contextualización; sin embargo, no se incluye mostrar valores de identificadores. Por último, el mensaje compuesto es cuando se muestran valores de identificadores debidamente contextualizados con un mensaje de referencia.

Antes de hacer nuestro primer ejercicio, se le pide al lector que escriba en una libreta cinco frases motivadoras o coloquiales para posteriormente hacer el diagrama que permita enviar esos mensajes a la ventana de *MasterConsole*. Algunos ejemplos son:

1. Ser los mejores no es la meta, es la consecuencia de nuestro esfuerzo.
2. Si supiera que el mundo se acaba mañana, no dudaría en plantar hoy un árbol de manzanas.

El primer diagrama que vamos a diseñar consiste en visualizar un mensaje simple en la ventana de *MasterConsole*. La frase trivial señala la edad de una persona. Se le pide al lector que realice los siguientes pasos:

1. Abra el Raptor e inicie un nuevo archivo en el menú *File*. Seleccione con el botón izquierdo del ratón la opción *Save* del menú *File* para darle nombre al diagrama. En la ventana de *Save as*, en la caja de texto *File name*, escriba el nombre del archivo, que en este caso se recomienda: **Algoritmo2-1 mensaje simple**.
2. En la paleta de Symbols coloque el apuntador del ratón sobre el símbolo de Output y presione el botón izquierdo del ratón; el contorno del Output se pondrá en color rojo.
3. Lleve el apuntador del ratón sobre la flecha que aparece entre los globos de Start y End de la pestaña *main*, hasta que el apuntador cambie a la forma de una manita .
4. Presione de nuevo el botón izquierdo del ratón y aparecerá el símbolo de Output entre los globos de Start y End. Observe que el nombre de la ventana es: Raptor-**Algoritmo2-1 mensaje simple.rap** (véase figura 2.9).

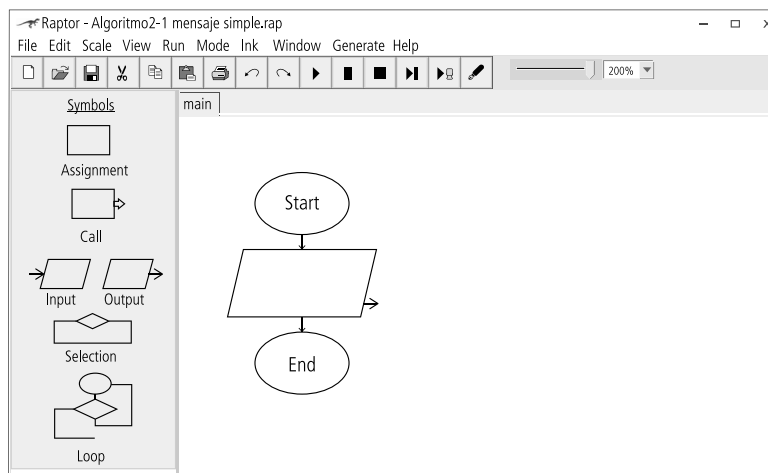


Figura 2.9 Pasos 1, 2, 3 y 4. Output.

5. Lleve el apuntador del ratón sobre la figura de Output que se encuentra entre los globos de Start y End, presione el botón izquierdo del ratón, dando doble clic, y aparecerá la ventana *Enter Output*.
6. En el único cuadro de texto de la ventana *Enter Output* escriba entre comillas el mensaje de contextualización que usted quiera que aparezca en la ventana de *MasterConsole*. En este caso es: **“Tengo 45 años de edad”**. Observe que la casilla de verificación de *End current line* se queda seleccionada (véase figura 2.10).

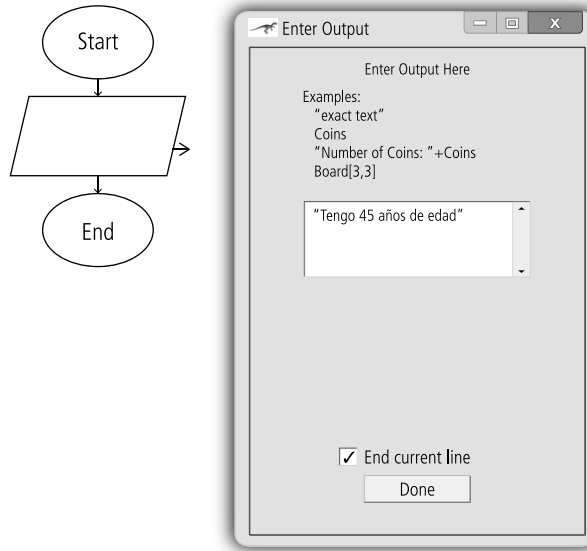


Figura 2.10 Pasos 5 y 6. Output y ventana *Enter Output*.

7. Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Output* y presione el botón izquierdo del ratón y verá la figura 2.11. Observe que el mensaje de contextualización aparece dentro del paralelogramo de Output y está precedido por la palabra *put*. Al final del mensaje aparece un símbolo “¶” que indica que está activada la casilla de verificación de *End current line*.

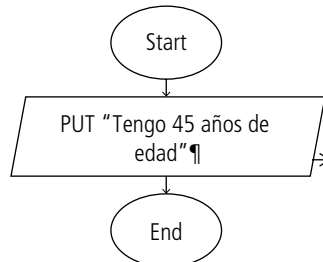


Figura 2.11 Paso 7. Output con mensaje simple. Algoritmo2-1 mensaje simple.

8. Finalmente, lleve el apuntador del ratón sobre el botón *Execute to Completion*, presione el botón izquierdo del ratón y de manera automática se abrirá la ventana de *MasterConsole* mostrando el mensaje simple de: **Tengo 45 años de edad**, que ahora no tiene las comillas (véase figura 2.12). El mensaje: **---Run complete. 3 symbols evaluated.---** es un aviso que Raptor muestra de manera automática e indica que el algoritmo ha ejecutado tres símbolos correctamente: Start, Output y End.

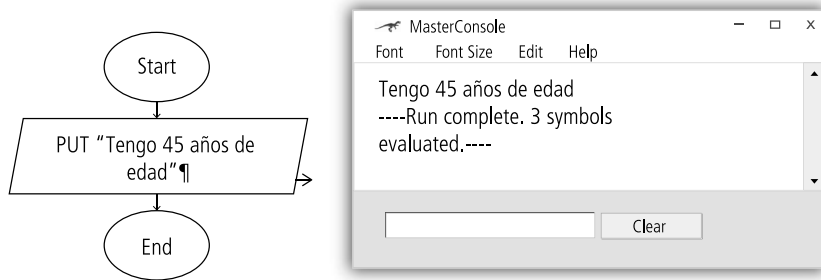


Figura 2.12 Paso 8. Output.

Practique el uso de Output:

1. Con las cinco frases que anteriormente se le pidió escribir en su libreta, utilice el símbolo de Output para enviar mensajes simples a *MasterConsole*.
2. Use la función `to_character()`, el símbolo de Output y el código ASCII; haga un diagrama que envíe un mensaje simple a *MasterConsole*. Investigue en el menú *Help* de Raptor, en el tema de String Operations, cómo se utiliza esta función.



2.4 Símbolo Assignment

Raptor emplea el símbolo Assignment para representar la acción computacional de proceso de datos. Las expresiones de asignación que este símbolo permite pueden o no involucrar operaciones matemáticas, en el primer caso a esta operación se le llama asignación constante; y en el segundo, asignación variable. Los valores asignados mediante Assignment se almacenan en una celda de la memoria RAM que se puede distinguir por su nombre de identificador.

El nombre del identificador es una invención a discreción del programador. Sin embargo, en este libro (en el capítulo 1), se dieron a conocer las reglas de calidad para dar nombre a identificadores variables y constantes, por lo que se le pide al lector revisar estas reglas.

Para hacer una asignación constante sólo debemos saber el nombre del identificador y el valor que le será asignado. Vea la figura 2.13, donde al identificador **EDAD** se le asigna el valor de 45. Observe que de izquierda a derecha aparece primero el nombre del identificador que recibe el valor; luego, aparece una flecha

que apunta hacia el identificador (o de derecha hacia la izquierda) y, finalmente, aparece el valor asignado al identificador. Esta expresión de asignación debe leerse de derecha a izquierda, es decir, “el 45 se asigna a EDAD”. Observe también que el nombre del identificador “**EDAD**” está escrito en mayúsculas, respetando las reglas de calidad para nombres de identificadores constantes.

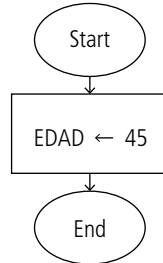


Figura 2.13 Asignación a un identificador constante.

Para hacer una asignación variable debemos contar con una fórmula matemática donde la variable dependiente esté completamente despejada, y distinguir entre operadores y operandos. Los operadores son los caracteres que representan cualquier operación matemática, como suma, resta, multiplicación, división, etc. Los operandos son las variables independientes y las constantes que tenga dicha fórmula. La variable dependiente será aquella que reciba el valor generado después de ejecutar todas las operaciones matemáticas que tenga la fórmula.

En la figura 2.14 podemos distinguir que la variable A y B son los operandos. El operador es el carácter positivo de “+”, que representa la suma. La variable C es dependiente. En este caso, primero se realizan todas las operaciones aritméticas y el resultado se asigna a la variable dependiente. Observe que los nombres de los tres identificadores no están de acuerdo con las reglas de calidad para dar nombre a las variables, esto se hace con fines de simplificación didáctica.

Para colocar una expresión de asignación en el símbolo Assignment se deben llenar las dos cajas de texto con que cuenta este block, que son Set y To. La caja Set es el lugar donde se escribe el nombre del identificador que va a recibir el valor. En la caja de To, si es una asignación constante, se escribe el valor que será asignado al identificador. Si es una asignación variable, en la caja de To se escribe la expresión matemática completa, incluyendo sólo operadores y operandos.

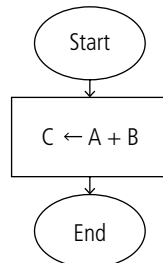



Figura 2.14 Asignación a un identificador variable.

Cuando un algoritmo se ejecuta, el valor que se asigna, mediante Assignment, puede visualizarse de forma inmediata en el recuadro de exhibición de identificadores y en forma diferida en la ventana de *MasterConsole*, utilizando el símbolo Output.

2.4.1 Asignación a un identificador constante

A continuación mostraremos los pasos a seguir para construir un diagrama que contemple la asignación de un valor a un identificador constante. En este caso, el identificador se llama **EDAD** y le vamos a asignar un valor de 45.

1. Abra el *Raptor* e inicie un nuevo archivo en el menú *File*. Seleccione con el botón izquierdo del ratón la opción *Save* del menú *File* para darle nombre al diagrama. En la ventana de *Save as*, en la caja de texto *File name*, escriba el nombre del archivo, que en este caso se recomienda: **Algoritmo2-2 asignación a un identificador constante**.
2. En la paleta de Symbols coloque el apuntador del ratón sobre el símbolo Assignment y presione el botón izquierdo del ratón, el contorno del Assignment se pondrá en color rojo.
3. Lleve el apuntador del ratón sobre la flecha que aparece entre los globos de Start y End de la pestaña *main*, hasta que el apuntador cambie a la forma de una manita  y presione nuevamente el botón izquierdo del ratón. Con lo anterior, el símbolo Assignment aparecerá entre los globos Start y End (véase figura 2.15).

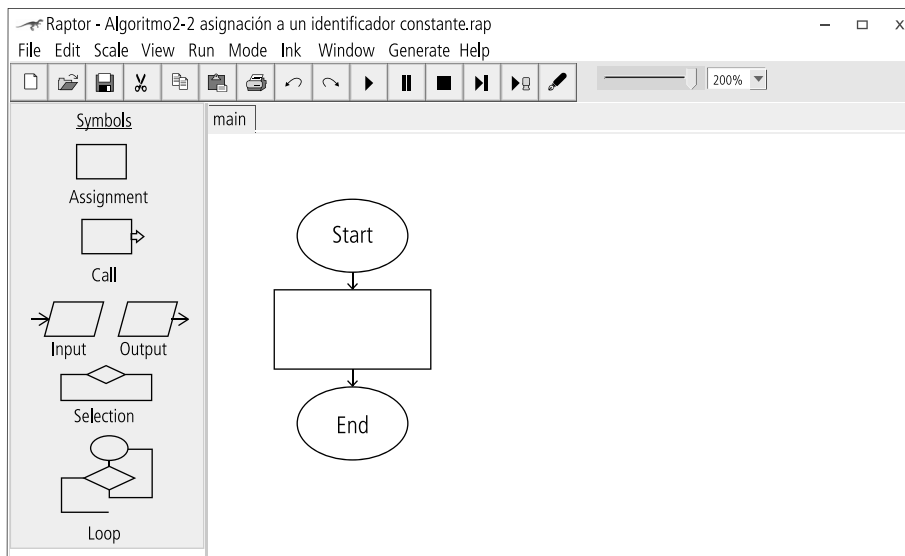


Figura 2.15 Pasos 1, 2 y 3. Símbolo Assignment.

4. Lleve el apuntador del ratón sobre la figura de Assignment que se encuentra entre los globos de Start y End, presione el botón izquierdo del ratón dando doble clic y aparecerá la ventana de diálogo llamada *Enter Statement*, que contiene dos cajas de texto. La primera se llama Set y la segunda, To.
5. En la caja de Set escriba el nombre del identificador que va a recibir el valor, en este caso, **EDAD**.
6. En la caja de To escriba el valor que se desea asignar al identificador **EDAD**, en este caso 45 (véase figura 2.16).

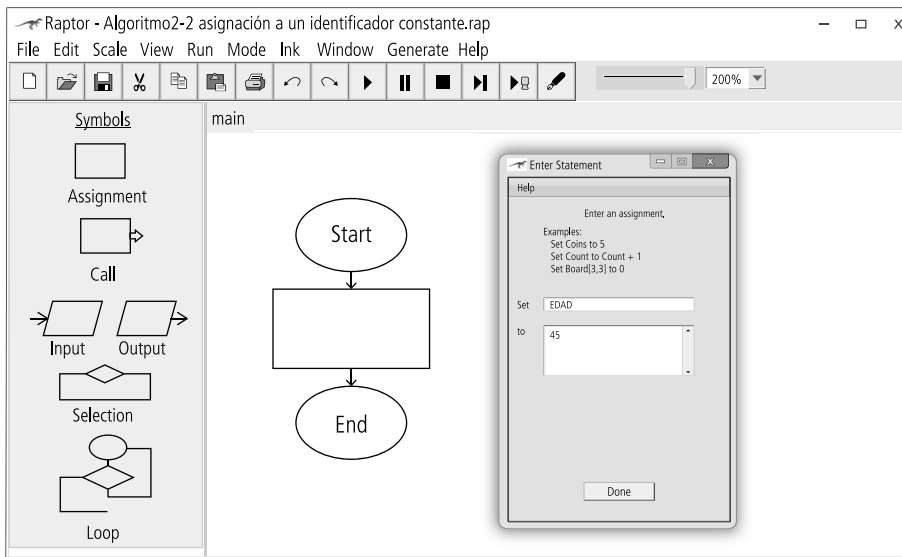


Figura 2.16 Pasos 4, 5 y 6. Assignment y ventana *Enter Statement*.

7. Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Statement* y presione el botón izquierdo del ratón. Con esto, se completa el llenado de las cajas de texto del símbolo Assignment.
8. Lleve el apuntador del ratón sobre el botón *Execute to Completion*, presione el botón izquierdo del ratón e inmediatamente vea el recuadro de exhibición de identificadores para que observe que el identificador **EDAD** ha tomado el valor: **edad: 45**. El programa Raptor en automático muestra el identificador en minúsculas, tal como se ve en la figura 2.17. En la ventana *MasterConsole* no se verá el valor del identificador **EDAD** porque no se ha utilizado el símbolo Output.

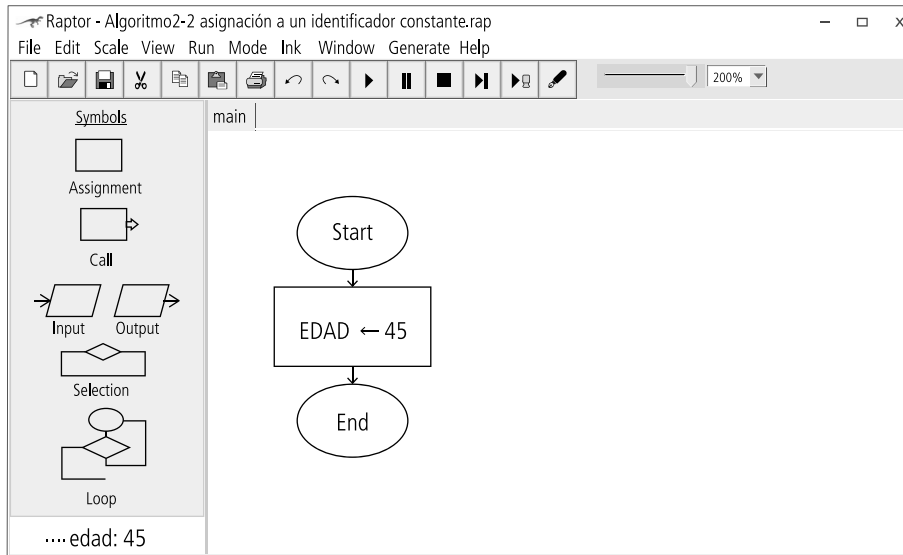


Figura 2.17 Pasos 7 y 8. Assignment. Algoritmo2-2 asignación a un identificador constante.

Si desea ver el resultado en *MasterConsole*, deberá completar el diagrama de acuerdo con la figura 2.18. Para ello, deberá utilizar el símbolo Output para que envíe un mensaje compuesto. Escriba, dentro del símbolo Output, la siguiente información: **“Tengo ” + EDAD + “ años de edad”**. A este nuevo diagrama le llamaremos: **Algoritmo2-3 asignación constante Mensaje compuesto** (véase figura 2.18).

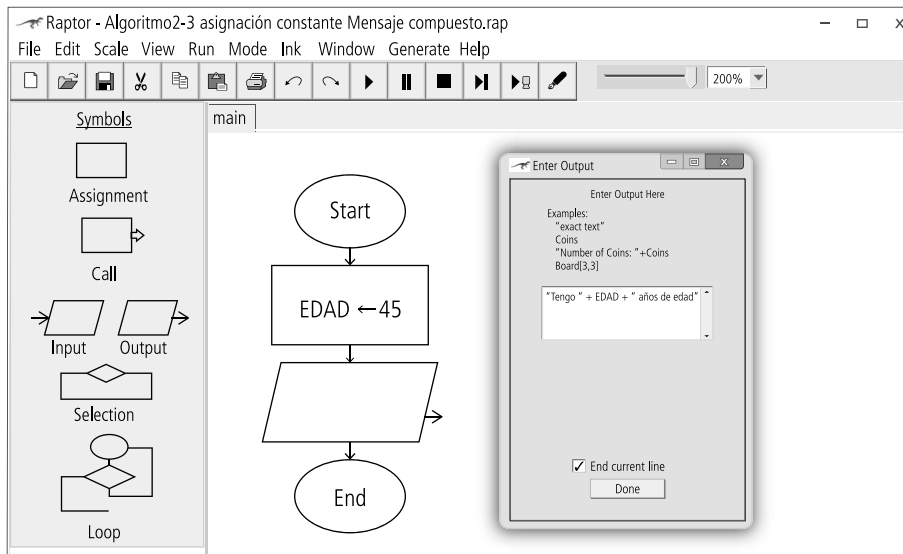


Figura 2.18 Agregar Output y ventana *Enter Output* con un mensaje compuesto.

Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Output* y presione el botón izquierdo del ratón. Con esto, se completa el llenado de la caja de texto del símbolo Output con un mensaje compuesto. El diagrama final deberá quedar de acuerdo con la figura 2.19.

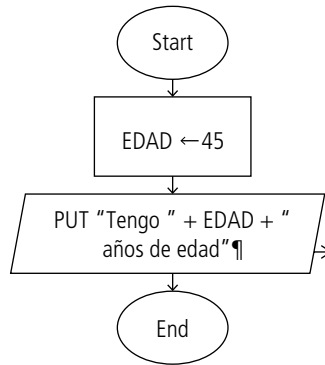


Figura 2.19 Algoritmo2-3 asignación constante Mensaje compuesto.

Lleve el apuntador del ratón sobre el botón *Execute to Completion*, presione el botón izquierdo del ratón y de manera automática se abrirá la ventana de *MasterConsole* mostrando el resultado de acuerdo con la figura 2.20.

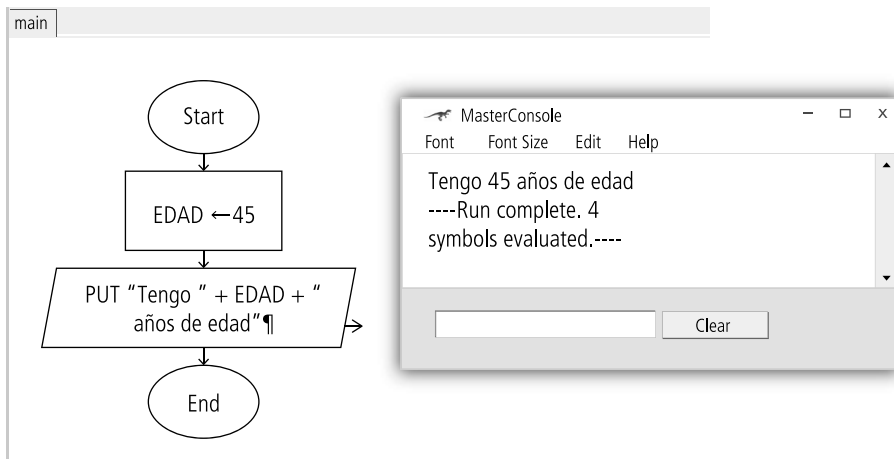


Figura 2.20 Asignación a una constante y visualización en *MasterConsole* de un mensaje compuesto.

En este caso, el mensaje compuesto está integrado por tres fragmentos de información.

Dos fragmentos son mensajes simples: “**Tengo** ” y “**años de edad**”. Al otro fragmento se le conoce como **detalle**, que es el valor del identificador **EDAD**. Los símbolos positivos (+) permiten concatenar, unir y juntar los fragmentos del mensaje. Por consiguiente, el mensaje compuesto está integrado así: **Mensaje simple1 +**

Identificador + Mensaje simple2. Para que el mensaje compuesto se visualice en la ventana de *MasterConsole*, sea legible y no luzca amontonado, al final del **Mensaje simple1** se ha dejado un espacio en blanco, también al principio del **Mensaje simple2** se ha dejado un espacio en blanco.

Observe el símbolo Output de la figura 2.20. No confunda la palabra edad que está dentro de las comillas con la que se encuentra fuera. Cuando una palabra está dentro de las comillas significa que es parte del texto del mensaje simple. Cuando está por fuera significa que es un identificador y forma parte del detalle.

2.5 Operadores matemáticos básicos

Antes de mostrar un ejemplo de una asignación variable mediante el símbolo Assignment, debemos conocer los operadores matemáticos. Recordemos que en una fórmula matemática se puede distinguir entre variables independientes, constantes y variable dependiente.

También necesitamos distinguir entre operandos y operadores. Los primeros son las variables independientes y constantes. Los segundos son los caracteres que representan una operación aritmética, como suma, resta, multiplicación, división, etcétera.

Los operadores aritméticos básicos de Raptor que se pueden utilizar en las fórmulas están listados en la tabla 2.1. Raptor tiene más operadores matemáticos, algunos de ellos los utilizaremos más adelante.

Los operadores aritméticos se pueden clasificar como:

1. **Binario.** Significa que el operador requiere como mínimo dos operandos.
2. **Unario.** Significa que el operador requiere como mínimo uno solo.

En la tabla 2.1 se puede ver la clasificación de cada uno de ellos.

Tabla 2.1 Operadores aritméticos básicos de Raptor

Operador	Acción	Tipo de operador	Nombre del carácter
+	Suma	Binario	Positivo
–	Resta	Binario	Guión medio
*	Multiplicación	Binario	Asterisco
/	División (cociente)	Binario	Barra inclinada
^	Exponenciación	Binario	Acento circunflejo
sqrt()	Raíz cuadrada	Unario	No aplica
%, MOD	División entera (residuo)	Binario	Porcentaje

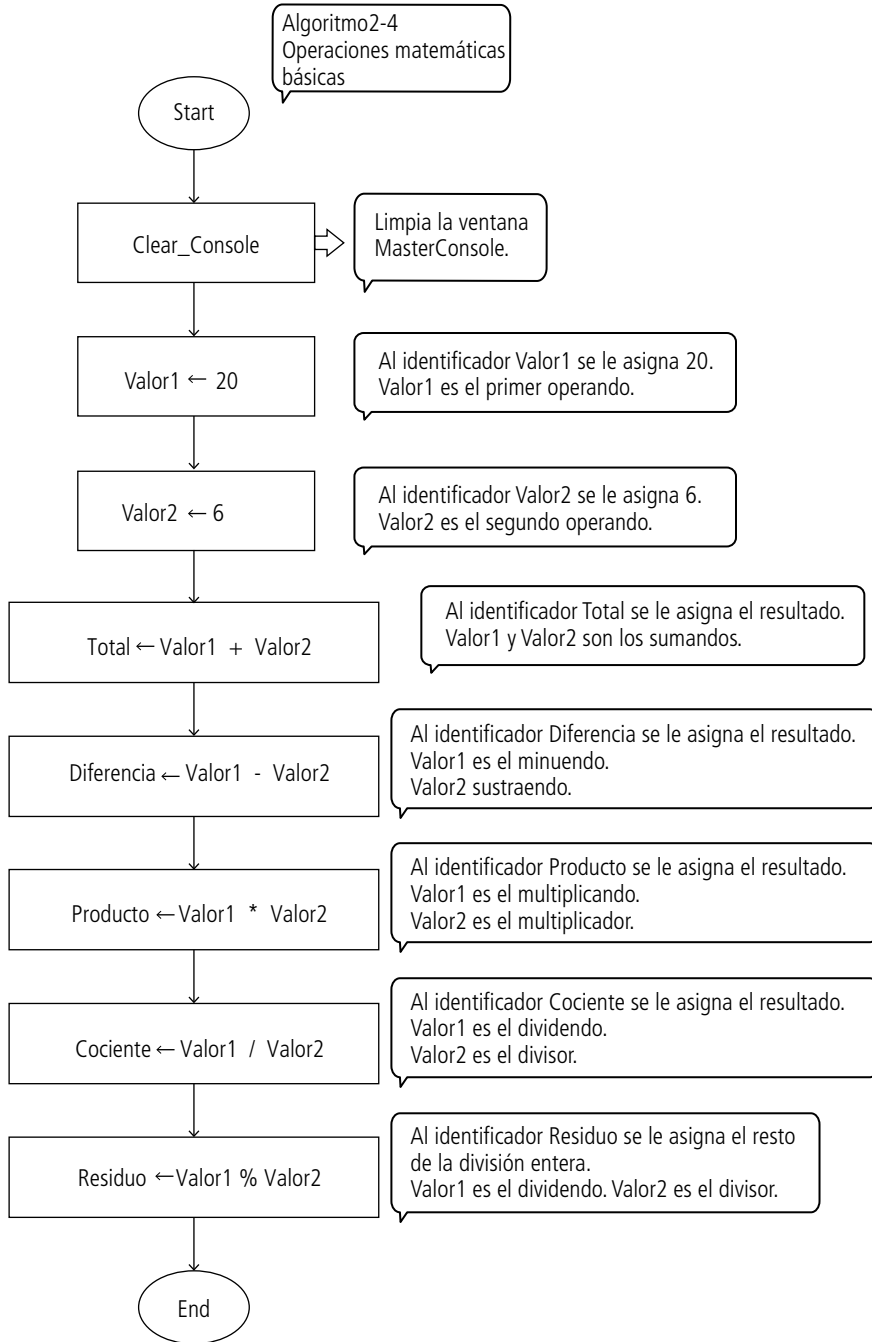
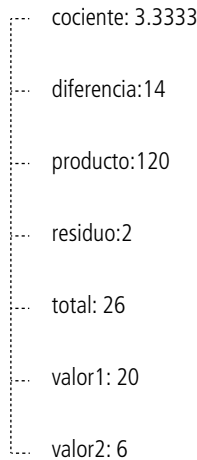


Figura 2.21 Algoritmo2-4 funcionamiento de las operaciones matemáticas básicas.

En el diagrama del **Algoritmo2-4** podrá ver cómo funcionan las operaciones aritméticas básicas de suma, resta, multiplicación, división y el residuo de la división entera. Los globos de diálogo que aparecen a un lado de los símbolos son comentarios y sirven para dar documentación interna a los diagramas, éstos se pueden ejecutar sin ellos. En este caso, se han agregado para que el usuario conozca qué hace cada símbolo. Más adelante, se mostrará cómo agregar estos comentarios (véase figura 2.21).

Al principio del diagrama, se ha colocado el símbolo Call y dentro de éste se escribió la función `Clear_Console` que sirve para limpiar la ventana de *MasterConsole*. El símbolo Call tiene una utilidad mucho más amplia, pero por el momento lo utilizaremos sólo con la función `Clear_Console`.

Presione el botón *Execute to Completion* y verá los resultados en el recuadro de exhibición de identificadores de acuerdo con la figura 2.22. Observe que las variables se muestran en orden alfabético y con letras minúsculas. Si desea ver los resultados en *MasterConsole* deberá agregar símbolos Output al diagrama.



```
... cociente: 3.3333
... diferencia: 14
... producto: 120
... residuo: 2
... total: 26
... valor1: 20
... valor2: 6
```

Figura 2.22 Resultados de las operaciones matemáticas básicas.

De las cinco operaciones básicas, observe que la variable **Residuo** arroja un resultado de 2, porque está mostrando el residuo de una división entera. Cambie los valores de las variables **Valor1** y **Valor2**, ejecute nuevamente el algoritmo y vea cómo se comportan los resultados del programa. Por ejemplo, si **Valor1** se queda con Veinte y **Valor2** cambia a 5, el **Cociente** será igual a 4 y el **Residuo** igual a 0.

Con el diagrama del **Algoritmo2-5** podrá observar cómo funcionan las operaciones matemáticas de elevar un número a una potencia y obtener la raíz cuadrada de un número (véase figura 2.23). En este caso, se han agregado los símbolos Output que muestran un mensaje compuesto, por lo que en *MasterConsole* podrá ver los resultados de acuerdo con la figura 2.24.

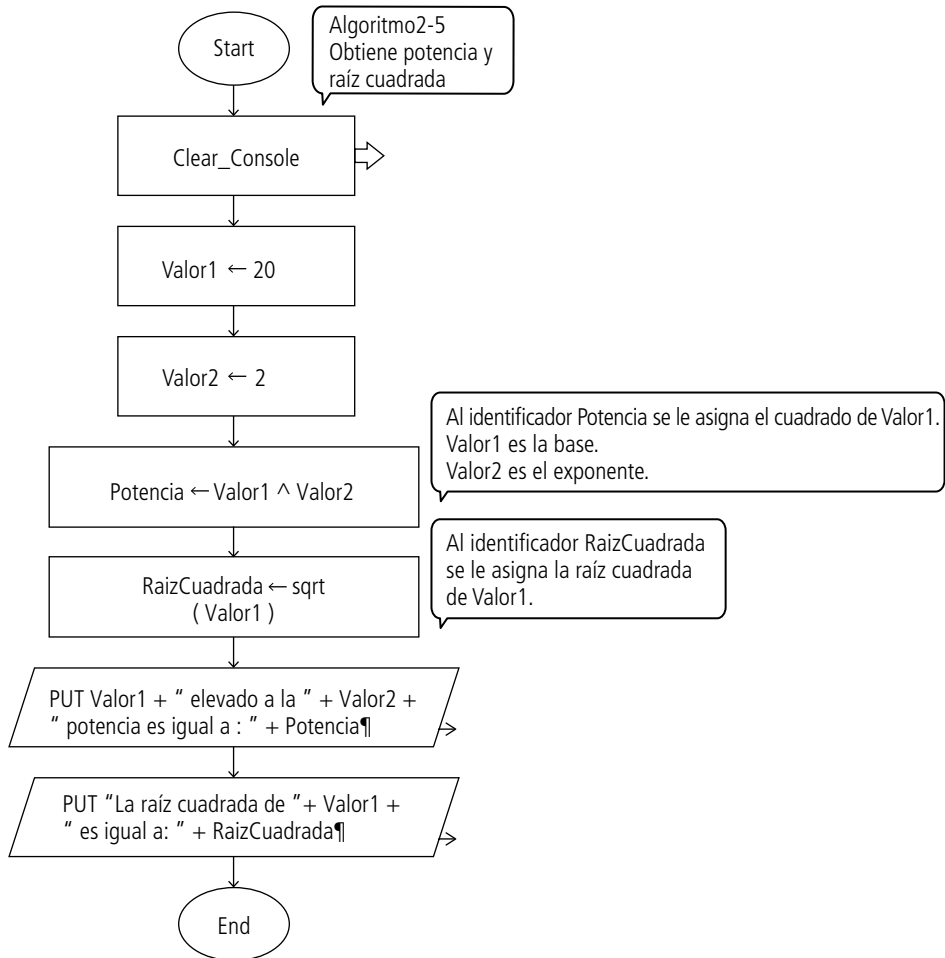


Figura 2.23 Algoritmo2-5 operaciones matemáticas para elevar una base a una potencia y obtener la raíz cuadrada de un número.

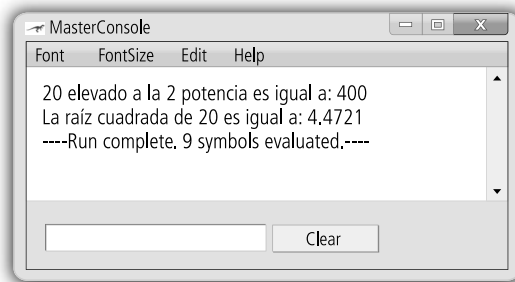



Figura 2.24 Visualización de resultados de potencia y raíz cuadrada.

2.6 Símbolo Input

Raptor utiliza el símbolo Input para representar la acción computacional de lectura de datos. En este caso, permite la captura manual de valores a un identificador por medio del teclado. Los valores se almacenan en una dirección o celda de la memoria RAM.

A continuación presentamos un ejemplo del uso del símbolo Input que permite agregar la edad de una persona. Se le pide que realice los siguientes pasos:

1. Abra el Raptor e inicie un nuevo archivo en el menú *File*. Seleccione con el botón izquierdo del ratón la opción *Save* del menú *File* para darle nombre al diagrama. En la ventana de *Save as*, en la caja de texto *File name*, escriba el nombre del archivo, que en este caso se recomienda: **Algoritmo2-6 inserción manual de un valor a una variable.**
2. En la paleta de Symbols coloque el apuntador del ratón sobre el símbolo Input y presione el botón izquierdo del ratón, verá que el contorno del Input se pondrá en color rojo.
3. Lleve el apuntador del ratón sobre la flecha que aparece entre los globos de Start y End, de la pestaña *main*, hasta que el apuntador cambie a la forma de una manita  y presione nuevamente el botón izquierdo del ratón. Con lo anterior, el símbolo Input aparecerá entre los globos Start y End (véase figura 2.25).

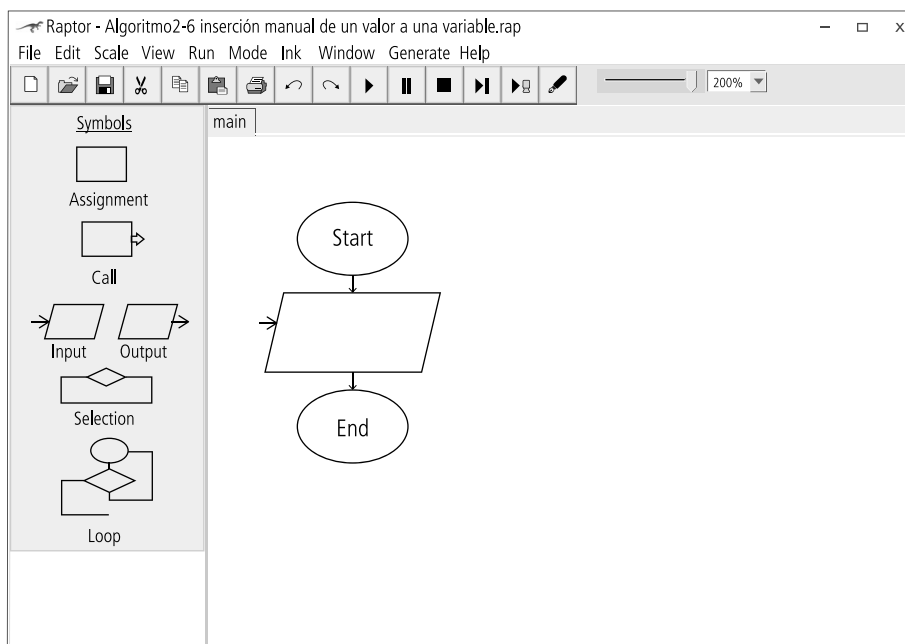


Figura 2.25 Pasos 1, 2 y 3. Símbolo Input.

4. Lleve el apuntador del ratón sobre la figura de Input que se encuentra entre los globos Start y End, presione el botón izquierdo del ratón dando doble clic y aparecerá la ventana de diálogo *Enter Input*, que contiene dos cajas de texto: *Enter Prompt Here* y *Enter Variable Here*.
5. En la caja de texto *Enter Prompt Here* escriba entre comillas el texto del mensaje simple que desee que aparezca en una ventana *Input* que le permitirá contextualizar el ingreso, mediante el teclado, de un valor a la memoria RAM y su asignación a una variable; es este caso, escribiremos la solicitud: “Favor de capturar su edad mediante el teclado”.
6. En la caja de texto *Enter Variable Here* escriba el nombre de la variable, en este caso **Edad** que recibirá el valor mediante del teclado (véase figura 2.26).

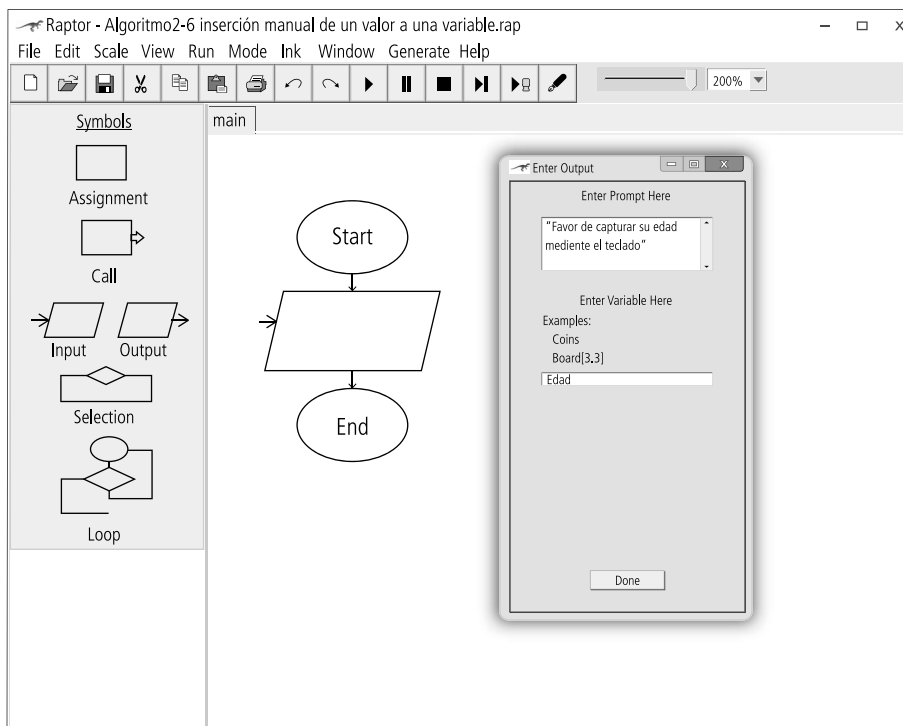


Figura 2.26 Pasos 4, 5 y 6. Símbolo Input y ventana *Enter Input*.

7. Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Input*, presione el botón izquierdo del ratón y verá la figura 2.27. Observe la palabra GET en medio del texto del mensaje de contextualización y la variable **Edad**. Antes de GET aparece el mensaje simple que se escribió en la caja de

texto *Enter Prompt Here*. Después de GET aparece lo que se escribió en la caja de texto *Enter Variable Here*, que es el nombre de la variable que va a recibir el valor por medio del teclado.

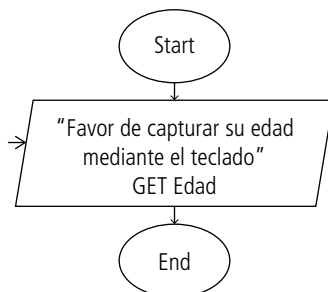


Figura 2.27 Paso 7. Input.

8. Realice los pasos descritos en la sección correspondiente para el símbolo Output, a fin de poder visualizar, en el *MasterConsole*, un mensaje compuesto. Escriba en la caja de texto de la ventana *Enter Output* lo siguiente: “Tu edad es de” + **Edad** + “ años”.
9. Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Output* y lo que verá es lo que se observa en la figura 2.28.

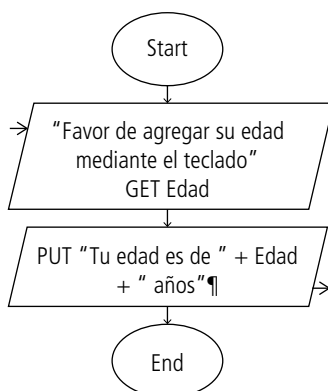


Figura 2.28 Pasos 8 y 9. Algoritmo2-6 inserción manual de un valor a una variable.

10. Lleve el apuntador del ratón sobre el botón *Execute to Completion*, presione el botón izquierdo del ratón y aparecerá la ventana de *Input*, donde podrá ver el mensaje de contextualización; en la caja de texto deberá escribir la información solicitada, en este caso: 45 (véase figura 2.29).

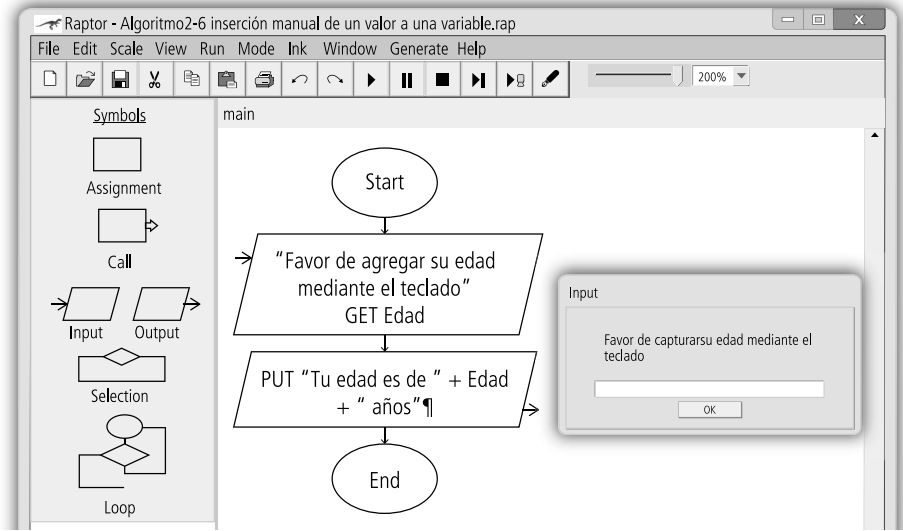


Figura 2.29 Paso 10.

11. Después de agregar el dato solicitado, lleve el apuntador del ratón sobre el botón *OK* de la ventana *Input* y presione el botón izquierdo del ratón. Observe el recuadro de exhibición de identificadores, en el cual podemos ver el valor almacenado en la variable. Finalmente, observe o active la ventana de *MasterConsole* y verá la imagen de la figura 2.30.

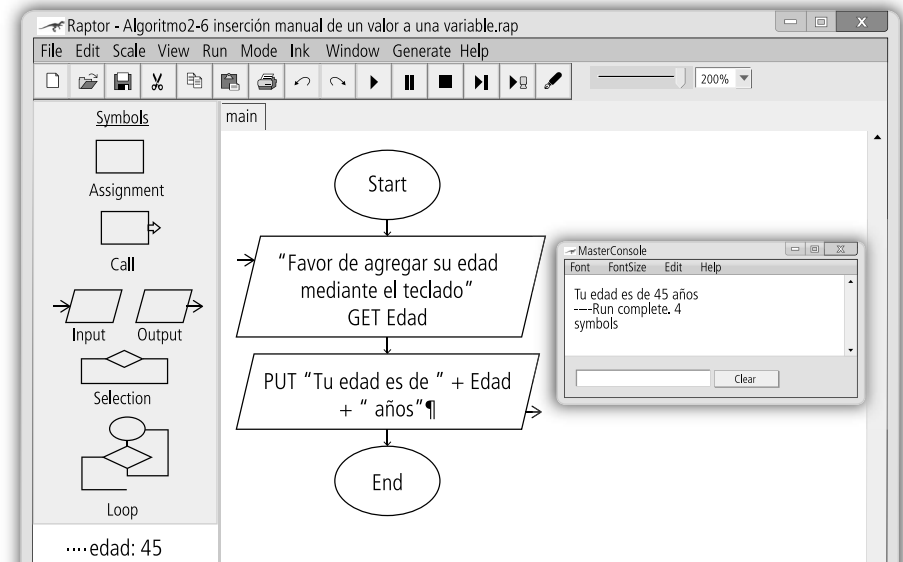


Figura 2.30 Paso 11.

El símbolo Input no puede verificar si el valor capturado es válido o no. En el caso del **Algoritmo 2-6 inserción manual de un valor a una variable**, se agregó la edad de la persona en forma numérica. Ya vimos que el diagrama funciona, sin embargo, podríamos capturar la edad en forma alfabética, es decir, cuarenta y cinco, y el diagrama como quiera funcionaría.

También podríamos capturar una edad, ya sea numérica o alfabética, negativa o muy superior a lo que humanamente es posible vivir y el diagrama como quiera funcionaría. Algo peor es que podríamos capturar cualquier cosa y como quiera, pero enviaría un resultado sin lógica, simplemente estaría fuera de la racionalidad, la realidad y la posibilidad.

Otra cosa que hay que tener en cuenta es que si los valores capturados con el símbolo Input se utilizan para realizar una operación matemática, pero al momento de la ejecución del programa el usuario agrega valores no numéricos, lo que sucederá es que se generará un error de ejecución.

En conclusión, el símbolo Input no tiene la capacidad de verificar los valores de los datos capturados. Sin embargo, Raptor tiene funciones que permiten saber si el dato capturado es numérico o no, pero dejaremos ese tema para más adelante.



2.7 Documentación interna en los diagramas

Los comentarios en los diagramas son útiles para documentar internamente el algoritmo, pero no son indispensables. Visualmente los podemos reconocer porque su representación gráfica es semejante a los globos de diálogo, que son rectángulos con sus cuatro vértices redondeados y una pequeña pestaña cerca de una de las esquinas. Éstos aparecen apuntando con su pestaña a los símbolos, pero no interfieren en la ejecución de los mismos.

Los comentarios pueden ser eliminados y los diagramas como quiera funcionan; sirven para hacer aclaraciones o para tener una mejor comprensión de los pasos que se realizan en un diagrama, o para conocer los detalles que se desee agregar, como el nombre del programador o la fecha de realización. Para agregar comentarios a un diagrama de flujo realice los siguientes pasos:

1. Obtenga cualquier diagrama de Raptor. En este caso, se abrirá el Algoritmo2-6 inserción manual de un valor a una variable.rap.
2. Lleve el apuntador del ratón sobre el símbolo en el que desee colocar el comentario. En este caso, se hará sobre el globo de Start.
3. Dé clic derecho con el ratón, aparecerá un menú auxiliar y en éste seleccione la opción *Comment*.
4. Aparecerá la ventana *Enter Comment*, entonces escriba su comentario. En este caso, se escribirá: **Algoritmo2-6**. Permite capturar manualmente la edad de una persona y la muestra en la ventana de *MasterConsole*.
5. Lleve el apuntador del ratón sobre el botón *Done* de la ventana *Enter Comment*, presione el botón izquierdo del ratón y verá la figura 2.31.

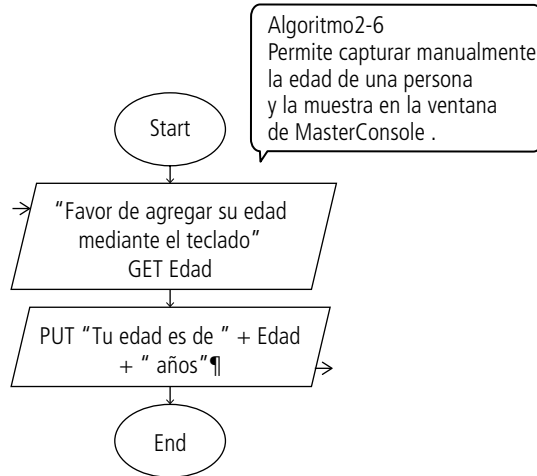


Figura 2.31 Pasos 1, 2, 3, 4 y 5. Comentario agregado en el globo Start.

El diagrama del **Algoritmo2-7** es otro ejemplo de un diagrama que utiliza sólo Input y Output. Este diagrama permite la captura del nombre y la edad de una persona mediante el teclado y ver el mensaje compuesto en *MasterConsole* (véase figura 2.32).

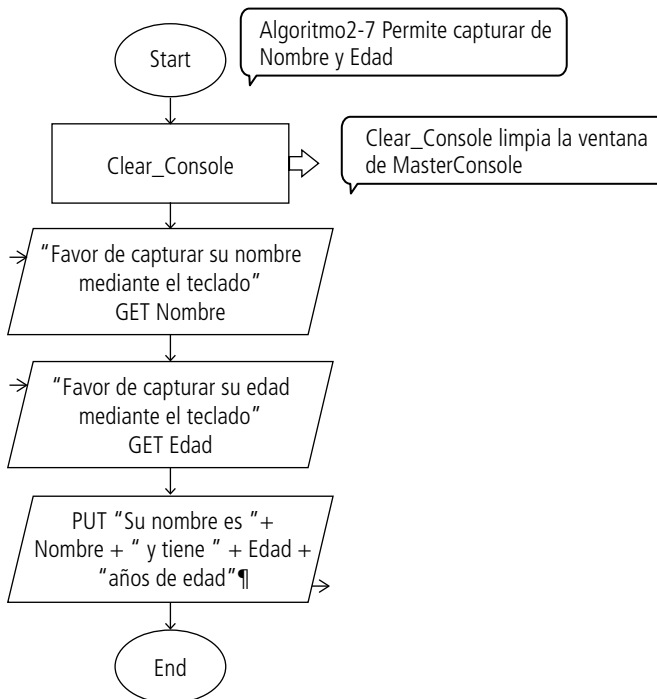


Figura 2.32 Algoritmo2-7 captura de Nombre y Edad.



2.8 Desarrollo de algoritmos básicos

El diseño básico de algoritmos computacionales se realiza con las acciones de lectura de datos, proceso de datos e impresión de resultados, las cuales se representan de manera gráfica con los símbolos de Raptor de Input, Assignment y Output, respectivamente. A partir de este momento, al nombre de los algoritmos se les agregará el sufijo **-básico** para señalar que pertenecen al tema de diseño básico de algoritmos.

Con el **Algoritmo2-8-básico** se calculará la fuerza a partir de la captura de la masa y la aceleración, cuyos valores deben multiplicarse. En esta fórmula tenemos dos variables independientes y una dependiente.

Primero, los valores de las variables independientes se van a capturar a la memoria RAM usando el símbolo Input, es decir, los valores se insertarán usando el teclado. Se requiere un símbolo Input por cada variable, por lo que se utilizarán dos Input. Posteriormente, se utilizará un símbolo Assignment, donde se plasma la fórmula matemática. Finalmente, usando el símbolo Output, se envía el mensaje compuesto para poder verlo en la ventana de *MasterConsole*.

Se deben considerar las restricciones de los valores que se pueden capturar a las variables independientes. Primero, a la variable masa sólo se le debe agregar valores positivos, de lo contrario estaríamos fuera de las leyes de la física. Segundo, la variable aceleración puede aceptar valores positivos y negativos. Finalmente, ambas variables pueden tomar valores enteros o racionales, es decir, con decimales.

El usuario de este programa puede agregar valores negativos a la masa y el diagrama funcionaría. Esto es, el valor almacenado en fuerza sería un valor negativo. Aunque físicamente la fuerza puede ser negativa, esto sólo es posible si el valor negativo pertenece a la aceleración. Por tanto, aunque el valor de fuerza es posible, por el valor original de los datos se tendría que desechar el resultado por estar fuera de las leyes de la física (véase figura 2.33).

Es posible hacer un diagrama que se “defienda” de usuarios que capturan valores no válidos a ciertas variables, pero implicaría una mayor complejidad; este tema se abordará más adelante.

El diagrama del **Algoritmo2-9-básico** permite hacer conversiones de distancias de pies a metros. El dato conocido es la cantidad de pies lineales de los que se desea conocer su equivalente en metros lineales. La restricción es que el valor que se capture a la variable pies debe ser mayor que cero (véase figura 2.34).

Por otro lado, con el **Algoritmo2-10-básico** se calcula la velocidad de un objeto que va hacia el suelo en caída libre al momento del impacto. El dato conocido es la altura en metros de donde se deja caer el objeto. La gravedad es una constante que tiene un valor de 9.8 metros/segundo². Al principio del diagrama se usa un símbolo Assignment para dar el valor a la constante. La restricción es que el valor que se agregue a la variable altura debe ser mayor que cero, de lo contrario el radicando tomará un valor negativo y el algoritmo generará un error de ejecución (véase la figura 2.35). La fórmula que permite calcular la velocidad final es:

$$\text{Velocidad final} = \sqrt{2 * \text{GRAVEDAD} * \text{Altura}}$$

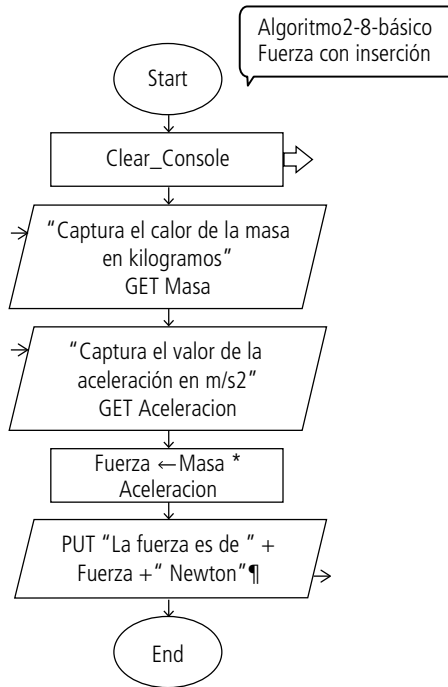


Figura 2.33 Algoritmo2-8-básico calcula la fuerza con captura de a variables independientes, asignación a la variable dependiente y la impresión de un mensaje compuesto.

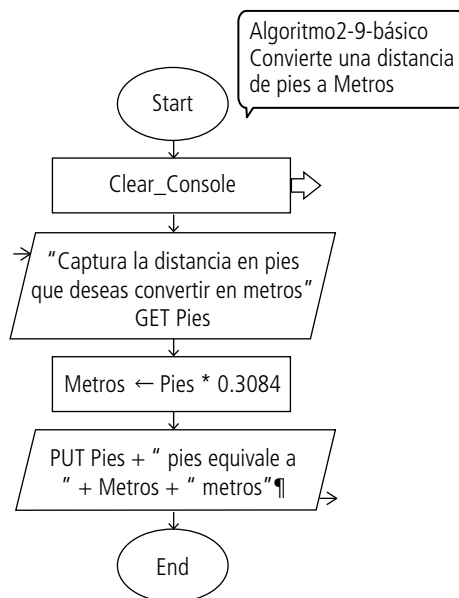


Figura 2.34 Algoritmo2-9-básico convierte una distancia de pies a metros.

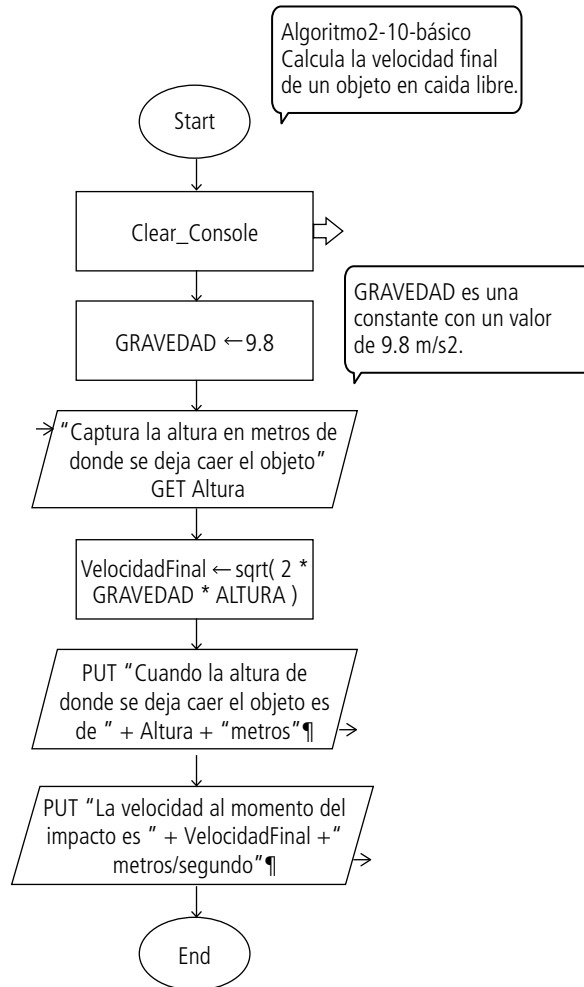


Figura 2.35 Algoritmo2-10-básico calcula la velocidad final de un objeto en caída libre.

Con el **Algoritmo2-11-básico** se calcula el área de un cuadrado si se tiene como dato conocido la longitud de sus lados en metros, que se debe elevar al cuadrado para obtener el resultado. La restricción es que el valor capturado a la variable lado debe ser mayor que cero (véase la figura 2.36).

Con el **Algoritmo2-12-básico** se calcula el área de un círculo si se tiene como dato conocido el diámetro del círculo en metros. La restricción es que el valor capturado a la variable Diámetro debe ser mayor que cero, el cual debe dividirse entre 2 para obtener el radio del círculo. Para calcular el área del círculo, el valor del Radio debe elevarse al cuadrado y multiplicarlo por el valor de PI. Observe que PI es una constante definida en Raptor, de modo que no es necesario asignarle valor porque ya lo tiene (véase figura 2.37).

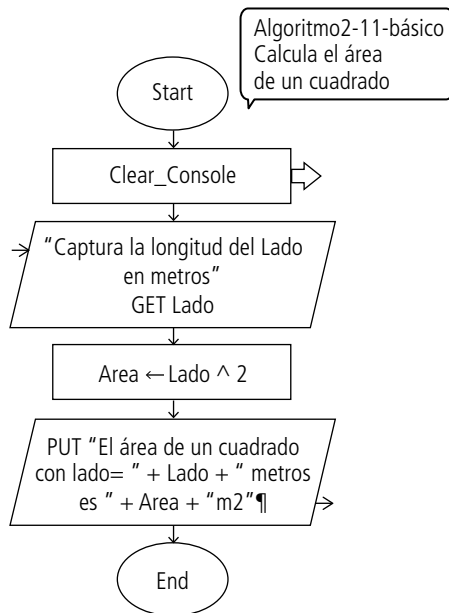


Figura 2.36 Algoritmo2-11-básico calcula el área de un cuadrado.

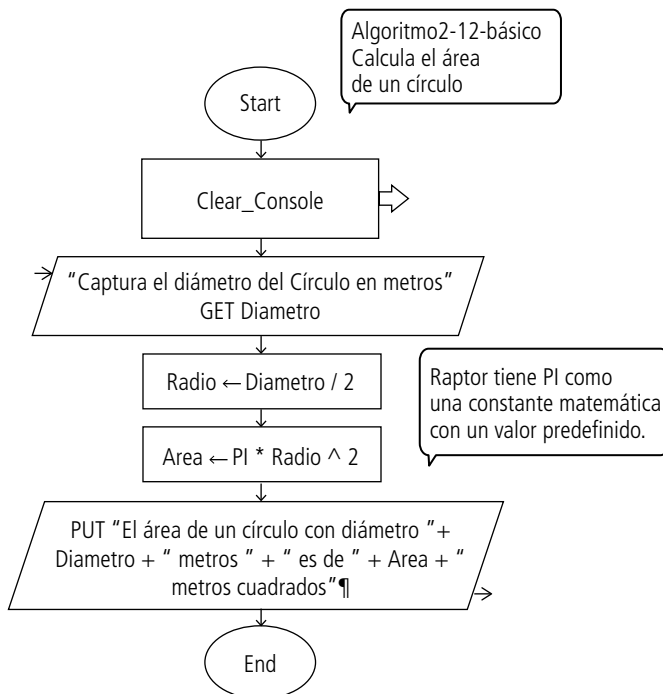


Figura 2.37 Algoritmo2-12-básico calcula el área de un círculo.

Con el **Algoritmo2-13-básico** se calcula el área de un triángulo si se tiene como datos conocidos la base y la altura del triángulo en metros. La restricción es que el valor asignado a las variables Base y Altura debe ser mayor a cero. Para obtener el área del triángulo ambos valores deben multiplicarse entre sí y finalmente dividir entre 2 esa cifra (véase figura 2.38).

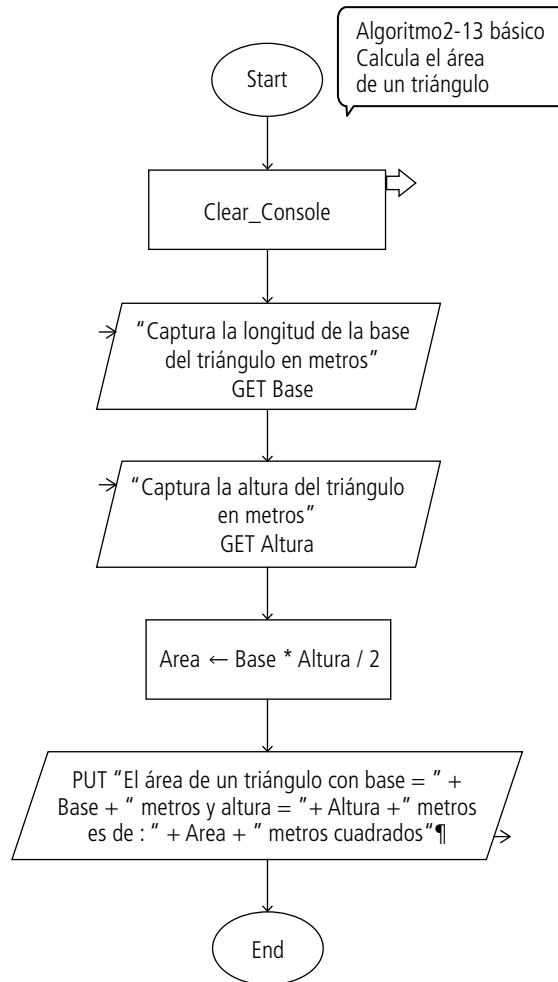


Figura 2.38 Algoritmo2-13-básico calcula el área de un triángulo.

Con el **Algoritmo2-14-básico** se pueden hacer conversiones de grados Fahrenheit a grados centígrados. En este caso, no hay restricciones, ya que la variable Fahrenheit puede tomar valores positivos o negativos, así como enteros o racionales (véase figura 2.39).

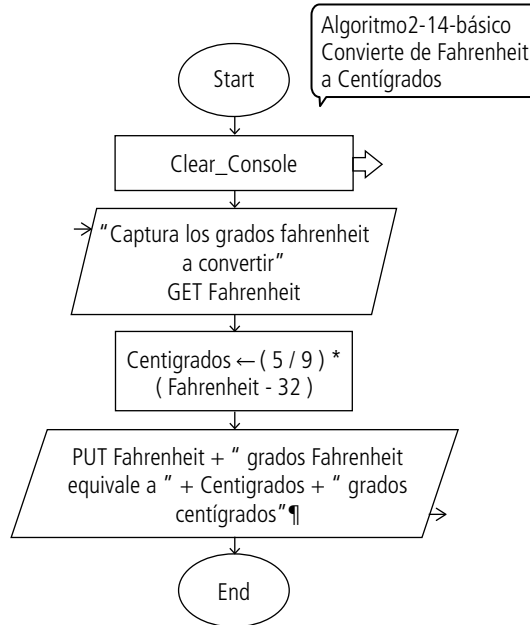


Figura 2.39 Algoritmo2-14-básico convierte de grados Fahrenheit a grados centígrados.

Enseguida, con el diagrama del **Algoritmo2-15-básico**, calculamos el área de un triángulo si se tiene como datos conocidos la longitud en metros de cada uno de sus lados.

La fórmula para determinar el área es:

$$\text{Área} = \sqrt{\text{Lados}(\text{Lados} - \text{Lado1})(\text{Lados} - \text{Lado2})(\text{Lados} - \text{Lado3})}$$

Donde:

$$\text{Lados} = \frac{(\text{Lado1} + \text{Lado2} + \text{Lado3})}{2}$$

Las restricciones son:

1. Los datos conocidos deben ser mayores a cero.
2. Los datos conocidos deben formar un triángulo. De lo contrario, el radicando será negativo, lo cual generará un error de ejecución porque no se puede obtener la raíz cuadrada.

Esto es, si Lado1 = 4, Lado2 = 5 y Lado3 = 8 metros, entonces estos lados sí pueden formar un triángulo; pero si Lado1 = 4, Lado2 = 5 y Lado3 = 12 metros, entonces estos lados no pueden formar un triángulo. En conclusión, si la sumatoria de los lados más pequeños es mayor que el lado más grande, se puede formar un triángulo. Para el primer caso, $4 + 5 = 9$, dado que 9 es mayor que 8, se forma el triángulo; pero en el segundo caso, donde $4 + 5 = 9$ y dado que 9 es menor que 12, no se puede formar el triángulo (véase figura 2.40).

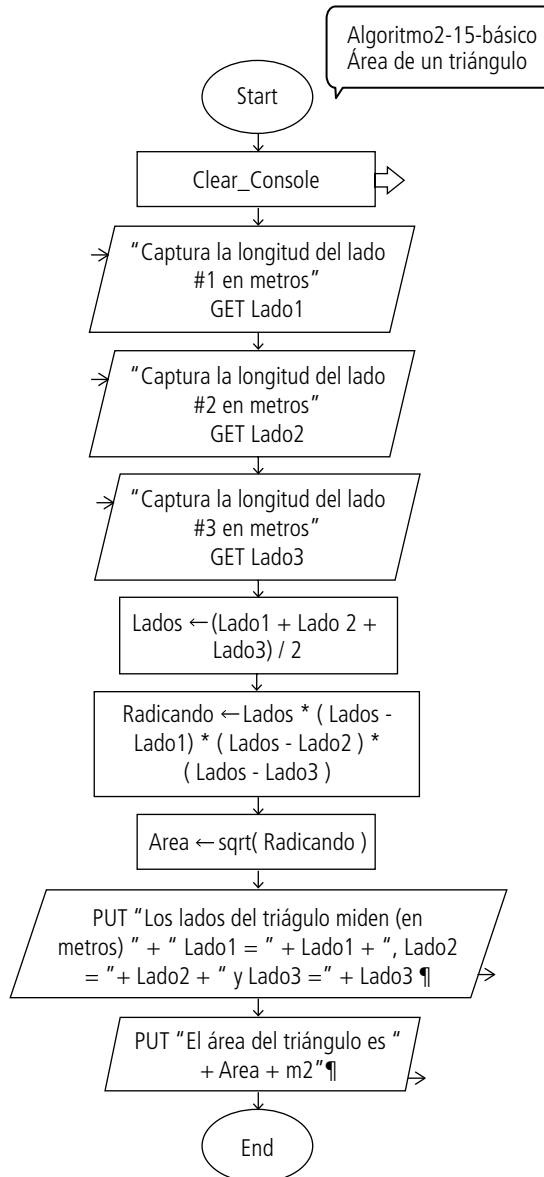


Figura 2.40 Algoritmo2-15-básico calcula el área de un triángulo. Se conoce la longitud de sus tres lados.

Finalmente, en el diagrama del **Algoritmo2-16-básico** se resuelve el siguiente problema. Un avión que se encuentra volando a una altura de 10,000 pies y a 150 millas por hora desarrolla velocidades relativas con respecto al viento de 300 millas por hora en la parte superior del ala y de 80 millas por hora debajo del ala. Si el área del ala es de 160 pies², ¿cuál será la fuerza perpendicular a ella?

Suponiendo que la densidad a 10,000 pies de altura es $\rho = 0.001756$ slugs/pies³. La fórmula es:

$$\text{Fuerza} = A * \frac{1}{2} \rho (V1^2 - V2^2)$$

Donde:

A = área del ala del avión: 160 pies².

ρ = densidad a 10,000 pies de altura: 0.00175.

$V1$ = velocidad relativa del viento en la parte superior del ala: 300 millas por hora.

$V2$ = velocidad relativa del viento en la parte inferior del ala: 80 millas por hora. El factor que debe usarse para convertir las millas/hora a pies/segundo es 44/30 (véase figura 2.41).

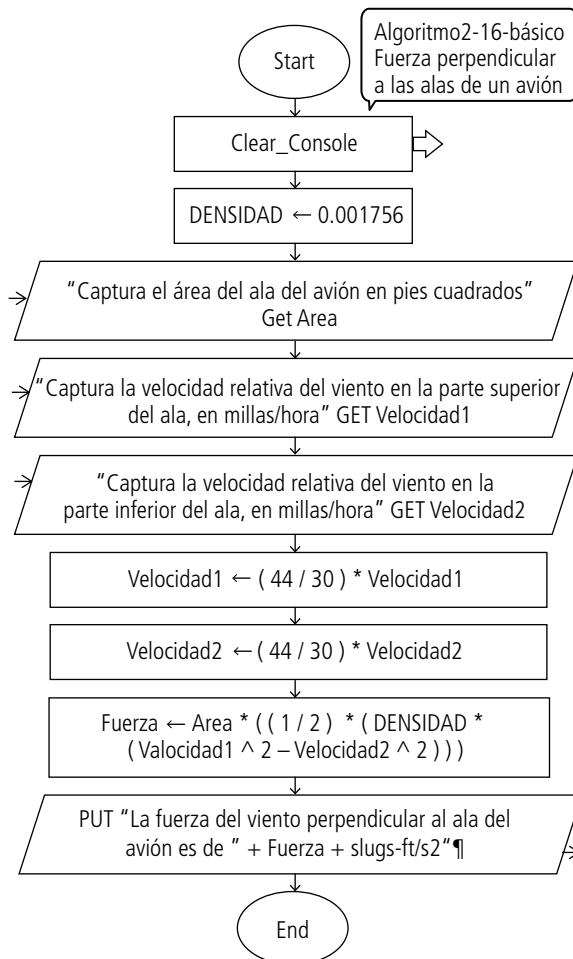


Figura 2.41 Algoritmo2-16-básico calcula el valor de la fuerza perpendicular al ala de un avión.



2.9 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 2.2.

- 2.1 ¿Cómo se les denomina a las herramientas que han permitido la automatización de las actividades del proceso de software reduciendo el costo económico, el tiempo y el esfuerzo de desarrollo, así como incrementar la productividad y competitividad de las personas y empresas que se dedican al desarrollo de software?
- 2.2 ¿Cómo se llama el botón de la interfaz principal de Raptor que permite ejecutar las acciones computacionales que componen un diagrama de flujo?
- 2.3 ¿Cómo se llama la pestaña de la interfaz principal de Raptor, la cual, de manera predefinida, se muestra al usuario que desea crear un diagrama de flujo, e incluye los globos elipsoidales de Start y End?
- 2.4 ¿Cómo se denomina al apartado de la interfaz principal de Raptor donde aparecen de manera predefinida todos los bloques de diagrama de flujo que el usuario puede utilizar para construir algoritmos computacionales?
- 2.5 ¿Cuál es el título de la ventana que es la interfaz secundaria de Raptor en la que se pueden exhibir mensajes y valores del estado actual de las variables almacenadas en la memoria RAM?
- 2.6 ¿Cómo se denomina el símbolo de Raptor que es la representación gráfica de la acción computacional de lectura de datos: un paralelogramo que tiene una flecha que “entra” en la esquina superior izquierda?
- 2.7 ¿Cómo se denomina el símbolo de Raptor que es la representación gráfica de la acción computacional de proceso de datos: un rectángulo que permite la asignación de un valor a una variable y la ejecución de operaciones matemáticas?
- 2.8 ¿Cómo se denomina el símbolo de Raptor que es la representación gráfica de la acción computacional de impresión de resultados: un paralelogramo que tiene una flecha que “sale” de la esquina inferior derecha?
- 2.9 ¿Cómo se denomina la tarea en la que Output permite visualizar en *MasterConsole* el valor de un identificador sin un mensaje que lo contextualice?
- 2.10 ¿Cómo se denomina la tarea en la que Output permite visualizar en *MasterConsole* la contextualización de una situación y no incluye mostrar valores de identificadores?

- 2.11 ¿Cómo se denomina la tarea en la que Output permite visualizar en *MasterConsole* valores de identificadores debidamente contextualizados con un mensaje de referencia?
- 2.12 ¿Cómo se denomina el tipo de asignación de un valor a un identificador que se hace utilizando el símbolo Assignment y no implica la ejecución de una operación aritmética?
- 2.13 ¿Cómo se denomina el tipo de asignación de un valor a un identificador que se hace usando el símbolo Assignment y que implica la ejecución de una operación aritmética?
- 2.14 ¿Cuál es la variable de una fórmula matemática que debe colocarse en la caja de texto llamado Set del símbolo Assignment y que recibirá el valor después de que se ejecuten todas las operaciones matemáticas que tenga la fórmula?
- 2.15 Cuando el carácter de positivo se utiliza en un símbolo de Assignment se refiere a la operación aritmética de la suma, pero cuando se usa dentro del símbolo Output se refiere a la operación de...
- 2.16 ¿Cómo se denomina el carácter que se refiere a la suma aritmética?
- 2.17 ¿Cómo se denomina el carácter que se refiere a la resta aritmética?
- 2.18 ¿Cómo se denomina el carácter que se refiere a la multiplicación aritmética?
- 2.19 ¿Cómo se denomina el carácter que se refiere a la división aritmética?
- 2.20 ¿Cómo se denomina el carácter que se refiere a la exponenciación aritmética?
- 2.21 ¿Cómo se denomina el carácter que se refiere a obtención de residuo de una división aritmética entera?
- 2.22 ¿Cómo se llama la caja de texto del símbolo Input, que permite escribir mediante el teclado el mensaje que contextualiza el ingreso de un valor a la memoria RAM y su asignación a una variable?
- 2.23 ¿Cómo se llama la caja de texto del símbolo Input en que se escribe el nombre del identificador que recibirá un valor usando el teclado?
- 2.24 Cuando se han llenado las dos cajas de texto del símbolo Input, hay una palabra que Raptor coloca de manera automática entre el mensaje de contextualización y el nombre de la variable que recibirá el valor. ¿Cuál es esa palabra?

- 2.25** ¿Cómo se llaman los globos de diálogo que pueden agregarse a los diagramas de flujo y sirven para documentar internamente los algoritmos? Son rectángulos con los cuatro vértices redondeados y una pestaña cerca de una de las esquinas, los cuales no interfieren en la ejecución de los algoritmos.
- 2.26** ¿Cómo se llama la función que se escribe dentro del símbolo Call y permite limpiar la ventana de *MasterConsole*?
- 2.27** ¿Qué tipo de error se genera cuando un símbolo Assignment intenta hacer una división entre cero u obtener la raíz cuadrada de un radicando negativo?

Tabla 2.2 Respuestas a los ejercicios de autoevaluación

2.1 <i>Computer Aided Software Engineering (CASE)</i>	2.2 <i>Execute to Completion</i>	2.3 <i>main</i>
2.4 Symbols	2.5 <i>MasterConsole</i>	2.6 Input
2.7 <i>Assignment</i>	2.8 Output	2.9 Exhibición de datos
2.10 Mensaje simple	2.11 Mensaje compuesto	2.12 Asignación constante
2.13 Asignación variable	2.14 La variable dependiente	2.15 Concatenar
2.16 Positivo	2.17 Guión medio	2.18 Asterisco
2.19 Barra inclinada	2.20 Acento circunflejo	2.21 Porcentaje
2.22 <i>Enter Prompt Here</i>	2.23 <i>Enter Variable Here</i>	2.24 <i>GET</i>
2.25 Comentarios	2.26 <i>Clear_Console</i>	2.27 Error de ejecución



2.10 Problemas propuestos

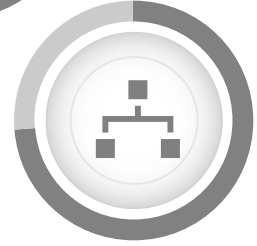
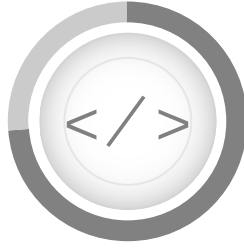
- 2.1** Elabore un algoritmo que pida el nombre del usuario y le dé la bienvenida a la clase.
- 2.2** Realice un algoritmo que eleve al cuadrado y al cubo cualquier número, y que imprima el número junto a su cuadrado y su cubo.
- 2.3** Elabore un algoritmo que determine la energía potencial de un cuerpo.

$$\text{Energía potencial} = \text{Masa} * \text{Altura} * \text{Gravedad}$$

- 2.4** Use un algoritmo para determinar la altura en problemas de caída libre.

$$\text{Altura} = \text{Velocidad inicial} * \text{Tiempo} + (\text{Gravedad} * \text{Tiempo}^2)/2$$

- 2.5** Calcule la calificación final de un estudiante si la ponderación en la materia es: Examen1: 30%, Examen2: 50% y Tareas: 20%.
- 2.6** Antes de despegar un avión, el piloto anuncia el tiempo estimado de vuelo en minutos. Realice un programa que le ayude a determinar el porcentaje de avance del vuelo, si tiene como dato conocido el tiempo transcurrido en minutos.
- 2.7** Un maestro desea determinar el porcentaje de aprobados y reprobados en un grupo; sólo sabe cuántos alumnos han aprobado y cuántos han reprobado. Realice un programa que le ayude a calcular estos porcentajes.
- 2.8** Una maestra midió la altura de Juan al principio y al final del año escolar. Realice un programa que le ayude a determinar el porcentaje de crecimiento.



Capítulo 3

Diseño de algoritmos con la estructura selectiva

- 3.1 Introducción
- 3.2 Operadores relacionales y lógicos
- 3.3 Símbolo Selection
- 3.4 Algoritmos con la estructura selectiva doble
- 3.5 Deficiencias en los algoritmos
- 3.6 Anidamiento de la estructura selectiva
- 3.7 Ejercicios de autoevaluación
- 3.8 Problemas propuestos

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Diseñar algoritmos computacionales utilizando la acción computacional de selección de alternativas.
- Utilizar la estructura selectiva doble de manera simple, compuesta y anidada.
- Plantear expresiones matemáticas de condición para la toma de decisiones.
- Aplicar la estructura de selección de alternativas utilizando los operadores relacionales y lógicos.
- Verificar que los valores de los identificadores de variables cumplen con el dominio adecuado, de acuerdo con el contexto y las restricciones del problema.



3.1 Introducción

A la acción computacional que permite configurar diferentes opciones mutuamente excluyentes y que se pueden elegir a partir de expresiones matemáticas condicionales se le llama **selección de alternativas** o **estructura selectiva**. Estas expresiones matemáticas se plantean con **operadores relacionales** y **lógicos**, y se les denomina con el nombre genérico de condición. Las condiciones matemáticas pasan por un proceso de evaluación por medio del cual se determina su veracidad o falsedad.

Las estructuras selectivas pueden ser dobles o múltiples. La diferencia radica en la cantidad de opciones diferentes a elegir. Son múltiples cuando hay más de dos opciones, y dobles cuando sólo hay dos.

La estructura selectiva doble se puede implementar de manera simple o compuesta. La forma simple es cuando únicamente por una de sus dos opciones se realizan una o varias acciones, y la compuesta es cuando por ambas opciones se realizan una o varias acciones.

Una característica de cualquier estructura selectiva es que independientemente de cuál camino se tome, una vez que se han realizado las acciones correspondientes a la opción seleccionada, las acciones que continúan son las mismas que para las demás opciones.

Raptor cuenta con un símbolo que permite la implementación de una estructura selectiva doble. El símbolo Selection habilita la selección de un camino a partir de dos posibles y que son mutuamente excluyentes. En este capítulo se presentarán diversos ejemplos para aprender a usar este símbolo.



3.2 Operadores relacionales y lógicos

Los operadores relacionales y lógicos permiten la construcción de expresiones matemáticas condicionales que se aplican en la toma de decisiones cuando se tienen opciones diversas de proceso o acción.

Los operadores relacionales permiten plantear expresiones matemáticas que se refieren a las relaciones de condición y orden que las entidades abstractas pueden tener unas con respecto a otras (véase tabla 3.1). Se emplean para comprobar la veracidad o falsedad de determinadas propuestas de relación. En realidad se trata de encontrar respuestas a preguntas de comparación o clasificación entre las entidades o los valores asignados a sus identificadores. Estos operadores son binarios, pues requieren dos operandos para poder funcionar. Un requisito para que los operadores relacionales puedan funcionar es que ambos operandos tengan asignados valores del mismo tipo de datos.

Tabla 3.1 Operadores relacionales de Raptor

Operador	Acción	Ejemplo	Significado	Tipo de relación
>	Mayor que	$A > B$	A es mayor que B	Orden
>=	Mayor que o igual que	$A \geq B$	A es mayor o igual que B	Orden
<	Menor que	$A < B$	A es menor que B	Orden
<=	Menor que o igual que	$A \leq B$	A es menor o igual que B	Orden
=	Igual	$A = B$	A es igual a B	Condición
!=	No igual	$A \neq B$	A no es igual a B	Condición

Los operadores lógicos permiten plantear expresiones de álgebra booleana que enlazan, separan o complementan a las expresiones relacionales (véase tabla 3.2). Los operadores lógicos se usan para soportar las operaciones de conjunción, disyunción inclusiva y negación, de acuerdo con la siguiente tabla de verdad (véase tabla 3.3).

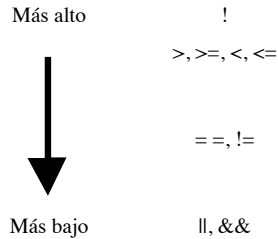
Tabla 3.2 Operadores lógicos de Raptor

Operador simbólico	Operador alfabético	Acción	Nombre del carácter
&&	<i>And</i>	Conjunción	Ampers <i>And</i>
	<i>Or</i>	Disyunción inclusiva	Barra vertical
!	<i>Not</i>	Negación	Signo final de exclamación

Tabla 3.3 Verdad de los operadores lógicos

x	y	$X \text{ and } y$	$X \text{ or } y$	$\text{not } x$
Falso	Falso	Falso	Falso	Verdadero
Falso	Verdadero	Falso	Verdadero	Verdadero
Verdadero	Verdadero	Verdadero	Verdadero	Falso
Verdadero	Falso	Falso	Verdadero	Falso

Los operadores **relacionales** y **lógicos** tienen menor prioridad que los aritméticos, y la relativa es la siguiente:



3.3 Símbolo Selection

Raptor representa gráficamente la estructura selectiva con un rombo con dos flechas divergentes que marcan rutas, caminos u opciones mutuamente excluyentes: Yes y No; independientemente cuál camino se tome, ambos vuelven a unirse (véase figura 3.1). En medio del rombo se escribe una expresión matemática condicional, que puede evaluarse como verdadera o falsa. Cuando la condición se evalúa como verdadera se toma como decisión la de seguir por el camino de Yes; por el contrario, cuando la condición se evalúa como falsa, se toma el camino de No.

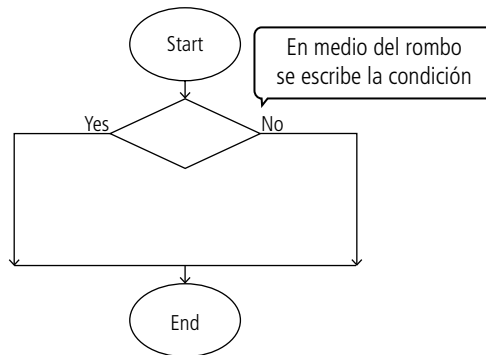


Figura 3.1 Símbolo Selection.

Por ejemplo, se calcula el promedio de calificación y a partir de ahí se determina si el alumno está aprobado o reprobado. Entonces, la condición es que si el promedio es mayor que o igual que 70, el alumno está aprobado; pero si el promedio es menor que 70, el alumno está reprobado. Pensando que la variable que contiene el promedio de la calificación se llama **promedio**, el símbolo Selection quedaría de acuerdo con la figura 3.2. Observe que el rombo de Selection se expandió y tomó una forma hexagonaloide.

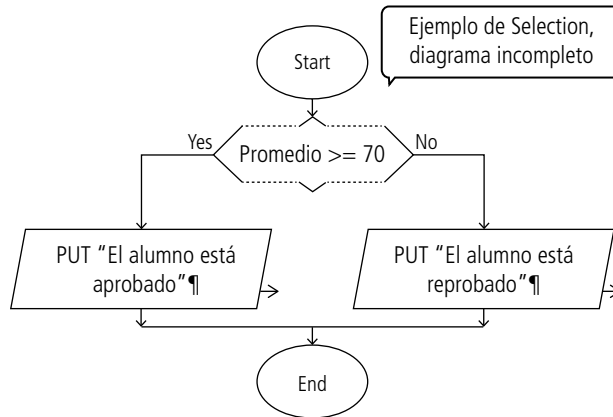


Figura 3.2 Ejemplo de condición de Selection. Diagrama incompleto.

El ejemplo de la figura 3.2 es un caso clásico de una estructura selectiva doble compuesta, ya que tiene por lo menos una acción tanto por la parte verdadera como por la falsa. Cuando la estructura selectiva sólo tiene acciones por uno de los dos caminos, se dice que ésta se implementó de manera simple. Este diagrama está incompleto para mostrar de manera específica el símbolo Selection; si intenta ejecutar este diagrama se generará un error porque no está terminado.



3.4 Algoritmos con la estructura selectiva doble

En el capítulo anterior, mencionamos que el símbolo Input no puede verificar que los datos agregados a una variable sean valores válidos respecto al problema que se está tratando de resolver. Esto tiene como inconveniente que la mayoría de los algoritmos de ese capítulo generen errores de ejecución porque al agregar un dato no numérico a una variable que se utilizará en una fórmula matemática, Raptor no puede procesarlo y por consecuencia crea ese tipo de errores. Otro inconveniente es que pueden producir resultados inconsistentes con el contexto del problema, como es el caso de asignar un valor negativo a una variable que representa la masa de un objeto.

También mencionamos que se pueden hacer diagramas que se “defiendan” de usuarios, ya sea que por accidente o con intención agregan valores no válidos a las variables. Una forma de hacer esos diagramas es que se utilicen funciones que permitan verificar el tipo de dato agregado. **El Algoritmo3-1-selectiva** verifica si el valor agregado al identificador es o no numérico, la función `Is_Number()` de Raptor se ha utilizado con este propósito (véase figura 3.3).

A partir de este momento, al nombre de los algoritmos se les agregará el sufijo **-selectiva** para señalar que pertenecen al tema de diseño de algoritmos con la estructura selectiva.

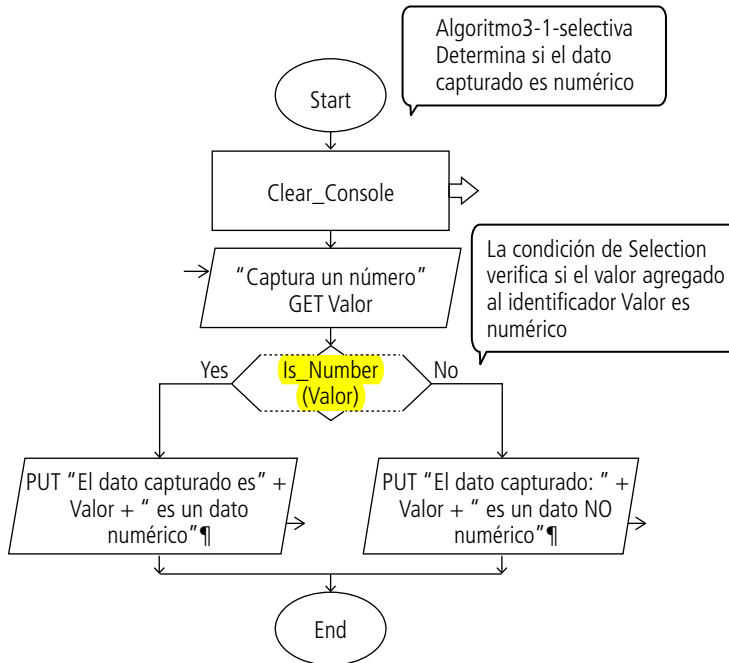


Figura 3.3 Algoritmo3-1-selectiva.

Hay ocasiones en las que no es suficiente con saber si el dato es o no numérico, sino que deseamos saber si cumple con una cierta condición. Con el **Algoritmo3-2-selectiva** podemos saber si el dato agregado es mayor que cero.

Aun cuando en el diagrama se pide al usuario que capture un valor numérico, la verdad es que no se puede hacer nada para obligarlo a obedecer, por lo que este algoritmo generará un error de ejecución si el dato capturado no es numérico. Esto se debe a que el símbolo Selection no puede comparar valores numéricos contra valores no numéricos. Si el usuario agrega un CERO, entonces el programa dará una información inconsistente y dirá que éste es un dato menor que cero; la respuesta correcta es que CERO es igual que CERO. Esta situación se resolverá más adelante.

Los algoritmos deben estar diseñados de tal forma que puedan tratar estos errores de captura para evitar los errores fatales de ejecución o los resultados o respuestas inconsistentes. Más adelante, en el tema 3.6: Anidamiento de la estructura selectiva, se perfeccionará este diagrama.

En este capítulo, con el propósito de no hacer más complejos los diagramas, sólo en contadas ocasiones se implementarán mecanismos de verificación de valores agregados a variables, por lo que partimos de que el usuario será cuidadoso al momento de agregar valores para las variables mediante el teclado (véase figura 3.4).

En el diagrama del **Algoritmo3-3-selectiva** se solicita al usuario que capture un número y se le informa si éste es positivo o negativo. La condición matemática

que se usa en este algoritmo es: “si el número es mayor que CERO”; cuando la condición se evalúa como verdadera, el resultado que se genera es que “el número es positivo” o, por el contrario, “el número es negativo” (véase figura 3.5).

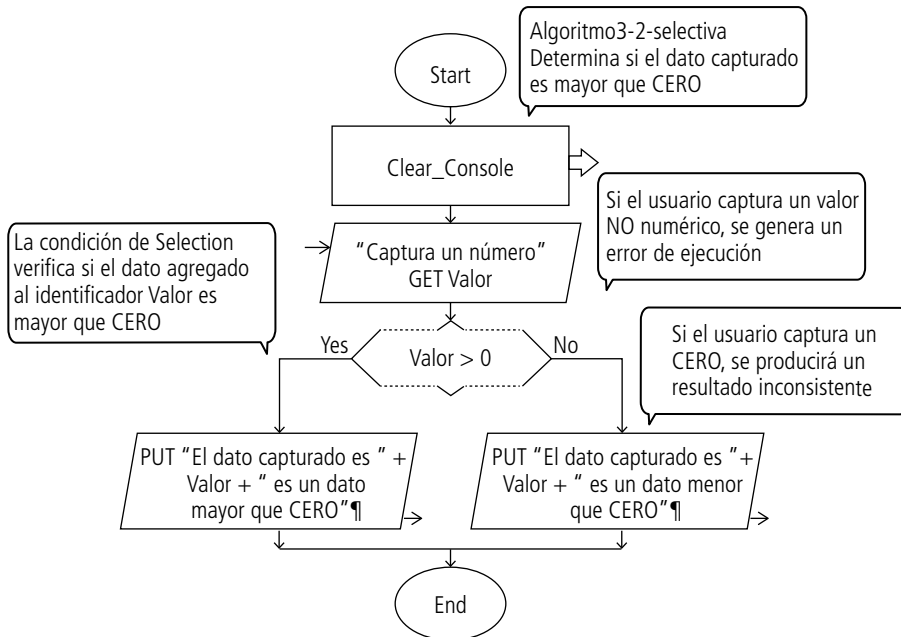


Figura 3.4 Algoritmo3-2-selectiva.

Si el usuario agrega un CERO, entonces el programa dará una información inconsistente y dirá que es negativo; la respuesta correcta es que el CERO es neutro. Esta situación se resolverá más adelante.

El diagrama del **Algoritmo3-4-selectiva** pide al usuario que capture un número entero e informa si el número es par o impar. Sólo los números enteros pueden clasificarse como pares o impares, es decir, aquellos números que no tienen decimales. Por el contrario, los números decimales, es decir, aquellos que tienen fracciones, no pueden entrar en la definición de la paridad de los números.

Para que el algoritmo pueda determinar si el número es par o impar, el dato capturado será sometido a una división entera entre 2 para obtener el residuo, partiendo de la base de que el residuo de todo número entero par dividido entre 2 es CERO. Por el contrario, para todo número entero impar dividido entre 2, el residuo que se genera es 1. Por tanto, la condición utilizada es: “si el residuo es igual a CERO”, la condición se evalúa como verdadera, entonces el número es par; pero si sucede lo contrario es impar.

Existe una discusión filosófica sobre si el CERO es par o no lo es. Aquí no tomaremos partido en relación con esta situación. En todo caso, si el número insertado

fuera CERO, entonces, el programa dará una información aceptable y dirá que éste es par y es aceptable toda vez que el CERO cumple con los requisitos para ser clasificado como tal. Para mayor información es conveniente revisar la teoría de números.

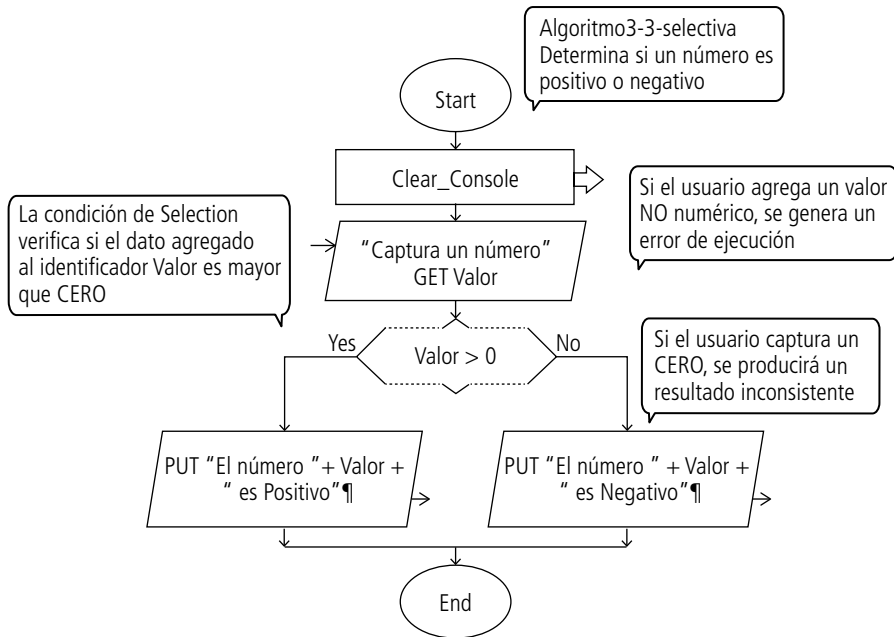


Figura 3.5 Algoritmo3-3-selectiva.

Si el número capturado fuera un número no entero, entonces el programa dará una información inconsistente y dirá que el número es impar. El símbolo de Input no puede verificar si un número es entero o no. Teniendo esto como referencia, el usuario debe ser cuidadoso al momento de agregar valores en este algoritmo.

Por otro lado, si se agrega un valor que no sea estrictamente numérico, se provocará un error de ejecución que será reportado por el Assignment que intenta utilizar ese valor para dividirlo entre 2 (véase figura 3.6).

En el diagrama del **Algoritmo3-5-selectiva** se pide al usuario que agregue dos números y se informa si son iguales o diferentes. Para llegar a la conclusión, se utiliza la condición que permite compararlos. Si la condición se evalúa como verdadera la respuesta es que son iguales, y por el contrario si la condición es falsa la respuesta es que son diferentes (véase figura 3.7).

El diagrama del **Algoritmo3-6-selectiva** permite saber si un alumno aprobó o no una materia, si se tiene como datos conocidos las calificaciones de dos exámenes parciales (véase figura 3.8).

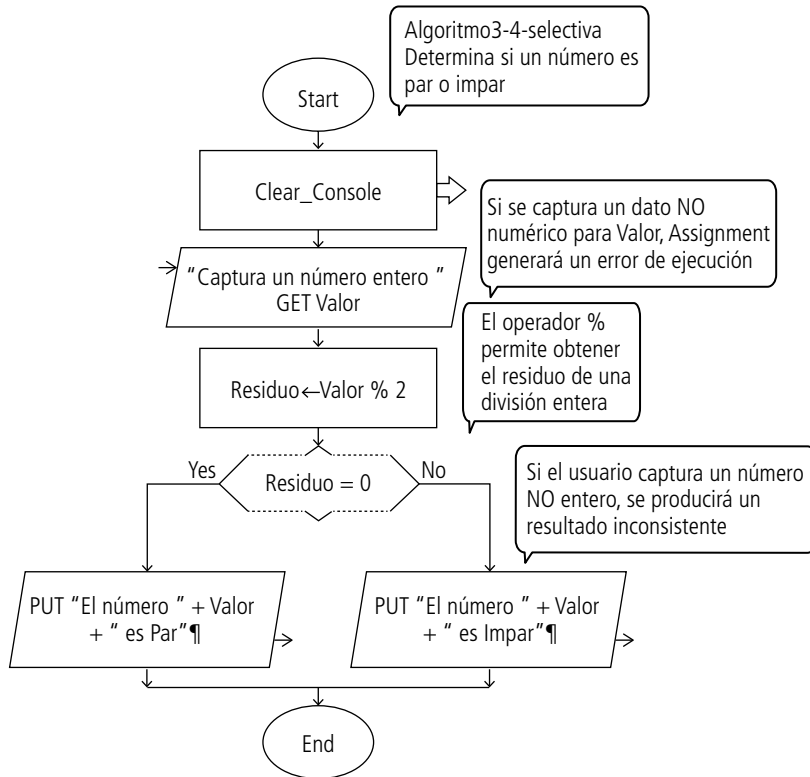


Figura 3.6 Algoritmo3-4-selectiva.

Con el diagrama del **Algoritmo3-7-selectiva** se resuelve el mismo problema anterior, pero se hacen algunas modificaciones. En el símbolo Selection se utiliza la variable **Situacion** para almacenar el estado del alumno. Dependiendo de la evaluación de la condición, tomará uno de dos valores: Aprobado o Reprobado. Observe que al momento de asignarle valor a la variable **Situacion**, éste se escribe entre comillas altas, por lo que se dice que esta variable es alfabética. Posteriormente, el resultado final del diagrama difiere si la condición se evalúa como verdadera o falsa (véase figura 3.9).

Es conveniente resaltar que los diagramas del **Algoritmo3-6-selectiva** y **Algoritmo3-7-selectiva** arrojarán resultados inconsistentes si alguna o ambas calificaciones están fuera del rango de CERO a CIEN, por lo que el usuario debe ser cuidadoso al momento de agregar estos valores. Para verificar si un valor capturado se encuentra dentro de un rango numérico específico, se requiere estudiar el uso de los operadores lógicos y el tema de verificación de los valores que toman las variables, situación que se aborda más adelante.

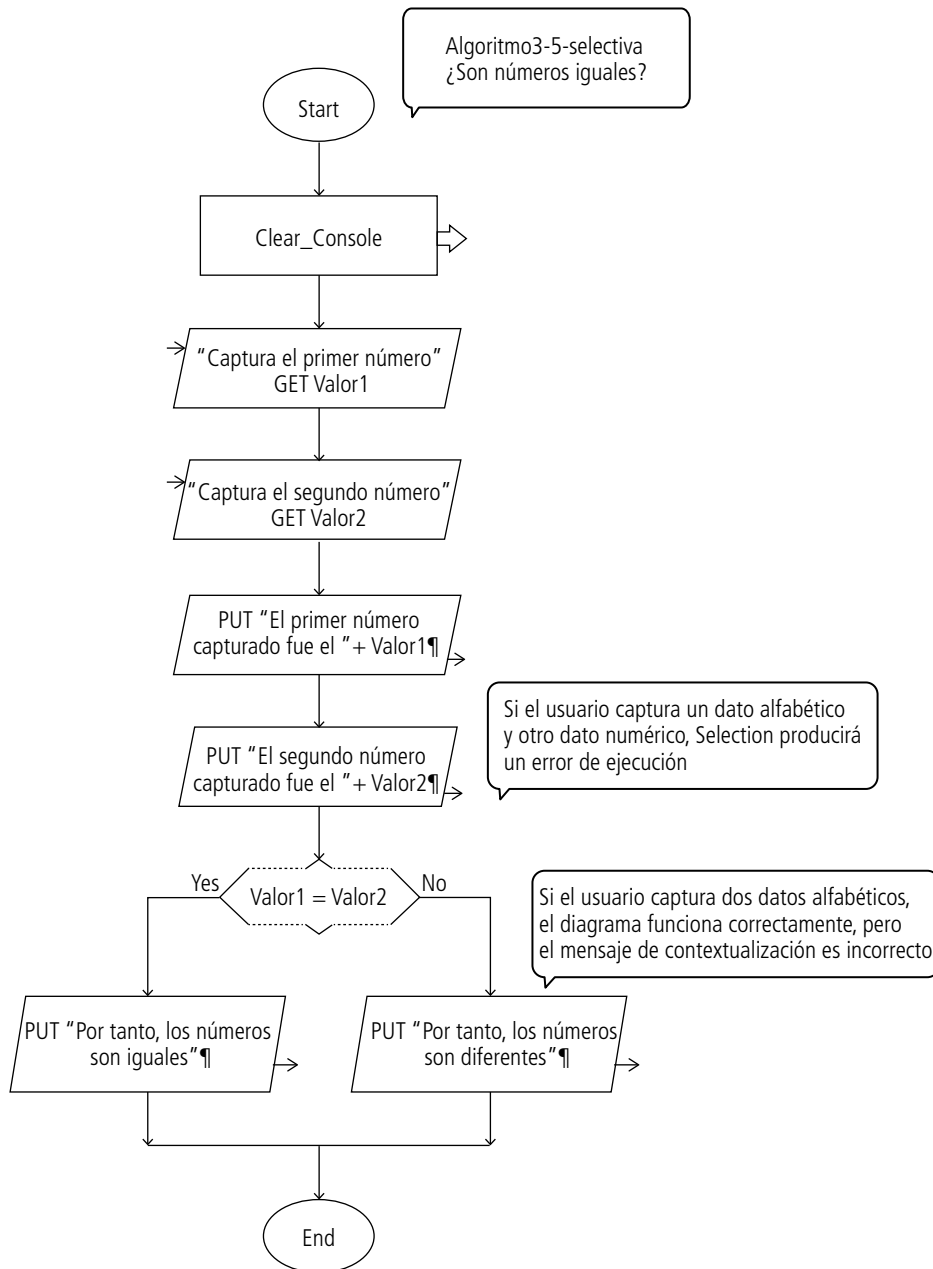


Figura 3.7 Algoritmo3-5-selectiva.

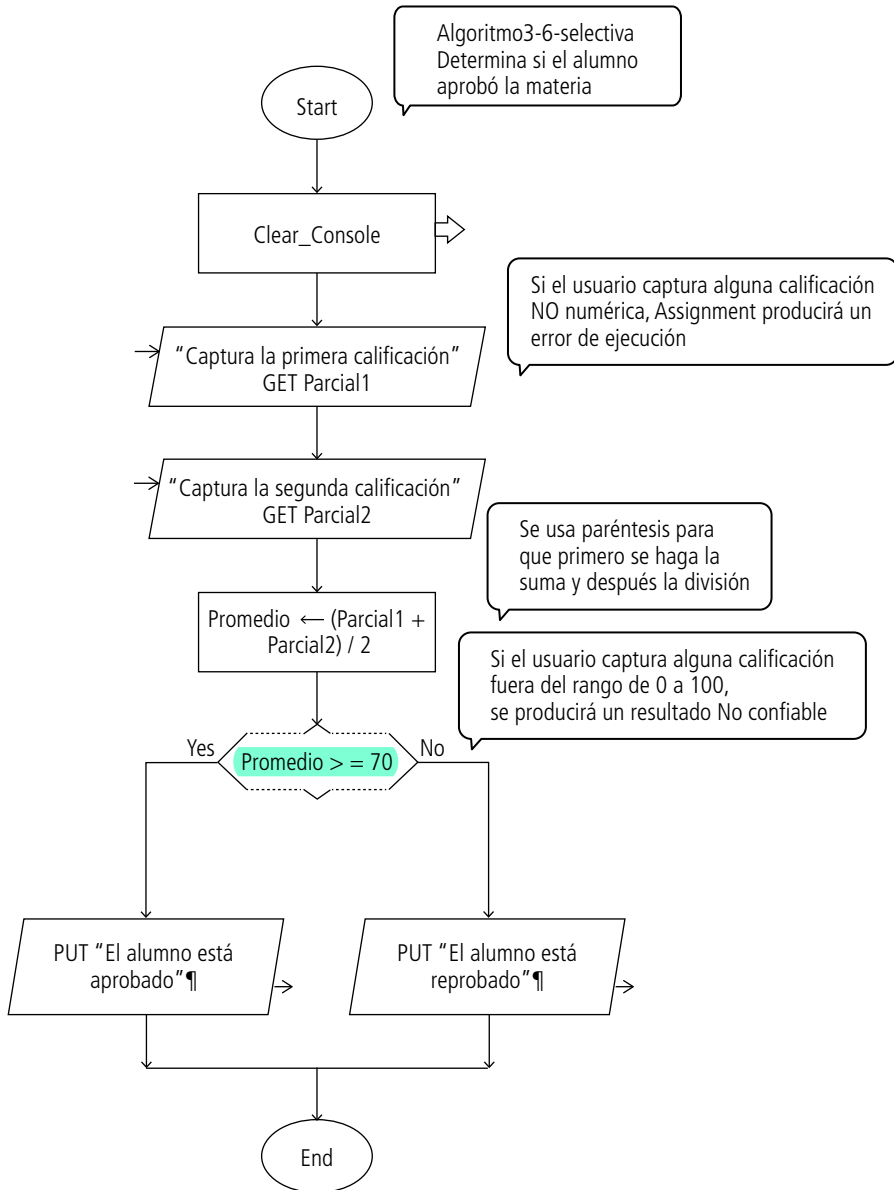


Figura 3.8 Algoritmo3-6-selectiva.

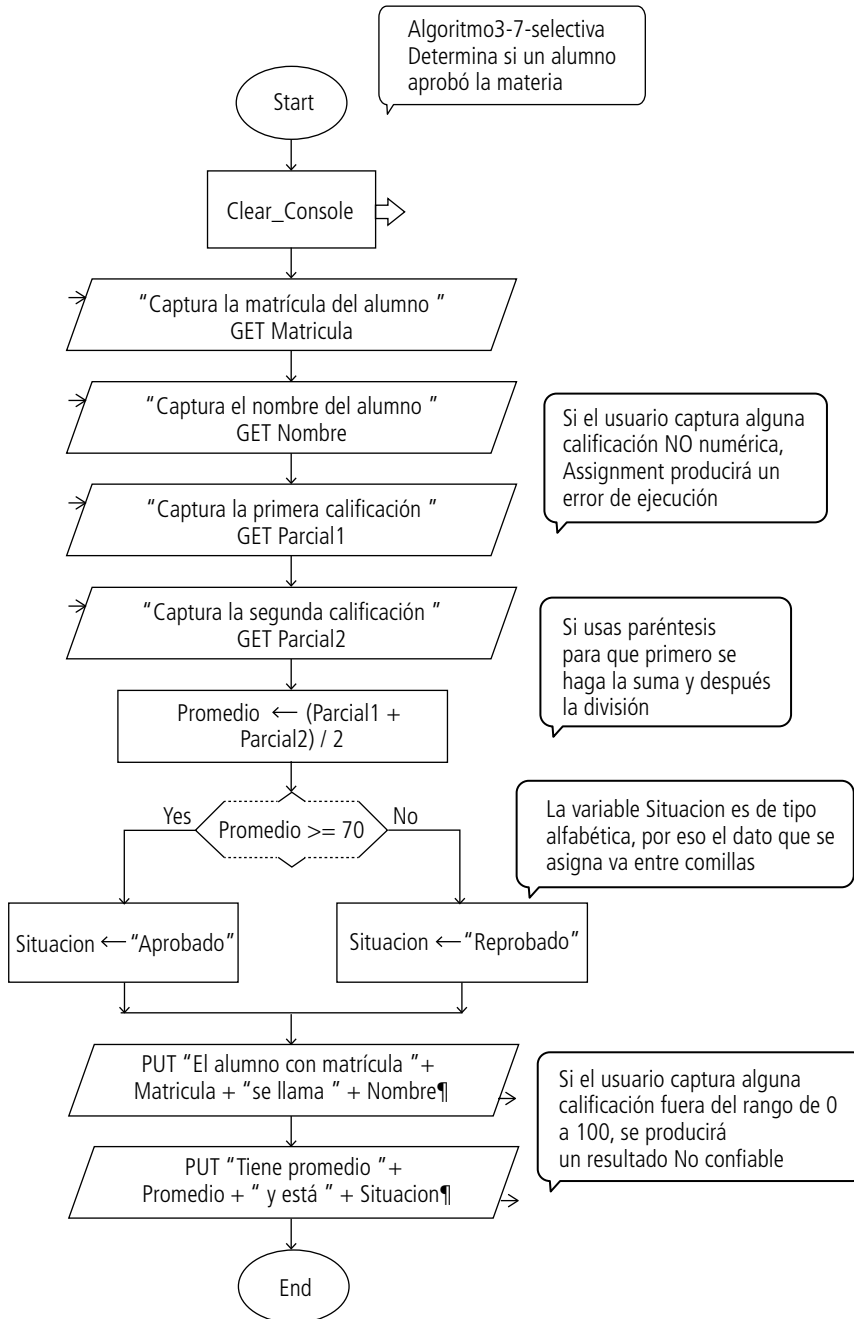


Figura 3.9 Algoritmo3-7-selectiva.



3.5 Deficiencias en los algoritmos

La mayor parte de los diagramas realizados en el capítulo 2 Diseño básico de algoritmos presentan algunas deficiencias, las cuales en la medida que se avance en este libro serán corregidas. La mayor parte de éstas se deben a la posible captura de datos que arrojan resultados fuera de contexto o inconsistentes con la realidad. Por ejemplo:

1. En el diagrama del **Algoritmo2-8-básico** que calcula la fuerza, si se tiene como datos conocidos la masa y aceleración. Si se agrega un valor negativo a la masa, el programa arrojará un resultado inconsistente.
2. En el diagrama del **Algoritmo2-9-básico** que hace la conversión de pies a metros, si se agregan números negativos el resultado que se obtendrá será inconsistente.
3. En el diagrama del **Algoritmo2-10-básico** que calcula la velocidad final de un objeto en caída libre cuando éste llega al suelo, si se agrega una altura negativa se produce un error de ejecución porque la función que calcula la raíz cuadrada: `sqrt()`, no puede generar un resultado si el radicando es negativo.
4. En el diagrama del **Algoritmo2-11-básico** que calcula el área de un cuadrado, al agregar un número negativo el resultado sería inconsistente.
5. En el diagrama del **Algoritmo2-12-básico** que calcula el área de un círculo, si se agrega un diámetro negativo el programa arrojará un resultado inconsistente.
6. En el diagrama del **Algoritmo2-13-básico** que calcula el área de un triángulo, si se agrega un valor negativo (ya sea para la base, la altura o para ambos), el resultado es inconsistente.
7. En el diagrama del **Algoritmo2-15-básico** que calcula el área de un triángulo, pero con una fórmula que involucra la longitud de los tres lados; si se insertan datos que no forman un triángulo el programa generará un error de ejecución, pues la función que calcula la raíz cuadrada: `sqrt()`, no puede generar un resultado si el radicando es negativo.
8. En el diagrama del **Algoritmo2-16-básico** que calcula la fuerza perpendicular en el ala de un avión se presentan problemas similares, pues si se agrega un área negativa del ala del avión los resultados serán inconsistentes.

En algunos de estos algoritmos, el programador debe decidir qué hacer cuando se agreguen valores de CERO, pues, por ejemplo, no se puede calcular el área de un cuadrado con `lado=0` porque no existe un cuadrado con esas dimensiones.

Por otra parte, es necesario que el programador sepa diferenciar entre lo que significa, por un lado, que un programa arroje resultados inconsistentes y, por otro, que genere un error de ejecución, ya que son situaciones completamente diferentes. Un resultado inconsistente es un error de lógica común, tomando como referencia ciencias como la física o las matemáticas. Pero, por ejemplo, cuando se trata de obtener una raíz cuadrada a partir de un radicando negativo, el programa produce un error de ejecución.

También hay que diferenciar entre errores de lógica común y lógica computacional. Por ejemplo, si un programa me dice que un cuadrado cuyo lado mide cero metros lineales y su área es cero metros cuadrados, entonces puede ser que el programa tenga lógica computacional, pero carezca de lógica común porque no puede existir un cuadrado con esas dimensiones.

En el apartado que continúa hemos retomado los ejemplos del tema de diseño básico de algoritmos y les hemos dado un tratamiento para verificar que los datos de entrada únicamente toman valores de acuerdo con su contexto. El objetivo es evitar que los diagramas generen resultados inconsistentes, derivados de datos de entrada inadecuados.

En el diagrama del **Algoritmo3-8-selectiva** se solicita al usuario que capture el valor de la masa en kilogramos, el cual debe ser mayor que CERO. Si el valor de la masa toma valores mayores a cero, entonces se permite capturar el valor de la aceleración. Posteriormente, se calcula la fuerza multiplicando los valores capturados y el programa muestra el resultado. En contraparte, si el valor de la masa es cero o menor se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno (véase figura 3.10).

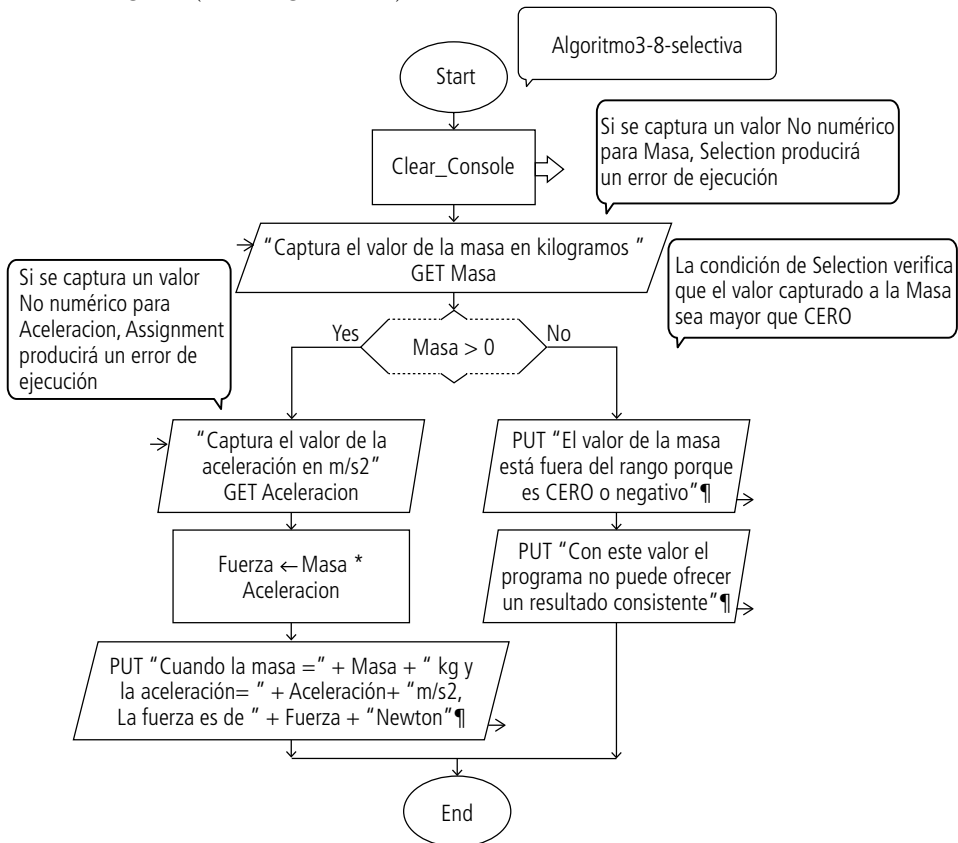


Figura 3.10 Algoritmo3-8-selectiva.

En el diagrama del **Algoritmo3-9-selectiva** se solicita al usuario que capture la cantidad en pies que desea convertir en metros lineales. Si la cantidad en pies es mayor de cero, se permite hacer la conversión y se muestra el resultado del diagrama. En contraparte, si la cantidad en pies es cero o menor, se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno (véase figura 3.11).

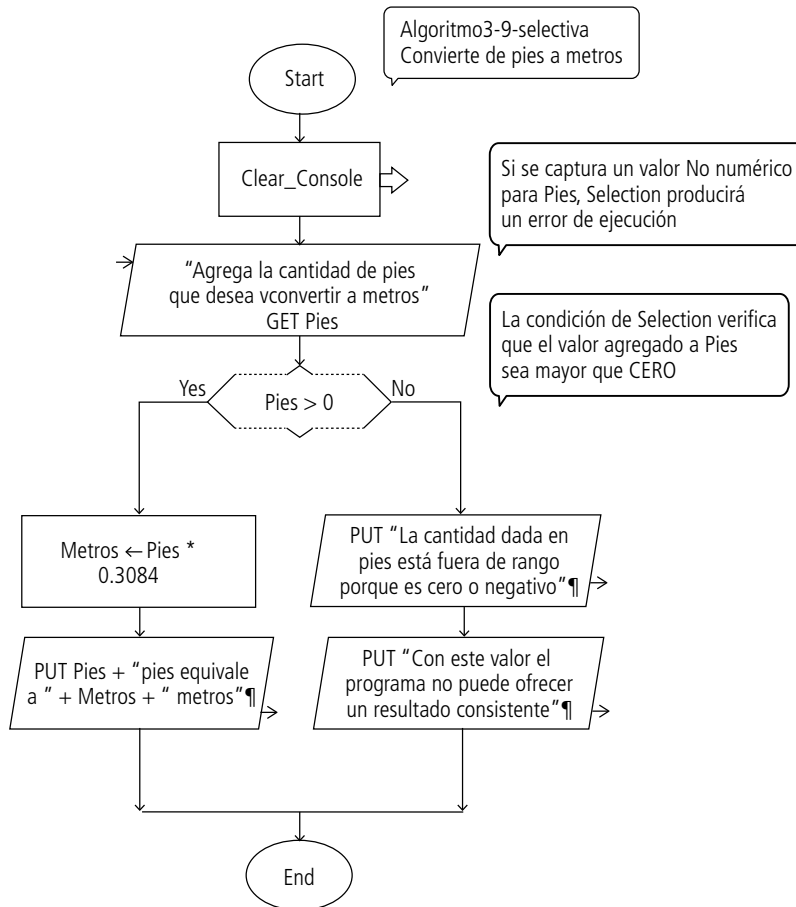


Figura 3.11 Algoritmo3-9-selectiva.

En el diagrama del **Algoritmo3-10-selectiva** se solicita al usuario que agregue la altura en metros desde donde se deja caer un objeto, la cual debe ser mayor que cero. Si la altura es mayor que cero, se permite calcular la velocidad final del objeto al momento del impacto en el suelo y se muestra el resultado del diagrama. En contraparte, si la altura es cero o menor, se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno (véase figura 3.12).

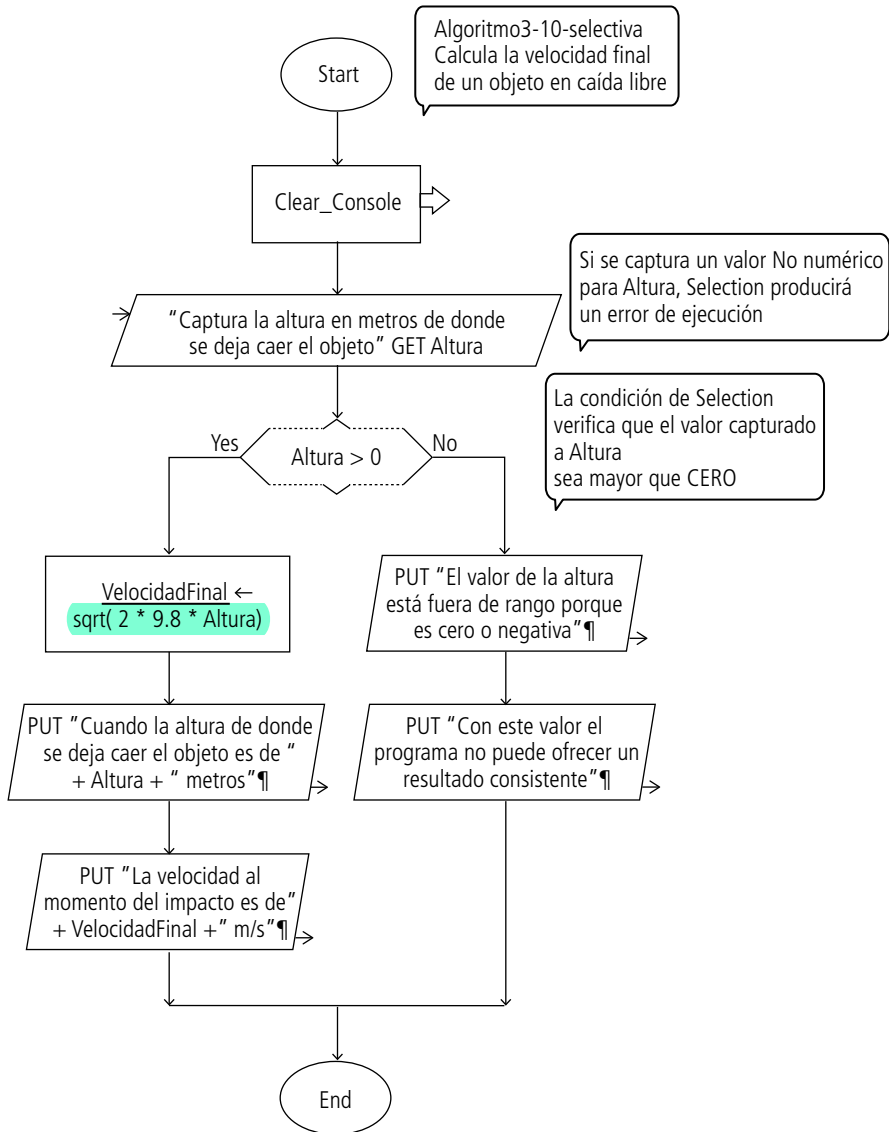


Figura 3.12 Algoritmo3-10-selectiva.

En el diagrama del **Algoritmo3-11-selectiva** se solicita al usuario que capture la longitud del lado del cuadrado en metros. Si la longitud del lado es mayor que cero, se permite calcular el área del cuadrado y se muestra el resultado del diagrama. En contraparte, si la longitud del lado es cero o menor, se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno (véase figura 3.13).

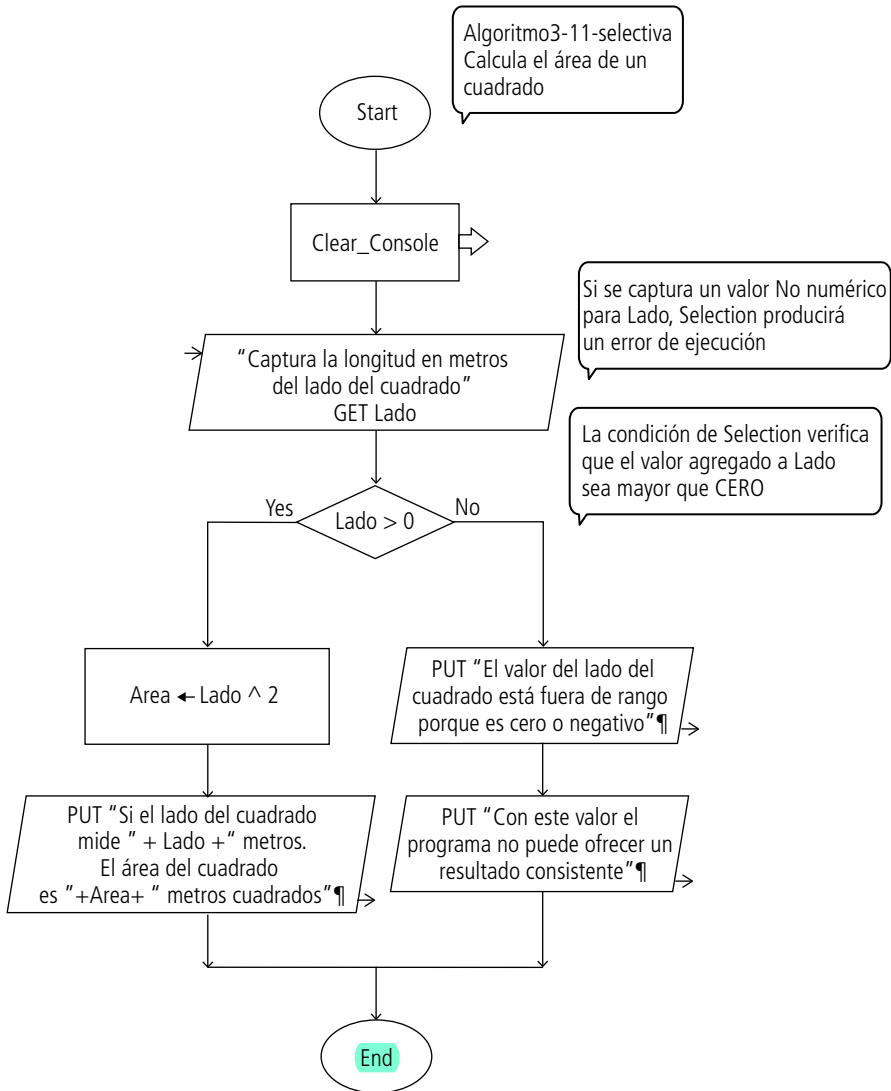


Figura 3.13 Algoritmo3-11-selectiva.

En el diagrama del **Algoritmo3-12-selectiva** se solicita al usuario que agregue la longitud en metros del diámetro del círculo. Si la longitud del diámetro es mayor que cero, se permite calcular el radio y el área del círculo, y se muestra el resultado del diagrama. En contraparte, si la longitud del diámetro es cero o menor, se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno (véase figura 3.14).

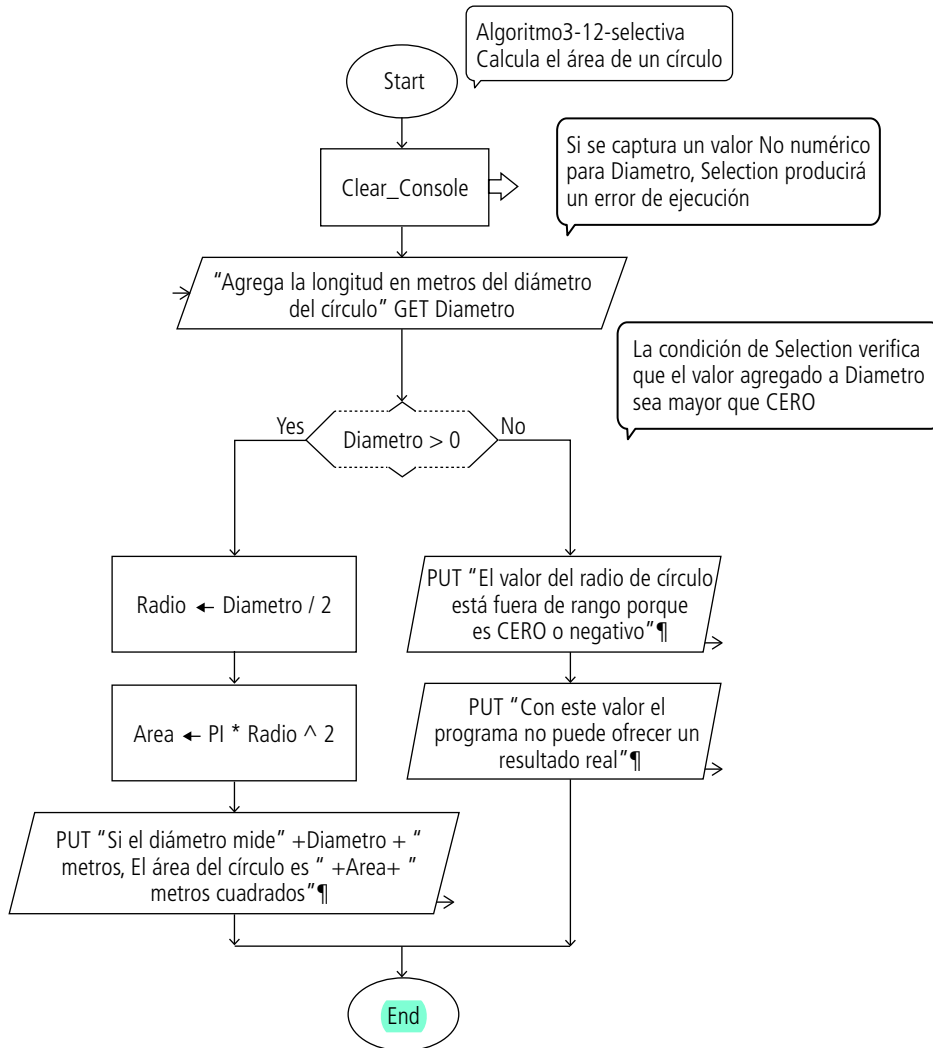


Figura 3.14 Algoritmo3-12-selectiva.

En el diagrama del **Algoritmo3-13-selectiva** se solicita al usuario la captura de la longitud en metros de la base y la altura del triángulo. Si ambos datos son mayores que cero, se permite calcular el área del triángulo y se muestra el resultado del diagrama. En contraparte, si uno o ambos datos son iguales a cero o menores, entonces se envía un mensaje de advertencia al usuario y el diagrama termina sin dar resultado alguno.

Observe que en este diagrama la condición de Selection utiliza el operador lógico *And* para verificar el valor de ambos datos al mismo tiempo (véase figura 3.15).

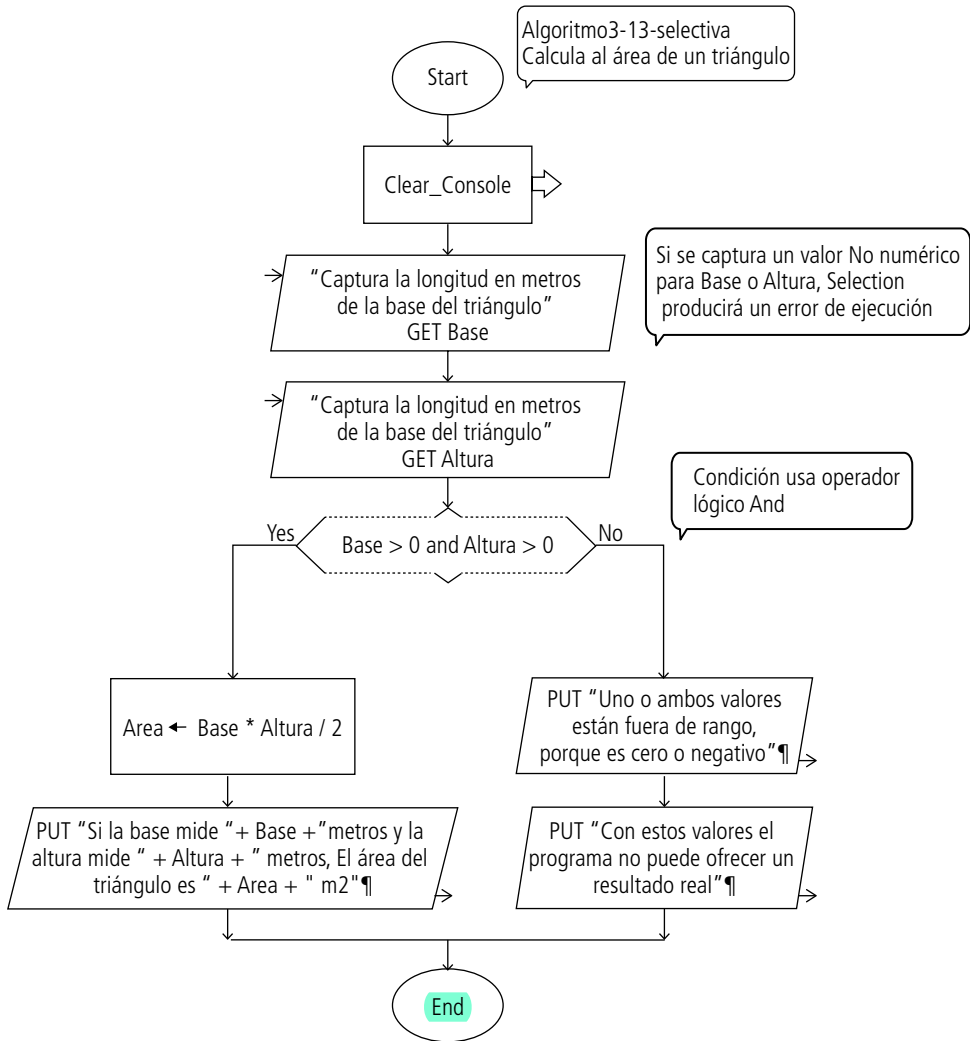


Figura 3.15 Algoritmo3-13-selectiva.

En el diagrama del **Algoritmo3-14-selectiva** se solicita al usuario que agregue dos números para determinar cuál de ellos es mayor. El programa funciona bien mientras ambos datos sean diferentes, porque si ambos números son iguales el programa arroja un resultado erróneo. Esto sucede debido a que el programa no está completo. Más adelante, se resuelve ese problema (véase figura 3.16).

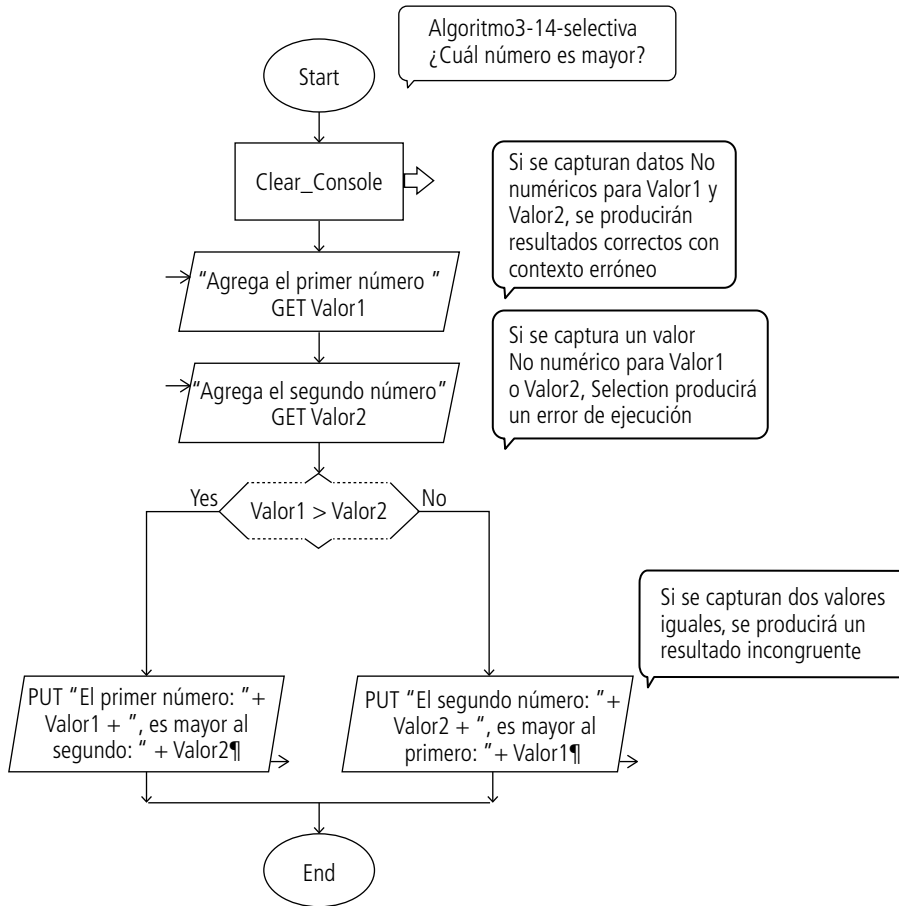


Figura 3.16 Algoritmo3-14-selectiva.

A continuación se plantea el siguiente problema.

Se desea calcular el pago del recibo de luz que una persona consume. Los datos de entrada son: número de medidor, cantidad de kilowatts consumidos, costo del kilowatt y saldo anterior. Si no se pagó el recibo anterior habrá un cargo extra de \$15. La manera de saber si el recibo anterior se pagó o no es preguntar si el saldo anterior es mayor que cero. Por tanto, el pago se calcula multiplicando la cantidad de kilowatts consumidos por el costo del kilowatt, y si el recibo anterior no fue pagado se agrega el cargo extra y el saldo anterior.

En el diagrama del **Algoritmo3-15-selectiva** se solicita al usuario que capture los datos de entrada y se calcula el pago sin recargos. Si el saldo anterior es mayor que cero, entonces al pago se le agregan los recargos; en caso contrario, el pago se deja como había sido calculado en un inicio. Independientemente del valor dado al saldo anterior, el programa termina mostrando el resultado (véase figura 3.17).

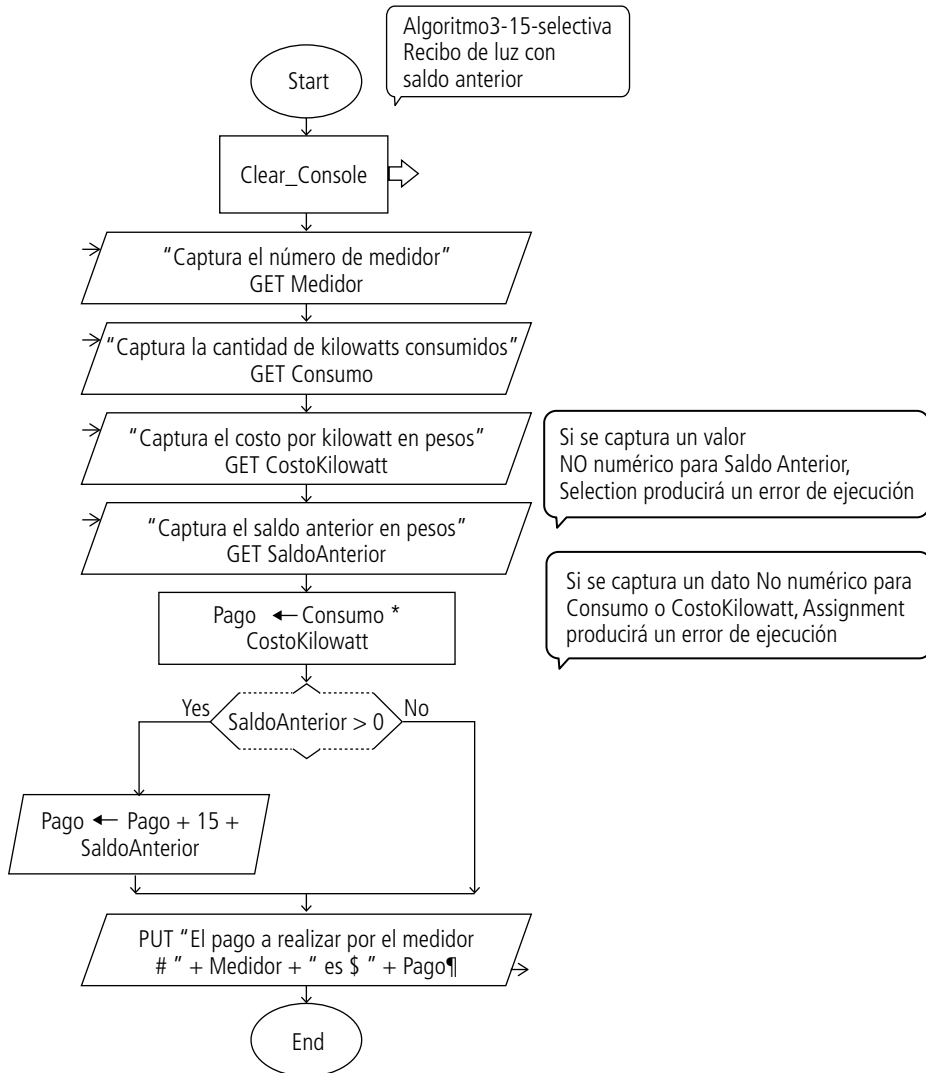


Figura 3.17 Algoritmo3-15-selectiva.

En el diagrama del **Algoritmo3-16-selectiva** se solicita al usuario que agregue la longitud en metros de los lados de un triángulo. Después, se calcula el radicando de la fórmula. Si el radicando es menor o igual que cero, el resultado es que los lados no pueden formar un triángulo. En caso contrario, sí pueden formar el triángulo (véase figura 3.18).

Con el diagrama del **Algoritmo3-17-selectiva** mostraremos cómo verificar si un valor numérico se encuentra dentro de un rango específico. En este caso, será la calificación de un examen que sólo puede estar entre cero y cien. Se requiere utilizar los operadores relacionales y lógicos. En este caso, se usa el operador lógico *And* (véase figura 3.19).

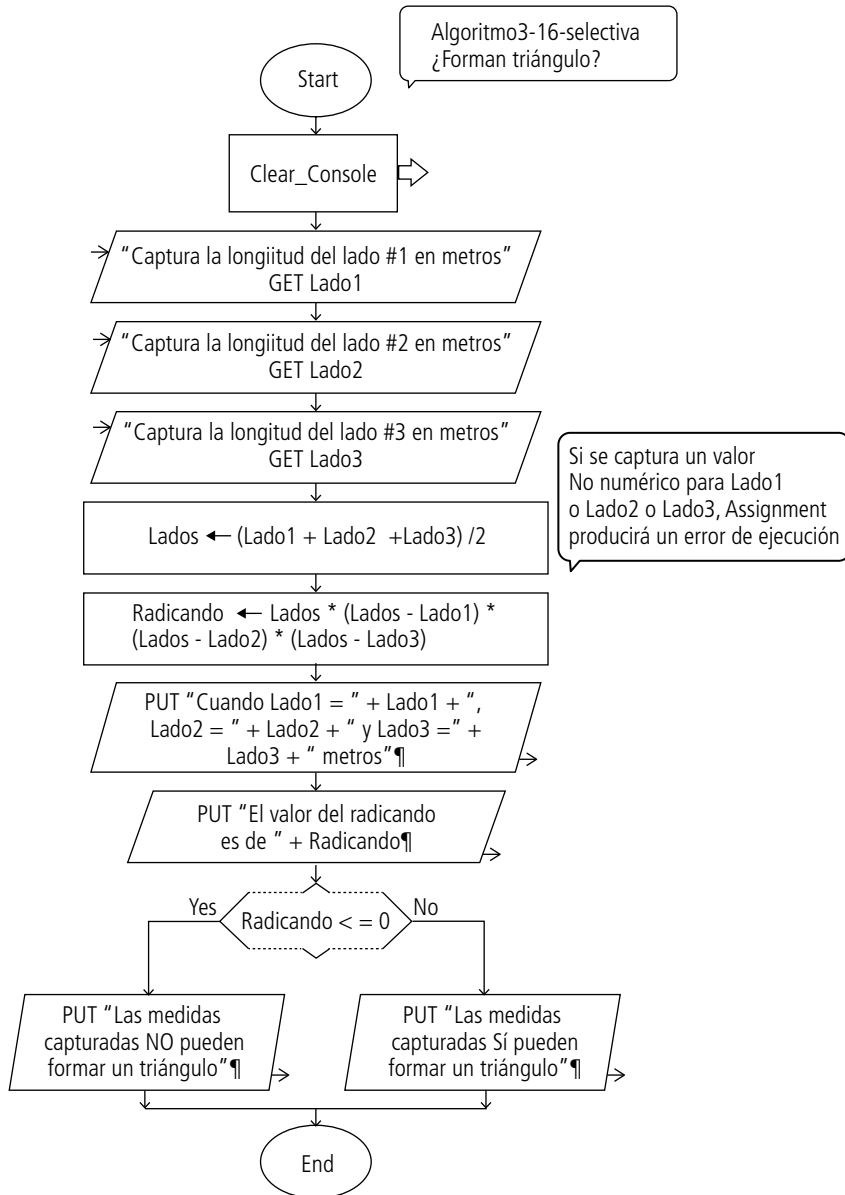


Figura 3.18 Algoritmo3-16-selectiva.

Con el diagrama del **Algoritmo3-18-selectiva** mostraremos el caso anterior, pero ahora se usará el operador lógico *Or* para que compare la operación complementaria para verificar un rango (véase figura 3.20).

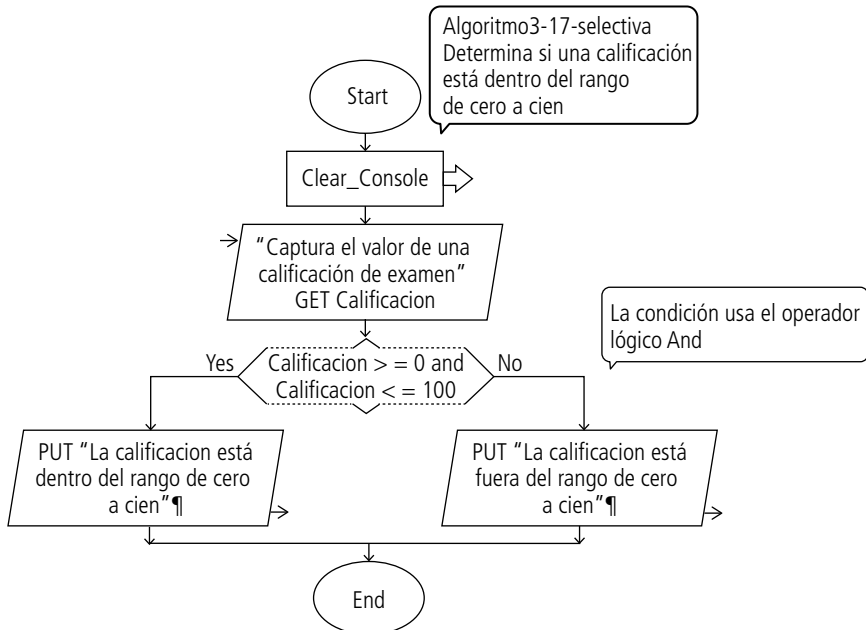


Figura 3.19 Algoritmo3-17-selectiva.

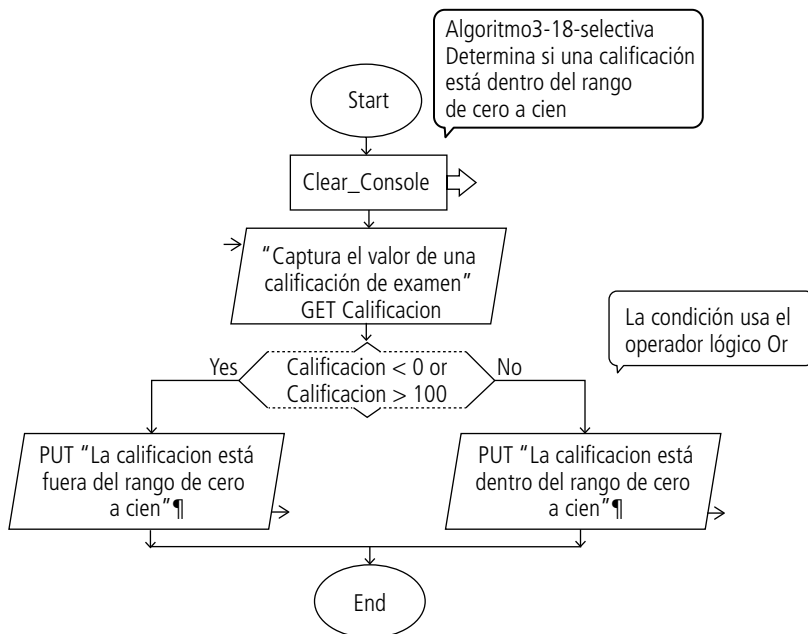


Figura 3.20 Algoritmo3-18-selectiva.



3.6 Anidamiento de la estructura selectiva

En los problemas en que existen opciones múltiples se puede recurrir al anidamiento de la estructura selectiva, esto es, una estructura selectiva dentro de otra estructura selectiva, ya sea por verdadero, falso o ambos.

A continuación presentamos algunos diagramas donde es necesario el uso de estructuras selectivas anidadas.

En el diagrama del **Algoritmo3-19-selectiva** se solicita al usuario que capture un número para determinar si es positivo o negativo. Primero comparamos si el dato capturado es igual a cero; dado que el cero está considerado como neutro, el resultado que se emite es que no se puede determinar si es positivo o negativo. Si el valor capturado es diferente a cero, entonces se hace una segunda pregunta. Si es mayor que cero, la respuesta es que el número es positivo; en caso contrario, es negativo (véase figura 3.21).

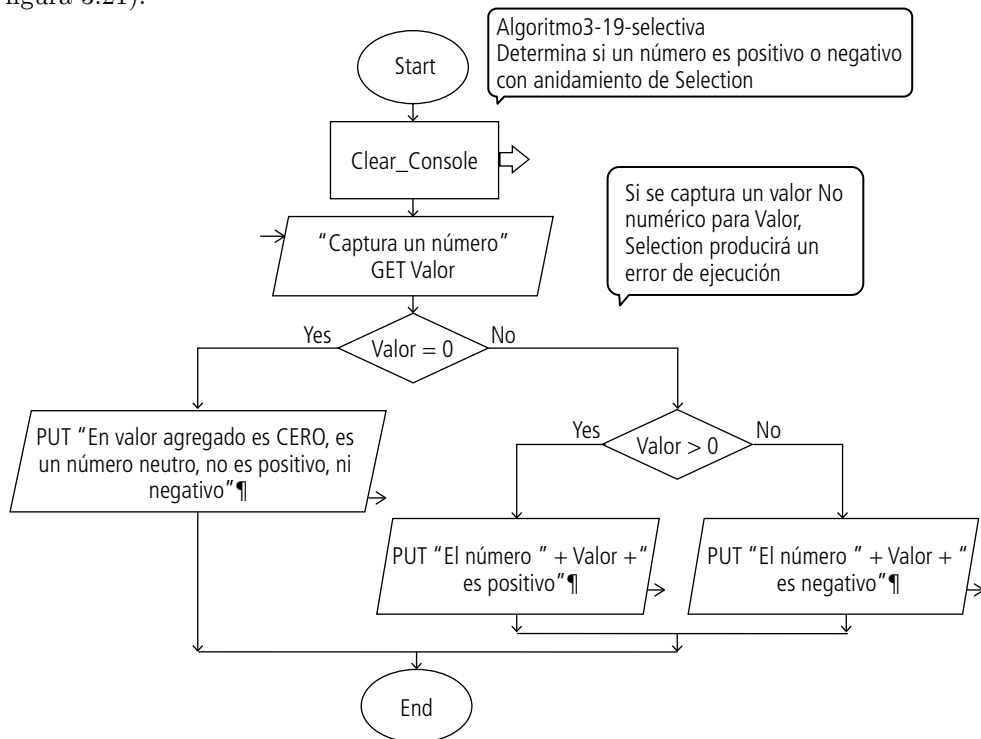


Figura 3.21 Algoritmo3-19-selectiva.

En el diagrama del **Algoritmo3-20-selectiva** se solicita al usuario que capture dos números. Primero comparamos si los números capturados son iguales entre sí, para dar como resultado que no se puede determinar cuál de los dos es mayor. En contraparte, cuando son diferentes entre sí, entonces se hace una segunda pregunta. Si el primero es mayor al segundo, en la respuesta se indica; en caso contrario, la respuesta es que el segundo número es mayor al primero (véase figura 3.22).

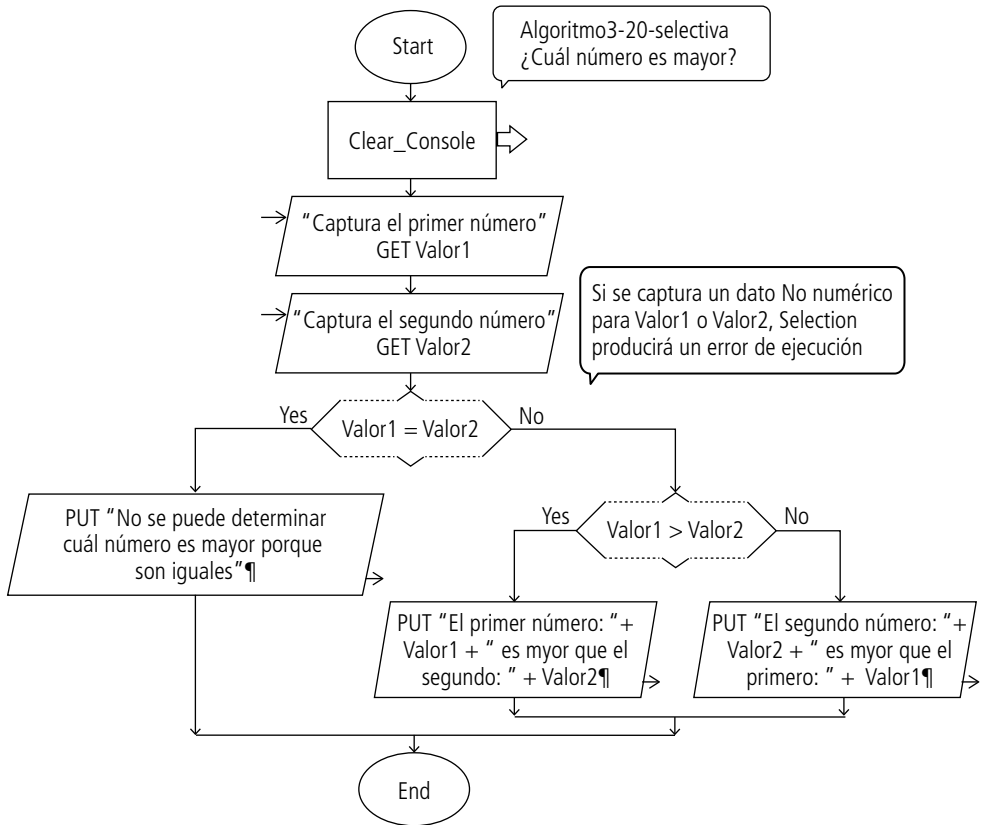


Figura 3.22 Algoritmo3-20-selectiva.

Con el diagrama del **Algoritmo3-21-selectiva** se plantea el siguiente problema. Conocemos los datos de una ecuación cuadrática, en este caso: el coeficiente cuadrático, el coeficiente lineal y el término independiente. Deseamos conocer el tipo de raíz que dichos datos arrojan (véase la figura 3.23).

La ecuación cuadrática suele representarse con el siguiente polinomio de segundo grado: $ax^2 + bx + c$ para $a \neq 0$. La obtención de las raíces se hace utilizando la siguiente fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Con el discriminante: $b^2 - 4ac$, se puede saber el tipo de raíces que tiene la ecuación cuadrática.

Cuando el discriminante es mayor que cero las raíces son reales y diferentes.

Cuando el discriminante es igual que cero las raíces son reales e iguales.

Cuando el discriminante es menor que cero las raíces son imaginarias.

El problema de saber si un número capturado es positivo o negativo todavía puede mejorarse si verificamos que el dato capturado sea realmente un valor numérico. Esto se muestra en el **Algoritmo3-22-selectiva**.

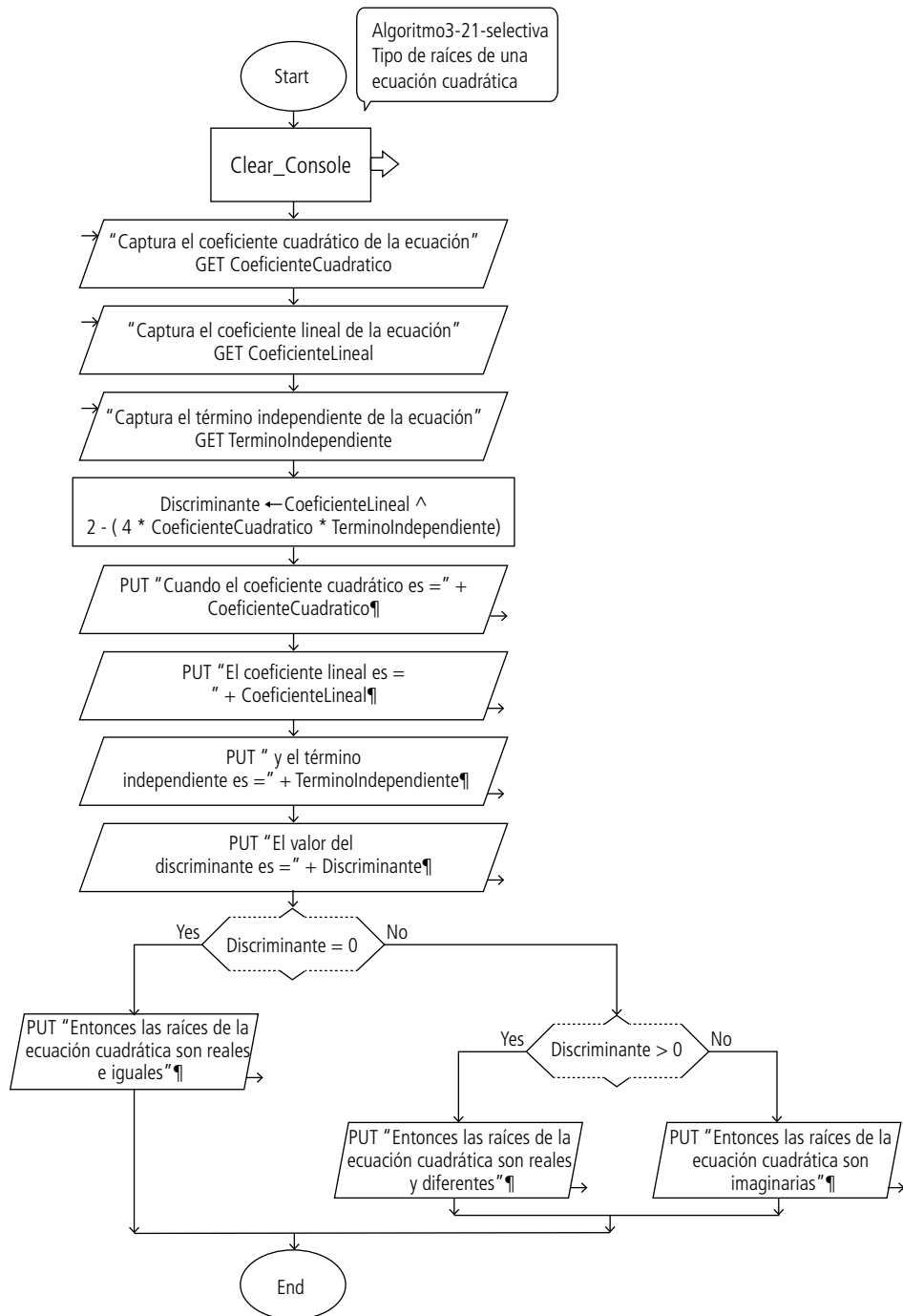


Figura 3.23 Algoritmo3-21-selectiva.

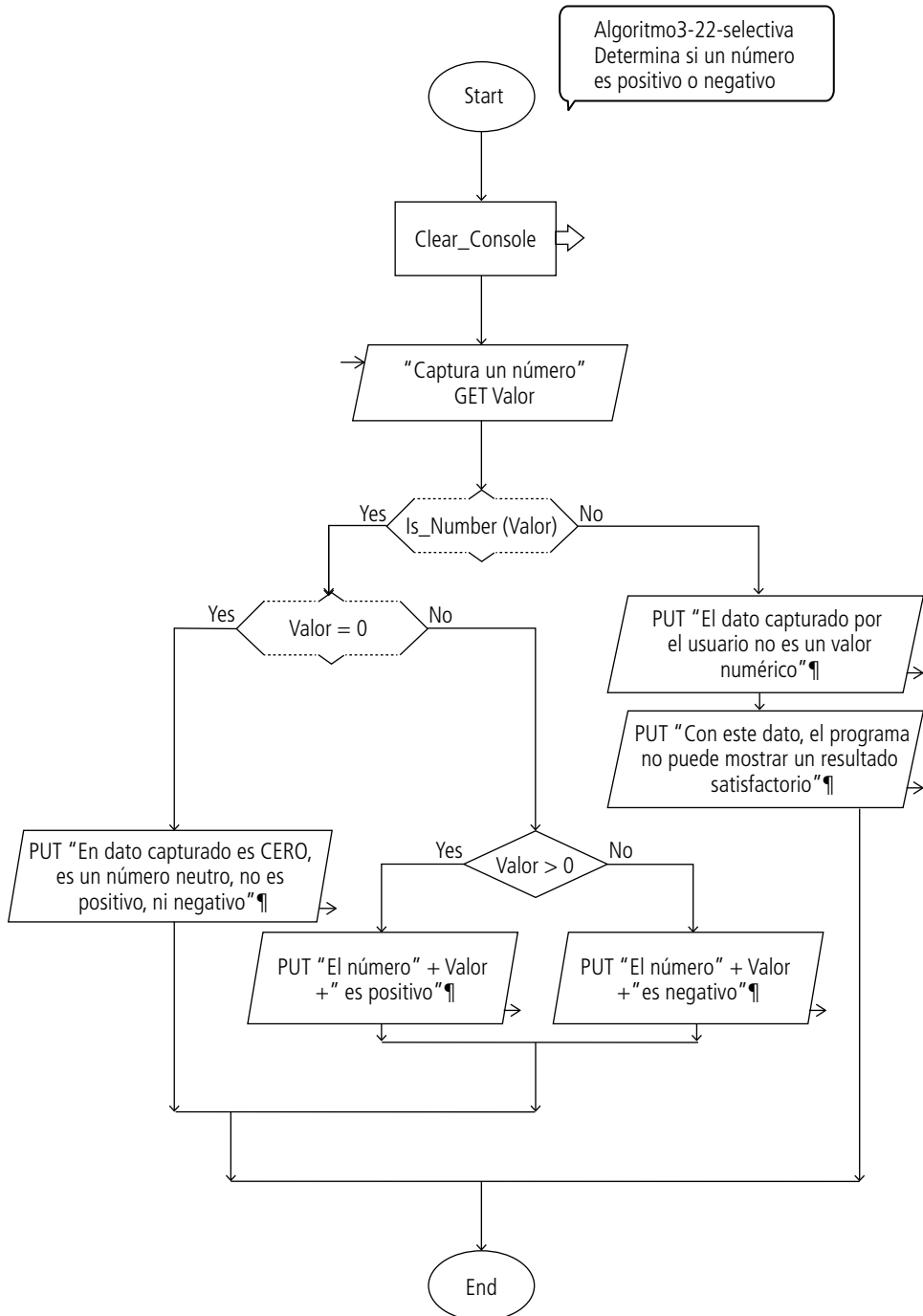


Figura 3.24 Algoritmo3-22-selectiva.

Con el ejemplo anterior (véase figura 3.24), se hace notar que a la mayoría de los diagramas en este capítulo podríamos agregarles la parte de verificación del tipo de dato para reducir los errores de ejecución. Sin embargo, los diagramas se volverían más complejos y largos, lo cual dificulta la exposición en un formato de libro porque los símbolos y sus letras quedarían muy pequeños y sería difícil su lectura. Por ello, se pide al lector que modifique los diagramas para que agregue esa verificación.

Por otro lado, tenemos el diagrama del **Algoritmo3-23-selectiva** que determina el grado del acero cuando se conocen los valores para T1 y T2. De esta manera, el acero se considera de grado 1, si T1 excede a 0.95 y T2 excede a 0.75; de grado 2, si T1 excede a 0.95 pero T2 no excede a 0.75; y de grado 3, si T1 no es mayor que 0.95.

Dado que los valores de T1 y T2 sólo pueden estar en el rango de cero a uno, cuando se insertan valores fuera de rango este programa no tiene un tratamiento alternativo ante dicha situación. El programa de cualquier forma arrojará un resultado, pero éste no es confiable. Esta situación lo único que muestra es que el programa no está debidamente concluido y requiere más lógica computacional para ser confiable (véase figura 3.25).

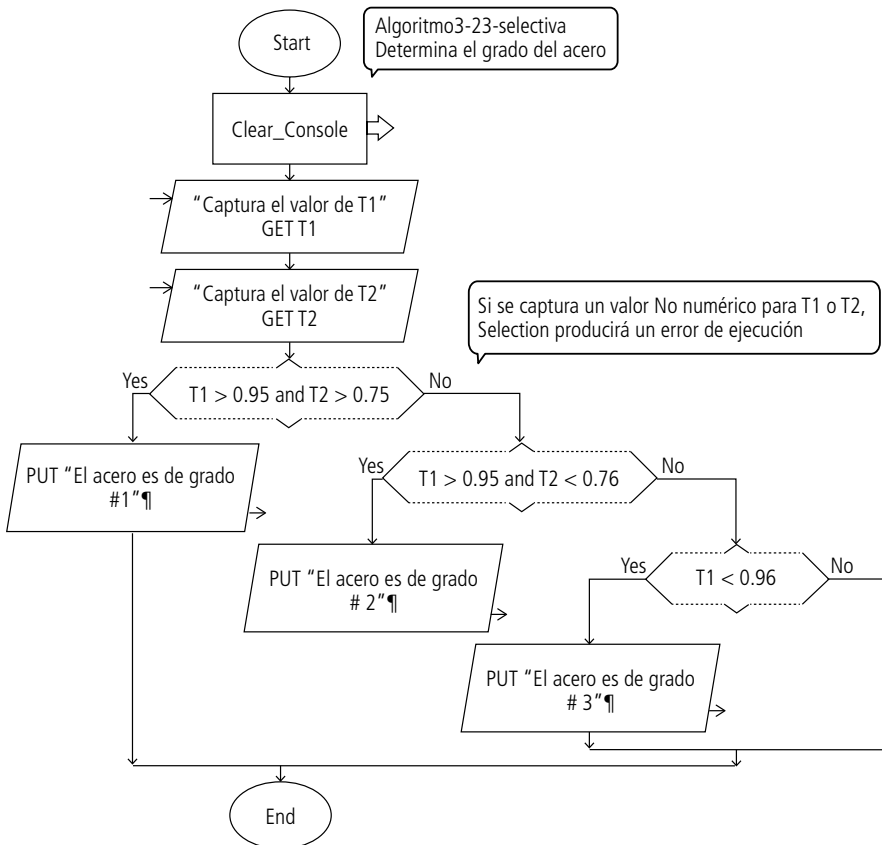


Figura 3.25 Algoritmo3-23-selectiva.

Enseguida tenemos el diagrama del **Algoritmo3-24-selectiva** que calcula los pagos de los recibos de luz, si se tiene como dato conocido la cantidad de kilowatts-hora consumidos con base en las siguientes tarifas (véase figura 3.26).

Tabla 3.4 Tarifas

Concepto	Kilowatt-hora	Precio
Básico	90 o menos	0.793
Intermedio	Siguientes 120	0.956
Excedente	Siguientes 235	2.802

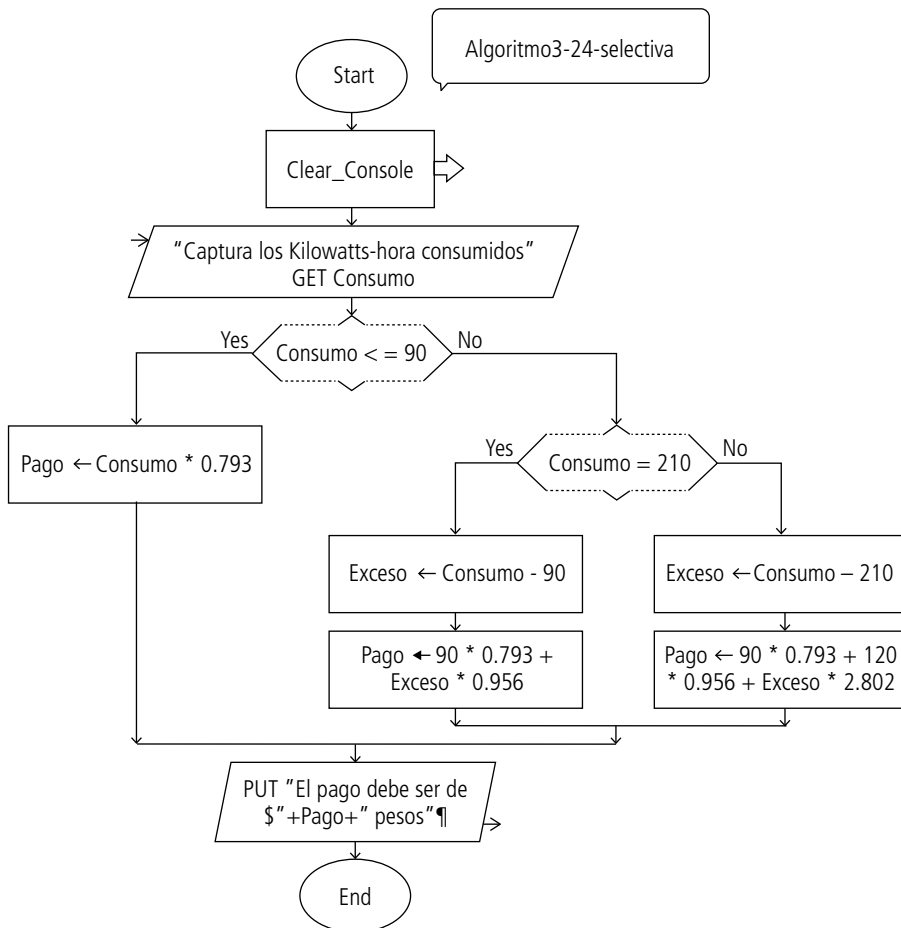


Figura 3.26 Algoritmo3-24-selectiva.

Por otro lado, los triángulos se clasifican en equiláteros, isósceles y escalenos, según la medida de las longitudes de sus lados. Sólo pueden tener cero, dos o tres lados iguales. Cualquier otro valor está fuera de rango.

En el diagrama del **Algoritmo3-25-selectiva** se solicita al usuario que capture la cantidad de lados iguales que tiene el triángulo. Si la cantidad de lados iguales es cero, entonces es escaleno. Si es dos, entonces es isósceles. Si es tres, entonces es equilátero. Si se inserta cualquier otro valor, el programa informa que el dato capturado está fuera de rango (véase figura 3.27).

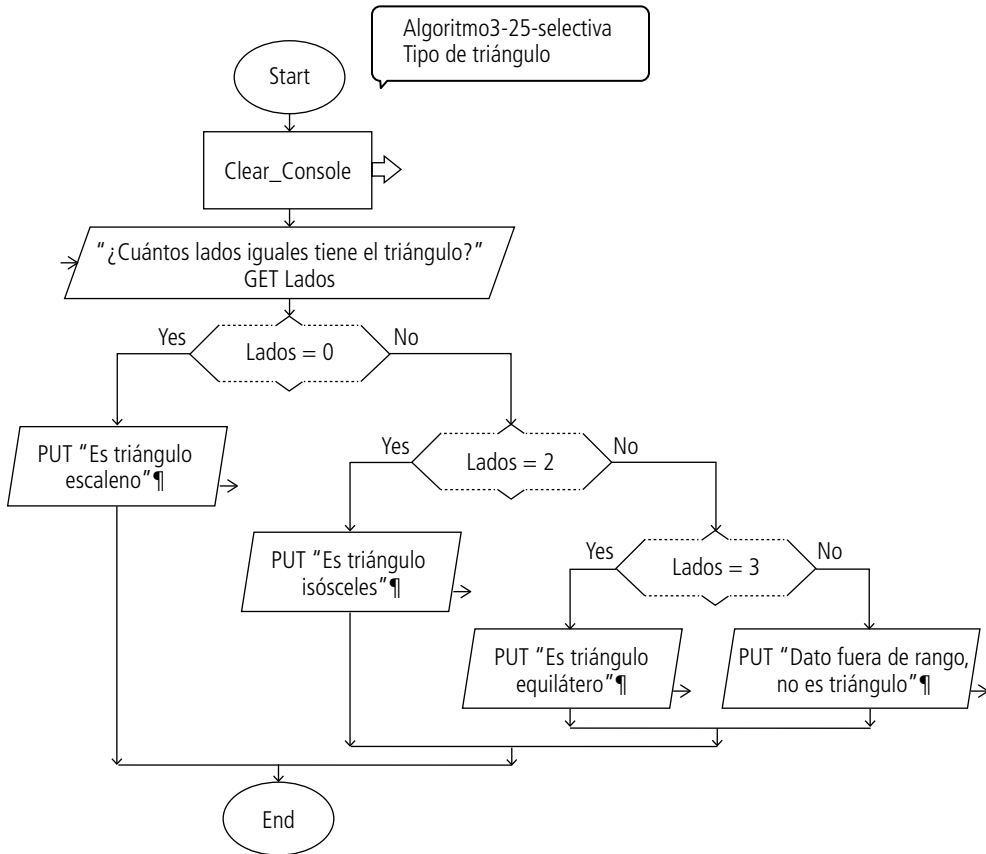


Figura 3.27 Algoritmo3-25-selectiva.

En el diagrama del **Algoritmo3-26-selectiva** presentamos el problema anterior, pero ahora conocemos las longitudes en metros de cada lado del triángulo.

Por cuestiones didácticas, los nombres de las variables no se ajustan a las reglas de calidad porque deseamos que el diagrama pueda visualizarse de mejor manera (véase figura 3.28).

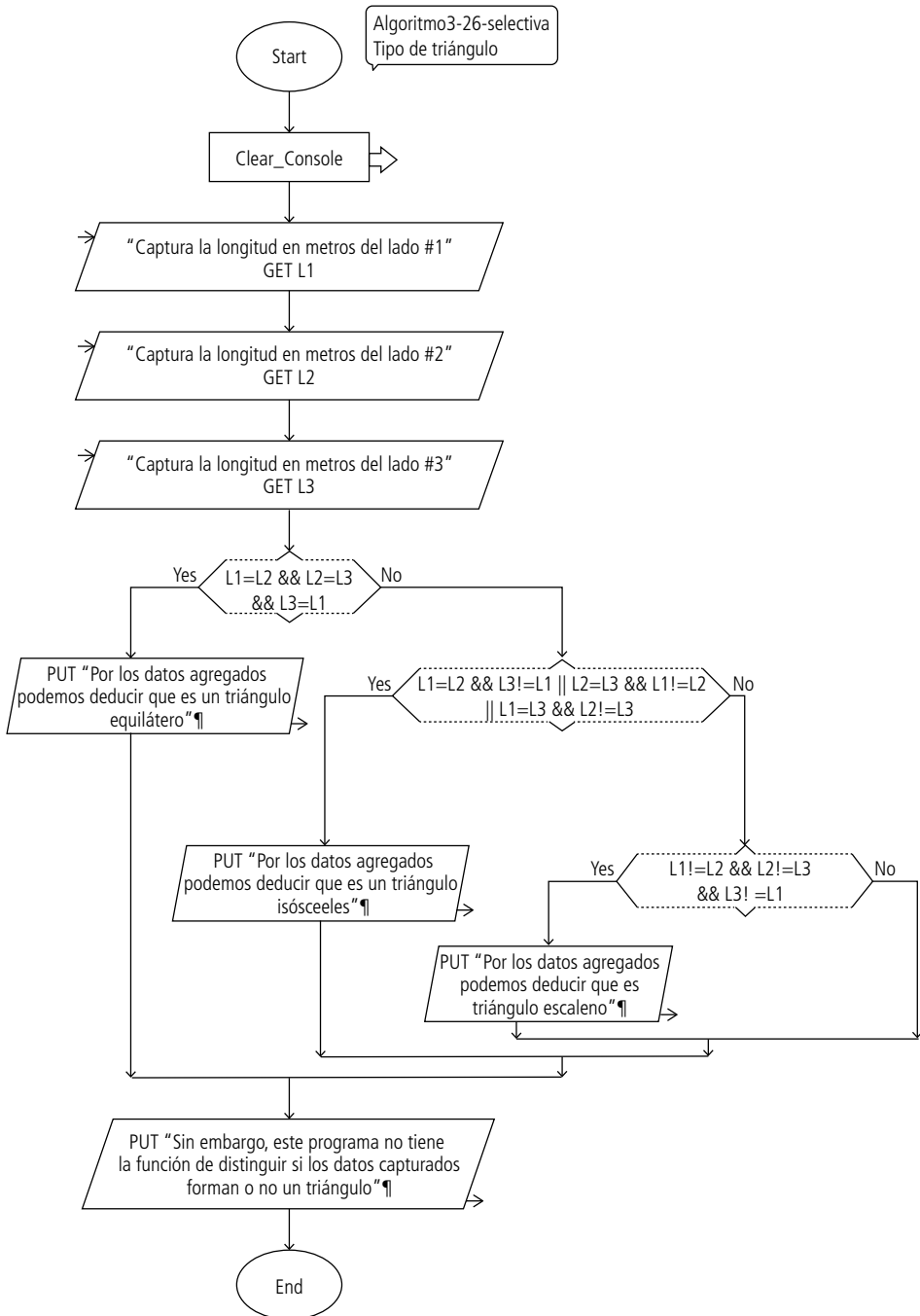


Figura 3.28 Algoritmo3-26-selectiva.



3.7 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 3.5.

- 3.1 ¿Cómo se denomina la acción computacional que permite configurar diferentes opciones mutuamente excluyentes que se pueden elegir a partir de expresiones matemáticas condicionales, las cuales se plantean con operadores relacionales y lógicos y que pasan por un proceso de evaluación por medio del cual se determina su veracidad o falsedad?
- 3.2 ¿Cómo se denomina la estructura selectiva que únicamente tiene dos opciones posibles para elegir?
- 3.3 ¿Cómo se denomina la estructura selectiva doble que tiene acciones computacionales por una de sus dos opciones?
- 3.4 ¿Cómo se denomina la estructura selectiva doble que por ambas opciones tiene acciones computacionales?
- 3.5 ¿Cómo se denomina el símbolo de Raptor que permite la implementación de una estructura selectiva doble? Se representa con la figura de un rombo con dos flechas divergentes que marcan rutas o caminos: Yes y No, independientemente de cuál camino se tome, ambos vuelven a unirse.
- 3.6 ¿Cómo se denominan los operadores que permiten plantear expresiones matemáticas que se refieren a la correspondencia de condición y orden que las entidades abstractas pueden tener unas con respecto a otras? Se utilizan para comprobar la veracidad o falsedad de determinadas propuestas de correspondencia. En realidad se trata de encontrar respuestas a preguntas de comparación o clasificación entre las entidades o los valores asignados a sus identificadores.
- 3.7 Los operadores relacionales requieren de dos operandos para funcionar, por ello se dice que son _____.
- 3.8 El requisito para que los operadores binarios funcionen es que los operandos tengan asignados valores _____.
- 3.9 ¿Cómo se denominan los operadores que permiten plantear expresiones de álgebra booleana que enlazan, separan o complementan a las expresiones relacionales y se usan para soportar las operaciones de conjunción, disyunción inclusiva y negación?
- 3.10 ¿Cuál es el operador simbólico que representa la operación de conjunción entre expresiones matemáticas relacionales?
- 3.11 ¿Cuál es el operador simbólico que representa la operación de disyunción inclusiva entre expresiones matemáticas relacionales?
- 3.12 ¿Cuál es el operador simbólico que representa la operación de negación de una expresión matemática relacional?

- 3.13** ¿Cuál es el operador alfabético que se refiere a la operación de conjunción entre expresiones matemáticas relacionales?
- 3.14** ¿Cuál es el operador alfabético que se refiere a la operación de disyunción inclusiva entre expresiones matemáticas relacionales?
- 3.15** ¿Cuál es el operador alfabético que se refiere a la operación de negación de una expresión matemática relacional?
- 3.16** ¿Cuál es el nombre genérico que reciben las expresiones matemáticas que se plantean con operadores relacionales y lógicos?
- 3.17** ¿Cuál es la forma que toma el rombo de Selection cuando se expande?
- 3.18** ¿Cuál es la función de Raptor que permite verificar si el valor capturado a un identificador es numérico?
- 3.19** Sólo los números _____ pueden clasificarse como pares o impares.
- 3.20** En una división entera, todo número entero par dividido entre 2 genera un residuo de _____.
- 3.21** En una división entera todo número entero impar dividido entre 2 genera un residuo de _____.
- 3.22** Cuando a un identificador se le asignan o agregan valores que son exclusivamente caracteres del abecedario se dice que es de tipo _____.
- 3.23** Cuando un valor alfabético se asigna a un identificador en un símbolo Assignment, ¿qué carácter se debe colocar, antes y después del valor?

Tabla 3.5 Respuestas a los ejercicios de autoevaluación

3.1 Estructura selectiva	3.2 Estructura selectiva doble	3.3 Estructura selectiva doble simple
3.4 Estructura selectiva doble compuesta	3.5 Selection	3.6 Operadores relacionales
3.7 Binarios	3.8 Del mismo tipo de dato	3.9 Operadores lógicos
3.10 &&	3.11	3.12 !
3.13 And	3.14 Or	3.15 Not
3.16 Condición	3.17 Hexagonaloides	3.18 Is_Number()
3.19 Enteros	3.20 Cero	3.21 Uno
3.22 Alfabético	3.23 Comillas altas	



3.8 Problemas propuestos

Al realizar los siguientes programas establezca mecanismos para impedir el cálculo de operaciones matemáticas que provoquen errores de ejecución o arrojen resultados erróneos o inconsistentes.

- 3.1 Hacer un algoritmo que permita calcular el promedio teniendo como datos 3 calificaciones de un alumno y mostrar en pantalla si el estudiante está aprobado o no. La mínima aprobatoria es 70.
- 3.2 Crear un algoritmo que permita capturar dos números enteros, el primero mayor que el segundo, y determinar si el primer número es divisible entre el segundo número.
- 3.3 Hacer un algoritmo que determine cuál es el mayor de tres números enteros diferentes.
- 3.4 Obtener un algoritmo que permita capturar tres valores diferentes y que los muestre ordenados de mayor a menor.
- 3.5 Determinar un algoritmo que calcule y muestre en pantalla el sueldo diario de un obrero que percibe \$11 por cada hora de trabajo y \$19.55 por cada hora extra. La jornada normal de trabajo consta de ocho horas, arriba de eso se considera como tiempo extra. Si la jornada excede las 13 horas de trabajo, se le retendrá el 10% como ahorro.
- 3.6 Hacer un algoritmo que a los estudiantes de una universidad les permita calcular el pago de sus mensualidades con base en el siguiente criterio:
 - a) Al adelantar mensualidades se aplicará un descuento de 10%.
 - b) Al atrasar las mensualidades se aplicará un cargo extra de 10%.
 - c) El pago de la mensualidad del mes corriente no lleva descuento, pero tampoco se le aplica cargo extra. El monto de la mensualidad es de \$650.

Pistas: Pago de una mensualidad adelantada y la del mes actual, el total es de \$1,235. Pago de una mensualidad atrasada y la del mes actual, el total es de \$1,365. Pago de dos mensualidades atrasadas y la del mes actual y una adelantada, el total es de \$2,665.
- 3.7 Hacer un algoritmo que muestre un menú y permita elegir al usuario cuál de las cuatro operaciones aritméticas básicas quiere hacer: suma, resta, multiplicación y división (incluir opción para salir).

- 3.8** Hacer un algoritmo que le permita al usuario seleccionar diversos tipos de conversiones del Sistema Internacional al Sistema Inglés. No olvidar la opción de salir.

Distancia: metros a yardas: $1 \text{ m} = 1.0936 \text{ yardas}$

Velocidad: m/min a ft/min: $1 \text{ m/min} = 3.28 \text{ ft/min}$

Temperatura: grados centígrados a Fahrenheit: $^{\circ}\text{C} = (^{\circ}\text{F} - 32)/1.8$

Masa: kilogramos a libras: $1 \text{ kg} = 2.2046 \text{ lb}$

Salir

- 3.9** Hacer el algoritmo que calcule la factura de energía eléctrica, según la tarifa de zona siguiente:

Zona 1 \rightarrow \$1.25 por kilowatt-hora consumido.

Zona 2 \rightarrow \$1.49 por kilowatt-hora consumido.

Zona 3 \rightarrow \$1.73 por kilowatt-hora consumido.

Zona 4 \rightarrow \$1.95 por kilowatt-hora consumido.

Se debe imprimir el número de servicio (es único para cada cliente), los kilowatt-hora consumidos y el total a pagar por el cliente.

- 3.10** Si conoce el espacio total en un disco duro y también, el espacio ocupado, realice un programa que calcule el porcentaje de espacio disponible.

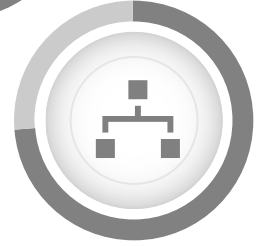
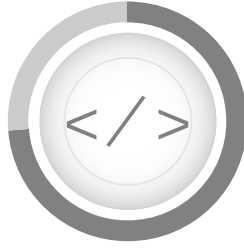
- 3.11** Está leyendo un libro. Dado que conoce el total de páginas y la página que está leyendo actualmente, realice un programa que le ayude a calcular el porcentaje de avance de lectura en cualquier momento.

- 3.12** Realice un programa que eleve al cuadrado y al cubo cualquier número y que imprima el número junto a su cuadrado y cubo.

- 3.13** Antes de despegar un avión, el piloto anuncia el tiempo estimado de vuelo en minutos. Realice un programa que le ayude a determinar el porcentaje de avance del vuelo, si tiene como datos conocidos el tiempo estimado y el tiempo transcurrido del vuelo en minutos.

- 3.14** Un profesor desea determinar el porcentaje de aprobados y reprobados en un grupo. Si sólo sabe cuántos alumnos han aprobado y cuántos, reprobado, realice un programa que le ayude a calcular estos porcentajes.

- 3.15** Una profesora midió la altura de Juan al principio y al final del año escolar. Realice un programa que le ayude a determinar el porcentaje de crecimiento de Juan.



Capítulo 4

Diseño de algoritmos con la estructura repetitiva

- 4.1 Introducción
- 4.2 Ciclo controlado por contador
- 4.3 Ciclo controlado por centinela
- 4.4 Ciclo para validación de datos
- 4.5 Ciclos para el manejo de un conjunto de datos similares
- 4.6 Sucesiones numéricas
- 4.7 Ejercicios de autoevaluación
- 4.8 Problemas propuestos

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Diseñar algoritmos computacionales utilizando la acción computacional del ciclo o estructura repetitiva.
- Utilizar la estructura repetitiva controlada por contador y centinela.
- Distinguir entre variables contadoras y acumuladoras.
- Utilizar la estructura repetitiva para validación de datos.
- Utilizar la estructura repetitiva para generar sucesiones numéricas.

4.1 Introducción

A la acción computacional con la que se puede configurar una opción repetitiva y una opción diferida que no son mutuamente excluyentes, se le denomina ciclo o estructura repetitiva. La opción repetitiva consiste en que una o más acciones computacionales puedan ejecutarse en la misma secuencia y de manera iterativa, bajo un ambiente controlado para que de manera eventual deje de dar vueltas, impidiendo generar ciclos infinitos. La ejecución o no de la opción repetitiva se plantea a partir de expresiones matemáticas condicionales con el uso de operadores lógicos y relacionales. La opción diferida consiste en una o más acciones computacionales cuya ejecución se posterga para después de que el ciclo haya llegado a su fin.

La condición matemática que se utiliza para controlar el ciclo pasa por un proceso de evaluación que determina su veracidad o falsedad. La opción repetitiva se puede ejecutar, ya sea por el lado falso o verdadero. Después de que la condición se cumple, se ejecutan las acciones de la opción diferida.

Para una mejor explicación de lo anterior, habrá que puntualizar que hay dos formas de configurar un ciclo: bajo la lógica de **HASTA QUE** y bajo la lógica de **MIENTRAS QUE**. En el primer caso, la repetición de acciones computacionales tiene lugar cuando la condición se evalúa como falsa, y una vez que se termina el ciclo se continúa por el camino verdadero. Por tanto, podemos decir que la repetición es **HASTA QUE** la condición sea verdadera.

Por el contrario, en la lógica de **MIENTRAS QUE** la repetición de acciones computacionales tiene lugar cuando la condición se evalúa como verdadera, y una vez que se termina el ciclo se continúa con el camino falso. Por tanto, podemos decir que la repetición es **MIENTRAS QUE** la condición se evalúe como verdadera.

Independientemente de la lógica en la que basen los ciclos, éstos se clasifican en los controlados por contador y los controlados por centinela. Además de mostrar ejemplos de ambos casos, destacaremos el uso de ciclos para validar los valores que se agregan a las variables.

Raptor cuenta únicamente con el símbolo Loop para establecer un ciclo y funciona bajo la lógica de **HASTA QUE**, es decir, da vueltas **HASTA QUE** la condición sea verdadera, o lo que es lo mismo, por el lado falso de la evaluación de la condición. Raptor no tiene un símbolo que permita usar los ciclos bajo la lógica de **MIENTRAS QUE**.

En la figura 4.1 se puede observar el símbolo Loop de Raptor. El globo que tiene dentro la palabra Loop implica el inicio del ciclo. Dentro del rombo se coloca una condición que puede ser evaluada como verdadera o falsa, tomando solamente uno de los dos caminos posible: Yes o No, los cuales no se unen. El camino de No es la opción repetitiva y regresa al globo del Loop, esto implica que cuando la evaluación de la condición es falsa se ejecutan las acciones que forman parte del ciclo. El camino de Yes es la opción diferida y sus acciones se ejecutan después de que el ciclo deja de dar vueltas (véase figura 4.1).

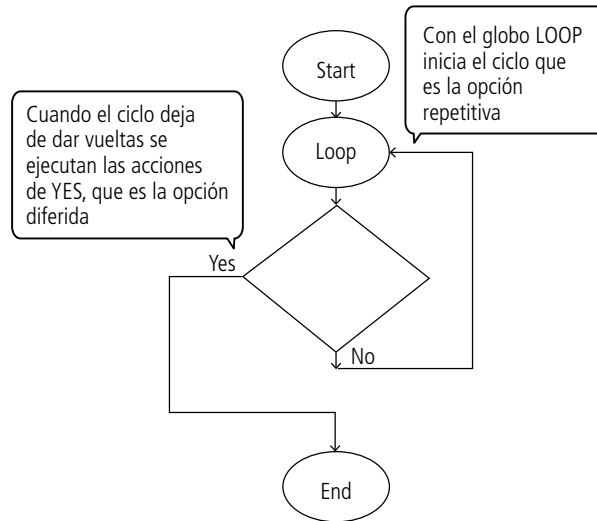


Figura 4.1 Símbolo Loop.

El símbolo Loop tiene la flexibilidad de permitir que las acciones que se van a repetir puedan estar antes o después de que la condición de control se evalúe como falsa. Que estén antes de que la condición de control sea evaluada implica que el ciclo por lo menos se ejecuta una vez. Si están después de que la condición de control es evaluada implica que el ciclo corre el riesgo de que no se ejecute ni una vez. Las acciones están antes de la condición de control cuando aparecen entre el globo de Loop y la condición de control. Las acciones están después de la condición de control cuando aparecen en el camino de *No*, es decir, entre la condición de control y globo de Loop.

4.2 Ciclo controlado por contador

Una de las características de un ciclo controlado por contador es que se conoce de antemano la cantidad de veces que se repetirán las acciones computacionales. Se requiere de una variable que controle el ciclo y que maneje tres parámetros:

1. El inicio del ciclo.
2. Una condición para terminar el ciclo.
3. Un movimiento de la variable controladora que con cada vuelta se acerque al cumplimiento de la condición, permitiendo, eventualmente, la salida o terminación del ciclo.

El símbolo de Raptor que permite establecer una repetición automática de un conjunto de acciones computacionales se llama Loop. En la mayor parte de la literatura que explica la programación de computadoras, se utiliza la variable i para controlar los ciclos. Para no contrariar a esa costumbre en este libro se seguirá con esa tradición, no sin antes aclarar que el nombre de esa variable no está bajo las reglas de calidad que aquí mismo hemos establecido.

En el **Algoritmo4-1-ciclo** hay una estructura repetitiva que da diez vueltas y en cada una de ellas se imprime, en el *MasterConsole*, la i -ésima vuelta en la que se encuentra. La característica de este diagrama es que las acciones computacionales que se repiten se encuentran antes de que la condición de control sea evaluada, entre el globo de Loop y la condición de control. Por ello, independientemente de lo que suceda, los pasos a repetirse se ejecutan por lo menos una vez (véase figura 4.2).

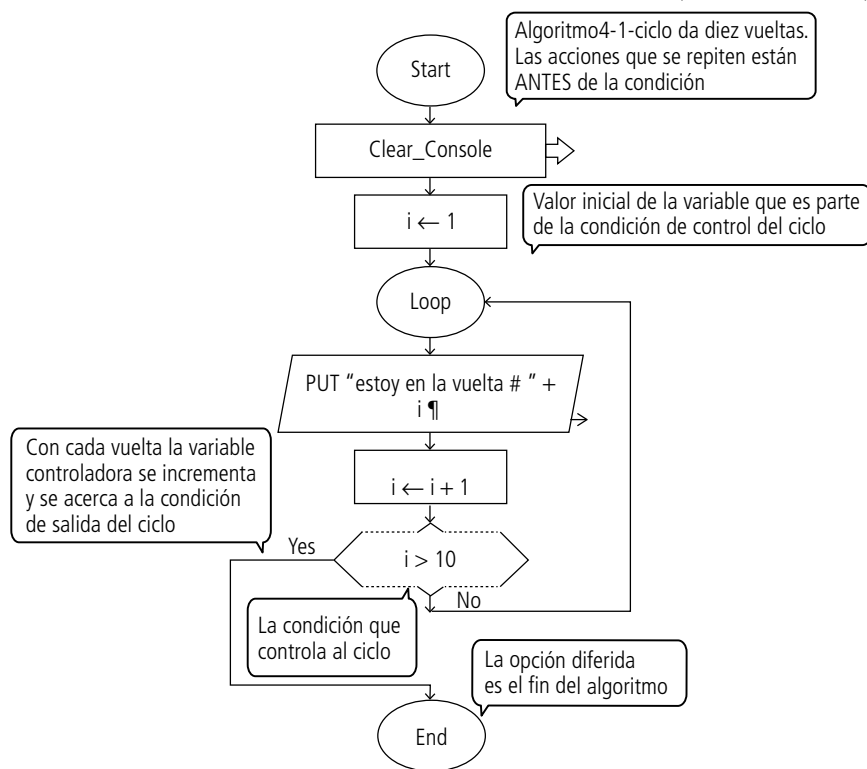


Figura 4.2 Algoritmo4-1-ciclo. Ejemplo de un ciclo que da diez vueltas. Las acciones que se repiten están ANTES de la condición de control.

A continuación con el **Algoritmo4-2-ciclo** mostramos otra forma de implementar el símbolo Loop; en este caso, hay un ciclo que da diez vueltas y en cada una de ellas se imprime, en el *MasterConsole*, la *i-ésima* vuelta en la que se encuentra. La característica de este diagrama es: las acciones computacionales que se repiten, se encuentran después de que la condición de control se ha evaluado; esto es, por el camino del No, entre la condición de control y el globo de Loop. Por ello, si el ciclo no está bien implementado, se corre el riesgo de que los pasos a repetirse no se ejecuten nunca (véase figura 4.3).

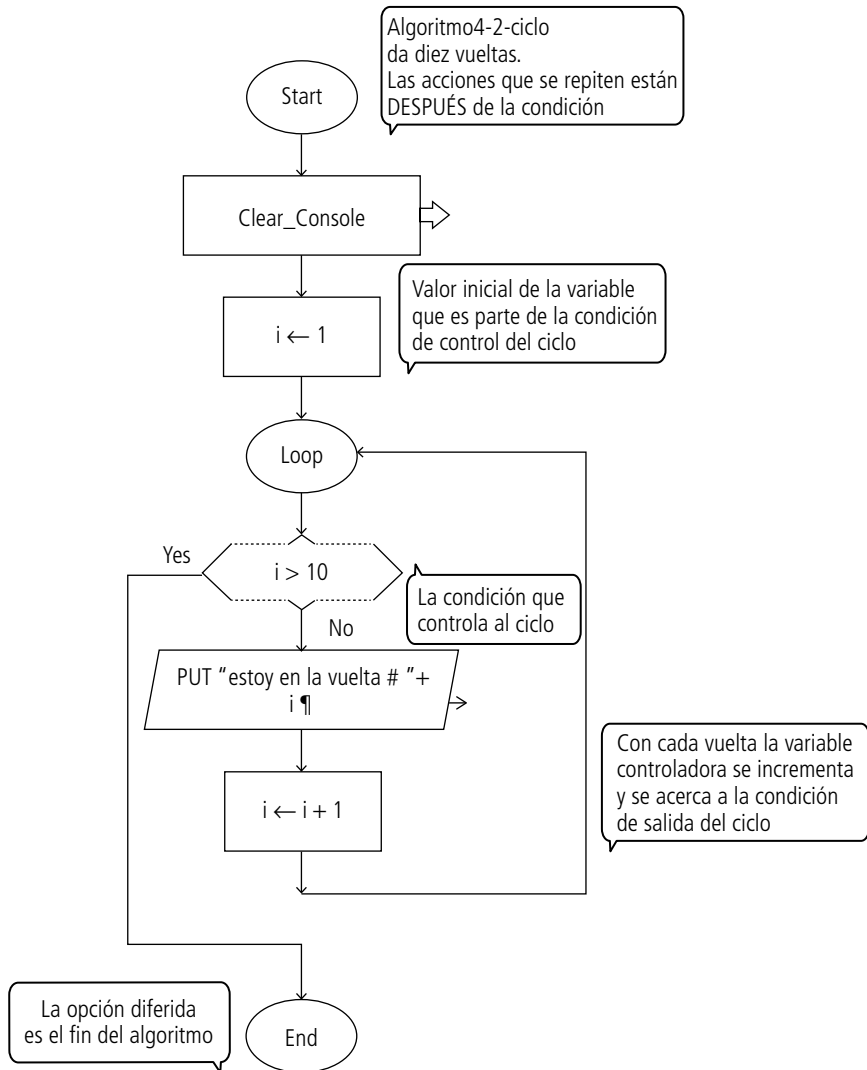


Figura 4.3 Algoritmo4-2-ciclo. Ejemplo de un ciclo que da diez vueltas. Las acciones que se repiten están después de que la condición de control se ha evaluado como falsa.

Compare el **Algoritmo4-1-ciclo** con el **Algoritmo4-2-ciclo**. Ambos diagramas hacen lo mismo, pero en el primero el Output y el Assignment están antes de la condición de control; mientras que en el segundo están después de la condición por el camino del No. El lugar adecuado deberá ser evaluado de acuerdo con cada programa, ya que en ocasiones el resultado puede variar.

Otra cosa que deseamos que observe es la asignación $i \leftarrow i + 1$, con esto se dice que la variable i se incrementa en uno. Como i es la variable controladora del ciclo, este incremento le permite con cada vuelta acercarse a la condición que le permitirá dejar de dar vueltas y, por tanto, terminar el ciclo. Si este incremento no existiera, el ciclo sería infinito y nunca terminar.

En el **Algoritmo4-3-ciclo** tenemos el ejemplo anterior, aunque antes se solicita al usuario que establezca la cantidad de vueltas que dará el ciclo (véase figura 4.4).

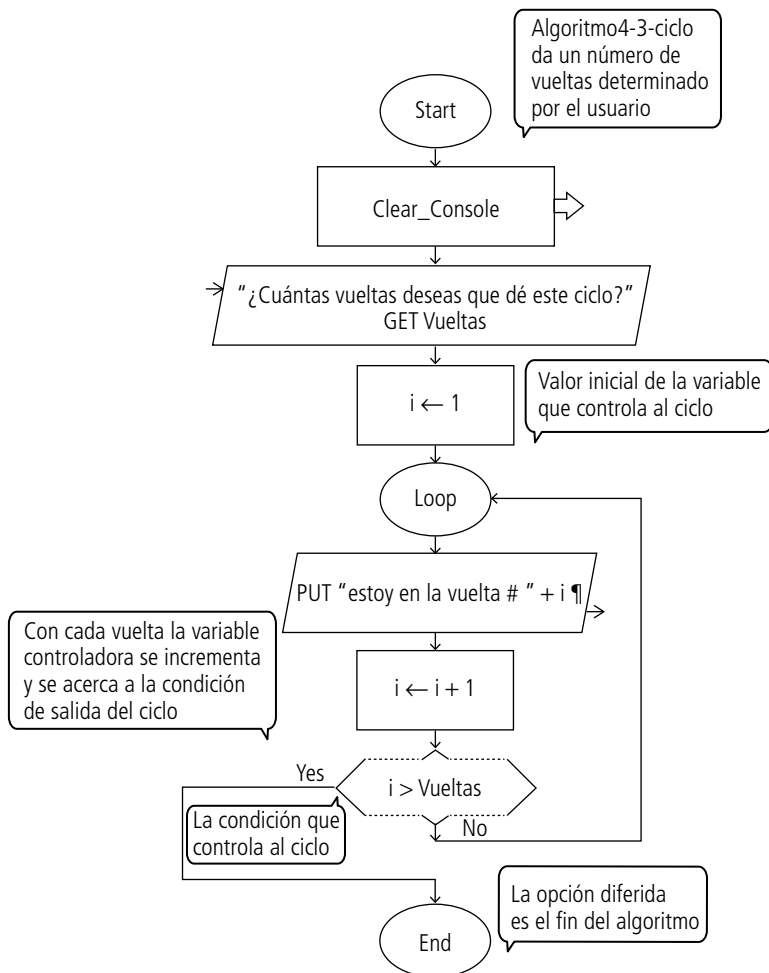


Figura 4.4 Algoritmo4-3-ciclo. La cantidad de vueltas es determinada por el usuario.

En el **Algoritmo4-4-ciclo** se solicita al usuario que agregue el número de la tabla de multiplicar que desea imprimir. El diagrama da diez vueltas y en cada una de ellas calcula y muestra el resultado en la ventana de *MasterConsole* (véase figura 4.5).

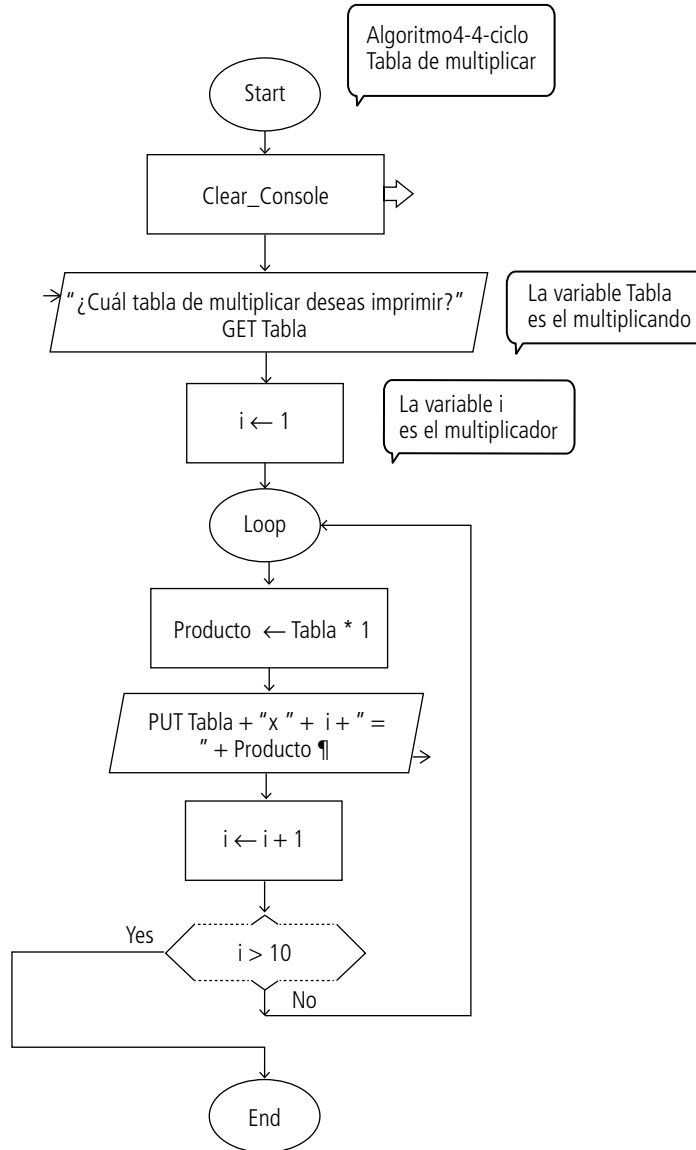


Figura 4.5 Algoritmo4-4-ciclo, imprime cualquier tabla de multiplicar.

En el **Algoritmo4-5-ciclo** se muestra la tabulación de los diez valores tomados por X y Y de la ecuación cuadrática: $Y = 6X^2 + 4X - 3$. Se solicita al usuario que agregue el valor inicial de la variable independiente. Luego, se entra a un ciclo que da diez vueltas,

con cada vuelta el valor de X tendrá un incremento de tres unidades, calculándose un nuevo valor para la variable Y . En la ventana de *MasterConsole* se muestran los nuevos valores tanto para X como para Y .

Los nombres de las variables no cumplen con nuestras reglas de calidad, pero por cuestiones didácticas hemos permitido esta situación. De acuerdo con estas reglas, la variable X debería llamarse *variable independiente*, y la variable Y , *variable dependiente* (véase figura 4.6).

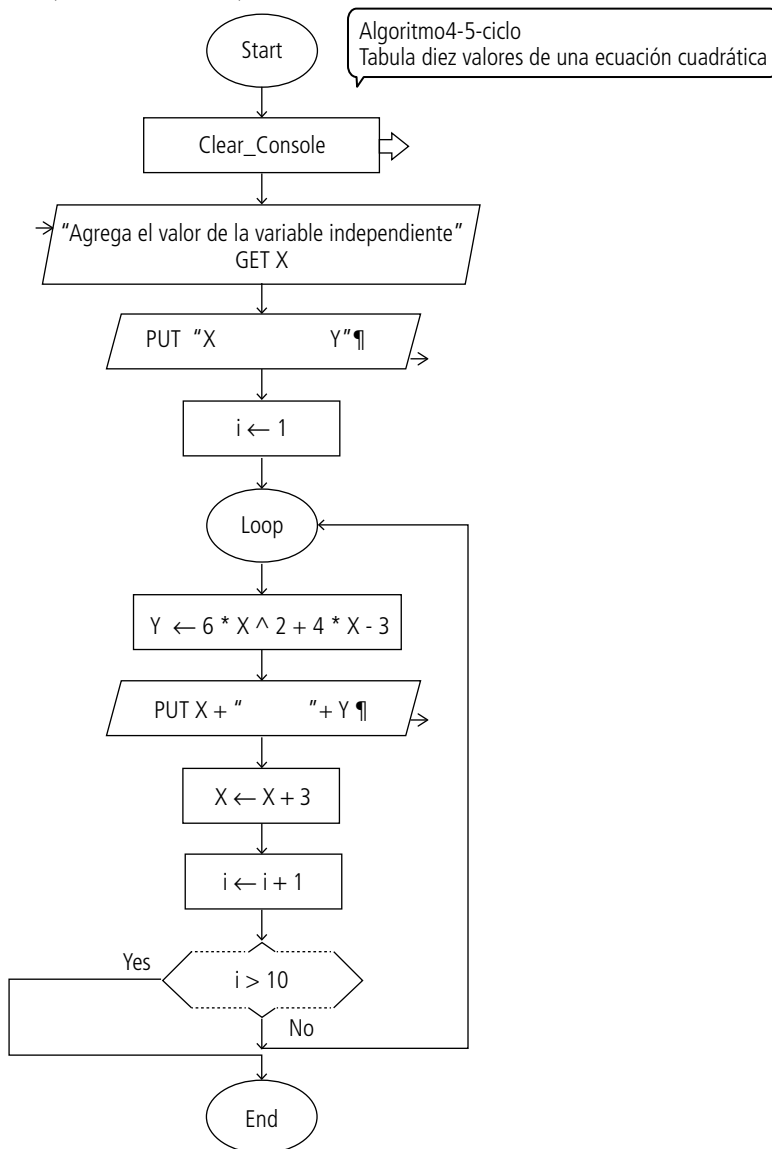


Figura 4.6 Algoritmo4-5-ciclo, presenta la tabulación de una ecuación cuadrática.

4.2.2 Contadores y acumuladores

En programación es común realizar la acción de contabilizar elementos y acumular valores, lo cual se establece dentro de la opción repetitiva de un ciclo. Un identificador que se distingue porque con cada iteración del ciclo recibe su incremento o decremento constante se denomina CONTADOR. Cuando el incremento o decremento del identificador es variable se llama ACUMULADOR. Su estructura es muy similar, ya que ambos deben iniciarse con algún valor antes de que inicie la opción repetitiva del ciclo. Con frecuencia este valor de inicialización es cero, pero puede tener cualquier otro valor.

Ejemplos:

1. Para conteos positivos.
Valor inicial: Contador $\leftarrow 0$
Incremento constante: Contador \leftarrow Contador + Constante
2. Para conteos negativos.
Valor inicial: Contador $\leftarrow 30$
Decremento constante: Contador \leftarrow Contador - Constante
3. Para acumulados positivos (mediante la suma).
Valor inicial: Acumulador $\leftarrow 0$
Incremento variable: Acumulador \leftarrow Acumulador + Variable
4. Para acumulados negativos (mediante la resta).
Valor inicial: Acumulador $\leftarrow 20$
Decremento Variable: Acumulador \leftarrow Acumulador - Variable

En el **Algoritmo4-6-ciclo** se suman los primeros cien números enteros. El diagrama da cien vueltas y en cada una de éstas va sumando los números e imprimiendo el acumulado de la suma. Una vez que el ciclo deja de dar vueltas, se imprime el resultado del programa. La variable i funciona como contador y la variable Suma, como acumulador (véase figura 4.7).

En el **Algoritmo4-7-ciclo** se solicita al usuario que agregue la cantidad de números que desea sumar, esa será la cantidad de vueltas que dará el ciclo. En cada vuelta se pide al usuario la captura manual de uno de los números a ser sumados. Una vez que el ciclo deja de dar vueltas, se imprime el resultado de la sumatoria.

Observe que el símbolo Input, donde se solicita la inserción del número a ser sumado, utiliza un mensaje compuesto. Con cada vuelta el mensaje va cambiando de acuerdo con la variable i (véase figura 4.8).

En el **Algoritmo4-7-ciclo**, la captura de los números fue de manera manual; por ello, el usuario usa el teclado para agregar uno por uno los datos. En el **Algoritmo4-8-ciclo** se ha implementado la asignación automática de los números, mediante la generación de números aleatorios entre cero y 1,000. La desventaja de este método es que el usuario no tiene el control sobre los números que forman parte de la lista. Por otro lado, cuando el diagrama deja de dar vueltas, se calcula el promedio de los números y se muestra el resultado en la ventana de *MasterConsole* (véase figura 4.9).

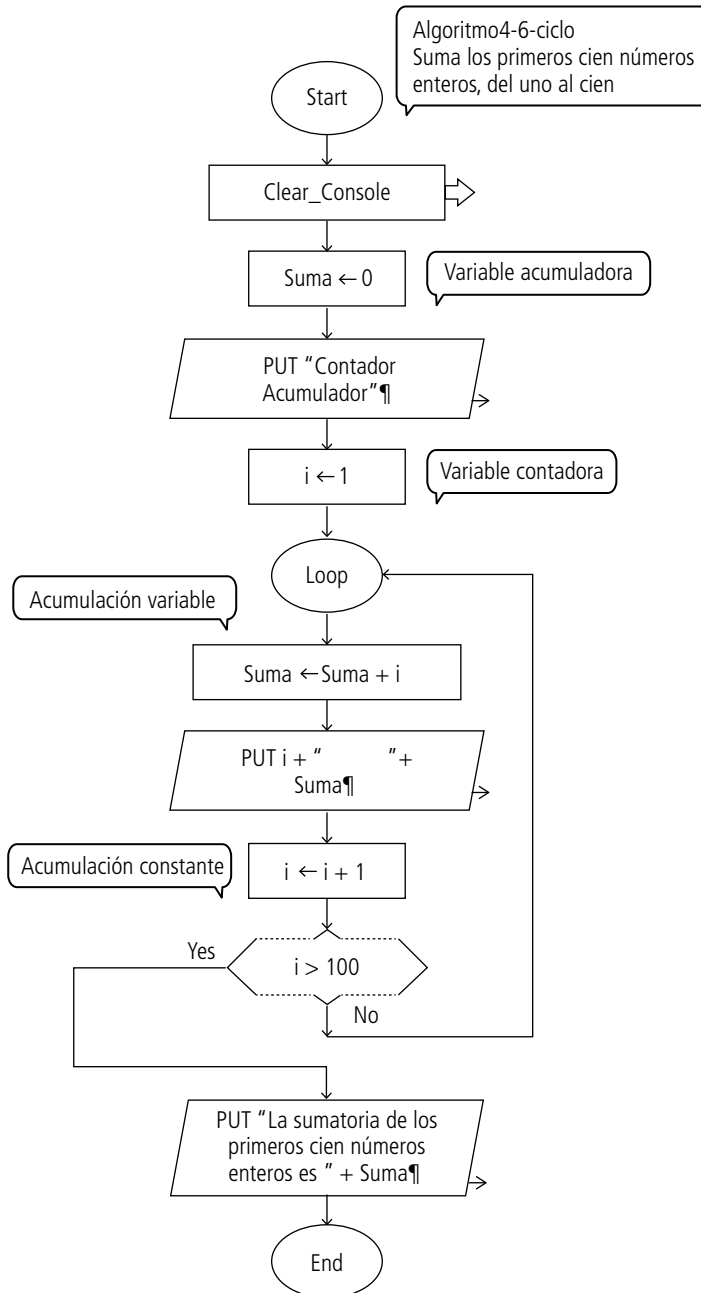


Figura 4.7 Algoritmo4-6-ciclo, suma los primeros cien números enteros.

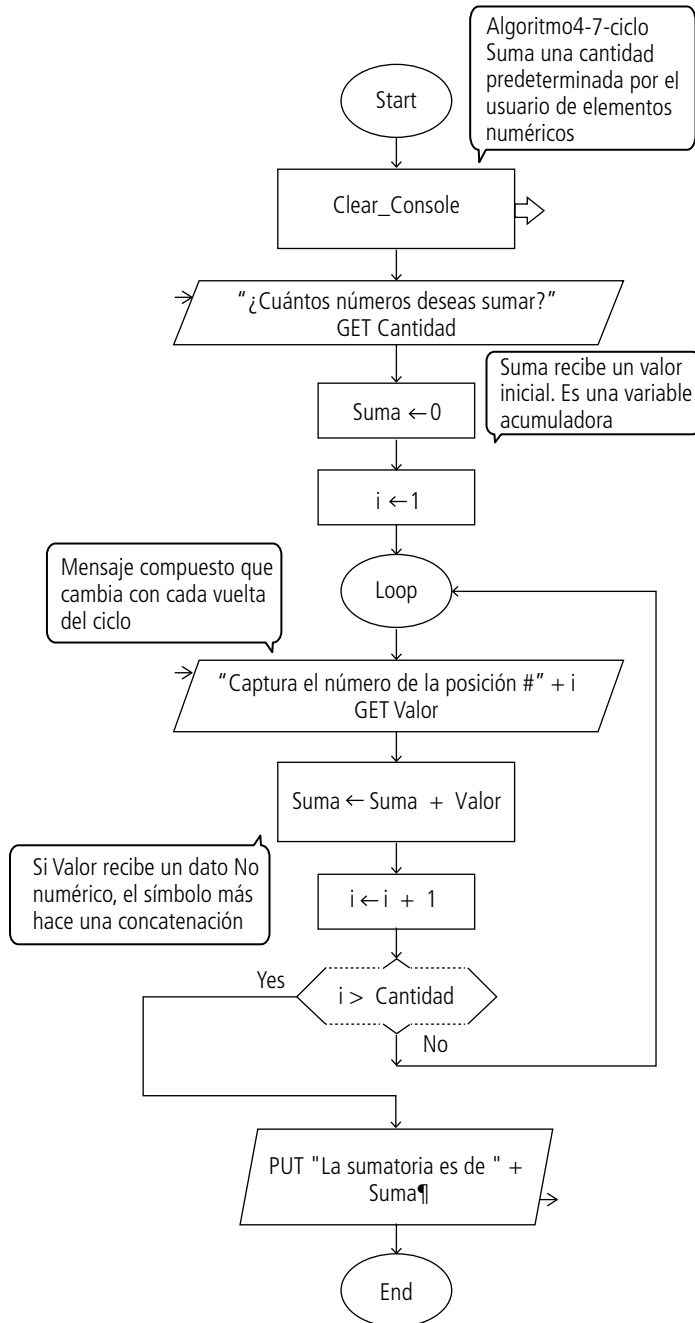


Figura 4.8 Algoritmo4-7-ciclo, suma una cantidad predeterminada más el usuario de elementos numéricos.

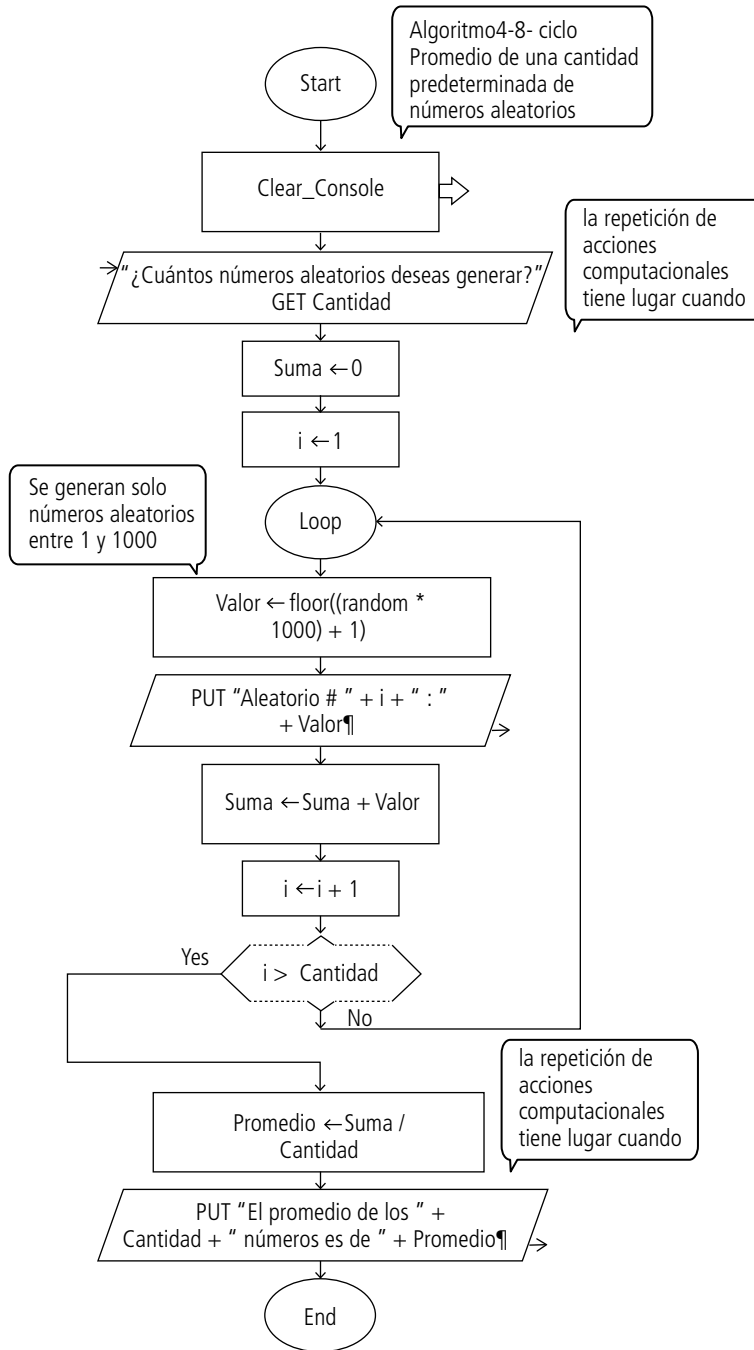


Figura 4.9 Algoritmo4-8-ciclo, obtiene el promedio de una cantidad predeterminada de números aleatorios.

Además de la asignación automática de números aleatorios, el **Algoritmo4-9-ciclo** determina cuál es el número mayor de la lista de números (véase figura 4.10).

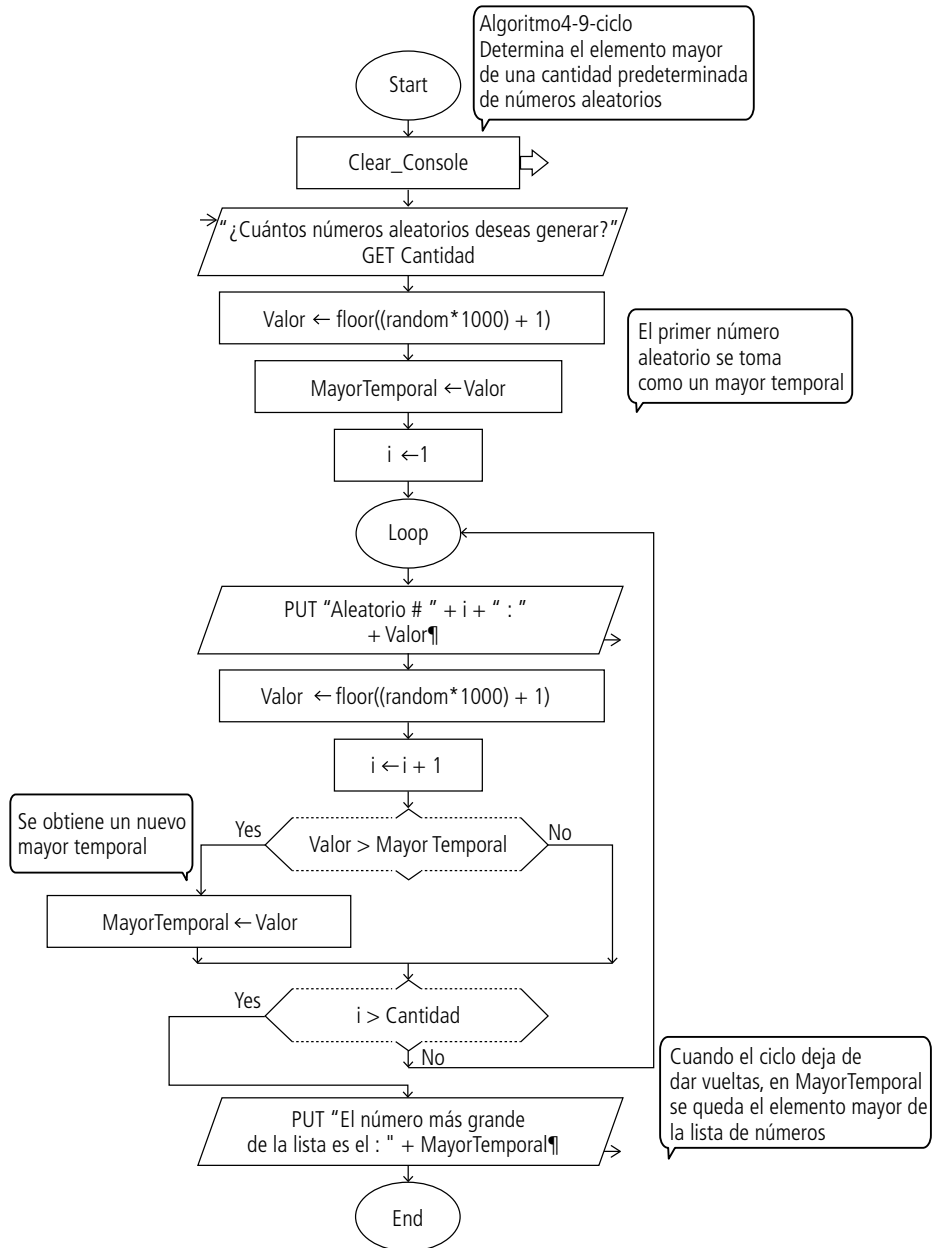


Figura 4.10 Algoritmo4-9-ciclo, determina el elemento mayor de una cantidad predeterminada de números aleatorios.

En el **Algoritmo4-10-ciclo** se solicita al usuario que agregue un número entero positivo para calcular su factorial. El factorial de 5 se calcula multiplicando los números enteros positivos desde uno hasta cinco, como se muestra en el ejemplo: $1*2*3*4*5$ igual a 120 (véase figura 4.11).

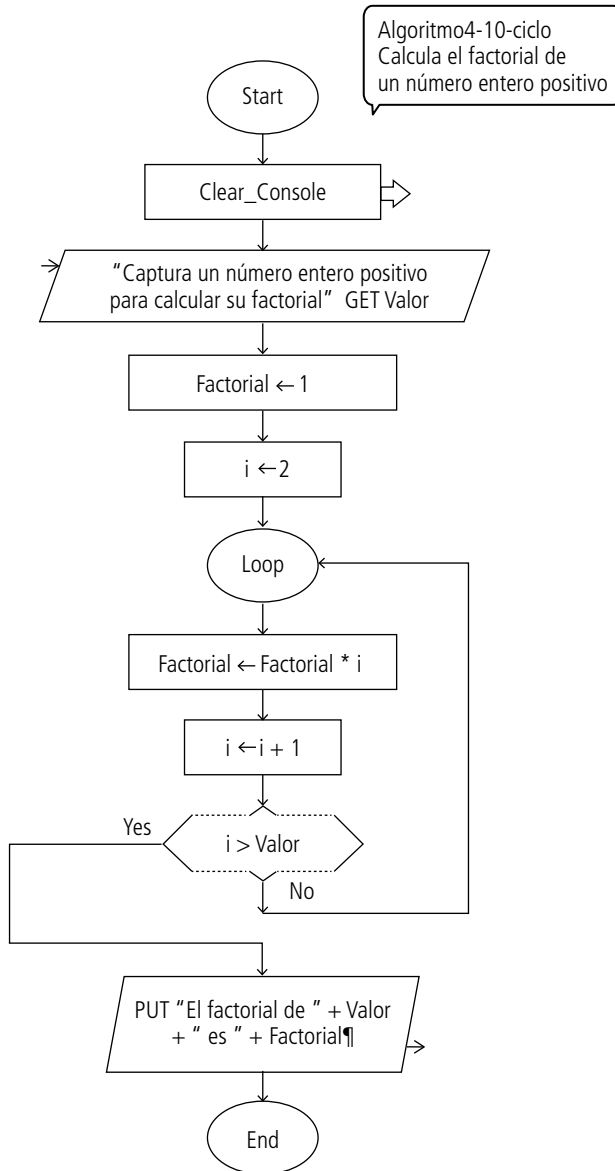


Figura 4.11 Algoritmo4-10-ciclo, cálculo del factorial de un número.

En el **Algoritmo4-11-ciclo** se solicita al usuario que agregue un número entero positivo mayor que 1 para analizar si es un número primo o no. Un número entero positivo es primo cuando sólo tiene dos divisores enteros que lo dividen de manera exacta y, por tanto, arrojan un residuo de cero. El número 1 no se considera primo. El número 2 es el único que es par y al mismo tiempo es primo.

Para poder hacer el programa se inventó una variable lógico-binaria o *switch* que sólo puede tomar dos valores. Cuando toma el valor de cero se dice que la variable es falsa o está apagada y cuando toma el valor de 1 se dice que la variable es verdadera o está prendida. A este tipo de variables también se les llama bandera. En este caso, la variable switch recibió el nombre de Testigo.

Cuando la variable Testigo vale 1, es señal de que el número es primo. Si vale cero, es señal de que el número no es primo. De manera conveniente, al principio del programa, se ha dado a la variable Testigo el valor de 1, el cual cambia sólo a cero cuando se detecta un divisor que divide exactamente el número analizado. Cuando el diagrama deja de dar vueltas se analiza el valor Testigo para enviar a la ventana de *MasterConsole* el resultado correspondiente.

Para tener la certeza de que un número N es primo, tendríamos que dividir dicho número desde el 2 hasta $(N - 1)$. Por ejemplo, si el número fuera 89 tendríamos que hacer 87 divisiones, es decir, dividiríamos desde el 89 entre 2 hasta 89 entre 88. Si todas las veces el residuo fuera cero, entonces confirmaríamos que el 89 es primo.

Sin embargo, al utilizar la lógica matemática tenemos que los divisores de un número sólo se encuentran en su primera mitad, esto es, los números que dividen al 24 se encuentran entre 2 y 12: 2, 3, 4, 6, 8 y 12. Por ello, el diagrama está hecho para que dé la mitad de las vueltas.

La diferencia estaría en que para saber si el 89 es primo o no, en lugar de hacer 87 divisiones sólo sería necesario hacer 45; esto es, la mitad de 89 más 1, aproximadamente.

Los números primos menores que 100 son los siguientes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 y 97.

Para saber si los números negativos pueden ser primos, se le sugiere consultar la teoría de los números (véase figura 4.12).



4.3 Ciclo controlado por centinela

Una de las características de una estructura repetitiva y controlada por centinela es que no se sabe con precisión la cantidad de vueltas que dará el ciclo. Se requiere de una variable que lo controle y que maneje los parámetros siguientes:

1. El inicio del ciclo.
2. Una condición para terminar el ciclo.
3. Una acción que permita modificar el estado de la variable controladora y la verificación en cada vuelta de la condición de control para que en algún momento el ciclo se pueda dar por concluido.

En el siguiente problema se muestra un ejemplo del uso de un ciclo controlado por centinela.

Se deja caer una pelota desde una altura determinada (en metros), rebota y cada vez su altura es de dos tercios de su altura en el rebote anterior (véase figura 4.13). Elabore un programa que determine el número del rebote en el que la altura de la pelota es igual o inferior a 50 centímetros. Además deberá imprimir el número de cada rebote y su correspondiente altura.

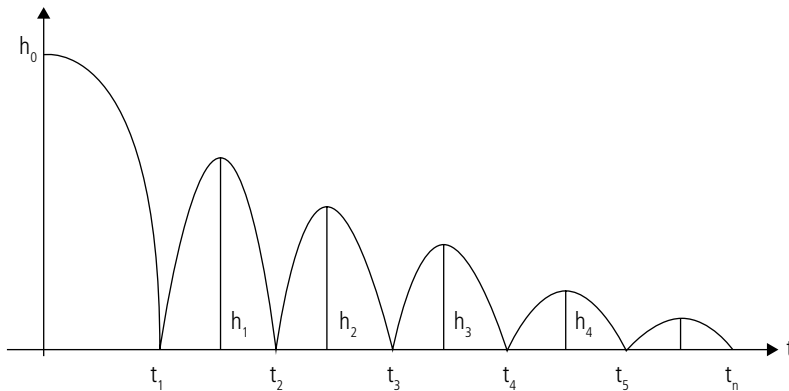


Figura 4.13 Rebote de una pelota.

En el **Algoritmo4-12-ciclo** se solicita al usuario que agregue la altura inicial en metros de donde se deja caer la pelota, pidiéndole que sea una altura superior a 50 centímetros. Si la altura inicial está fuera de rango se hace saber al usuario de esta situación. Si está dentro del rango, se inicia un ciclo que permite calcular los datos de salida del problema.

Observe que la variable *Rebote* permite contar las vueltas que da el ciclo, pero no forma parte de la condición de control que hace que el ciclo termine. Como no sabemos de antemano la cantidad de vueltas que dará el ciclo porque depende del valor de la altura inicial que capture el usuario, entonces este ciclo no se controla por un contador de vueltas, sino por un centinela que está a la espera de que se cumpla una condición para entonces actuar (véase figura 4.14).

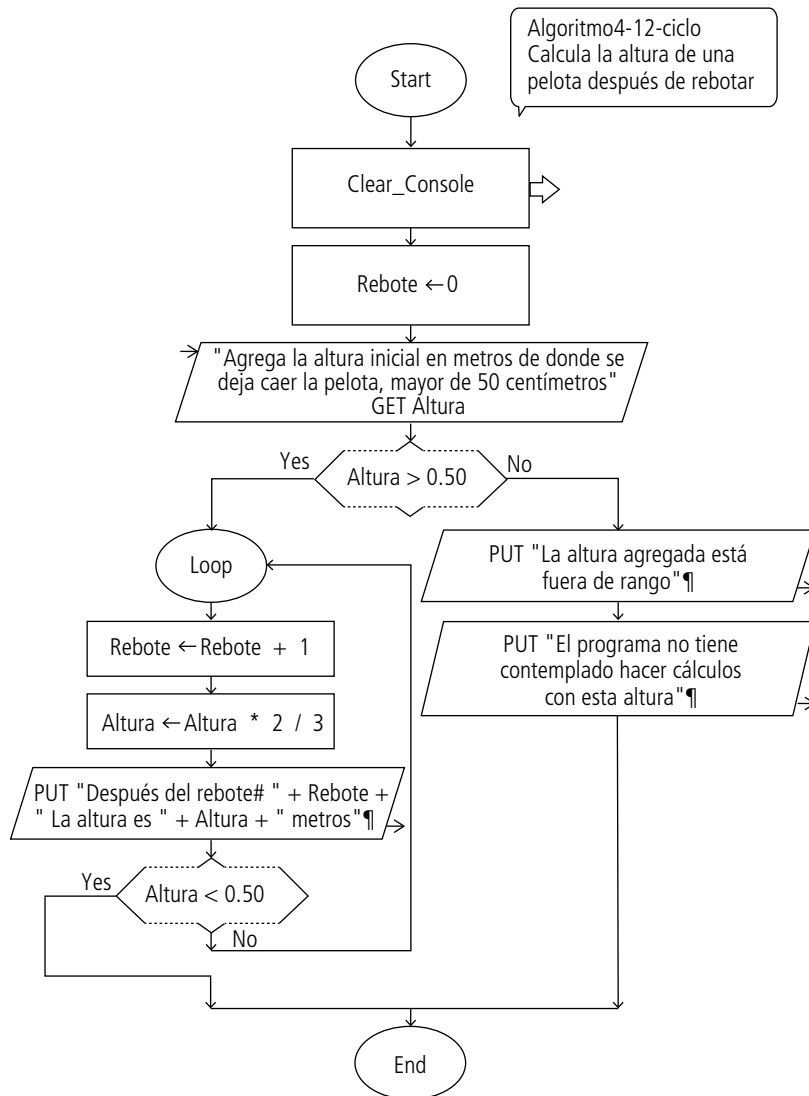


Figura 4.14 Algoritmo4-12-ciclo.

4.4 Ciclo para validación de datos

En realidad, esta situación es una variante del ciclo controlado por centinela, pero dado que su uso es muy frecuente queremos dejar un espacio especial para este tipo de tratamiento.

Una vez que comprenda mejor el objetivo de la validación de variables, se dará cuenta de que en el **Algoritmo4-12-ciclo** no se está haciendo ninguna validación.

La validación de campos o variables implica asegurarse de que a los identificadores se les agrega un valor que tiene el rango o forma correcta, lo que garantiza que el diagrama no generará resultados con errores o inconsistencias. Por ejemplo, si estamos capturando calificaciones, éstas sólo deben estar entre cero y cien, de otra forma se vuelve un dato incongruente para el contexto del problema que se está tratando. La validación envía un mensaje de advertencia al usuario de que ha incurrido en un error y le da oportunidad de corregirlo mediante la captura de un nuevo valor, lo cual se logra con el empleo del símbolo Loop (véanse figuras 4.15 y 4.16).

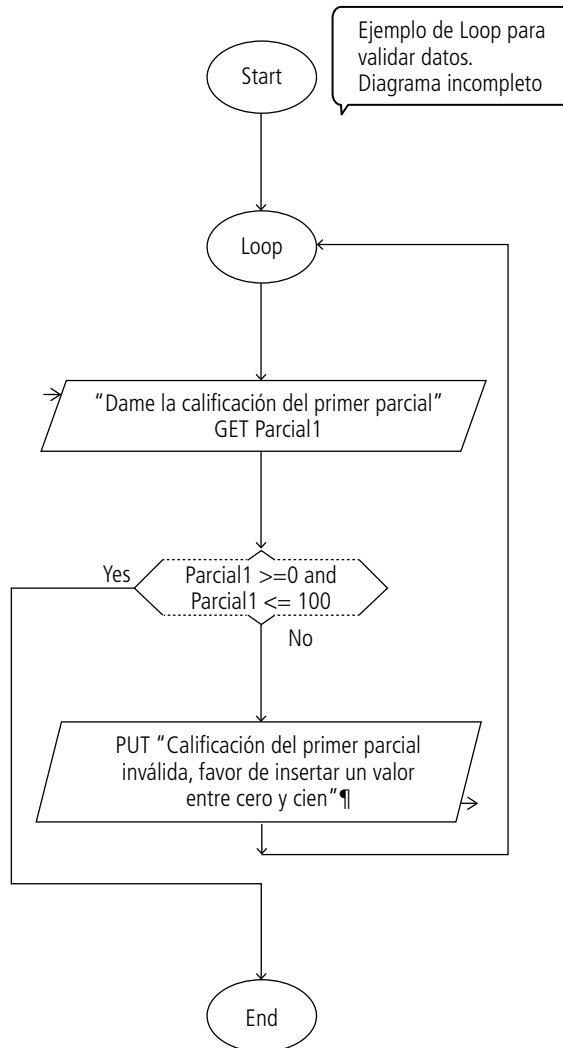


Figura 4.15 Ejemplo de Loop para validar datos. Diagrama incompleto.

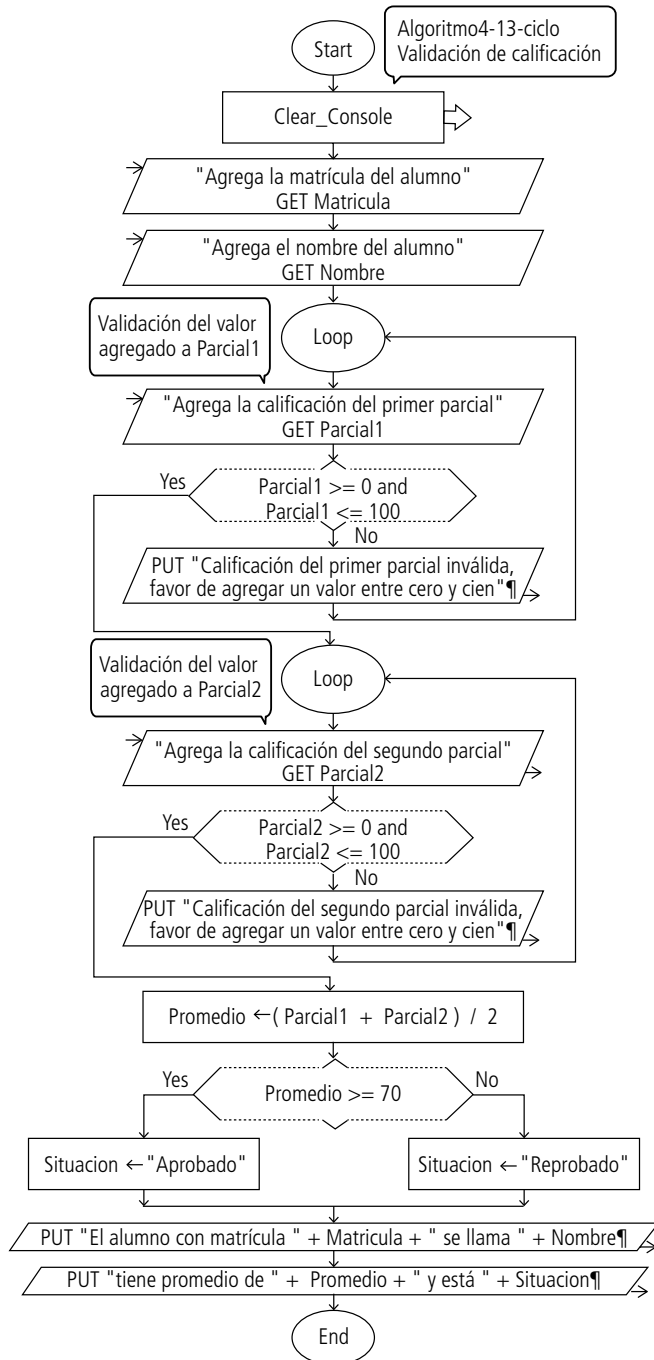


Figura 4.16 Algoritmo4-13-ciclo.

En el **Algoritmo4-14-ciclo** se calcula el valor de la fuerza teniendo como datos conocidos a la masa y la aceleración. Observe que la captura del valor de la masa está validada (véase figura 4.17).

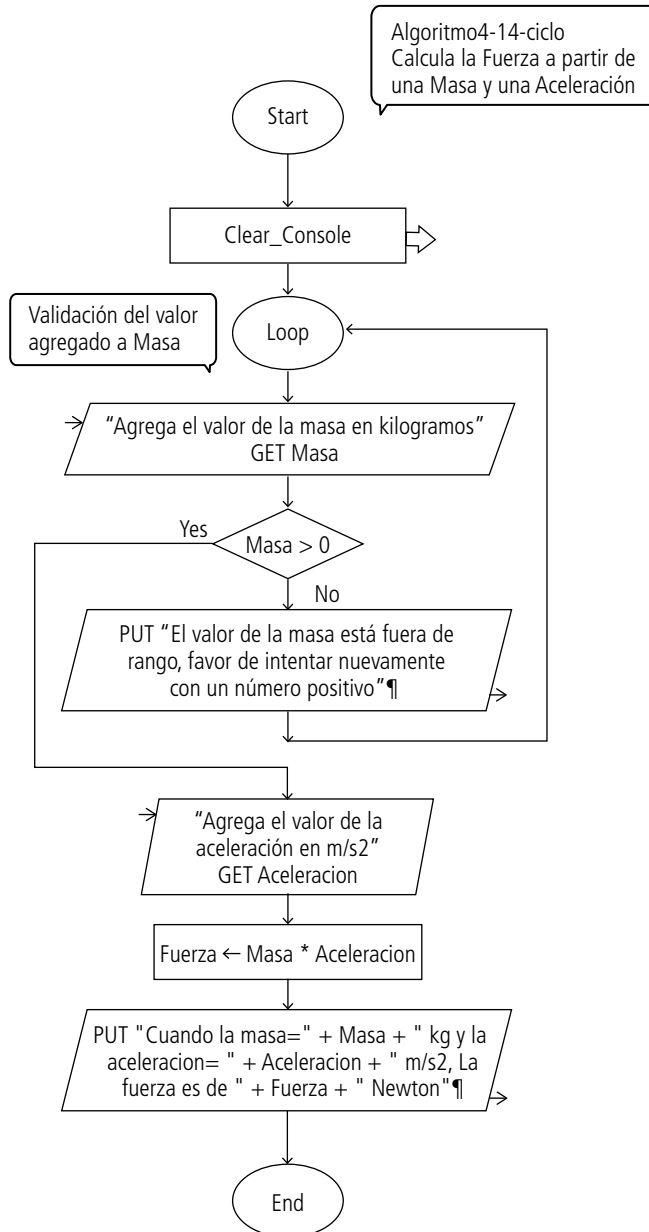


Figura 4.17 Algoritmo4-14-ciclo.

En el **Algoritmo4-15-ciclo** se calcula el área de un cuadrado teniendo como dato conocido la longitud de su lado. Observe que la captura del valor del lado está validada (véase figura 4.18).

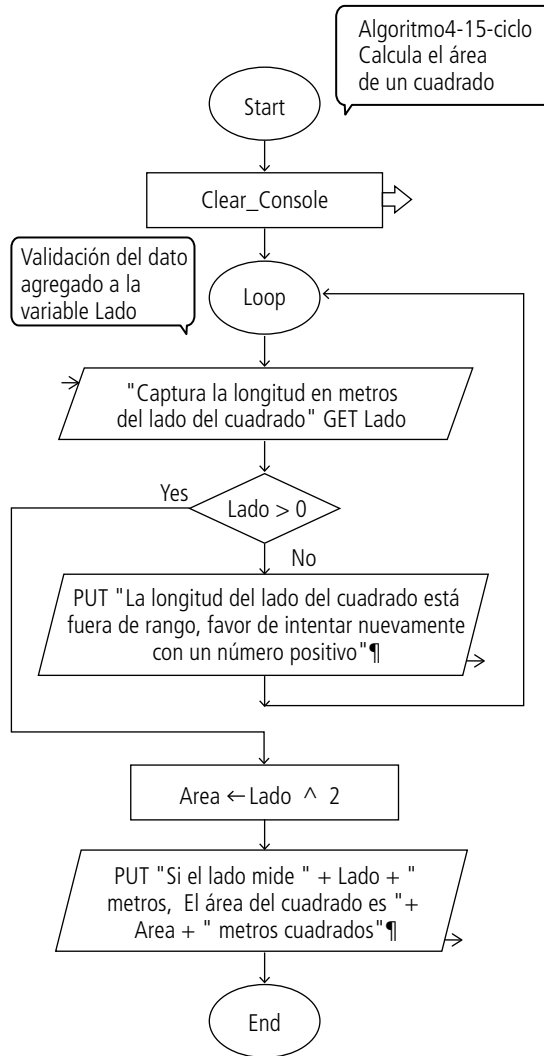


Figura 4.18 Algoritmo4-15-ciclo, validación del lado de un cuadrado.

En **Algoritmo4-16-ciclo** se calcula el área de un círculo teniendo como dato conocido la longitud de su diámetro. Observe que la captura del valor del diámetro está validada (véase figura 4.19).

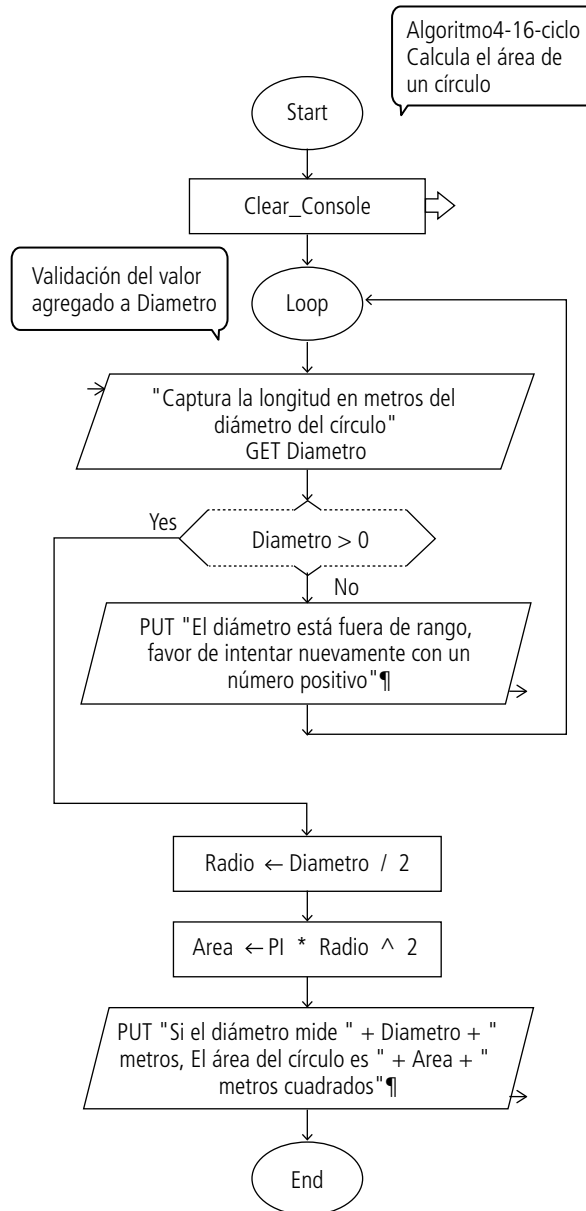


Figura 4.19 Algoritmo4-16-ciclo, validación del diámetro de un círculo.

En el **Algoritmo4-17-ciclo** se calcula el área de un triángulo teniendo como datos conocidos la longitud de la base y la altura. Observe que la captura de ambos valores está validada (véase figura 4.20).

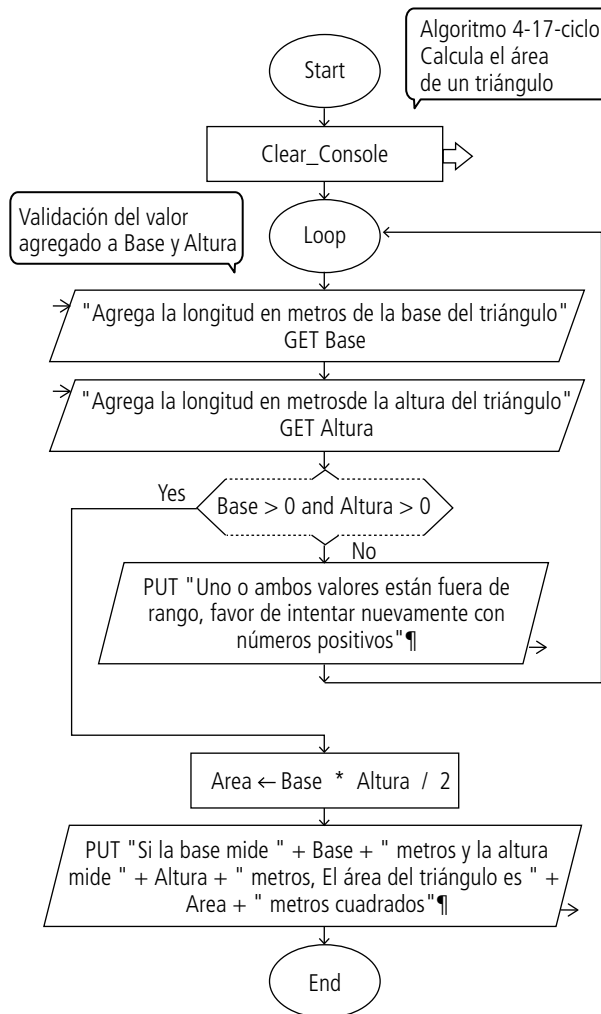


Figura 4.20 Algoritmo4-17-ciclo, validación de base y altura de un triángulo.



4.5 Ciclos para el manejo de un conjunto de datos similares

Muchos son los casos en que se requiere que la repetición del algoritmo sea automática, esto se debe a que existen varios datos semejantes que deben ser procesados. En los siguientes ejemplos encontrará esta situación. El **Algoritmo4-18-ciclo** suma exclusivamente números positivos. Observe que hay un Loop dentro de otro Loop. El Loop externo es un ciclo controlado por contador y el Loop interno es un ciclo controlado por centinela (véase figura 4.21).

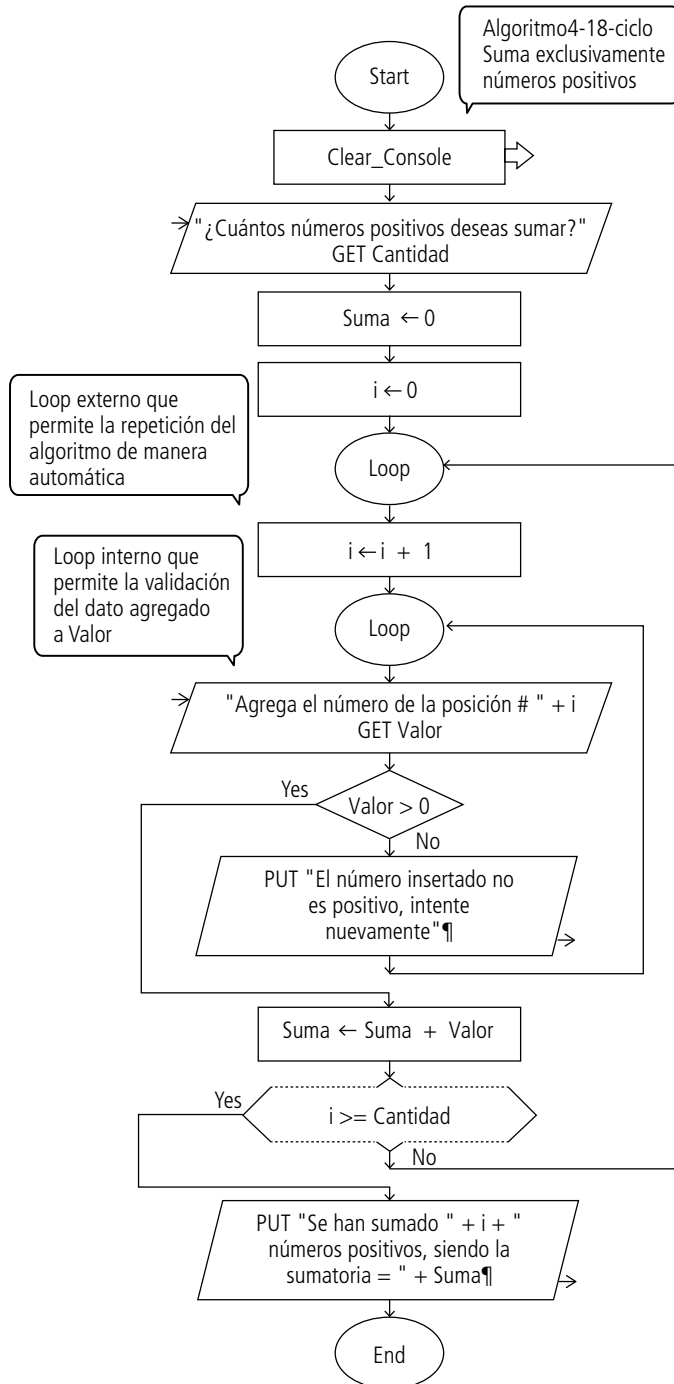


Figura 4.21 Algoritmo4-18-ciclo.

Para el **Algoritmo4-19-ciclo** considere que tenemos una cantidad indeterminada de temperaturas en grados Fahrenheit que deseamos convertir a grados centígrados. Éste es un caso clásico de un ciclo controlado por centinela, donde el valor de la variable que controla el ciclo puede cambiar de estado según la voluntad del usuario, utilizando el teclado de la computadora. El algoritmo quedaría de la siguiente forma (véase figura 4.22):

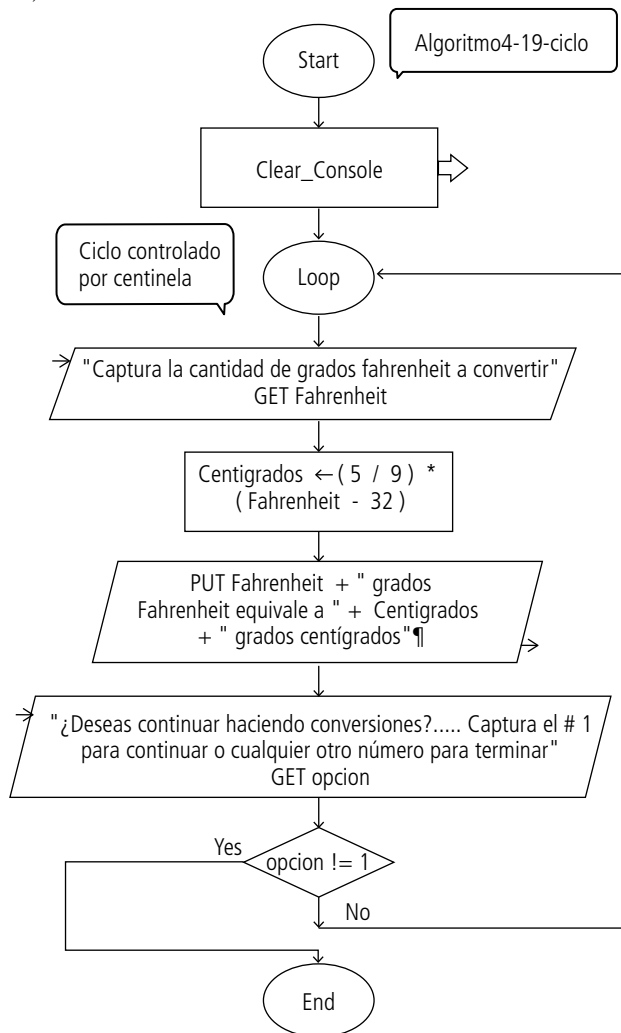


Figura 4.22 Algoritmo4-19-ciclo.

En el **Algoritmo4-20-selectiva** se presenta el mismo planteamiento del problema del diagrama **Algoritmo3-24-ciclo**, pero ahora tenemos una cantidad predeterminada de recibos de luz que procesar. Éste es un ciclo controlado por contador (véase figura 4.23).

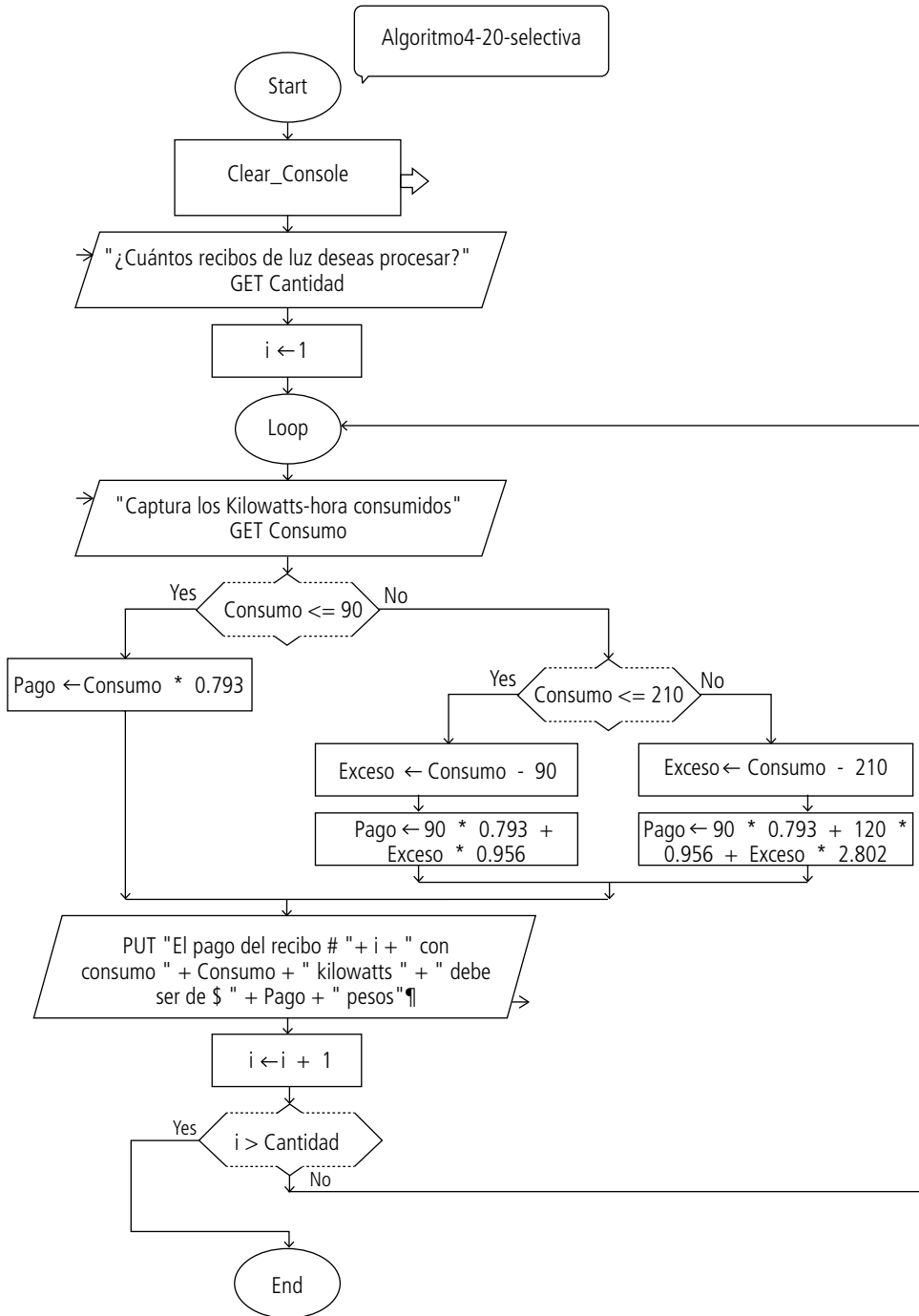


Figura 4.23 Algoritmo4-20-ciclo.



4.6 Sucesiones numéricas

Un tema interesante y recurrente en programación con ciclos es el de las sucesiones de números; por ejemplo, la serie de los números pares o la de los números impares. Analicemos sus particularidades.

Las sucesiones numéricas con frecuencia se representan mediante fórmulas que permiten generar la sucesión. Estas fórmulas se clasifican en **explícitas** y **recursivas**. Las primeras son fórmulas que permiten obtener cualquier término de la sucesión numérica de forma directa; por ejemplo, la sucesión de los números pares 2, 4, 6, 8, 10,..., tiene una fórmula muy conocida que es $T_i = 2 * i$ para $i = 1, 2, 3, \dots$, donde T indica un término de la sucesión e i , el lugar o la posición ordinal del término (véase figura 4.24):

2,	4,	6,	8,...	134,	$2n$
\nearrow	\nearrow	\nearrow	\nearrow	\nearrow	\nearrow
T_1	T_2	T_3	T_4	T_{67}	T_n

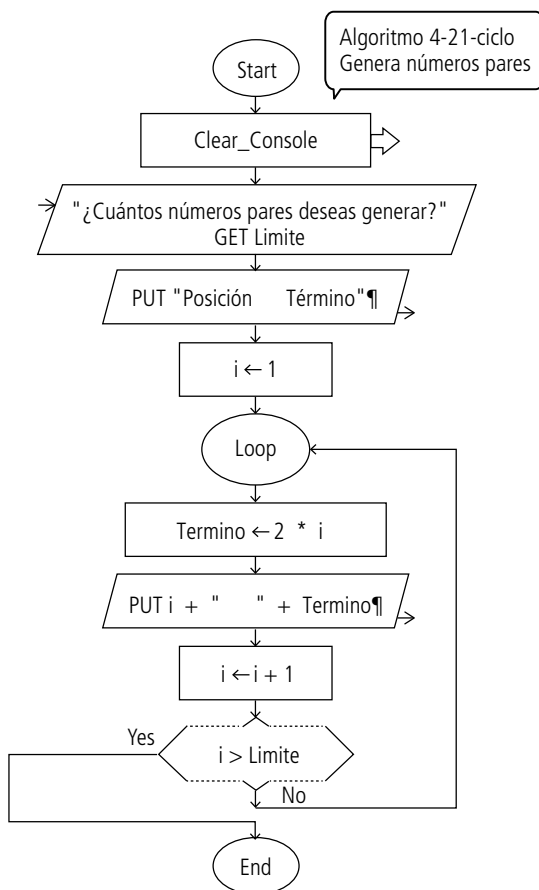


Figura 4.24 Algoritmo4-21-ciclo, genera la sucesión numérica de pares con fórmula explícita.

La sucesión de los números impares: 1, 3, 5, 7, 9,..., tiene una fórmula que también es muy conocida, y es $T_i = 2*i - 1$ para $i = 1, 2, 3, \dots$, donde T indica un término de la sucesión e i , el lugar o la posición ordinal del término (véase figura 4.25):

1,	3,	5,	7,...	133,	$2n - 1$
\nearrow	\nearrow	\nearrow	\nearrow	\nearrow	\nearrow
T_1	T_2	T_3	T_4	T_{67}	T_n

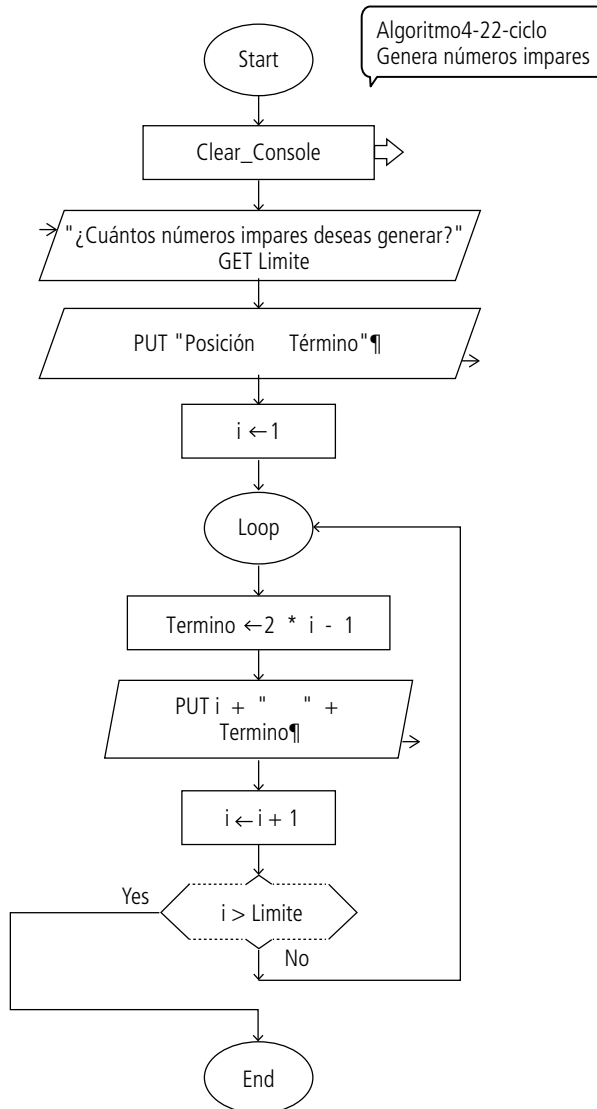


Figura 4.25 Algoritmo4-22-ciclo, genera la sucesión numérica de impares con fórmula explícita.

La sucesión de los cuadrados de los números a partir de uno es fácil de conseguir, ya que es $T_i = i * i$ para $i = 1, 2, 3, \dots$, (que es una fórmula explícita) (véase figura 4.26), aunque también se puede generar con la fórmula $T_i = T_{i-1} + (2 * i - 1)$ para $i = 2, 3, 4, \dots$, con $T_1 = 1$, donde T_{i-1} hace referencia al valor anterior (y esta fórmula es recursiva) (véase figura 4.27). A continuación se muestra un ejemplo de la sucesión:

1,	4,	9,	16,...	625,	$n * n$
↗	↗	↗	↗	↗	↗
T_1	T_2	T_3	T_4	T_{25}	T_n

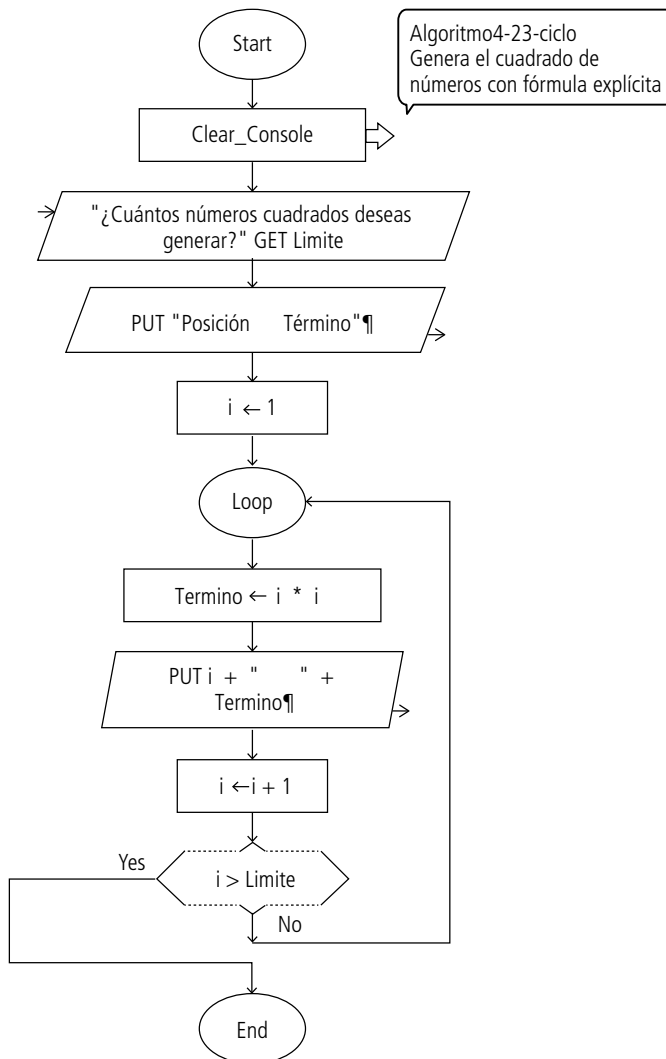


Figura 4.26 Algoritmo4-23-ciclo, genera la secuencia de números cuadrados con fórmula explícita.

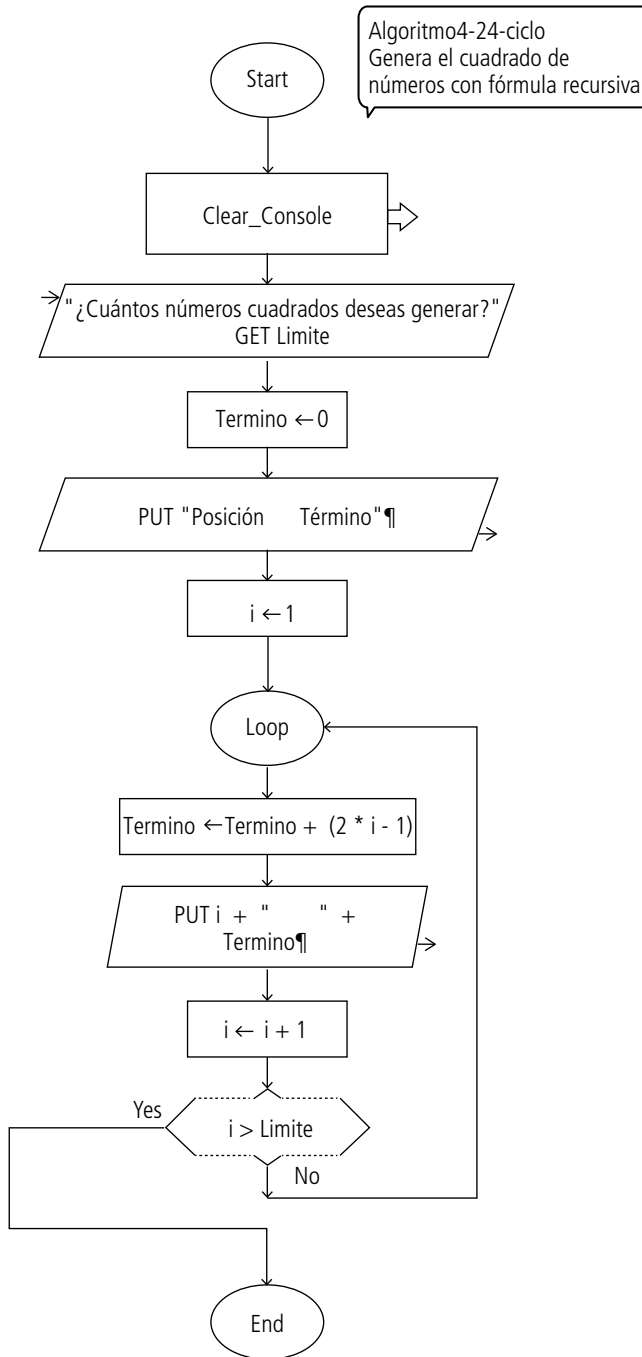


Figura 4.27 Algoritmo4-24-ciclo, genera la secuencia de números cuadrados de manera recursiva.



4.7 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 4.1.

- 4.1 ¿Cómo se denomina la acción computacional con la que se puede configurar una opción repetitiva y una opción diferida que NO son mutuamente excluyentes?
- 4.2 ¿Cuál es la opción del ciclo que consiste en que una o más acciones computacionales puedan ejecutarse en la misma secuencia y de manera iterativa bajo un ambiente controlado para que eventualmente deje de dar vueltas impidiendo generar ciclos infinitos?
- 4.3 ¿Cómo se denomina el elemento que permite la ejecución o no de la opción repetitiva de un ciclo?
- 4.4 ¿Cuál es la opción del ciclo que consiste en que la ejecución de una o más acciones computacionales se posterguen para después de que el ciclo haya llegado a su fin?
- 4.5 ¿Cuál es la lógica que permite la ejecución de la opción repetitiva del ciclo cuando la condición se ha evaluado como falsa?
- 4.6 ¿Cuál es la lógica que permite la ejecución de la opción repetitiva del ciclo cuando la condición se ha evaluado como verdadera?
- 4.7 ¿Cómo se llama el símbolo de Raptor que permite establecer un ciclo?
- 4.8 La opción No de Raptor permite que se ejecute la opción _____.
- 4.9 La opción Yes de Raptor permite que se ejecute la opción _____.
- 4.10 ¿Cómo se denomina el ciclo del que se conoce de antemano la cantidad de veces que dará vueltas y el cual requiere el manejo de tres parámetros: 1) el inicio del ciclo, 2) una condición para terminar el ciclo y 3) un movimiento de la variable controladora que con cada vuelta se acerque al cumplimiento de la condición, permitiendo, eventualmente, la salida o terminación del ciclo?
- 4.11 ¿Cómo se denomina el identificador que por lo general está dentro de la opción repetitiva de un ciclo y que sufre un incremento o decremento constante con cada iteración del ciclo? _____
- 4.12 ¿Cómo se denomina el identificador que generalmente está dentro de la opción repetitiva de un ciclo y que sufre un incremento o decremento variable con cada iteración del ciclo?
- 4.13 Los identificadores que tienen la tarea de contar o acumular valores deben recibir un valor _____ antes de que comience la opción repetitiva.

- 4.14** ¿Cómo se denomina el ciclo del que no se sabe con precisión la cantidad de veces que dará de vueltas y el cual requiere el manejo de tres parámetros: 1) el inicio del ciclo, 2) una condición para terminar el ciclo y 3) una acción que permita modificar el estado de la variable controladora y la verificación en cada vuelta de la condición de control para que en algún momento el ciclo se pueda dar por concluido?
- 4.15** ¿Cómo se denomina la tarea de asegurarse de que a los identificadores se les agrega un valor que tiene el rango o forma correcta, lo que garantiza que el diagrama no generará resultados con errores o inconsistencias. Esta tarea envía un mensaje de advertencia al usuario de que ha incurrido en un error y le da oportunidad de corregir mediante la inserción de un nuevo valor?

Tabla 4.1 Respuestas a los ejercicios de autoevaluación

4.1 Ciclo o estructura repetitiva	4.2 Opción repetitiva	4.3 Expresión matemática condicional
4.4 Opción diferida	4.5 Hasta que	4.6 Mientras que
4.7 Loop	4.8 Repetitiva	4.9 Diferida
4.10 Ciclo controlado por contador	4.11 Contador	4.12 Acumulador
4.13 Inicial	4.14 Ciclo controlado por centinela	4.15 Validación



4.8 Problemas propuestos

- 4.1** Un rico comerciante italiano del siglo XIII, introdujo una serie de números conocidos hoy en día como números de Fibonacci. Los primeros 16 de ellos son: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610. Cada número de una secuencia de números de Fibonacci es la suma de los dos números que preceden inmediatamente al número considerado.

Esto es: $1 + 1 = 2$, $1 + 2 = 3$, $2 + 3 = 5$, $3 + 5 = 8$, $5 + 8 = 13$, $8 + 13 = 21$, etcétera

Esta secuencia tiene aplicaciones prácticas en botánica, teoría de redes eléctricas y otros campos. Realice el programa que determine los primeros 50 números de la serie de Fibonacci.

- 4.2** Realice un programa que calcule la sumatoria de la siguiente serie: $c = \sum a_i * b_i = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + a_4 * b_4$, desde que $i = 1$ hasta que $i = 4$.

- 4.3** Una dama de edad avanzada desea pintar el piso del kiosco sin desperdiciar nada de pintura. Ella sabe por experiencia que se necesita un cuarto de pintura para cubrir 37 pies cuadrados de área. Si el piso del kiosco tiene diez pies de diámetro, ¿qué cantidad de pintura debería comprar? Realice un programa que resuelva el anterior problema.
- 4.4** Realice un programa que calcule los valores desde que $N = 1$ hasta $N = 20$ en incrementos de uno e imprima encabezados de lo siguiente:

$$N \quad N^2 \quad \sqrt{N} \quad \sqrt{10N} \quad N^3 \quad \sqrt[3]{N}$$

- 4.5** Realice un programa que calcule la depreciación de un automóvil e imprima los resultados según se muestran en la tabla que se presenta a continuación.

Costo = \$230,000

Vida útil = 6

Valor de rescate = \$160,000

Tabla 4.2 Depreciación de un vehículo

Año	Depreciación anual	Depreciación acumulada	Valor de rescate anual
1	\$11,666.67	\$11,666.67	\$218,333.33
2	\$11,666.67	\$23,333.33	\$206,666.67
3	\$11,666.67	\$35,000.00	\$195,000.00
4	\$11,666.67	\$46,666.67	\$183,333.33
5	\$11,666.67	\$58,333.33	\$171,666.67
6	\$11,666.67	\$70,000.00	\$160,000.00

La fórmula de la depreciación anual para cada año de vida útil del automóvil:

$$\text{Depreciación} = \frac{\text{Costo} - \text{Valor de rescate}}{\text{Vida útil}}$$

- 4.6** En la fábrica de sillas Bibliomodel el pago a sus empleados está basado en una tarifa diaria más un incentivo, éste depende del número de sillas producidas durante el día. Calcule el salario de cada empleado. Por ejemplo, a un salario básico de 3.50 dólares por hora y con 0.60 centavos de incentivo por cada

silla producida por encima de 50 unidades, un empleado que ensamble 76 sillas recibirá:

$$(3.50)(8) + (76 - 50)(0.60) = 28.00 + 15.60 = 43.60 \text{ dólares}$$

- 4.7** Dado un conjunto de números (mínimo 20, máximo 50), realice un programa que:
- Sume los números dados.
 - Muestre cuál es el número menor.
 - Muestre cuál es el número mayor.
 - Muestre la cantidad de valores iguales que 25.
 - Muestre la cantidad de valores mayores que 20 y menores que 70.
- 4.8** Suponga que ha invertido \$1,000 en una caja popular, en el entendido de que la junta directiva le ha asegurado que duplicará su inversión cada dos años. Elabore un programa que calcule su inversión cada dos años. El programa contempla una tabla semejante a la siguiente:

Tabla 4.3 Proyección del capital bianual

Tiempo	Monto
2 años	\$2,000
4 años	\$4,000
6 años	\$8,000
8 años	\$16,000
10 años	\$32,000
12 años	\$64,000
14 años	\$128,000

- 4.9** Realice un programa dirigido a los niños de educación primaria. El programa deberá hacer diez preguntas sencillas de historia, biología o matemáticas. Asimismo, en cada pregunta se deberán establecer por lo menos cuatro opciones de respuesta, y si se agrega una opción inválida el programa deberá advertir al niño para que escoja sólo entre las opciones ofrecidas. Después de cada respuesta, el programa deberá mostrar un registro de la cantidad de preguntas contestadas correctamente, de las incorrectas y de

las que faltan por contestar. Al terminar deberá imprimir una calificación.

- 4.10** El número de sonidos emitidos por un grillo en un minuto es una función de la temperatura. Como resultado de esto, es posible determinar el nivel de temperatura utilizando el grillo como termómetro. La fórmula de la función es: $t = (N/4) + 10$, donde t representa la temperatura en grados Fahrenheit y n representa el número de sonidos emitidos en un minuto. Elabore un programa que determine e imprima los valores para t cuando n toma los valores de 40, 50, 60, 70,..., 140, 150.
- 4.11** El tiempo que requiere un satélite para dar una revolución completa alrededor de la Tierra y a una determinada altura es una función de su velocidad. La fórmula para una altitud de 100 millas es: $t = 1.540/s$, donde t es el tiempo y s es la velocidad del satélite en miles de millas por hora. Elabore un programa que calcule e imprima t para los valores de $s = 18, 19, 20, \dots 24$.
- 4.12** Si el algoritmo se repite, solicite al usuario que agregue dos valores numéricos cualquiera y haga las cuatro operaciones aritméticas básicas desplegando el valor de cada una de ellas.
- 4.13** Elabore el algoritmo que pide el nombre del estudiante y dos calificaciones, luego calcule el promedio de las calificaciones; en pantalla, muestre el nombre del estudiante y el promedio que obtuvo. Permita que el algoritmo pueda repetirse para un número indeterminado de operaciones.
- 4.14** Algoritmo para una tienda de abarrotes. El algoritmo nos permite introducir los precios de los artículos que lleva un cliente y al final muestra el total a pagar y la cantidad de artículos vendidos.
- 4.15** Elabore el algoritmo que permita introducir un predeterminado número de calificaciones de un grupo de estudiantes y que las vaya sumando para calcular el promedio del grupo. También debe calcular y mostrar en pantalla cuántos están aprobados y cuántos reprobados. La mínima aprobatoria es 70.
- 4.16** Hacer un algoritmo que sume los predeterminados números naturales (enteros) que se encuentren entre dos valores límite (límite inicial y límite final, donde límite inicial < límite final). Mejore el algoritmo verificando que el límite inicial es menor que el límite final.
- 4.17** Hacer un algoritmo que sume los primeros N múltiplos de 3.

- 4.18** Hacer un algoritmo que sume los primeros N múltiplos de cualquier número.
- 4.19** Elabore el algoritmo que le permita obtener los mismos resultados que el protagonista de la anécdota de la suma del 1 al 100. El algoritmo debe funcionar para obtener la suma entre cualquier par de números que sean inicio y final de la suma de números. Estos límites se circunscriben a dos casos: 1) el primer número debe ser impar y el segundo, par; 2) el primer número debe ser par y el segundo, impar. **Pista:** Puede usar una hoja de cálculo para hacer una simulación con dos listas de números para revisar el resultado de ambas y le servirá de guía para hacer el algoritmo.
- 4.20** Modifique el algoritmo que permite saber si un número es primo o no y cuente la cantidad de divisores que tiene el número.
- 4.21** Hacer un programa que imprima las tablas de multiplicar del 1 al 9 en el orden de izquierda a derecha y de arriba hacia abajo.

Tabla del 1	Tabla del 4	Tabla del 7
Tabla del 2	Tabla del 5	Tabla del 8
Tabla del 3	Tabla del 6	Tabla del 9

- 4.22** Hacer un programa que imprima las tablas de multiplicar del 1 al 9 en el orden de izquierda a derecha y de arriba hacia abajo.

Tabla del 8	Tabla del 9	Tabla del 7
Tabla del 5	Tabla del 6	Tabla del 4
Tabla del 2	Tabla del 3	Tabla del 1

- 4.23** Elabore los algoritmos para hacer e imprimir una cantidad pre-determinada de términos de las siguientes sucesiones numéricas.

3, 7, 11, 15, 19, 23,...

3, 6, 5, 8, 7, 10, 9,...

1, 4, 1, 8, 3, 12, 5, 16, 7, 20,...

3, 6, 9, 12, 15, 36, 21, 24, 81, 30,...

Pista: Tome $T_1 = 3$

1, 1, 2, 3, 5, 8, 13, 21,... Es la serie de Fibonacci.

Pista: Tome $T_1 = 1$, $T_2 = 1$.

- 4.24** Haga un algoritmo que calcule y muestre en pantalla la potencia n -ésima de cualquier número.

- 4.25** Realice el algoritmo que permita sumar de manera algebraica los N términos siguientes:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \dots$$

Primero, use la división modular. Después, utilice una bandera para cambiar de operación.

Pista: El primer término puede tomarse como $\frac{1}{1}$.

- 4.26** Realice el algoritmo que permita sumar los N términos siguientes:

$$1 + \frac{1}{2} + \frac{2}{3} - \frac{3}{4} + \frac{4}{5} + \frac{5}{6} + \frac{6}{7} - \frac{7}{8} \dots$$

Pista: El primer término puede tomarse como $\frac{1}{1}$.

- 4.27** Calcule el algoritmo $\ln(1 + x)$:

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots + (-1)^{n-1} \frac{x^n}{n} + \dots, -1 < x \leq 1$$

Pista: Con $x = 0.8$ y 15 términos, el resultado es 0.5887909192.

- 4.28** Haga el algoritmo para calcular e^x :

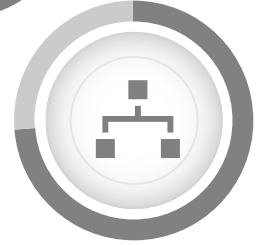
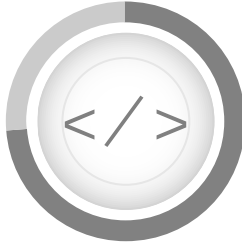
$$e^x = 1 + \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \forall x; n \in \mathbb{N}_0$$

Pista: Con $x = 3$ y 20 términos, el resultado debe ser 20.085539.

- 4.29** Haga un algoritmo que calcule el valor de $W_0(x)$:

$$W_0(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} x^n = x - x^2 + \frac{3}{2}x^3 - \frac{8}{3}x^4 + \frac{125}{24}x^5 - \dots$$



Capítulo 5

Diseño de algoritmos por módulos

- 5.1 Introducción
- 5.2 Diseño de algoritmos por módulos sin paso de parámetros
- 5.3 Diseño de algoritmos por módulos con paso de parámetros
- 5.4 Ejercicios de autoevaluación
- 5.5 Problemas propuestos

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Diseñar algoritmos computacionales modularizados.
- Utilizar módulos sin paso de parámetros.
- Comprender el concepto de parámetros de un módulo.
- Utilizar módulos con paso de parámetros.



5.1 Introducción

El diseño de algoritmos por módulos consiste en dividir el algoritmo en tareas pequeñas que abarquen una o varias acciones computacionales y que en su conjunto realicen una función muy específica. A los algoritmos que no están diseñados por módulos, Deitel los llama monolíticos; y a los que sí lo están les llama funcionalizados. Hay suficiente literatura en la disciplina de la Ingeniería de Software que permite afirmar que cuando se construye un programa grande es mejor diseñarlo con algoritmos por módulos que con algoritmos monolíticos, debido a que facilita su desarrollo y mantenimiento.

Los algoritmos por módulos pueden implementarse de dos maneras: sin y con el paso de parámetros. En el caso de Raptor, los módulos que no manejan parámetros se denominan *Subchart*, y los que sí se llaman *Procedure*. Parámetro es el valor de las variables y constantes que se transmiten de un módulo hacia otro.

A los módulos se les debe asignar un identificador o nombre para distinguirlos unos de otros. Las reglas de calidad para dar nombre a los módulos de Raptor son las siguientes (véase tabla 5.1):

1. Otorgue a los módulos un nombre significativo, es decir, que el nombre ayude a deducir qué es lo que hace el módulo o cuál es la función que va a realizar.
2. Evite nombres de módulos con una sola letra o abreviaciones.
3. Comience los nombres de módulos con una letra. Utilice sólo letras mayúsculas. Cuando dos o más palabras describen mejor el módulo, utilice guión bajo entre cada palabra para ayudar a su visualización.
4. En la medida de lo posible, utilice verbos en infinitivo para las primeras palabras que forman parte del nombre del módulo y que describa la función que realizarán.
5. Si el nombre del módulo requiere un número, escríbalo continuo a las letras. Dé preferencia a usar el número al final del nombre.

Tabla 5.1 Ejemplo de nombres aceptados para módulos

CAPTURAR_REGISTROS	CALCULAR_SALARIO_NETO
CALCULAR_AREA	OBTENER_PRECIO_VENTA
CALCULAR_TOTAL	AGREGAR_OPCION
VALIDAR_CALIFICACION	IMPRIMIR_RECIBO



5.2 Diseño de algoritmos por módulos sin paso de parámetros

Hasta este momento, hemos diseñado diagramas de flujo monolíticos, pues sólo tienen acciones computacionales en la pestaña principal llamada *main*. En Raptor, para diseñar diagramas de flujo funcionalizados, los programadores deben crear nuevas pestañas denominadas módulos, los cuales deben recibir un identificador o nombre que debe estar de acuerdo con las reglas establecidas tanto por Raptor como las de calidad. Las nuevas pestañas serán visibles a un lado de la pestaña *main*. Las pestañas recién creadas, de manera preestablecida, contarán con los globos de Start y End.

A cada pestaña genéricamente se le nombra módulo, pero Raptor le denomina *Subchart* a los módulos que no utilizan parámetros. Cada módulo debe contar con una o más acciones computacionales. Raptor reconoce que el diseño de diagramas con módulos *Subchart* es para personas que están aprendiendo a diseñar algoritmos. Vea el menú de Raptor MODE y su opción NOVICE. Además de los módulos *Subchart*, Raptor cuenta con los módulos *Procedure*, los cuales explicaremos más adelante.

La característica de los módulos *Subchart* es que el uso de parámetros es transparente y el diseñador no tiene que entender el concepto técnico de parámetro, el cual tiene un cierto grado de complejidad. Por el momento, el objetivo principal es explicar el tema de diseño modular para posteriormente abordar el tema de paso de parámetros.

En Raptor, las acciones computacionales que forman parte de un módulo *Subchart* se ejecutan con el símbolo Call; aunque anteriormente ya lo hemos usado con la función Clear_Console, su uso tiene una aplicación práctica más amplia.

Antes de ejecutar las acciones contenidas en un módulo, éste debe crearse. Por tanto, se debe considerar que una cosa es crear el módulo, otra diseñarlo y otra más pedir que se ejecuten sus acciones computacionales que contiene. En el siguiente ejemplo se explica paso a paso cómo hacer esto.

El ejemplo que a continuación veremos es un reconocimiento a un personaje de la historia que además de mostrarnos su fe en cosas inmateriales y la esperanza de un mejor futuro, se anticipó al futuro, pues nos invitó a hacer conciencia del combate al cambio climático. A Martin Luther King Jr., defensor de los derechos civiles de la comunidad afroamericana en los Estados Unidos, se le atribuye la siguiente frase: “Si supiera que el mundo se acaba mañana, no dudaría en plantar hoy, un árbol de manzanas”.

Para explicar el ejemplo que se pretende, la frase de Luther King se ha dividido en dos líneas:

1. “*Si supiera que el mundo se acaba mañana,*”.
2. “*no dudaría en plantar hoy, un árbol de manzanas*”.

Para esto, crearemos tres módulos que llevarán por nombre: LINEA1, LINEA2 y PERSONAJE.

El nombre del archivo para este diagrama es **Algoritmo5-1-módulos**. Para crear una pestaña de *Subchart* coloque el apuntador del ratón sobre la pestaña *main* y luego presione el botón derecho del ratón, con lo que aparecerá un menú auxiliar del cual seleccionaremos la opción *Add Subchart* (véase figura 5.1).

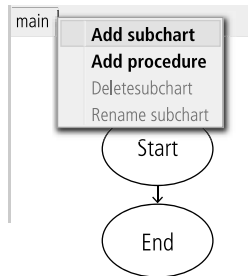


Figura 5.1 Menú auxiliar para crear una pestaña de *Subchart*.

Enseguida aparecerá la ventana *Name Subchart* y en la caja de texto escriba el nombre de la pestaña, en este caso, *LINEA1* (véase figura 5.2).

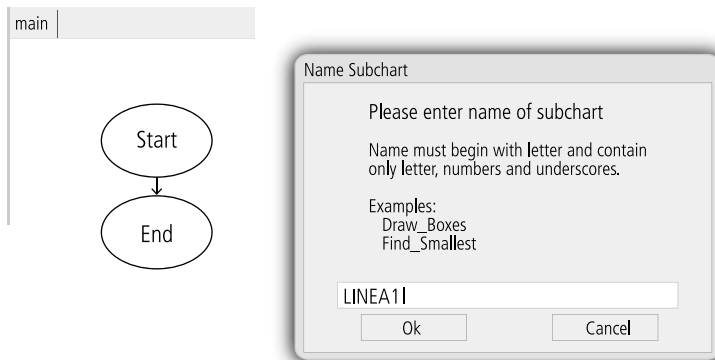


Figura 5.2 Ventana *Name Subchart*.

Después, lleve el apuntador del ratón sobre el botón *Ok* de la ventana *Name Subchart* y presione el botón izquierdo del ratón y verá la pestaña *LINEA1* al lado de la pestaña *main* (véase figura 5.3).

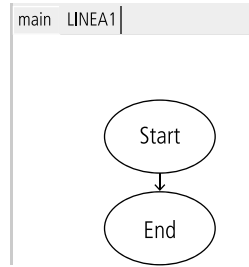



Figura 5.3 Pestaña *LINEA1*.

Repita los pasos de creación de *Subchart* para agregar las pestañas *LINEA2* y *PERSONAJE*, lo que le permitirá obtener cuatro pestañas que tienen los globos de *Start* y *End*.

Posteriormente, seleccione la pestaña de *main* para agregar cuatro símbolos *Call* entre los globos *Start* y *End*. Para ello, coloque el apuntador del ratón sobre el símbolo *Call* que está en la paleta de *Symbols* y presione el botón izquierdo del ratón, el contorno se pondrá en color rojo. Mueva el apuntador del ratón sobre la flecha de la pestaña *main* que está entre *Start* y *End* hasta que aparezca la manita  y presione el botón izquierdo. Repita otras tres veces y obtendrá la figura 5.4.

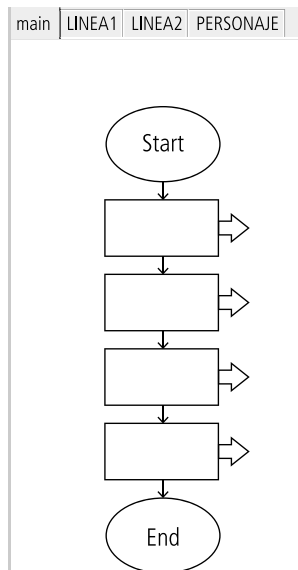


Figura 5.4 Pestaña *main*.

Una vez que ya tiene los cuatro símbolos *Call* en el *Subchart* de la pestaña *main*. Agregue la función *Clear_Console* en el primer *Call* y después los nombres de las

pestañas en los siguientes Call. Para ello, debe abrir la ventana *ENTER CALL* llevando el apuntador del ratón sobre cada uno de los *Call* y presionando dos veces el botón izquierdo del ratón; escriba el nombre de cada pestaña en la caja de texto. No olvide ir pulsando el botón *DONE* de la ventana *ENTER CALL* después de escribir en la caja de texto. Se recomienda agregar un comentario en el globo Start de la pestaña *main*, tal como se muestra en figura 5.5

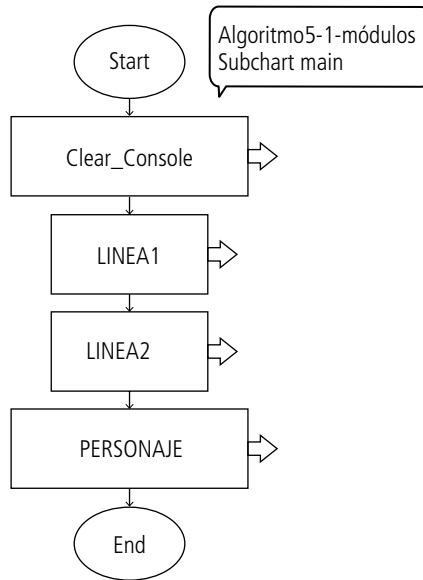


Figura 5.5 *Subchart main* del Algoritmo5-1-módulos.

Una vez que ha realizado los pasos anteriores, el *Subchart main* tiene cuatro *Call*, pero los *Subchart LINEA1*, *LINEA2* y *PERSONAJE* sólo tienen los globos de *Start* y *End*. Seleccionada las pestañas de cada *Subchart*, agregue a los símbolos que se muestran en los siguientes tres diagramas (véanse figuras 5.6 a 5.8).

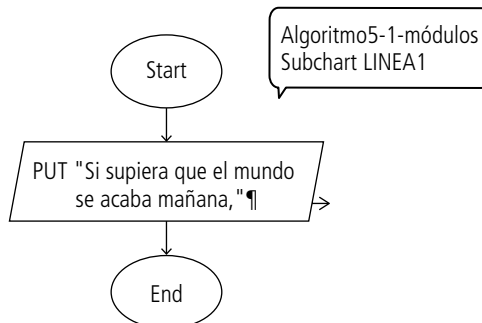


Figura 5.6 Algoritmo5-1-módulos *Subchart LINEA1*.

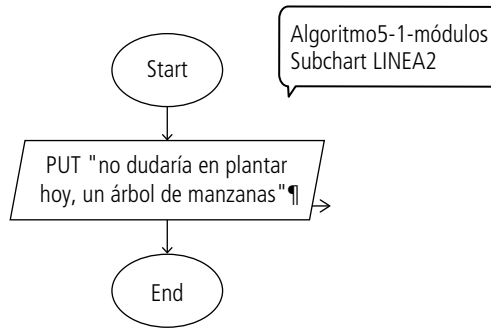


Figura 5.7 Algoritmo5-1-módulos *Subchart LINEA2*.

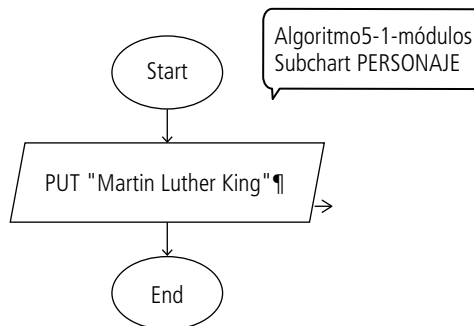


Figura 5.8 Algoritmo5-1-módulos *Subchart PERSONAJE*.

Una vez que han concluido los pasos anteriores, ejecute el diagrama para que vea el resultado en *MasterConsole* (véase figura 5.6)

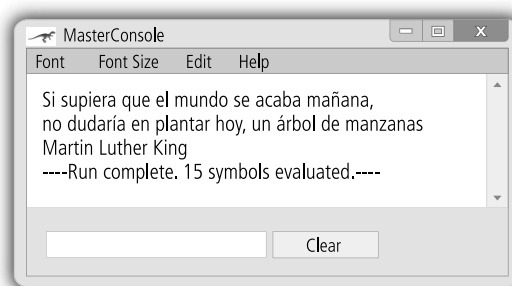


Figura 5.9 *MasterConsole* Algoritmo5-1-módulos.

5.2.1 Aplicación del enfoque de sistemas al diseño modular de algoritmos

Como ya se mencionó, el diseño modular consiste en dividir el algoritmo en tareas pequeñas que abarquen una o varias acciones computacionales y que en su conjunto realicen una función específica. Para las personas que van a empezar a aprender a programar por módulos, pueden comenzar con visualizar las tareas basándose en el enfoque de sistemas donde se distinguen tres fases:

1. Entrada de datos.
2. Proceso de datos.
3. Salida de resultados.

Estas fases serán nuestros módulos.

El módulo de entrada de datos abarca su asignación o inserción a la memoria RAM. El módulo de proceso considera los datos conocidos para obtener los datos que se desean conocer, lo cual se logra con la aplicación de alguna fórmula matemática. El módulo de salida de resultados permite que el usuario observe los datos que se han obtenido o calculado usando los datos conocidos, los cuales también pueden formar parte de la visualización de los resultados.

A continuación presentamos algunos ejemplos sencillos como el hecho de imprimir la edad de una persona. El **Algoritmo5-2-módulos** consta de módulos de ENTRADA y SALIDA. En este caso, el módulo del proceso de datos se omite porque no se hará ningún cálculo matemático.

En el *Subchart main* se ejecutan los módulos ENTRADA y SALIDA. En el *Subchart ENTRADA* se le asigna un valor a la variable Edad y en el *Subchart SALIDA* se imprime el valor de la variable Edad en un mensaje compuesto (véanse figuras 5.10 a 5.12).

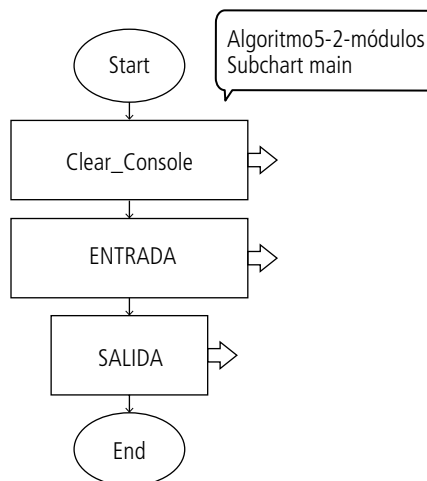


Figura 5.10 Algoritmo5-2-módulos *Subchart main*.

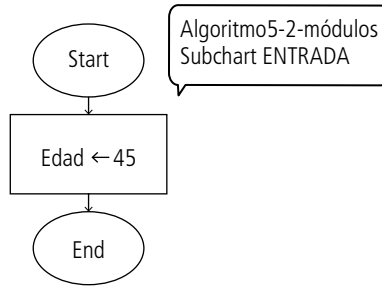


Figura 5.11 Algoritmo5-2-módulos *Subchart* ENTRADA.

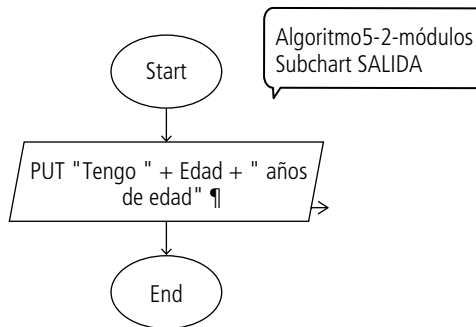
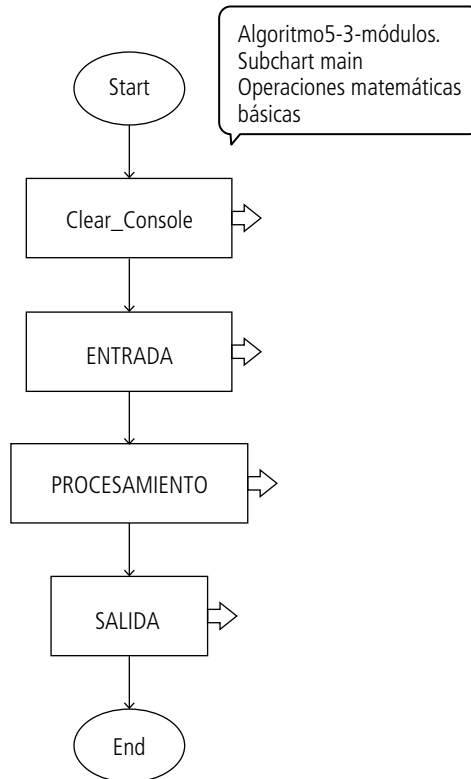
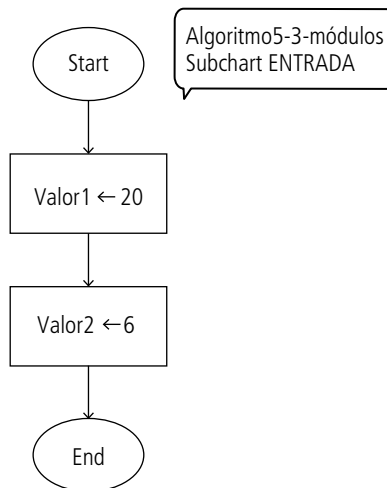


Figura 5.12 Algoritmo5-2-módulos *Subchart* SALIDA.

El diagrama del **Algoritmo5-3-módulos** tiene tres módulos: ENTRADA, PROCESAMIENTO y SALIDA. En el *Subchart main* se manda a ejecutar a los tres módulos. En el *Subchart* ENTRADA se asignan valores numéricos a las variables Valor1 y Valor2. En el *Subchart* PROCESAMIENTO se hacen cinco operaciones matemáticas y en el *Subchart* SALIDA se imprime el resultado (véanse figuras 5.13 a 5.16).

El **Algoritmo5-4-módulos** tiene tres módulos: ENTRADA, PROCESAMIENTO y SALIDA. En el *Subchart main* se manda ejecutar los tres módulos. En el *Subchart* ENTRADA, se pide al usuario que capture valores para las variables Masa y Aceleracion. En el *Subchart* PROCESAMIENTO se ejecuta la multiplicación de los valores capturados para obtener la Fuerza y en el *Subchart* SALIDA se imprime el resultado.

Un inconveniente de este algoritmo es que no se validan los valores capturados para Masa y Aceleracion, lo cual parece un retroceso comparado con los ejemplos mostrados en capítulos anteriores. Sin embargo, esta deficiencia se subsana más adelante en este mismo capítulo. El propósito de mostrar este ejemplo tal y como se expone aquí, es centrarnos en la explicación del diseño modular (véanse figuras 5.17 a 5.20).

**Figura 5.13** Algoritmo5-3-módulos *Subchart main*.**Figura 5.14** Algoritmo5-3-módulos *Subchart ENTRADA*.

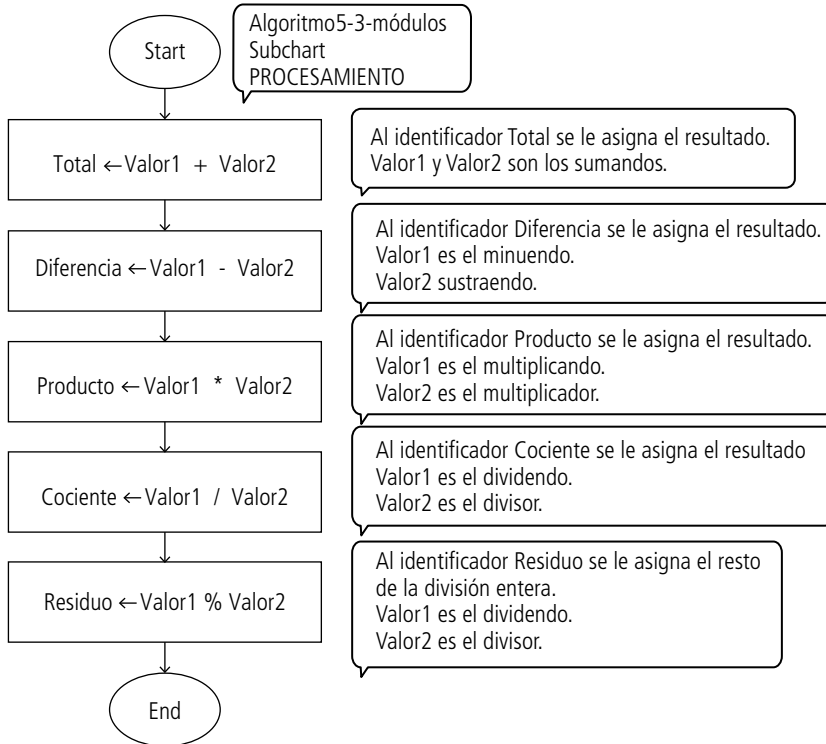


Figura 5.15 Algoritmo5-3-módulos Subchart PROCESAMIENTO.

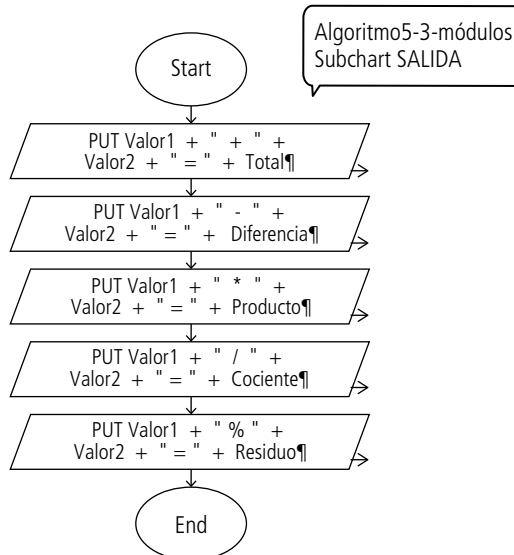


Figura 5.16 Algoritmo5-3-módulos Subchart SALIDA.

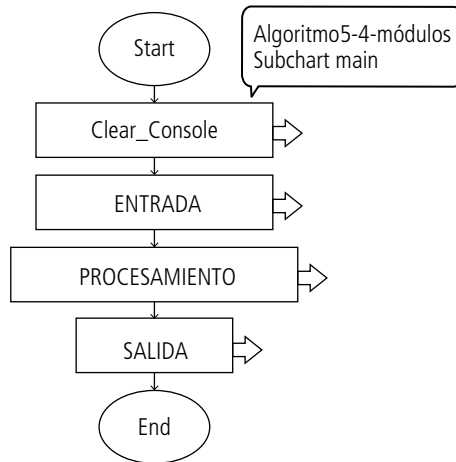


Figura 5.17 Algoritmo5-4-módulos *Subchart main*.

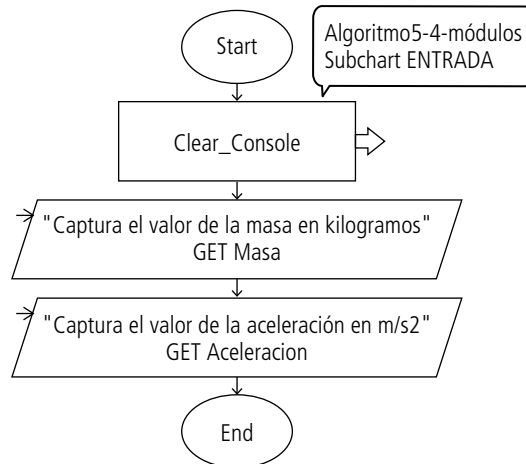


Figura 5.18 Algoritmo5-4-módulos *Subchart ENTRADA*.

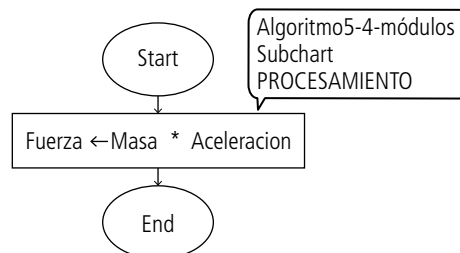


Figura 5.19 Algoritmo5-4-módulos *Subchart PROCESAMIENTO*.

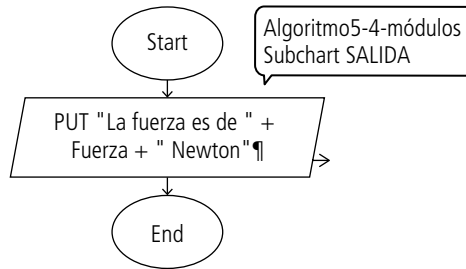


Figura 5.20 Algoritmo5-4-módulos *Subchart SALIDA*.

El diagrama del **Algoritmo5-5-módulos** tiene tres módulos: ENTRADA, PROCESAMIENTO y SALIDA. El *Subchart main* manda ejecutar los tres módulos. Primero, en el *Subchart ENTRADA* se le pide al usuario que capture un valor numérico distinto de cero. Si se agrega un dato que no cumple con los requisitos, el diagrama indica al usuario que lo corrija y lo intente con un nuevo valor. Luego, en el *Subchart PROCESAMIENTO* se hace la asignación de un valor, de los dos posibles, a la variable **Testigo**; esto depende del resultado de la condición establecida en el menú SELECTION. Por último, en el *Subchart SALIDA*, dependiendo del valor almacenado en Testigo, se imprime si el valor capturado por el usuario es positivo o negativo.

En el diseño de algoritmos computacionales es común utilizar variables que se reservan para que puedan tomar dos valores, los cuales denotan los estados de verdadero o falso. A estas variables se les denomina booleanas. Se dice que si el valor que toman es UNO, el estado indica VERDAD. Por el contrario, si la variable toma el valor de CERO, el estado indica FALSEDAD. En este caso, si la variable Testigo toma el valor de UNO significa que el valor capturado es positivo. Por el contrario, si la variable Testigo toma el valor de CERO quiere decir que el valor capturado es negativo (véanse figuras 5.21 a 5.24).

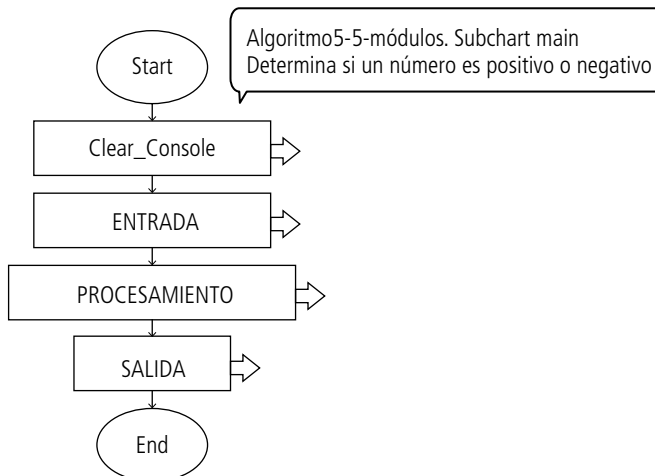


Figura 5.21 Algoritmo5-5-módulos *Subchart main*.

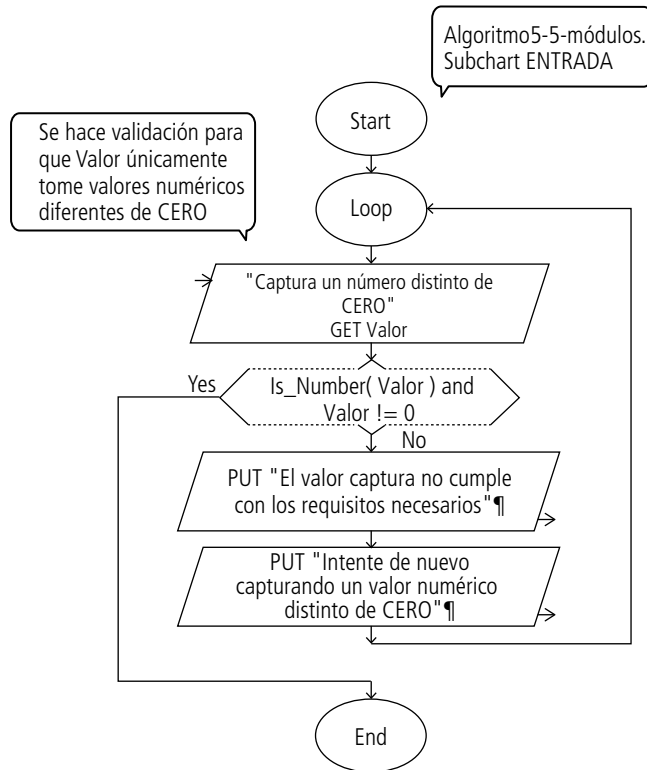


Figura 5.22 Algoritmo5-5-módulos Subchart ENTRADA.

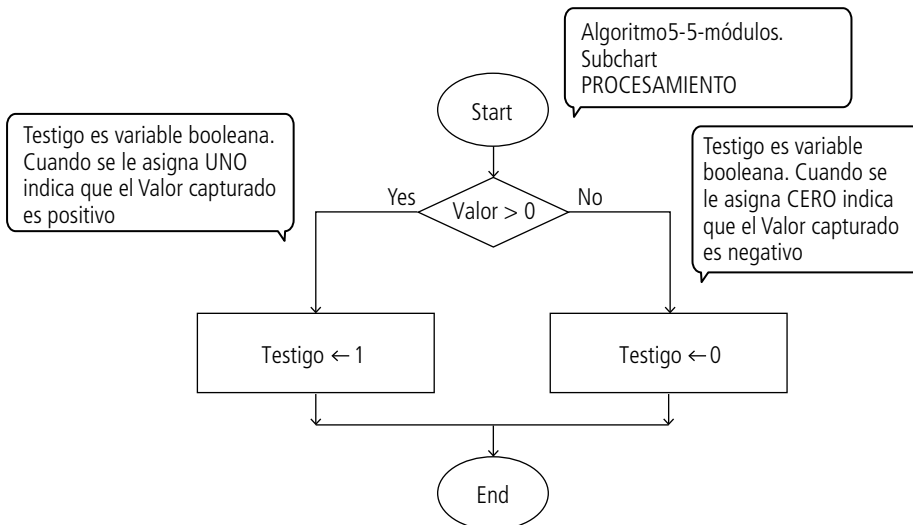


Figura 5.23 Algoritmo5-5-módulos Subchart PROCESAMIENTO.

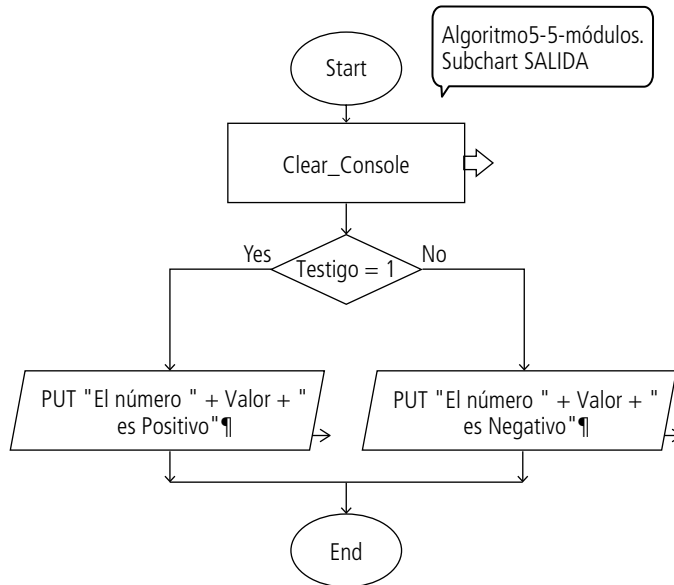


Figura 5.24 Algoritmo5-5-módulos *Subchart* SALIDA.

5.2.2 Refinación del enfoque de sistemas en el diseño modular de algoritmos

Se sugiere refinar el enfoque de sistemas en lo que respecta a las variables a capturar. Es conveniente crear un módulo por cada variable que se capture porque permite validar los valores agregados.

Considere que conocemos la matrícula, el nombre y dos calificaciones de un estudiante y queremos calcular su promedio y si aprobó o no la materia. En el diagrama del **Algoritmo5-6-módulos** puede observar que por cada dato a capturar se ha creado un módulo; por tanto, tenemos siete *Subchart*, incluyendo a *main*, y son:

1. CAPTURAR_MATRICULA
2. CAPTURAR_NOMBRE
3. CAPTURAR_CALIFICACION1
4. CAPTURAR_CALIFICACION2
5. PROCESAMIENTO
6. SALIDA

En el *Subchart main* se manda a ejecutar todos módulos que son parte de este algoritmo.

En el *Subchart* CAPTURAR_MATRICULA se le pide al usuario que capture la matrícula del estudiante, la cual debe ser numérica y positiva. Si no se cumple con los requisitos, se le pide al usuario que vuelva a intentar con otro valor. En el *Subchart* CAPTURAR_NOMBRE se le pide al usuario que capture con letras mayúsculas el nombre del estudiante.

En los módulos de CAPTURAR_CALIFICACION1 y CAPTURAR_CALIFICACION2, se pide al usuario que capture las calificaciones que se van a promediar, las cuales deben ser numéricas y estar entre el rango de cero a cien. Si la calificación está fuera de rango, se pide al usuario volver a intentarlo. En el *Subchart* PROCESAMIENTO se determina el promedio de las dos calificaciones y se hace una asignación a la variable Situacion, según el resultado de la condición. Finalmente, en el *Subchart* SALIDA se imprimen los datos del estudiante, promedio y situación (véanse figuras 5.25 a 5.31).

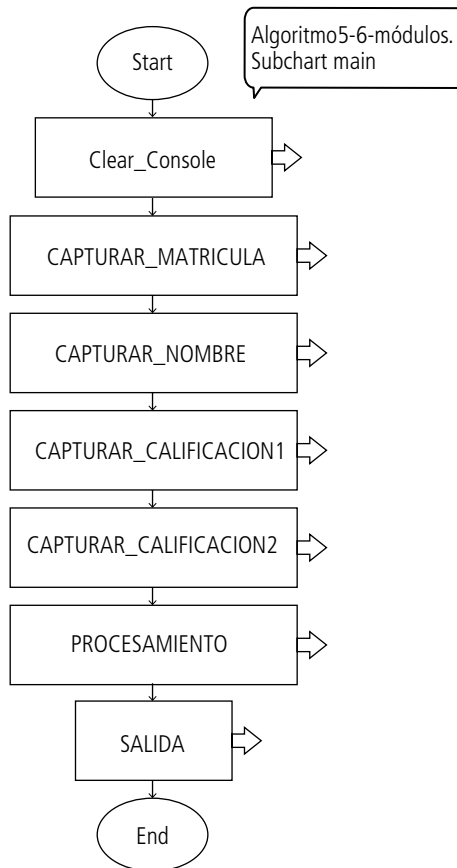


Figura 5.25 Algoritmo5-6-módulos *Subchart main*.

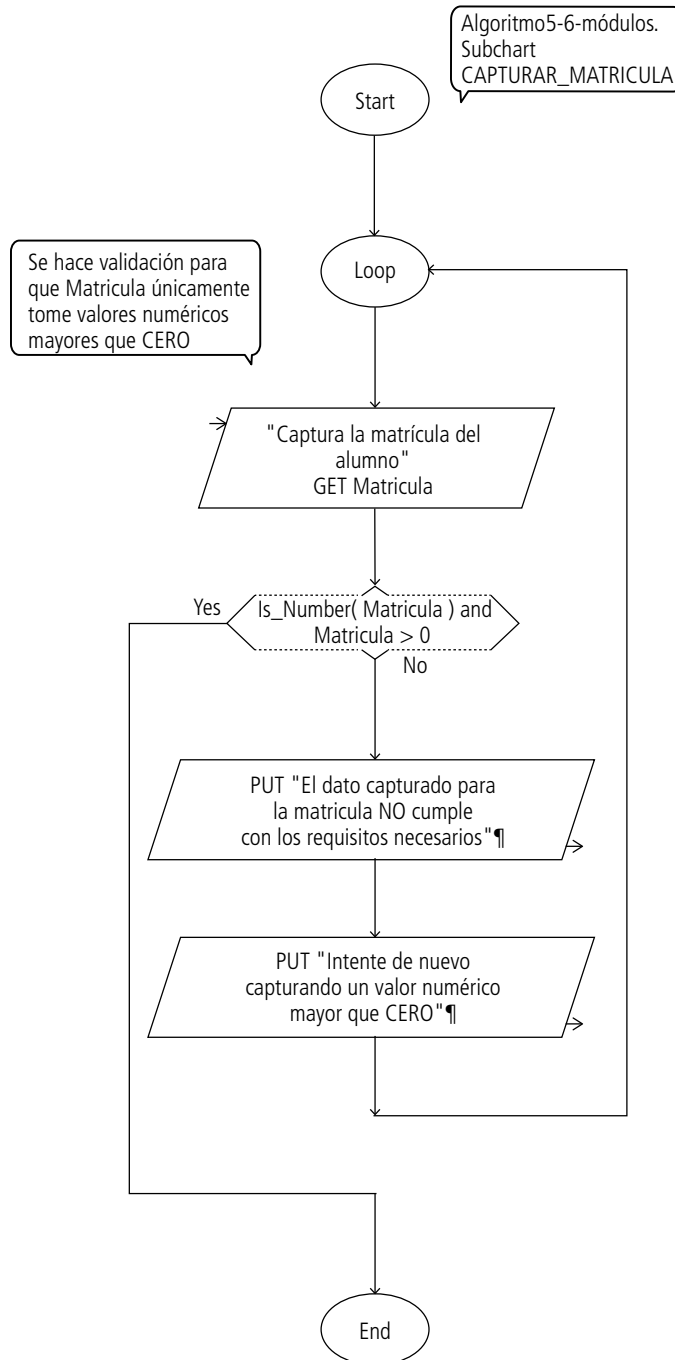


Figura 5.26 Algoritmo5-6-módulos Subchart CAPTURAR_MATRICULA.

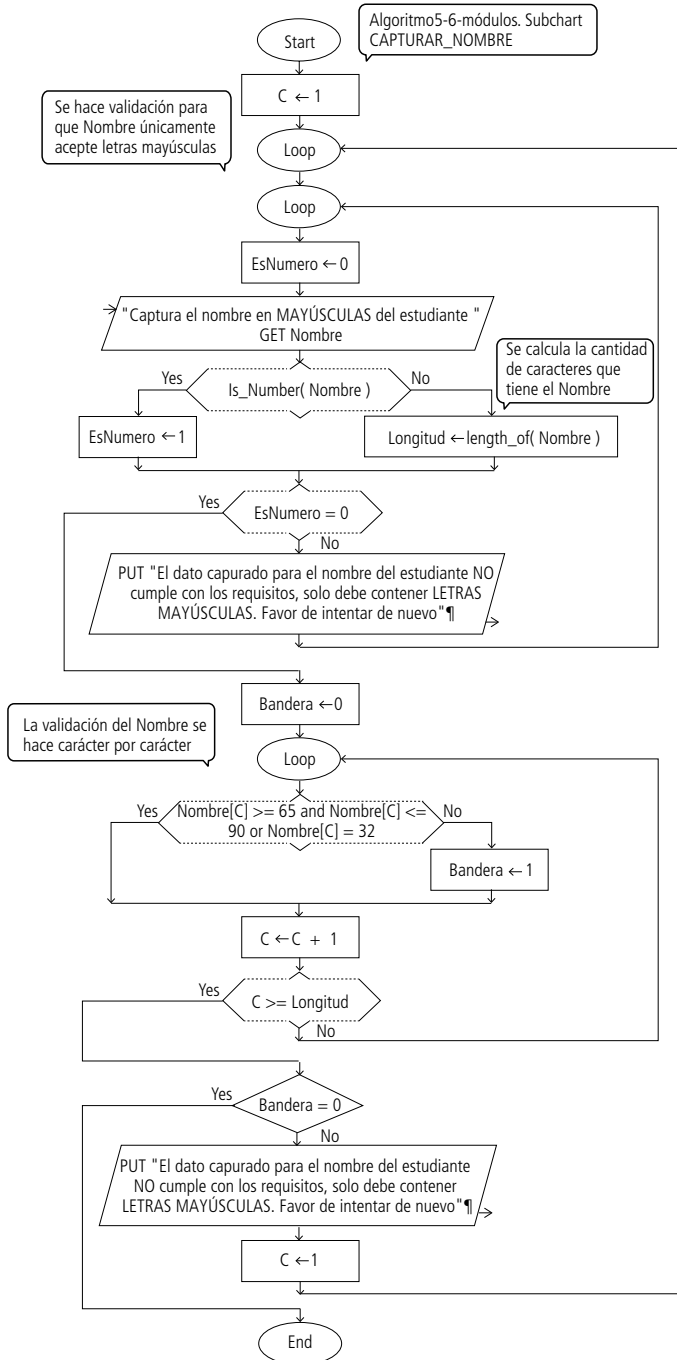


Figura 5.27 Algoritmo5-6-módulos Subchart CAPTURAR_NOMBRE.

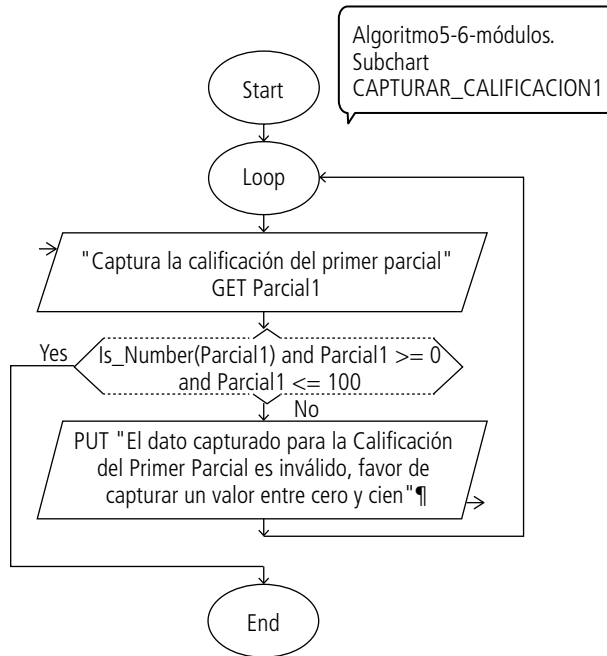


Figura 5.28 Algoritmo5-6-módulos Subchart CAPTURAR_CALIFICACION1.

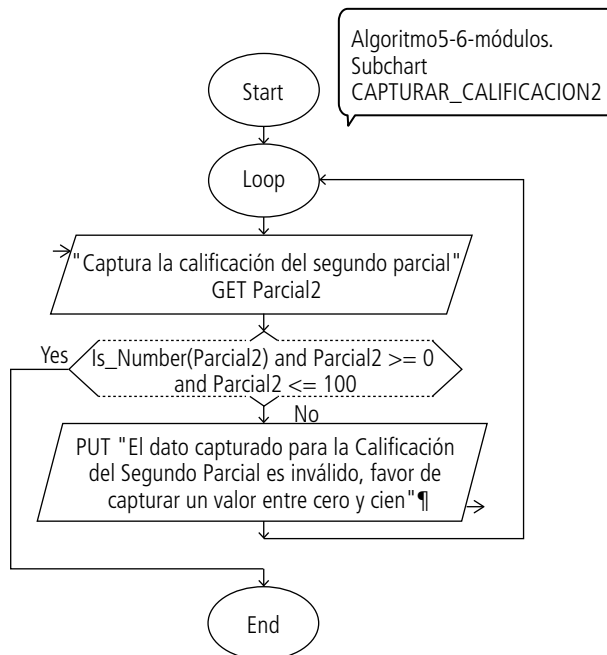


Figura 5.29 Algoritmo5-6-módulos Subchart CAPTURAR_CALIFICACION2.

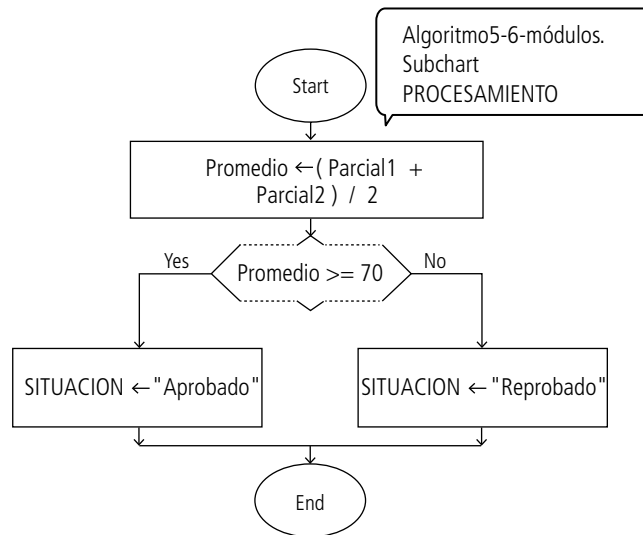


Figura 5.30 Diagrama algoritmo5-6-módulos *Subchart* PROCESAMIENTO.

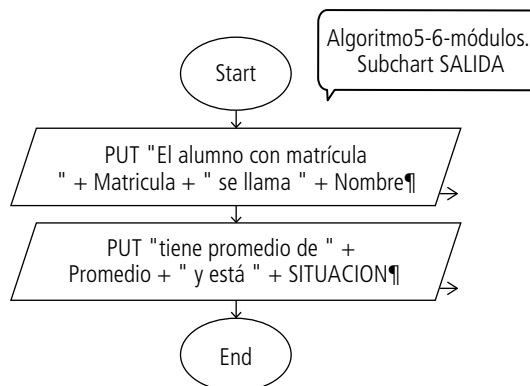


Figura 5.31 Diagrama algoritmo5-6-módulos *Subchart* SALIDA.



5.3 Diseño de algoritmos por módulos con paso de parámetros

Un parámetro es el valor de las variables y constantes que se transmiten de módulo hacia otro. Los módulos pueden emitir y recibir valores que se pueden clasificar como parámetros de entrada y parámetros de salida. Cuando un módulo envía un parámetro hacia otro módulo, para el módulo emisor es un parámetro de salida, pero para el módulo receptor es un parámetro de entrada.

Para que una variable o constante pueda ser un parámetro de salida, dentro del módulo emisor debe recibir un valor. Sin embargo, para que una variable o constante sea un parámetro de entrada, antes de ejecutar el módulo receptor la variable o constante debió tomar un valor.

Hay módulos que emiten parámetros, otros sólo los reciben y otros más hacen ambas cosas, lo cual depende de la voluntad del programador y el diseño que prefiera implementar. Los módulos en Raptor pueden tener uno o varios parámetros de entrada y uno o varios de salida; hay parámetros que pueden ser de entrada y salida al mismo tiempo.

En Raptor, las acciones computacionales que forman parte de un módulo con paso de parámetros se ejecutan con el símbolo Call y reciben el nombre de *Procedure*. Los identificadores de los módulos se visualizan en las pestañas que aparecen al lado de la pestaña *main*. Raptor clasifica los parámetros que un *Procedure* envía como de salida (OUT) y los que se reciben como de entrada (IN). Para evitar confusiones o ambigüedades es conveniente afirmar que los *Subchart* son módulos donde el paso de parámetros es transparente y no requiere de una manipulación explícita de los mismos; por el contrario, los *Procedure* son módulos donde la manipulación de los parámetros debe ser explícita.

Las variables también se pueden clasificar, según su alcance, en variables globales y variables locales; sin embargo, Raptor no hace esta distinción de manera explícita. Cuando un algoritmo en Raptor utiliza módulos bajo el diseño de *Subchart*, el programador no tiene qué preocuparse en saber si es parámetro de entrada o de salida, pues todos los valores de las variables y constantes se consideran como parámetros globales. Un parámetro tiene alcance global cuando su valor puede ser utilizado por cualquier módulo que forme parte del algoritmo. Un parámetro tiene alcance local cuando su valor puede ser utilizado por el módulo donde se usó o recibió por primera vez su valor; para que otro módulo pueda emplearlo debe ser enviado y recibido de manera explícita.

El concepto de parámetro global permite que el programador prefiera utilizar módulos *Subchart*. Sin embargo, hay suficientes razones de Ingeniería de Software como para desechar esta postura de comodidad y no esfuerzo. Por tal motivo, se le sigue que si quiere ser un programador profesional debe conocer y aplicar los conceptos del paso de parámetros entre módulos.

Cuando un algoritmo en Raptor utiliza módulos bajo el diseño de *Procedure*, los valores de las variables y constantes pueden ser tratados como parámetros globales o locales, según sea el caso. Para crear módulos bajo esta perspectiva, se recomienda pensar en la función y en el producto generado por el módulo. Para determinar con claridad la función del módulo se debe especificar cuáles son los parámetros que requiere y que le permitan generar el producto que se desea.

Por ejemplo, para un módulo *Procedure* cuya función sea “la captura y validación de un dato que debe estar en un rango determinado”; el producto generado será “una variable que tiene un valor, el cual cumple con las especificaciones adecuadas” para realizar un correcto proceso de datos.

Los pasos a seguir para crear una pestaña de *Procedure* es el mismo que para el *Subchart*. Pero hay que asegurarse que en el menú MODE de Raptor esté activada la opción *Intermediate*, de acuerdo con la figura 5.32.

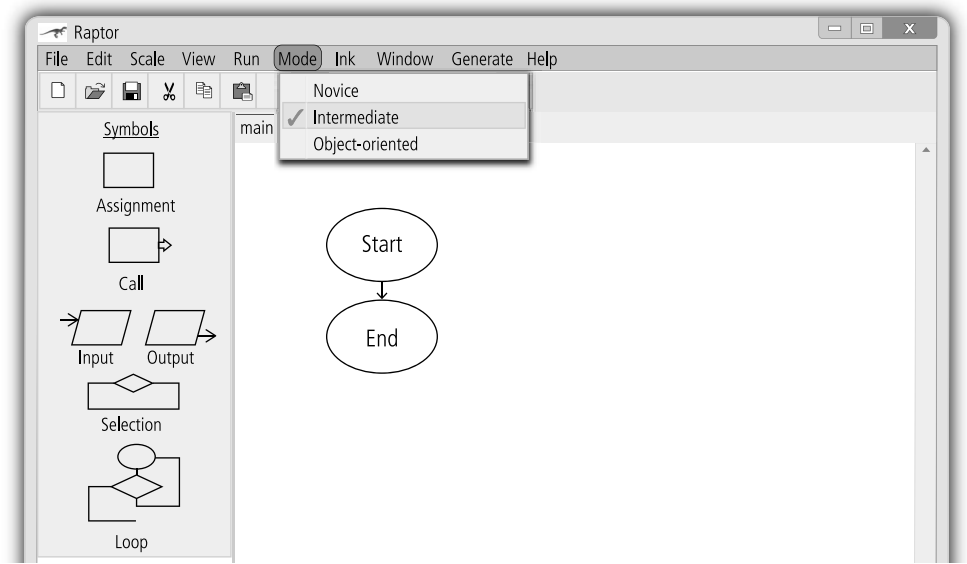


Figura 5.32 Menú MODE opción *Intermediate*.

El diagrama del **Algoritmo5-7-procedimientos** cuenta dos módulos que son INICIO y FIN, además de *main*. Este diagrama permite imprimir la edad de una persona. Véase la tabla 5.2 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.2 Planeación de los módulos para el **Algoritmo5-7-procedimientos**

Módulo	Función	Producto generado	Parámetros
INICIO	Permitir que a la variable Edad se le asigne un valor.	La variable Edad con un valor asignado.	Edad es parámetro de salida.
FIN	Utilizar el valor de la variable Edad en un símbolo Output.	Mostrar en <i>MasterConsole</i> el valor de la variable Edad en un mensaje de contextualización.	Edad es parámetro de entrada.
<i>main</i>	Controlar el programa y servir como receptor-emisor del valor de la variable Edad.	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función del *Procedure* INICIO es permitir que a la variable Edad se le asigne un valor. Segundo, la función del *Procedure* FIN es utilizar el valor de la variable Edad en un símbolo *OUTPUT*. Tercero, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor del valor de la variable Edad.

El producto generado de INICIO es la variable Edad con un valor asignado; Edad es su parámetro de salida y su valor se envía al *Subchart main*. En este caso en particular, Edad se considera como variable local a INICIO. Por otro lado, para que FIN pueda hacer su función requiere del valor de la variable Edad; esta variable es su parámetro de entrada. El producto generado de FIN es mostrar en *MasterConsole* el valor de la variable Edad en un mensaje de contextualización.

Observe que dentro del *Subchart main* se ejecutan los *Procedure* INICIO y FIN, lo que provoca que INICIO sea un módulo emisor del parámetro Edad y *main* sea un módulo receptor de Edad. Dado que *main* recibe el valor de la variable Edad, entonces, *main* puede enviar hacia FIN el valor de la variable Edad. Esto es posible porque el *Procedure* FIN está explícitamente capacitado para recibir ese valor, ya que Edad se estableció como parámetro de entrada para FIN.

Observe con detenimiento que *main* es un *Subchart* que recibe y emite el valor de la variable Edad. Sin embargo, esta capacidad de *main* no está explícitamente establecida, por lo que para *main* la recepción y emisión de parámetros es transparente. En este caso, *main* se convierte en un módulo que mantiene el control del programa, y ese control lo traspasa de manera momentánea a INICIO hasta que termina con su acción; posteriormente, *main* retoma el control y lo traspasa a FIN. Una vez que FIN ha concluido con su acción, *main* vuelve a tomar el control del programa para dar por terminado el algoritmo al ejecutar su globo End.

Otra observación que se le pide hacer es que los símbolos Call se refieren a los *Procedure*, los parámetros van entre paréntesis pero no se puede distinguir si son argumentos de entrada o de salida. Para poder ver lo anterior, seleccione las pestañas de los *Procedure* y en el globo Start es donde se puede saber si el parámetro es de entrada, de salida o incluso de ambos. Por cuestiones didácticas y para facilitar esta visualización, se ha agregado a los símbolos Call un comentario donde se muestra si el parámetro es de entrada o de salida (véanse figuras 5.33 a 5.35).

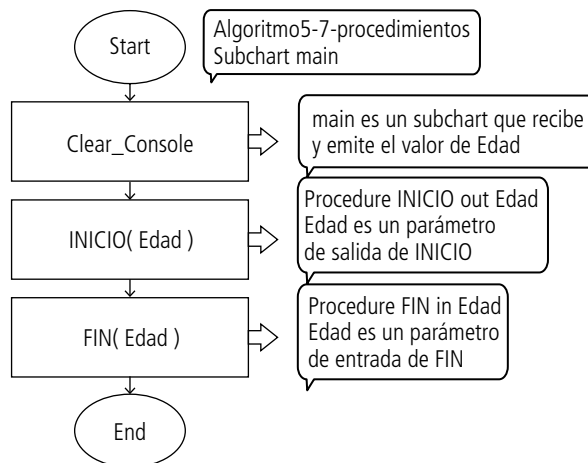


Figura 5.33 Algoritmo5-7-procedimientos *Subchart main*.

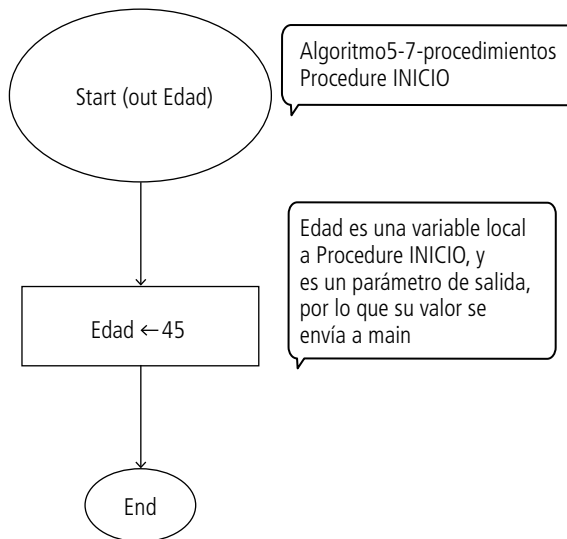


Figura 5.34 Algoritmo5-7-procedimientos *Procedure INICIO*.

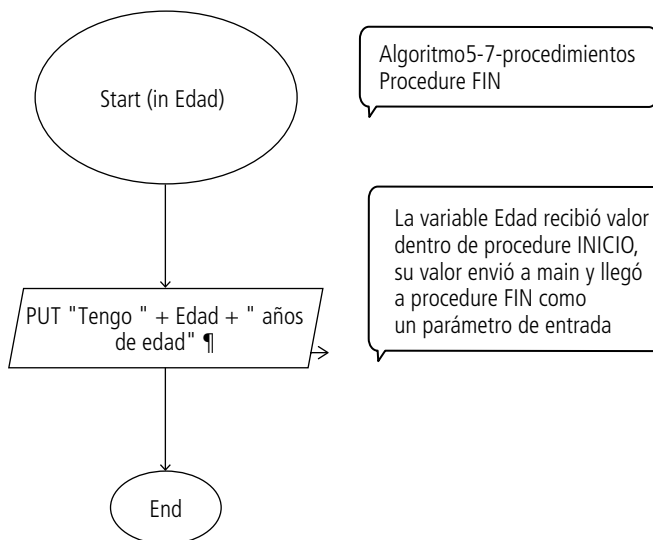


Figura 5.35 Algoritmo5-7-procedimientos *Procedure FIN*.

En el **Algoritmo5-8-procedimientos** se calcula el valor de la hipotenusa con la fórmula propuesta por Pitágoras. Este algoritmo cuenta con los módulos **ASIGNACION**, **OPERACIONES** y **RESULTADO**; además de *main*. Vea la tabla 5.3 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.3 Planeación de los módulos para el **Algoritmo5-8-procedimientos**

Módulo	Función	Producto generado	Parámetros
ASIGNACION	Permitir que las variables CatetoAdyacente y CatetoOpuesto adquieran un valor.	La variable Cateto adyacente y Cateto opuesto con un valor asignado.	Cateto adyacente y Cateto opuesto son parámetros de salida.
OPERACIONES	Recibir los valores de Cateto adyacente y Cateto opuesto, y usarlos en una operación matemática para calcular el valor para la variable Hipotenusa .	Un valor calculado y asignado a la variable Hipotenusa.	Cateto adyacente y Cateto opuesto son parámetros de entrada. Hipotenusa es parámetro de salida.
RESULTADO	Recibir los valores de Cateto adyacente, Cateto opuesto e Hipotenusa para utilizarlos en símbolos Output.	Mostrar en <i>MasterConsole</i> el valor de las variables Cateto adyacente, Cateto opuesto e Hipotenusa en un mensaje de contextualización.	Cateto adyacente, Cateto opuesto e Hipotenusa son parámetros de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función de ASIGNACION es permitir que las variables Cateto Adyacente y Cateto Opuesto adquieran un valor. Segundo, la función de OPERACIONES es recibir los valores de Cateto Adyacente, Cateto Opuesto, utilizarlos en una operación matemática para calcular el valor para la variable Hipotenusa y enviar este parámetro a *main*. Tercero, la función de RESULTADO es recibir los valores de Cateto Adyacente, Cateto Opuesto e Hipotenusa para utilizarlos en símbolos Output. Por último, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado de ASIGNACION es la variable Cateto adyacente y Cateto Opuesto con un valor asignado; ambos catetos son sus parámetros de salida, cuyos valores se envían a *main*. En este caso, Cateto Adyacente y Cateto Opuesto son variables locales a ASIGNACION.

El producto generado de OPERACIONES es un valor calculado y asignado a Hipotenusa; por tanto, es su parámetro de salida. Para que OPERACIONES puede calcular un valor para Hipotenusa requiere los parámetros de entrada de Cateto Adyacente y Cateto Opuesto. En este caso, Hipotenusa es una variable local para OPERACIONES.

El producto generado de RESULTADO es mostrar en *MasterConsole* el valor de las variables Cateto Adyacente, Cateto Opuesto e Hipotenusa en un mensaje de contextualización. Para que RESULTADO pueda realizar su función requiere como parámetro de entrada a Cateto Adyacente, Cateto Opuesto e Hipotenusa.

Dado que en este ejemplo los módulos emiten y reciben varios parámetros, se debe tener cuidado en el orden en que aparecen tanto en los símbolos Call como en los globos Start. Si hay una discrepancia en el orden en que se emiten o reciben los parámetros, la posibilidad de que haya errores en los resultados es alta. También si

hay omisión de parámetros o parámetros extra, ya sea en los símbolos Call o en los globos Start, se ocasionará errores de ejecución.

Al igual que en el ejemplo anterior, *main* es un *Subchart* que recibe y emite el valor de diversos parámetros. Sin embargo, esta capacidad de *main* no está establecida, por lo que para *main* la recepción y emisión de parámetros es transparente. De esta manera, *main* se convierte en un módulo que mantiene el control del programa y ese control lo traspasa de manera momentánea a otros módulos hasta que terminan con su acción; eventualmente *main* retoma el control para traspasarlo a otro módulo. Por último, *main* retoma el control del programa para dar por terminado el algoritmo al ejecutar su globo End (véase figura 5.36 a 5.39).

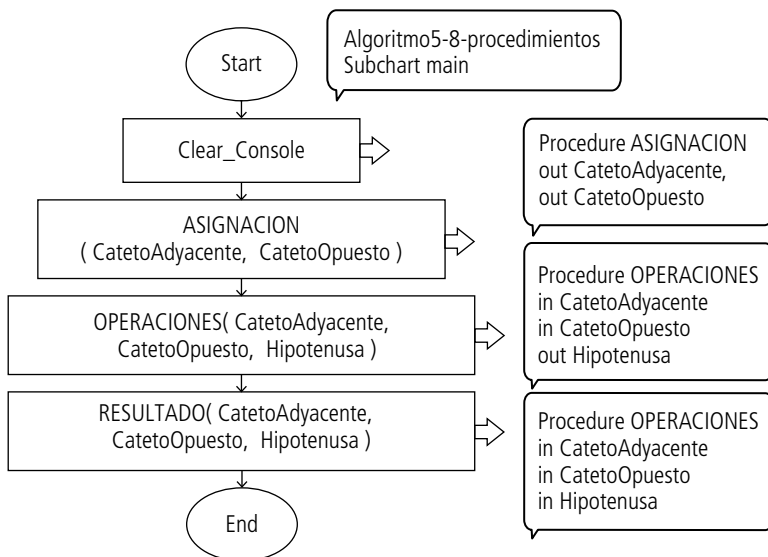


Figura 5.36 Algoritmo5-8-procedimientos *Subchart main*.

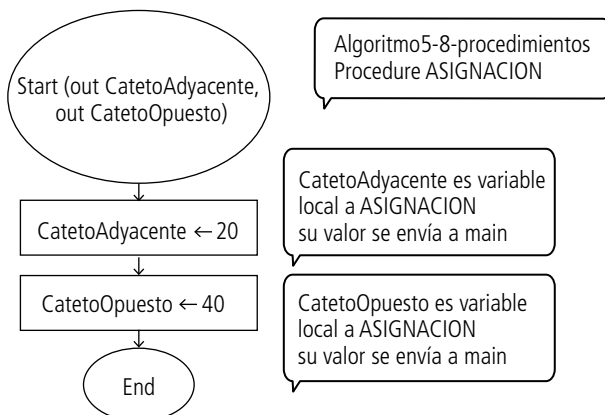
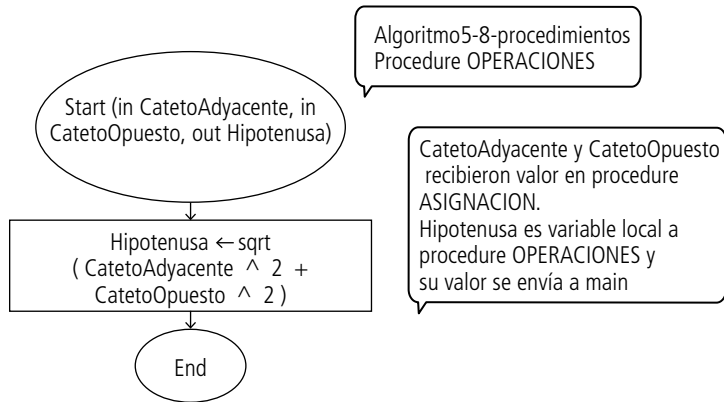
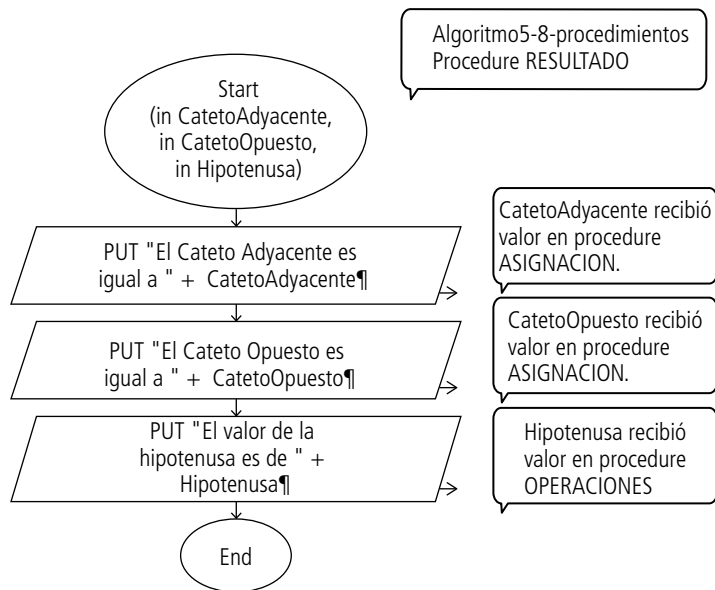


Figura 5.37 Algoritmo5-8-procedimientos *Procedure ASIGNACION*.

Figura 5.38 Algoritmo5-8-procedimientos *Procedure OPERACIONES*.Figura 5.39 Algoritmo5-8-procedimientos *Procedure RESULTADO*.

Con el **Algoritmo5-9-procedimientos** volveremos al problema de calcular la **Fuerza** a partir de la captura del dato de la **Masa** y la **Aceleracion**. En este algoritmo se muestra cómo debe diseñarse un *Procedure* por cada variable a capturar, lo cual permite validar el valor del dato agregado por el usuario. Recuerde que el objetivo de la validación es impedir la captura de datos que no cumplen con los requerimientos teóricos del problema, evitando así la ejecución de operaciones matemáticas que generen errores de ejecución o de resultados fuera de contexto. Vea la tabla 5.4 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.4 Planeación de los módulos para el **Algoritmo5-9-procedimientos**

Módulo	Función	Producto generado	Parámetros
CAPTURAR_MASA	Permitir al usuario que agregue un valor para la variable Masa, el cual debe ser numérico y mayor que cero.	La variable Masa con un valor que cumple con los requerimientos teóricos.	Masa es un parámetro de salida.
CAPTURAR_ACELERACION	Permitir al usuario que agregue un valor para la variable Aceleracion, el cual debe ser numérico.	La variable Aceleracion con un valor que cumple con los requerimientos teóricos.	Aceleracion es un parámetro de salida.
PROCESAMIENTO	Recibir los valores capturados por el usuario de Masa y Aceleracion, y realizar una operación matemática que calcule el valor de Fuerza.	Un valor calculado y asignado a la variable Fuerza.	Masa y Aceleracion son parámetros de entrada. Fuerza es parámetro de salida.
RESULTADO	Recibir los valores de Masa, Aceleracion y Fuerza para utilizarlos en símbolos <i>Output</i> .	Mostrar en <i>MasterConsole</i> el valor de las variables Masa, Aceleracion y Fuerza en un mensaje de contextualización.	Masa, Aceleracion y Fuerza son parámetros de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Este algoritmo tiene cuatro módulos, además de *main*, que son: **CAPTURAR_MASA**, **CAPTURAR_ACELERACION**, **PROCESAMIENTO** y **RESULTADO**. Primero, la función de **CAPTURAR_MASA** permite al usuario que capture un valor para la variable Masa, el cual debe ser numérico y mayor que cero; el valor de Masa se envía a *main*. Segundo, la función de **CAPTURAR_ACELERACION** permite al usuario que capture un valor para la variable Aceleracion, el cual debe ser numérico; el valor de Aceleracion se envía a *main*. Tercero, la función de **PROCESAMIENTO** recibe los valores capturados por el usuario de Masa y Aceleracion, y realiza una operación matemática que calcula el valor de Fuerza, este valor se envía a *main*. Cuarto, la función de **RESULTADO** es recibir los valores de Masa, Aceleración y Fuerza para usarlos en símbolos *Output*. Por último, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado **CAPTURAR_MASA** y **CAPTURAR_ACELERACION** es una variable con un valor que cumple con los requerimientos teóricos del problema que se intenta solucionar. Lo anterior implica que esa variable será su parámetro de salida, tal como ocurre con el valor de la variable Masa en **CAPTURAR_MASA** y con Aceleracion en **CAPTURAR_ACELERACION**, las cuales son variables locales de su respectivo *Procedure*.

El producto generado de PROCESAMIENTO será el cálculo y asignación de un valor para Fuerza a partir de los datos capturados para Masa y Aceleracion; por tanto, éstos son sus parámetros de entrada y el nuevo valor calculado, es decir, la Fuerza, es un parámetro de salida, el cual se considera como una variable local de PROCESAMIENTO.

El producto generado por RESULTADO será Mostrar en *MasterConsole* el valor de las variables Masa, Aceleracion y Fuerza, en un mensaje de contextualización, dichos valores son sus parámetros de entrada (véase figura 5.40 a 5.44).

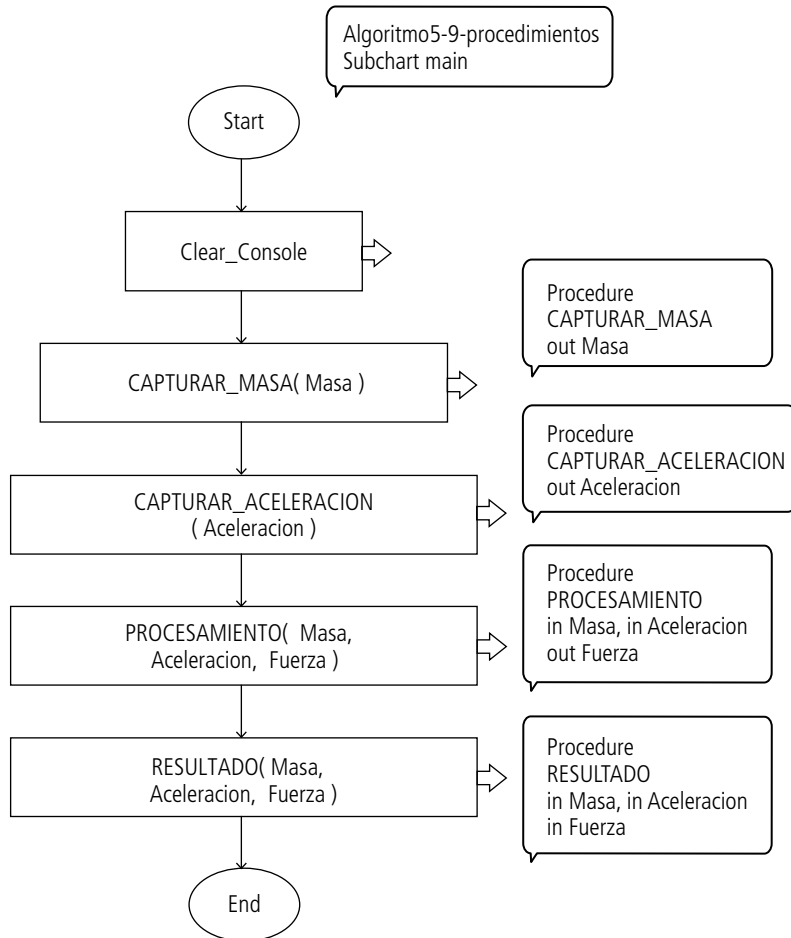


Figura 5.40 Algoritmo5-9-procedimientos *Subchart main*.

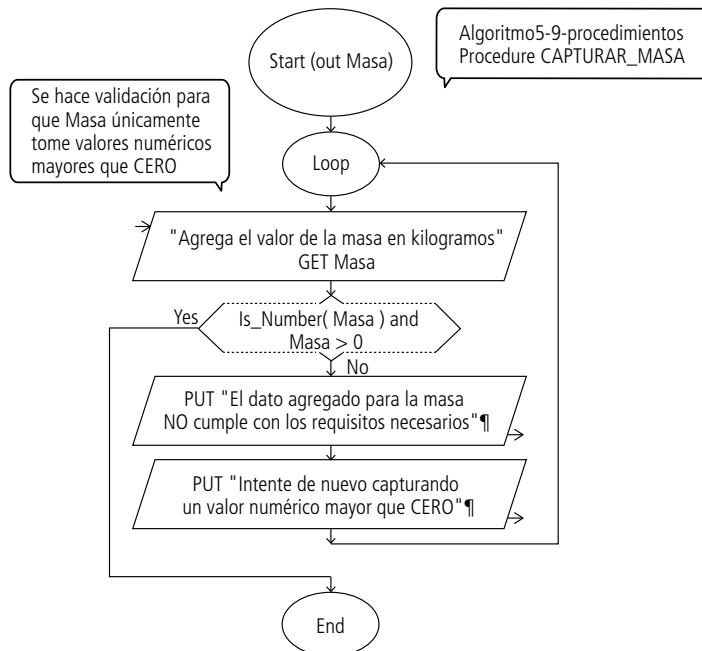


Figura 5.41 Algoritmo5-9-procedimientos *Procedure CAPTURAR_MASA*.

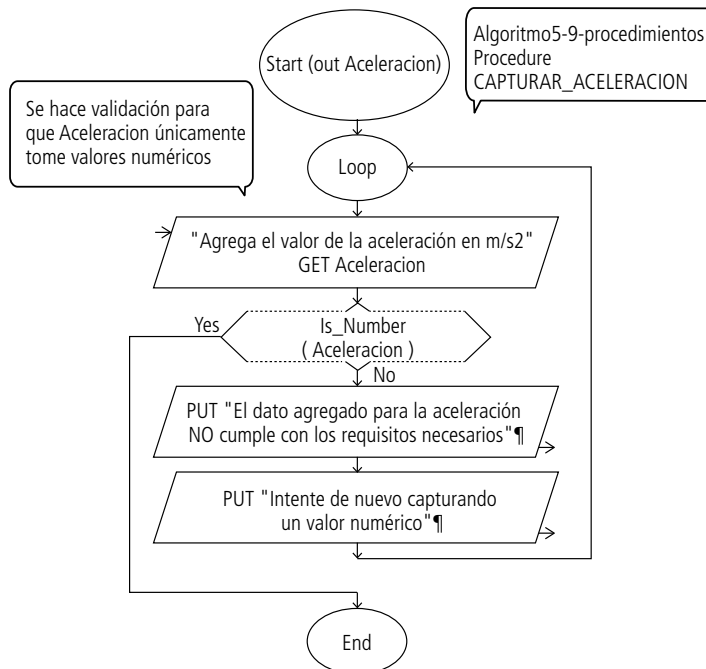


Figura 5.42 Algoritmo5-9-procedimientos *Procedure CAPTURAR_ACELERACION*.

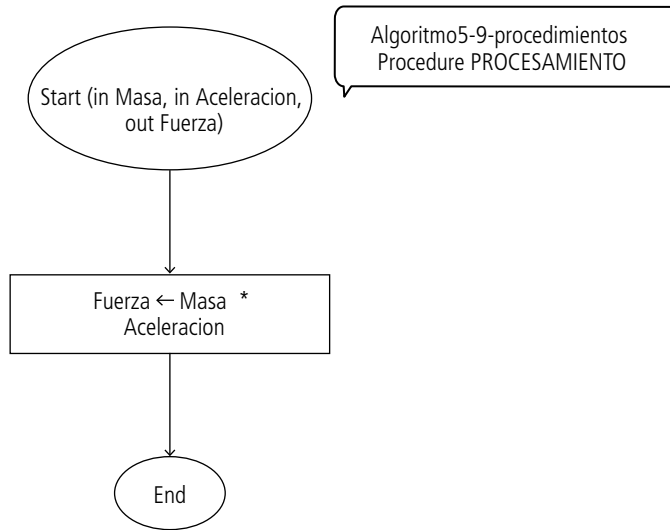
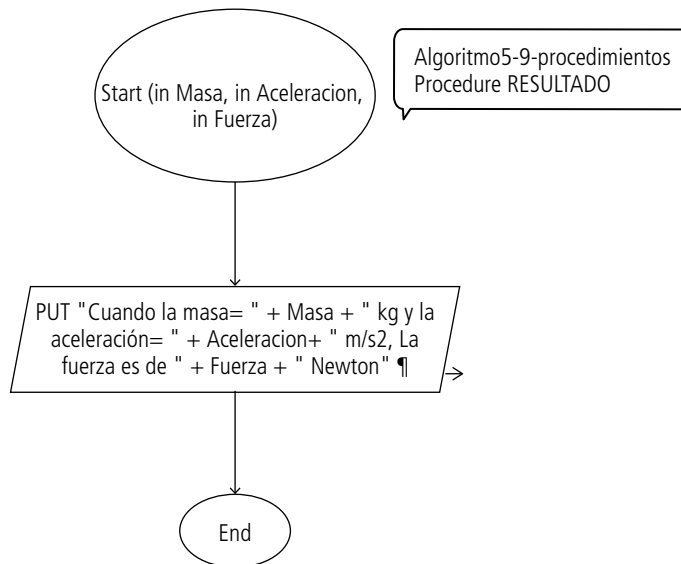


Figura 5.43 Algoritmo5-9-procedimientos *Procedure PROCESAMIENTO*.



Figuras 5.44 Algoritmo5-9-procedimientos *Procedure RESULTADO*.

En el **Algoritmo5-10-procedimientos** a partir de una lista que contiene una cantidad de elementos numéricos determinada por el usuario, se calcula el promedio de ellos. Este diagrama tiene cuatro módulos, además de *main*, que son: **INICIO**, **PROCESO**, **CAPTURAR_NUMERO** y **RESULTADO**. Vea la tabla 5.5 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.5 Planeación de los módulos para el Algoritmo5-10-procedimientos

Módulo	Función	Producto generado	Parámetros
INICIO	Permitir al usuario que agregue un valor numérico y mayor que cero, correspondiente a la Cantidad de números a considerar en el promedio.	La variable Cantidad con un valor que cumple con los requerimientos lógicos del problema.	Cantidad es un parámetro de salida.
PROCESO	Recibir el valor capturado por el usuario de Cantidad, ejecutar el módulo que permite capturar los valores numéricos, y calcular la sumatoria y promedio de los números.	Un valor calculado y asignado para la variable Suma y Promedio.	Cantidad es un parámetro de entrada. Suma y Promedio son parámetros de salida.
CAPTURAR_ NUMERO	Recibir el valor de la <i>i-ésima</i> posición de la cantidad numérica a capturar y permitir que el usuario capture valores numéricos que son parte de la lista de números.	Una cantidad numérica agregada a la variable Valor.	La letra <i>i</i> representa un parámetro de entrada. Valor es parámetro de salida.
RESULTADO	Recibir los valores de Cantidad, Suma y Promedio para utilizarlos en símbolos Output.	Mostrar en <i>MasterConsole</i> el valor de las variables Cantidad, Suma y Promedio en un mensaje de contextualización.	Cantidad, Suma y Promedio son parámetros de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función de **INICIO** es permitir al usuario que agregue un valor numérico y mayor que cero, correspondiente a la Cantidad de números de los cuales se calculará el promedio. Segundo, la función de **PROCESO** es recibir el valor capturado para Cantidad, ejecutar el módulo que permite capturar los valores numéricos, y calcular la Suma y Promedio de los valores numéricos. Tercero, la función de **CAPTURAR_ NUMERO** es recibir el valor de la *i-ésima* posición de la cantidad numérica a capturar y permitir que el usuario capture exclusivamente valores numéricos que son parte de la lista de números. Cuarto, la función de **RESULTADO** es recibir y permitir que el usuario vea en el *MasterConsole* el valor de las variables Cantidad, Suma y Promedio. Por último, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado de INICIO será la variable Cantidad con la asignación de un valor numérico positivo que se refiere a la cantidad de números que se capturarán

y de los cuales se obtendrá el promedio, por tal motivo dicha variable será su parámetro de salida; en este caso, Cantidad es una variable local al módulo INICIO y su valor se envía a *main*.

El producto generado de **PROCESO** es un valor calculado y asignado a Suma y Promedio, variables que son consideradas como parámetros de salida. Para que PROCESO pueda realizar su función requiere como parámetro de entrada al valor agregado a Cantidad. En este caso, Suma, Promedio, Valor e *i* son variables locales a PROCESO; los valores de las variables Suma y Promedio se envían a *main*.

El producto generado de **CAPTURAR_NUMERO** es una cantidad numérica agregada a la variable Valor que se envía a PROCESO. En este caso, **Valor** es una variable local a **CAPTURAR_NUMERO**.

El producto generado de **RESULTADO** es mostrar en *MasterConsole* el valor de las variables Cantidad, Suma y Promedio en un mensaje de contextualización, entonces, Cantidad, Suma y Promedio son sus parámetros de entrada (véanse figuras 5.45 a 5.49).

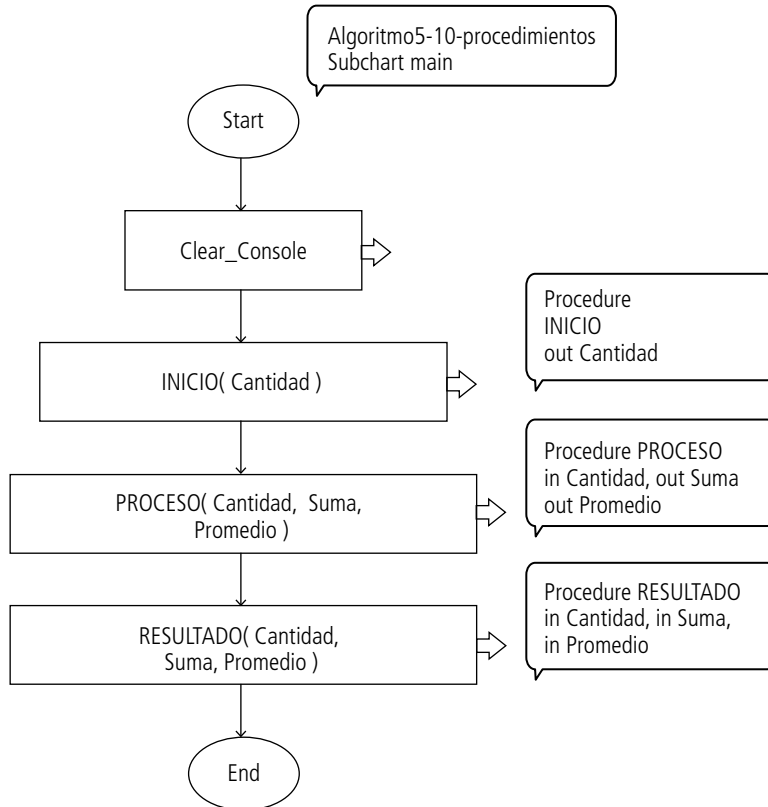


Figura 5.45 Algoritmo5-10-procedimientos *Subchart main*.

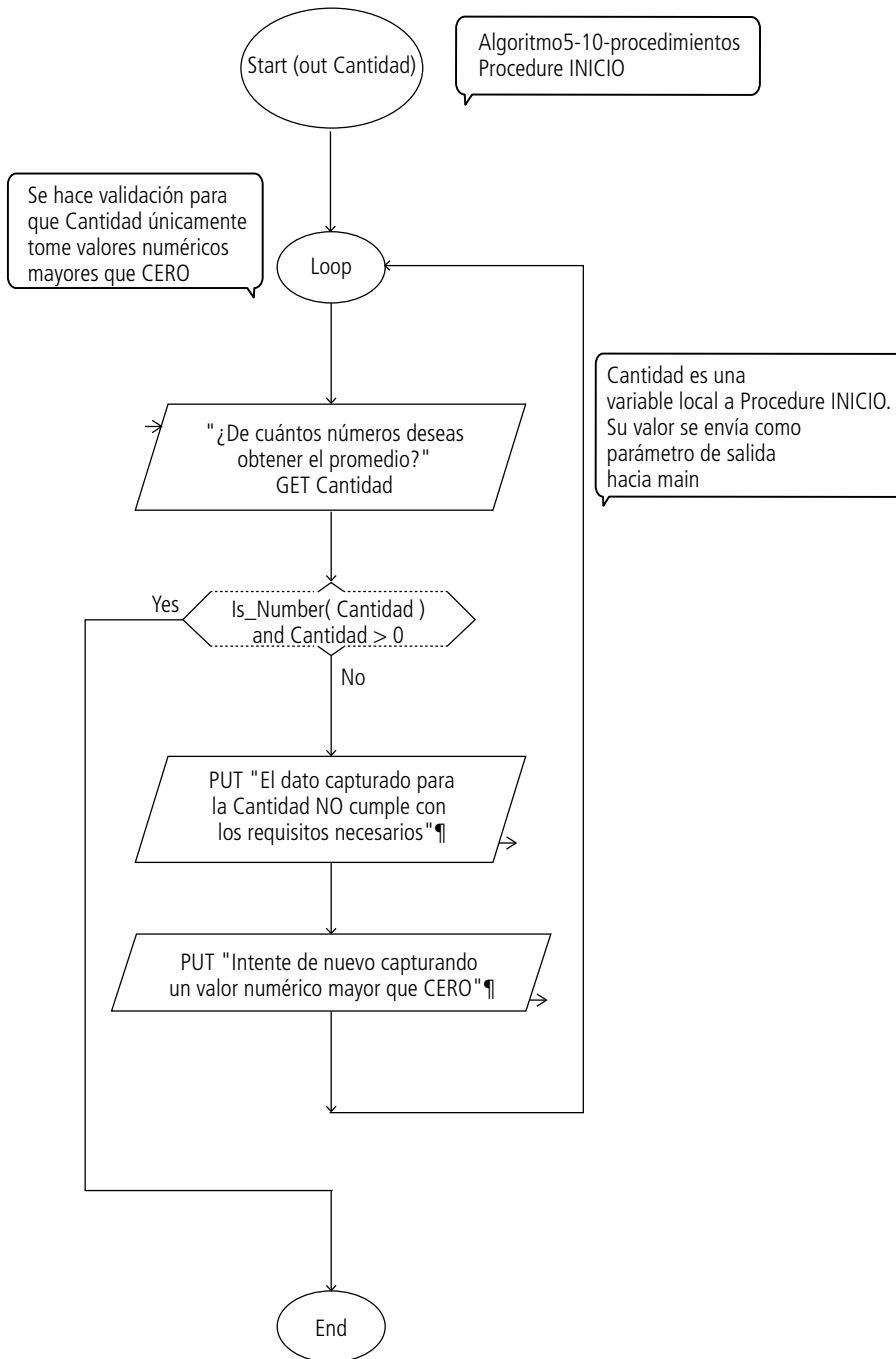


Figura 5.46 Algoritmo5-10-procedimientos *Procedure INICIO*.

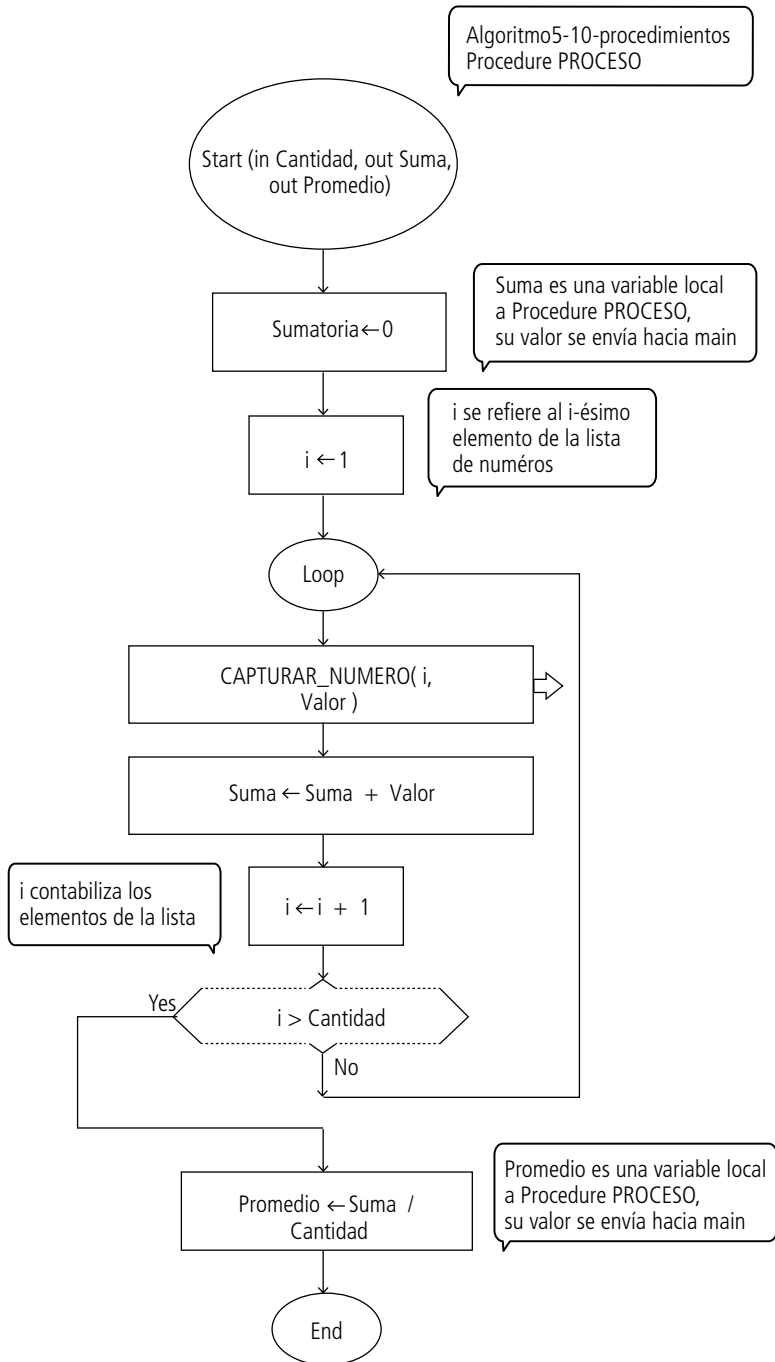


Figura 5.47 Algoritmo5-10-procedimientos *Procedure PROCESO*.

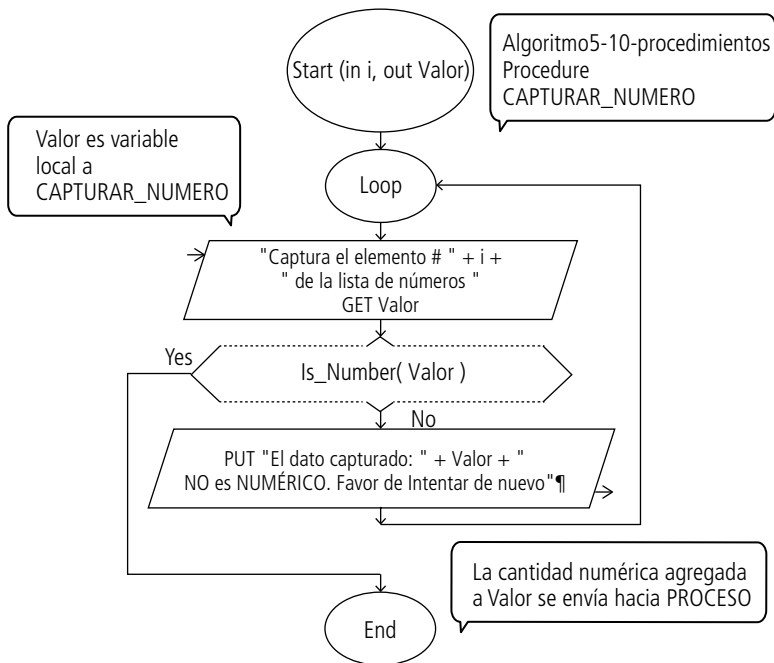


Figura 5.48 Algoritmo5-10-procedimientos *Procedure* CAPTURAR_NUMERO.

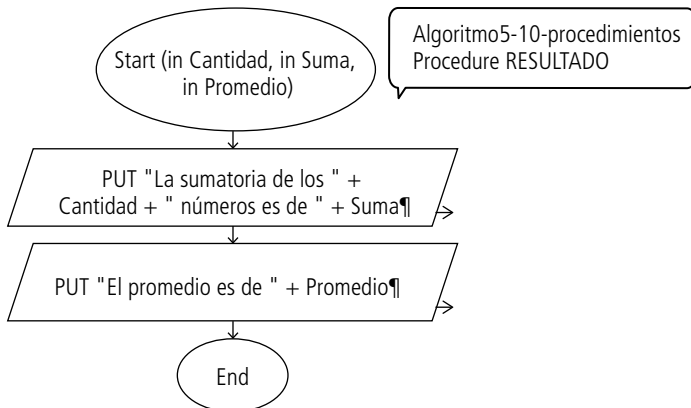


Figura 5.49 Algoritmo5-10-procedimientos *Procedure* RESULTADO.

En el **Algoritmo5-11-procedimientos** a partir de una lista que contiene una cantidad, predeterminada por el usuario, de elementos numéricos, se detecta cuál es el número más grande de todos ellos. Este diagrama tiene cuatro módulos, además de *main*, que son: **INICIO**, **ELEMENTO_MAYOR**, **CAPTURAR_NUMERO** y **RESULTADO**. Vea la tabla 5.6 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.6 Planeación de los módulos para el **Algoritmo5-11-procedimientos**

Módulo	Función	Producto generado	Parámetros
INICIO	Permitir al usuario que capture un valor numérico y mayor que cero, correspondiente a la Cantidad de números a considerar en el promedio.	La variable Cantidad con un valor que cumple con los requerimientos lógicos del problema.	Cantidad es un parámetro de salida.
ELEMENTO_ MAYOR	Recibir el valor capturado por el usuario de Cantidad, ejecutar el módulo que permite escribir los valores numéricos y determinar el elemento mayor de la lista de números.	Un valor asignado a la variable Mayor correspondiente al elemento más grande de la lista de números.	Cantidad es parámetro de entrada. Mayor es parámetro de salida.
CAPTURAR_ NUMERO	Recibir el valor de la <i>i-ésima</i> posición de la cantidad numérica a capturar y permitir que el usuario capture de manera exclusiva valores numéricos que son parte de la lista de números.	Una cantidad numérica agregada a la variable Valor.	La letra <i>i</i> es parámetro de entrada. Valor es parámetro de salida.
RESULTADO	Recibir el valor de Mayor para utilizarlo en un símbolo Output.	Mostrar en <i>MasterConsole</i> el valor de la variable Mayor en un mensaje de contextualización.	Mayor es parámetro de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función de INICIO es permitir al usuario que agregue un valor numérico y mayor que cero, correspondiente a la Cantidad de números de los cuales buscará el número más grande. Segundo, la función de ELEMENTO_ MAYOR es recibir el valor capturado para Cantidad, ejecutar el módulo que permite escribir los valores numéricos y determinar cuál es el elemento mayor. Tercero, la función de CAPTURAR_ NUMERO es recibir el valor de la *i-ésima* posición de la cantidad numérica a capturar y permite que el usuario capture exclusivamente valores numéricos que son parte de la lista de números. Cuarto, la función de RESULTADO es recibir y permitir que el usuario vea en el *MasterConsole* el valor de la variable Mayor. Por último, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado de INICIO es la variable Cantidad con la asignación de un valor numérico positivo que se refiere a la cantidad de números que se capturarán y de los cuales se determinará el elemento mayor, por tal motivo dicha variable será su parámetro de salida; en este caso, Cantidad es una variable local del módulo INICIO y su valor se envía a *main*.

El producto generado de `ELEMENTO_MAYOR` es un valor asignado a la variable `Mayor` correspondiente al elemento más grande de la lista de números, variable que se considera como parámetro de salida. Para que `ELEMENTO_MAYOR` pueda realizar su función requiere como parámetro de entrada al valor agregado a `Cantidad`. En este caso, `Mayor`, `Valor` e `i` son variables locales a `ELEMENTO_MAYOR`; el valor de la variable `Mayor` se envía a *main*.

El producto generado de `CAPTURAR_NUMERO` es que una cantidad numérica agregada a la variable `Valor` que se envía hacia `ELEMENTO_MAYOR`. En este caso, `Valor` es una variable local a `CAPTURAR_NUMERO`.

El producto generado de `RESULTADO` es mostrar en *MasterConsole* el valor de la variable `Mayor` en un mensaje de contextualización, la cual es su parámetro de entrada (véanse figuras 5.50 a 5.54).

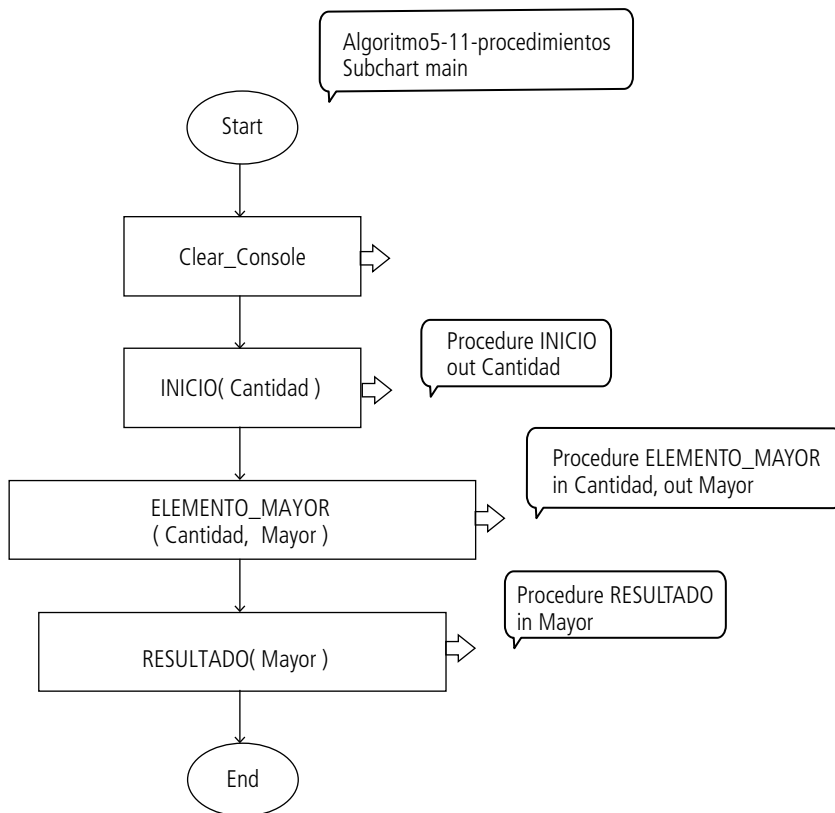


Figura 5.50 Algoritmo5-11-procedimientos *Subchart main*.

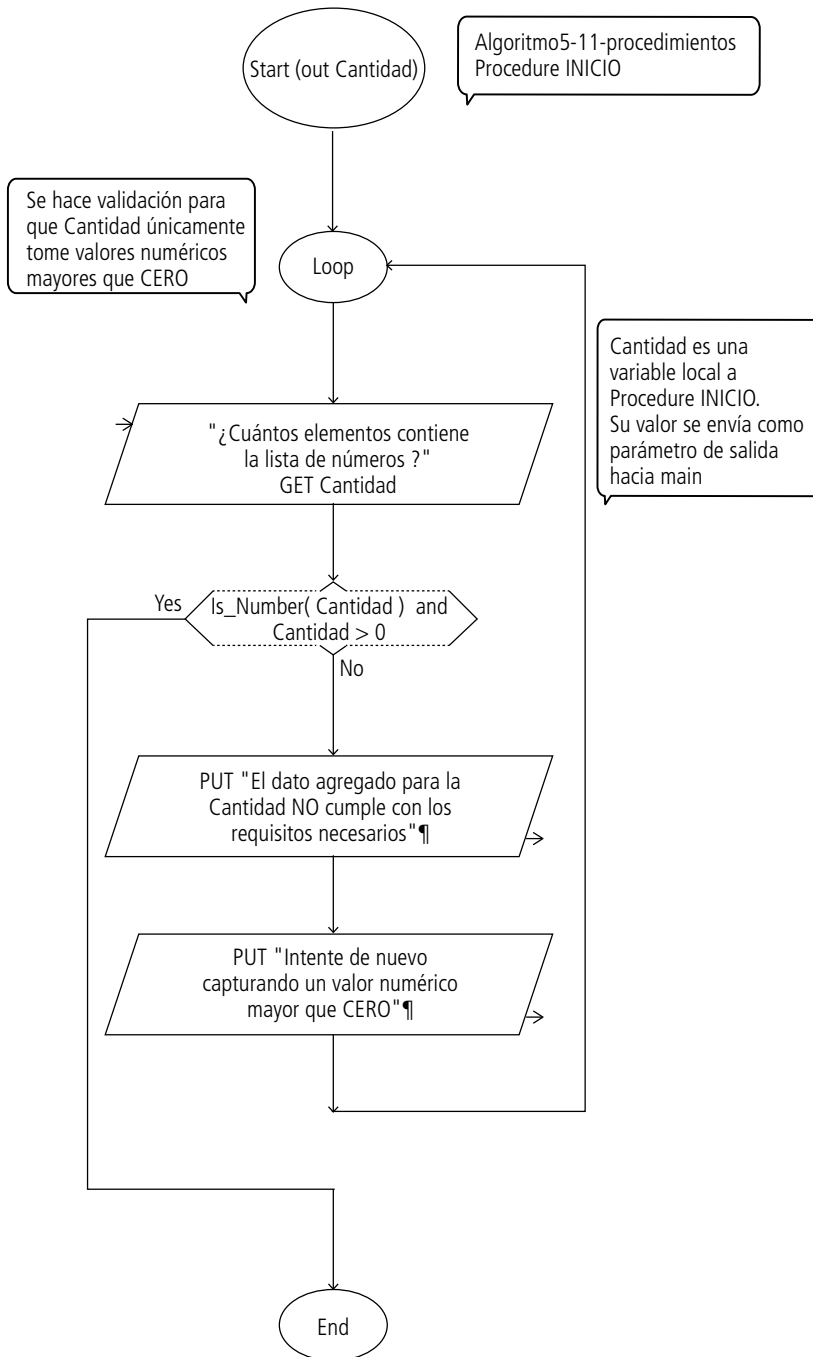


Figura 5.51 Algoritmo5-11-procedimientos *Procedure INICIO*.

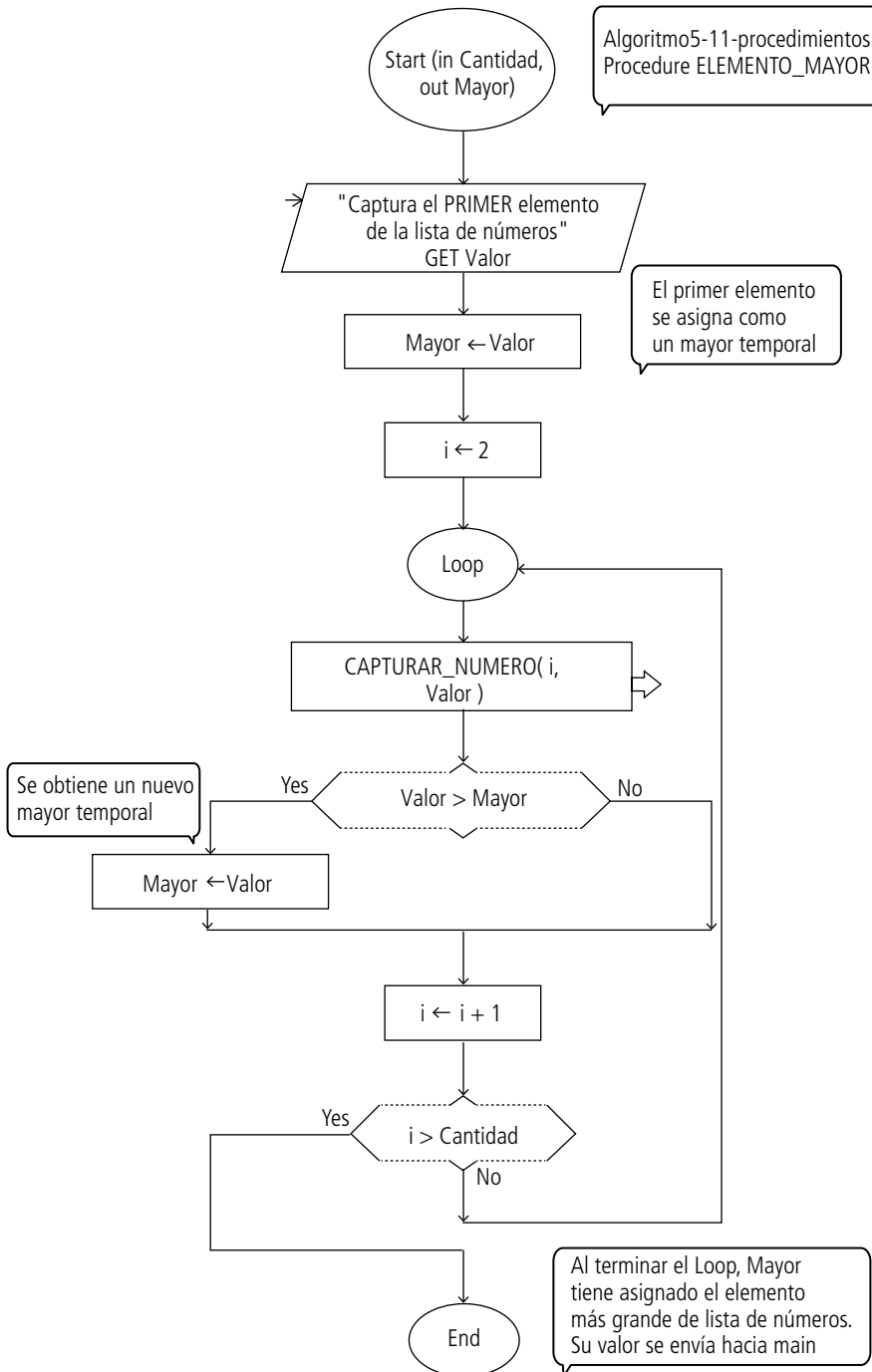


Figura 5.52 Algoritmo5-11-procedimientos *Procedure* ELEMENTO_MAYOR.

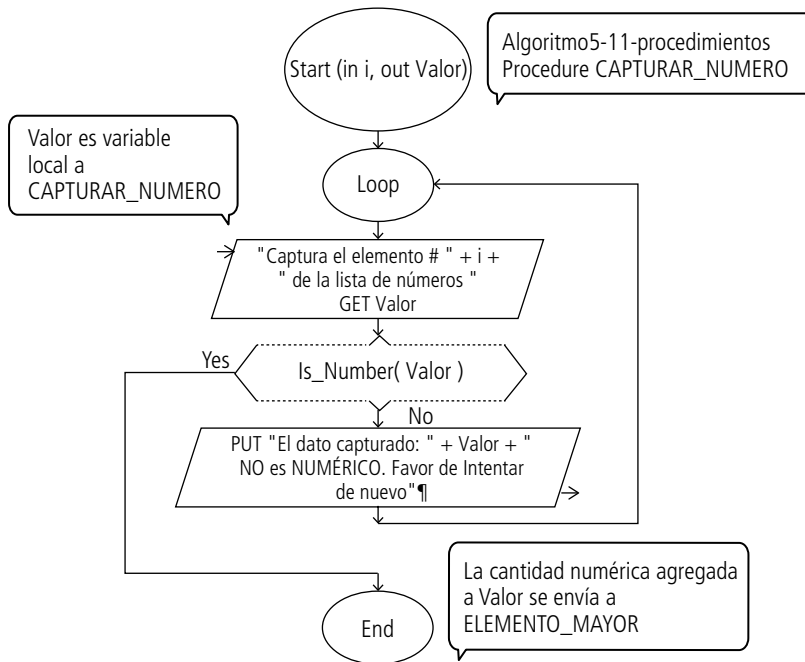


Figura 5.53 Algoritmo5-11-procedimientos *Procedure CAPTURAR_NUMERO*.

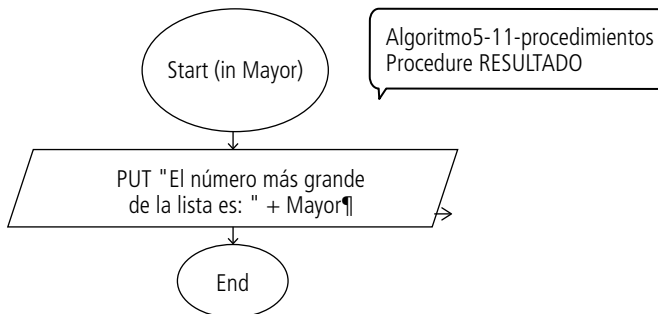


Figura 5.54 Algoritmo5-11-procedimientos *Procedure RESULTADO*.

La habilidad para determinar cuándo un parámetro es de entrada, de salida o ambos, se obtiene con la práctica. No hay reglas generales que puedan proporcionarse para determinar esto, mucho tiene que ver la discrecionalidad y el diseño a conveniencia del programador. Sin embargo, damos algunas ideas que pueden servir al momento de precisar cómo se manejarán los parámetros en los módulos.

1. Si un módulo va a utilizar una variable o constante como operando en una operación matemática, el programador debe asegurar que a dicho identificador se le asigne un valor antes de ejecutar la operación aritmética o asegurarse

que la variable es un parámetro de entrada que ha recibido un valor en un módulo previo de acuerdo con la secuencia del algoritmo.

2. Si en un módulo alguna variable obtiene un valor por primera vez, según la secuencia del algoritmo, se les considera parámetro local a ese módulo.
3. El valor de una variable local a un módulo puede enviarse como parámetro de salida hacia otro módulo.
4. Un módulo que recibe el valor de una variable o constante se considera parámetro de entrada para el módulo receptor.

Con el **Algoritmo5-12-procedimientos** se resolverá el siguiente problema. El Señor López ha recibido una oferta de empleo de la Compañía de juguetes “Ensueño S.A. de C.V.” con la oportunidad de elegir entre dos sistemas de pago. La primera opción es que pueda recibir un salario mensual de \$5,000 y \$50 de aumento cada mes. La segunda es que pueda recibir un salario mensual de \$5,000 con un aumento anual de \$800. Realice un programa que determine el salario mensual para los siguientes tres años. El programa debe imprimir la corrida de ambas opciones con los siguientes datos: el número de mes, el salario mensual y el acumulado mensual. Después de ambas corridas imprima el total al final de los 3 años de las dos opciones y a partir de esta información el diagrama debe determinar el mejor método de pago.

Este diagrama tiene siete módulos, además de *main*, que son: **CAPTURAR_SALARIO_MENSUAL**, **CAPTURAR_AUMENTO_MENSUAL**, **CAPTURAR_AUMENTO_ANUAL**, **CAPTURAR_PERIODO**, **OPCION1**, **OPCION2** y **DECISION**. Vea la tabla 5.7 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.7 Planeación de los módulos para el **Algoritmo5-12-procedimientos**

Módulo	Función	Producto generado	Parámetros
CAPTURAR_SALARIO_MENSUAL	Permitir al usuario que capture un valor para la variable Salario Mensual .	La variable Salario Mensual con un valor numérico mayor que cero.	Salario Mensual es un parámetro de salida.
CAPTURAR_AUMENTO_MENSUAL	Permitir al usuario que capture un valor para la variable Aumento Mensual .	La variable Aumento Mensual con un valor numérico mayor que cero.	Aumento Mensual es un parámetro de salida.
CAPTURAR_AUMENTO_ANUAL	Permitir al usuario que capture un valor para la variable Aumento Anual .	La variable Aumento Anual con un valor numérico mayor que cero.	Aumento Anual es un parámetro de salida.

CAPTURAR_ PERIODO	Permitir al usuario que capture un valor para la variable Periodo correspondiente a la cantidad de años a considerar en la corrida de las opciones.	La variable Periodo , con un valor numérico mayor o igual que uno y menor o igual que cinco años. La variable Meses con un valor numérico correspondiente a la equivalencia del periodo en meses.	Periodo y Meses son parámetros de salida.
OPCION1	Recibir el valor capturado por el usuario de Salario Mensual, Aumento Mensual, Periodo y el valor calculado de Meses, y generar los datos de la corrida de la primera opción.	Mostrar en <i>MasterConsole</i> los resultados correspondientes a la corrida de la primera opción. Un valor calculado y asignado a la variable Total1 , correspondiente al total acumulado de la primera opción.	Salario Mensual, Aumento Mensual, Meses y Periodo son parámetros de entrada. Total1 es parámetro de salida.
OPCION2	Recibir el valor capturado por el usuario de Salario Mensual, Aumento Anual, Meses y Periodo y el valor calculado de Meses y generar los datos de la corrida de la segunda opción.	Mostrar en <i>MasterConsole</i> los resultados correspondientes a la corrida de la segunda opción. Un valor calculado y asignado a la variable Total2 correspondiente al total acumulado de la segunda opción.	Salario Mensual, Aumento Anual, Meses y Periodo son parámetros de entrada. Total2 es parámetro de salida.
DECISION	Recibir los valores de Periodo, Total1 y Total2 para utilizarlos en símbolos Output.	Mostrar en <i>MasterConsole</i> los resultados del problema en mensajes de contextualización.	Periodo, Total1 y Total2 son parámetros de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función de **CAPTURAR_SALARIO_MENSUAL**, **CAPTURAR_AUMENTO_MENSUAL**, **CAPTURAR_AUMENTO_ANUAL** y **CAPTURAR_PERIODO** es permitir que el usuario agregue los datos conocidos del problema. Segundo, la función de **OPCION1** es recibir el valor capturado por el usuario de Salario Mensual, Aumento Mensual, Periodo y el valor calculado de Meses y generar el total acumulado de la primera opción. Tercero, la función de **OPCION2** es recibir el valor capturado por el usuario de Salario Mensual, Aumento Anual, Periodo y el valor calculado de Meses y generar el total acumulado de la segunda opción. Cuarto, la función de **DECISION** es recibir los valores de Periodo, Total1 y Total2 para imprimir los resultados del problema. Por último, la función del *Subchart* la *main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado de **CAPTURAR_SALARIO_MENSUAL**, **CAPTURAR_AUMENTO_MENSUAL**, **CAPTURAR_AUMENTO_ANUAL** y **CAPTURAR_PERIODO** son las variables Salario Mensual, Aumento Mensual, Aumento Anual,

Periodo y Meses, respectivamente, con un valor que cumple con los requerimientos del problema; estos valores se envían a *main* y se consideran como variables locales a su respectivo módulo.

El producto generado de OPCION1 es mostrar en *MasterConsole* los resultados de la corrida de la primera opción; además, de un valor calculado y asignado a la variable Total1 y al total acumulado de la primera opción; este valor se envía a *main*. En este caso, Total1 y Mes son variables locales a OPCION1.

El producto generado de OPCION2 es mostrar en *MasterConsole* los resultados de la corrida de la segunda opción y un valor calculado y asignado a la variable Total2 del total acumulado de la segunda opción; este valor se envía a *main*. En este caso, Total2, Mes y Cambio Anual se consideran variables locales de la OPCION2.

Finalmente, el producto generado de DECISION es mostrar en *MasterConsole* los resultados del problema en mensajes de contextualización (véanse figuras 5.55 a 5.62).

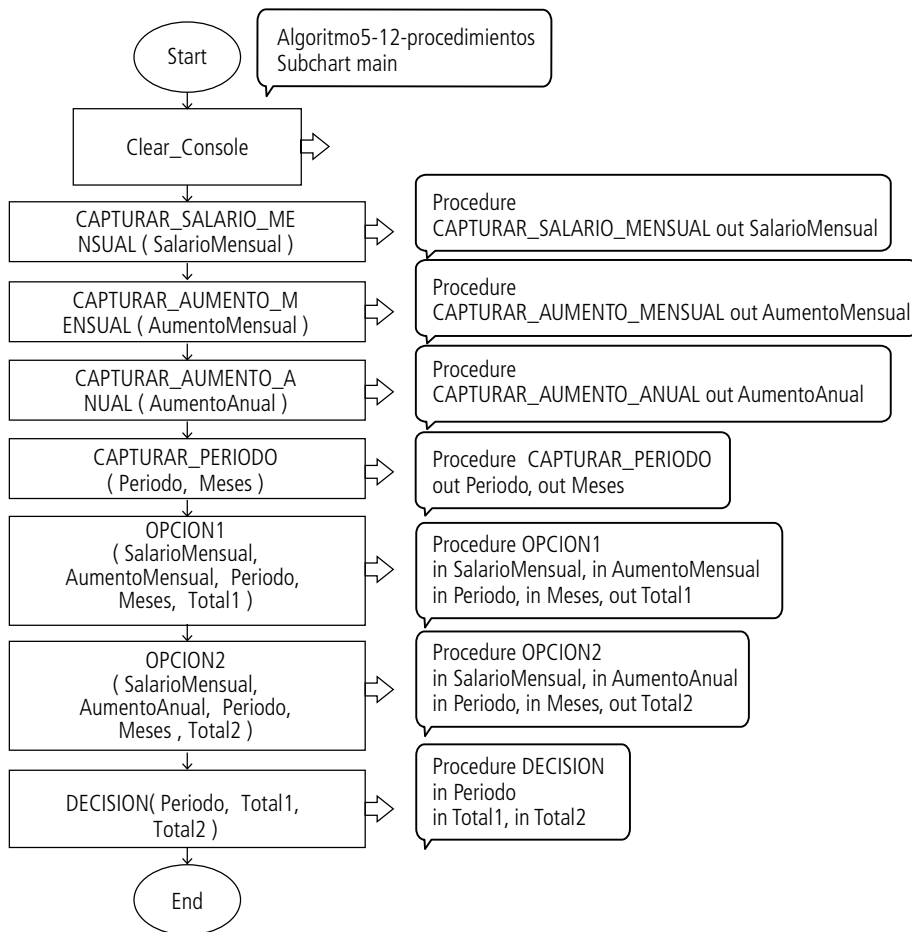


Figura 5.55 Algoritmo5-12-procedimientos *Subchart main*.

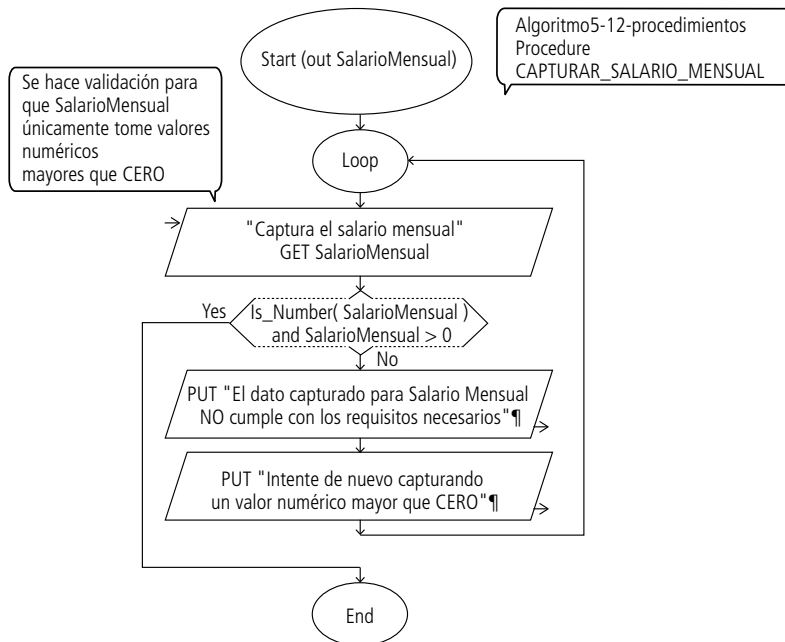


Figura 5.56 Algoritmo5-12-procedimientos *Procedure* CAPTURAR_SALARIO_MENSUAL.

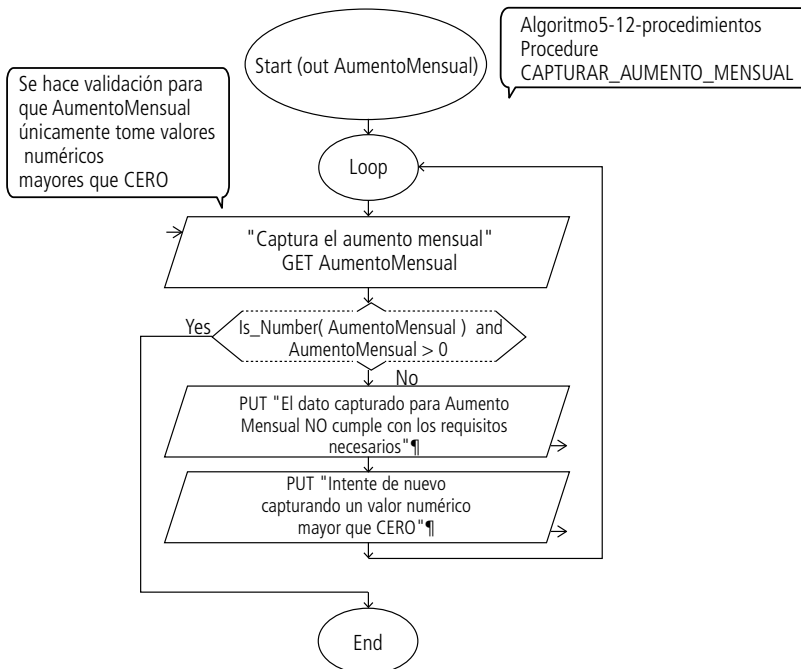


Figura 5.57 Algoritmo5-12-procedimientos *Procedure* CAPTURAR_AUMENTO_MENSUAL.

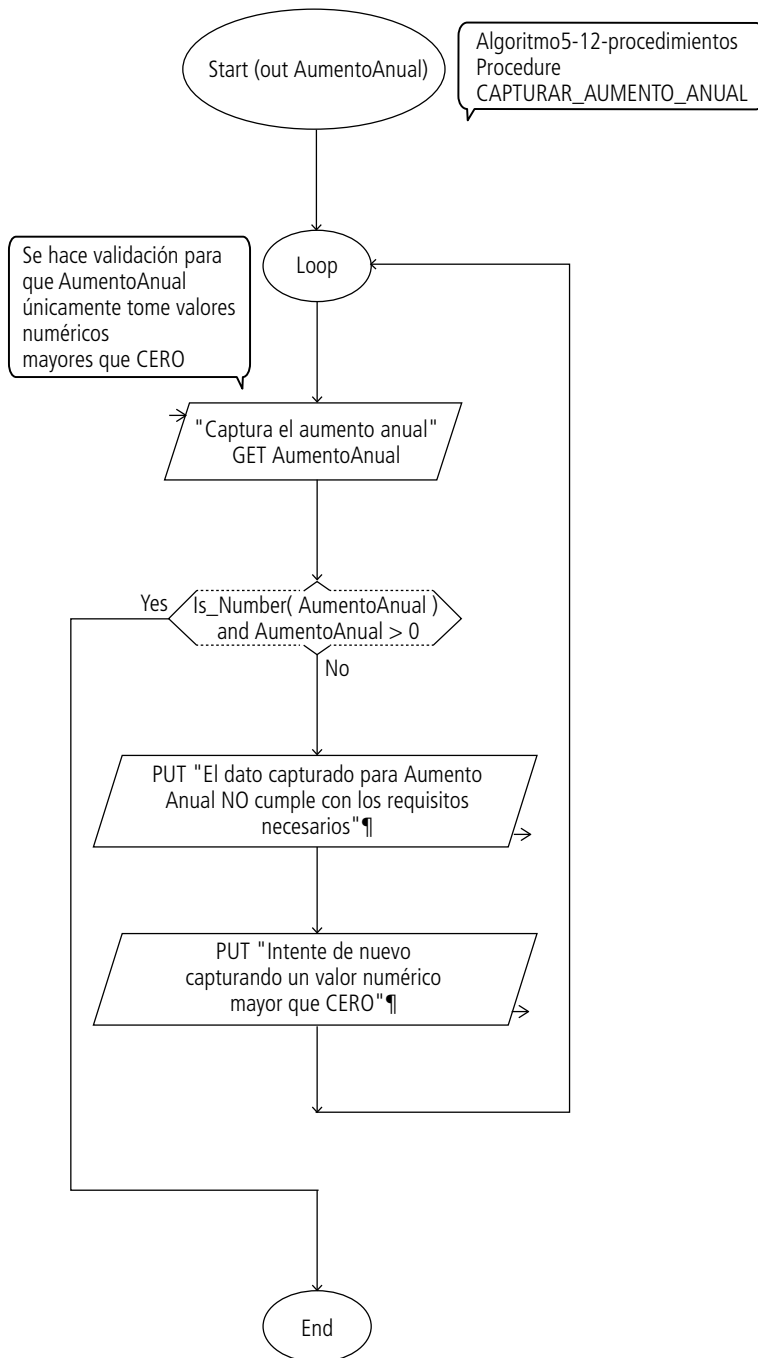


Figura 5.58 Algoritmo5-12-procedimientos *Procedure* CAPTURAR_AUMENTO_ANUAL.

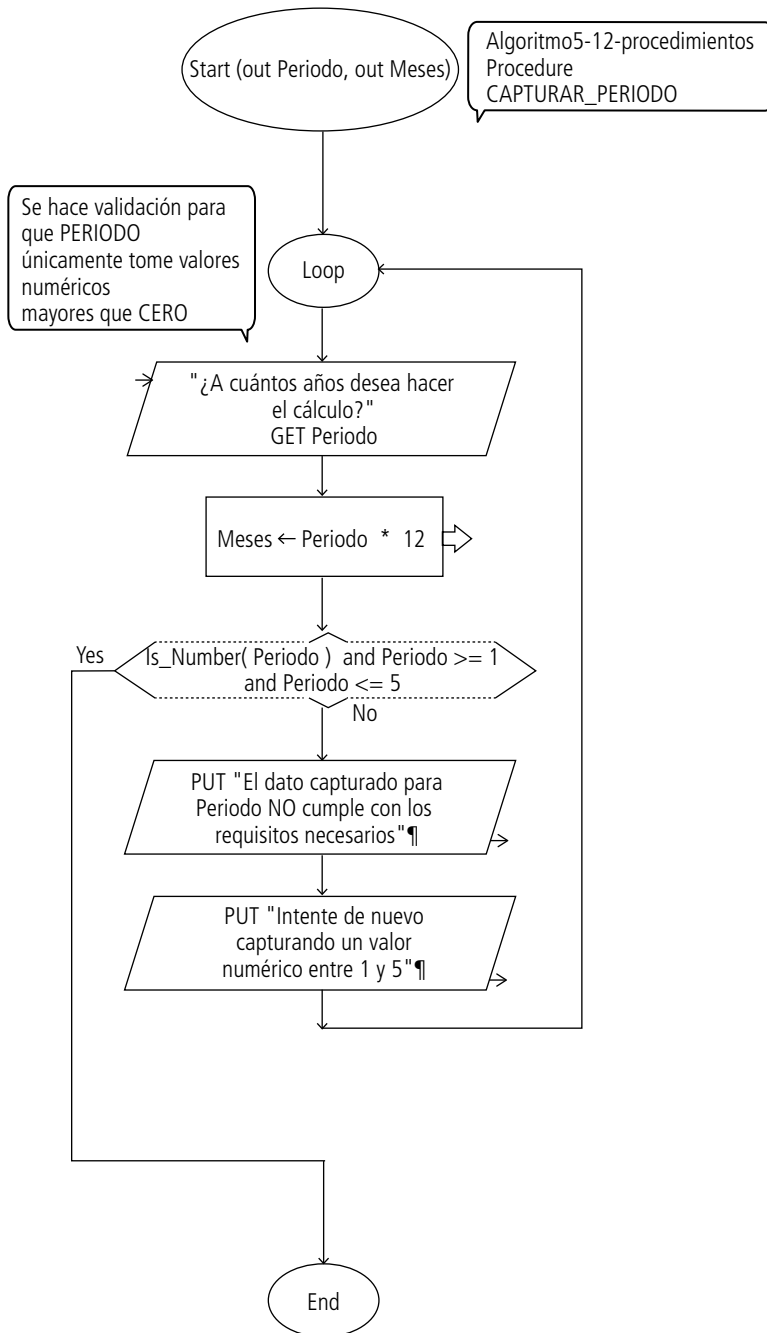


Figura 5.59 Algoritmo5-12-procedimientos *Procedure* CAPTURAR_PERIODO.

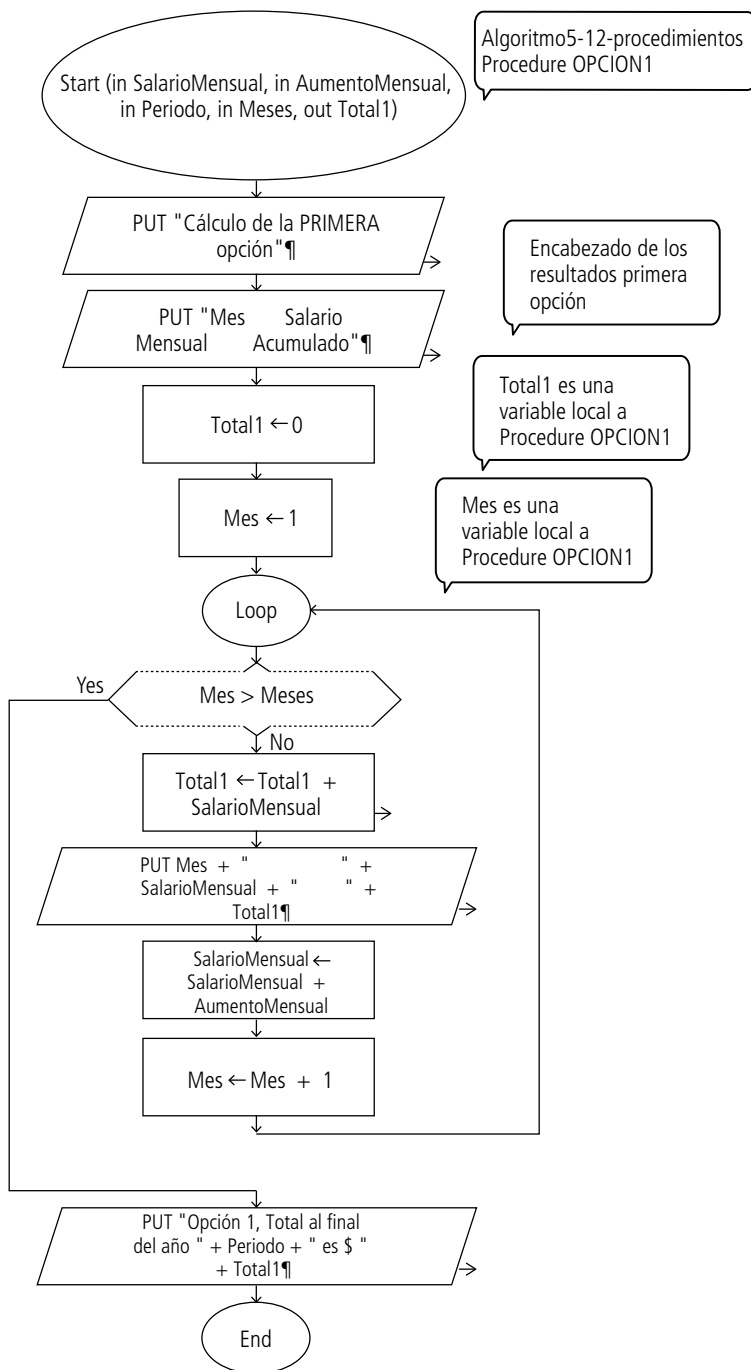


Figura 5.60 Algoritmo5-12-procedimientos Procedure OPCION1.

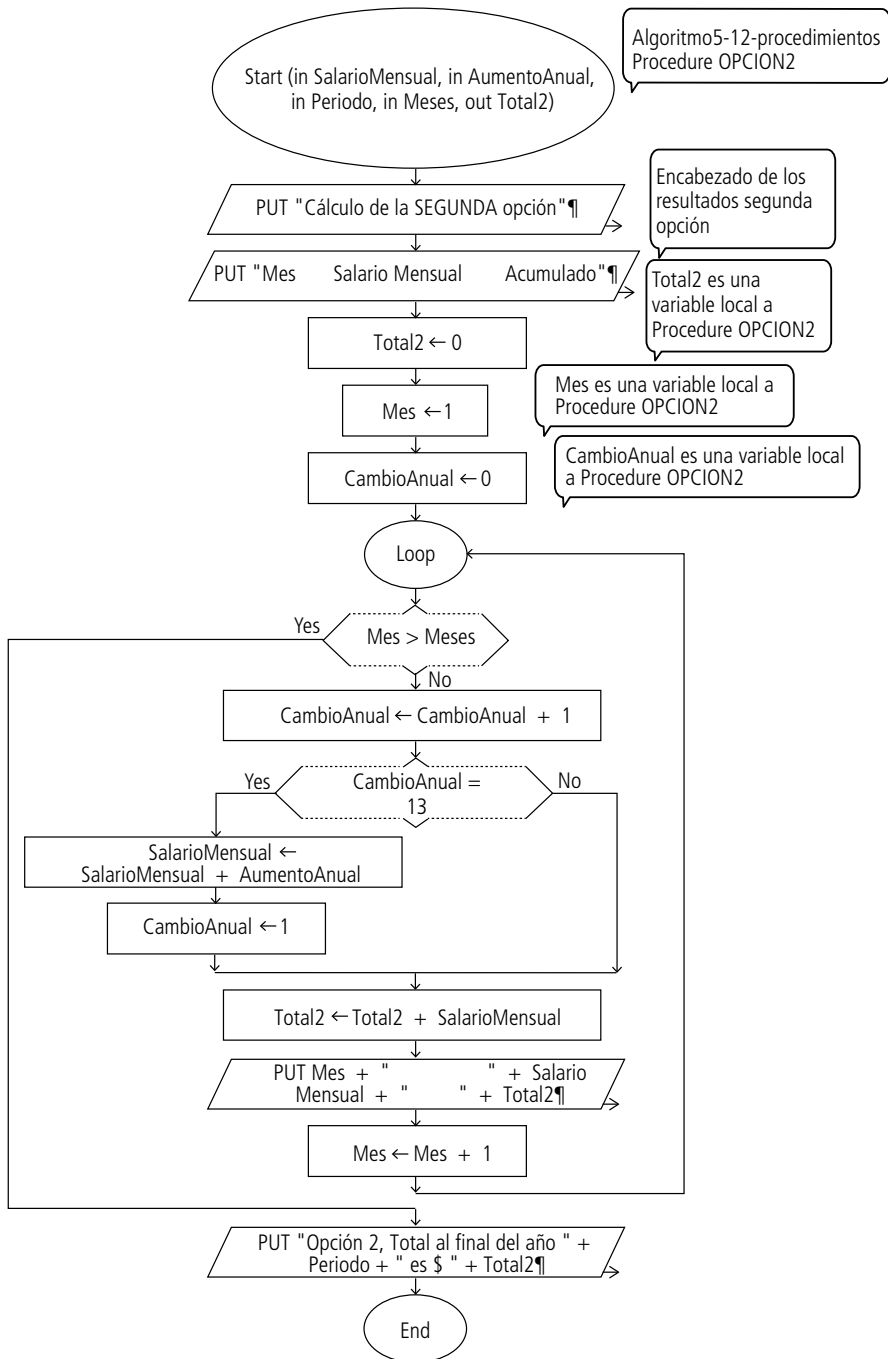


Figura 5.61 Algoritmo5-12-procedimientos *Procedure* OPCION2.

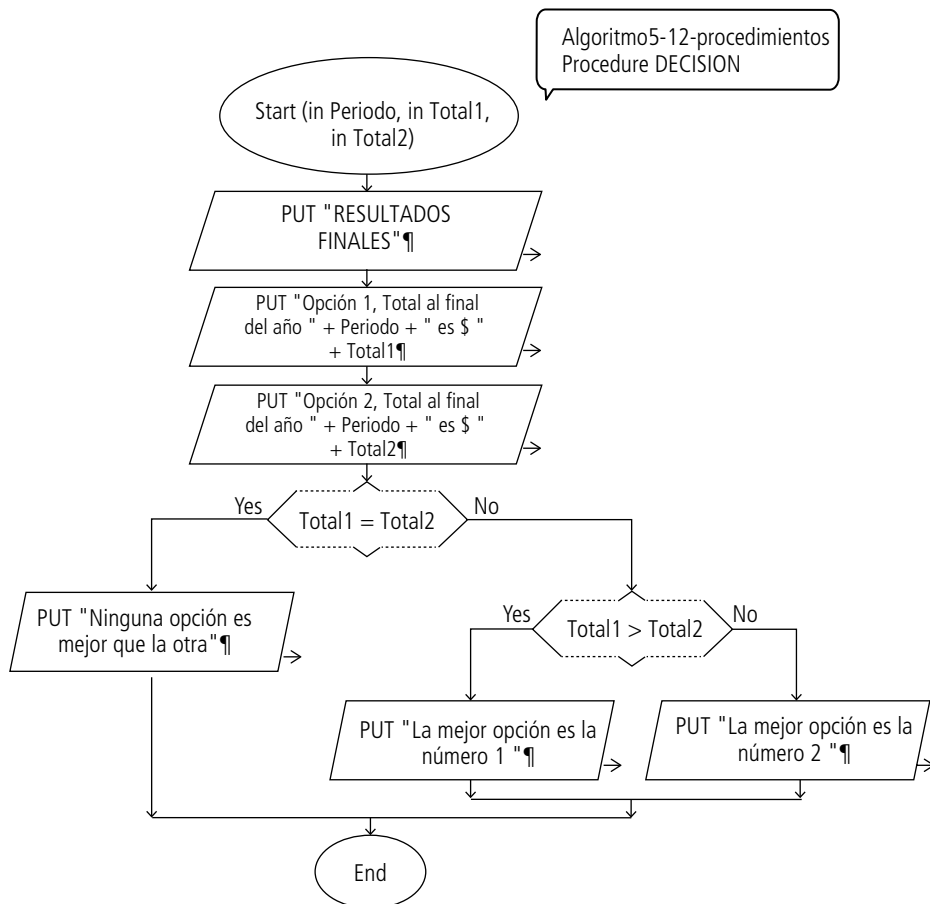


Figura 5.62 Algoritmo5-12-procedimientos *Procedure DECISION*.

Con el diagrama del **Algoritmo5-13-procedimientos** podemos saber si tres medidas lineales pueden formar un triángulo. El teorema de desigualdad triangular indica que “En todo triángulo la suma de las longitudes de dos lados cualquiera es siempre mayor a la longitud del lado restante”.

En este algoritmo, el usuario deberá capturar las medidas lineales de los tres lados con las mismas unidades de medición, ya sea metros, centímetros, pulgadas, yardas, etc. Los valores capturados se agregarán a tres variables diferentes. Aún y cuando el usuario utilizara las mismas medidas lineales para los tres lados, si partimos de la base que el orden en el que se capturarán puede variar, entonces, la cantidad de posibles escenarios diferentes sólo se puede determinar aplicando una fórmula de permutaciones matemáticas.

De esta manera, cuando se trata de los valores de un triángulo escaleno, las posibles permutaciones pueden ser seis. Para un triángulo isósceles pueden ser tres y

para un triángulo equilátero sólo uno. Vea la tabla 5.8. Considere que estas mismas permutaciones se pueden dar para valores que no forman un triángulo.

Tabla 5.8 Permutaciones de mismos valores de medidas lineales asignados a variables para un triángulo

Lado1	Lado2	Lado3	¿Forman triángulo?	Tipo de triángulo
5	7	9	Sí	Escaleno
5	9	7	Sí	Escaleno
7	5	9	Sí	Escaleno
7	9	5	Sí	Escaleno
9	5	7	Sí	Escaleno
9	7	5	Sí	Escaleno
5	5	9	Sí	Isósceles
5	9	5	Sí	Isósceles
9	5	5	Sí	Isósceles
5	5	5	Sí	Escaleno
Valores semejantes podrían mostrarse sin que formen un triángulo.				

Nuestro algoritmo se basa en una simplificación heurística del teorema de desigualdad triangular, derivada de la observación y el análisis lógico. En esta simplificación generalizada se establece que para saber si tres medidas lineales forman un triángulo, se deben sumar las dos medidas lineales más pequeñas y comparar contra la medida lineal más larga. Si la sumatoria de las medidas lineales más cortas es mayor que la medida lineal más larga, entonces forman un triángulo.

Sin embargo, la simplificación generalizada es válida cuando el triángulo es escaleno, porque cuando el triángulo es isósceles se deben considerar dos posibilidades:

1. Que el lado desigual sea menor que los dos iguales.
2. Que el lado que desigual sea mayor que los dos iguales.

En el primer caso, cualquier lado igual se toma como el más largo, así que se suma uno de los lados iguales con el lado desigual y se compara con el largo del otro lado igual. En el segundo caso, se suman los dos lados iguales y se compara con el lado desigual. Independientemente de cualquiera de las dos posibilidades, la conclusión de la simplificación generalizada es válida, la cual ayudará a determinar si las medidas lineales pueden o no formar un triángulo.

Por otro lado, la simplificación generalizada también tiene consideraciones a tomar en cuenta si el triángulo es equilátero. En este caso, cualquier lado se toma como el más largo, y los otros dos lados se suman. Luego, para saber si las medidas lineales forman un triángulo se compara la sumatoria de dos de los lados contra la medida lineal del tercer lado. En este caso, la conclusión de la simplificación generalizada sigue

siendo válida y ayudará a determinar si las medidas lineales pueden o no formar un triángulo.

Toda la explicación anterior se expresa de manera computacional en el algoritmo siguiente y con el objetivo de que el diagrama pueda visualizarse mejor, hemos permitido que variables de este algoritmo no cumplan con las reglas de calidad. Este diagrama tiene siete módulos, además de *main*, que son: **CAPTURAR_MEDIDA_LINEAL**, **EQUILATERO**, **ISOSCELES1**, **ISOSCELES2**, **ISOSCELES3**, **ESCALENO** y **DECISION**. Vea la tabla 5.9 para distinguir la función, el producto generado y los parámetros de cada módulo.

Tabla 5.9 Planeación de los módulos para el **Algoritmo5-13-procedimientos**

Módulo	Función	Producto generado	Parámetros
CAPTURAR_MEDIDA_LINEAL	Permitir al usuario que capture un valor para la variable Medida Lineal correspondiente a la longitud de uno de los lados del triángulo.	La variable L1, L2 y L3 , con un valor numérico mayor que cero.	Lado es un parámetro de entrada y de salida. Medida Lineal es parámetro de salida.
EQUILATERO	Recibir el valor asignado a L1, L2 y L3, y verificar si las tres medidas lineales son iguales.	Definir a un triángulo como equilátero y ejecutar el módulo DECISION .	L1, L2 y L3 son parámetros de entrada.
ISOSCELES1	Recibir el valor capturado por el usuario de L1, L2 y L3, y verificar si dos medidas lineales son iguales y una diferente.	Definir a un triángulo como isósceles y ejecutar el módulo DECISION .	L1, L2 y L3 son parámetros de entrada.
ISOSCELES2	Recibir el valor capturado por el usuario de L1, L2 y L3, y verificar si dos medidas lineales son iguales y una diferente.	Definir a un triángulo como isósceles y ejecutar el módulo DECISION .	L1, L2 y L3 son parámetros de entrada.
ISOSCELES3	Recibir el valor capturado por el usuario de L1, L2 y L3, y verificar si dos medidas lineales son iguales y una diferente.	Definir a un triángulo como isósceles y ejecutar el módulo DECISION .	L1, L2 y L3 son parámetros de entrada.
ESCALENO	Recibir el valor capturado por el usuario de L1, L2 y L3, y verificar si las tres medidas lineales son diferentes.	Definir a un triángulo como escaleno y ejecutar el módulo DECISION .	L1, L2 y L3 son parámetros de entrada.
DECISION	Recibir el valor capturado por el usuario de L1, L2, L3, Lados, Largo y Tipo .	Mostrar en <i>MasterConsole</i> los resultados del problema en mensajes de contextualización.	L1, L2, L3, Lados, Largo y Tipo son parámetros de entrada.
main	Controlar el programa y servir como receptor-emisor del valor de los parámetros de los <i>Procedure</i> .	Permitir que el diagrama se ejecute con lógica computacional.	

Primero, la función de `CAPTURAR_MEDIDA_LINEAL` permite que el usuario capture la longitud de uno de los lados del triángulo. Segundo, la función de `EQUILATERO` es tomar el valor capturado por el usuario de las longitudes de cada uno de los lados del triángulo y verificar si las medidas lineales son iguales. Tercero, la función de `ISOSCELES1`, `ISOSCELES2` e `ISOSCELES3` es recibir el valor capturado por el usuario de las longitudes de cada uno de los lados del triángulo y verificar si dos medidas lineales son iguales y una diferente. Cuarto, la función de `ESCALENO` es tomar el valor capturado por el usuario de las longitudes de cada uno de los lados del triángulo y verificar si las tres medidas lineales son diferentes. Quinto, la función de `DECISION` es considerar el valor capturado por el usuario de las longitudes de cada uno de los lados del triángulo y los valores calculados correspondiente a la sumatoria de los dos lados más cortos y el lado más largo del triángulo. Por último, la función del *Subchart main* es la de controlar el programa y servir como receptor-emisor de los parámetros de los *Procedure*.

El producto generado de `CAPTURAR_MEDIDA_LINEAL` son las variables `L1`, `L2` y `L3` que corresponden a la longitud de cada uno de los lados del triángulo con un valor que cumple con los requerimientos del problema. Se captura un valor a la variable Medida Lineal cuyos valores se envían a *main* y se asignan a `L1`, `L2` o `L3`, según sea el módulo que se ejecute. Medida Lineal es una variable local a `CAPTURAR_MEDIDA_LINEAL`. Este módulo es un *Procedure REUTILIZABLE* porque se utiliza para asignar valores a diferentes variables.

El producto generado de **`EQUILATERO`**, **`ISOSCELES1`**, **`ISOSCELES2`**, **`ISOSCELES3`** y **`ESCALENO`** es la definición del tipo de triángulo que podrían formar las tres medidas lineales capturadas por el usuario y la llamada al módulo `DECISION`.

El producto generado de `DECISION` es la definición sobre si las tres longitudes lineales forman o no un triángulo y Mostrar en *MasterConsole* los resultados del problema en mensajes de contextualización.

El módulo `EQUILATERO` se ejecuta cuando las tres longitudes son iguales. El módulo `ISOSCELES1` se ejecuta cuando las longitudes 1 y 2 son iguales, pero la longitud 3 es diferente. El módulo `ISOSCELES2` se ejecuta cuando las longitudes 1 y 3 son iguales, pero la longitud 2 es diferente. El módulo `ISOSCELES3` se ejecuta cuando las longitud 2 y 3 son iguales, pero la longitud 1 es diferente y el módulo `ESCALENO` se ejecuta cuando las tres longitudes son diferentes (véanse figuras 5.63 a 5.70).

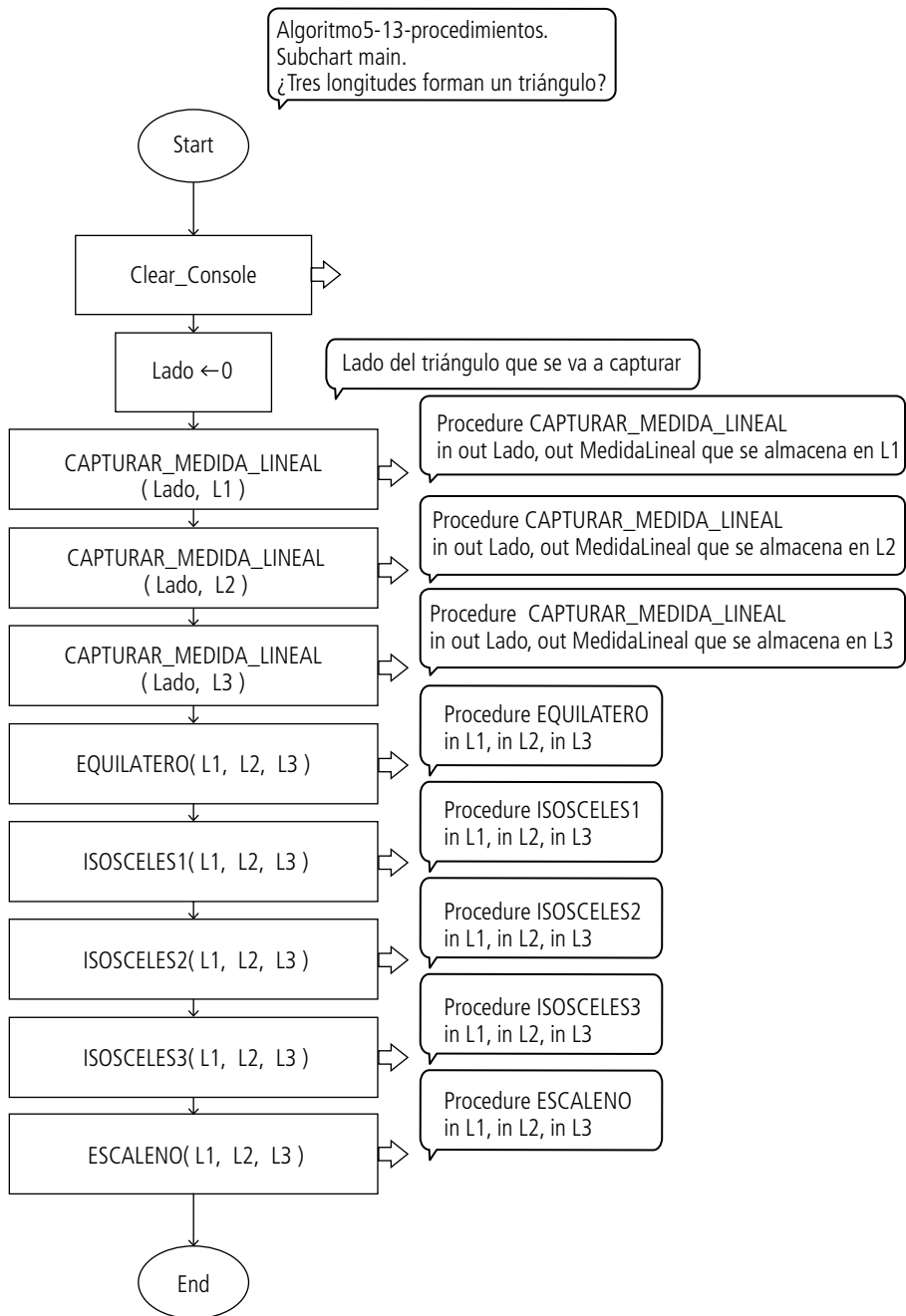


Figura 5.63 Algoritmo5-13-procedimientos *Subchart main*.

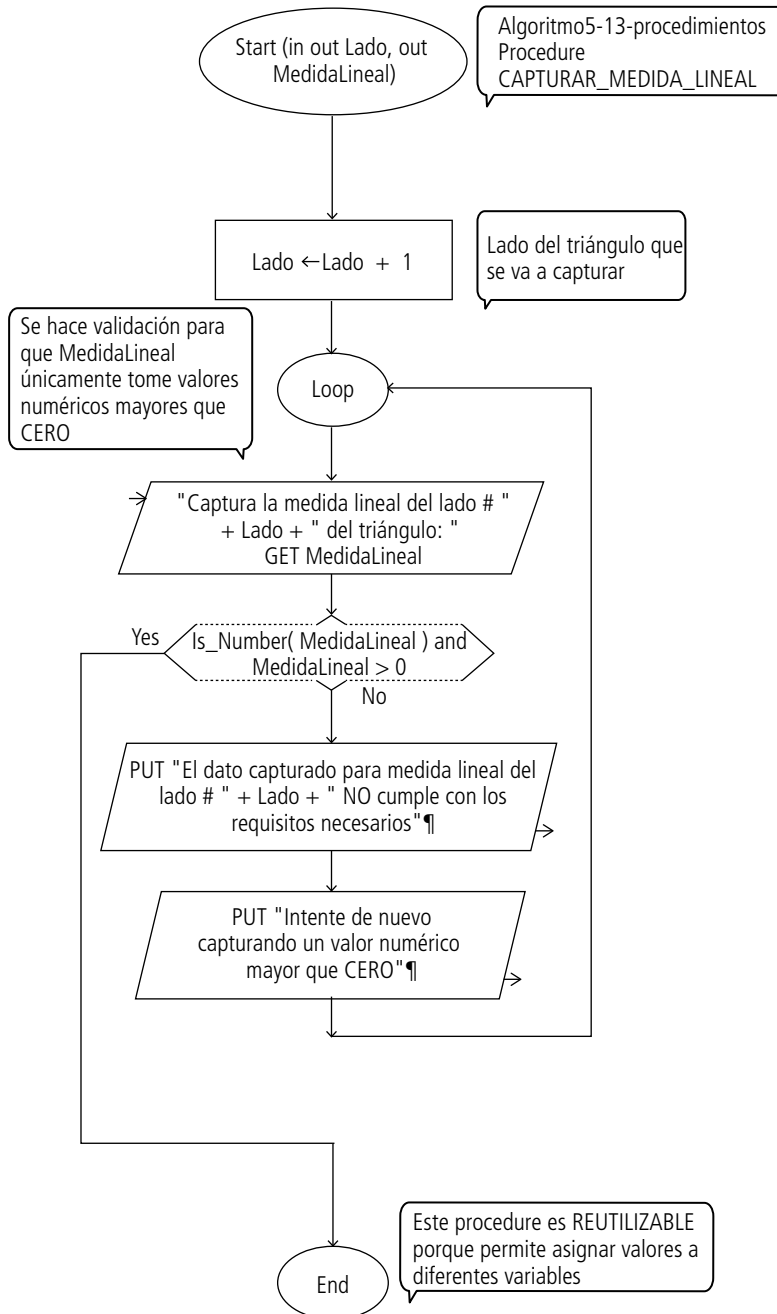


Figura 5.64 Algoritmo5-13-procedimientos *Procedure* CAPTURAR_MEDIDA_LINEAL.

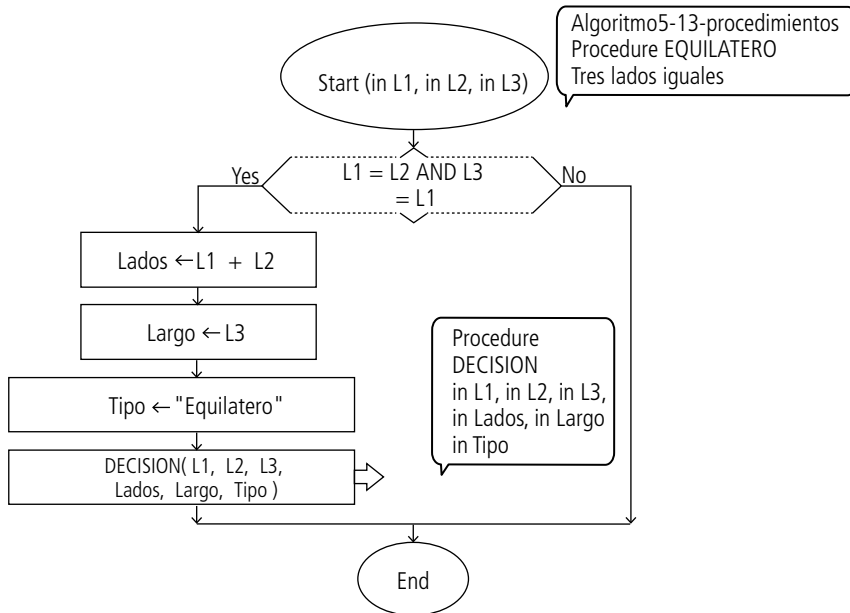


Figura 5.65 Algoritmo5-13-procedimientos *Procedure EQUILATERO*.

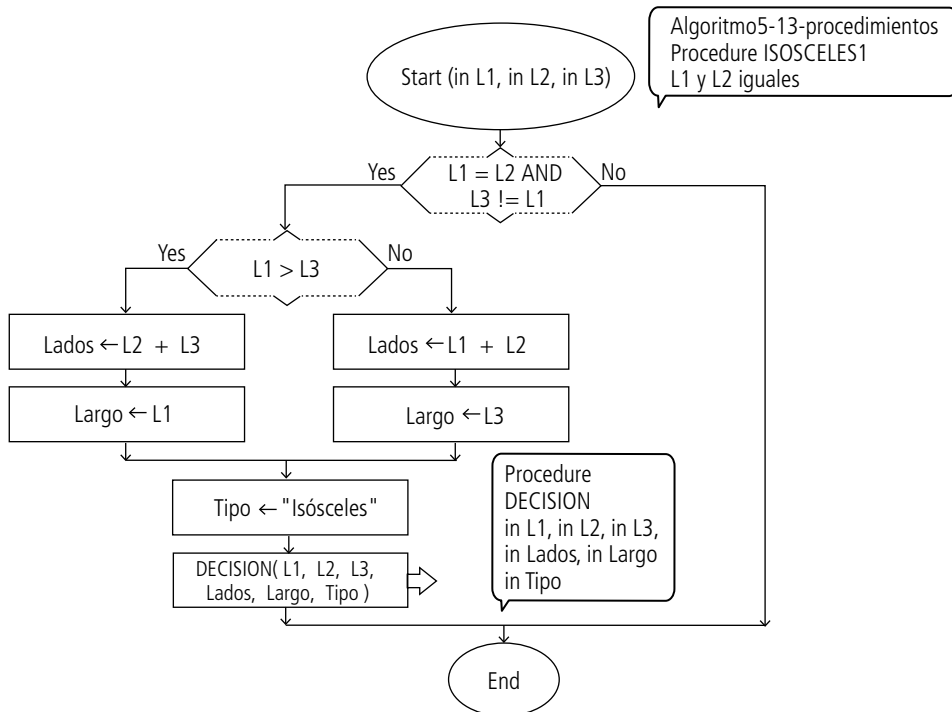
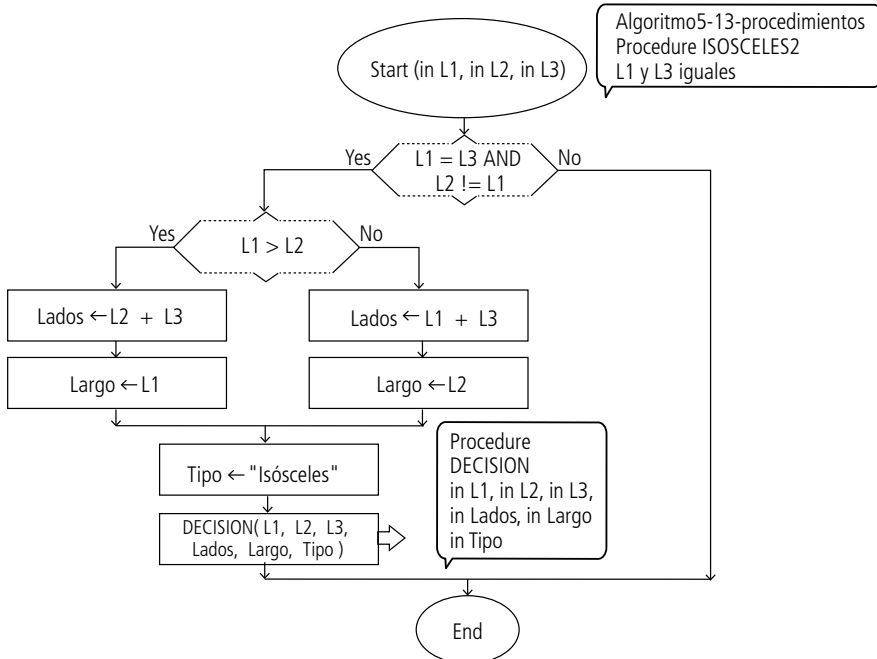
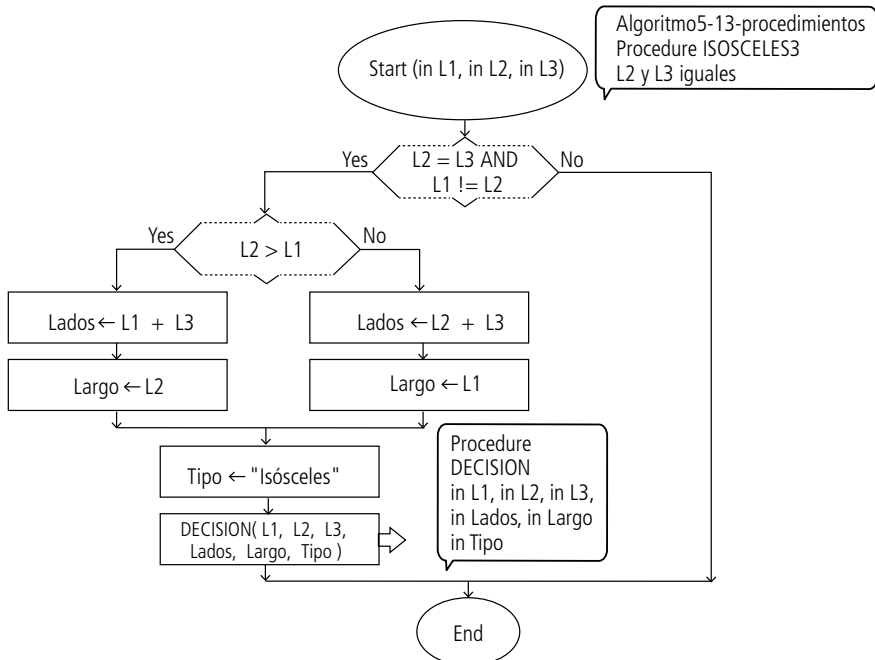


Figura 5.66 Algoritmo5-13-procedimientos *Procedure ISOSCELES1*.

Figura 5.67 Algoritmo5-13-procedimientos *Procedure ISOSCELES2*.Figura 5.68 Algoritmo5-13-procedimientos *Procedure ISOSCELES3*.

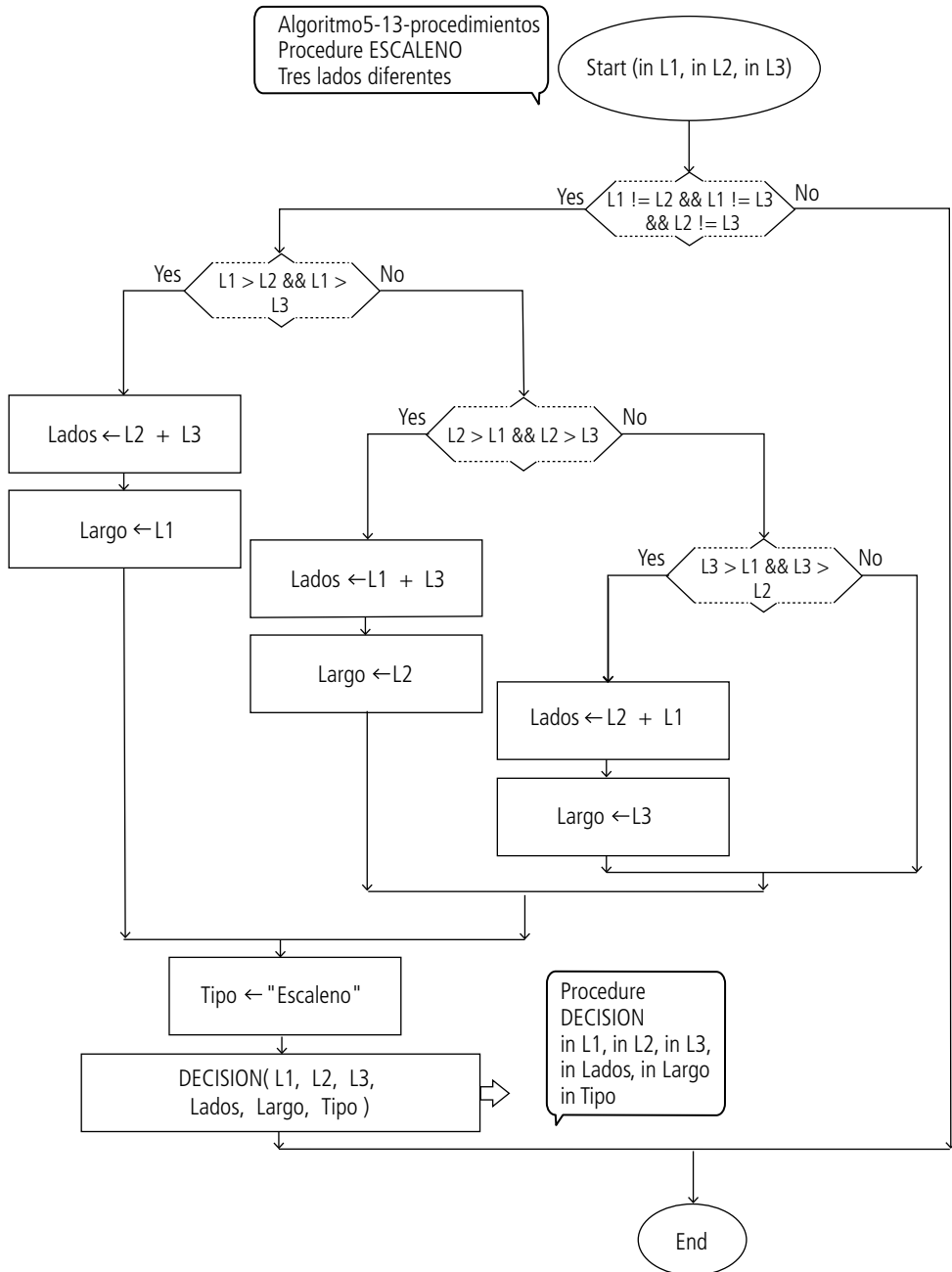


Figura 5.69 Algoritmo5-13-procedimientos *Procedure ESCALENO*.

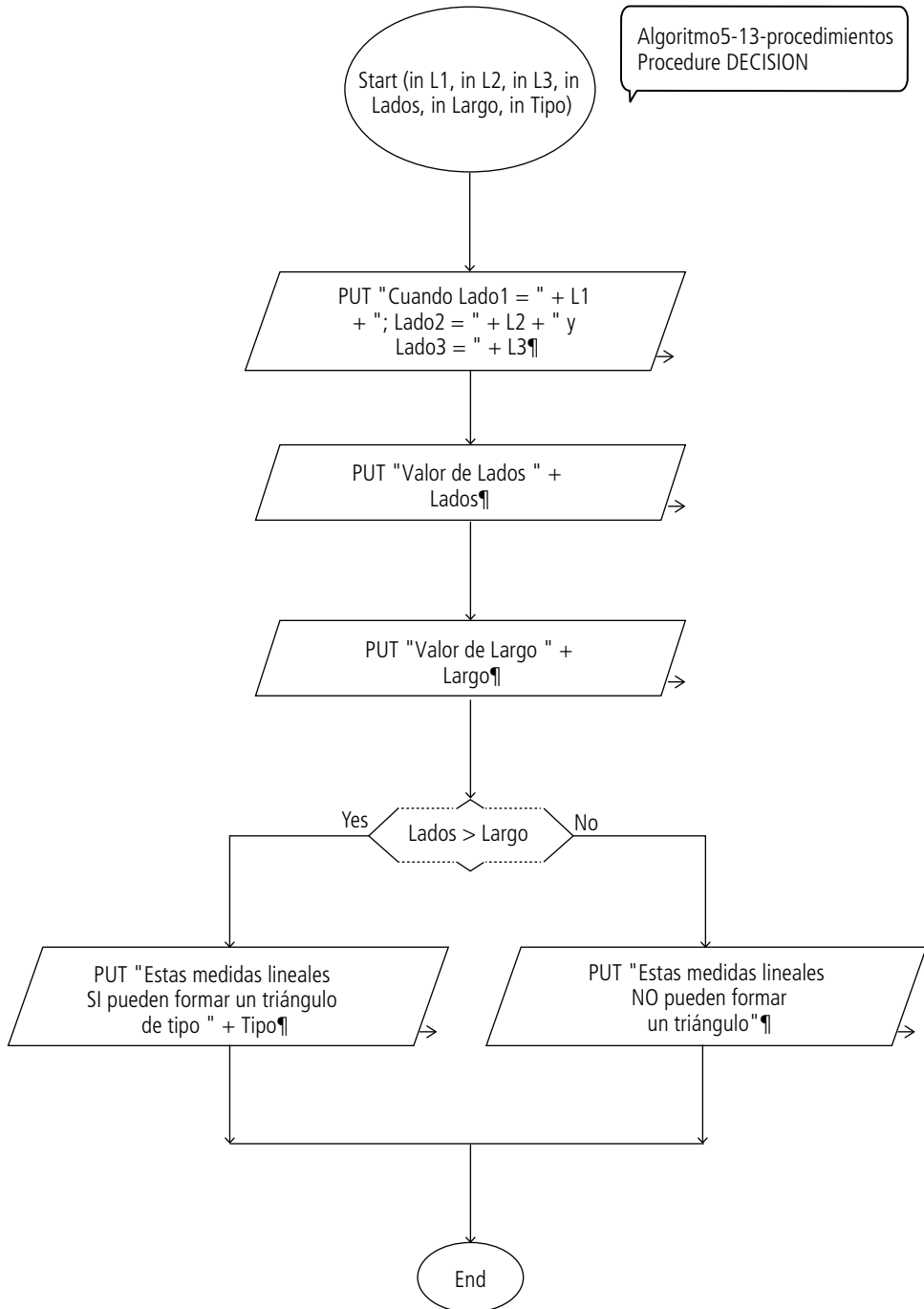


Figura 5.70 Algoritmo5-13-procedimientos *Procedure DECISION*.



5.4 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 5.10.

- 5.1 ¿Cómo se nombra al conjunto de tareas pequeñas que abarca una o varias acciones computacionales y que en su conjunto realizan una función específica, si este conjunto de tareas es una parte de un algoritmo computacional más grande?
- 5.2 ¿Cómo se llaman los algoritmos que no están modularizados?
- 5.3 ¿Cómo se denominan los algoritmos que están modularizados?
- 5.4 ¿Cómo se designan los valores de los identificadores que se transmiten entre módulos?
- 5.5 ¿Cómo se nombran los parámetros emitidos por un módulo?
- 5.6 ¿Cómo se llaman los parámetros recibidos por un módulo?
- 5.7 ¿Cómo se designa los módulos en Raptor que no pueden manipular parámetros?
- 5.8 ¿Cómo se denominan los módulos en Raptor que manipulan parámetros?
- 5.9 ¿Cómo se llaman los valores de las variables y constantes en un algoritmo modularizado que exclusivamente cuenta con *Subchart*?
- 5.10 ¿Cómo se designa el valor de una variable o constante que puede ser utilizado por cualquier módulo sin necesidad de hacer explícito el envío o recepción del mismo?
- 5.11 ¿Cómo se denomina el valor de una variable o constante que se usa por primera vez en un módulo y su valor no puede ser empleado por otro módulo, a menos que se haga explícito la posibilidad de enviarlo o recibirlo?

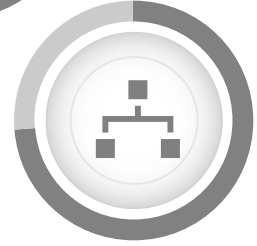
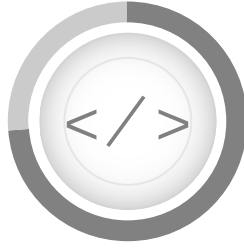
Tabla 5.10 Respuestas a los ejercicios de autoevaluación

5.1 Módulo	5.2 Algoritmo monolítico	5.3 Algoritmo funcionalizado
5.4 Parámetros	5.5 Parámetros de salida	5.6 Parámetros de entrada
5.7 <i>Subchart</i>	5.8 <i>Procedure</i>	5.9 Parámetros globales
5.10 Parámetro global	5.11 Parámetro local	



5.5 Problemas propuestos

Utilice los ejemplos de los capítulos anteriores y haga los diagramas usando módulos.



Capítulo 6

Diseño de algoritmos con arreglos de memoria

- 6.1 Introducción
- 6.2 Introducción a los arreglos de memoria unidimensionales
- 6.3 Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y sin paso de parámetros
- 6.4 Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y con paso de parámetros
- 6.5 Introducción a los arreglos de memoria bidimensionales
- 6.6 Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y sin paso de parámetros
- 6.7 Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y con paso de parámetros
- 6.8 Proceso de datos con corte de control
- 6.9 Ejercicios de autoevaluación
- 6.10 Problemas propuestos: arreglos de memorias unidimensionales y bidimensionales

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:

- Diseñar algoritmos computacionales que permitan el almacenamiento de un conjunto de datos relacionados y del mismo tipo en la memoria principal de la computadora.
- Comprender el funcionamiento de la estructura de datos de arreglos de memoria unidimensional.
- Comprender el funcionamiento de la estructura de datos de arreglos de memoria bidimensional.
- Procesar datos mediante corte de control.



6.1 Introducción

Cuando se tiene un conjunto de datos relacionados y del mismo tipo se recomienda almacenarlos en arreglos de memoria porque se facilita su procesamiento. Los datos son del mismo tipo cuando tienen iguales características; por ejemplo, todos ellos son números enteros, o todos son números decimales, etc. Los datos están relacionados cuando pertenecen a un mismo contexto de información, como las ventas diarias de un comercio en el mes de enero, caso para el cual tendríamos 31 datos numéricos que se refieren a una situación muy específica.

El procesamiento manual de un conjunto de datos relacionados suele ser monótono, repetitivo y proclive a que se cometan errores de cálculo, los cuales se minimizan cuando se utiliza un dispositivo electrónico rápido y confiable como la computadora. La programación de computadoras es una herramienta que permite resolver problemas con rapidez y precisión mediante la automatización de los cálculos matemáticos. Esto le permite al usuario concentrarse en el razonamiento de un problema y no en las operaciones aritméticas.

Aquellos profesionistas que en su trabajo cotidiano resuelven problemas por medio de múltiples cálculos matemáticos deben saber programar para identificar en qué momento es conveniente emplear un algoritmo computacional, que permita obtener resultados confiables, rápidos y precisos.

Un programador debe comprender las diversas opciones de almacenar datos en la memoria RAM de la computadora. Un arreglo de memoria es una estructura de datos que consiste en un grupo consecutivo de localidades de memoria RAM relacionadas por el hecho de tener el mismo nombre y características. En el ejemplo de las ventas en el mes de enero, las localidades se pueden concebir como celdas. Cada dato numérico se almacena en una celda, por tanto, tendríamos 31 celdas y en cada celda estaría almacenada la cifra correspondiente a la venta de un día específico.



6.2 Introducción a los arreglos de memoria unidimensionales

Los arreglos de memoria unidimensionales son una estructura de datos donde las localidades o celdas de la memoria RAM se identifican con un nombre de variable y un elemento único que se llama subíndice. Se pueden concebir como las celdas de una columna en una hoja electrónica, por ejemplo Excel. El nombre de la columna es el nombre de la variable y se puede referir a cada celda con el subíndice. Las celdas son contiguas y el subíndice indica el lugar o posición ordinal que ocupa cada una de ellas. De manera informal se les llama vectores.

Las variables que son arreglo de memoria unidimensional pueden distinguirse de las que no lo son porque su identificador se acompaña con corchetes a manera de sufijo, el que abre y cierra, y en medio de éstos se coloca el subíndice, el cual puede clasificarse en estáticos y dinámicos. Cuando el elemento que está entre corchetes es una constante, se considera como subíndice estático. Cuando el elemento que está entre corchetes es una variable, se considera como subíndice dinámico.

En el siguiente ejemplo podemos ver que la variable Vector cuenta con cinco celdas con las siguientes asignaciones:

Vector [1] = 227, Vector [2] = 139, Vector [3] = 417, Vector [4] = 252,
Vector [5] = 96 (véase tabla 6.1).

Tabla 6.1 Ejemplo de la variable vector

Subíndice	Vector
1	227
2	139
3	417
4	252
5	96

En el **Algoritmo6-1-vector** se muestra cómo pueden almacenarse datos, mediante la asignación, en las celdas de un arreglo de memoria unidimensional con subíndices estáticos (véanse figuras 6.1, 6.2 y 6.3).

Los subíndices también pueden ser dinámicos, tal y como se muestra en el **Algoritmo6-2-vector**. En la mayoría de los textos que muestran el uso de arreglos de memoria unidimensionales, el subíndice dinámico se representa con la variable *i*. Para no contrariar esa costumbre, aquí se hará lo mismo, no sin antes aclarar que el nombre de esa variable no cumple con las reglas de calidad (véanse figuras 6.4, 6.5 y 6.6).

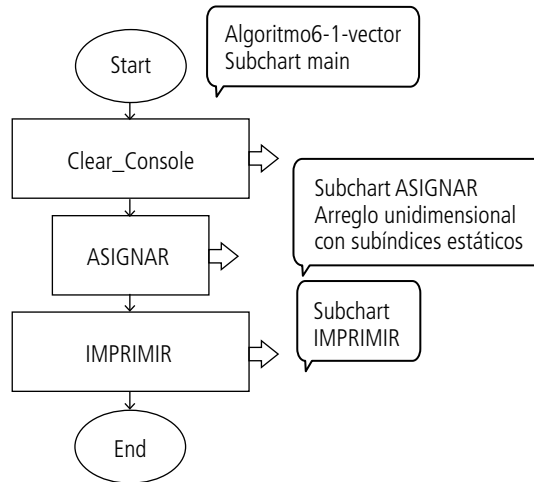


Figura 6.1 Algoritmo6-1-vector *Subchart main*.

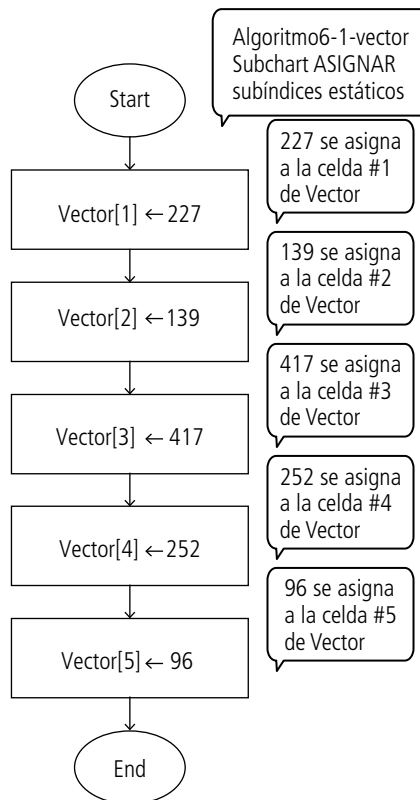


Figura 6.2 Algoritmo6-1-vector *Subchart ASIGNAR*.

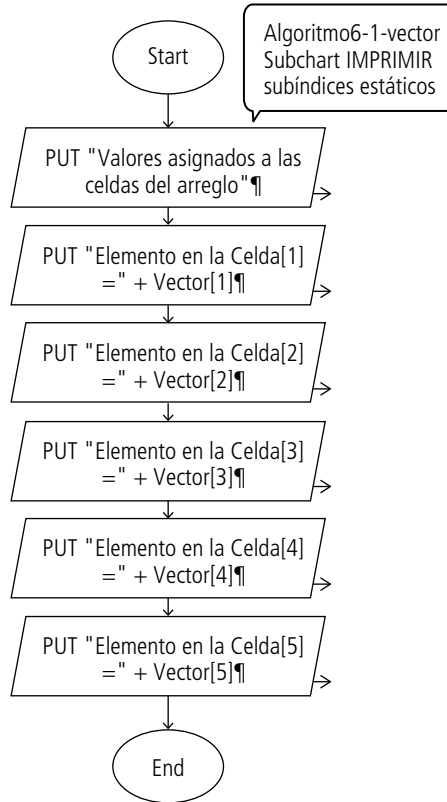


Figura 6.3 Algoritmo6-1-vector *Subchart IMPRIMIR*.

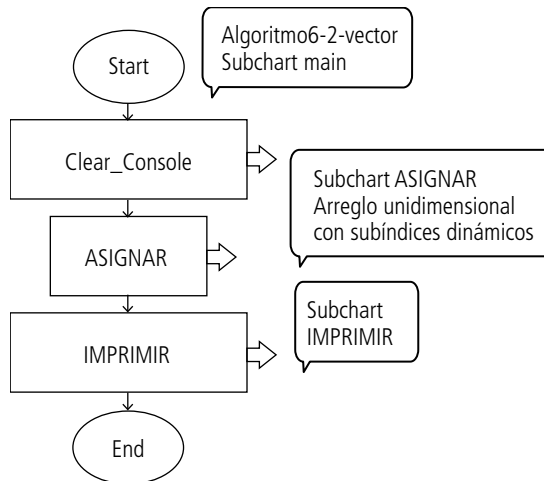


Figura 6.4 Algoritmo6-2-vector *Subchart main*.

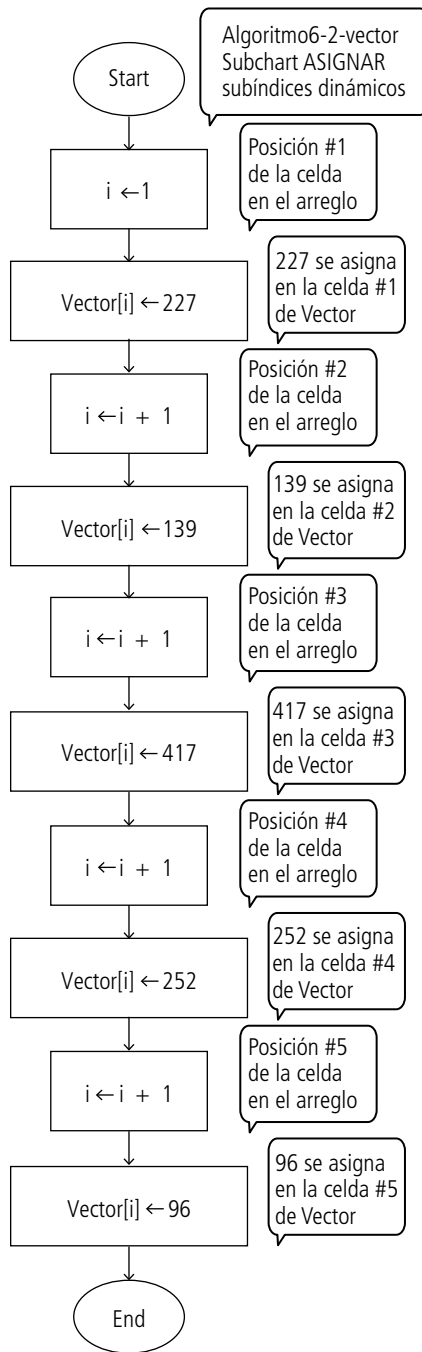


Figura 6.5 Algoritmo6-2-vector Subchart ASIGNAR.

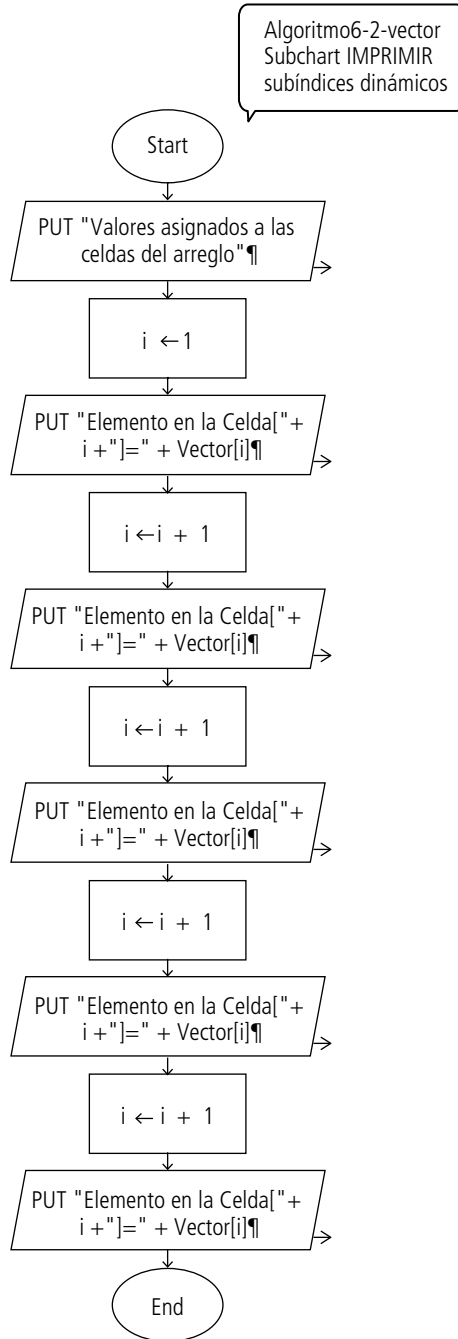


Figura 6.6 Algoritmo6-2-vector *Subchart* IMPRIMIR.

Con el **Algoritmo6-3-vector** se permite al usuario agregar, de manera manual, los elementos a un arreglo de memoria, usando el teclado de la computadora. Éste es un algoritmo rígido e inflexible porque sólo funciona para asignar cinco elementos en el arreglo. Si deseáramos aumentar o disminuir la cantidad de elementos del arreglo debe modificarse el diagrama. No se muestra el *subchart* IMPRIMIR porque es igual al ejemplo anterior (véanse figuras 6.7 y 6.8).

De manera, ideal los elementos que forman parte del mismo arreglo deben ser del mismo tipo de datos; es decir, todos deben ser numéricos o alfabéticos. Sin embargo, con este ejemplo podrá darse cuenta de que esto no es necesario. El problema se presentará cuando el usuario intente procesar los elementos del arreglo en una operación matemática. En este caso, si algún elemento del arreglo no es numérico lo más probable es que ocasione un error de ejecución. Para asegurarnos de que todos los elementos de un arreglo sean del mismo tipo, tendrá que hacer uso de la validación de datos que se vio en el capítulo 4.

Algoritmo6-4-vector es más flexible que el del ejemplo anterior porque se permite al usuario agregar de manera manual los elementos a un arreglo de memoria, utilizando el teclado de la computadora; sin embargo, el uso de la estructura repetitiva permite aumentar o disminuir, a discreción del usuario, la cantidad de elementos en el arreglo, sin necesidad de hacer adecuaciones al diagrama (véanse figuras 6.9 a 6.11).

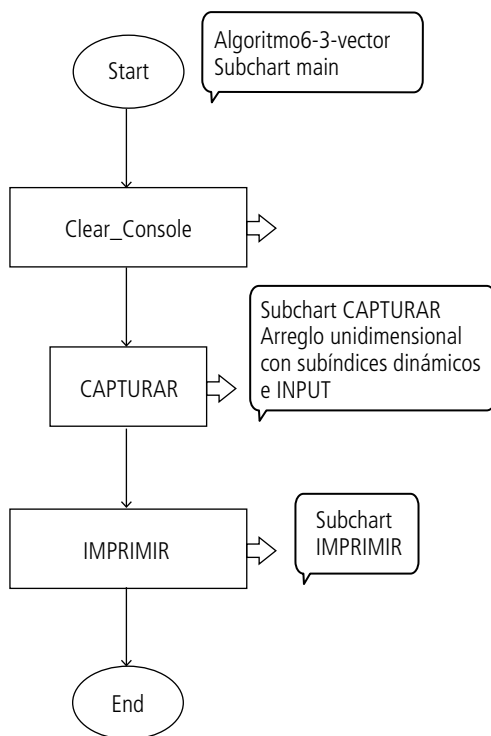


Figura 6.7 Algoritmo6-3-vector *Subchart main*.

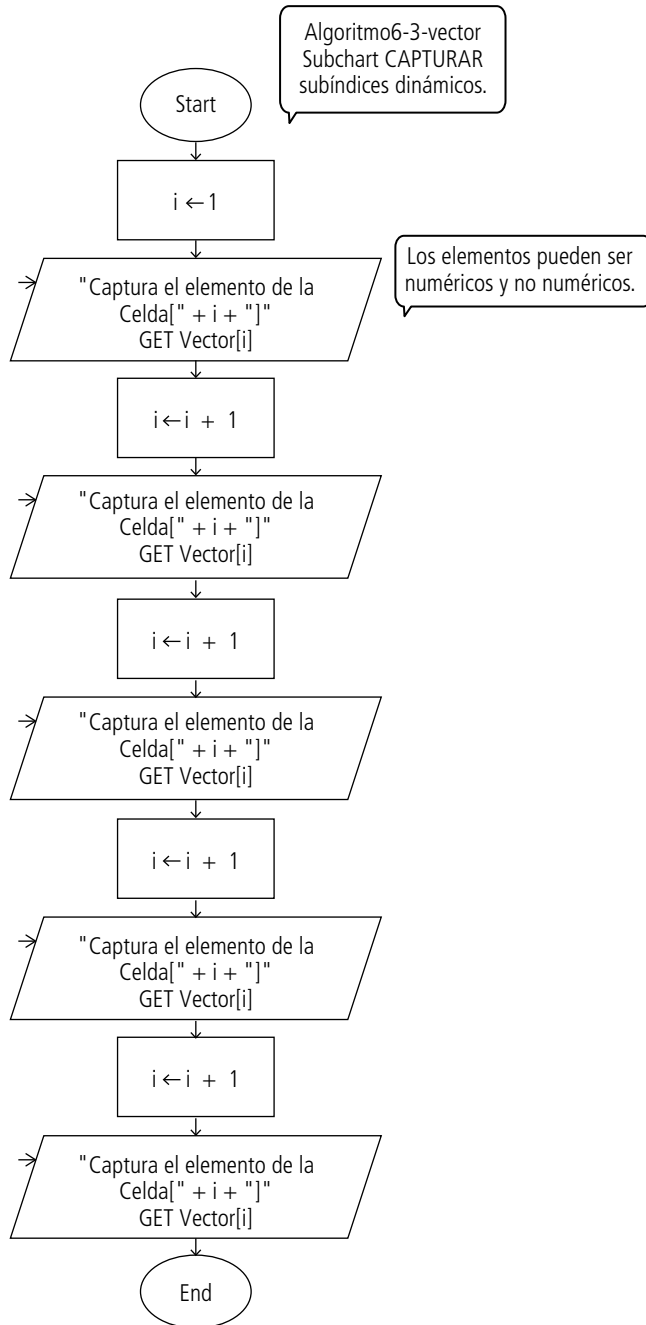


Figura 6.8 Algoritmo6-3-vector *Subchart* CAPTURAR.

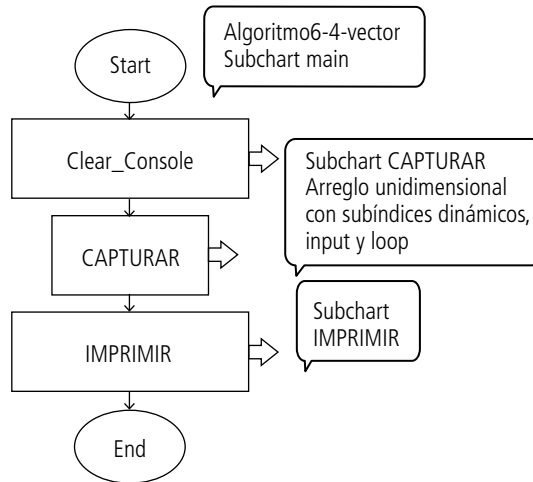


Figura 6.9 Algoritmo6-4-vector Subchart main.

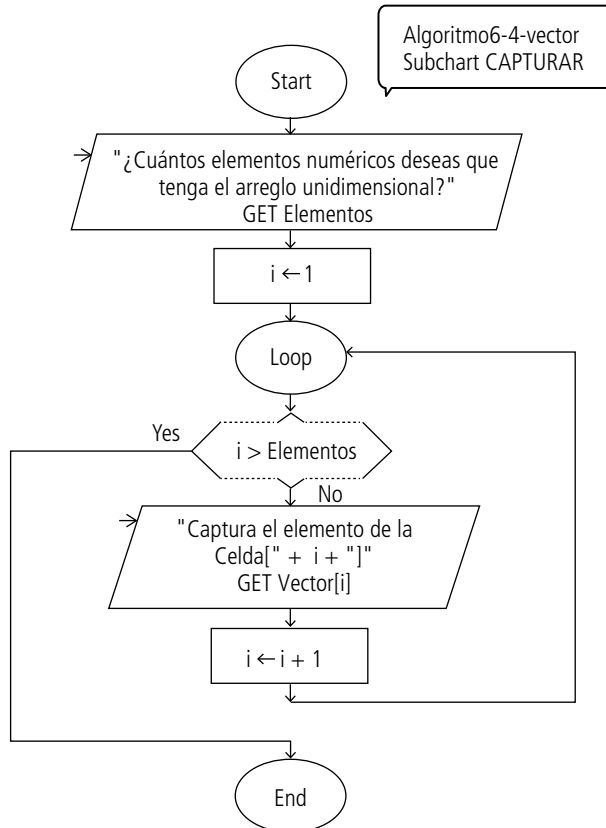


Figura 6.10 Algoritmo6-4-vector Subchart CAPTURAR.

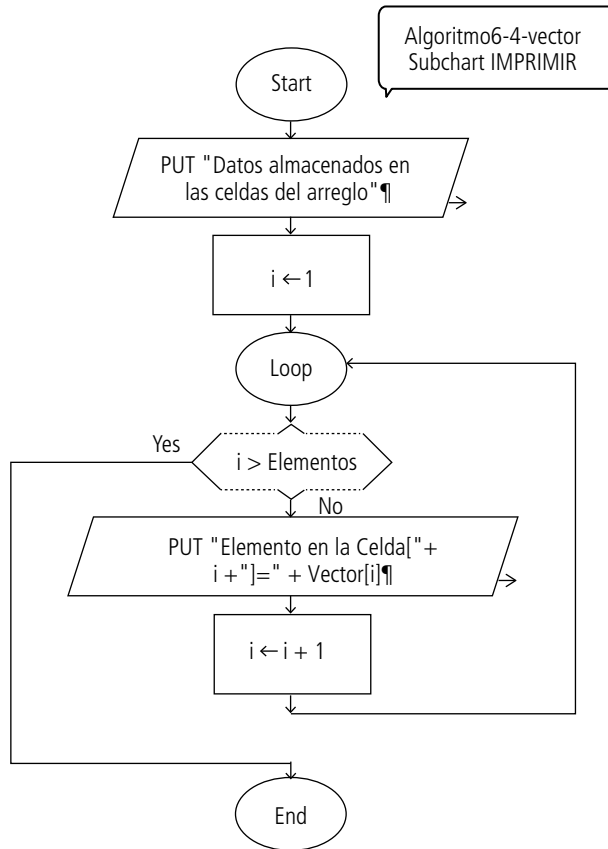


Figura 6.11 Algoritmo6-4-vector Subchart IMPRIMIR.



6.3 Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y sin paso de parámetros

En realidad, todos los diagramas anteriores de este capítulo son parte de este tema, por lo que sólo mostraremos un ejemplo más. En el **Algoritmo6-5-vector** se asignan automáticamente elementos numéricos y aleatorios a un arreglo de memoria llamado Vector, los cuales se suman. Con el módulo **ASIGNACION**, el usuario ya no tiene control sobre los números asignados en el arreglo, pues con la función de redondeo (*floor*), aleatorio (*random*) y su parámetro se generan números al azar entre uno y 999. Además, el módulo **ACUMULAR** permite sumar los elementos insertados en el arreglo; en este caso, a la variable sumatoria se le denomina variable acumuladora

(véanse figuras 6.12 a 6.15). Enseguida se muestra la representación matemática que permite sumar el contenido de las celdas del arreglo:

$$\text{Sumatoria} = \sum_{i=1}^{i=N} \text{Vector}[i]$$

La representación matemática de la sumatoria de los elementos celda por celda queda como sigue:

$$\text{Sumatoria} = \text{Vector}[1] + \text{Vector}[2] + \text{Vector}[3] + \text{Vector}[4] + \dots + \text{Vector}[N]$$

Tabla 6.2 Ejemplo de variable Vector con N elementos

Subíndice	Vector
1	227
2	139
3	417
4	252
5	96
	...
	...
	...
N	415

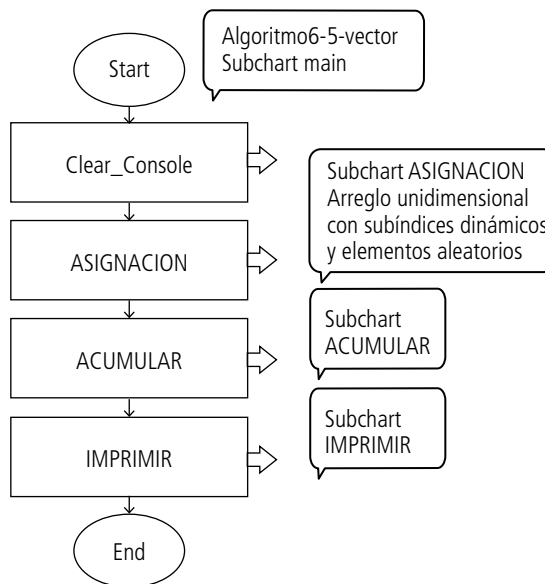
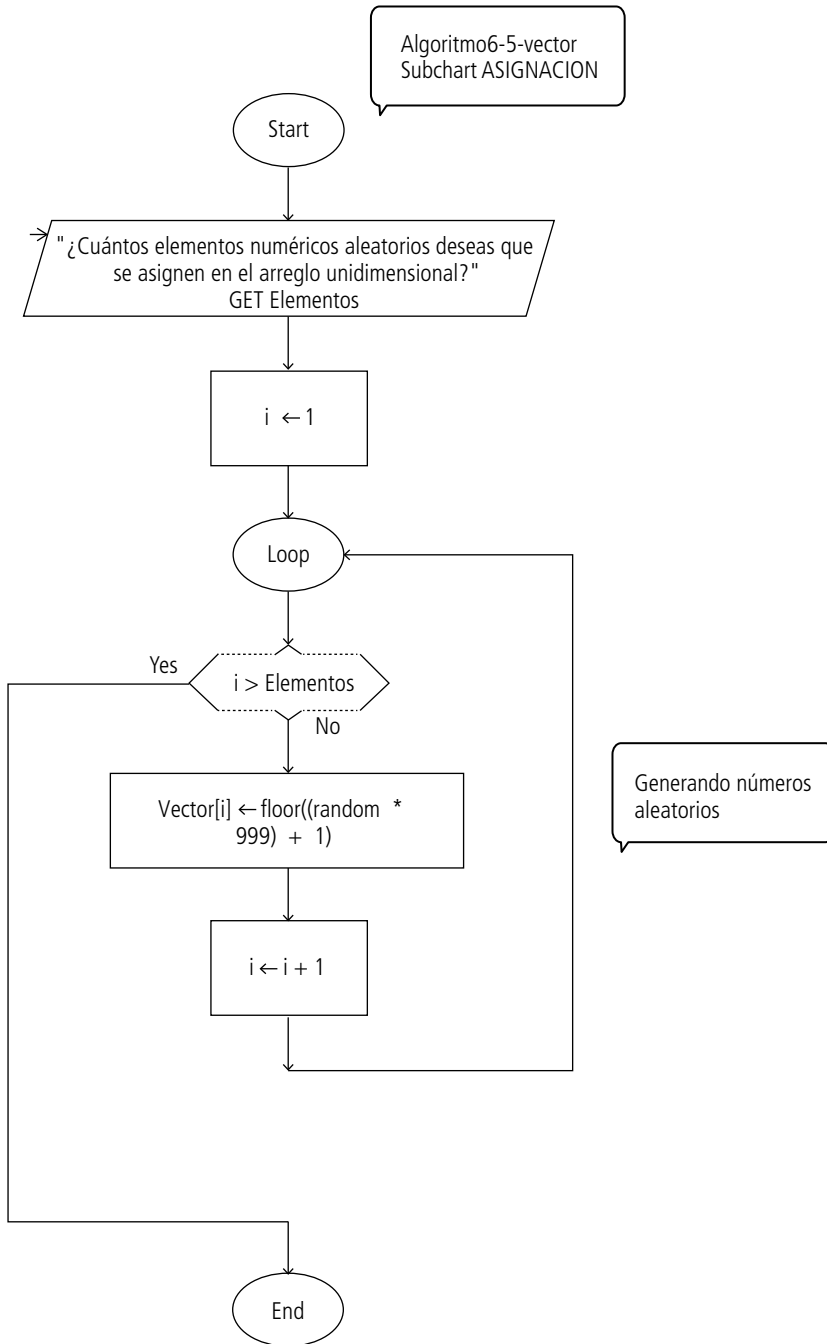


Figura 6.12 Algoritmo6-5-vector *Subchart main*.

**Figura 6.13** Algoritmo6-5-vector Subchart ASIGNACION.

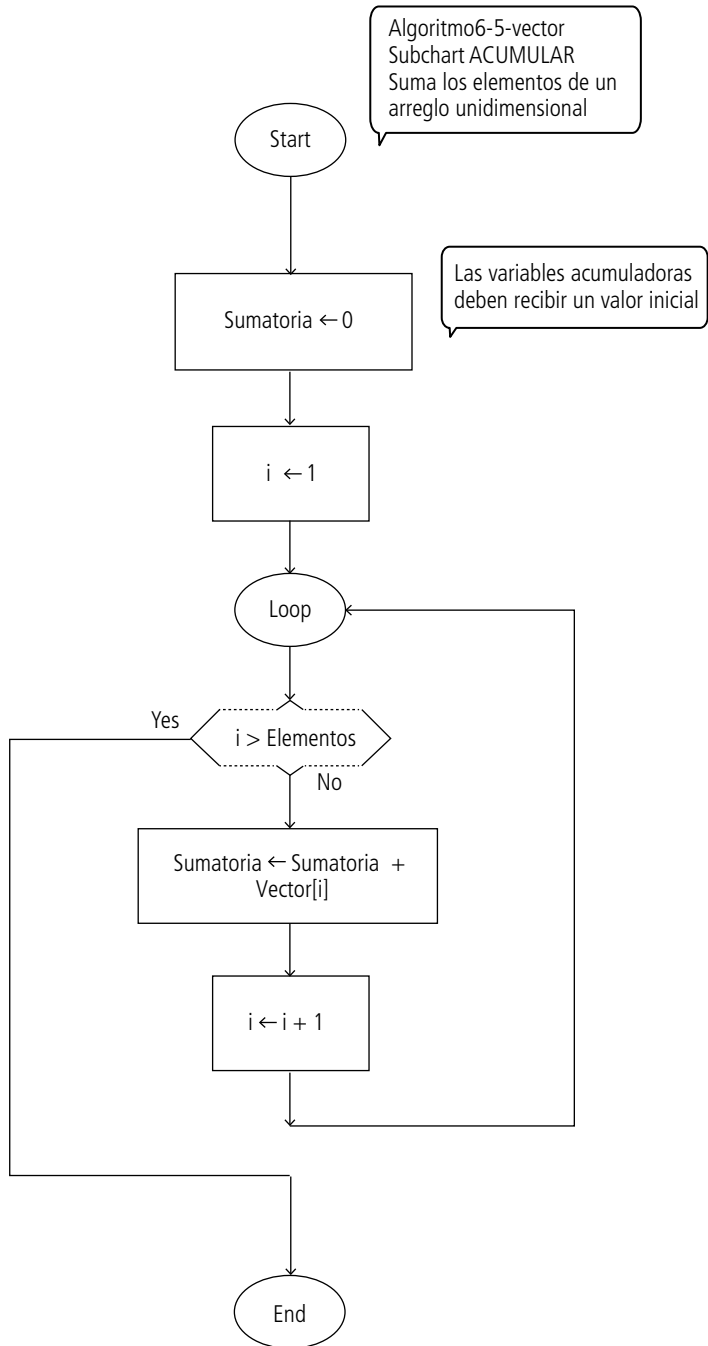


Figura 6.14 Algoritmo6-5-vector Subchart ACUMULAR.

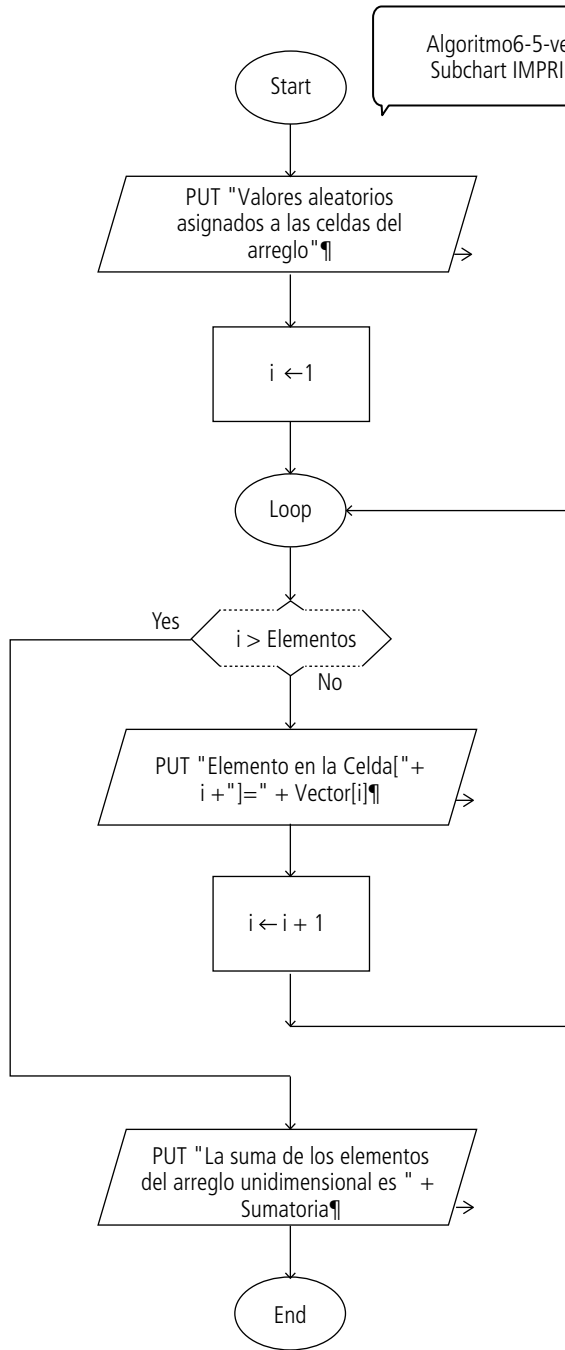


Figura 6.15 Algoritmo6-5-vector Subchart IMPRIMIR.



6.4 Diseño de algoritmos con arreglos de memoria unidimensionales, módulos y con paso de parámetros

Las buenas prácticas de la ingeniería de software se orientan a construir programas de calidad cuyo mantenimiento sea más fácil y económico. Hay suficiente investigación que permite decir que el software construido con módulos y paso de parámetros es una característica que permite disminuir los costos de mantenimiento.

A continuación encontrará ejemplos donde se usan los arreglos de memoria y los módulos con paso de parámetros. En el **Algoritmo6-6-vector** se utiliza el módulo **ASIGNACION** que permite almacenar números aleatorios a un arreglo de memoria de manera automática. Después, con el módulo **CLASIFICAR** se cuenta cuántos de esos elementos son pares y cuántos impares. Finalmente, con el módulo **IMPRIMIR** se muestran los resultados (véanse figuras 6.16 a 6.19).

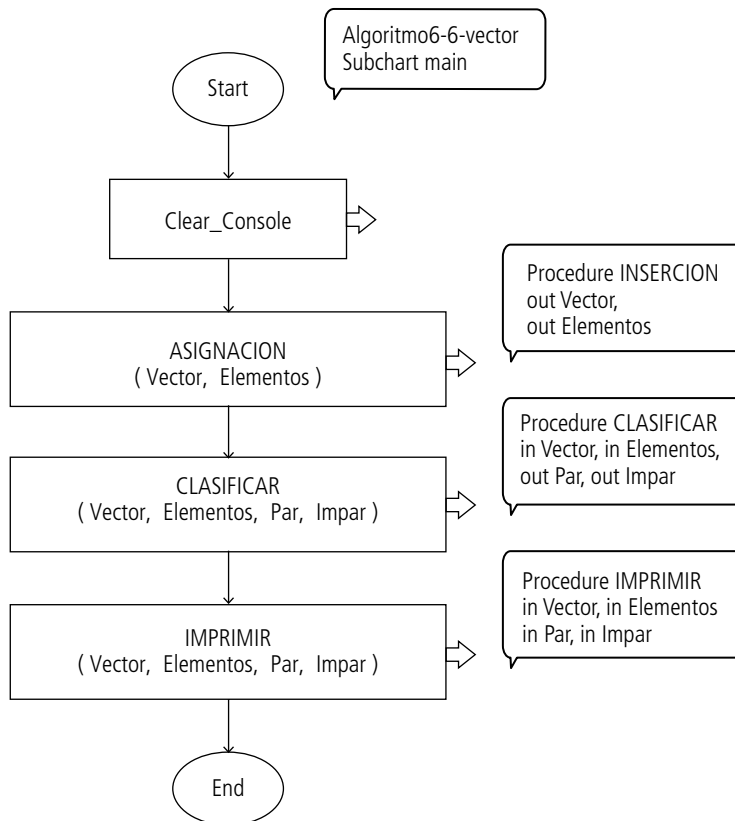
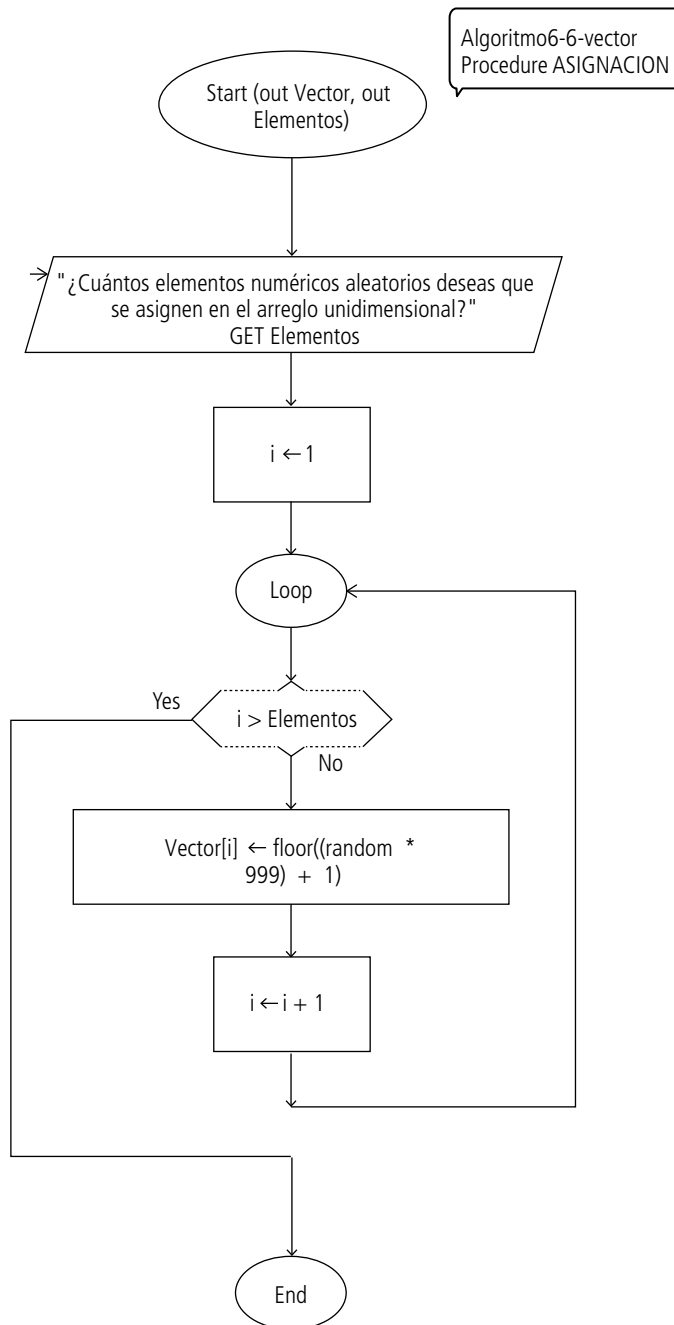


Figura 6.16 Algoritmo6-6-vector Subchart main.

**Figura 6.17** Algoritmo6-6-vector *Procedure ASIGNACION*.

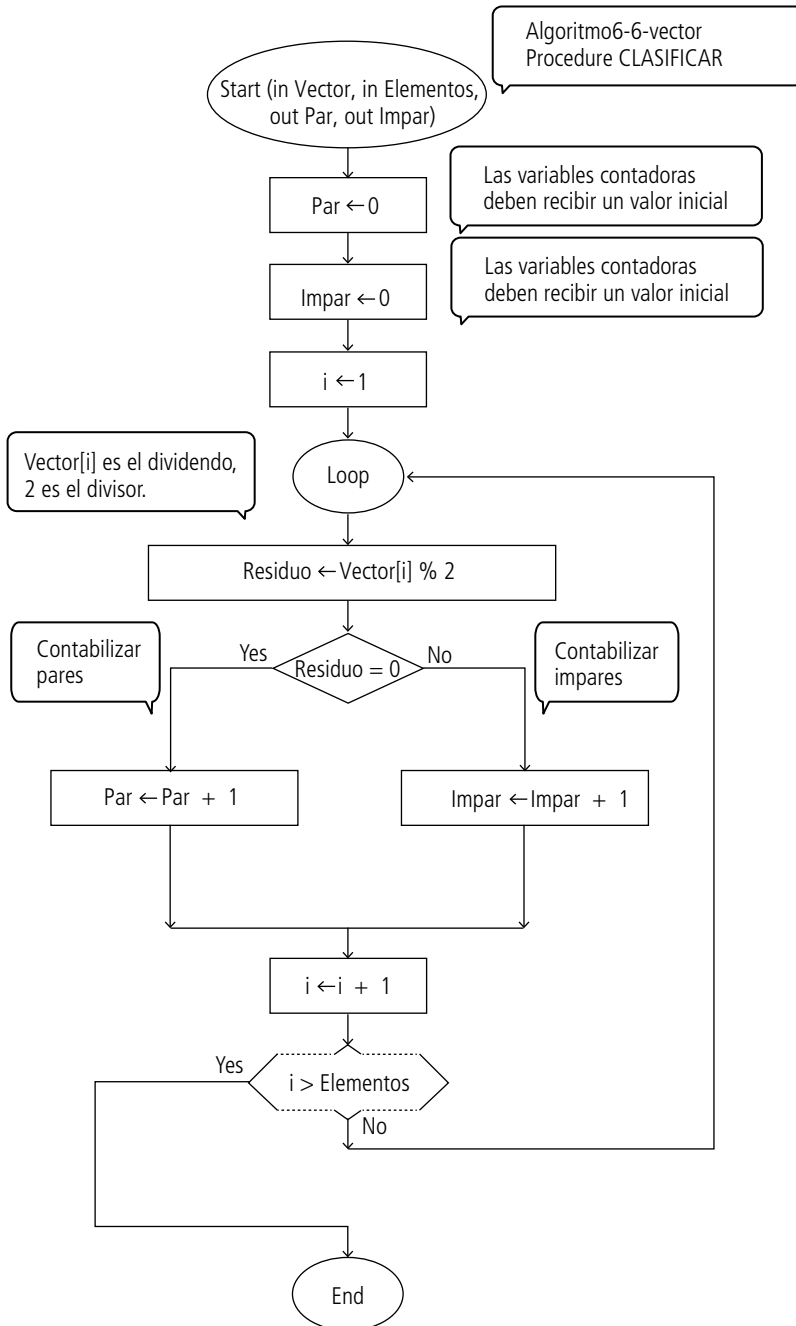


Figura 6.18 Algoritmo6-6-vector Procedure CLASIFICAR.

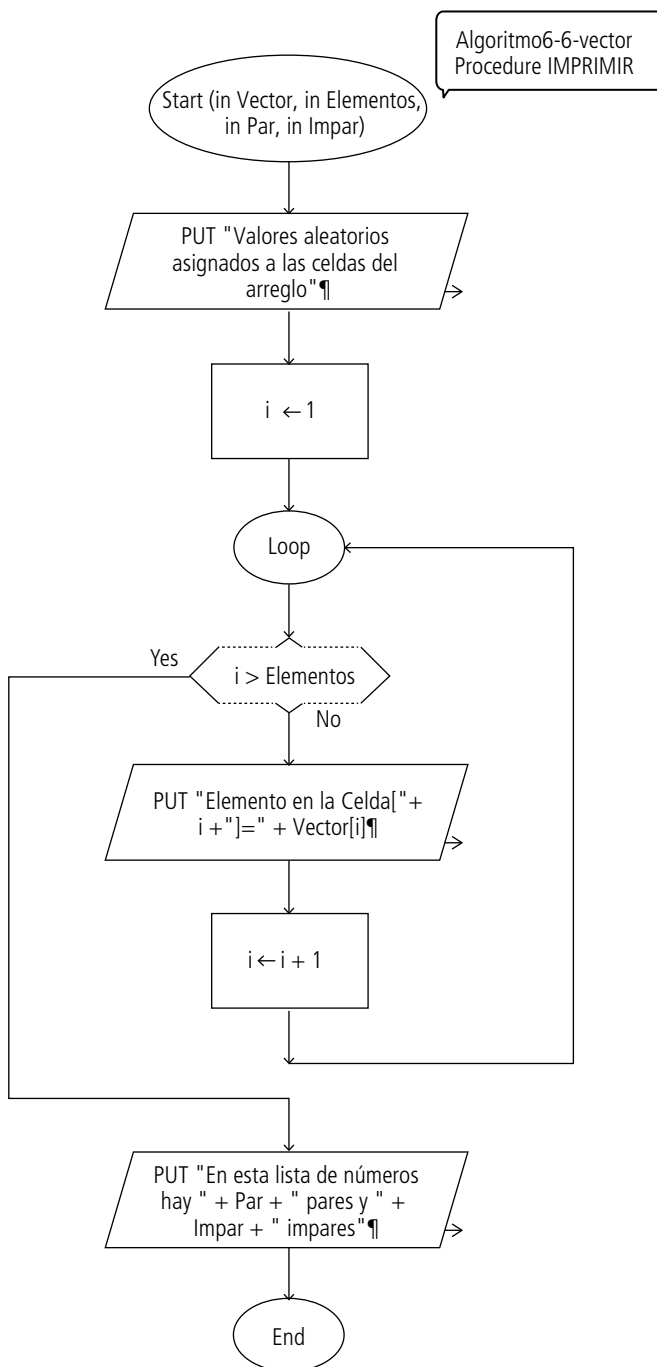


Figura 6.19 Algoritmo6-6-vector Procedure IMPRIMIR.

En el **Algoritmo6-7-vector**, con el módulo **ASIGNACION** se almacenan números aleatorios a un arreglo de memoria. Después, con el módulo **ELEMENTO_MAYOR** se determina cuál es el número mayor de los elementos del arreglo. Finalmente, con el módulo **IMPRIMIR** se muestran los elementos del arreglo y el número mayor (véanse figuras 6.20 a 6.23).

En el **Algoritmo6-8-vector** con el módulo **ASIGNACION** se almacenan números aleatorios a un arreglo de memoria. Después, con el módulo **CONTAR_REPETIDO** se hace una búsqueda secuencial de un elemento en el arreglo, por lo que se solicita al usuario que agregue el número que desea buscar, el cual puede estar repetido. Finalmente, con el módulo **IMPRIMIR** se muestran los elementos del arreglo, también si el elemento buscado fue o no encontrado y la cantidad de veces en que está repetido en la lista de números (véanse figuras 6.24 a 6.27)

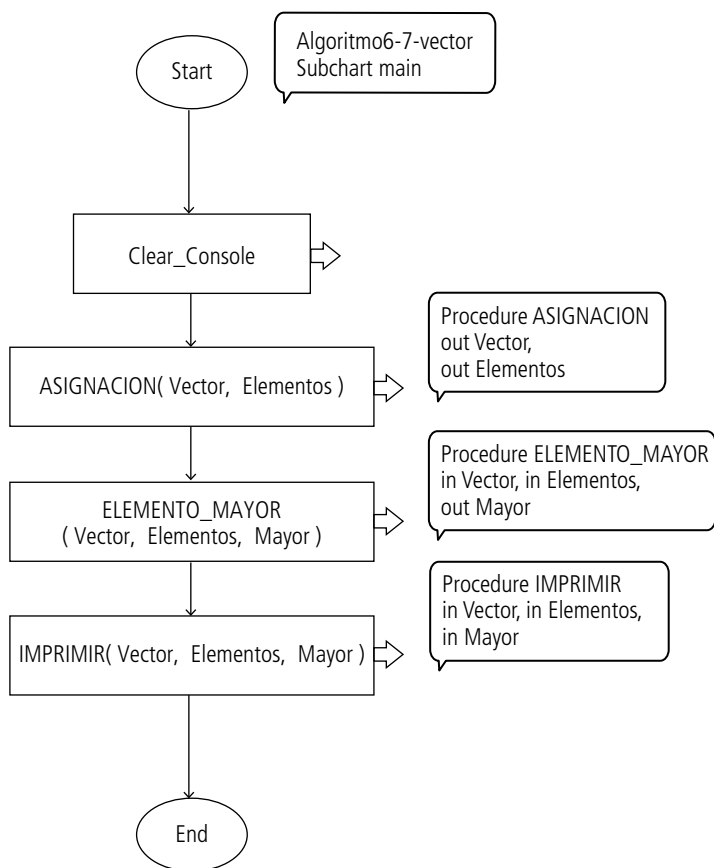


Figura 6.20 Algoritmo6-7-vector Subchart main.

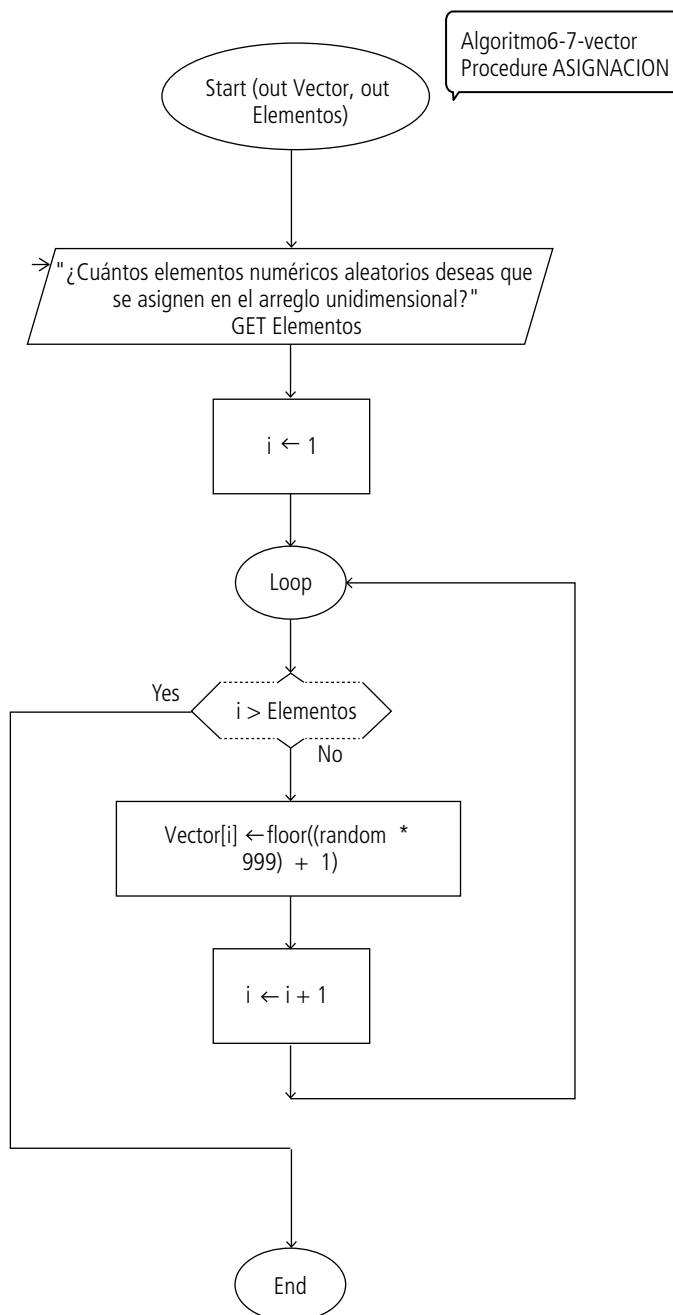


Figura 6.21 Algoritmo6-7-vector *Procedure ASIGNACION*.

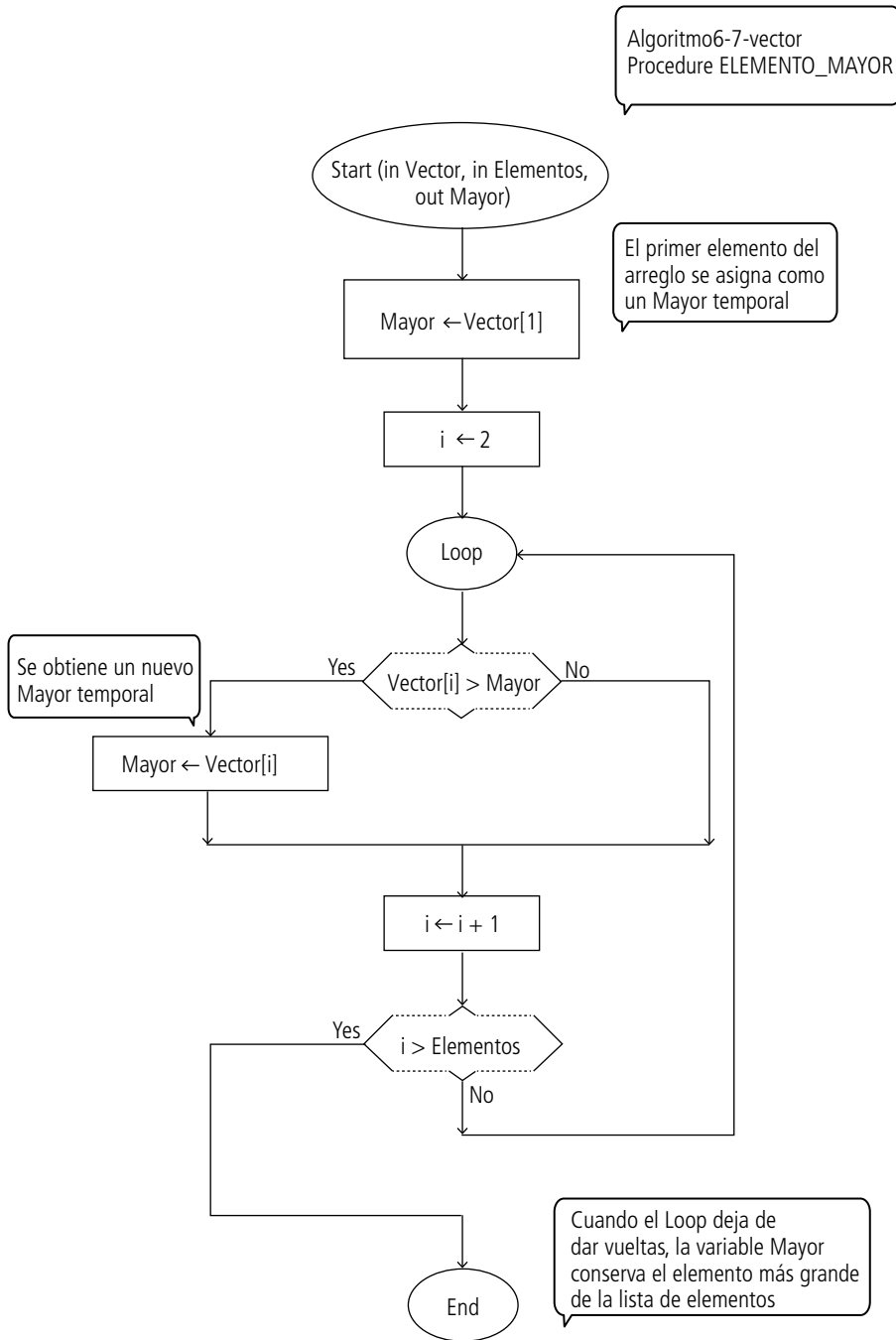


Figura 6.22 Algoritmo6-7-vector Procedure ELEMENTO_MAYOR.

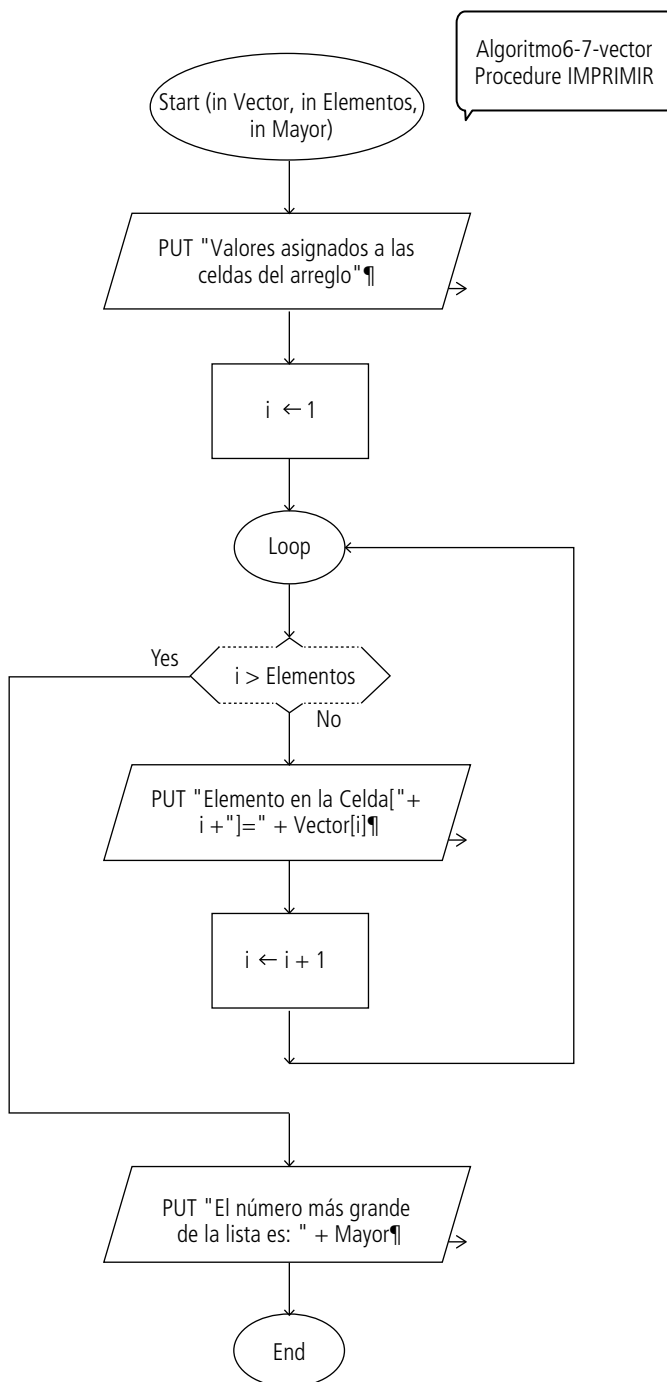


Figura 6.23 Algoritmo6-7-vector Procedure IMPRIMIR.

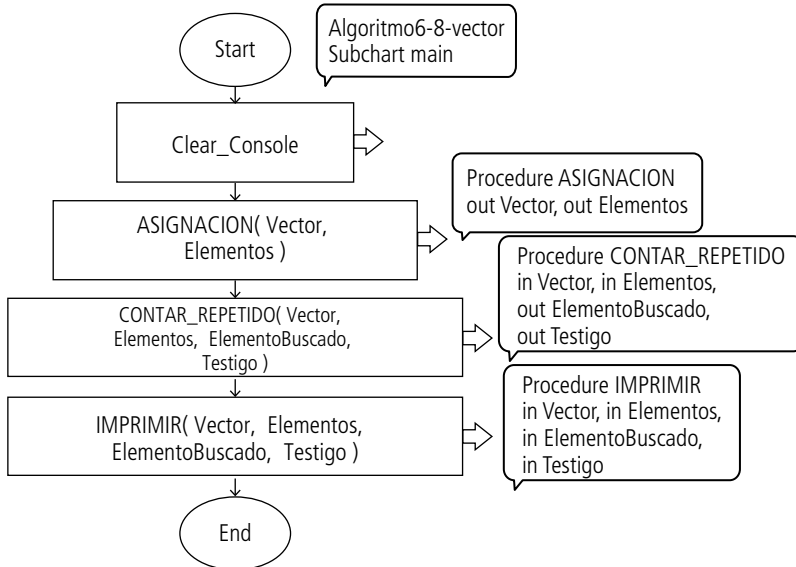


Figura 6.24 Algoritmo6-8-vector Subchart main.

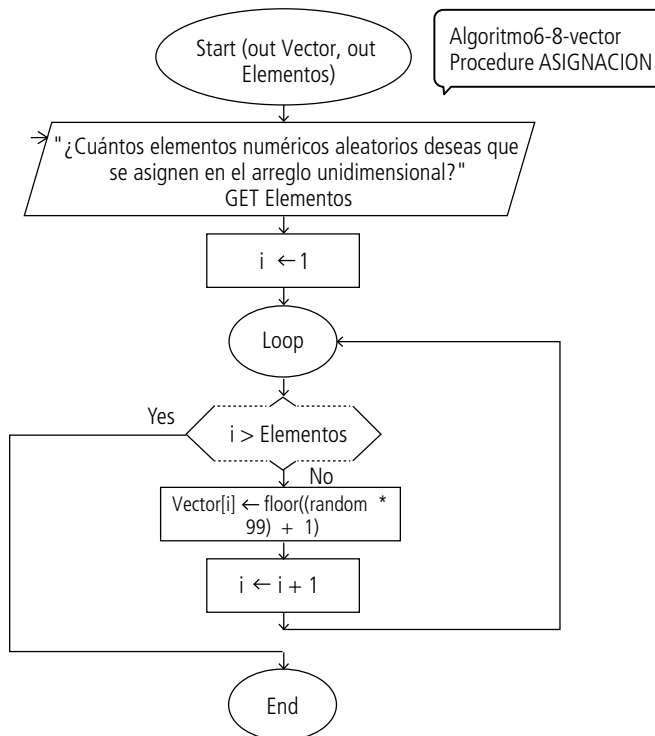


Figura 6.25 Algoritmo6-8-vector Procedure ASIGNACION.

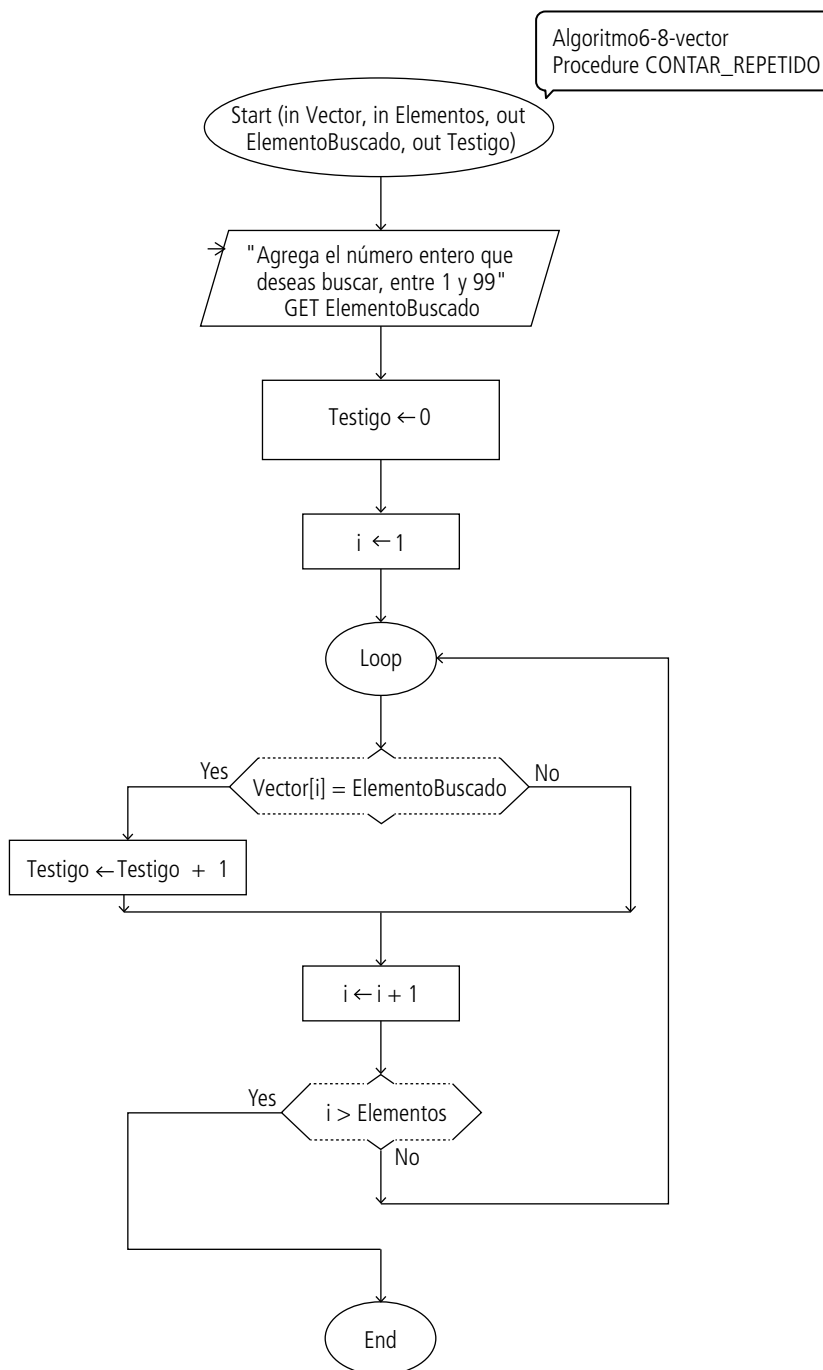


Figura 6.26 Algoritmo6-8-vector Procedure CONTAR_REPETIDO.

Algoritmo6-8-vector
Procedure IMPRIMIR

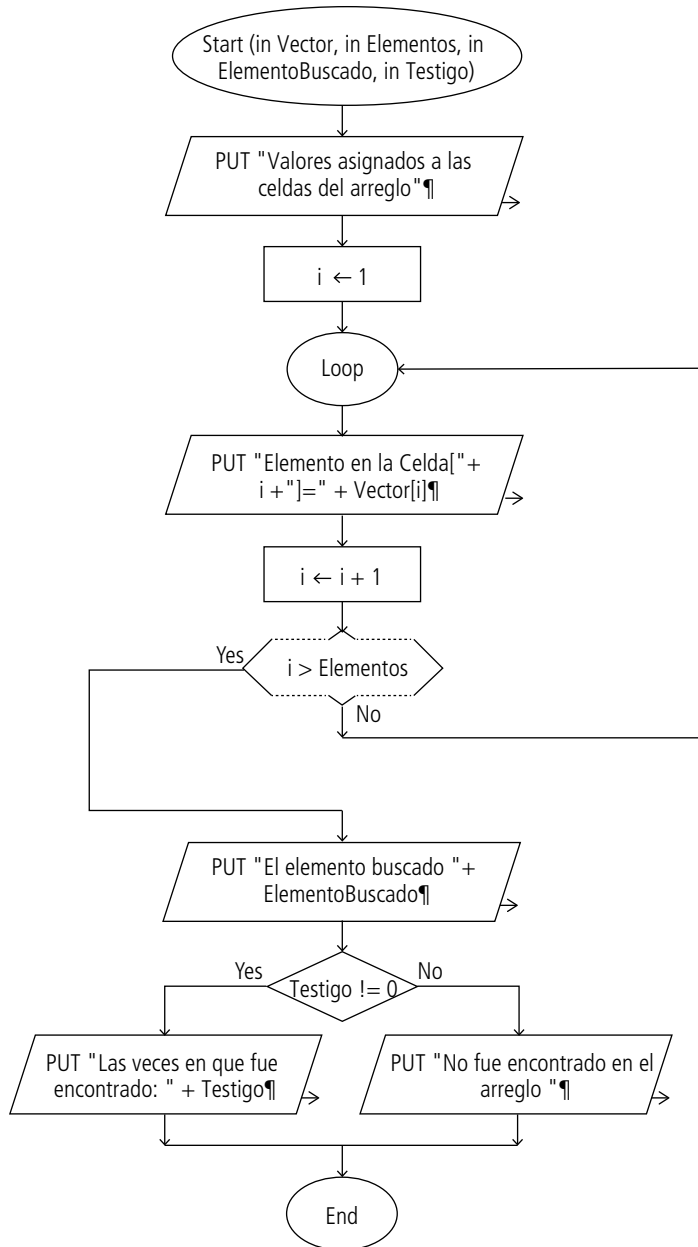


Figura 6.27 Algoritmo6-8-vector Procedure IMPRIMIR.



6.5 Introducción a los arreglos de memoria bidimensionales

Los arreglos de memoria bidimensionales son una estructura de datos donde las localidades o celdas de la memoria RAM se identifican con un nombre de variable y dos elementos que se llaman subíndices. Son un arreglo matricial semejante al de una hoja electrónica donde existen renglones y columnas, como en Excel. Los subíndices permiten referirse a una posición en particular que es la intersección de un renglón y una columna. Informalmente se les llama matrices y son un conjunto de datos relacionados que tienen las mismas características técnicas, como una matriz de números enteros.

Por ejemplo, si tuviéramos las ventas mensuales de un comercio que cuenta con cinco sucursales, donde los renglones se refieran a cada sucursal y las columnas a cada uno de los 12 meses del año, entonces tendríamos 60 datos o cifras numéricas que pertenecen al mismo contexto de información. En este caso, tendríamos un arreglo matricial de 5×12 ; por convención se colocan primero los renglones y después las columnas.

Por otro lado, cuando nos enfrentamos a un problema cuya solución se expresa en un sistema de ecuaciones lineales, donde la cantidad de incógnitas es igual a la cantidad de ecuaciones (véase tabla 6.3), el valor de las incógnitas puede obtenerse con métodos matemáticos matriciales como el Gauss-Jordan y Montante, etcétera.

Tabla 6.3 Sistemas de ecuaciones lineales

$5x - 2y + 8z = 9$
$x + 7y + 2z = -3$
$3x + 5y - 6z = 4$

Del anterior sistema de ecuaciones lineales se puede obtener la matriz de coeficientes y la matriz aumentada, según se muestra a continuación:

$$\begin{array}{rcccl}
 & & 5 & -2 & 8 \\
 \text{Matriz de coeficientes} & = & 1 & 7 & 2 \\
 & & 3 & 5 & -6 \\
 \\
 & & 5 & -2 & 8 & 9 \\
 \text{Matriz aumentada} & = & 1 & 7 & 2 & -3 \\
 & & 3 & 5 & -6 & 4
 \end{array}$$

A partir de la representación matemática, podemos decir que cada posición de la matriz puede considerarse una celda que tiene una ubicación única que es la intersección de un renglón y una columna, lo que permite representar una matriz de posiciones, donde el primer número se refiere a los renglones y el segundo, a las columnas. Vea enseguida la matriz de posiciones considerando únicamente la matriz de coeficientes:

		1,1	1,2	1,3
Matriz de posiciones	=	2,1	2,2	2,3
		3,1	3,2	3,3

A continuación considere una matriz de coeficientes de cuatro por cuatro elementos:

		25	30	89	−45
Matriz	=	12	90	−25	50
		−70	−36	79	−81
		63	42	78	−27

Dado que la variable se llama **Matriz**, entonces corresponde la siguiente notación: Matriz [1,1] = −25, Matriz [1,2] = 30, Matriz [1,3] = 89, Matriz [1,4] = −45, Matriz [2,1] = 12, Matriz [2,2] = 90, Matriz [2,3] = −25, Matriz [2,4] = 50, Matriz [3,1] = −70, Matriz [3,2] = −36, Matriz [3,3] = 79, Matriz [3,4] = −81, Matriz [4,1] = 63, Matriz [4,2] = 42, Matriz [4,3] = 78, Matriz [4,4] = −27.

Una mejor visualización de la anterior representación matricial se observa en la tabla 6.4. En la mayoría de los textos de arreglos de memoria bidimensionales se usa la variable *i* para referirse a los renglones y la variable *j* para las columnas. Debe considerar que el nombre de ambas variables no cumplen con las reglas de calidad.

Tabla 6.4 Representación de una matriz cuadrada de cuatro por cuatro elementos

		Columnas			
		<i>j</i> = 1	<i>j</i> = 2	<i>j</i> = 3	<i>j</i> = 4
Renglones	<i>i</i> = 1	25	30	89	−45
	<i>i</i> = 2	12	90	−25	50
	<i>i</i> = 3	−70	−36	79	−81
	<i>i</i> = 4	63	42	78	−27

Las variables que son arreglo de memoria bidimensional pueden distinguirse porque su identificador se acompaña con corchetes a manera de sufijo, el que abre y cierra, y en medio de ellos se colocan dos subíndices separados por una coma. La convención indica que el primer subíndice se refiere al renglón y el segundo, a la columna. Los subíndices pueden clasificarse en estáticos y dinámicos. Cuando los elementos que están entre corchetes son constantes, se consideran como subíndices estáticos. Cuando los elementos que están entre corchetes son variables, se consideran como subíndices dinámicos.

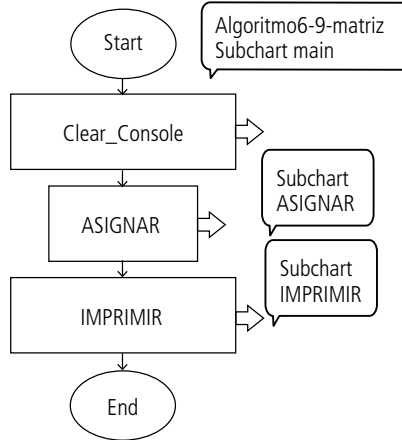
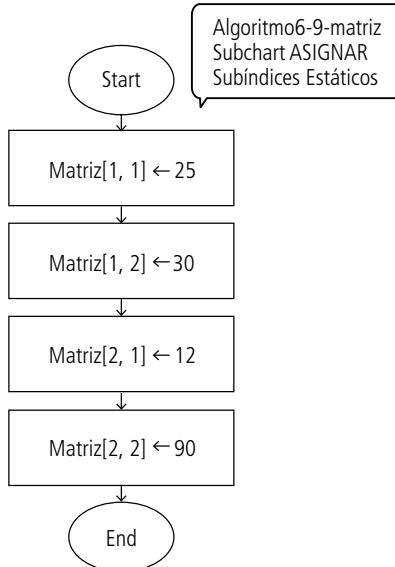
En esta primera parte de los arreglos de memoria bidimensionales nos concentraremos en entender cómo agregar valores numéricos a la memoria RAM organizados bajo la lógica de un arreglo matricial.

En el **Algoritmo6-9-matriz** se muestra un ejemplo del uso de arreglos de memoria bidimensionales con subíndices estáticos (véase tabla 6.5).

Tabla 6.5 Matriz de dos por dos

25	30
12	90

Observe que dentro del *Subchart IMPRIMIR*, para mostrar el arreglo en forma matricial, en algunos símbolos Output se desactivó la casilla de verificación **End current line**, por lo que el símbolo ¶ no aparece. Esto permite que el símbolo Output no haga un brinco línea ni retorno de carro, por lo que la impresión de datos se hace sobre la misma línea (véanse figuras 6.28 a 6.30).

**Figura 6.28** Algoritmo6-9-matriz *Subchart main*.**Figura 6.29** Algoritmo6-9-matriz *Subchart ASIGNAR*.

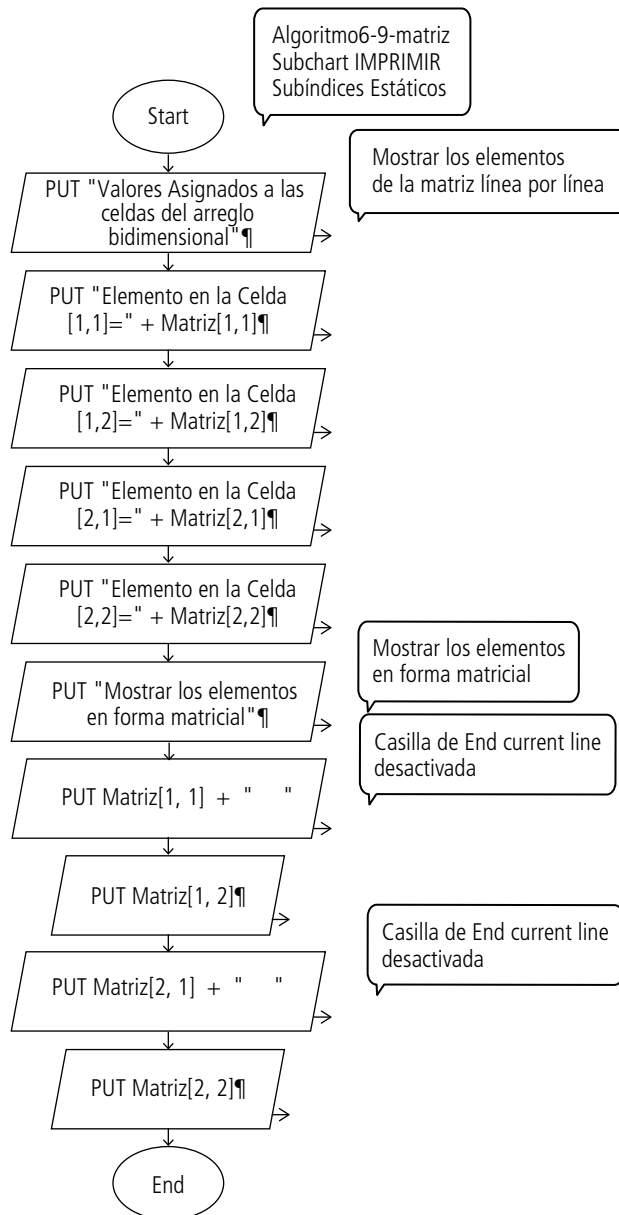


Figura 6.30 Algoritmo6-9-matriz *Subchart* IMPRIMIR.

Los subíndices también pueden ser dinámicos, tal y como se muestra en el **Algoritmo6-10-matriz**. Recuerde que cuando se usan índices dinámicos, los renglones se representan con la variable i y las columnas, con la variable j (véanse figuras 6.31 a 6.33).

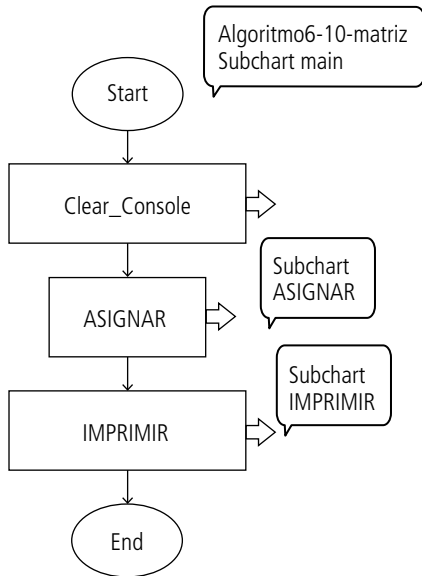


Figura 6.31 Algoritmo6-10-matriz Subchart main.

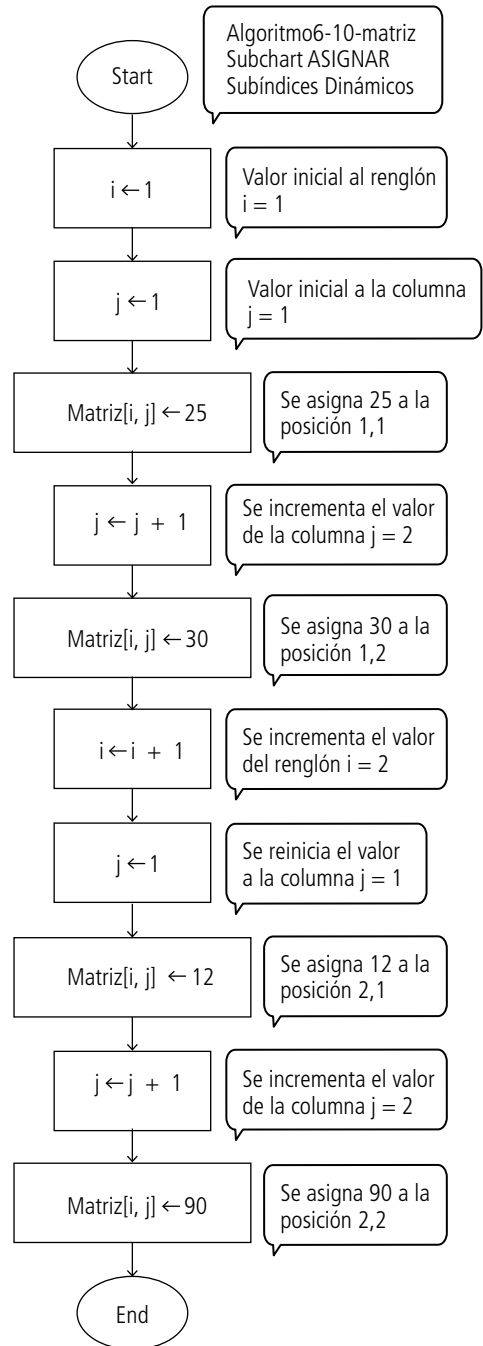


Figura 6.32 Algoritmo6-10-matriz Subchart ASIGNAR.

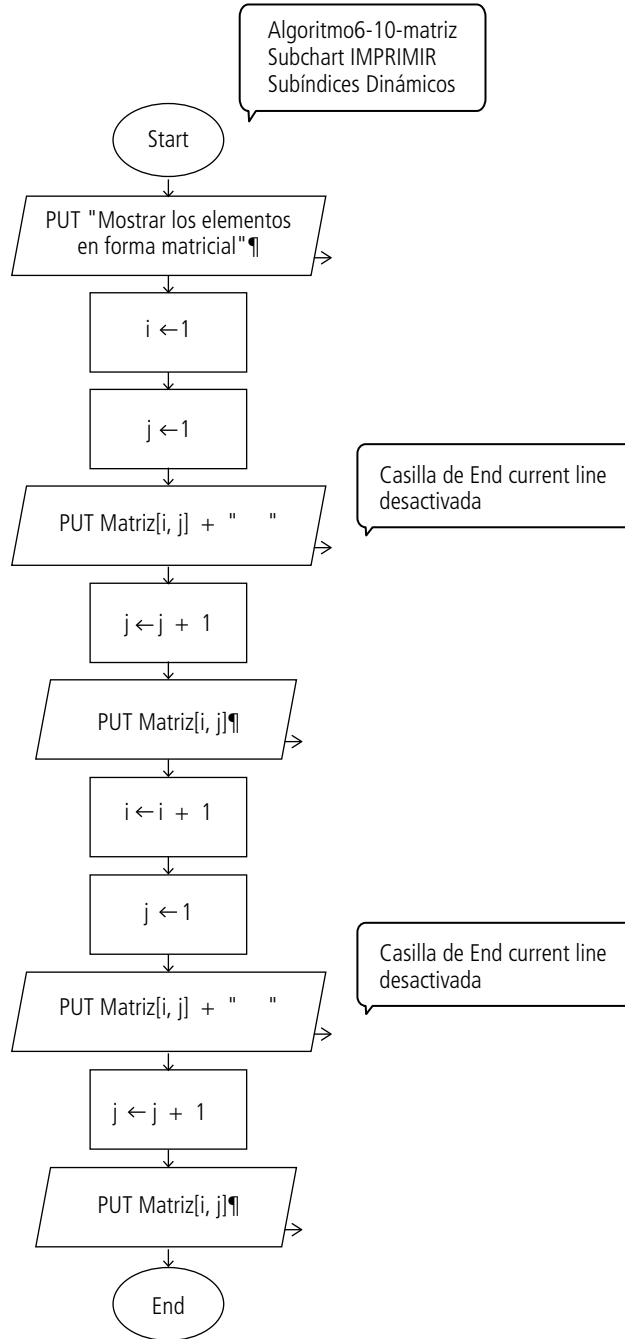


Figura 6.33 Algoritmo6-10-matriz *Subchart* IMPRIMIR.

Con el **Algoritmo6-11-matriz** se permite al usuario la captura de elementos a un arreglo de memoria bidimensional. Éste es un algoritmo rígido e inflexible porque sólo funciona para insertar cuatro elementos en un arreglo matricial con una dimensión dedos por dos. Si deseáramos aumentar o disminuir la cantidad de elementos en la matriz, debe modificarse el diagrama (véanse figuras 6.34 a 6.36).

Con el **Algoritmo6-12-matriz** se permite al usuario la captura de elementos a un arreglo de memoria bidimensional; además, el uso de la estructura repetitiva permite que el algoritmo sea más flexible y adaptable para aumentar o disminuir la cantidad de elementos en la matriz. En este caso, se trata de una matriz cuadrada, es decir, tiene la misma cantidad de filas que de columnas.

Para permitir la captura en forma matricial, en el *subchart* **CAPTURA** se utilizó un Loop dentro de otro Loop. El Loop externo controla los reglones con la variable i . El Loop interno controla las columnas con la variable j y “corre” más rápido que el externo, semejante a las manecillas del reloj, donde la manecilla de los minutos “corre” más rápido que la de las horas (véanse figuras 6.37 a 6.39)

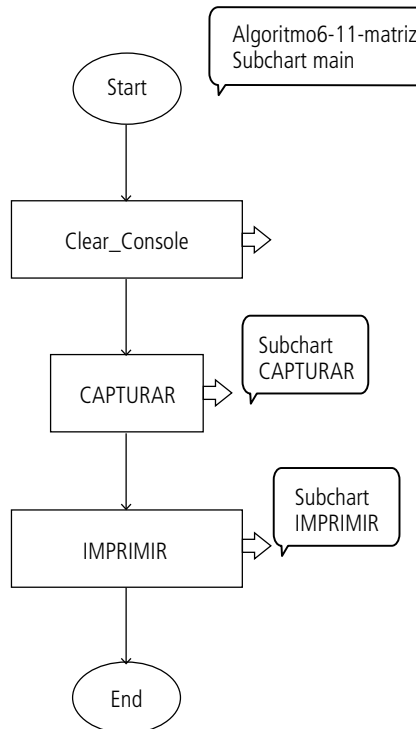


Figura 6.34 Algoritmo6-11-matriz *Subchart main*.

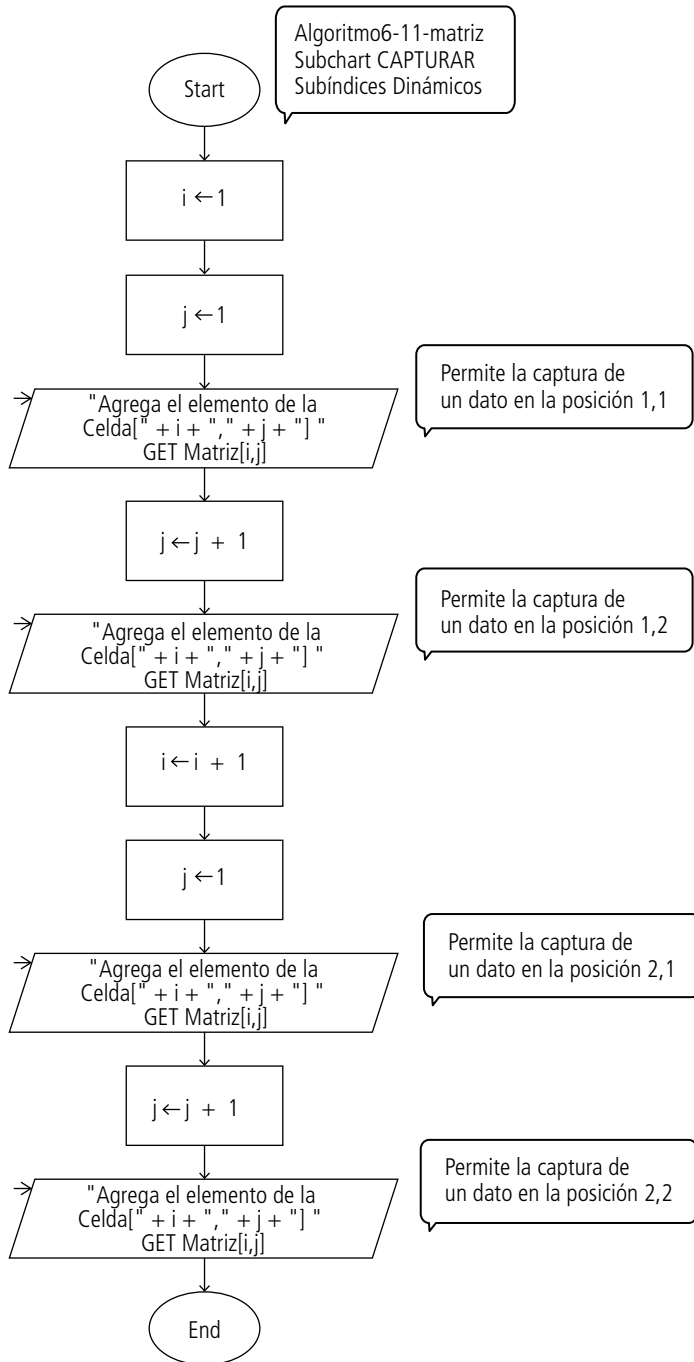


Figura 6.35 Algoritmo6-11-matriz *Subchart* CAPTURAR.

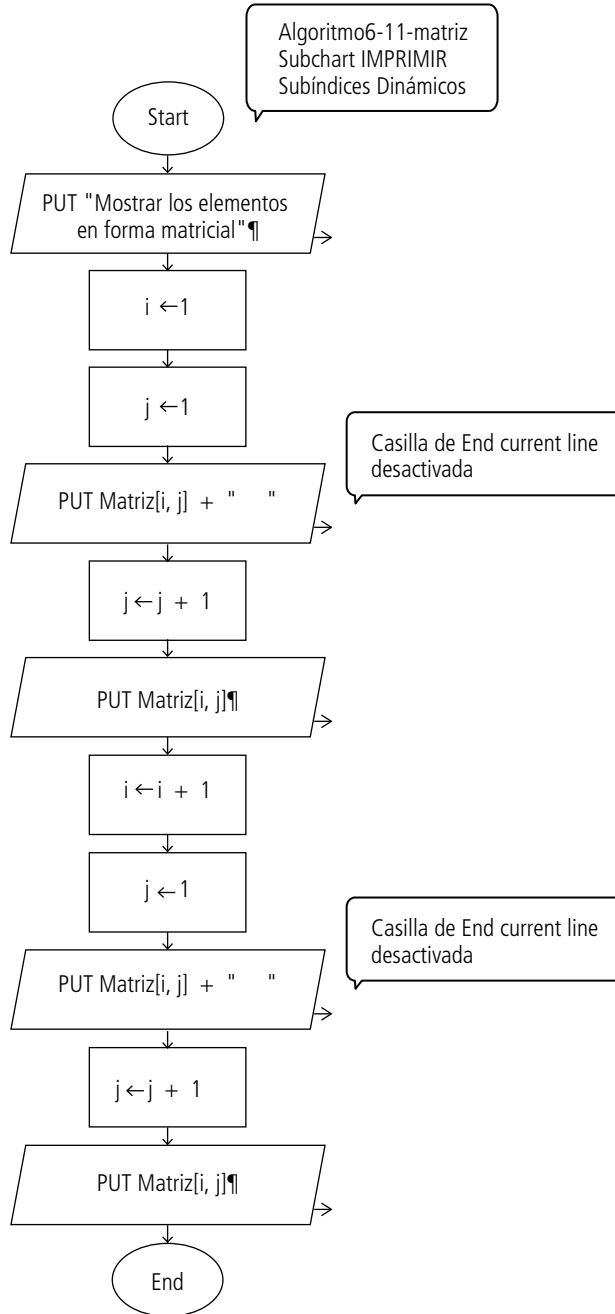


Figura 6.36 Algoritmo6-11-matriz Subchart IMPRIMIR.

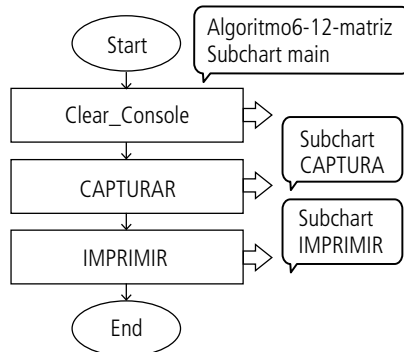


Figura 6.37 Algoritmo6-12-matriz *Subchart main*.

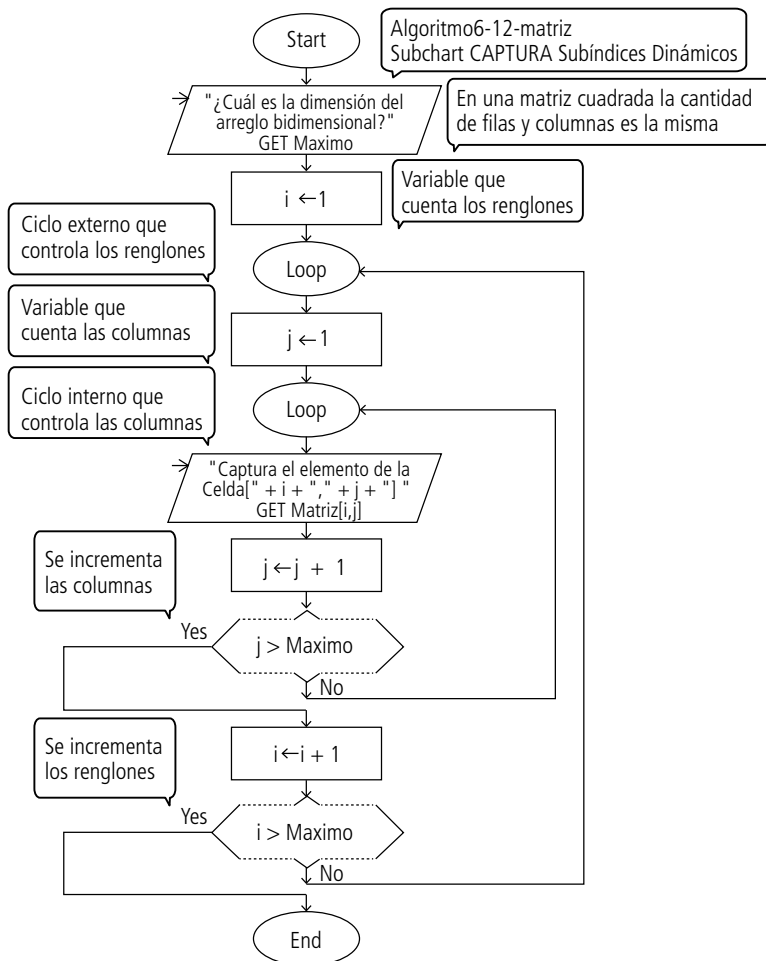


Figura 6.38 Algoritmo6-12-matriz *Subchart CAPTURA*.

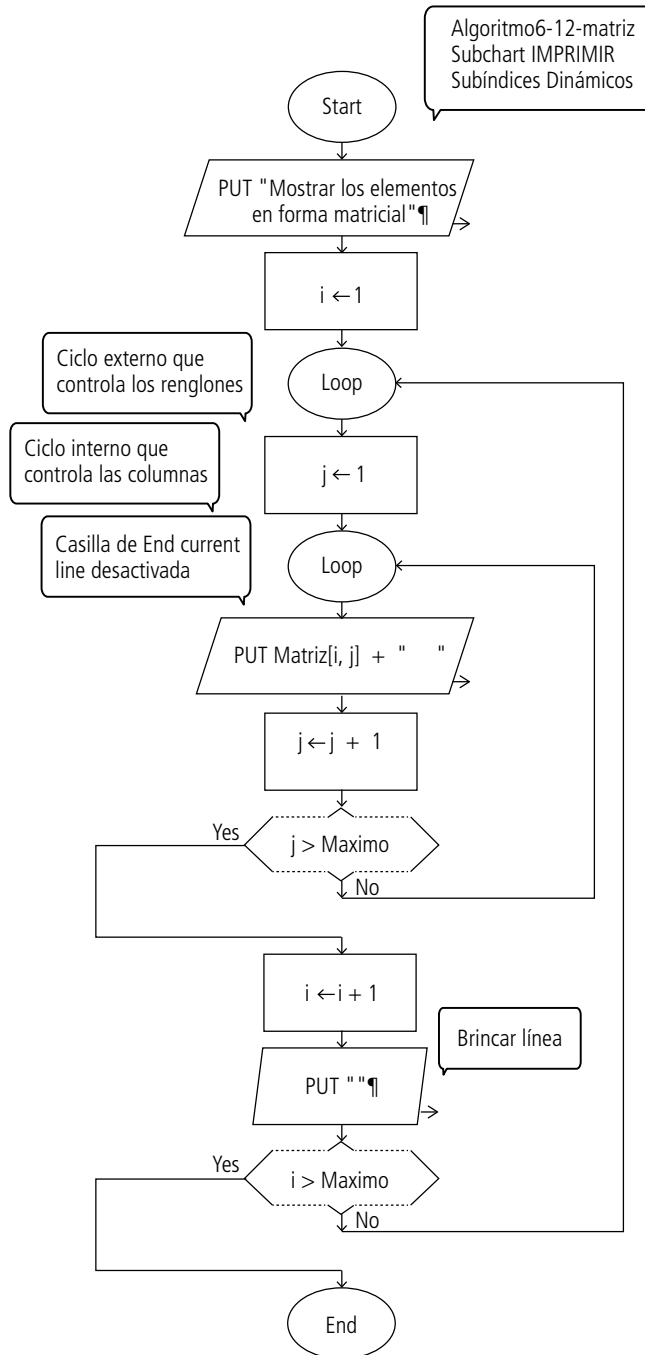


Figura 6.39 Algoritmo6-12-matriz Subchart IMPRIMIR.



6.6 Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y sin paso de parámetros

En realidad, todos los diagramas anteriores referentes a arreglos de memoria bidimensionales son parte de este mismo tema, por lo que sólo mostraremos un ejemplo más.

Con el **Algoritmo6-13-matriz** se asignan de manera automática elementos a un arreglo de memoria bidimensional. Con el módulo **ASIGNACION**, el usuario ya no tiene control sobre los números asignados en el arreglo, pues con la función de redondeo (*floor*), aleatorio (*random*) y su argumento se generan números aleatorios entre uno y 99. Además, el módulo **ACUMULAR** permite sumar los elementos insertados en el arreglo. En este caso, la variable Sumatoria acumula los valores de los elementos (véanse figuras 6.40 a 6.43). La representación matemática de la suma es la siguiente:

$$\text{Sumatoria} = \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} \text{Matriz}[i,j]$$

La representación de la suma, celda por celda, considerando una matriz de tres por tres elementos es la siguiente:

$$\text{Sumatoria} = \text{Matriz}[1,1] + \text{Matriz}[1,2] + \text{Matriz}[1,3] + \text{Matriz}[2,1] + \text{Matriz}[2,2] + \text{Matriz}[2,3] + \text{Matriz}[3,1] + \text{Matriz}[3,2] + \text{Matriz}[3,3]$$

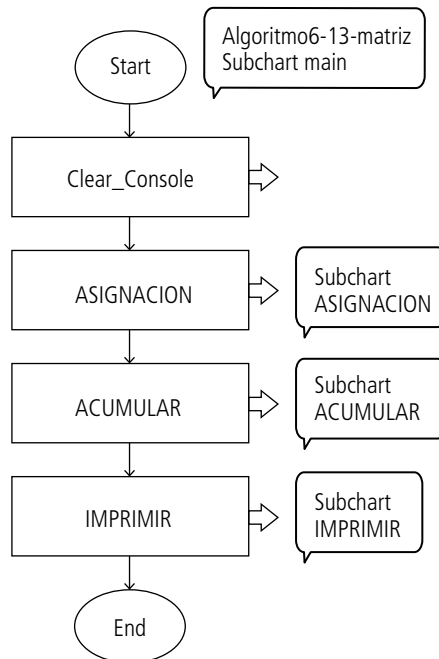


Figura 6.40 Algoritmo6-13-matriz Subchart main.

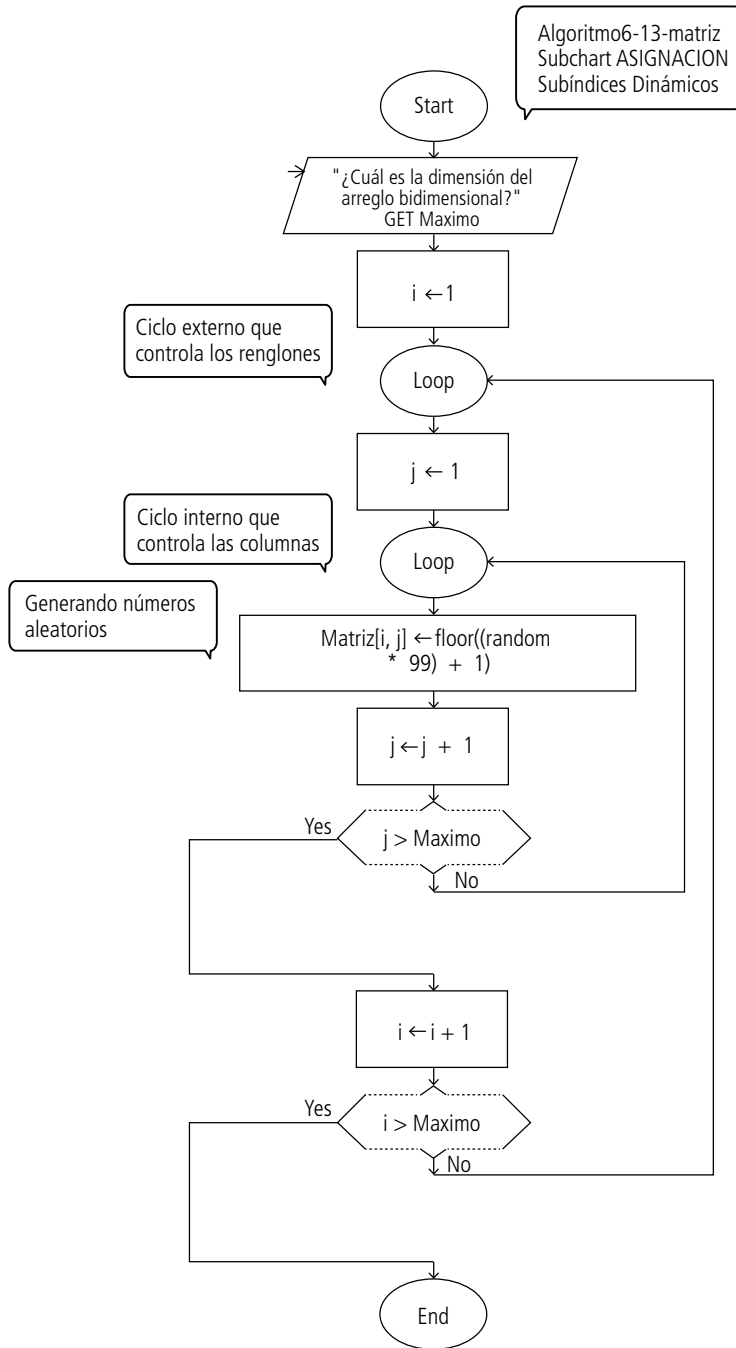


Figura 6.41 Algoritmo6-13-matriz Subchart ASIGNACION.

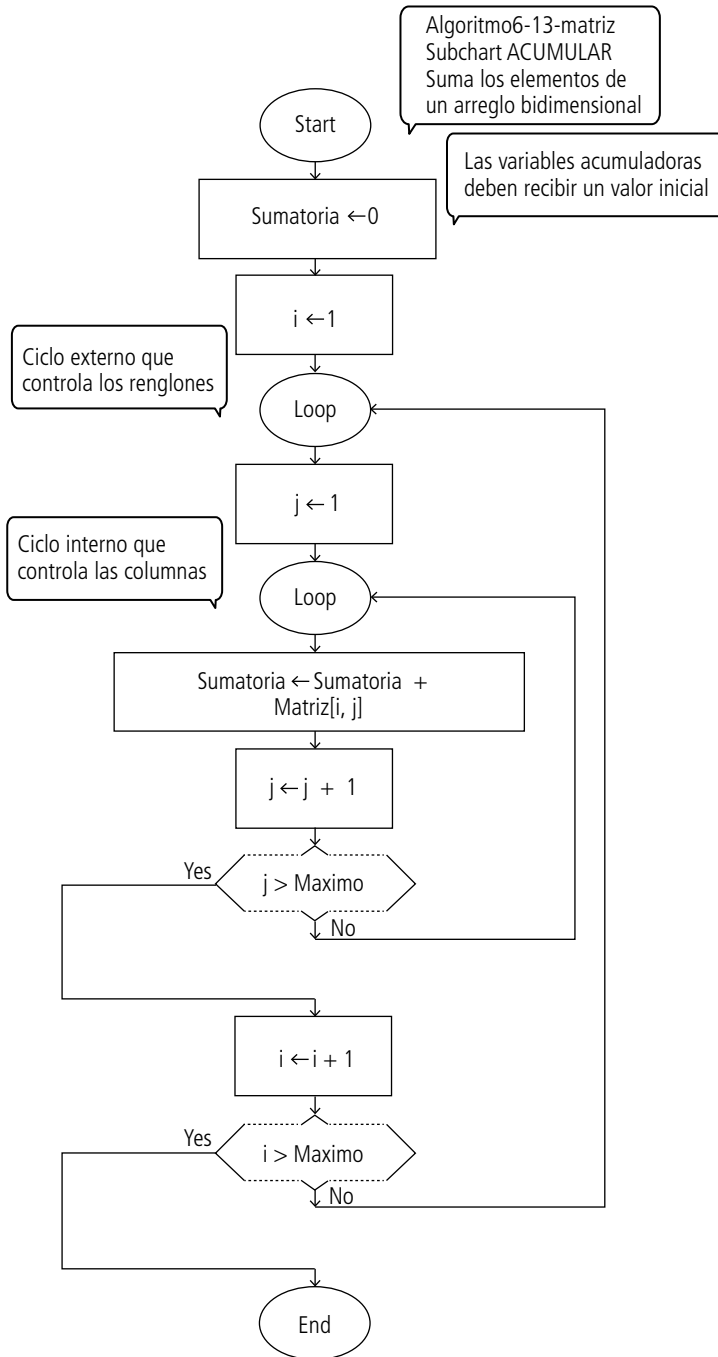


Figura 6.42 Algoritmo6-13-matriz *Subchart* ACUMULAR.

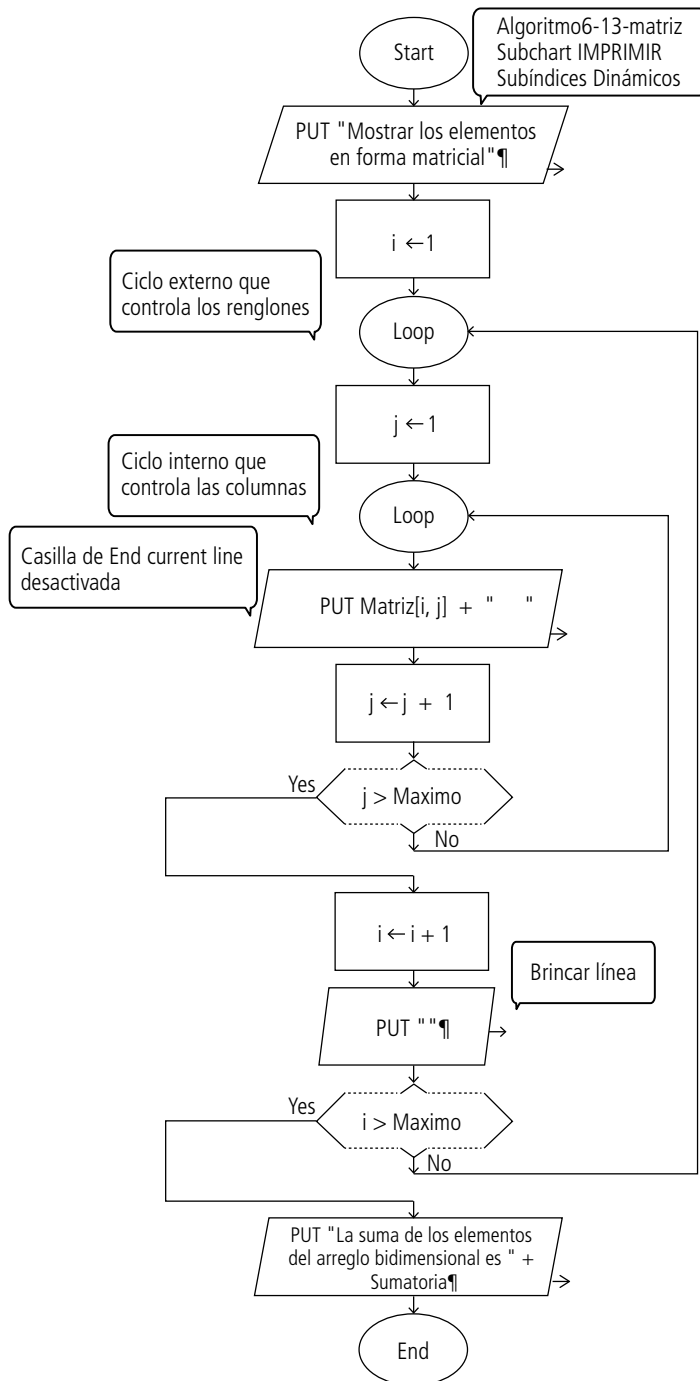


Figura 6.43 Algoritmo6-13-matriz Subchart IMPRIMIR.



6.7 Diseño de algoritmos con arreglos de memoria bidimensionales, módulos y con paso de parámetros

De acuerdo con la ciencia matemática, los elementos de un arreglo matricial forman parte de conceptos como diagonal principal, matriz triangular superior, matriz triangular inferior y diagonal secundaria.

En el siguiente arreglo matricial de cuatro por cuatro, se ha colocado DP en las posiciones de los elementos de la diagonal principal y DS en las posiciones de los elementos de la diagonal secundaria. En las demás posiciones se ha colocado el número cero:

$$\text{Matriz} = \begin{array}{cccc} \text{DP} & 0 & 0 & \text{DS} \\ 0 & \text{DP} & \text{DS} & 0 \\ 0 & \text{DS} & \text{DP} & 0 \\ \text{DS} & 0 & 0 & \text{DP} \end{array}$$

De tal forma que si quisiéramos sumar exclusivamente los elementos de la diagonal principal o de la diagonal secundaria, la representación matemática estará dada por las siguientes ecuaciones.

$$\text{Diagonal principal} = \text{Matriz}[1,1] + \text{Matriz}[2,2] + \text{Matriz}[3,3] + \text{Matriz}[4,4]$$

$$\text{Diagonal secundaria} = \text{Matriz}[1,4] + \text{Matriz}[2,3] + \text{Matriz}[3,2] + \text{Matriz}[4,1]$$

Por otro lado, en el siguiente arreglo matricial de cuatro por cuatro se ha colocado MTS en las posiciones de los elementos de la matriz triangular superior y MTI, en las posiciones de los elementos de la matriz triangular inferior. En las demás posiciones se ha anotado el número cero:

$$\text{M} = \begin{array}{cccc} 0 & \text{MTS} & \text{MTS} & \text{MTS} \\ \text{MTI} & 0 & \text{MTS} & \text{MTS} \\ \text{MTI} & \text{MTI} & 0 & \text{MTS} \\ \text{MTI} & \text{MTI} & \text{MTI} & 0 \end{array}$$

De tal forma que si quisiéramos sumar de manera exclusiva los elementos de la matriz triangular superior o de la matriz triangular inferior, la representación matemática estará dada por las siguientes ecuaciones.

$$\text{MT Superior} = \text{M}[1,2] + \text{M}[1,3] + \text{M}[1,4] + \text{M}[2,3] + \text{M}[2,4] + \text{M}[3,4]$$

$$\text{MT Inferior} = \text{M}[2,1] + \text{M}[3,1] + \text{M}[3,2] + \text{M}[4,1] + \text{M}[4,2] + \text{M}[4,3]$$

Dado que los elementos de cada concepto son fijos para cualquier tamaño de una matriz cuadrada, para poder hacer los programas que permitan sumar los elementos de cada uno de ellos es conveniente encontrar un comportamiento repetitivo. Éste se resume en la tabla 6.6.

Tabla 6.6 Conceptos de una matriz cuadrada

Concepto	Comportamiento repetitivo	Condición para sumar
Diagonal principal	La fila y la columna siempre son iguales.	$i = j$
Diagonal secundaria	La fila más la columna es igual a la dimensión de la matriz más uno.	$i + j = N + 1$
Matriz triangular superior	La fila siempre es inferior a la columna.	$i < j$
Matriz triangular inferior	La fila siempre es mayor que la columna.	$i > j$

Con el **Algoritmo6-14-matriz** se insertan de manera automática elementos aleatorios a un arreglo de memoria bidimensional y se suman los elementos de la diagonal principal (véanse figuras 6.44 a 6.47).

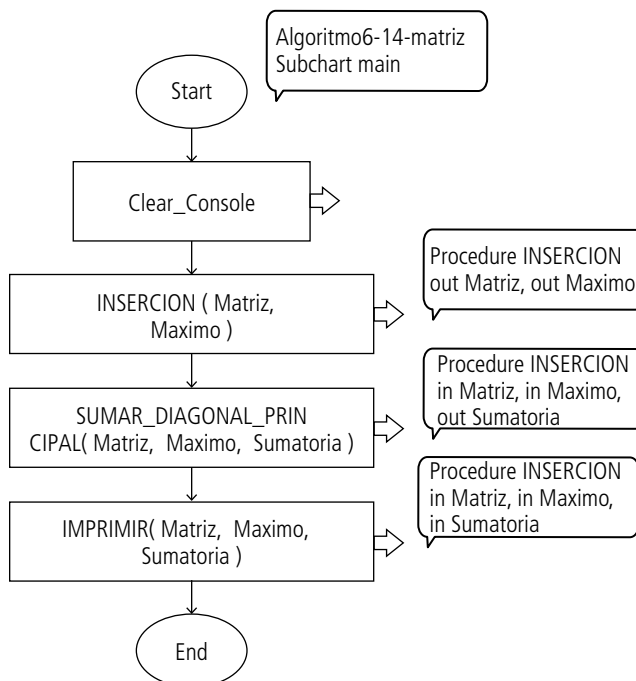


Figura 6.44 Algoritmo6-14-matriz Subchart main.

Algoritmo6-14-matriz
Procedure INSERCIÓN
Subíndices Dinámicos

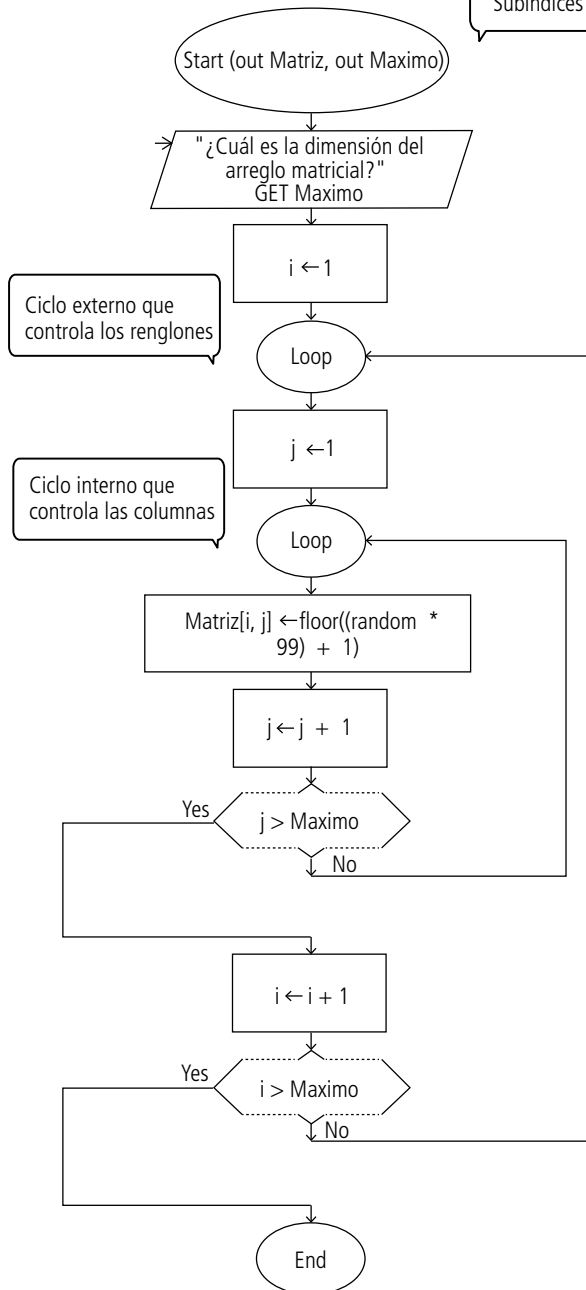


Figura 6.45 Algoritmo6-14-matriz Procedure INSERCIÓN.

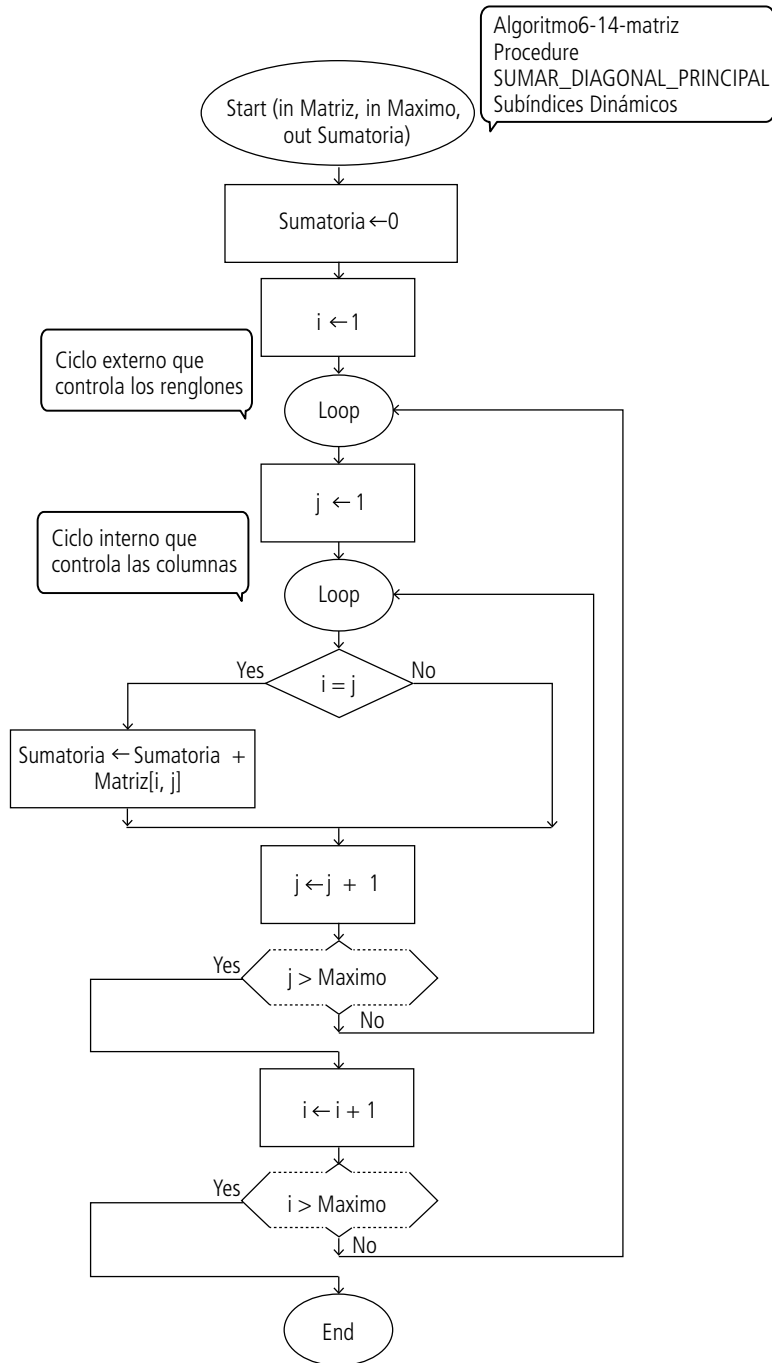


Figura 6.46 Algoritmo6-14-matriz *Procedure* SUMAR_DIAGONAL_PRINCIPAL.

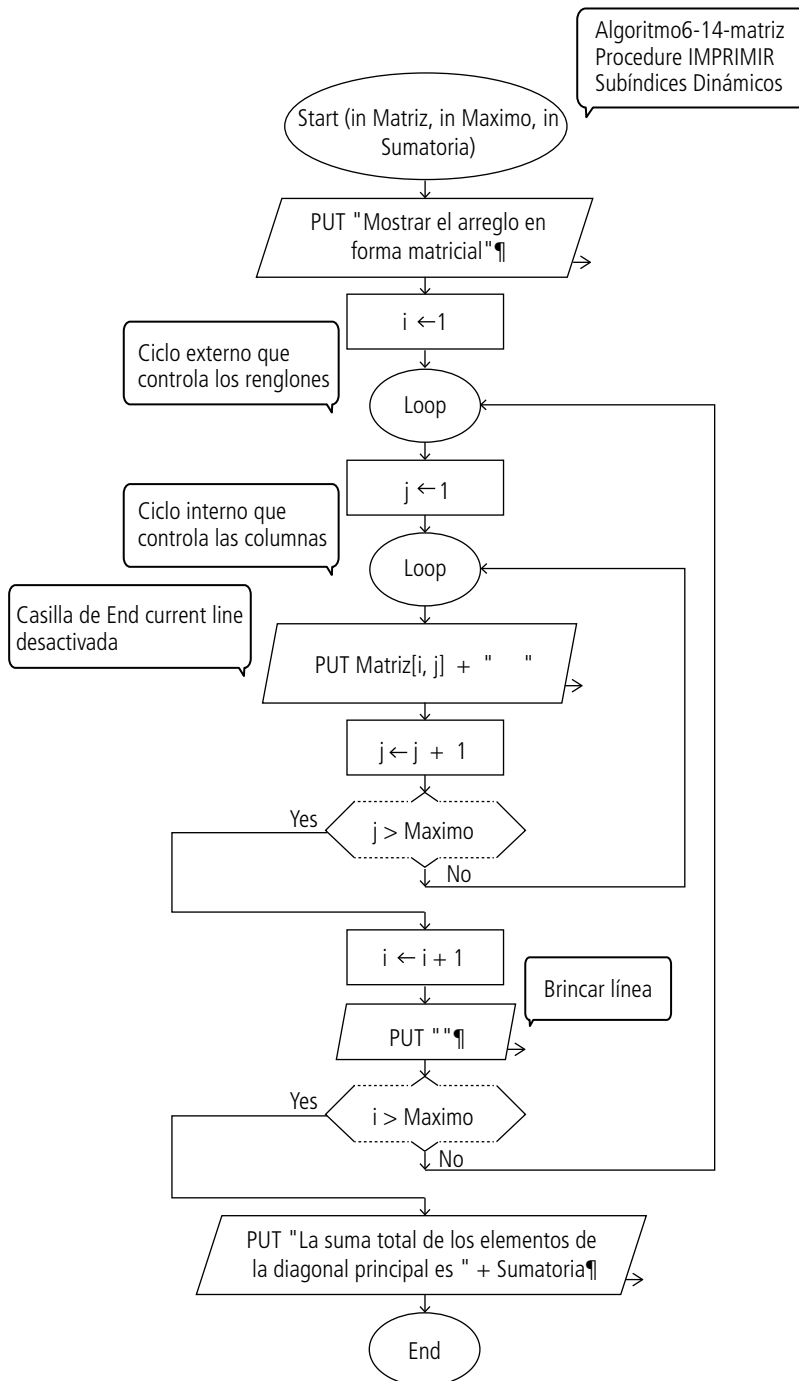


Figura 6.47 Algoritmo6-14-matriz *Procedure IMPRIMIR*.

Para sumar los diversos conceptos de la matriz tendrá que modificar el símbolo Selection del módulo **SUMAR_DIAGONAL_PRINCIPAL**. En su lugar, colocará la condición para sumar, de acuerdo con la tabla 6.6.

Con el **Algoritmo6-15-matriz** se insertan de manera automática elementos aleatorios a un arreglo de memoria bidimensional y se determina el elemento mayor de la matriz. Se ha omitido el módulo de **INSERCIÓN** por ser igual que en el ejemplo anterior (véanse figuras 6.48 a 6.50).

El **Algoritmo6-16-matriz** realiza la búsqueda secuencial de un número en arreglo de memoria bidimensional, se imprime si éste fue encontrado o no. Los elementos pueden repetirse y se deberá imprimir la cantidad de veces que se ha hallado el número (véanse figuras 6.51 a 6.54).

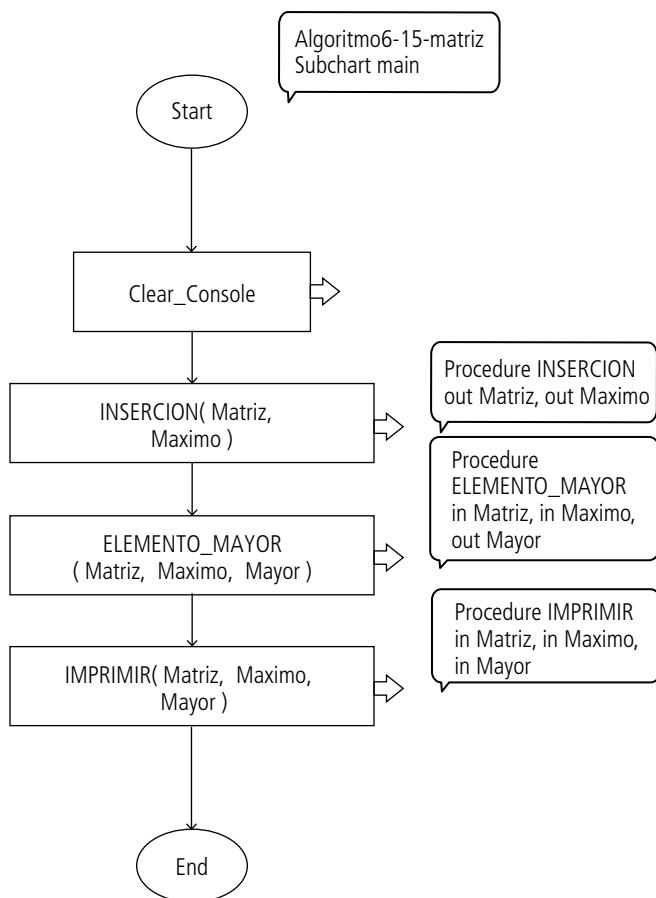


Figura 6.48 Algoritmo6-15-matriz Subchart main.

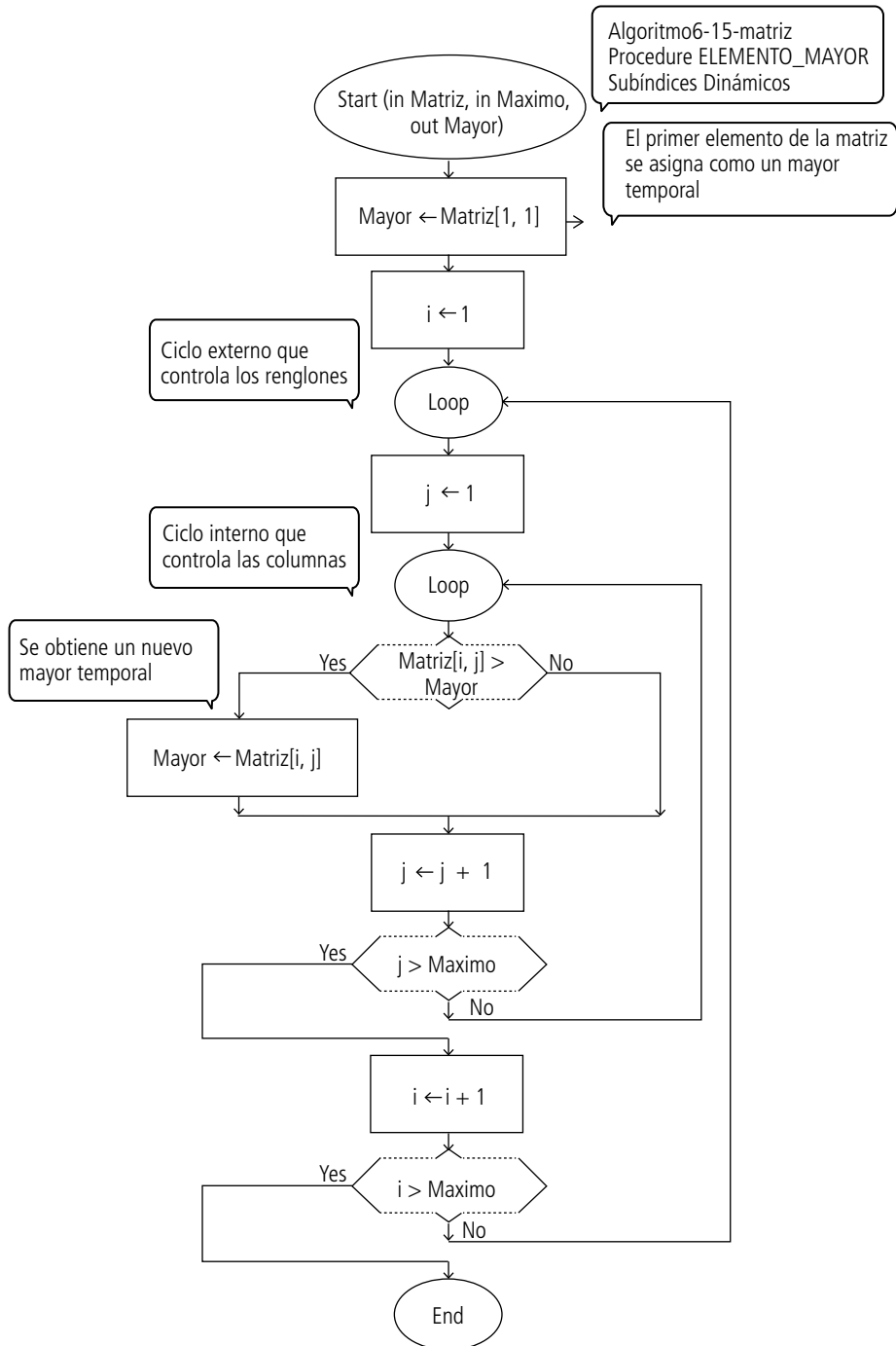


Figura 6.49 Algoritmo6-15-matriz Procedure ELEMENTO_MAYOR.

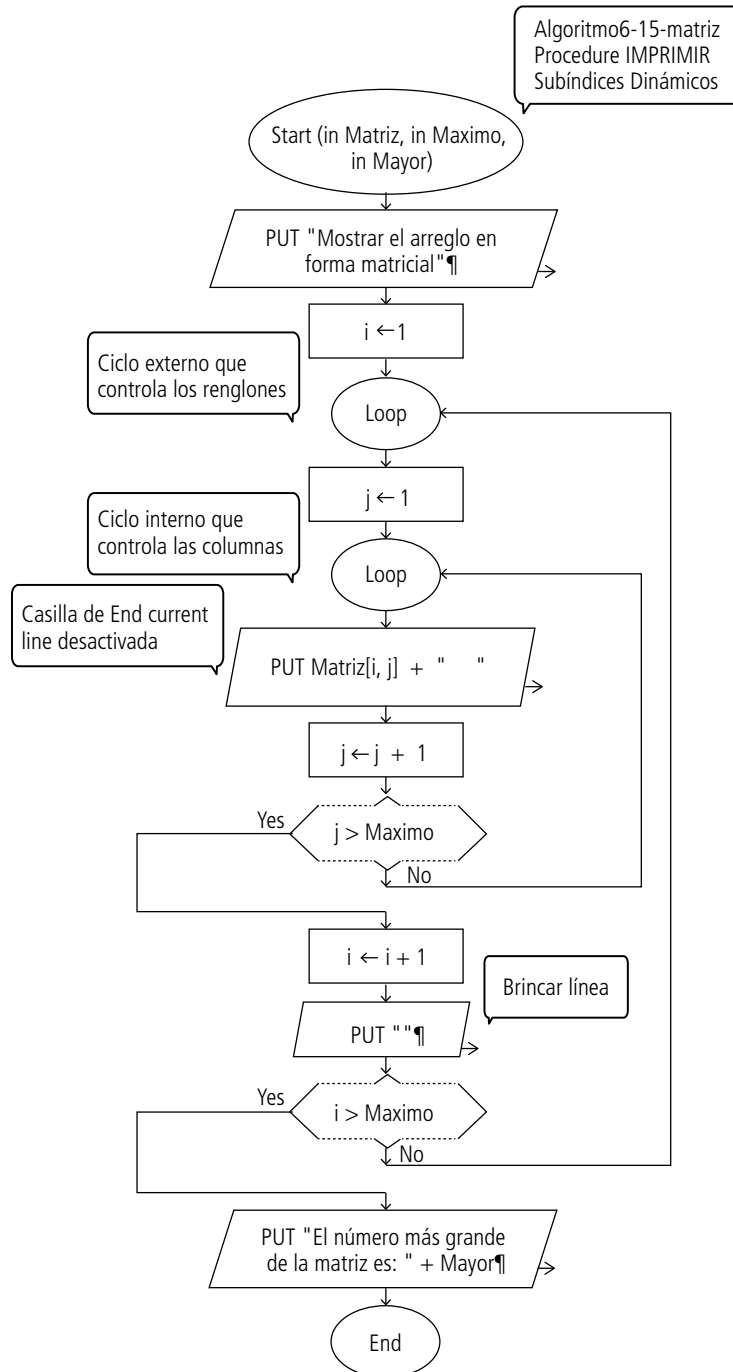


Figura 6.50 Algoritmo6-15-matriz Procedure IMPRIMIR.

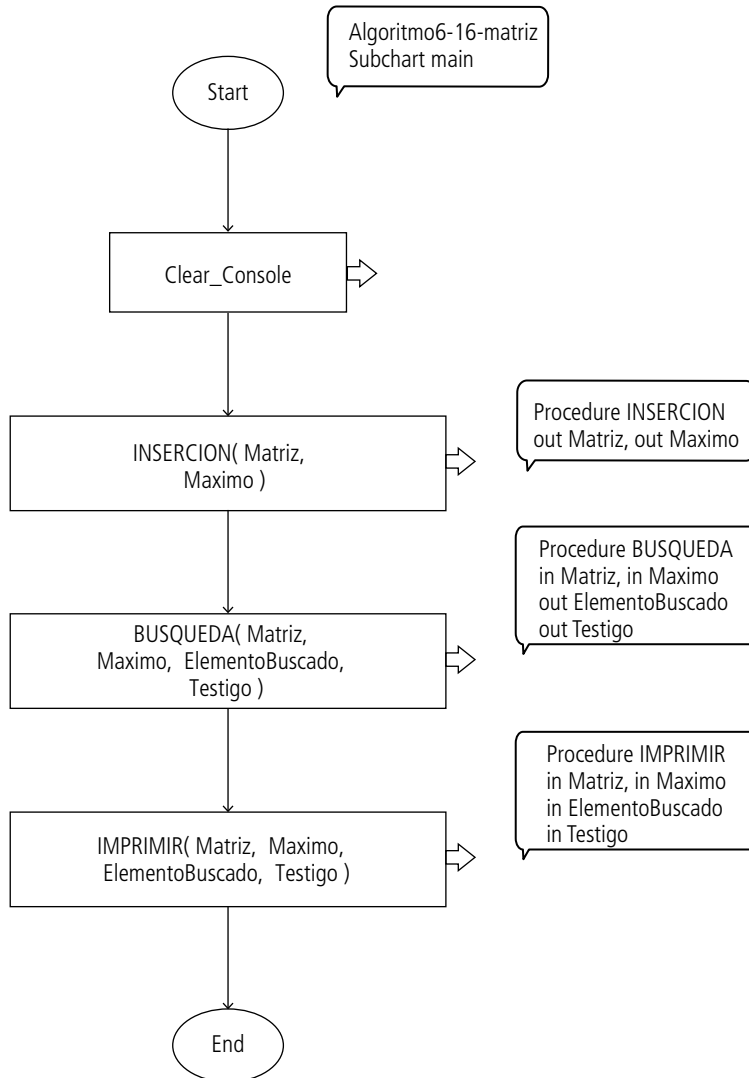


Figura 6.51 Algoritmo6-16-matriz Subchart main.

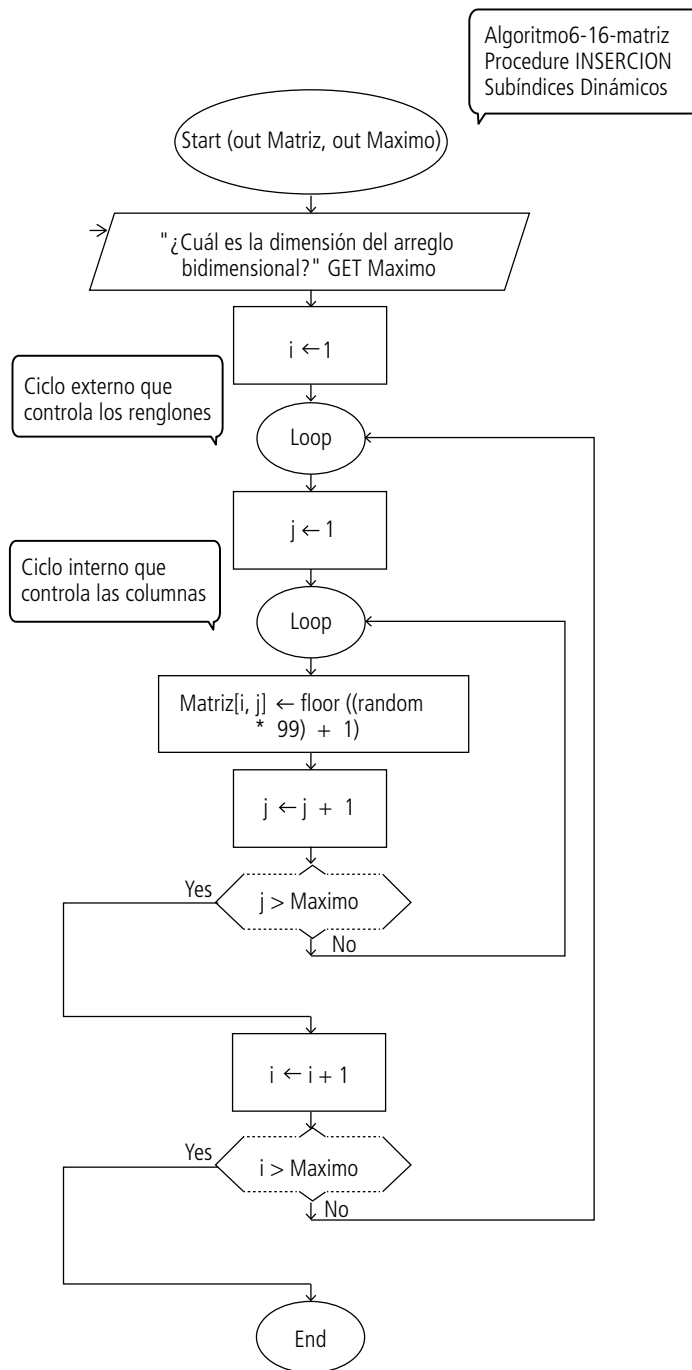


Figura 6.52 Algoritmo6-16-matriz Procedure INSERCIÓN.

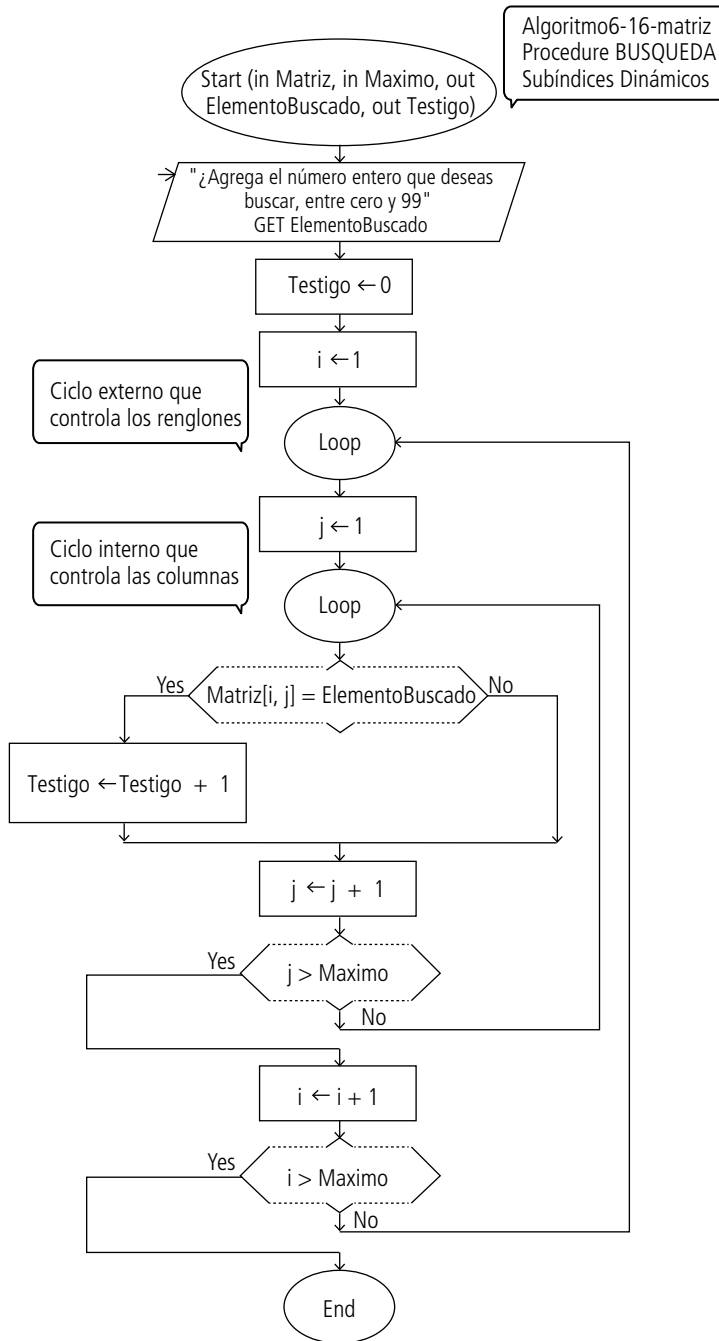


Figura 6.53 Algoritmo6-16-matriz *Procedure* BUSQUEDA.

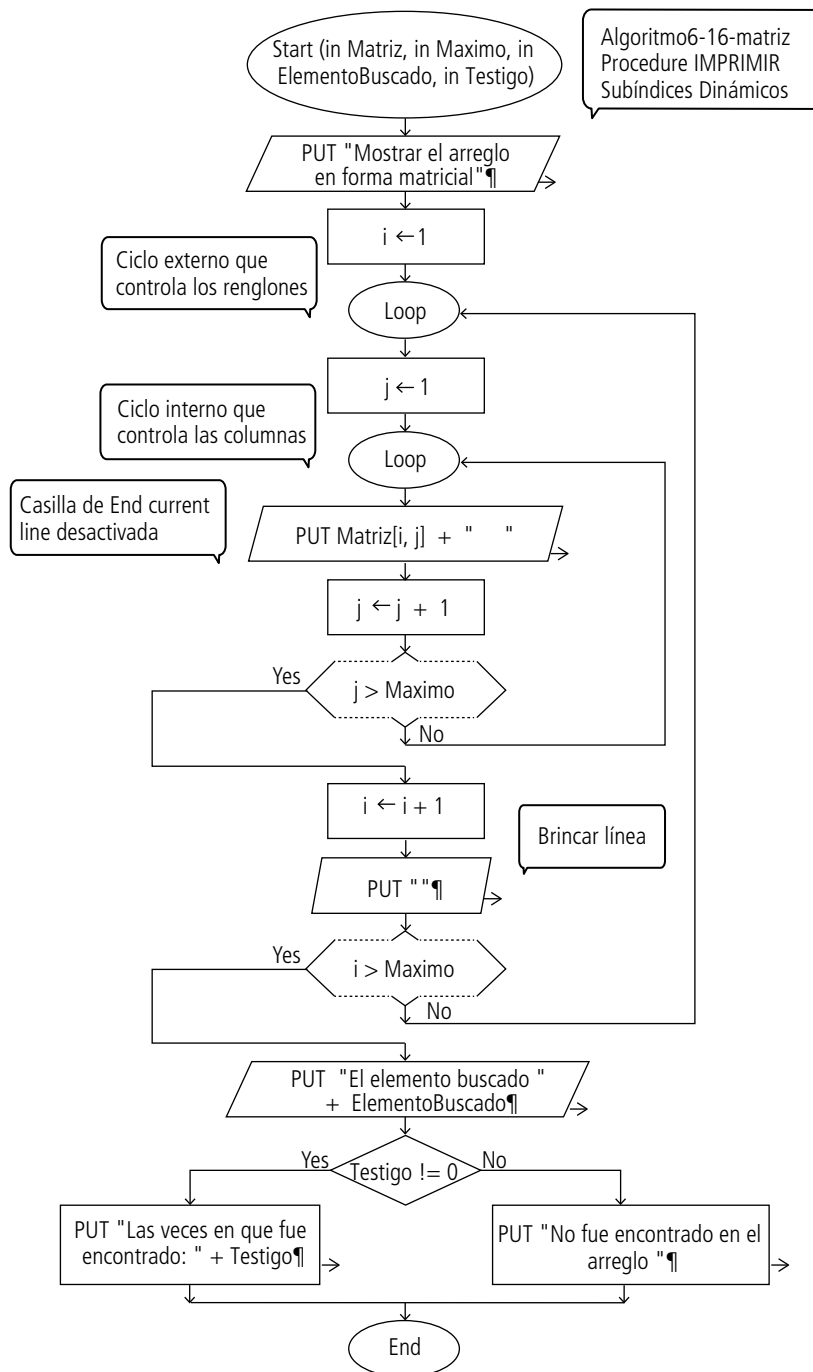


Figura 6.54 Algoritmo6-16-matriz Procedure IMPRIMIR.



6.8 Proceso de datos con corte de control

Cuando se tiene un conjunto de datos relacionados y de diverso tipo que identifican a una misma entidad y desean procesarse para obtener un reporte, se debe recurrir a la técnica de corte de control. Esta técnica tiene como condición que los datos tengan un dato en común que los identifique como pertenecientes a la misma entidad y que estén agrupados de manera contigua. Si estas condiciones no se cumplen, entonces no puede usarse esta técnica.

Por lo general, se recurre al ordenamiento por medio del campo en común de los datos para asegurar el agrupamiento contiguo de los datos. Una vez que se ha logrado lo anterior, entonces se usa la técnica de corte de control.

Considere a un maestro que aplica varias actividades a sus alumnos y a partir de esos datos obtiene un promedio final. Los datos tienen en común la matrícula de los estudiantes (véanse tablas 6.7 y 6.8).

Tabla 6.7 Ejemplo incompleto de los datos de los estudiantes

Matrícula	Descripción	Calificación de la actividad
1111111	Actividad A	90
1111111	Actividad B	50
1111111	Actividad C	80
1222222	Actividad A	60
1222222	Actividad B	80
1222222	Actividad C	70
1333333	Actividad A	100
1333333	Actividad B	90
1333333	Actividad C	80

Tabla 6.8 Resultados del procesamiento de datos de la tabla 6.7

Matrícula	Cantidad de actividades	Promedio
1111111	3	73.33
1222222	3	70.00
1333333	3	90.00

Cuando se tienen datos semejantes a los mostrados en la tabla 6.7 se procede primeramente a capturarlos en arreglos de memoria. En este caso, serán unidimensionales y se requerirá un arreglo por cada dato diferente. La captura no requiere que se haga en forma ordenada. Posteriormente, se hace un ordenamiento de los datos, ya sea ascendente o descendente, para que queden contiguos. Por último, se utiliza el procesamiento por corte de control y se obtienen los resultados deseados, de acuerdo con el Algoritmo6-17-CorteControl (véanse figuras 6.55 a 6.62).

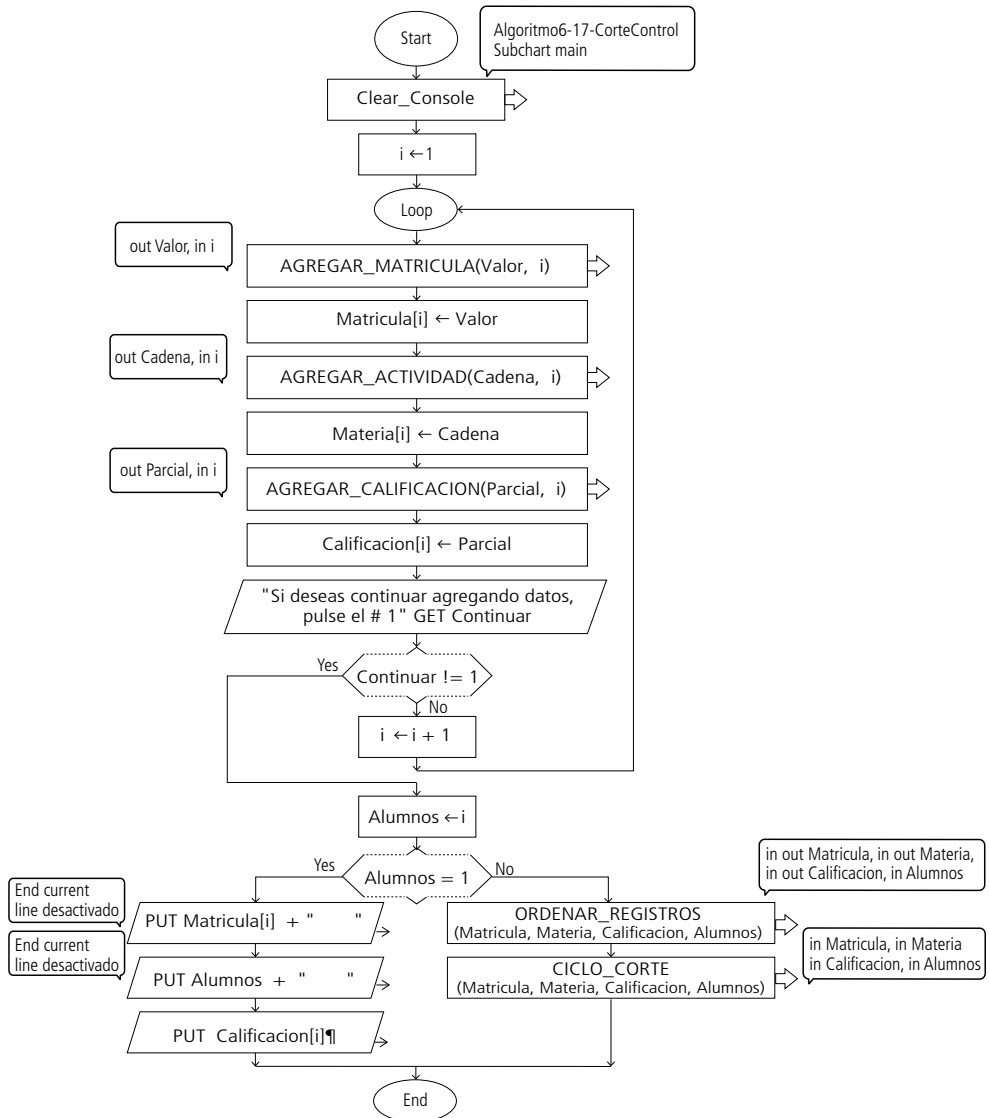


Figura 6.55 Algoritmo6-17-Corte Control Subchart main.

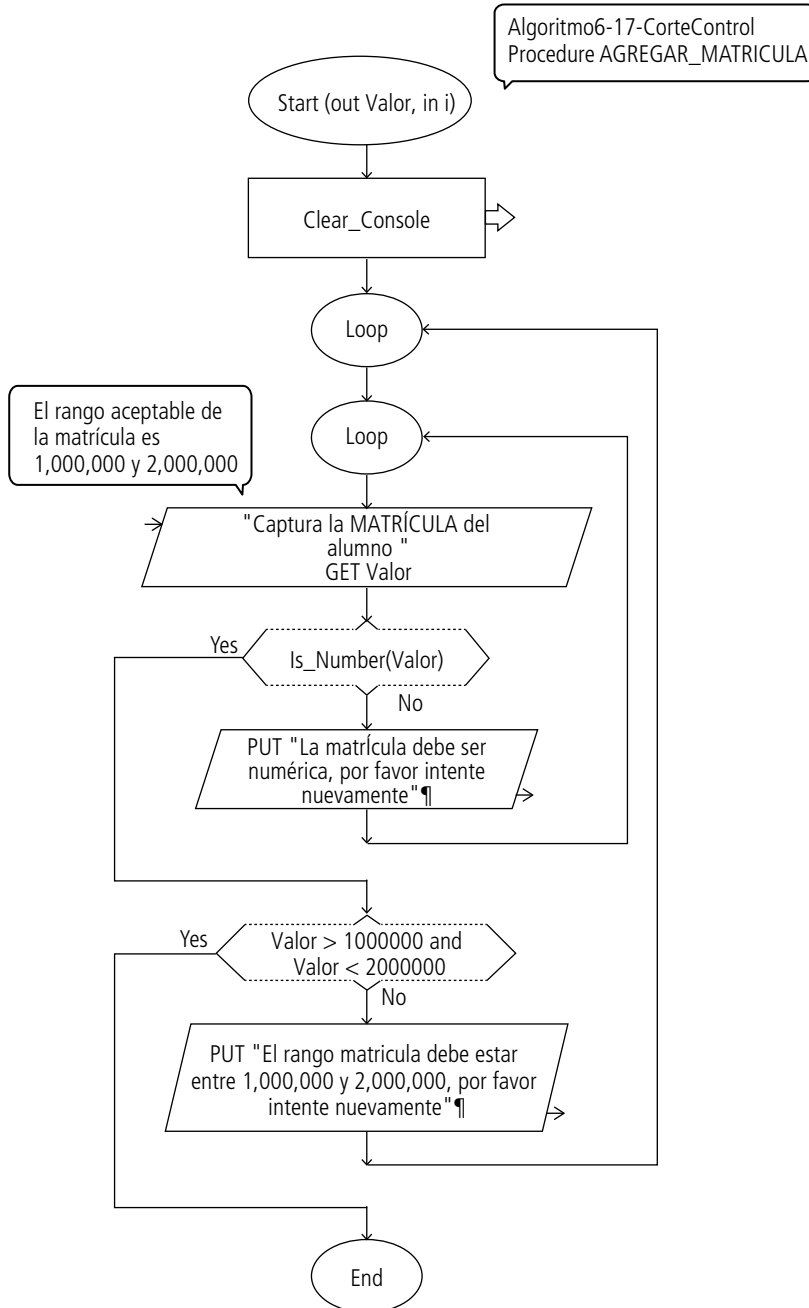


Figura 6.56 Algoritmo6-17-CorteControl Procedure AGREGAR_MATRICULA.

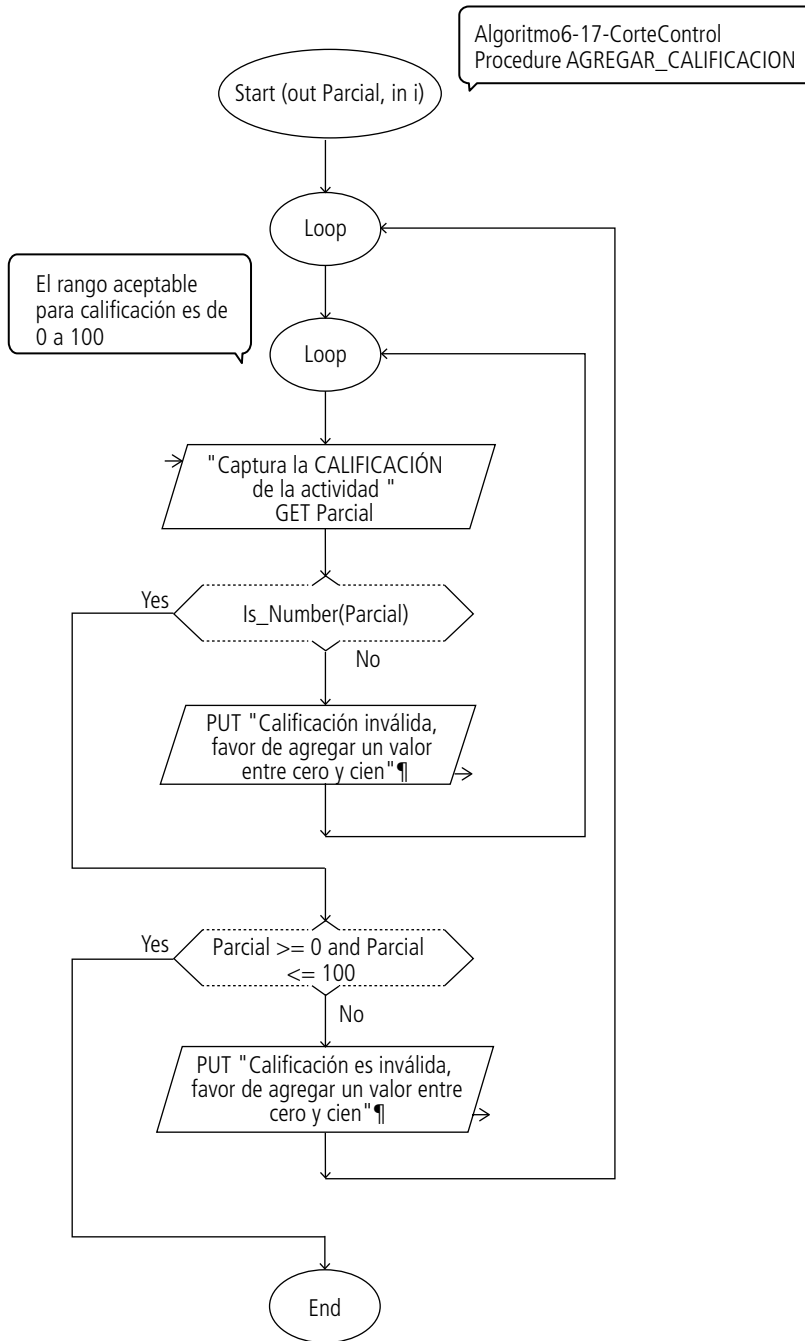


Figura 6.57 Algoritmo6-17-CorteControl *Procedure* AGREGAR_CALIFICACION.

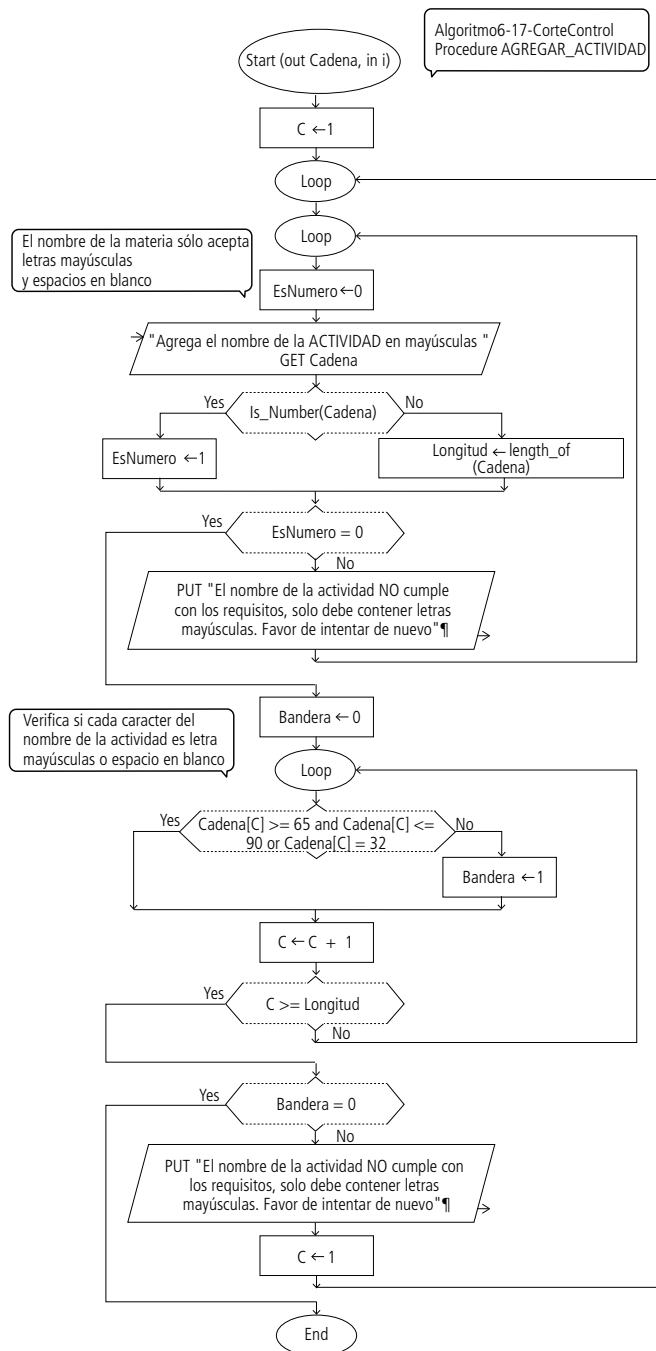


Figura 6.58 Algoritmo6-17-CorteControl Procedure AGREGAR_ACTIVIDAD.

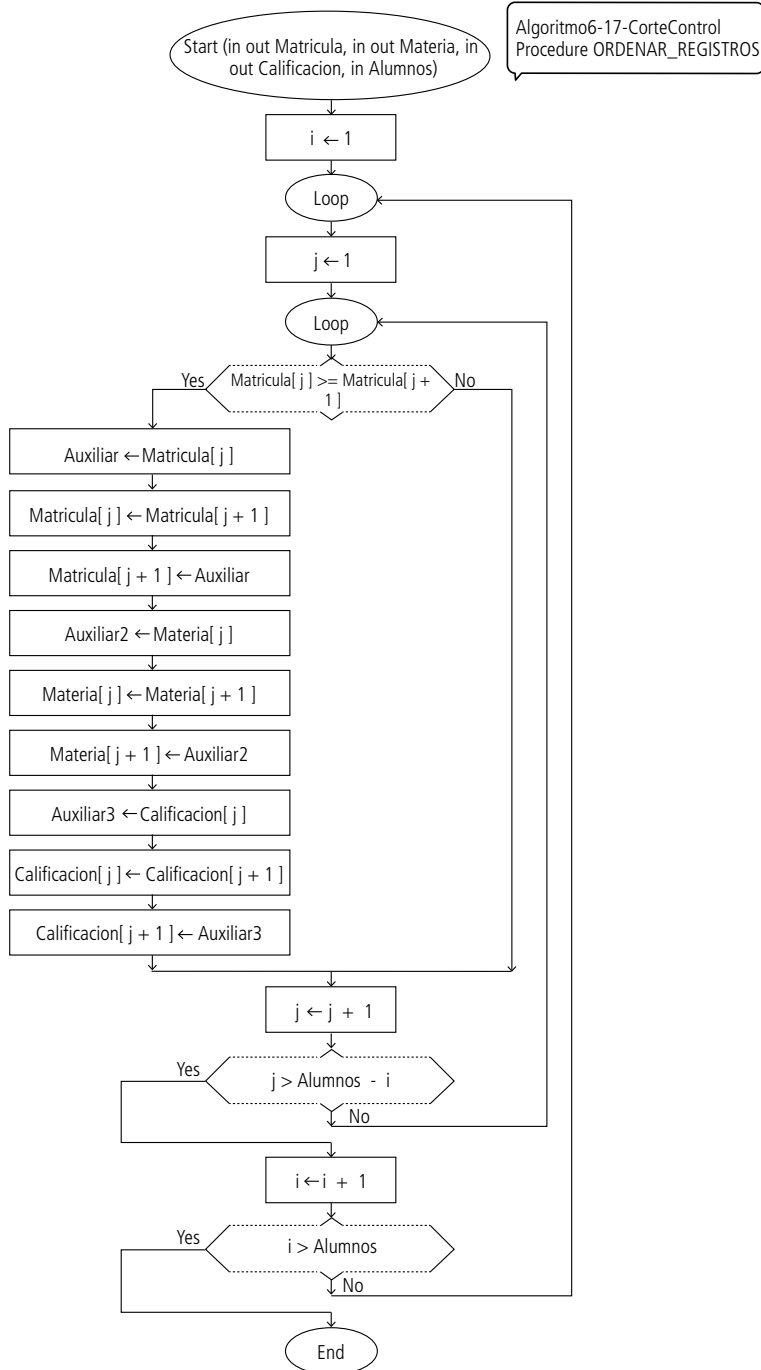


Figura 6.59 Algoritmo6-17-CorteControl Procedure ORDENAR_REGISTROS.

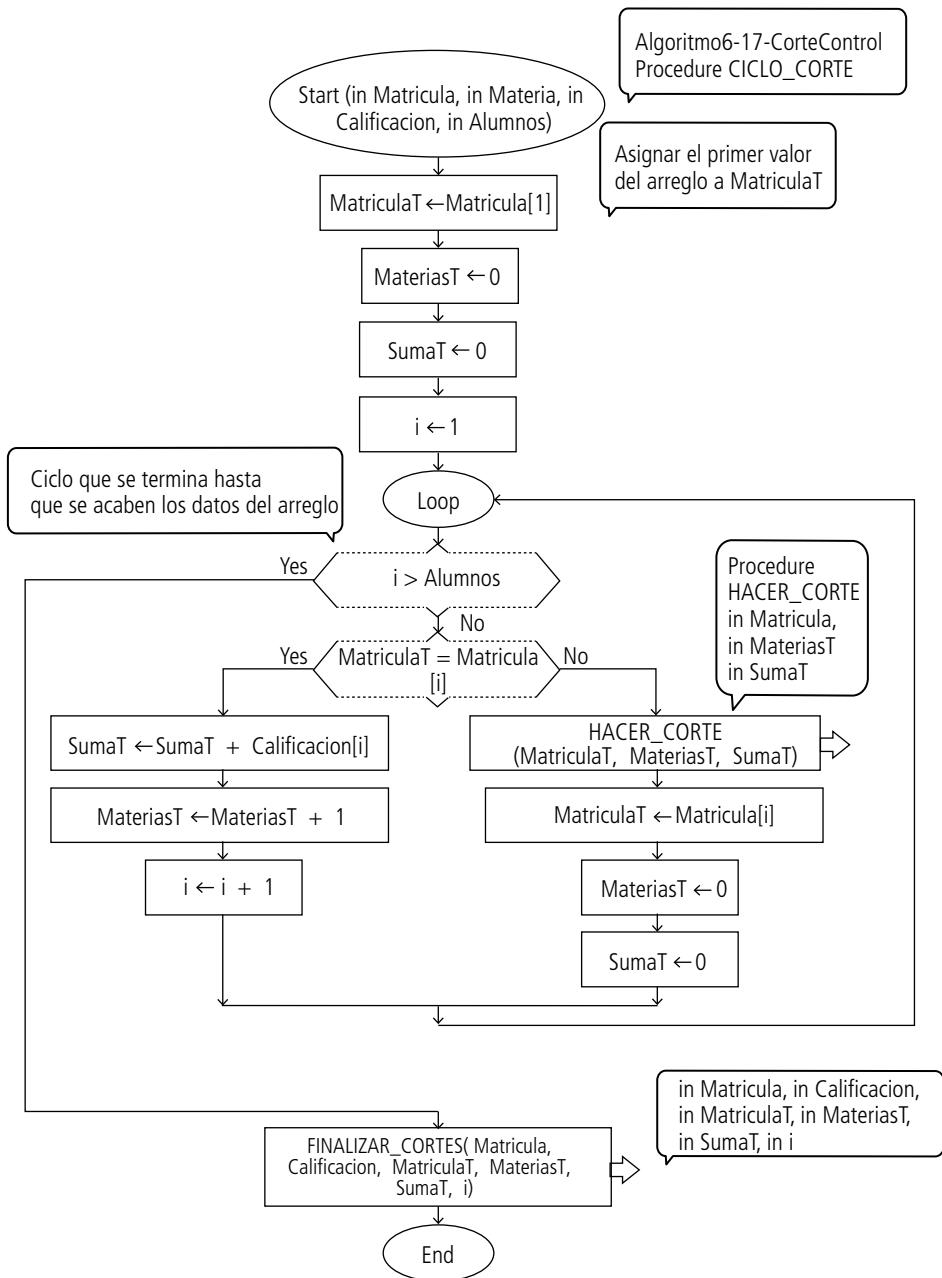


Figura 6.60 Algoritmo6-17-CorteControl Procedure CICLO_CORTE.

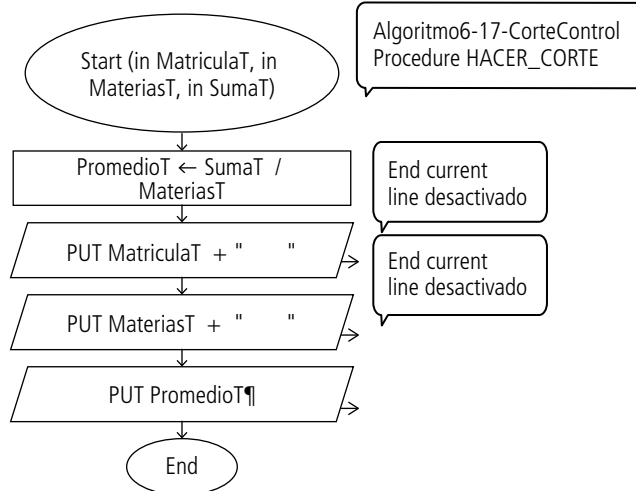


Figura 6.61 Algoritmo6-17-CorteControl Procedure HACER_CORTE.

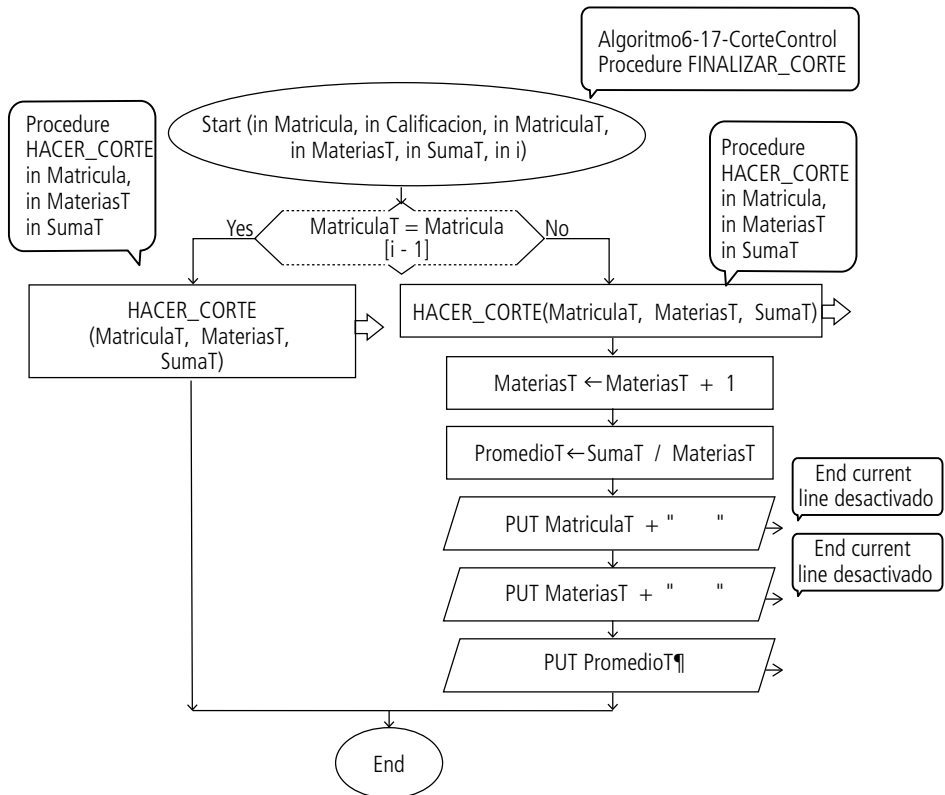


Figura 6.62 Algoritmo6-17-CorteControl Procedure FINALIZAR_CORTE.



6.9 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 6.9.

- 6.1 Cuando un conjunto de datos tiene las mismas características se dice que son del mismo _____.
- 6.2 Cuando un conjunto de datos pertenecen al mismo contexto de información se les dice que están _____.
- 6.3 ¿Cómo se denomina la estructura de datos que consiste en un grupo consecutivo de localidades de la memoria RAM relacionadas por tener el mismo identificador y las mismas características? Cada localidad puede referirse mediante el mismo nombre de variable y un subíndice que indica el lugar que ocupa cada una de ellas.
- 6.4 ¿Cómo se denomina la localidad de un arreglo de memoria?
- 6.5 ¿Cómo se denomina el elemento que es único y que permite identificar el lugar que ocupa una celda dentro de un arreglo de memoria unidimensional?
- 6.6 ¿Cómo se denominan informalmente los arreglos de memoria unidimensional?
- 6.7 ¿Cuáles son los caracteres que acompañan, a manera de sufijo, a los identificadores de las variables que son arreglo de memoria?
- 6.8 Cuando el elemento que está entre corchetes de un arreglo de memoria es una constante, se dice que es un subíndice _____.
- 6.9 Cuando el elemento que está entre corchetes de un arreglo de memoria es una variable, se dice que es un subíndice _____.
- 6.10 ¿Cómo se denomina la estructura de datos que consiste en un grupo de localidades o celdas de la memoria RAM relacionadas por tener el mismo identificador y las mismas características? Cada localidad puede referirse mediante el mismo nombre de variable y dos subíndices que indican la posición que ocupa cada una de ellas, es decir, en la intersección de un renglón con una columna.
- 6.11 ¿Cómo se denominan informalmente los arreglos de memoria bidimensional?
- 6.12 Los subíndices de un arreglo de memoria bidimensional se refieren a la _____ que ocupa un elemento en un arreglo matricial.
- 6.13 En un sistema de ecuaciones lineales que consiste en un número igual de ecuaciones y de incógnitas, ¿cómo se llama la matriz que únicamente se refiere a los valores de acompañan a las incógnitas?

- 6.14** En un sistema de ecuaciones lineales que consiste en un número igual de ecuaciones y de incógnitas, ¿cómo se llama la matriz que se refiere a todos los valores de las ecuaciones?
- 6.15** Por convención, el primer subíndice de un arreglo matricial se refiere a los _____.
- 6.16** Por convención, el segundo subíndice de un arreglo matricial se refiere a las _____.
- 6.17** La posición de los elementos de una matriz donde el renglón es igual que la columna pertenece a la _____.
- 6.18** La posición de los elementos de una matriz donde la suma de un renglón más la columna es igual que la dimensión de la matriz más uno pertenece a la _____.
- 6.19** La posición de los elementos de una matriz donde el renglón es menor que la columna pertenece a la _____.
- 6.20** La posición de los elementos de una matriz donde el renglón es mayor que la columna pertenece a la _____.

Tabla 6.9 Respuestas a los ejercicios de autoevaluación

6.1 Tipo	6.2 Relacionados	6.3 Arreglo de memoria unidimensional
6.4 Celda	6.5 Subíndice	6.6 Vector
6.7 Corchetes	6.8 Estático	6.9 Dinámico
6.10 Arreglo de memoria bidimensional	6.11 Matrices	6.12 Posición
6.13 Matriz de coeficientes	6.14 Matriz aumentada	6.15 Renglones
6.16 Columnas	6.17 Diagonal principal	6.18 Diagonal secundaria
6.19 Matriz triangular superior	6.20 Matriz triangular inferior	



6.10 Problemas propuestos: arreglos de memorias unidimensionales y bidimensionales

- 6.1** Realice un programa que imprima la tabla de multiplicar de cualquier número.
- 6.2** Realice un programa que haga la búsqueda secuencial de un número en una lista de N números e imprima si fue encontrado o no. Suponga que los elementos de la lista no se repiten.

- 6.3** Realice un programa que sume los elementos de dos listas de números y guarde los valores en una tercera lista. Ambas tienen la misma cantidad de elementos.
- 6.4** En dos arreglos unidimensionales se tiene la siguiente información de N personas. En el primero, se tiene la altura en metros. En el segundo, se tiene el peso en kilogramos. En un tercer arreglo unidimensional deberá calcular el Índice de Masa Corporal (IMC), según la siguiente fórmula:

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2}$$

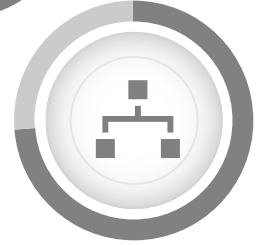
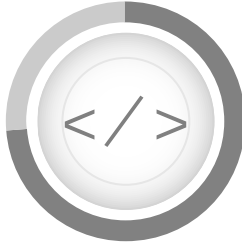
En un cuarto arreglo unidimensional deberá almacenar el resultado de acuerdo con el IMC, según la siguiente tabla.

Tabla 6.10 Información de N personas

Índice de masa corporal	Diagnóstico
Menos de 19	Bajo en peso
19 a 25	Peso normal
25 a 30	Sobrepeso
Mayor de 30	Obesidad

- Elabore un programa que determine e imprima el IMC promedio.
 - Determine e imprima cuántas y cuáles personas están por encima o igual que el promedio, y cuántas y cuáles por debajo del mismo.
 - Establezca e imprima cuántas personas pertenecen a cada categoría del diagnóstico y el porcentaje que representan del total.
- 6.5** Se deja caer una pelota desde una altura de 50 metros, rebota y cada vez su altura es de dos tercios de su altura en el rebote anterior.
- Elabore un programa que determine e imprima la altura de la pelota desde el primer rebote hasta que ésta sea igual o menor a un centímetro.
 - Determine e imprima en cuál rebote la altura de la pelota es igual o inferior a 50 centímetros.
 - Calcule e imprima la altura promedio de la pelota en todos los rebotes.
 - Calcule e imprima cuántos rebotes le proporcionan a la pelota una altura superior al promedio.

- 6.6** En un arreglo bidimensional se tienen almacenadas las calificaciones de diez alumnos en cuatro exámenes diferentes. Realice un programa que obtenga:
- a) El promedio de calificaciones de los diez alumnos en los cuatro exámenes.
 - b) El alumno o alumnos que obtuvieron la mayor calificación en el tercer examen; en cualquier caso debe imprimir cuántos alumnos fueron.
 - c) El examen en el que el promedio del grupo fue el más alto.
 - d) Cuántos alumnos están aprobados y cuántos reprobados, así como el porcentaje que representa.
- 6.7** Construya un algoritmo que sume los elementos centrales de una matriz.
- 6.8** Construya un algoritmo que sume los elementos periféricos de una matriz.
- 6.9** Construya un algoritmo que sume dos matrices de igual dimensión y guarde el resultado en una tercera matriz. Imprima las tres matrices, pero sólo haga un módulo para imprimir las tres matrices, mediante la reutilización del algoritmo de impresión.
- 6.10** Construya un algoritmo que determine cuántos elementos pares y cuántos impares hay en una matriz y cuántos de ellos son cero.
- 6.11** Construya un algoritmo que sume las filas y las columnas de una matriz. Se deberá imprimir la sumatoria por fila y columnas.
- 6.12** Construya un algoritmo que multiplique matrices.
- 6.13** Construya un algoritmo que resuelva un sistema de ecuaciones lineales a través del método Gauss-Jordan y Montante.
- 6.14** En un arreglo bidimensional de doce por tres se tienen los costos de producción de tres departamentos: dulces, bebidas y conservas. Los costos de producción corresponden a cada mes del año anterior. Escriba un programa que pueda proporcionar la siguiente información.
- a) ¿En qué mes se registró el mayor costo de producción de dulces?
 - b) ¿Promedio anual de los costos de producción de bebidas?
 - c) ¿En qué mes se registró el mayor costo de producción en bebidas y en qué mes el de menor costo?
 - d) ¿Cuál fue el rubro que tuvo el menor costo de producción en diciembre?



Capítulo 7

Diseño de algoritmos con archivos de datos

- 7.1 Introducción
- 7.2 Almacenamiento de datos en archivos digitales
- 7.3 Lectura de datos desde archivos digitales
- 7.4 Agregar datos a un archivo digital
- 7.5 Tratamiento de información
- 7.6 Tratamiento de información con corte de control
- 7.7 Ejercicios de autoevaluación
- 7.8 Problemas propuestos: tratamiento básico de información
- 7.9 Problemas propuestos: corte de control

Objetivos de aprendizaje

Una vez que haya estudiado este capítulo, será capaz de:



- Diseñar algoritmos computacionales que permitan el almacenamiento de datos en dispositivos externos.
- Almacenar un conjunto de datos relacionados y de diferente tipo en un dispositivo externo.
- Procesar datos almacenados en dispositivos externos mediante la técnica de corte de control.



7.1 Introducción

Durante el estudio de los capítulos que anteceden a éste hemos estado almacenando datos y resultados en la memoria RAM. Este dispositivo de almacenamiento digital interno, también llamado primario, tiene la ventaja de procesar los datos de manera más rápida, pero su capacidad de almacenamiento es limitado comparado contra los dispositivos externos o secundarios. Entre estos últimos, podemos citar los más comunes: el disco duro, la memoria USB (*Universal Serial Bus*), los discos externos, las tarjetas SIM (*Subscriber Identity Module*), etcétera.

Otra diferencia entre los dispositivos internos y externos es la permanencia a lo largo del tiempo, esto es, la capacidad de recuperar los datos para su procesamiento en un periodo de tiempo largo. Se dice que la memoria RAM es volátil porque sólo puede asegurar el almacenamiento de datos mientras reciba suministro de energía eléctrica, es decir, si la computadora está encendida. Cuando ésta se apaga, todos los datos almacenados en la memoria RAM se pierden.

La opción que los usuarios tenemos para almacenar datos a largo plazo son los dispositivos secundarios. Por lo que en este capítulo abordaremos cómo los datos y resultados de los algoritmos se pueden almacenar en una memoria externa. En este caso, se almacenarán en archivos digitales, los cuales pueden ser abiertos y su contenido leído utilizando software de edición de textos como el block de notas  o un procesador de palabras como Word .



7.2 Almacenamiento de datos en archivos digitales

Los archivos de datos digitales que maneja Raptor se les conoce como archivos de texto, que se caracterizan por no tener un formato previo o predefinido. Se acostumbra agregar al nombre del archivo la extensión txt, pero no es obligatorio; por ejemplo, escolar.txt. La extensión queda a discreción del usuario pero se sugiere no utilizar extensiones que se asocien a otro tipo de archivos, como doc, xls, ppt, dbf, etcétera.

La acción computacional para almacenar datos a un archivo de texto requiere que se haga un redireccionamiento sobre el símbolo Output. De manera preestablecida, el símbolo Output envía sus mensajes a la ventana de *MasterConsole*. Para modificar

lo anterior, Raptor cuenta con la función *Redirect_Output*, la cual se utiliza dentro de un símbolo Call; con dicha función el símbolo Output enviará sus mensajes a un archivo de texto.

La información que se verá en el archivo de texto tiene el mismo formato que el que se muestra en la ventana de *MasterConsole*. Por tanto, de igual forma que el programador puede controlar lo que el símbolo Output despliega en *MasterConsole*, incluyendo líneas nuevas con el *End Current Line*, de esa misma manera tiene control de lo que se envía al archivo de texto.

Antes de que un archivo de datos pueda almacenar información, éste debe ser creado. La función *Redirect_Output* tiene la capacidad no sólo de crear el archivo, sino que lo deja abierto con el objetivo de que se puedan almacenar datos en él. A este respecto, se dice que el archivo se abre para escritura. Para que *Redirect_Output* pueda funcionar requiere que el programador escriba el parámetro de creación y apertura de un archivo que cambia de diversas formas.

Cuando el parámetro de creación y apertura de la función *Redirect_Output* contempla sólo el nombre del archivo, se creará el archivo en la carpeta donde se encuentra el diagrama que lo está ejecutando. El nombre del archivo se otorga a discreción del programador.

Cuando el parámetro de creación y apertura de la función *Redirect_Output* contempla la dirección completa de la carpeta y el nombre del archivo, se creará el archivo en la carpeta indicada. Sin embargo, si la dirección de la carpeta no existe, se generará un error de ejecución.

En los casos anteriores, si el nombre del archivo ya existe en la carpeta, se procede a eliminarlo sin que se haga advertencia al usuario y se crea un nuevo archivo con el nombre indicado. Esto es un riesgo porque si el archivo contenía información importante para el usuario se perderán los datos almacenados.

Cuando el parámetro de creación y apertura de la función *Redirect_Output* contempla la palabra *yes* o *true*, se detiene la ejecución del diagrama, se abre una ventana y le permite al usuario seleccionar el archivo de texto que recibirá el redireccionamiento de Output (véase tabla 7.1).

Tabla 7.1 Modalidades de la función *Redirect_Output*

Redirect_Output ("archivo.txt")	El archivo.txt se creará en la misma carpeta donde se encuentre el diagrama que ejecuta la función de creación y apertura. Observe que al nombre del archivo se le agregó una extensión.
Redirect_Output ("c:\algoritmos\archivo")	Se da la dirección completa de la carpeta donde se creará el archivo. El nombre del archivo no tiene extensión, pero podría llevarla.
Redirect_Output (true)	Se detiene la ejecución del diagrama, se abre una ventana y le permite al usuario seleccionar el archivo de texto que recibirá el redireccionamiento de Output.

En todas las modalidades, después del redireccionamiento, los subsecuentes símbolos Output que aparezcan en el diagrama enviarán sus mensajes al archivo que se ha especificado.

Cuando por alguna situación el archivo de datos ya no se va a utilizar, éste debe cerrarse. Esto se logra con la función *Redirect_Output* y se agrega el parámetro de *false*. Por ejemplo: *Redirect_Output (false)*. Una vez que se cerró el archivo, los subsecuentes Output vuelven a enviar sus mensajes a *MasterConsole*.

Como regla general, antes de agregar datos a un archivo digital primero se tiene que crear y abrir para escritura con la función *Redirect_Output*, contemplando alguna de sus modalidades. Cuando el archivo ya no se va a utilizar, se tiene que cerrar usando esta misma función, pero llevando como parámetro la opción *false*.

Para ejemplificar lo anterior, considere que deseamos crear un archivo de texto que contenga de manera exclusiva números aleatorios, enteros, positivos entre 1 y 999. En el **Algoritmo7-1-archivos**, primero, el usuario elige la cantidad de números aleatorios que quiere que haya en el archivo. Segundo, éstos se agregan a un arreglo de memoria unidimensional llamado Vector. Finalmente, los valores agregados a la variable Vector se envían y almacenan en el archivo de texto (véanse figuras 7.1 a 7.3).

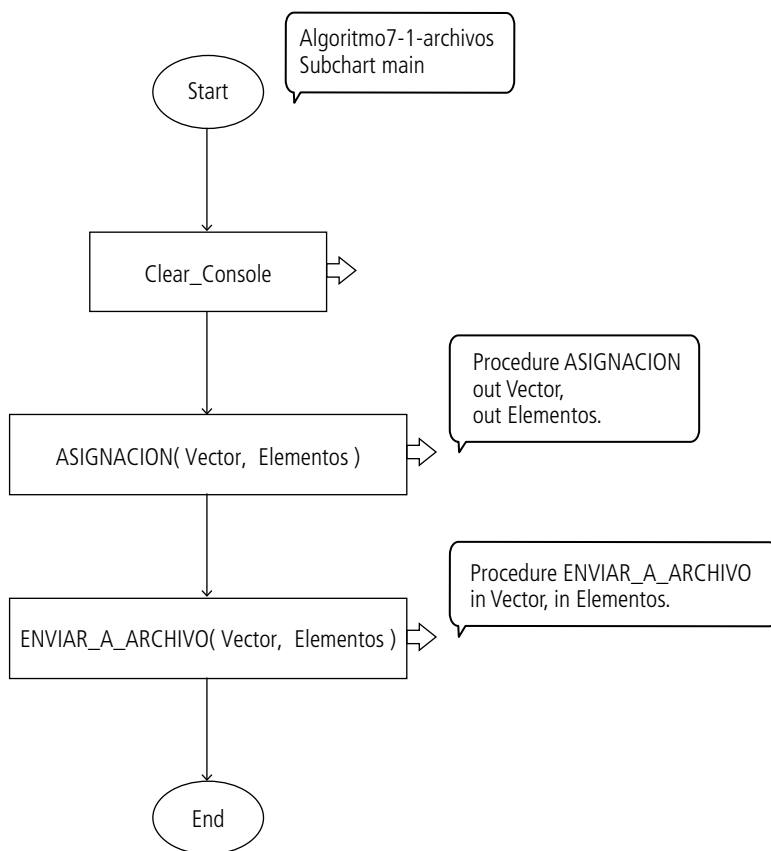


Figura 7.1 Algoritmo7-1-archivos *Subchart main*.

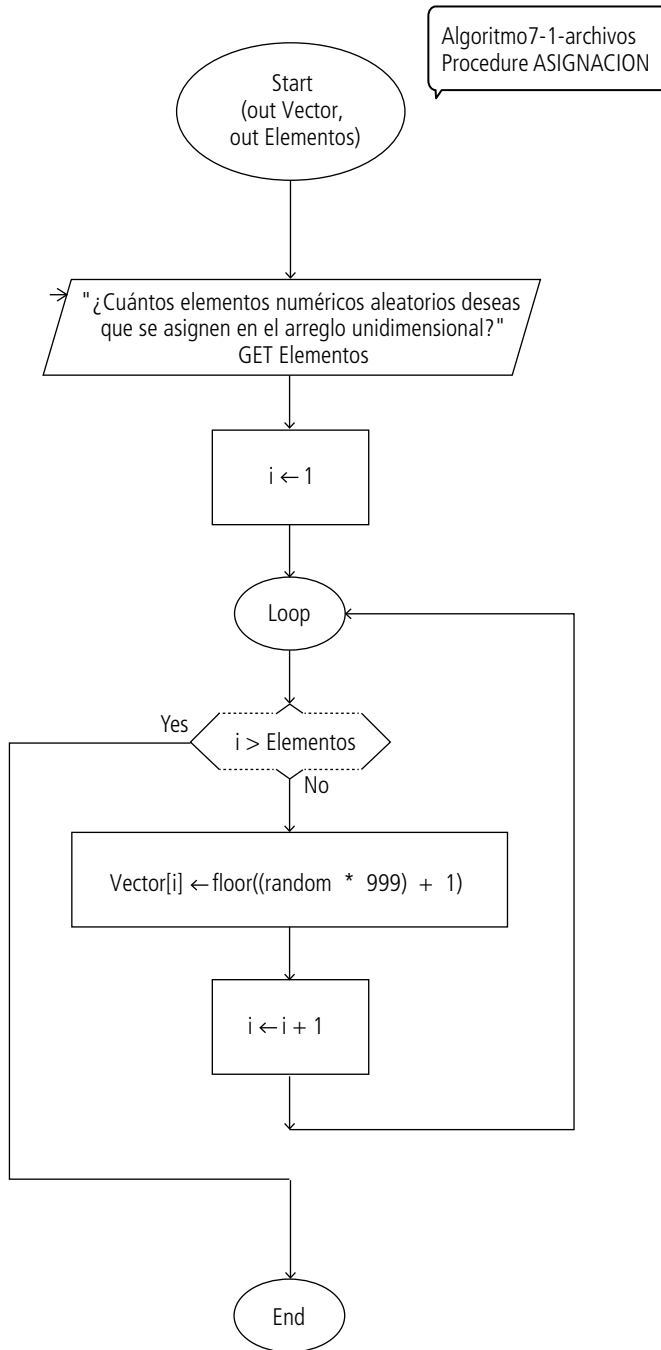


Figura 7.2 Algoritmo7-1-archivos *Procedure ASIGNACION*.

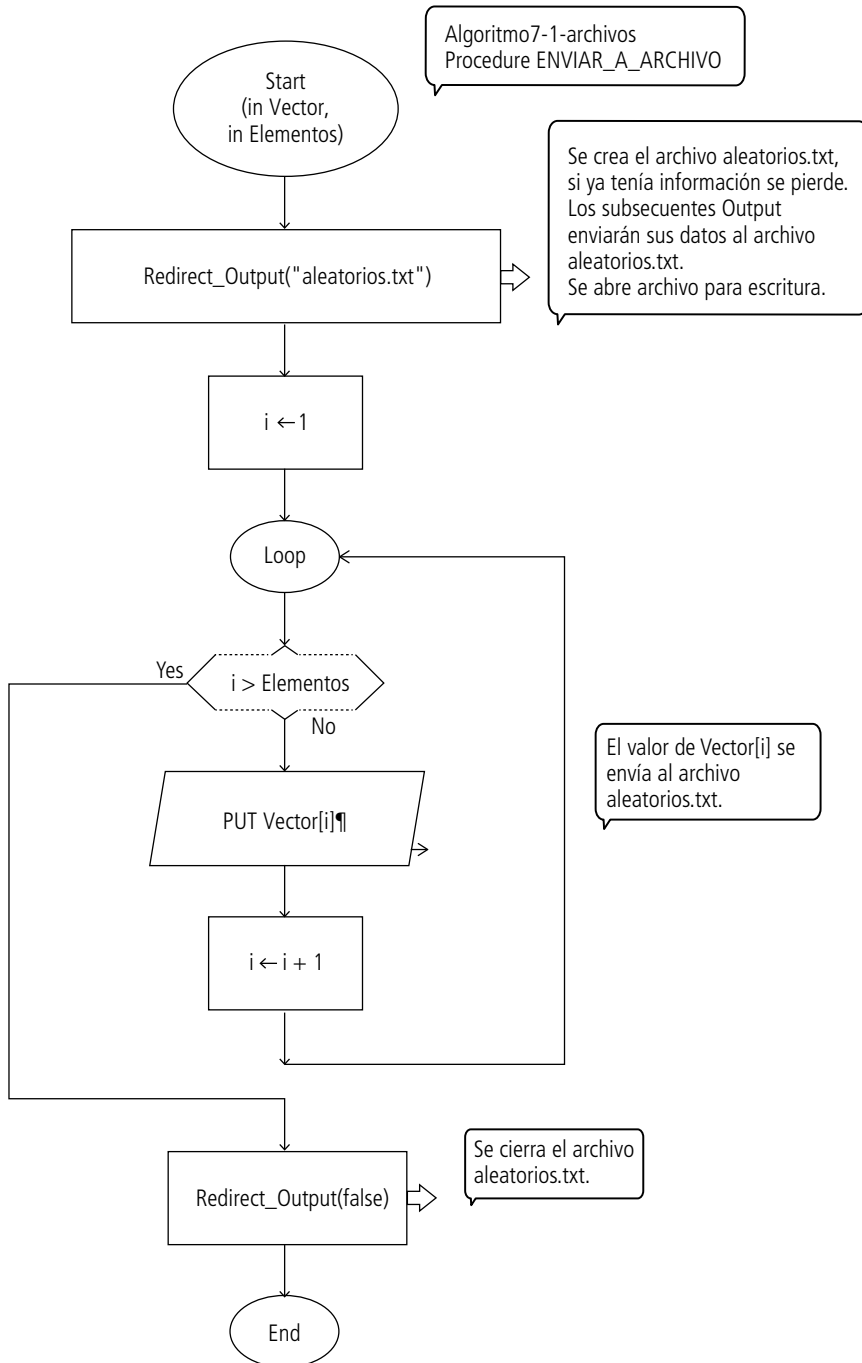


Figura 7.3 Algoritmo7-1-archivos *Procedure ENVIAR_A_ARCHIVO*.

Después de ejecutar el **Algoritmo7-1-archivos** utilice un editor de textos para abrir el archivo **aleatorios.txt** y verá algo semejante a la figura 7.4.

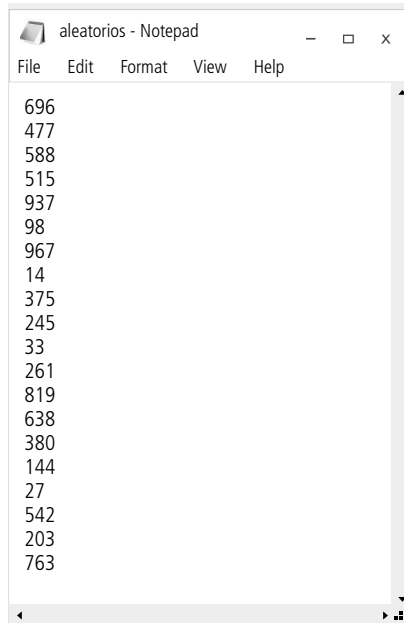


Figura 7.4 Datos contenidos en el archivo aleatorios.txt.



7.3 Lectura de datos desde archivos digitales

Cuando un archivo de texto contiene datos que deseamos procesar, lo que se debe hacer es la acción computacional de extraerlos del dispositivo externo y colocarlos en el dispositivo interno.

La acción computacional de extraer datos de un archivo requiere que se haga un redireccionamiento sobre el símbolo Input. De manera preestablecida, el símbolo Input permite al usuario utilizar el teclado para agregar datos a la memoria RAM. Para modificar lo anterior, Raptor cuenta con la función *Redirect_Input* que se utiliza dentro de un símbolo Call; con dicha función, el símbolo Input extraerá los datos del archivo especificado y los colocará en la memoria RAM, dándole valores a las variables indicadas.

Antes de que a un archivo se le puedan extraer datos, éste debe abrirse. La función *Redirect_Input* permite la apertura de un archivo en su modalidad de lectura. Para que *Redirect_Input* pueda funcionar requiere que el programador escriba el parámetro de apertura para lectura que puede variar de diversas formas.

Cuando el parámetro de apertura para lectura de la función *Redirect_Input* contempla únicamente el nombre del archivo, se busca el archivo indicado en la carpeta donde está almacenado el diagrama que lo ejecuta. Si el archivo no existe, se generará un error de ejecución.

Cuando el parámetro de apertura para lectura de la función *Redirect_Input* contempla la dirección completa de la carpeta y el nombre del archivo, se busca el archivo indicado en la carpeta señalada. Si el archivo o la dirección no existen, se generará un error de ejecución.

Cuando el parámetro de apertura para lectura de la función *Redirect_Input* contempla la palabra *yes* o *true*, se detiene la ejecución del diagrama, se abre una ventana y le permite al usuario seleccionar el archivo de texto que recibirá el redireccionamiento de Input (véase tabla 7.2).

Tabla 7.2 Modalidades de la función *Redirect_Input*

Redirect_Input ("archivo.txt")	El archivo de donde se extraerán los datos debe residir en la misma carpeta donde se encuentra el diagrama que se está ejecutando.
Redirect_Input ("c:\algoritmos\archivo.txt")	Se ha dado la dirección completa de la carpeta donde se debe buscar el archivo del cual se extraerán los datos, así como el nombre del mismo.
Redirect_Input (true)	Se detiene la ejecución del diagrama, se abre una ventana y le permite al usuario seleccionar el archivo de texto que recibirá el redireccionamiento de Input.
En las modalidades que contempla el nombre del archivo se requiere la extensión en el nombre del archivo. Después del redireccionamiento, los subsecuentes símbolos Input que aparezcan en el diagrama utilizarán el archivo especificado para extraer datos e insertarlos en la memoria RAM en las variables indicadas.	

Cuando por alguna situación el archivo de datos ya no se va a utilizar, el archivo debe cerrarse. Esto se logra con la función *Redirect_Input* y se agrega el parámetro de *false*. Ejemplo de uso: *Redirect_Input (false)*. Una vez cerrado el archivo, los subsecuentes Input permitirán al usuario utilizar el teclado para insertar valores a la memoria RAM.

Como regla general, antes de extraer los datos a un archivo digital, primero se tiene que abrir el archivo usando la función *Redirect_Input* contemplando alguna de sus modalidades. Cuando el archivo ya no se va a utilizar, se tiene que cerrar usando esta misma función, pero llevando como parámetro la opción *false*.

Cuando la extracción de datos con el símbolo Input se hace línea por línea, es común que el programador no sepa la cantidad de datos que contiene el archivo. Para poder extraer todos los datos, se puede utilizar la función *End_Of_Input*, la cual regresa un valor verdadero una vez extraídas todas las líneas del archivo. Por lo general, esta función se utiliza como condición de control en el símbolo Loop.

A manera de ejemplo, con el **Algoritmo7-2-archivos** vamos a extraer los datos del archivo creado con el **Algoritmo7-1-archivos**, el cual lleva por nombre

aleatorios.txt. Estos datos se extraerán del dispositivo externo y se colocarán en la memoria RAM en un arreglo unidimensional llamado vector. Luego, la variable Vector se someterá a un proceso de ordenamiento que permitirá que los números aleatorios estén de manera ordenada en forma ascendente, es decir, de menor a mayor. Finalmente, los datos almacenados en la variable Vector se enviarán a un archivo llamado **aleatorios_ordenados.txt**. Observe muy bien que en el *Procedure ORDENAR* la variable Vector es un argumento de entrada y salida; mencionamos lo anterior porque éste es uno de los pocos ejemplos en este libro que tiene esta característica (véanse figuras 7.5 a 7.8).

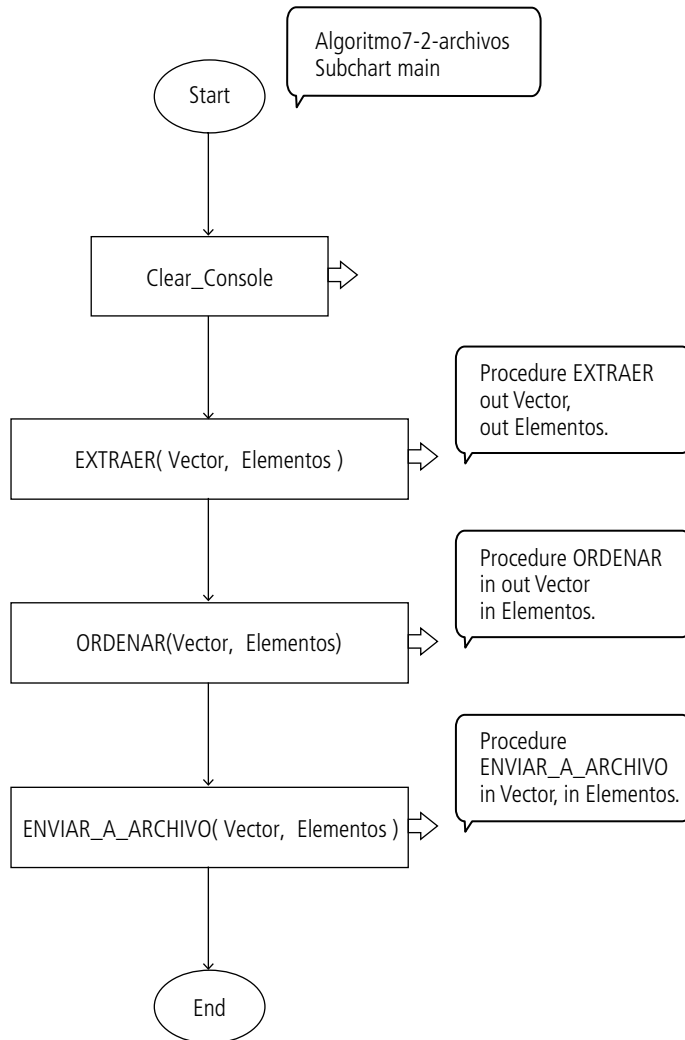


Figura 7.5 Algoritmo7-2-archivos Subchart main.

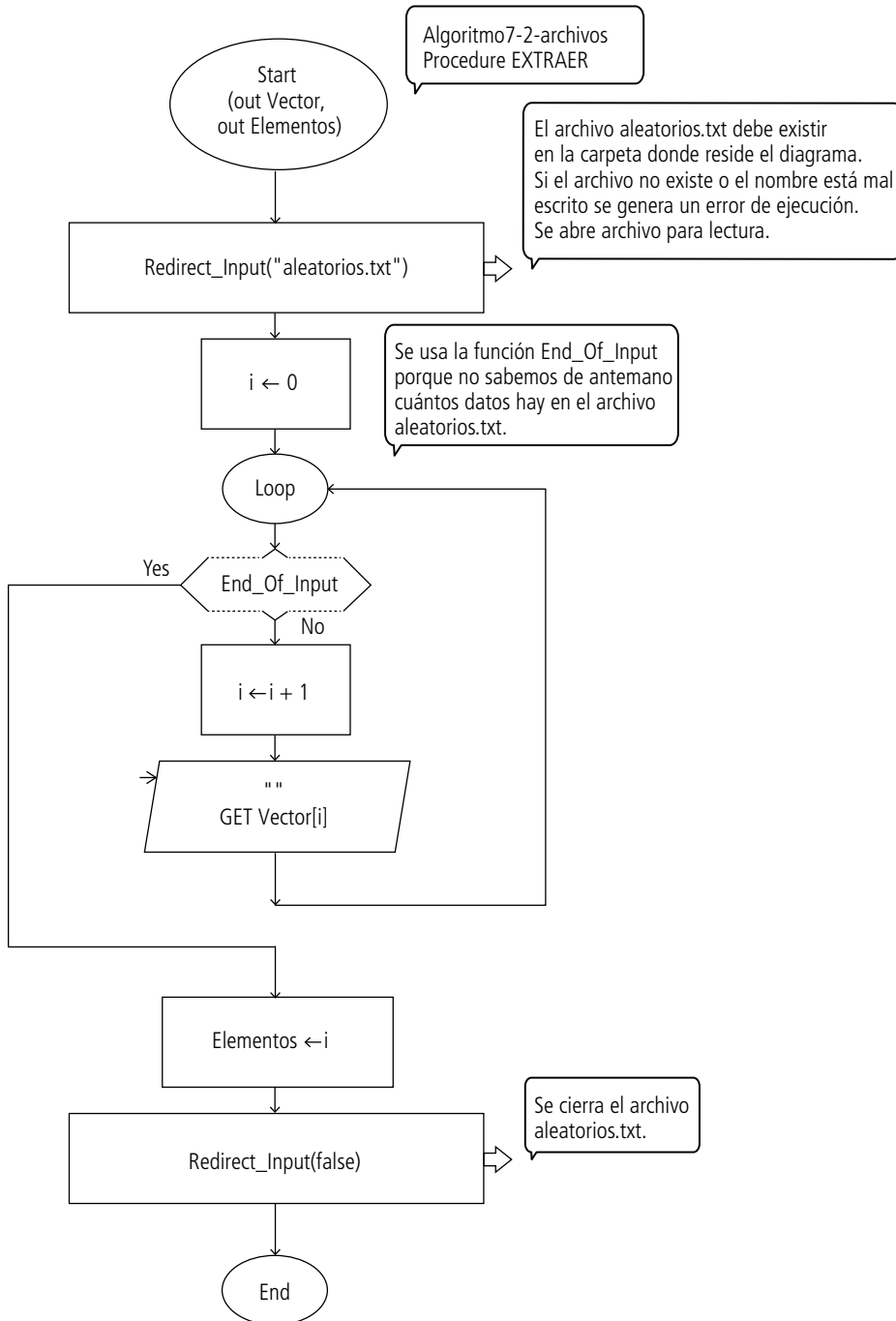


Figura 7.6 Algoritmo7-2-archivos *Procedure* EXTRAER.

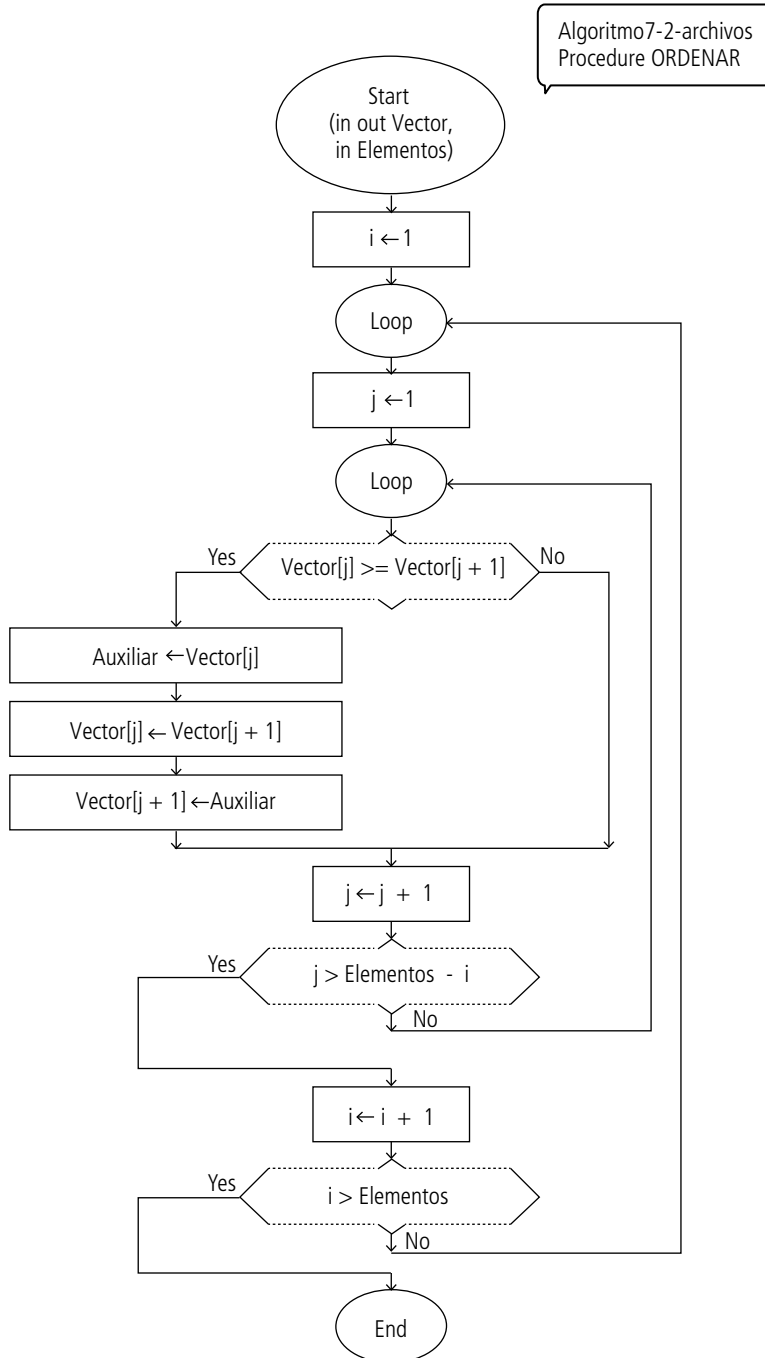
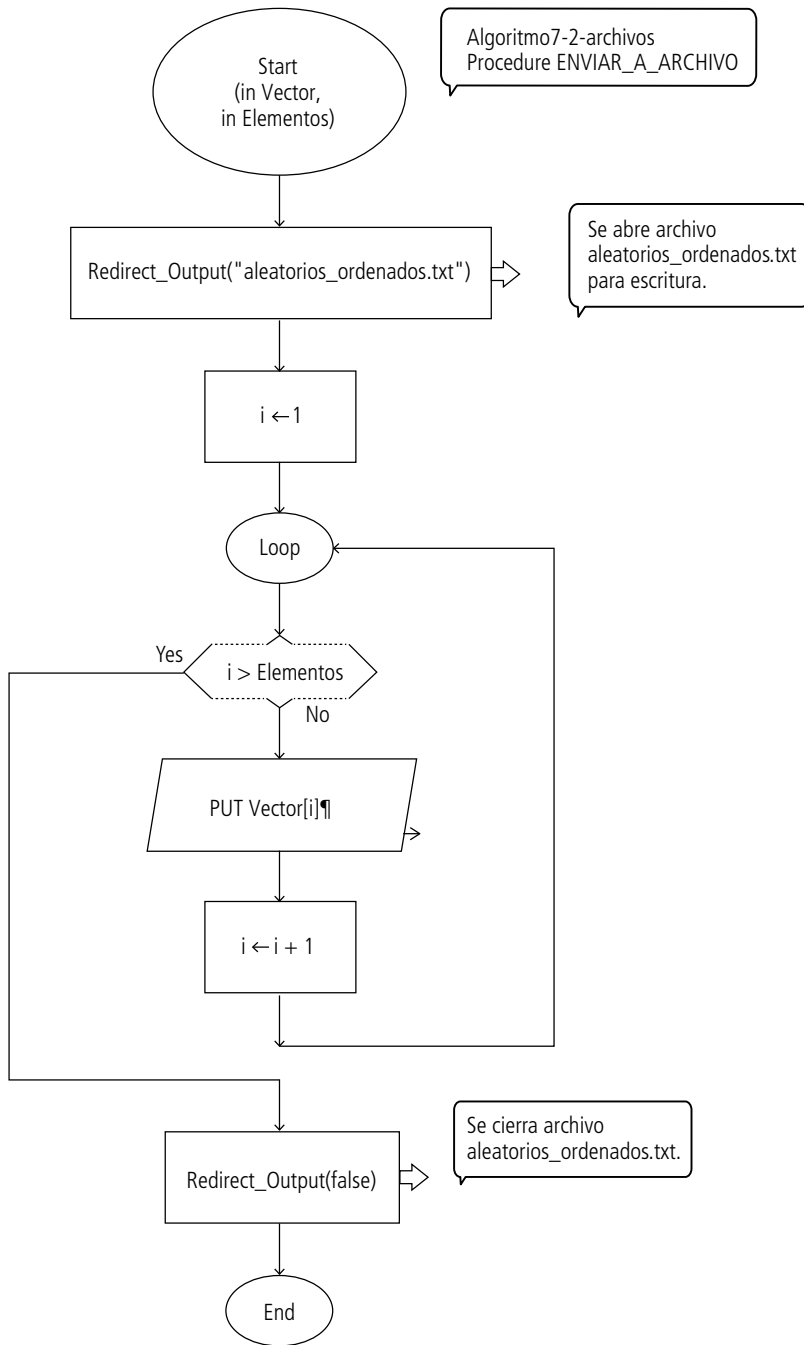


Figura 7.7 Algoritmo7-2-archivos *Procedure ORDENAR*.

**Figura 7.8** Algoritmo7-2-archivos *Procedure* ENVIAR_A_ARCHIVO.



7.4 Agregar datos a un archivo digital

Como se explicó en el **Algoritmo7-1-archivos**, cuando se usa la función `Redirect_Output` con el objetivo de agregar datos a un archivo, si éste no existe se crea uno nuevo, pero si ya existe y contiene datos los mismos se pierden sin advertir al usuario.

En el manejo de archivos es muy común querer agregar datos a un archivo sin perder los que ya contiene. La estrategia que nos queda por aplicar se puede dar en dos versiones.

1. Extraer los datos del archivo y depositarlos en un arreglo de memoria unidimensional. Después del último dato, en el arreglo de memoria, se le agregan los datos nuevos, al final se pasan todos del arreglo al archivo.
2. Agregar los datos nuevos en un arreglo de memoria unidimensional. Después del último dato del arreglo, se agregan los datos existentes en el archivo y, luego, se pasan todos los datos del arreglo al archivo.

Al final de ambas estrategias, el efecto es el mismo; es decir, se le habrán agregado datos nuevos a un archivo que ya contenía datos sin perder los anteriores. Esto es lo que puede mostrarse en el **Algoritmo7-3-archivos**, en el cual se ha utilizado la estrategia 1 (véanse figuras 7.9 a 7.12).

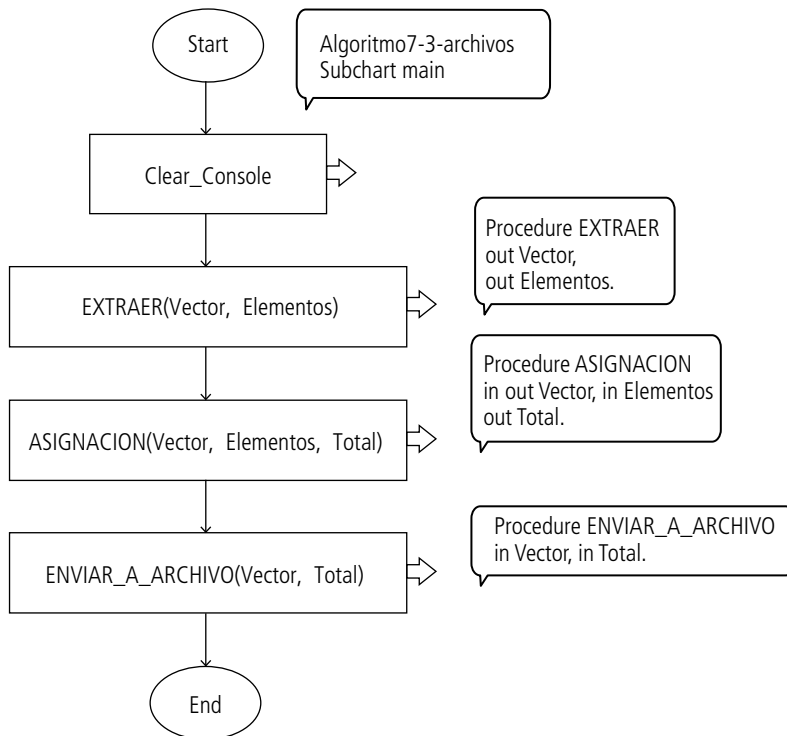


Figura 7.9 Algoritmo7-3-archivos Subchart main.

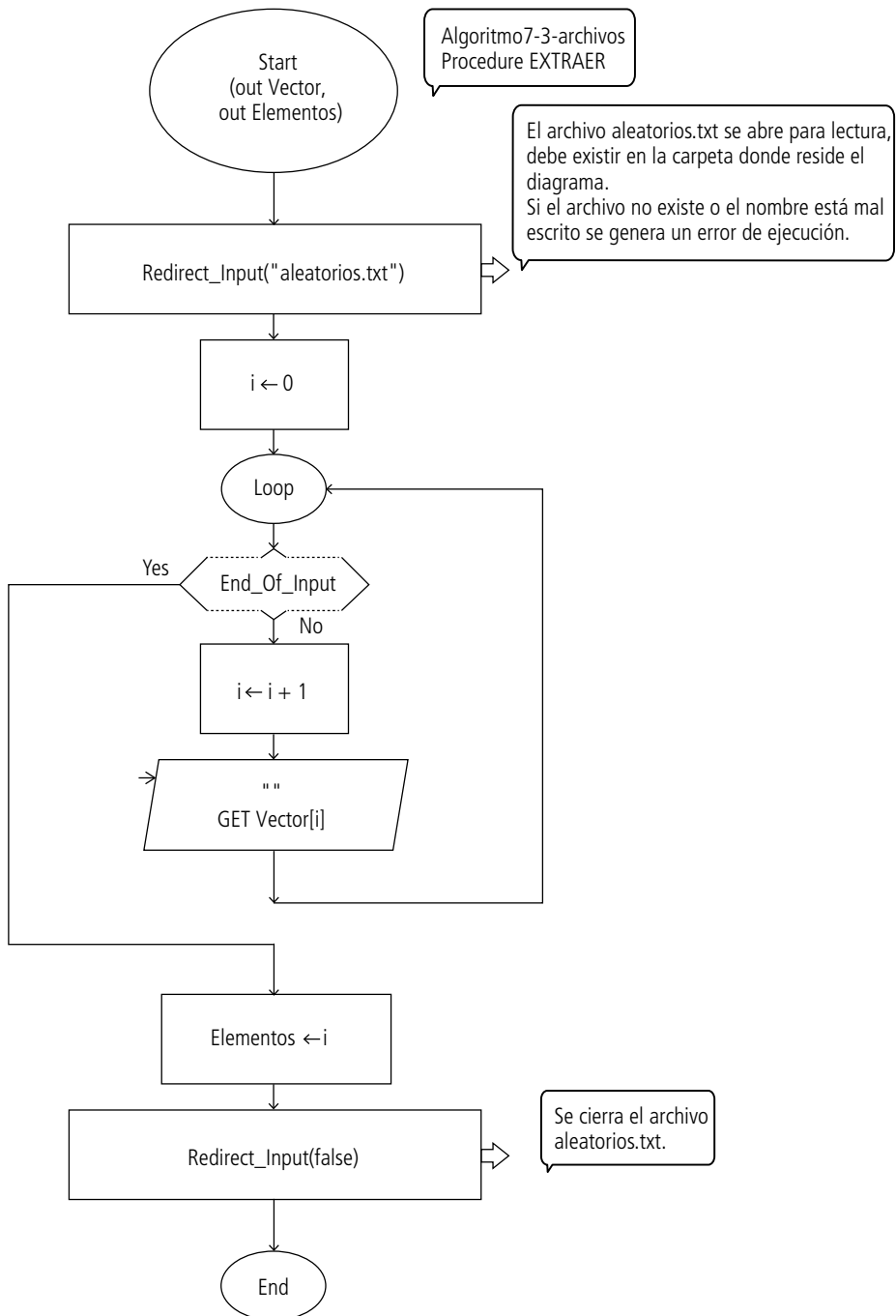


Figura 7.10 Algoritmo7-3-archivos *Procedure* EXTRAER.

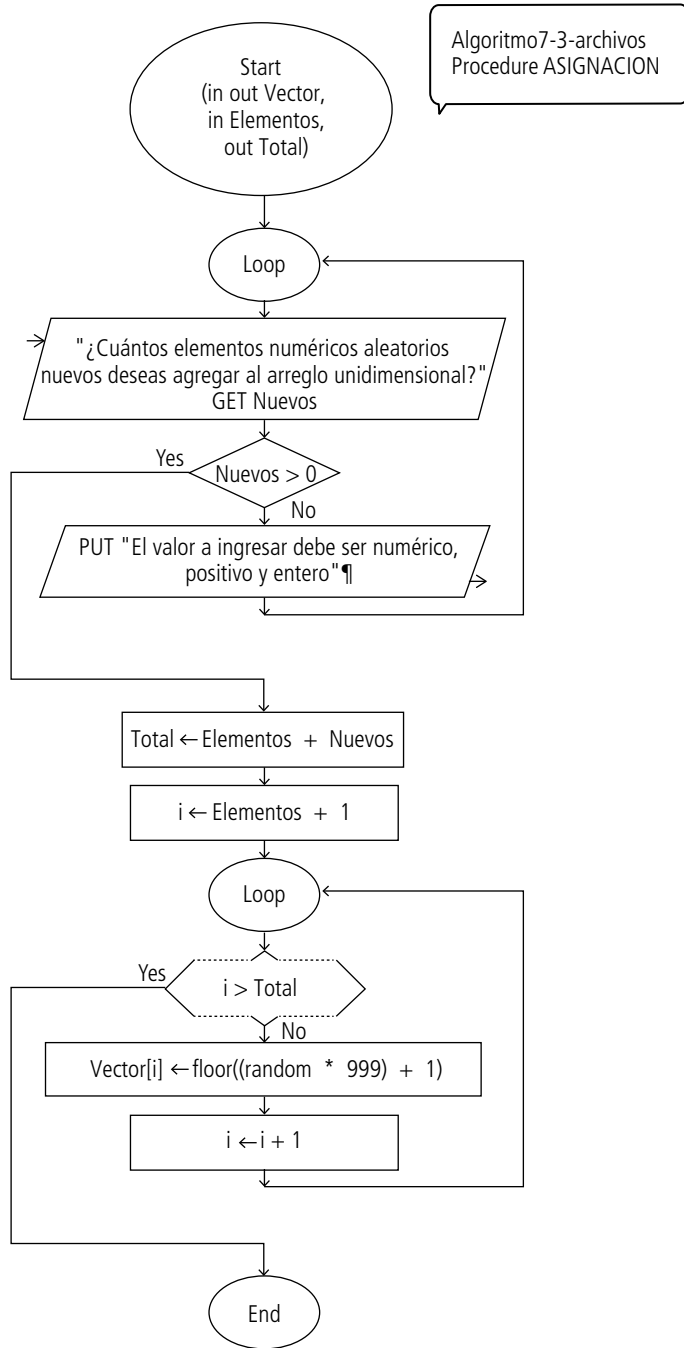


Figura 7.11 Algoritmo7-3-archivos *Procedure ASIGNACION*.

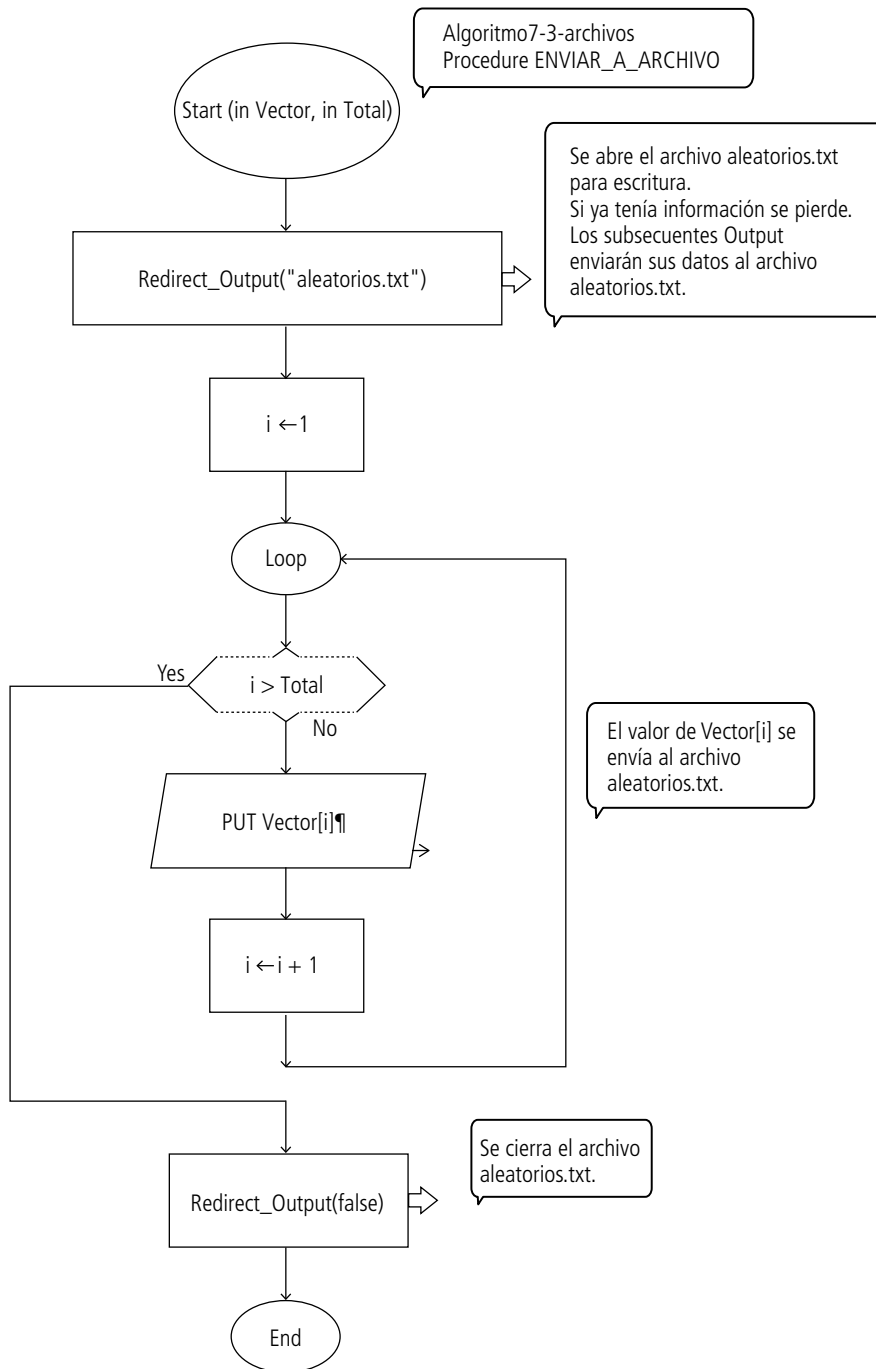


Figura 7.12 Algoritmo7-3-archivos *Procedure ENVIAR_A_ARCHIVO*.



7.5 Tratamiento de información

Desde el primer capítulo, se hizo énfasis en que los datos que estudia el tratamiento de información son aquéllos relacionados con alguna entidad o que tienen un significado dentro de algún contexto o situación en particular, y que además puedan procesarse para generar información.

Cuando se tiene un conjunto de datos relacionados y con diversas características, éstos forman registros de datos. Por diversas características nos referimos a que pueden ser numéricos, alfabéticos o alfanuméricos; sin embargo, están relacionados porque juntos tienen un significado en algún contexto. Por ejemplo, los datos que una persona da a una organización cuando se registra como parte de su matrícula estudiantil.

Para el tratamiento de información existen diversos conceptos que están claramente definidos, tales como:

1. **Dato.** Es un carácter o conjunto de caracteres que tienen un significado dentro de un contexto en particular.
2. **Registro.** Es un conjunto de datos relacionados que pueden tener diferentes características técnicas y que identifican una entidad en particular.
3. **Archivo de datos.** Es un conjunto de registros que se refieren a entidades semejantes relacionadas entre sí por un contexto en particular.

A los datos que guardan relación con otros y forman un registro se les denomina campos. De esta manera, si una persona tiene los datos personales y escolares de un solo alumno, se dice que tiene el registro de ese alumno, pero si tiene los datos personales y escolares de todos los alumnos, se dice que tiene el archivo de los alumnos.

Los ejemplos que hasta ahora hemos mostrado se han limitado a agregar o extraer un tipo de dato de los archivos; es decir, datos que tienen las mismas características técnicas. A continuación, mencionaremos cómo se agregan y extraen datos de diversos tipos de un archivo digital.

Raptor agrega los datos a un archivo, línea por línea, con el símbolo Output. Se hace de esta manera porque predetermina y selecciona la casilla de verificación **End current line**. Si a esta casilla se le quita la selección, entonces, los datos ya no se agregarían línea por línea.

Raptor extrae los datos de un archivo línea por línea con el símbolo Input. Se hace así porque el símbolo Input ya está predeterminado. Esto viene a plantearnos algunas consideraciones técnicas que debemos tener presentes al momento de extraer los datos de un archivo para almacenarlos en memoria RAM y asignarlos a variables.

Si partimos de que la inserción de datos a un archivo es línea por línea, cuando se van a almacenar diversos datos en un mismo archivo se tiene que hacer una planeación minuciosa y un diagrama escrupuloso que permita la inserción de datos válidos. De lo contrario, se corre el riesgo de que, por ejemplo, cuando se extraigan valores que son alfabéticos o alfanuméricos, como el nombre de una persona o su registro federal de causantes respectivamente, y se asignen a variables que se utilizan en operaciones aritméticas, esto ocasionará errores de ejecución.

Lo que se debe tener en cuenta es que Raptor clasifica sus variables como numéricas o cadenas. Las numéricas son aquellas que contienen números y tienen uno de los formatos mostrados en la tabla 7.3. Las de cadena son aquellas que pueden contener otros caracteres diferentes a números o que aun con números se consideran cadenas porque tienen un formato que hace que Raptor las considere de esta manera. Vea de nuevo la tabla 7.3. Una vez que Raptor ha clasificado una variable como numérica o cadena, si se vuelven a asignar valores que no son consistentes con el tipo de dato previo se ocasionará un error de ejecución.

Tabla 7.3 Datos numéricos vs. datos cadena

Dato considerado numérico	Dato considerado cadena
3e5 (equivale a 3×10^5)	3 5 (Tiene un espacio en blanco entre el 3 y el 5)
123	12.3.2 (Tiene un punto entre el 12 y el 3, y entre el 3 y el 2)
-55	(Nada, si nada se agrega, se considera una cadena vacía)
58.7	"123" (Las comillas son parte de la cadena)

Con el **Algoritmo7-4-archivos** se puede almacenar en un archivo los datos de varios estudiantes, como son: la matrícula, el nombre y las calificaciones del primer y segundo exámenes. Un archivo de texto con los datos de un solo alumno luciría de la siguiente forma:

```
1234567
DANILO DOMÍNGUEZ VERA
100
95
```

En este caso, la planeación consiste en analizar los datos para establecer los valores válidos para cada uno de los datos, esto ayudará a evitar que aparezcan errores fatales al ejecutar el diagrama.

La matrícula y ambas calificaciones son datos numéricos y el nombre es un dato cadena. Además, las restricciones para la matrícula es que debe ser un dato con siete números. El rango debe estar entre 1,000,000 y 2,000,000. En cuanto al nombre, éste debe contener sólo letras mayúsculas o espacios en blanco, ni será permitido ningún otro carácter. Las calificaciones del primer y segundo exámenes deben estar en el rango de cero a cien.

En este caso, los datos se almacenarán en el archivo llamado **escolar.txt**. Se debe tomar en cuenta que el algoritmo no está programado para impedir que se agreguen datos repetidos al archivo.

En una institución educativa no debe haber dos estudiantes que tengan la misma matrícula. Esto hace que el dato matrícula sirva para identificar a una única persona y no confundirla con otra.

A continuación, el caso que presentaremos es el almacenamiento de los registros de varios estudiantes. Cada registro tiene cuatro campos: la matrícula, el nombre y las calificaciones del primer y segundo exámenes.

Se debe estar consciente de que cada registro almacena las calificaciones obtenidas en una materia específica, pero no sabemos cuál porque no se proporciona dato al respecto. Si hubiera un alumno que llevara más de una materia no tendríamos manera de distinguir un registro de otro porque los datos son insuficientes.

Lo mejor que podemos hacer es considerar que los registros en este archivo son de un maestro que imparte una sola materia y desea calcular el promedio de sus alumnos. De esta manera, los registros pertenecen a un solo alumno. Es conveniente que la matrícula no se repita en este archivo, si esto sucediera se quedaría en la incertidumbre, pues no se sabría cuáles son las calificaciones que realmente ha obtenido el alumno.

Observe que los *Procedure* **AGREGAR_MATRÍCULA**, **AGREGAR_CALIFICACIÓN1** y **AGREGAR_CALIFICACIÓN2** sólo permiten la captura de datos numéricos que están en cierto rango; es decir, tienen una rutina que permite validar los valores.

El *Procedure* **AGREGAR_NOMBRE** sólo permite la captura de datos que tengan letras mayúsculas o espacios en blanco. Para hacer la validación se analiza la cadena por carácter para ver si cada uno está en el rango que establece el código ASCII. Esto es, entre 65 y 90 para las letras mayúsculas, y 32 para el espacio en blanco (véanse figuras 7.13 a 7.18).

El tratamiento de información incluye el almacenamiento, la extracción y el procesamiento de los datos para generar nueva información, por lo que continuaremos con este tema.

Se usarán los datos almacenados por el **Algoritmo7-4-archivos** en el archivo **escolar.txt**, el cual contiene los siguientes datos: la matrícula, el nombre del alumno y las calificaciones del primer y segundo exámenes.

Para poder procesar los datos almacenados en un archivo, recurriremos a la siguiente estrategia.

Primero, se extraerán los datos del archivo y se colocarán en arreglos de memoria. Como en este caso son cuatro datos diferentes, entonces se utilizarán cuatro arreglos unidimensionales, uno por cada dato. Segundo, en un quinto arreglo unidimensional se calculará el promedio de las dos calificaciones. Finalmente, los datos de los cinco arreglos unidimensionales se almacenarán en archivos digitales. En este caso, el **Algoritmo7-5-archivos** ofrece dos resultados porque se quiere mostrar la diferencia entre un archivo sin formato y un archivo con formato. Utilice un editor de textos para ver los datos contenidos en los archivos **promedio_sin_formato.txt** y **promedio_con_formato.txt** (véanse figuras 7.19 a 7.23).

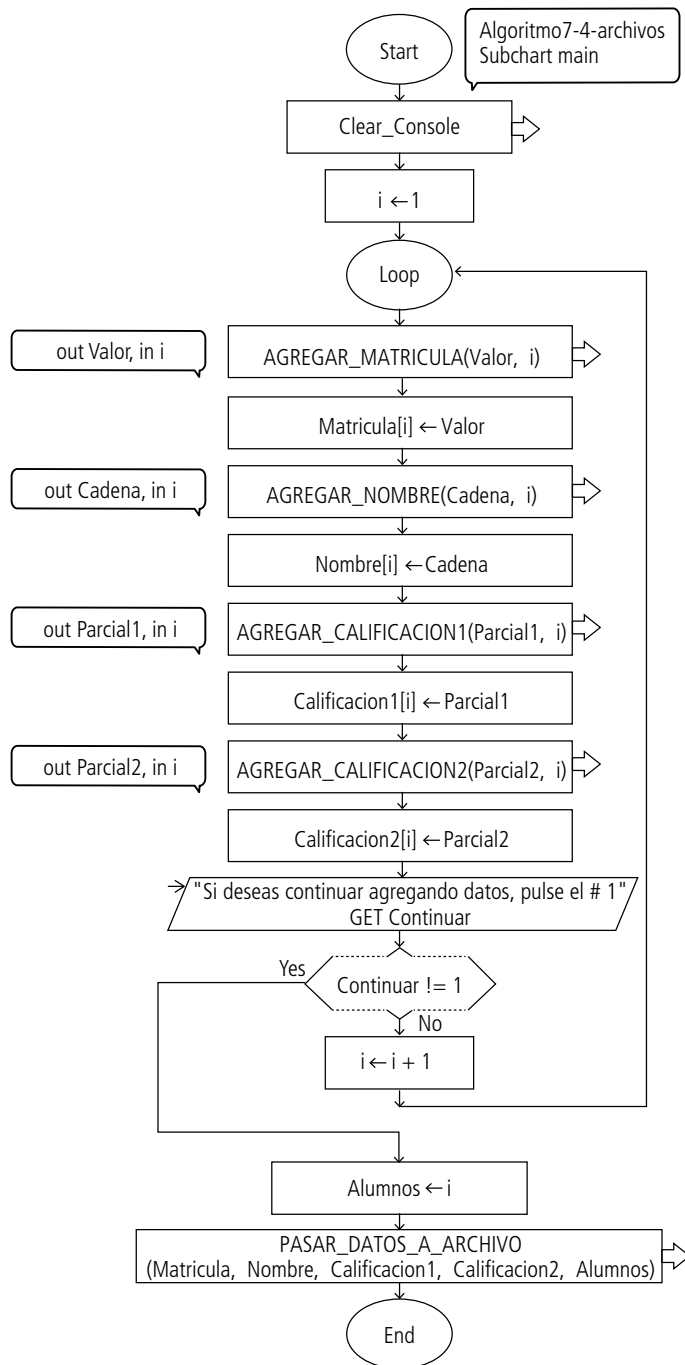


Figura 7.13 Algoritmo7-4-archivos *Subchart main*.

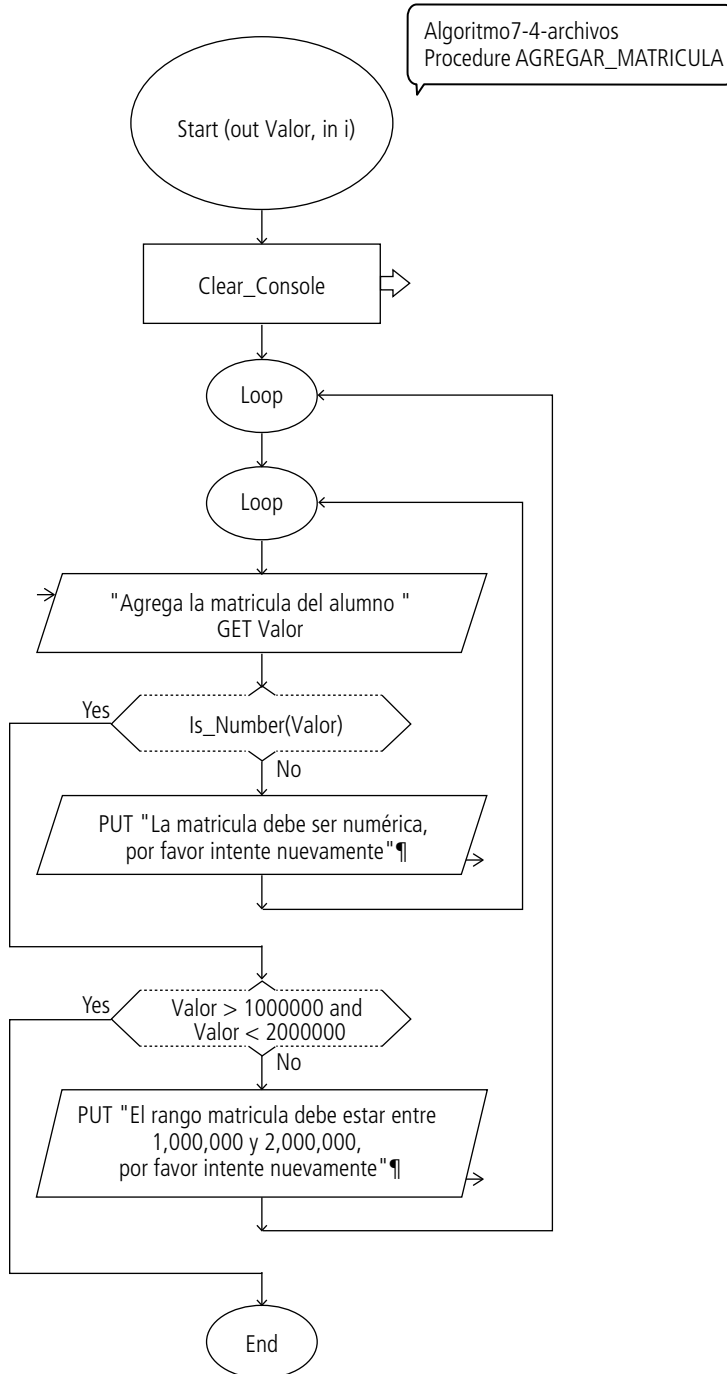


Figura 7.14 Algoritmo7-4-archivos *Procedure* AGREGAR_MATRICULA.

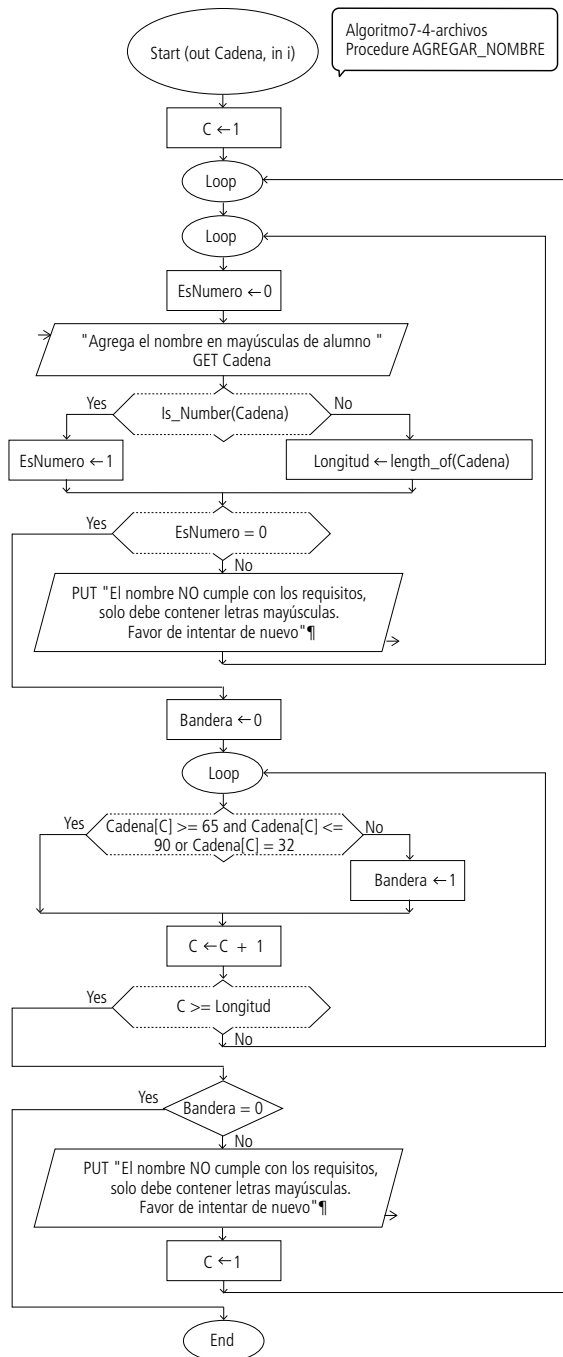


Figura 7.15 Algoritmo7-4-archivos *Procedure* AGREGAR_NOMBRE.

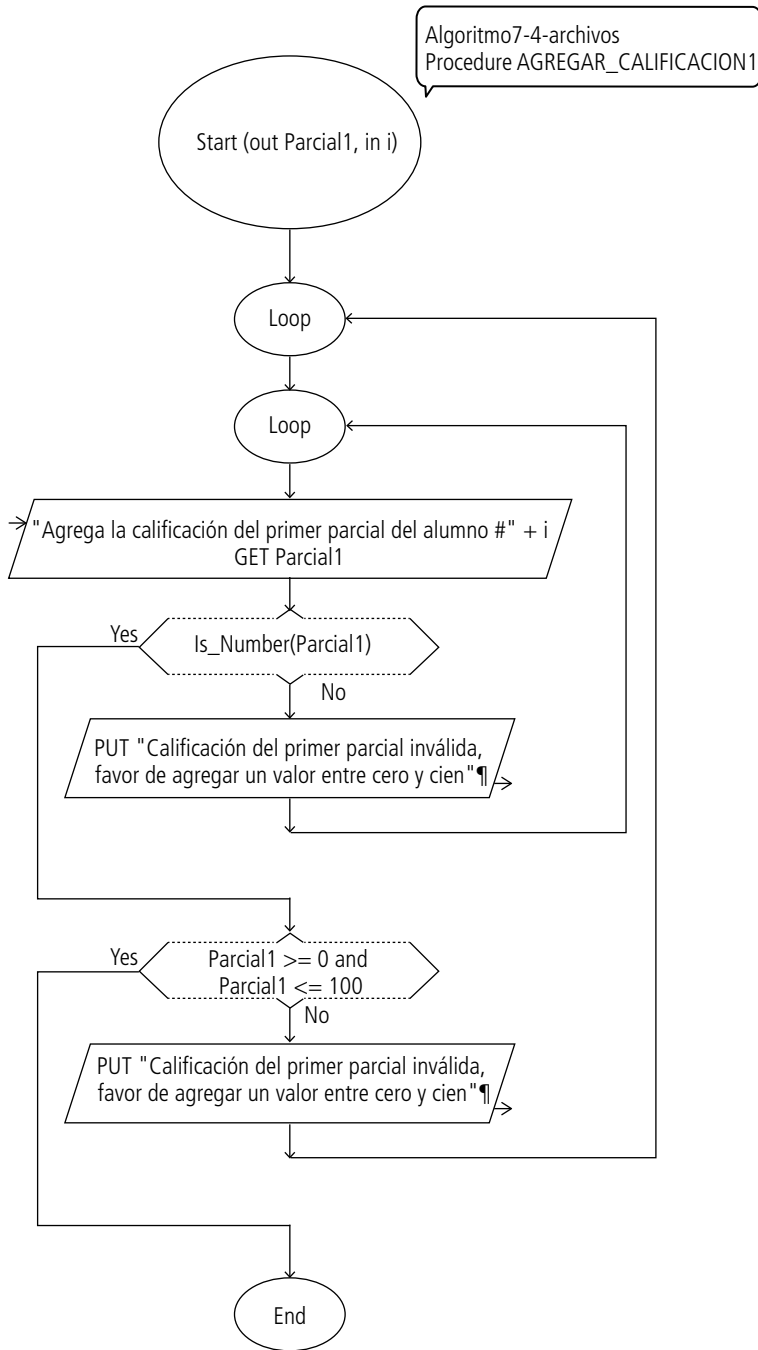


Figura 7.16 Algoritmo7-4-archivos Procedure AGREGAR_CALIFICACION1.

Algoritmo7-4-archivos
Procedure AGREGAR_CALIFICACION2

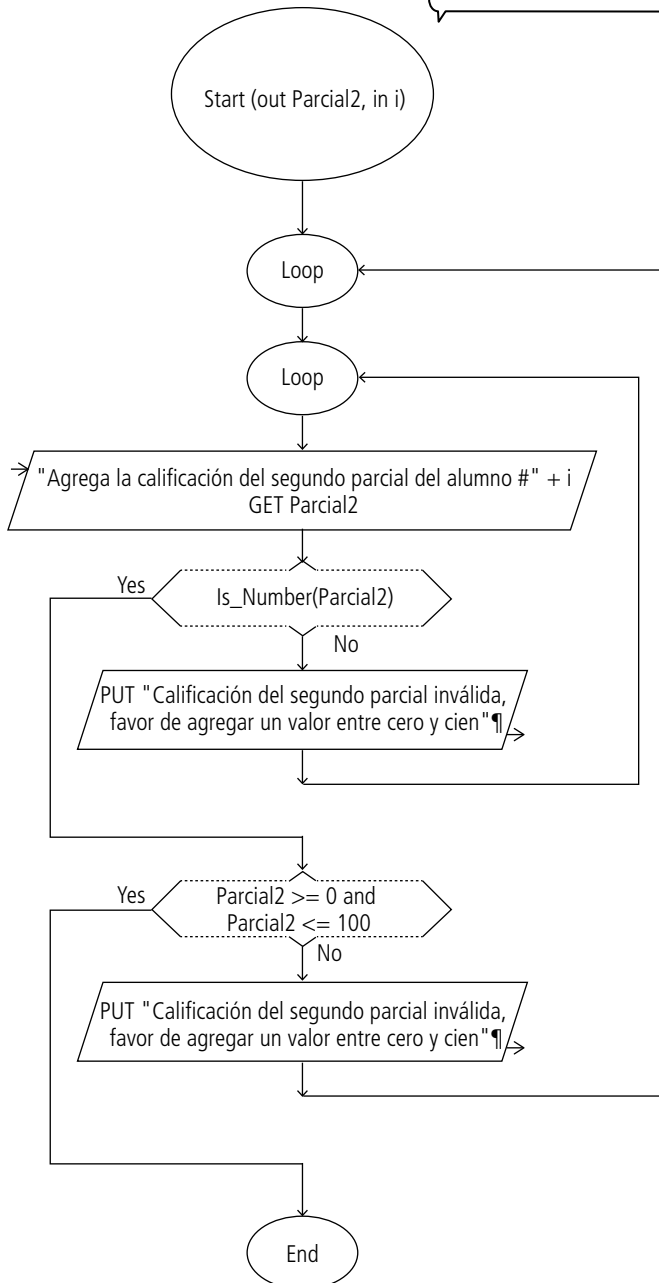


Figura 7.17 Algoritmo7-4-archivos *Procedure* AGREGAR_CALIFICACION2.

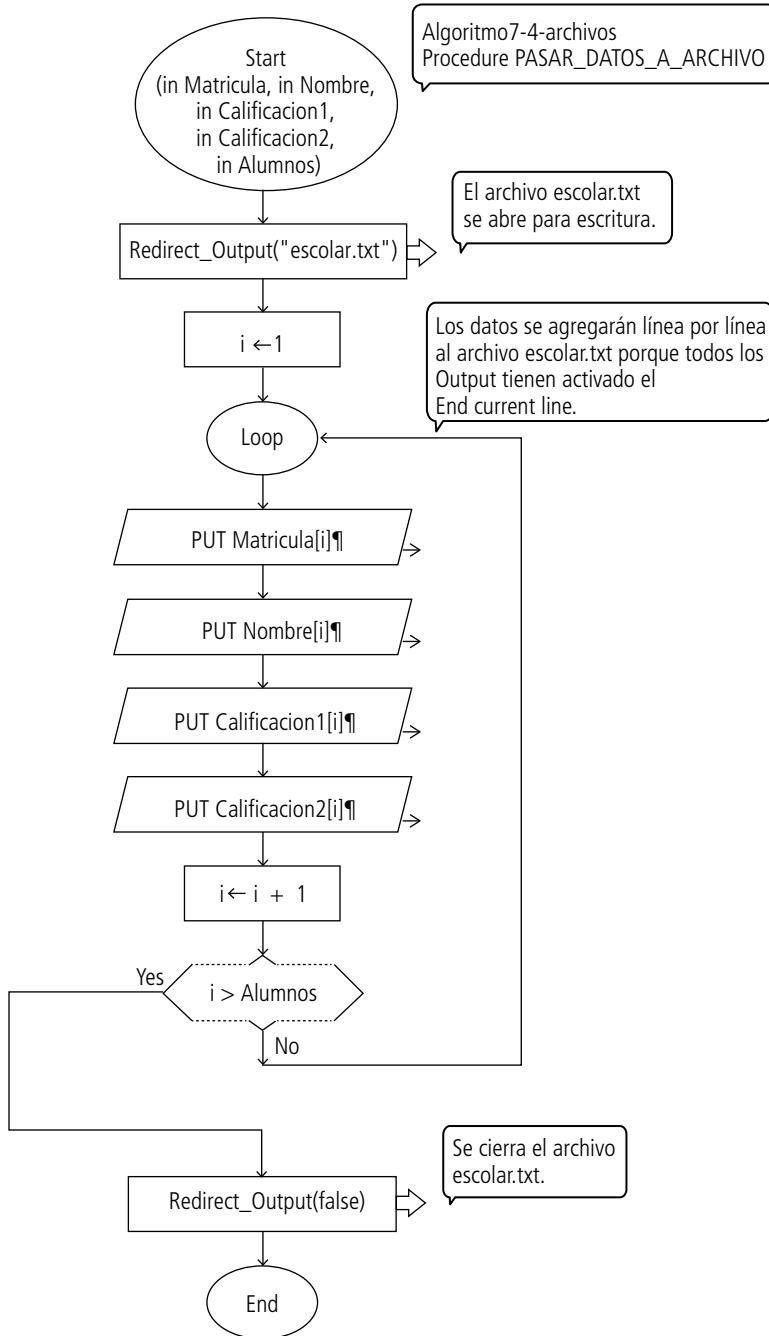


Figura 7.18 Algoritmo7-4-archivos *Procedure PASAR_DATOS_A_ARCHIVO*.

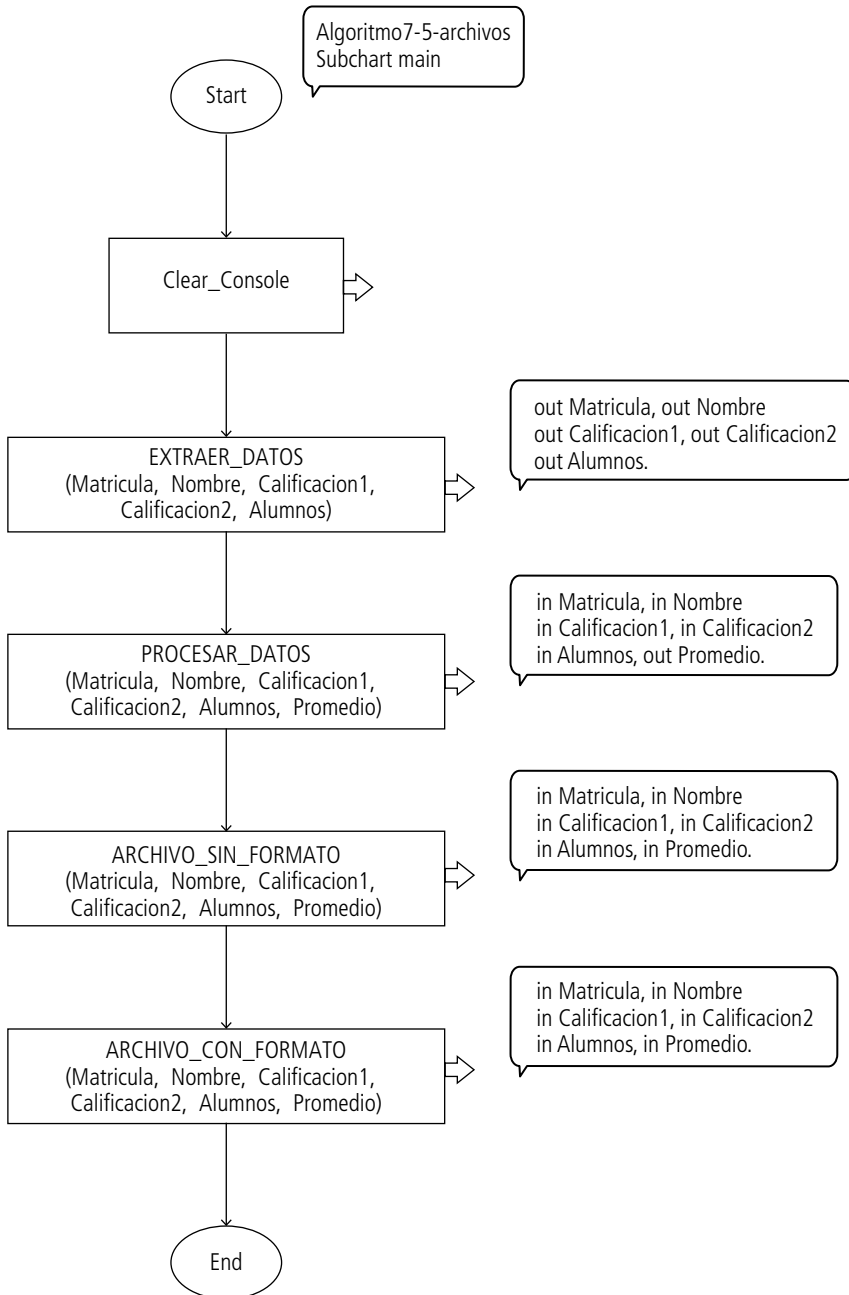


Figura 7.19 Algoritmo7-5-archivos *Subchart main*.

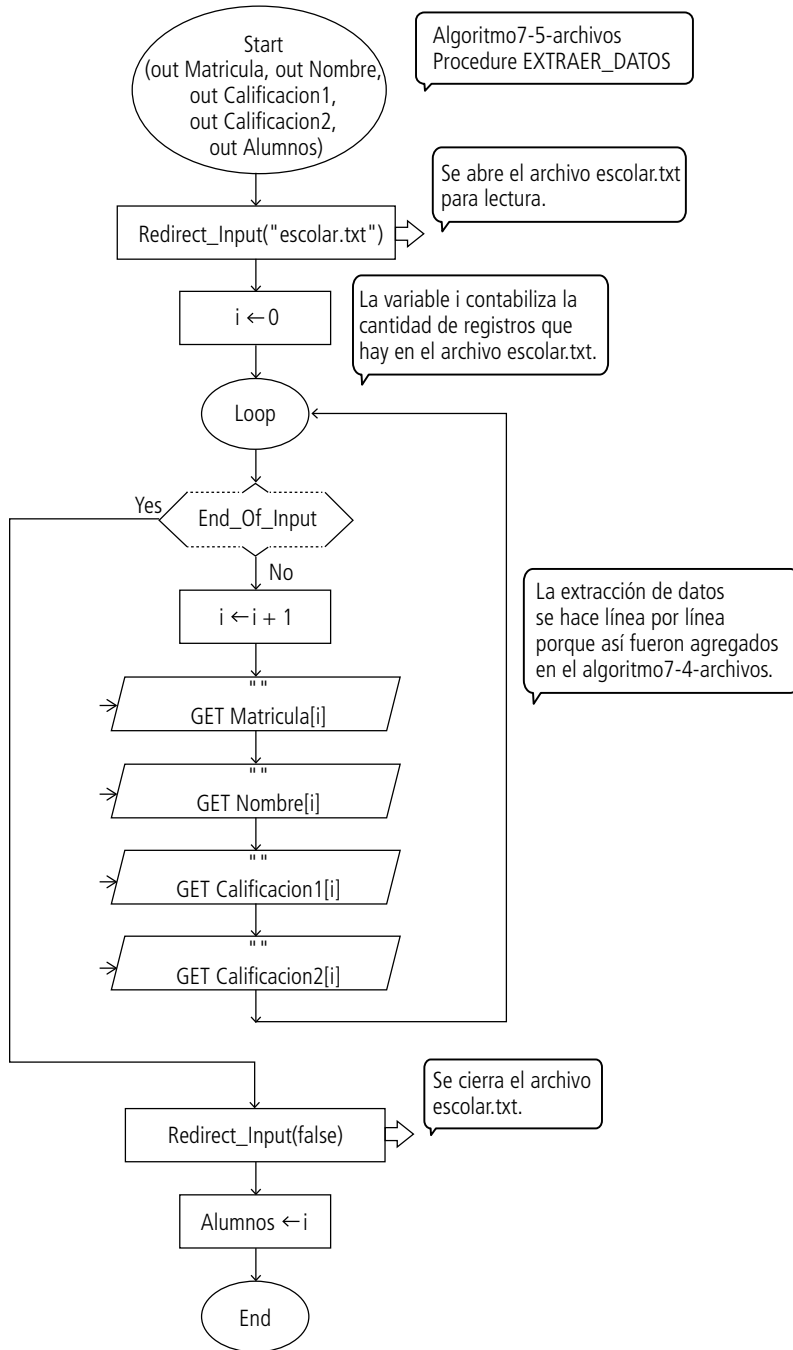


Figura 7.20 Algoritmo7-5-archivos *Procedure* EXTRAER_DATOS.

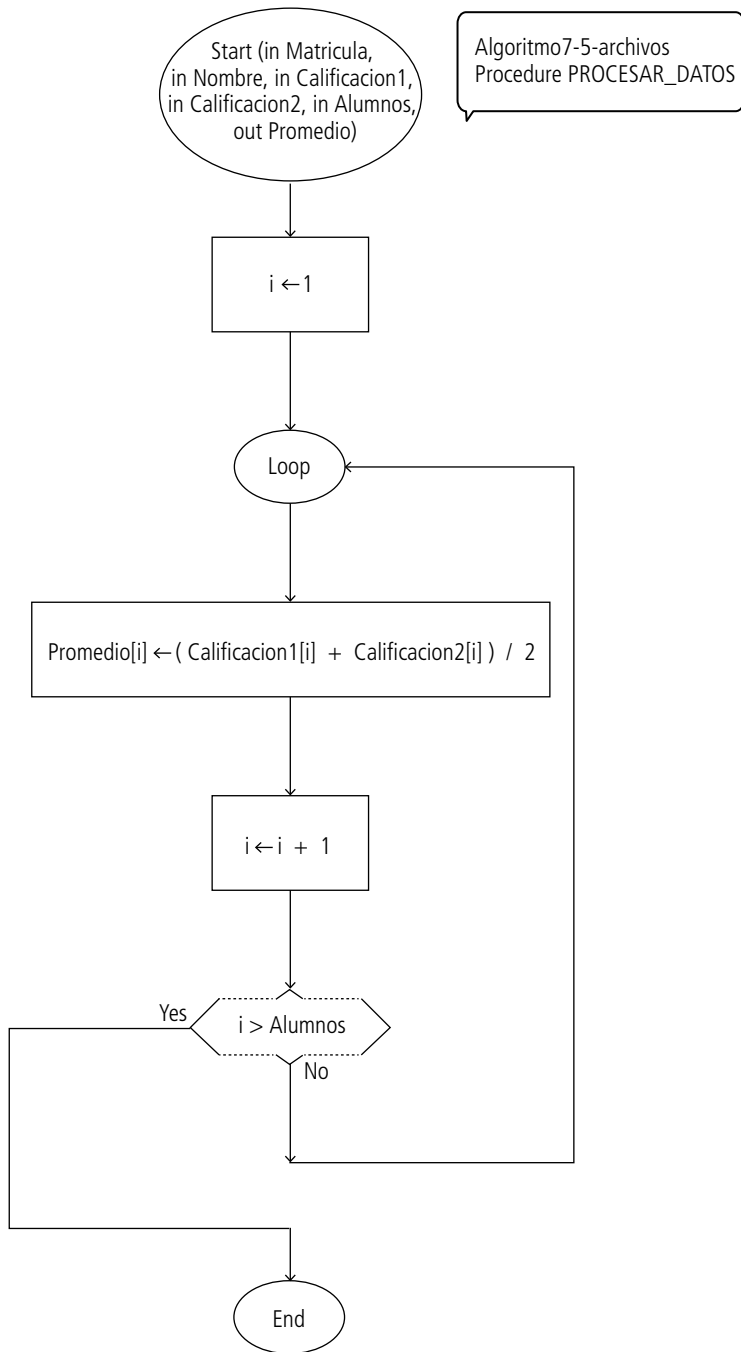


Figura 7.21 Algoritmo7-5-archivos *Procedure PROCESAR_DATOS*.

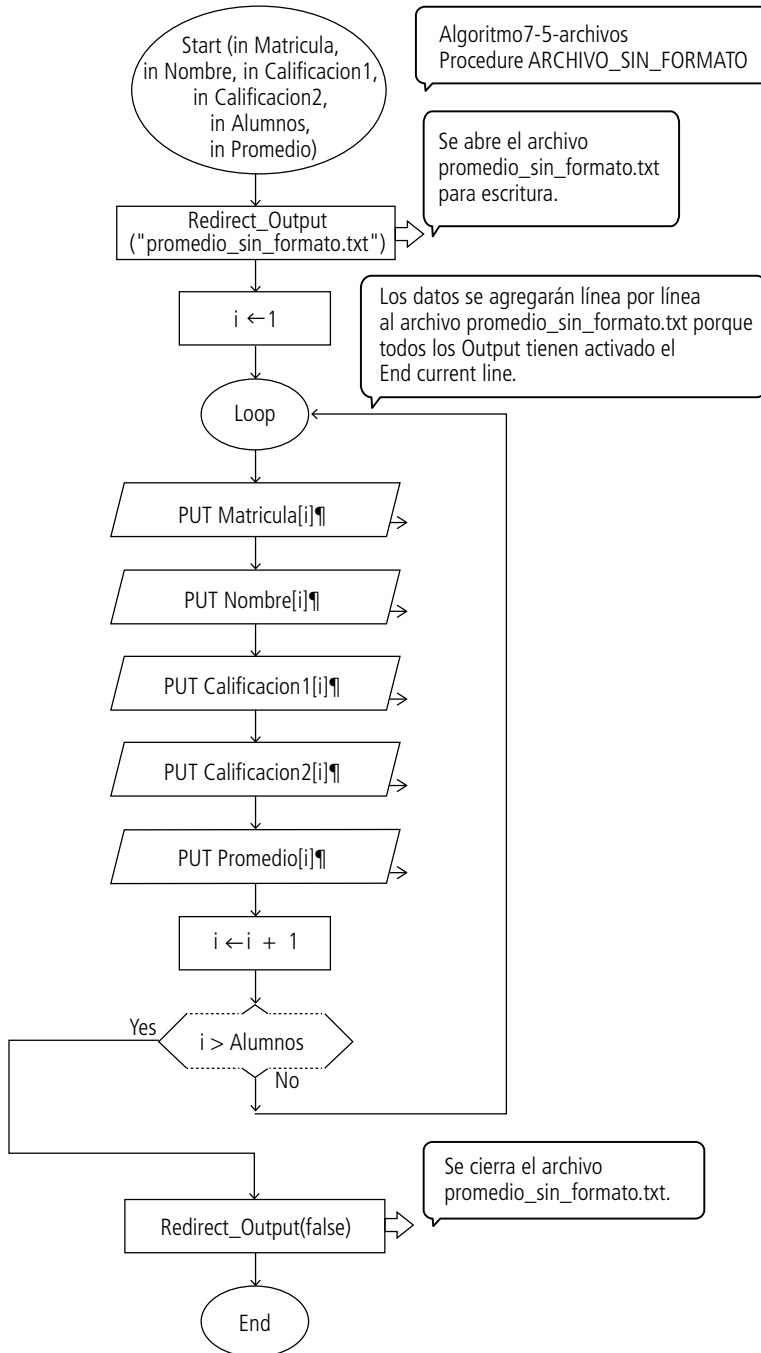


Figura 7.22 Algoritmo7-5-archivos *Procedure* ARCHIVO_SIN_FORMATO.

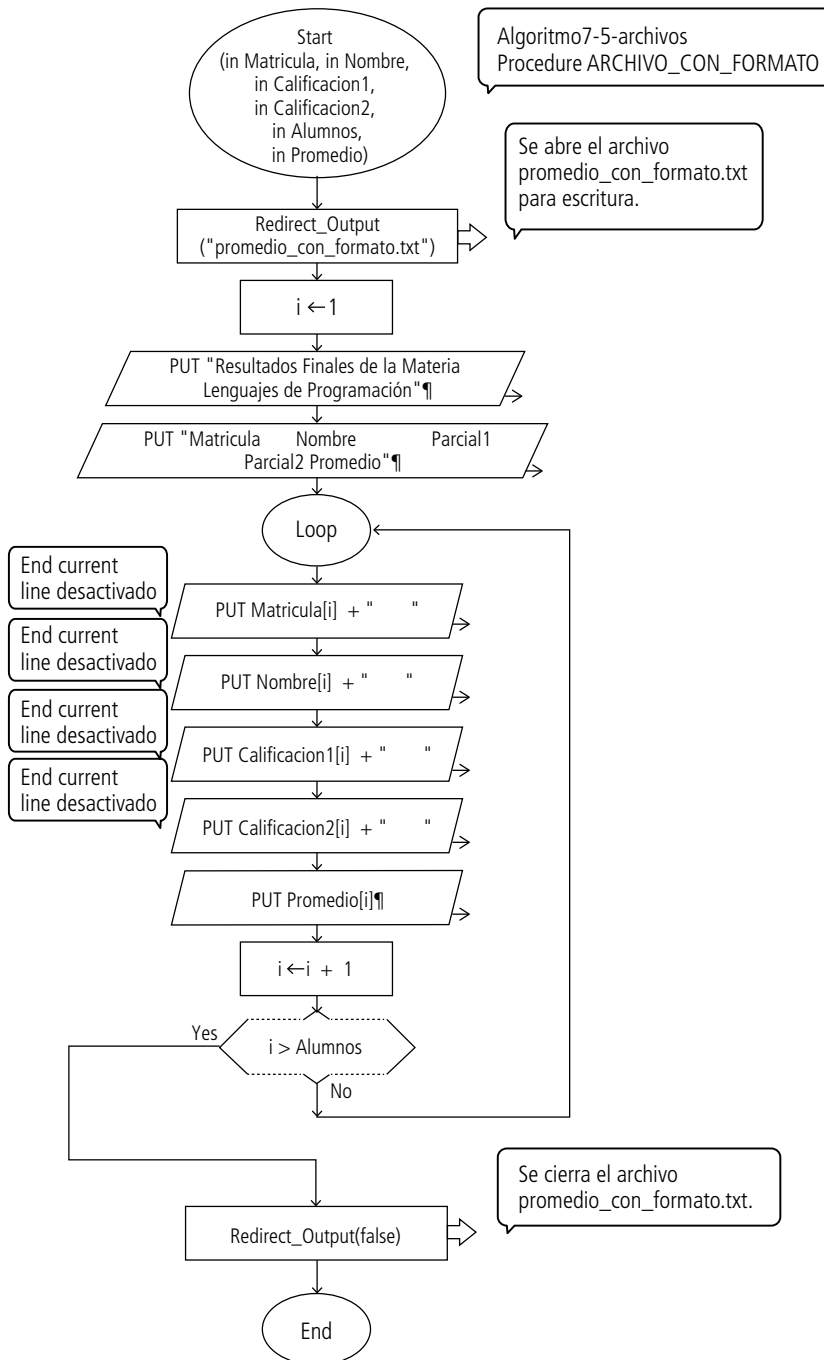


Figura 7.23 Algoritmo 7-5-archivos *Procedure ARCHIVO_CON_FORMATO*.



7.6 Tratamiento de información con corte de control

Cuando un archivo contiene varios registros que pertenecen o identifican a una misma entidad y desean procesarse para obtener un reporte, se debe recurrir a la técnica de corte de control. Esta técnica tiene como condición que los datos tengan un campo en común que los identifique como pertenecientes a la misma entidad y que estén agrupados de manera contigua. Si estas condiciones no se cumplen, entonces, no puede usarse esta técnica.

Por lo general, se recurre al ordenamiento por medio del campo en común de todos los registros del archivo para forzar el agrupamiento contiguo de los registros. Una vez que se ha logrado lo anterior, se usa la técnica de corte de control.

Considere una institución educativa con 10,000 estudiantes, en la cual se imparten 300 materias diferentes y cada estudiante puede llevar de una hasta siete materias en un semestre. Entonces, el archivo contiene los datos de matrícula, folio de la materia y la calificación obtenida en esa materia.

A cada materia se le asigna un folio numérico para poder identificarla (véase tabla 7.4). El archivo maestro de materias está incompleto porque sólo se muestran seis registros, cuando en realidad deberían ser 300 porque se imparten 300 materias.

Tabla 7.4 Ejemplo incompleto de Archivo maestro de materias

Folio de la materia	Descripción
150	Matemáticas I
151	Física I
152	Programación I
153	Matemáticas II
154	Física II
155	Programación II

El archivo de calificaciones finales que contiene la matrícula del estudiante, el folio de la materia y la calificación final, considerando que un alumno lleva tres materias en un semestre en particular y que los datos se han agregado línea por línea en un archivo, podría lucir así:

```

1234567
150
80
1234567
151
90
1234567
152
100

```

Si registra lo anterior en una tabla, se obtiene:

Tabla 7.5 Archivo general de calificaciones finales

Matrícula	Folio de materia	Calificación final
1234567	150	80
1234567	151	90
1234567	152	100

A partir de un archivo de datos semejante al anterior, se desea calcular su promedio para cada uno de los 10,000 estudiantes. En este caso, antes de utilizar la técnica de corte de control, los datos deben ordenarse por medio del campo llave, que en este caso es Matrícula.

Un programa de corte de control incluye las siguientes acciones computacionales.

1. Abrir el archivo del que se extraerán los datos.
2. Abrir el archivo en el que se almacenarán los resultados.
3. Extraer el primer registro del archivo y almacenar el valor del campo llave en una variable temporal y dar inicialización a otras variables que así lo requieran.
4. Entrar a un ciclo en el que extraerán registro por registro del archivo hasta que se terminen.
5. Cada vez que se extrae un registro se debe preguntar si el valor del campo llave actual es igual a la variable temporal del paso 3.
6. Si el punto 5 es verdad, se realizan las operaciones matemáticas pertinentes y se extrae otro registro.
7. Si el punto 5 es falso, se realiza el corte de control y se actualiza el valor de la variable temporal que hace referencia al punto 3 y se reinician las variables que así lo requieran.
8. Una vez que se termina el ciclo del punto 4, se deben finalizar los cortes y cerrar los archivos de datos.

Con el **Algoritmo7-6-corte**, se calculan los promedios de los estudiantes de acuerdo con la explicación que previamente se ha dado. En este algoritmo no se incluye el módulo que ordena los registros porque se parte de que ya están agrupados. Si esto no fuera así, el diagrama no ofrecerá resultados satisfactorios. Se sugiere que sea el propio lector el que incluya el módulo de ordenamiento, según el ejemplo mostrado al principio de este capítulo (véanse figuras 7.24 a 7.27).

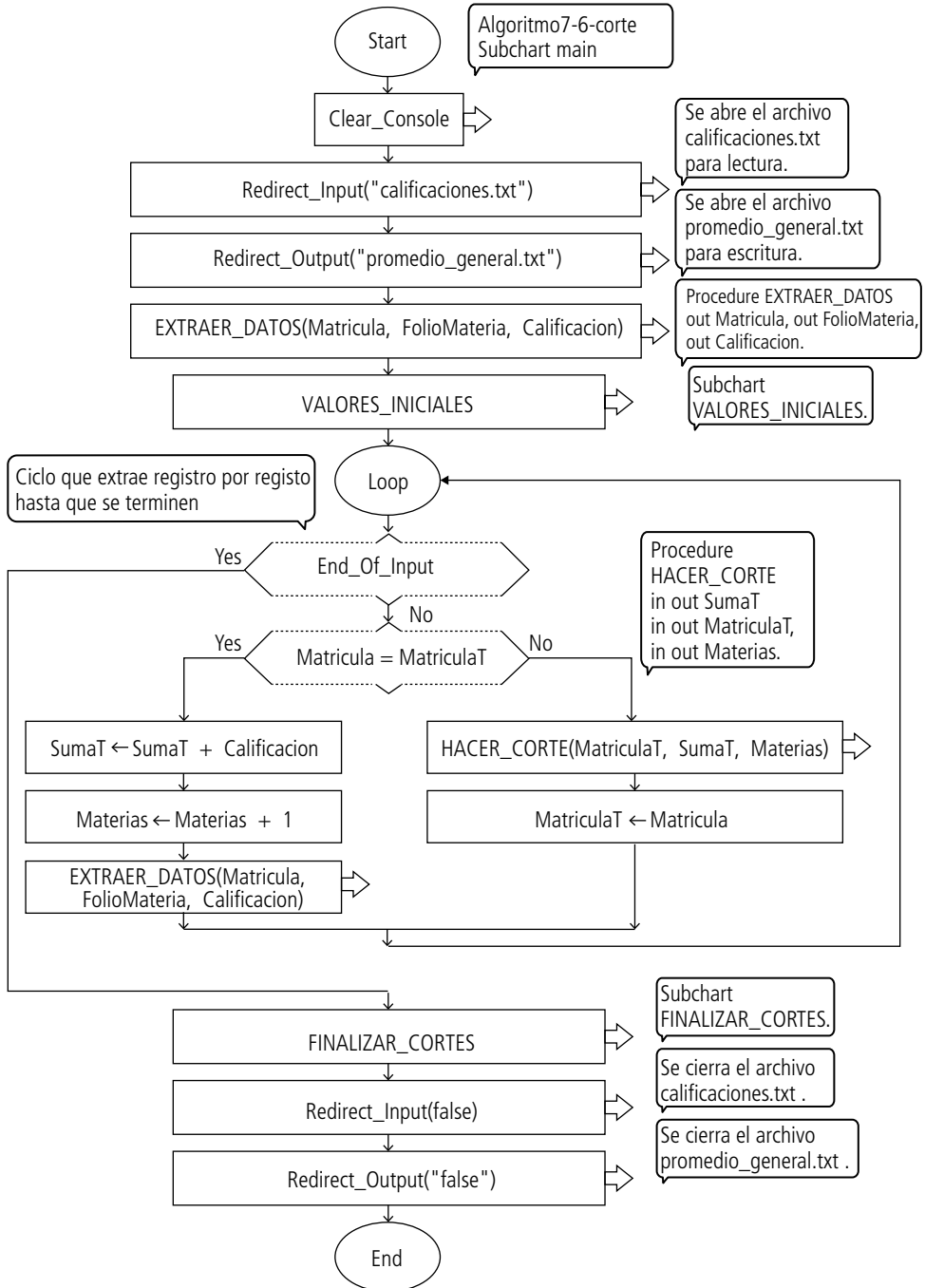


Figura 7.24 Algoritmo7-6-corte Subchart main.

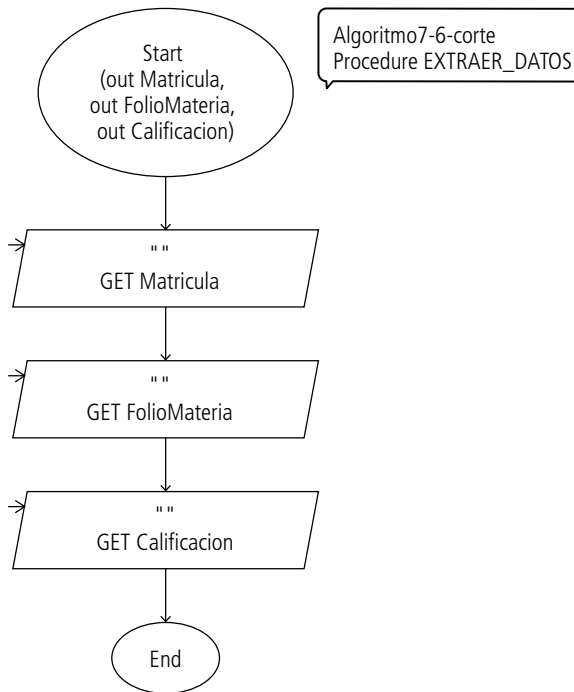


Figura 7.25 Algoritmo7-6-corte Procedure EXTRAER_DATOS.

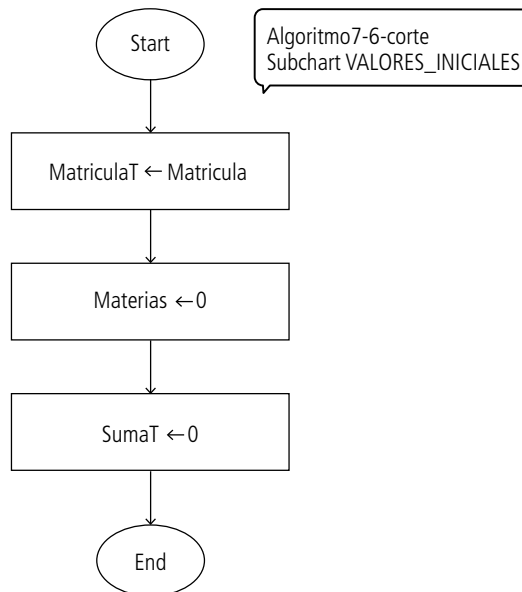


Figura 7.26 Algoritmo7-6-corte Procedure VALORES_INICIALES.

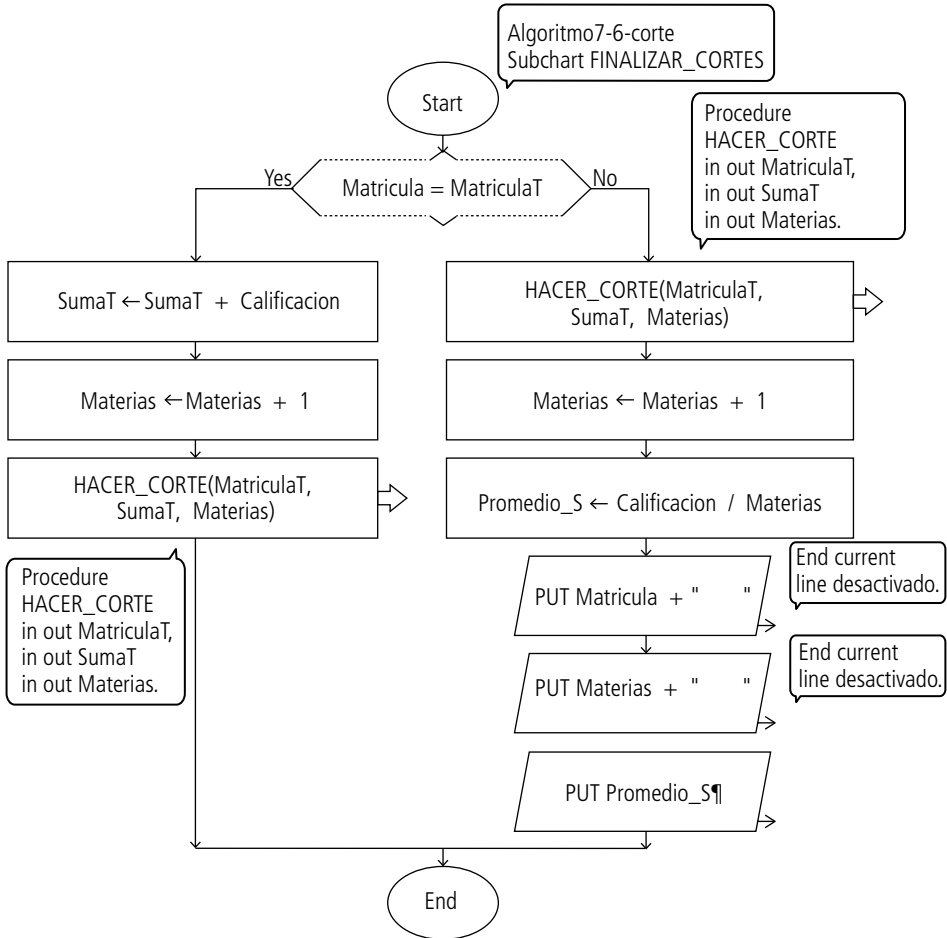


Figura 7.27 Algoritmo 7-6-corte Procedure FINALIZAR_CORTES.



7.7 Ejercicios de autoevaluación

Las respuestas correctas a estos ejercicios están contenidas en la tabla 7.6.

- 7.1** El disco duro de la computadora es un dispositivo de almacenamiento _____.
- 7.2** De manera preestablecida, el símbolo Output envía sus mensajes a _____.

- 7.3** ¿Cuál es la función que permite enviar datos hacia un archivo de datos, en lugar de enviarlos a *MasterConsole*?
- 7.4** `Redirect_Output` puede crear y abrir un archivo para _____.
- 7.5** Cuando un archivo de datos ya no se va a utilizar debe _____.
- 7.6** ¿Cuál es el parámetro de `Redirect_Output` que permite cerrar un archivo?
- 7.7** De manera preestablecida, el símbolo `Input` permite que el usuario utilice el _____ para agregar datos a la memoria RAM.
- 7.8** ¿Cuál es la función que permite extraer datos de un archivo para colocarlos en la memoria RAM?
- 7.9** `Redirect_Input` puede abrir un archivo de datos para _____.
- 7.10** ¿Cuál es el parámetro de `Redirect_Input` que permite cerrar un archivo?
- 7.11** ¿Cuál es la función que se puede utilizar en el Loop cuando se desea extraer todos los datos de un archivo?
- 7.12** ¿Cómo se denomina el carácter o conjunto de caracteres que tienen un significado dentro de un contexto en particular?
- 7.13** ¿Cómo se denomina el conjunto de datos relacionados que pueden tener diferentes características técnicas y que identifican a una entidad en particular?
- 7.14** ¿Cómo se denomina el conjunto de registros que se refieren a entidades semejantes relacionadas entre sí por un contexto en particular?
- 7.15** ¿Cómo se denominan los datos que guardan una relación con otros y forman un registro?

Tabla 7.6 Respuestas a los ejercicios de autoevaluación

7.1 Externo	7.2 <i>MasterConsole</i>	7.3 <i>Redirect_Output</i>
7.4 Escritura	7.5 Cerrarse	7.6 <i>False</i>
7.7 Teclado	7.8 <i>Redirect_Input</i>	7.9 Lectura
7.10 <i>False</i>	7.11 <i>End_Of_Input</i>	7.12 Datos
7.13 Registro	7.14 Archivo	7.15 Campo



7.8 Problemas propuestos: tratamiento básico de información

En los problemas propuestos en esta sección, los datos de los archivos se muestran en una tabla para facilitar su visualización. Sin embargo, considere que los datos en los archivos de entrada están almacenados línea por línea.

7.1 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- a) Archivo de entrada: nombre del empleado, horas trabajadas y tarifa por hora.
- b) Archivo de salida: nombre del empleado, pago bruto, IMSS y pago neto.

Notas:

- a) $\text{Pago bruto} = (\text{Horas trabajadas}) * (\text{Tarifa por hora})$
- b) $\text{IMSS} = 4.8\% \text{ del pago bruto}$
- c) $\text{Pago neto} = (\text{Pago bruto}) - (\text{IMSS})$

Ejemplo:

Tabla 7.7 Archivo de Entrada del problema 7.1

Nombre del empleado	Horas trabajadas	Tarifa por hora
Juan Molina	30	\$4.50
Dora Vélez	40	\$3.90
Sara Castro	45	\$4.70
Saúl Hidalgo	39	\$3.50
Clara Mora	46	\$4.20

Tabla 7.8 Archivo de salida del problema 7.1

Nombre del empleado	Pago bruto	IMSS	Pago neto
Juan Molina	\$135	\$6.48	\$128.52
Dora Vélez	\$156	\$7.48	\$148.52
Sara Castro	\$211.5	\$10.15	\$201.35
Saúl Hidalgo	\$136.5	\$6.55	\$129.95
Clara Mora	\$193.20	\$9.27	\$183.93

7.2 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- Archivo de entrada: número de factura, nombre del cliente e importe de venta.
- Archivo de salida: número de factura, nombre del cliente, importe de venta, porcentaje de descuento e importe de descuento neto.

Notas:

- Si el importe de venta excede de \$100, se dará 3% de descuento; si el importe de venta es menor o igual que \$100, se dará 2% de descuento.
- Importe de descuento = (Importe de venta) * (Porcentaje de descuento)
- Importe de descuento neto = (Importe de venta) - (Importe de descuento)

Tabla 7.9 Archivo de entrada del problema 7.2

Número de factura	Nombre	Venta
1596	Rafael Sosa	\$350
1852	Perla Vela	\$90
1956	Rosa Méndez	\$260
1784	Raúl Zavala	\$55
2106	Laura Vázquez	\$195

Tabla 7.10 Archivo de salida del problema 7.2

Número de factura	Nombre	Venta	% descuento	Importe de descuento neto
1596	Rafael Sosa	\$350	3	\$339.50
1852	Perla Vela	\$90	2	\$88.20
1956	Rosa Méndez	\$260	3	\$252.20
1784	Raúl Zavala	\$55	2	\$53.90
2106	Laura Vázquez	\$195	3	\$189.15

7.3 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- a) Archivo de entrada: nombre del estudiante, número de créditos.
- b) Archivo de salida: nombre del estudiante, número de créditos y colegiatura.

Nota: Si un estudiante toma 12 créditos o menos, la colegiatura es de \$15 por cada crédito; si toma más de 12 créditos, la colegiatura es de \$150.

Tabla 7.11 Archivo de entrada del problema 7.3

Nombre del estudiante	Créditos
Gloria Mejía	15
Carlos Pedraja	10
Sonia López	12
Santiago Zapata	11
Carmen Salas	13

Tabla 7.12 Archivo de Salida del problema 7.3

Nombre del estudiante	Créditos	Colegiatura
Gloria Mejía	15	\$150
Carlos Pedraja	10	\$150
Sonia López	12	\$180
Santiago Zapata	11	\$165
Carmen Salas	13	\$150

- 7.4** Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- a) Archivo de entrada.
Nombre del empleado, horas trabajadas y tarifa por hora.
- b) Archivo de salida.
Nombre del empleado y el sueldo.

Notas:

- a) $\text{Sueldo} = (\text{Horas normales}) * (\text{Tarifa por hora}) + (\text{Horas extra}) * (3.5)$
- b) Las horas extra son aquellas que pasan de 40.

Ejemplo:

Tabla 7.13 Archivo de entrada del problema 7.4

Nombre del empleado	Horas trabajadas	Tarifa por hora
Juan Molina	30	\$4.50
Dora Vélez	40	\$3.90
Sara Castro	45	\$4.70
Saúl Hidalgo	39	\$3.50
Clara Mora	46	\$4.20

Tabla 7.14 Archivo de salida del problema 7.4

Nombre del empleado	Sueldo
Juan Molina	\$135
Dora Vélez	\$156
Sara Castro	\$205.50
Saúl Hidalgo	\$136.50
Clara Mora	\$189

7.5 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- Archivo de entrada: nombre del empleado, código (1.Salario, 2.Sueldo, 3.Comisión), importe 1 e importe 2.
- Archivo de salida: nombre del empleado y el importe de lo ganado.

Notas:

- Si es salario, multiplique el importe 1 por el importe 2 para obtener el importe de lo ganado.
- Si es sueldo, el importe de lo ganado es igual al importe 1.
- Si es comisión, multiplique el importe 1 por el importe 2 y agregue 8% adicional para obtener el importe de lo ganado.

Tabla 7.15 Archivo de entrada del problema 7.5

Nombre del empleado	Código	Importe 1	Importe 2
Raúl Garza	2	\$530	\$150
Aída Sánchez	1	\$50	\$15
Hilda Ibarra	3	\$45	\$10
Yazmín Iglesias	2	\$630	\$20
Édgar Pérez	3	\$30	\$25

Tabla 7.16 Archivo de salida del problema 7.5

Nombre del empleado	Importe ganado
Raúl Garza	\$530
Aída Sánchez	\$750
Hilda Ibarra	\$486
Yazmín Iglesias	\$630
Édgar Pérez	\$810



7.9 Problemas propuestos: corte de control

7.1 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- a) Archivo de entrada.
Número de vendedor, nombre del vendedor e importe de venta.
- b) Archivo de salida.
Número de vendedor, nombre del vendedor y sueldo.
Si el importe de venta es menor que \$1,000, la comisión es de 5%; si es mayor o igual que \$1,000 y menor que \$3,000, la comisión es de 7%; y si es mayor o igual que \$3,000, la comisión es de 10%.

Nota: El archivo de entrada está completamente validado y clasificado en forma ascendente por grupos de número de vendedor.

Ejemplo:

Tabla 7.17 Archivo de entrada del problema 7.6

Número de vendedor	Nombre del vendedor	Importe de venta
119	Arturo Soto	\$3,200
119	Arturo Soto	\$1,100
119	Arturo Soto	\$ 900
673	Aída Mata	\$5,100
673	Aída Mata	\$2,500
673	Aída Mata	\$1,200
673	Aída Mata	\$1,000
710	Saúl Molina	\$1,500
710	Saúl Molina	\$2,100

Tabla 7.18 Archivo de salida del problema 7.6

Número de vendedor	Nombre del vendedor	Sueldo
119	Arturo Soto	\$442
673	Aída Mata	\$815
710	Saúl Molina	\$252

7.2 Realice un programa que genere un archivo de salida a partir de un archivo de entrada, el cual contiene:

- Archivo de entrada: número de factura, descripción del producto, cantidad y precio unitario.
- Archivo de salida: número de factura y pago.

Nota: El archivo de entrada está completamente validado y clasificado en forma ascendente por grupos de número de factura.

Ejemplo:

Tabla 7.19 Archivo de entrada del problema 7.7

Número de factura	Descripción	Cantidad	Precio unitario
523	Puntillas	2	\$4.60
523	Libreta Scribe	3	\$5.90
523	Aplique Fox Pro	1	\$140
558	Mapa Monterrey	1	\$50
558	Plumas Bic	4	\$2.50
591	Pc World	1	\$25
591	Peter Norton	1	\$150
591	Libreta Scribe	2	\$5.90
591	Access Facil	1	\$120

Tabla 7.20 Archivo de salida del problema 7.7

Número de factura	Pago
523	\$166.90
558	\$ 60.00
591	\$306.80

Bibliografía

- Deitel, H. M., & Deitel, P. J. (2004). *Cómo Programar en C/C++ y Java*, México: Pearson Prentice Hall, 1,152 pp.
- Domínguez Vera, E. D. (2014). *Programación estructurada: Raptor y Lenguaje C*, Ciudad de México: Alfaomega Grupo Editor, 296 pp.
- Gottfried, B. (2005). *Programación en C*, 2ª. edición revisada, Madrid: McGraw-Hill/Interamericana, 590 pp.
- Hegedüs, P. (2013). “Revealing the Effect of Coding Practices on Software Maintainability”. IEEE Conference Publication, pp. 1-8.
- Humphrey, W. S. (2009). *PSP: A Self-Improvement for Software Engineers*. Westford, Massachusetts: Pearson Education, Inc, 368 pp.
- Li, X. (2006). “Effectively Teaching Coding Standards in Programming”, IEEE Conference Publications, pp. 9-14.
- Pressman, R. S. (2010). *Ingeniería del software: Un enfoque práctico*, México: McGraw-Hill, 777 pp.
- Rose, J. P. (2011). *Designing and Checking Coding Standards for Ada*. ACM SIGAda Ada Letter, pp. 13-14.
- Schildt, H. (1991). *Programación en turbo C*, 2ª. edición, Madrid: McGraw-Hill, 449 pp.
- Spencer, D. (1977). *Solución de problemas con Fortran*, Englewood Cliffs N.J.: Prentice/Hall International, 319 pp.
- Takai, Y., Kobayashi, T., & Agusa, K. (2001). *Software Metrics Based on Coding Standards Violations*. IEEE Conference Publications, pp. 273-278.

Índice analítico

acción, 10, 12

algoritmo

 computacional, 6, 11, 12

 ciclo, 7

 impresión de resultados, 7

 lectura de datos, 7

 proceso de datos, 7

 selección de alternativas, 7

 creación de, 7

 diseño por módulos, 140

archivo(s), 273

 de texto, 268, 273, 284

arreglos de memoria, 202

 bidimensionales, 227

 unidimensionales, 203

asignación

 constante, 37

 variable, 37

bandera, 115

binarios, 17

bus del sistema, 5

 de datos, 5

 de direcciones, 5

campos, 283

carácter, 8

celda, 203

ciclo, 102

cliente, 11

codificación, 8, 12

columna

 nombre de la, 203

computadora(s), 2, 7

 programación de, 8, 202

condición matemática, 102

constante, 159, 203

dato(s), 10, 202, 283

 alfabético, 10

 alfanuméricos, 10

 archivo de, 283

 conjunto de, 202

 inserción de, 283

 lectura de, 14, 16

 numérico, 10, 202

 proceso de, 14, 16

diagrama

 de flujo, 12, 15

 pseudocódigo, 12

dispositivos

 externos, 268

 internos, 268

ecuación cuadrática, 89

errores

 computacionales, 78

 de lógica común, 78

- estructura
 - repetitiva, 102
 - selectiva, 66
 - anidamiento, 88
- fórmulas
 - explícitas, 128
 - recursivas, 128
- hardware, 4
- impresión de resultados, 17
- informática, 2
- lenguaje de programación, 7
- matriz, 228
 - aumentada, 227
 - de coeficientes, 227
- medios de almacenamiento, 5
- método de pseudocódigo, 15
- módulo(s), 141, 158
 - de entrada de datos, 146
 - de proceso, 146
 - de salida de resultados, 146
- monolíticos, 140
- números, 8
- opción diferida, 102
- operador(es), 16, 38, 43
 - aritméticos, 43
 - lógicos, 66, 67, 68
 - relacionales, 66, 68
- operandos, 16, 43
 - dependientes, 38
- parámetro(s), 140, 158, 159
 - de entrada, 158
 - de salida, 158
 - globales, 159
- pensamiento manual, 202
- primario, 268
- proceso de software, 6, 10
 - codificación, 12
 - comunicación, 11
 - compilación, 13
 - diseño, 12
 - planeación, 11
 - prueba, 13
 - revisión de la codificación, 13
 - revisión del diseño, 12
- producto generado
 - de ASIGNACION, 163
 - de INICIO, 175
 - de OPERACIONES, 165
 - de Resultado, 163
- programación, 109
 - acumulados
 - negativos, 109
 - positivos, 109
- programador, 202
- RAM
 - memoria, 268
- Raptor, 15, 30, 102, 284
 - beneficios, 30
- registro, 283
- selección de alternativas, 66
- símbolo(s), 15
 - Assignment, 33, 53

- Input, 33
- Loop, 102, 104
- Output, 34
- sistema operativo, 3, 4
- software, 4
 - de aplicación, 4
 - de base, 4
 - de Raptor, 30
 - de sistemas, 4
 - ingeniería de, 6
- soporte, 5
- subexpresión, 18
- subíndice(s), 203, 227
 - dinámico, 203, 228
 - estático, 203, 228
- sucesiones de números, 128
 - impares, 129
- técnica de corte de control, 297
- teorema de desigualdad triangular, 188
- testigo, 115
- tratamiento
 - de información, 2, 283, 284
- unidad
 - central de proceso, 4
 - aritmético-lógica, 4
 - de control, 4
 - de entrada, 5
 - de memoria, 4
 - Ram, 4
 - Rom, 5
 - de salida, 5
 - de almacenamiento secundario, 5
 - unarios, 17
 - usuario, 11
 - validación
 - de campos, 119
 - programas, 119
 - valor, 176
 - variable(s), 159, 203, 228
 - acumulada, 211
 - booleanas, 151
 - globales, 159
 - locales, 159
 - sumatoria, 211
 - vector(es), 203, 211, 275

