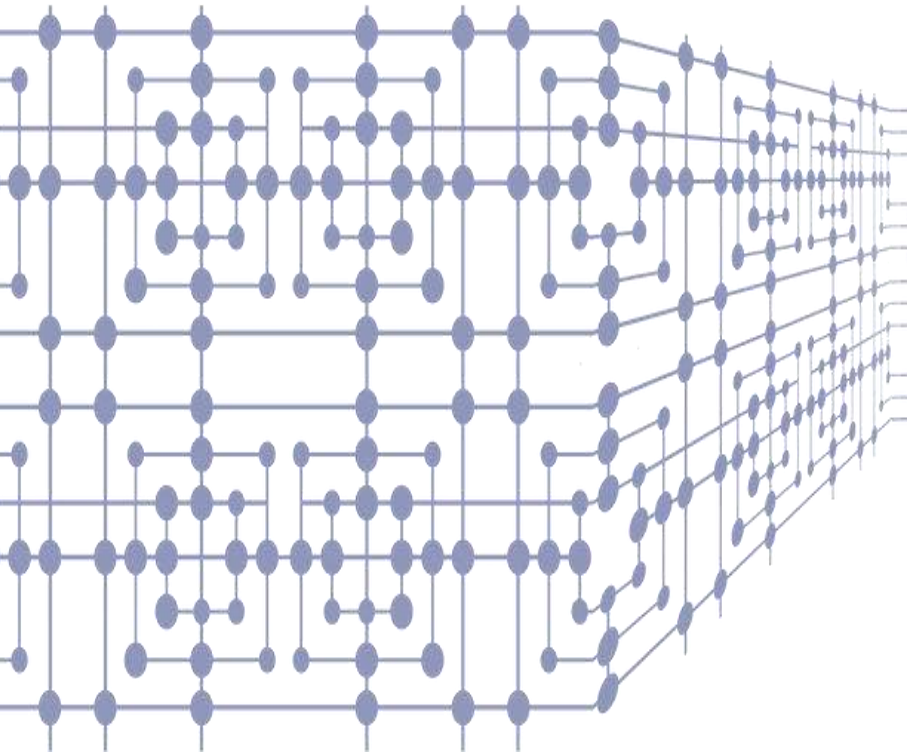
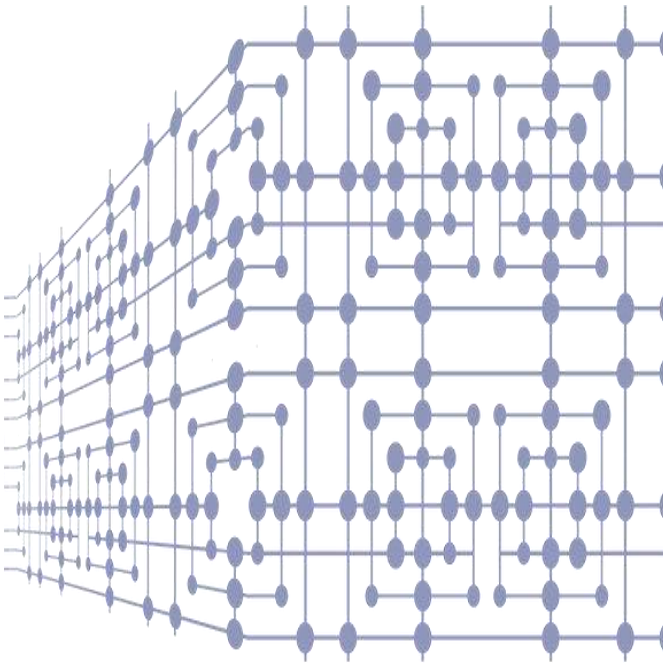


Algoritmos y Resolución de Problemas



Licenciatura en Ciencias de la Computación
Licenciatura en Sistemas de Informacion
Tecnicatura en Programacion Web
Departamento de Informática
FCEFN
UNSJ

2020

La programación es una disciplina de las Ciencias de Computación, cuyas aplicaciones generalmente requieren de la resolución de problemas. La mayor dificultad de los alumnos está en la interpretación de enunciados del problema y no en su expresión en un lenguaje de programación. Por ello se comienza con el análisis de herramientas que ayudan a la interpretación de los enunciados y metodologías que favorecen la construcción de los algoritmos para la resolución de problemas.

Para la construcción de algoritmos se utilizará un lenguaje sencillo, familiar al alumno, esto es cercano al lenguaje coloquial, llamado seudocódigo, que permitirá que su atención se centre en las estrategias de resolución de las problemáticas planteadas y no en una mera sintaxis.

El uso de seudocódigo les permitirá expresar de manera precisa y sin ambigüedades algoritmos que luego podrán ser traducidos a cualquier lenguaje de programación.

El documento está estructurado en tres ejes:

- Estrategias de resolución de Problemas para la construcción de Algoritmos.
- Análisis de eficiencia de Algoritmos
- Verificación de Algoritmos

Si bien existe bibliografía que aborda los temas aquí tratados, el equipo de la cátedra Algoritmos y Resolución de Problemas, sacando provecho de varios años de experiencia ha elaborado este material para acompañar al alumno que se inicia en la tarea de programar.

Redacción a cargo de:

Dr. Mario Díaz
Prof. Evangelina Sanz
Mg. Rosa Pósito
Lic. Daniela Villafañe
Lic. Andrea Ferreyra
Prog. Univ. Hugo Orellano
Lic. Magdalena Arrón

Compaginación:

Dr. Mario Díaz

Índice

Programa de Exámen 2020

Unidad 1: Algoritmos y Resolución de Problemas

Etapas en la resolución de problemas. Estrategias de resolución de problemas: Noción de Espacio de estado. Introducción a los conceptos de precondiciones y poscondiciones. Ejemplos.

Concepto de algoritmo. Propiedades de Algoritmos. Algoritmos y Seudocódigo.

Conceptos de constantes y variables. Tipos de datos simples. Expresiones: Aritméticas, Relacionales y Lógicas. Acciones Simples. Estructuras de control: Secuencia, Iteración y Selección. Anidamiento de estructuras de control. Contadores y Acumuladores. Cálculo de Máximos y Mínimos

Unidad 2*: Subprogramas

Subprogramas: Concepto. Estructura de un Algoritmo utilizando subprogramas: Definición e Invocación de subprogramas. Tipo de argumentos y resultados de subprogramas. Ejemplos. Seguimiento de Algoritmos

Unidad 3*: Tipo de datos estructurados: arreglos y registros

Arreglos: definición. Almacenamiento. Inicialización y Carga de arreglos. Algoritmos de búsqueda secuencial. Algoritmo de búsqueda binaria. Determinación de pre condiciones y poscondiciones

Registros: definición de tipo y declaración de variables. Acceso a los campos de un registro Operaciones sobre registros Registros anidados. Arreglos como campo de un registro. Arreglo de registros.

Algoritmos de ordenamiento Cuadrático: Método de la Burbuja Mejorado, Método de Inserción Directa, Método de Selección. Ejemplos. Seguimiento de Algoritmos

Unidad 4: Análisis de Complejidad de Algoritmos

Eficiencia de los Algoritmos de búsqueda. Eficiencia de Algoritmos de Ordenamiento: Método de la Burbuja Mejorado, Método de Inserción Directa. Método de Selección

Unidad 5: Verificación Formal de Algoritmos

Conceptos Introductorios acerca de corrección de programas. Representación formal: terna de Hoare. Concepto de estado. Precondiciones y Postcondiciones . Reglas básicas de verificación.

Reglas específicas de verificación: asignación simple, composición secuencial, y alternativa

Dr. Mario Diaz

Prof. Titular Algoritmos y Resolución de Problemas

Sistema de Evaluación y Promoción

Está planificado un sistema de evaluación continua durante el desarrollo de las distintas unidades. La asignatura Algoritmos y Resolución de Problemas se puede aprobar con un régimen promocional o regular. Las siguientes son las condiciones para acceder a cada uno de ellos.

Régimen Regular

El alumno regular rinde un examen final integrador. Para obtener la condición de alumno regular deberá cumplir con el 75% de asistencia exigida para las clases prácticas, aprobar las prácticas obligatorias en máquina y los tres parciales planificados, ya sea en la primera instancia o en la recuperación. El alumno que hubiera aprobado dos de los tres parciales tiene derecho a una instancia de recuperación extraordinaria. (Ordenanza Nº 1/92 CD-FCEFN).

Régimen Promocional

Esta asignatura se promociona cumpliendo con el 75% de asistencia exigida para las clases prácticas, aprobando las prácticas obligatorias en máquina y los tres parciales con sus controles correspondientes a las unidades temáticas que se desarrollan. El régimen de aprobación de los parciales se ajusta a la Ordenanza Nº 1/92 CD-FCEFN que reglamenta el régimen de cursado promocional de las asignaturas en el ámbito de la FCEFN. Específicamente en lo establecido por los siguientes artículos:

- **Artículo 4**: Para tener derecho a las evaluaciones parciales el alumno deberá acreditar la **aprobación del 100%** de las actividades de enseñanza- aprendizaje obligatoria establecidas por la cátedra para cada período, **previo a la evaluación parcial.**
- **Artículo 5:** Las evaluaciones parciales serán correlativas entre sí y en el orden establecido por las cátedras, de manera que **el alumno no podrá rendir una evaluación parcial sin haber aprobado la precedente.** Se debe establecer entre las evaluaciones, instancias de recuperación **no prorrogables,** para alumnos reprobados o ausentes. Esta fecha se fija entre los 7 y 15 días **corridos** posteriores a la evaluación correspondiente Vencida esta fecha se pierde el derecho de recuperación.
- **Artículo 7:** La calificación de las evaluaciones parciales será de 0 a 10 puntos. Una evaluación se considerará aprobada cuando la calificación obtenida sea igual o mayor a 7 puntos.
- **Artículo 13:** Se considerará promocionado al alumno en la asignatura cuando haya **aprobado la totalidad de las evaluaciones parciales** y cumplido con los requisitos establecido por el artículo 4. La nota final será el promedio de las notas obtenidas en las evaluaciones parciales aprobadas, redondeándose a la cifra más próxima o por exceso.
- **Artículo 14:** El alumno quedará separado del régimen de promoción por evaluaciones parciales por las siguientes causas:
  - Decisión voluntaria
  - Incumplimiento de los requisitos establecidos en el artículo 4
  - No aprobación de las evaluaciones parciales

- **Artículo 15:** El alumno separado de este sistema podrá seguir el cursado de la materia por el régimen ordinario**.** A esos efectos los programas analíticos y de los trabajos prácticos serán idénticos

Se debe tener en cuenta que todas las evaluaciones: controles de información y Parciales se rinden de forma presencial, en algunos de los dos turnos en los que se dicta la asignatura

El alumno que por algún motivo quedara excluido del régimen Promocional podrá seguir cursando como alumno Regular.

El examen final será oral y escrito, y estará basado en el programa analítico presentado por la cátedra.

Cronograma de Evaluaciones

Se detallan los temas incluidos en cada Parcial y las actividades obligatorias que deben ser aprobadas para poder rendir cada uno de ellos en el régimen promociona y regular.

Cronograma Evaluaciones Régimen Promocional

Se detallan los temas incluidos en cada Parcial y las actividades obligatorias que deben ser aprobadas para poder rendir cada uno de ellos en el régimen promocional.

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL I	Resolución de problemas con una computadora. Etapas para resolución de problemas usando como herramienta la computadora. Algoritmos: concepto de algoritmo, datos, tipos de datos, expresiones, acciones. Resolución de problemas. Estructura de algoritmos que utilizan subprogramas. Seguimientos de algoritmos. Acciones estructuradas: Secuencia. Selección o alternativa. Selección múltiple. Iteración. Bucles anidados Contadores y acumuladores. Cálculo de máximos y mínimos de un conjunto de valores. Bandera lógica. **Parcial 1** **Fecha: 16/04** **Recuperación: 23/04**	**1. Control 1.1** (individual-escrito)**:** Resolución de problemas con una computadora. Etapas para resolución de problemas usando como herramienta la computadora. Algoritmos: concepto de algoritmo, datos, tipos de datos, expresiones, acciones. Resolución de problemas. Estructura de algoritmos que utilizan subprogramas. Seguimientos de algoritmos **2. Control 1.2** (individual-escrito)**:** Acciones estructuradas: Secuencia .Selección o alternativa. Selección múltiple. Iteración. Bucles anidados **3. Control 1.3** (grupal–oral)**.** Solución de Actividad en pseudocódigo y lenguajes c **Práctica en máquina grupal** **Recuperación Práctica en máquina**	**19/03** **31/03** **07/04** **7/04 - 10/04** **13/04 - 15/04**

Para rendir el **Parcial 1** o el **Recuperatorio del Parcial 1**, el estudiante debe haber **rendido** los tres controles información, haber **aprobado dos** como mínimo y **tener aprobada la práctica grupal en máquina**.

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL II	Arreglo lineal. Operaciones básicas sobre arreglos Registro. Arreglo de registros Algoritmos de ordenamiento **Parcial 2** **Fecha: 21/05** **Recuperación: 28/05**	**1. Control 2.1:** Carga ordenada y aleatoria. Manipulación. Búsquedas. **2. Control 2.2** Subarreglos.. Arreglos usados como acumuladores y contadores **3. Control 2.3:**. Registros, arreglos de registros **Práctica en máquina individual** **Recuperación Práctica en máquina**	**30/04** **07/05** **11/05** **11/05 - 15/05** **18/05 - 20/05**

Para rendir el **Parcial 2** o el **Recuperatorio del Parcial 2**, el estudiante debe haber **rendido** los tres controles información, haber **aprobado** como mínimo dos y tener **aprobada la práctica en máquina individual**.

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL III - Integrador	Análisis de eficiencia de Algoritmos - Análisis de métodos de búsqueda y de ordenamiento. Verificación de Algoritmos **Parcial 3 - Integrador** **Fecha: 11/06** **Recuperación: 18/06**	**1. Control 3.1:** Conceptos de Eficiencia. **2. Control 3.2:** Conceptos de verificación. Secuencia. **3. Control 3.3:** Conceptos de Verificación. Selección.	**01/06** **04/06** **08/06**

Para rendir y aprobar el **Parcial 3** el estudiante debe haber **rendido** los tres controles y haber **aprobado dos como mínimo**.

El estudiante que por algún motivo quedara excluido del régimen Promocional podrá seguir cursando como estudiante Regular. Aquellos parciales que haya obtenido nota igual o mayor a 7 pasarán a estar como "aprobados" en el régimen de estudiante Regular.

Obtención de Promocionalidad

El estudiante que haya aprobado los tres parciales obtiene la promoción de la materia y se colocará como nota final el promedio de notas de los tres parciales.

Tabla resumen de Evaluaciones Parciales Promocionales

Parcial	Recuperación
Parcial 1 16/04	23/04

Parcial 2	21/05	28/05
Parcial 3	11/06	18/06

Cronograma Evaluaciones Régimen Regular

El siguiente es el cronograma de evaluaciones correspondie
nte a estudiantes Regulares:

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL I	Resolución de problemas con una computadora. Etapas para resolución de problemas usando como herramienta la computadora. Algoritmos: concepto de algoritmo, datos, tipos de datos, expresiones, acciones. Resolución de problemas. Estructura de algoritmos que utilizan subprogramas. Seguimientos de algoritmos. Acciones estructuradas: Secuencia. Selección o alternativa. Selección múltiple. Iteración. Bucles anidados Contadores y acumuladores. Cálculo de máximos y mínimos de un conjunto de valores. Bandera lógica. **Parcial 1** **Fecha: 16/04** **Recuperación: 23/04**	**Práctica en máquina grupal** **Recuperación Práctica en máquina grupal**	**07/04 - 10/04** **13/04 - 15/04**

Para rendir el **Parcial 1** o el **Recuperatorio del Parcial 1** el estudiante debe haber **aprobado la práctica en máquina grupal.**

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL II	Arreglo lineal. Operaciones básicas sobre arreglos Registro. Arreglo de registros Algoritmos de ordenamiento **Parcial 2** **Fecha: 21/05** **Recuperación: 28/05**	**Práctica en máquina individual** **Recuperación Práctica en máquina individual**	**11/05 - 15/05** **18/05 - 20/05**

Para rendir el **Parcial 2 o el Recuperatorio del Parcial 2** el estudiante debe haber **aprobado la práctica en máquina individual**.

	TEMAS TEÒRICOS	ACTIVIDADES OBLIGATORIAS	FECHA
PARCIAL III	Análisis de eficiencia de Algoritmos - Análisis de métodos de búsqueda y de ordenamiento. Verificación de Algoritmos	**Asistencia obligatoria a la Práctica de Revisión Teórica del día 28-5**	**28/05**
	Parcial 3 **Fecha: 11/06** **Recuperación: 18/06**		

Para rendir el **Parcial 3** el estudiante debe haber asistido y participado en la clase **Práctica de Revisión Teórica**, en caso de no hacerlo deberá realizar una actividad extraúlica señalada por el docente a cargo.

Obtención de Boleta

El estudiante regular que haya aprobado dos de los tres parciales, tiene derecho a una recuperación extraordinaria.
Haber aprobado los tres parciales implica tener boleta, con lo cual el estudiante esta habilitado para rendir la materia en mesa de examen final.
El examen final es oral y/o escrito (a criterio de la mesa examinadora), y estará basado en el programa analítico presentado por la cátedra.

Tabla resumen de Evaluaciones Parciales Regulares

Parcial	Recuperación
Parcial 1 16/04	23/04
Parcial 2 21/05	28/05
Parcial 3 11/06	18/06
Extraordinario (para estudiantes Regulares) 25/06	

Contrato Pedagógico

Con el objetivo de promover y facilitar tu aprendizaje, la cátedra Algoritmos y Resolución de Problemas adopta un sistema de tutorías. Los tutores de la cátedra son docentes que te ***orientarán y acompañarán*** a transitar el proceso de enseñanza/aprendizaje para que puedas lograr paulatinamente ***mayores niveles de autonomía***.

La tarea del docente tutor no se limita a transmitir información, a dar respuestas, sino que debe brindar herramientas para que busques los caminos que den respuestas a tus interrogantes y te permitan solucionar las problemáticas que te planteamos. Procuraremos *enseñarte a aprender.*

Las siguientes son las obligaciones que ambos debemos respetar si deseamos cumplir los objetivos propuestos:

OBLIGACIONES DEL DOCENTE

1. Brindar las herramientas necesarias para el desarrollo del 100% de los contenidos a ser evaluados para la aprobación de la asignatura.
2. Orientar y acompañar el proceso de aprendizaje de sus alumnos, atendiendo sus características e intereses.
3. Promover la participación activa, crítica y reflexiva del alumno en clases.
4. Procurar el desarrollo autónomo de sus alumnos.
5. Promover la integración e interacción entre los alumnos.
6. Proponer trabajos en equipo para que el alumno pueda proponer y confrontar sus ideas un marco de respeto y cordialidad.
7. Cumplir los horarios de clase y de consulta con responsabilidad, puntualidad e idoneidad.
8. Relacionarse con sus alumnos en forma cordial y respetuosa, con predisposición para atender requerimientos

OBLIGACIONES DEL ALUMNO

1. Asistir regularmente a clases Teóricas y Prácticas.
2. Asistir regularmente a consultas.
3. Conocer los contenidos conceptuales pertinentes al asistir a la clase Práctica.
4. Realizar las actividades propuestas por los docentes.
5. Adquirir responsabilidad y compromiso por su propio aprendizaje.
6. Planificar y organizar su tiempo de estudio.
7. Desarrollar capacidad para comunicar sus puntos de vista en forma oral y escrita.
8. Participar en grupos de trabajos, respetando las ideas de los demás, a fin de lograr consenso.

Guía de Lectura

Este material se divide en unidades, cada una desarrolla un boque temático del programa de la materia, se incluyen ejemplos y actividades. Al final de cada unidad se encuentra bibliografía complementaria y un Práctico con ejercicios para resolver en clase.

El contenido está acompañado de referencias iconográficas en los márgenes, su significado es el siguiente:

Foco: hace referencia conceptos y aspectos destacados

Burbuja de pensamiento: indica un contenido a reflexionar e interpretar con detenimiento

Lápiz: indica una actividad o ejercicio a desarrollar.

Señalador de dirección:

Se usa para indicar qué parte del contenido se está desarrollando. Conjuntamente se utiliza la **lupa** (ver abajo) para indicar con precisión el tema que se desarrolla en ese punto del documento.
La Lupa aparece sobre el Esquema Temático de la Unidad.

Unidad 1: Algoritmos

Introducción

Constantemente el hombre se enfrenta a innumerables problemas que debe resolver. Muchas soluciones las logra apelando a lo aprendido con su experiencia que ha logrado internalizar con el correr del tiempo. Para otras, necesita el apoyo de herramientas aportadas por diversas ramas de la tecnología, siendo la computadora una de las de mayor influencia.

La computadora permite resolver problemáticas, mediante la ejecución de programas previamente cargados en ella. La construcción de estos programas requiere una serie de etapas desde el planteamiento del problema hasta llegar a obtener los resultados del mismo.

Es necesario tener en cuenta que la computadora no puede resolver problemas sin que el hombre la programe, es decir, determine la forma, y recursos necesarios para hacerlo.

La computadora en sus inicios aceptaba programas con instrucciones escritas en ceros y unos (lenguaje binario). Esta condición dificultó la creación de programas más grandes y complejos lo que originó el puntapié inicial para la generación de un canal de comunicación hombre-máquina, que permitiera interpretar nuestras solicitudes y a nosotros comprender sus resultados y mensajes. Esta necesidad de comunicación llevó al desarrollo de los lenguajes de programación.

Aprender un lenguaje de programación no resulta una tarea complicada, lo complejo es encontrar el camino adecuado para la resolución de la problemática planteada.

Es claro, que dar respuesta a un problema utilizando una computadora exige conocimientos, reflexión, razonamiento lógico y alguna dosis de ingenio y sagacidad.

Algunas personas aplican, generalmente de forma inconsciente, una serie de métodos y mecanismos que suelen resultar adecuados para abordar problemas. Estas operaciones mentales se conocen como procesos heurísticos. Una heurística es una regla práctica basada en la experiencia, sobre la cual no existe garantía de llegar a una solución.

Una persona puede aprender estrategias que aumenten su capacidad para resolver problemas, que de otra manera le hubieran resultado “difíciles”. Una estrategia, es un método general, una guía que puede aplicarse para hallar la resolución de muchas clases de problemas. Existen diferentes estrategias para enfrentar la resolución de un problema.

El objetivo de esta unidad es resolver problemas utilizando como herramienta una computadora. Para ello se aprenderá a construir algoritmos, esto es, un conjunto de pasos que la computadora deberá realizar para llegar a la solución requerida.

Figura 1.1 Resolución de problemas con la computadora

Para cumplir con el objetivo propuesto se requiere el abordaje de los tres ejes involucrados: el concepto de **problema,** sus componentes y etapas necesarias para su resolución, **las acciones básicas que puede realizar una computadora y** el concepto de **algoritmos** que incluye el análisis de su estructura.

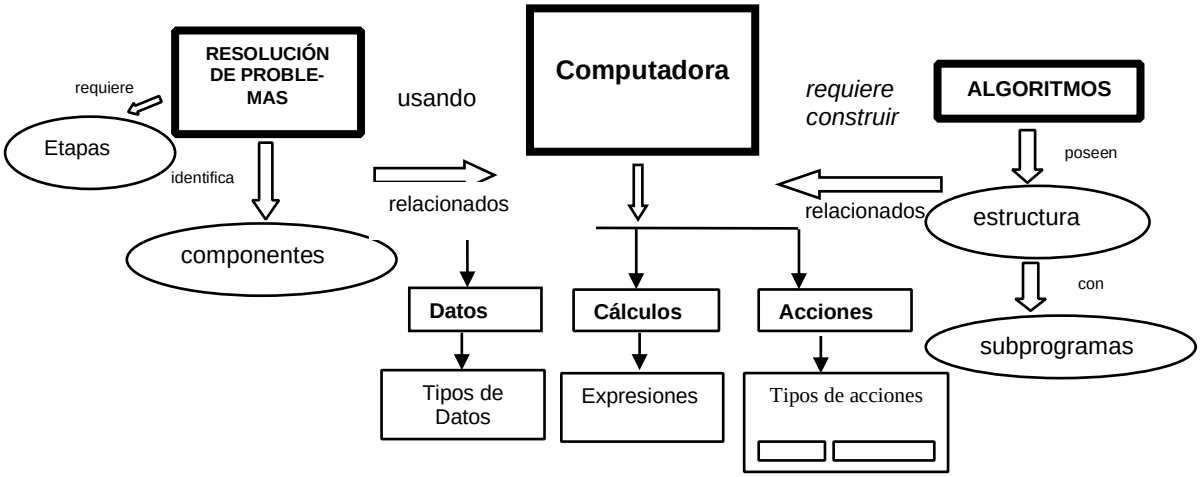
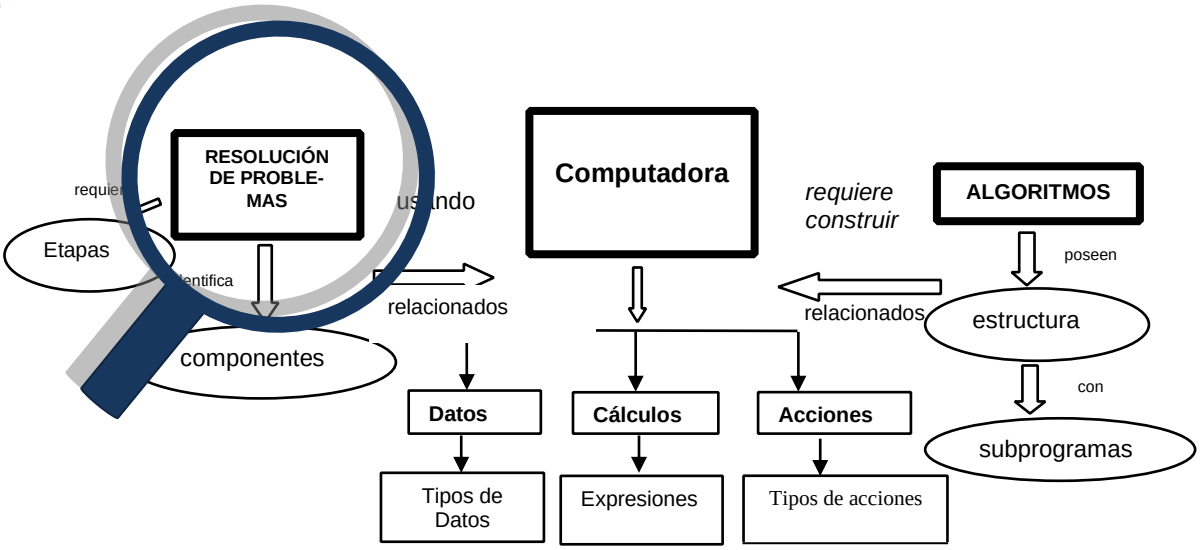


Figura 1.2 Esquema Temático de la Unidad

Se comenzará la unidad definiendo lo que se entiende por problema, cuáles son sus componentes y se describirán las etapas necesarias para que sea resuelto por una computadora.



1.1 Resolución de problemas con una computadora

¿Qué es un Problema?

Un problema es un desafío intelectual, ya que para llegar a una solución se deben desarrollar actitudes, hábitos y formas de pensamiento; se requiere aplicar y vincular conocimientos previos, probablemente de áreas diferentes, buscando nuevas relaciones.

Es importante definir qué se entiende por problema, considerando que esta palabra es usada en contextos diferentes y con matices diversos.

Etimológicamente, la palabra problema deriva del griego ***proballein*** y significa *algo lanzado hacia delante*. Un problema es un asunto o un conjunto de cuestiones que se plantean para ser resueltas.

El diccionario de la Real Academia Española da cinco acepciones diferentes para la palabra problema, pero la más adecuada en el contexto de este documento es: “Planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos”.

Otras definiciones son:

…"Tener un problema significa buscar, conscientemente, alguna acción apropiada para alcanzar o lograr el propósito claramente concebido; pero no inmediatamente alcanzable. Resolver un problema significa encontrar tal acción." (Polya, 1965)

"...es una situación en la cual un individuo actúa con el propósito de alcanzar una meta utilizando para ello una estrategia particular " (Chi y Glaser, 1983).

En este último caso, lograr una meta significa alcanzar una solución. Resolver el problema significa realizar actividades que operen sobre un estado inicial para lograr la solución.

En general se habla de un *problema*, cuando se plantea una situación cuya solución no es evidente, representa un obstáculo.

¿Cuáles son los componentes de un Problema?

En los problemas se pueden distinguir cuatro componentes (Mayer, 1983):
Meta: lo que se desea lograr.
Datos: información disponible para comenzar a analizar la situación problemática. Esto es, el conjunto de elementos que representa el conocimiento relacionado con el problema.
Restricciones: factores que limitan la vía para llegar a la solución.
Métodos: se refieren a los procedimientos utilizados para resolver el problema.

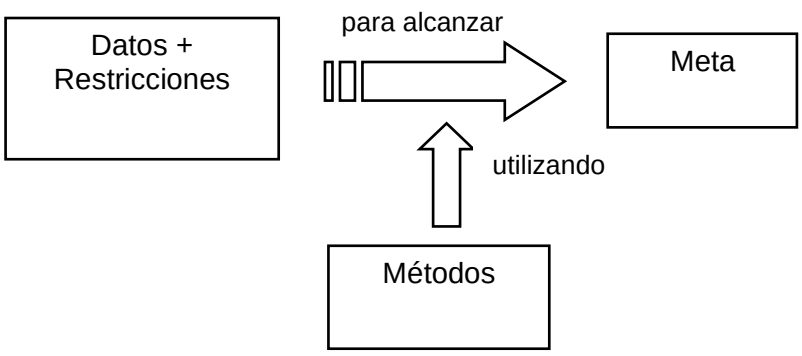


Figura 1.3 Componentes de un Problema

Existen problemas de distinta naturaleza: matemáticos, químicos, filosóficos, etc. En este libro, se analizarán problemas cuyas soluciones se deben obtener utilizando una serie de órdenes que se introducen a la computadora para que ésta las ejecute.

Etapas para resolución de problemas usando como herramienta la computadora.

La computadora es un dispositivo electrónico programable capaz de almacenar y procesar información. Pero, por sí sola no es capaz de resolver un problema; es necesario considerar tres elementos en el proceso:

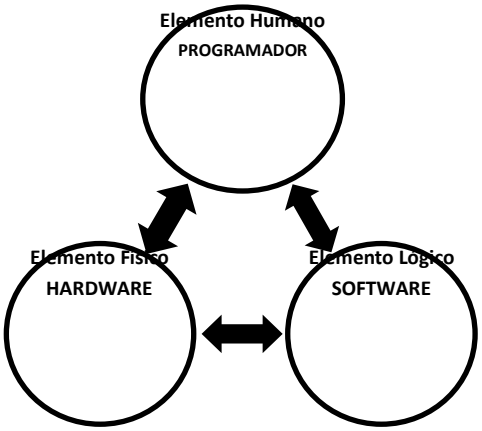


Figura 1.4 Elementos que intervienen en la resolución de un problema con una computadora.

- **Elemento humano**: llamado programador, es el elemento más importante, sin él la computadora no podría resolver ningún problema.
- **Elemento lógico**: llamado Software: es el conjunto de programas, datos, información, que hacen posible el uso y funcionamiento del computador. Se clasifica en Software de base (Sistemas operativos, que controla el funcionamiento del equipo físico y gestiona todos sus recursos) y los softwares de aplicación (conjunto de programas diseñados con el fin de resolver distintos problemas).
- **Elemento físico**: llamado Hardware: conformado por los componentes propios de la computadora. Entre sus principales componentes citamos: las unidades o dispositivos de entrada y de salida, la unidad central de proceso (unidad de control, unidad aritmética lógica y memoria). (Ver Fig. 1.5)

Mediante el hardware -elemento físico- y el software de base -sistemas operativos- la computadora es capaz de realizar las siguientes acciones básicas:

Acciones básicas de una computadora

1. Reconocer distintos tipos de datos
2. Almacenar los datos en memoria.
3. Realizar operaciones aritméticas, relacionales y lógicas, mediante un subsistema conocido como Unidad Aritmético-Lógica (UAL).
4. Reconocer y ejecutar acciones simples y estructuras lógicas de control.

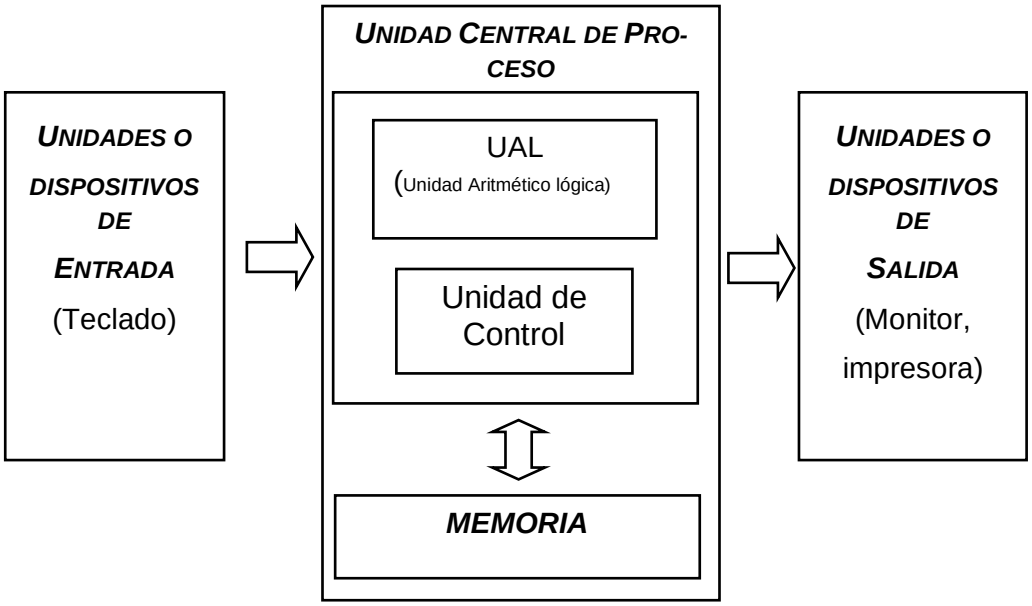


Figura 1.5 Componentes de hardware de una computadora

A partir de estas acciones básicas es posible que el programador pueda construir programas (software de Aplicación) que permitan resolver problemas de la vida real.

Buscar la solución de problemas es una tarea compleja de sistematizar, puede pensarse como un proceso de búsqueda en un espacio de soluciones potenciales, y para ello se pueden tener en cuenta ciertas pautas para facilitarla

Así, George Polya, considerado e*l padre de las Estrategias para la Solución de Problemas* fue un matemático húngaro cuyos aportes incluyen documentos y libros que promueven un acercamiento al conocimiento y desarrollo de estrategias en la solución de problemas.

Su famoso libro *Cómo Plantear y Resolver Problemas*, introduce su método de cuatro pasos y estrategias específicas útiles en la solución de problemas:

Los cuatro pasos propuestos por Polya son los siguientes:

1. Entender el problema.
2. Configurar un plan.
3. Ejecutar el plan.
4. Mirar hacia atrás (verificar procedimientos y comprobar resultados)

La teoría de Polya brinda bases sobre las cuales se pueden establecer las etapas para resolver un problema con una computadora.

Para llegar a ser un programador eficaz se necesita aprender a resolver los problemas de un modo riguroso y sistemático. Existen diversas estrategias para configurar el plan

que resuelve un problema, la que se adecua y se utilizará es la de construcción de algoritmos.

A partir de las etapas que define Polya se proponen las siguientes etapas para la resolución de problemas con una computadora.

Etapas resolución de problemas utilizando computadoras	***Objetivo***
Análisis del problema	Ayuda a la comprensión del problema, a comprender **¿qué se debe hacer?**
Diseño del Algoritmo	Se determina **¿cómo se debe hacer?** Permite indicar cómo el algoritmo realizará la tarea
Codificación: Construcción del programa	Se traduce el algoritmo a un lenguaje de programación para que pueda ser interpretado y ejecutado por una computadora.
Compilación y ejecución	
Validación	Permite comprobar la solución obtenida. Validación mediante pruebas Validación mediante verificación o Validación formal
Documentación	Se registra lo que se realiza durante todo el proceso, y en cada una de las distintas etapas.

Tabla 1.1 Etapas resolución de problemas utilizando computadoras

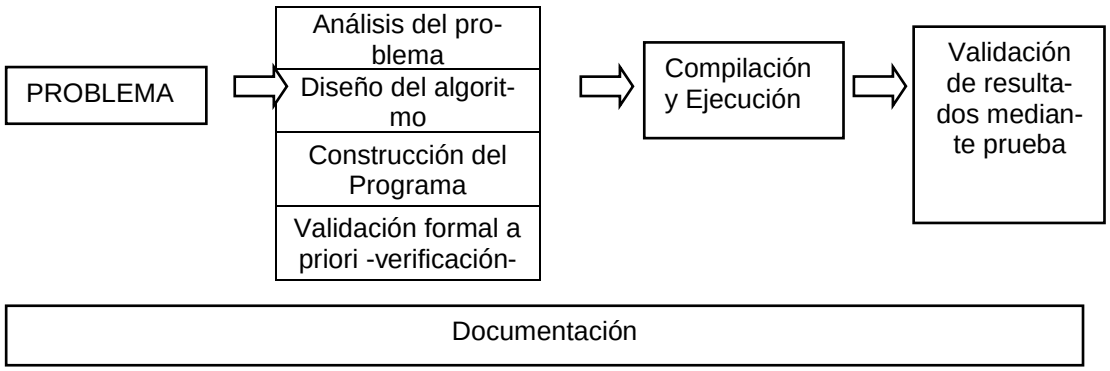


Figura 1.6 Temporalidad de las etapas en la resolución de problemas con la computadora

Si bien el objetivo de este libro está centrado las dos primeras etapas, se describirá brevemente en qué consiste cada una de ellas.

Etapa I: Análisis del problema
La primera condición al momento de resolver un problema, es contar con un enunciado preciso y claro, que no contenga ningún tipo de ambigüedades. Esto es fundamental ya que la correcta resolución de un problema viene determinada en gran medida por el planteamiento inicial, no se puede abordar una solución mientras no se tenga claro a donde se quiere llegar. El programador, no comenzará a intentar solución alguna hasta que no tenga claro las especificaciones y requerimientos del usuario. Un planteamiento correcto evitará perder tiempo en la implementación de algoritmos incorrectos.

Por lo tanto en esta etapa se determina **¿qué se debe hacer?** Para poder definir bien un problema es conveniente identificar las especificaciones de entrada y salida del problema, es decir, responder a las siguientes preguntas: **¿Qué datos de entradas se requieren? (cantidad y tipo) ¿Cuál es la salida deseada (resultados)? (cantidad y tipo) ¿Que método produce la salida deseada?**

Etapa II: Diseño del Algoritmo.

En la etapa de análisis se determina ¿**qué hacer?,** en la etapa de diseño se debe determinar ¿**cómo se debe hacer**? El procedimiento elegido para la solución se debe expresar como una secuencia ordenada de acciones, que el procesador deberá ejecutar para obtener los resultados pretendidos. Esta secuencia de acciones expresadas en un lenguaje simbólico se conoce con el nombre de **Algoritmo**. Inicialmente, el algoritmo se expresa en lenguaje natural, el que se usa habitualmente para comunicarse con otras personas, este seudo-lenguaje recibe el nombre de **Seudocódigo**.

Etapa III: Codificación. Construcción del Programa.

Un algoritmo es una especificación simbólica que debe traducirse a un lenguaje de programación para que pueda ser interpretado por la computadora, obteniéndose de esta manera lo que se conoce como ***programa fuente***. El proceso de traducción del algoritmo a un lenguaje de programación se conoce como **codificación**.

Etapa IV: Compilación y ejecución

Utilizando el editor del lenguaje seleccionado, se podrá depurar el programa fuente de los errores de sintaxis. Si el código contiene errores debidos a que el programador no respetó la sintaxis del lenguaje, es decir errores de compilación, el compilador los detecta y especifica por medio de mensajes. El programa ejecutable se creará una vez solucionados estos errores.

Etapa V: Validación

La validación permite comprobar que un programa cumple con sus especificaciones, es decir resuelve correctamente el problema para el que fue diseñado.

Los métodos de validación se pueden clasificar en dos grandes grupos:

- Validación mediante pruebas
- Validación mediante verificación o Validación formal

En las primeras unidades se comenzará realizando validación por pruebas, e introduciendo conceptos de la Validación mediante verificación, los cuales serán ampliados y formalizados en la Unidad V.

La ***Validación mediante pruebas*** consiste en *e*jecutar el programa con lotes de prueba para comprobar si los resultados obtenidos coinciden con los esperados.

Los resultados anómalos deberán ser analizados para detectar su causa, los cuales se eliminan en la etapa de depuración del programa. El procedimiento de prueba-depuración se repite hasta que no se detectan errores y se tenga confianza que el programa fue lo suficientemente probado. El problema de este tipo de validación es de-

terminar si el juego de pruebas es lo suficientemente confiable. Lo ideal sería contar con el *caso de pruebas perfecto* que permita asegurar que el programa se ejecutará correctamente para cualquier entrada posible. Algunos autores sugieren definir juegos de prueba que aseguren, que la ejecución del programa pase por todos los caminos posibles (juegos de prueba mínimamente perfectos)

Una vez que el programa es depurado de los errores de sintaxis, puede ocurrir que al ejecutarse no realice lo que se pretendía, que no se llegue al objetivo propuesto. El programa puede tener errores de lógica, éstos pueden resultar difíciles de detectar, se producen cuando el algoritmo está mal implementado y deben ser corregidos por el programador. Se debe recordar que la computadora realiza lo que se le indica, aunque estas acciones estén incorrectas.

En general para la verificación se utilizan datos válidos representativos del problema. Si bien es imposible realizar una verificación exhaustiva, se debe tratar de incluir algunos casos excepcionales válidos y datos no válidos, para los que el algoritmo debe dar una respuesta adecuada.

La ***Validación por verificación*** consiste en demostrar formalmente que el programa es correcto. Se realiza el análisis *sin ejecutar* el programa, con lo cual se asegura que el programa es válido para cualquier valor de los datos.

La facilidad de verificación y depuración de errores de funcionamiento del programa, conducen a mejorar su calidad, este es uno de los objetivos de la ingeniería de Software.

Etapa VI: Documentación

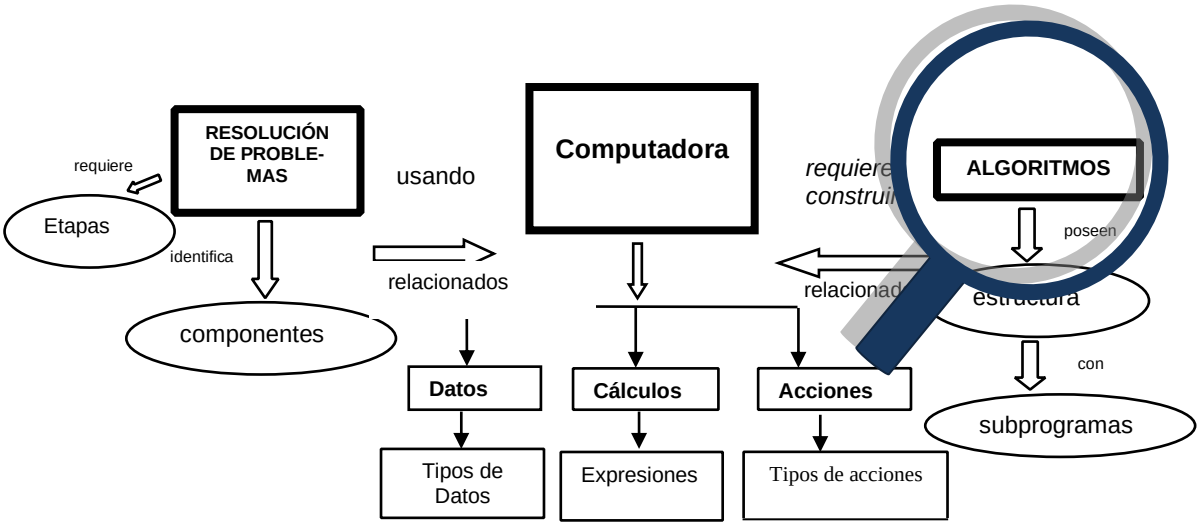
Consiste en ir registrando todo el proceso desde el análisis hasta la validación, fundamentalmente agregando al algoritmo y al programa construido, los comentarios que faciliten comprensión de lo que hace cada acción. Se consigue de esta manera algoritmos y programas más fáciles de interpretar y mantener.

Para el caso de programas de una envergadura importante, es conveniente colocar un comentario que especifique en forma general el objetivo del programa, si está dividido en partes indicar la función de cada una, el autor, fecha de realización y toda información que pueda resultar de interés. Estos comentarios serán de utilidad para un programador que debe utilizar un programa después de un cierto período de tiempo o si el mismo debe ser utilizado por otros programadores.

ACTIVIDAD 1

Realizar una lista con todos aquellos conceptos nuevos y significativos, analizar cómo se relacionan y elaborar un esquema conceptual con ellos.

***Hasta el momento se estudiaron aspectos importantes referidos al problema, ahora se analizará el concepto y estructura de un algoritmo*.**



1.2 Algoritmos

Concepto de Algoritmo

El proceso de análisis del mundo real para identificar los aspectos esenciales de un problema se llama **abstracción** y la expresión de estos aspectos, atendiendo a las especificaciones del problema se denomina **modelización**. Los elementos que participan en ese modelado se llaman **datos**. Resolver un problema requiere transformar mediante un proceso, los *datos de entrada* en datos de salida. Este proceso es representado por un algoritmo que una vez codificado en un lenguaje de computación se transforma en un programa.

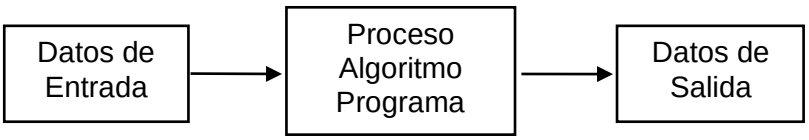


Figura 1.7 Procesamientos de datos

La palabra "algoritmo" deriva del nombre latinizado del matemático árabe Mohamed Ibn Moussa Al Kow Rizmi, el cual escribió entre los años 800 y 825 su obra Quitab Al Jabr Al Mugabala, donde se recogía el sistema de numeración hindú y el concepto del cero. Fue Fibonacci, el que tradujo su obra al latín y la inició con las palabras: Algoritmi dicit. Así, de la palabra *algorismo*, que originalmente hacía referencia a las reglas de uso de la aritmética utilizando dígitos árabes, se evolucionó a la palabra latina, derivación de al-Khwarizmi, algobarismus, que más tarde mutaría a *algoritmo* en el siglo XVIII.

Un Algoritmo, se puede definir como:

- Una secuencia de acciones que representan un modelo de solución para determinado tipo de problemas.
- Un conjunto de acciones que realizadas en orden, conducen a obtener la solución de un problema.

A partir de estas definiciones se puede concluir que:

Un Algoritmo es un conjunto ordenado y finito de pasos que permite solucionar un problema

De esta definición se desprenden importantes propiedades que debe cumplir un algoritmo:

Precisión: Cada acción debe tener un significado preciso, esto es, su interpretación no debe dar lugar a ambigüedades.

Finitud: Debe ejecutarse con una cantidad finita de pasos.

Efectividad: Un algoritmo es efectivo si sus acciones conducen al resultado buscado y se realizan en un tiempo finito. Se debe tener en cuenta que un algoritmo puede ser finito y no efectivo.

Otras características que debería poseer un algoritmo son:

Robustez: significa que un algoritmo debe contemplar todas las posibles facetas del problema que se quiere resolver, es decir, debe ser flexible a cambios. Si se logra construir un algoritmo robusto, cualquier giro inesperado del problema será controlado por él.

Correctitud: un algoritmo es correcto cuando da una solución al problema, cumple con todas las especificaciones y alcanza los objetivos planteados.

Completitud: significa que el algoritmo cuenta con todos los recursos para poder llegar a una solución satisfactoria.

Eficacia: un algoritmo es *eficaz* cuando da una solución al problema planteado.

Eficiencia: un algoritmo es *eficiente* cuando cumple los requerimientos funcionales especificados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria y tiempo de resolución.

Si bien los conceptos de **algoritmo** y **programa** están muy vinculados, es importante no confundirlos entre sí.

Algoritmo, es la especificación de un conjunto de pasos orientados a la resolución de un problema de un modo entendible para el ser humano pero que no puede ser ejecutado por la computadora, mientras que un **programa** es ese conjunto de pasos especificadas en un determinado lenguaje de programación, como por ejemplo el lenguaje C.

Un **algoritmo** constituye la documentación principal que se necesita para iniciar la fase de codificación y para representarlo se utilizan dos tipos de notación: **pseudocódigo y diagramas de flujo**.

Diagrama de flujo: esta técnica utiliza símbolos gráficos para la representación del conjunto de pasos que constituye un algoritmo.

Pseudocódigo: es una técnica de representación de algoritmos que utilizando palabras clave en lenguaje natural (el español) expresan el conjunto de pasos/acciones que requiere la solución del problema

El diseño de un algoritmo es independiente del lenguaje que se utilice en la codificación.

Estructura de un Algoritmo

Un algoritmo escrito en pseudocódigo se organiza en dos secciones: *cabecera* y *cuerpo*.

En la sección de **cabecera** se escribe el nombre del algoritmo.

La sección del **cuerpo** consta de las **declaraciones** de variables y las **acciones**.

En la parte de declaraciones se especifican los objetos (variables y constantes) que va a utilizar el algoritmo.

En la parte de acciones están descriptas las instrucciones necesarias para resolver el problema planteado.

Como ejemplo, la Figura 1.8 muestra un algoritmo que permite leer el nombre y las tres notas obtenidas en cada materia correspondiente al curso de ingreso, por cada uno de los 125 alumnos que rindieron. Y mediante un mensaje indica si el alumno aprobó o no el examen, sabiendo que se aprueba con promedio mayor a 6.50. En la Figura están destacadas cada una de las partes que conforman la estructura de un algoritmo.

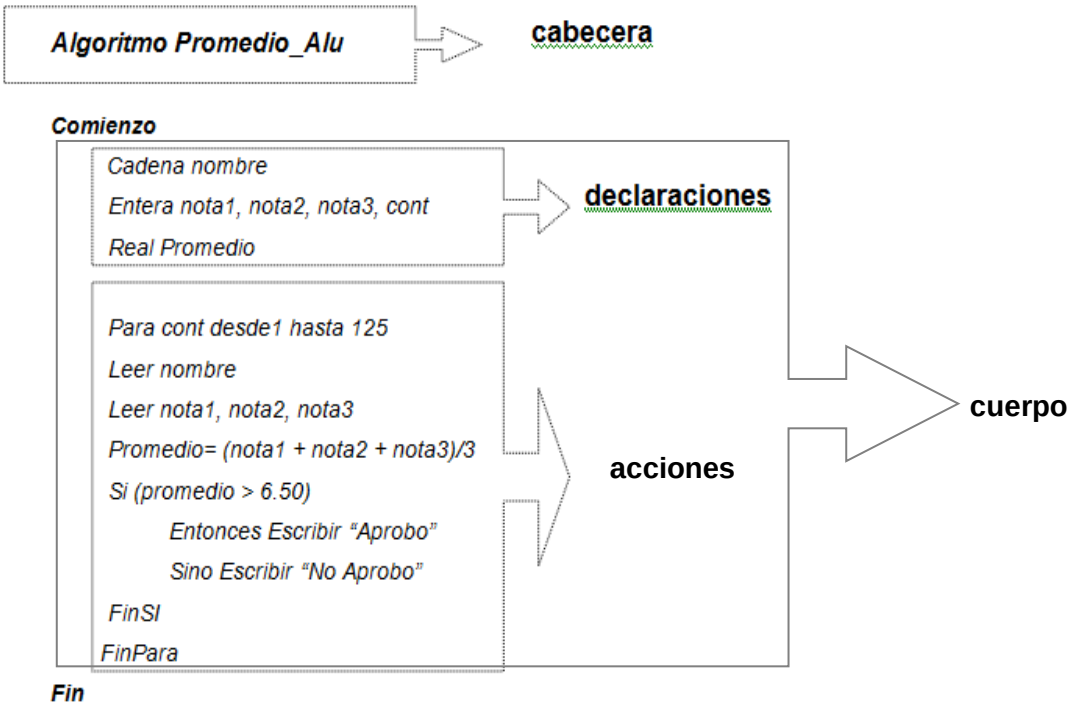


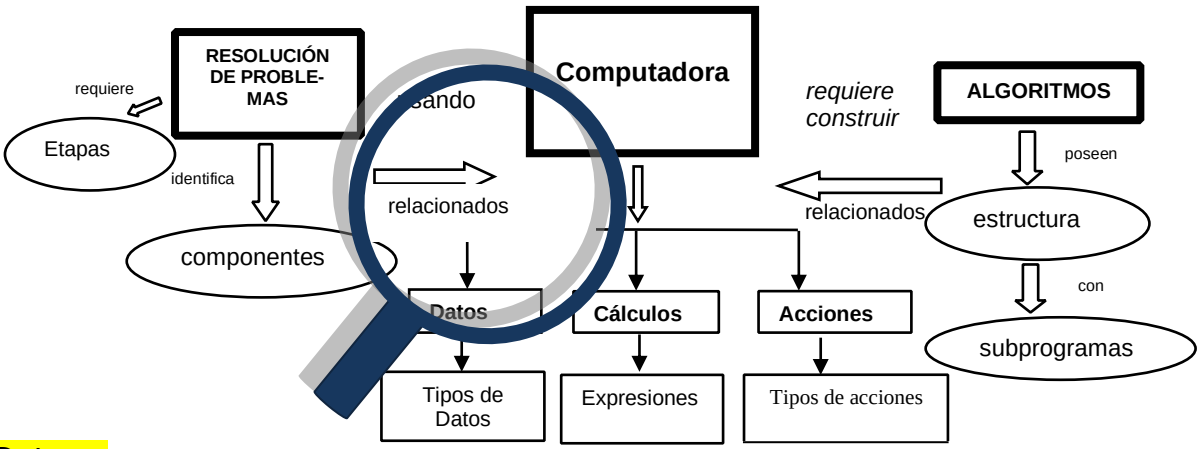
Figura 1.8 Ejemplo de Algoritmo, destacando su estructura

Elementos de un Algoritmo

Diseñar un algoritmo es una tarea donde la creatividad y experiencia del programador juega un papel importante y que difícilmente podrá ser del todo automatizada. Es necesario tener en cuenta algunas consideraciones a la hora de construir un algoritmo.

Como se dijo, los algoritmos serán escritos utilizando un lenguaje simbólico llamado pseudocódigo. Si bien es cercano al lenguaje natural, utiliza un conjunto de acciones y reglas expresadas en un vocabulario limitado, de tal manera que pueda ser interpretado por la comunidad de programadores.

Para diseñar algoritmos, en primer lugar, se analizarán los componentes de cada una de sus partes. En la sección declaraciones se coloca el nombre y tipo de los datos. Se mencionaron los conceptos de datos de entrada y datos de salida, esto requiere la necesidad de definir qué es un dato y cuáles son los tipos de datos que puede manipular una computadora.



Datos

Cuando se resuelven problemas con computadora, los objetos reales se deben modelar mediante objetos abstractos, representables y entendibles por una computadora.

Un **dato** es la representación de un elemento u objeto de la realidad, de tal manera que pueda ser procesado por la computadora.

Los datos pueden representar distintos objetos de la realidad como por ejemplo: temperaturas, distancias, edades, categoría, antigüedad, nombres, objetos que son representados por códigos preestablecidos, etc.

Atributos de los datos

Un dato tiene asociado un **Nombre**, un **Valor**, un **Tipo**, y una **dirección de memoria** de la computadora. En general esto se simboliza así:

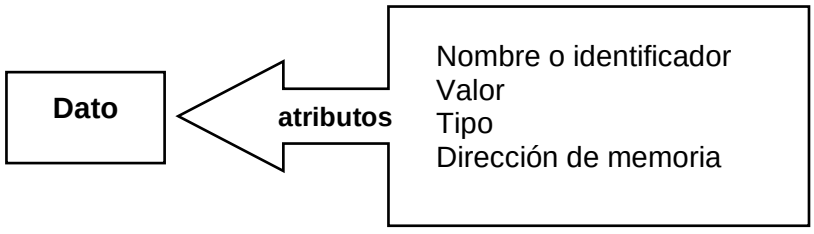


Figura 1.9 Atributos de los datos

Es importante destacar que todo dato a ser procesado por la computadora debe estar almacenado en la memoria principal. Se puede considerar conceptualmente, que la memoria está constituida por un segmento en donde se almacenan los datos y otro en donde se almacenan las acciones del algoritmo.

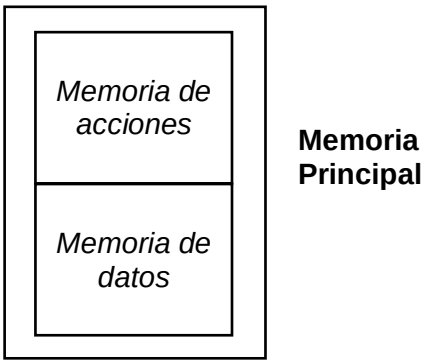
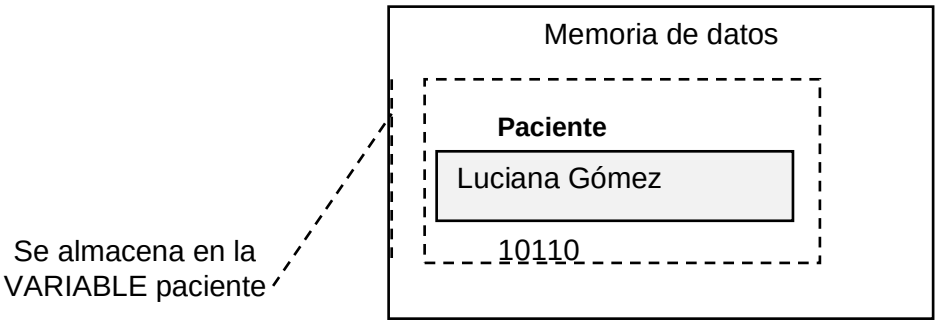


Figura 1.10 Representación de la Memoria Principal de la Computadora

Ejemplo 1
Si se cuenta con información de un paciente de una clínica, un dato podrá ser el nombre del paciente
Entonces:

- *Nombre del dato* → → Paciente
- *Valor del dato* → → Luciana Gómez
- *Tipo de dato* → → Conjunto de caracteres alfabéticos
- Dirección de memoria → → 10110

Para este ejemplo el almacenamiento en memoria se graficaría del siguiente modo:



Variables y Constantes

En general, en un algoritmo es necesario manipular datos cuyo valor cambia, estos datos son llamados variables. Entonces se puede definir:

Una **variable** es una posición de memoria que se referencia con un nombre y donde se almacena el valor que puede cambiar durante la ejecución del programa.

Cuando los datos tienen asociado un valor que no cambia durante el procesamiento del algoritmo, se habla de *Constantes*.

Una **constante** es el nombre simbólico de una posición de memoria donde se almacena el valor de un dato que no puede cambiar durante la ejecución del programa.

Identificador de variables y constantes

El nombre con que se designa una variable o una constante, se llama también identificador y en general está representado por una secuencia de uno o más caracteres: letras, dígitos y/o algunos caracteres especiales. El primer carácter debe ser una letra y el nombre no debe contener espacios en blanco.

Se aconseja seleccionar identificadores para variables y constantes representativos de los datos asociados para facilitar su interpretación y/o su posterior modificación, se dice que los identificadores deben ser autoexplicativos.

ACTIVIDAD 2

Proporcionar ejemplos de identificadores para las variables que almacenen los siguientes datos: distancias, edades, categoría, antigüedad, sueldo, rendimiento académico de un alumno, el éxito o fracaso de un tratamiento para dejar de fumar, nombre de una persona.

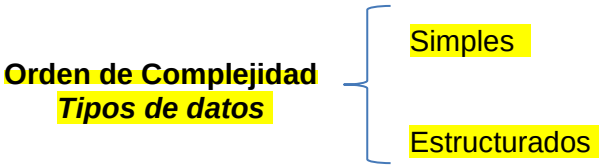
Tipos de Datos

Para determinar el proceso que deberá realizar una computadora para obtener el resultado o salida de un problema planteado, es necesario conocer el tipo de datos que ella puede manipular.

Los datos que utilizan los programas se pueden clasificar en base a diferentes criterios. Uno de ellos los clasifica en simples o estructurados.

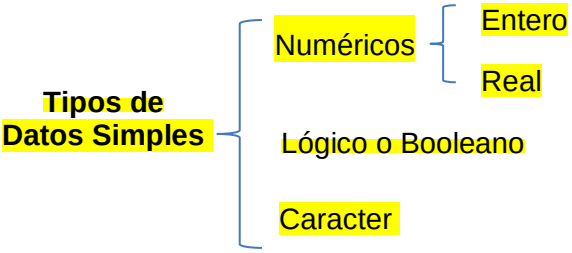
Si un dato contiene un valor que siempre se trabaja como una unidad, se dice que es un **dato simple**. Cuando el dato está formado por una colección de otros datos, se dice que es una estructura de datos o un **dato estructurado.**

Un **tipo de dato** define el rango de valores que puede tomar un dato y las operaciones que se pueden realizar con él.



Tipos de Datos Simples

Un dato simple es indivisible (atómico), es decir, no se puede descomponer. Los tipos de datos simples que definen la mayoría de los lenguajes de programación son:



Tipo de Dato Numérico

Los datos de tipo numéricos son aquellos que representan cantidades y con ellos se pueden realizar todas las operaciones aritméticas y relaciones aplicables a los números.
A su vez los datos numéricos pueden ser Tipo entero y Tipo real

***Tipo entero*:** este tipo permite representar los valores de los números enteros, por lo tanto, son útiles para simbolizar edades, códigos numéricos, números de calles, resultados provenientes de procesos de conteo, etc.
***Tipo real*:** este tipo permite representar valores con punto decimal, valores de números reales. Ejemplos de variables de tipo real son las que representan notas, temperaturas, presión arterial, altura, peso, costos, sueldo, etc.

Ejemplo 2

- Un año es un dato simple de tipo entero. Ejemplo 2017
- Una altura es un dato simple de tipo real. Ejemplo 1.70

Declaración de variables

Toda variable que se utilice en un algoritmo debe ser declarada, esto es, se debe especificar su identificador y el tipo de dato asociado. La declaración de variables permite a la computadora reservar el espacio de memoria necesario para el almacenamiento de su valor, como así también verificar si las operaciones que se realizan con ella corresponden al tipo declarado.
Por convención y para adecuarnos a la forma en que trabaja el lenguaje C, para declarar una variable se coloca el identificador de la misma precedido por su tipo.

Ejemplo 3
Las siguientes son las declaraciones de algunas variables de los datos de la actividad 2:

Dato**: temperatura** → Declaración Variable: **real temp**
Dato**: edad** → Declaración Variable: **entero ed**
Dato**: sueldo** → Declaración Variable: **real suel**

Estas declaraciones se interpretan de la siguiente forma:

temp es una variable de tipo numérico real que almacenará una temperatura
ed es una variable de tipo numérico entero que almacenará una edad
suel es una variable de tipo numérico real que almacenará un sueldo

Tipo de Dato Caracter

Una variable tipo caracter es aquella que puede almacenar un caracter del conjunto de caracteres que maneja el sistema operativo de la computadora. La mayoría de las computadoras utiliza el código ASCII -Código Estándar Americano para el Intercambio de Información-.

Esto es una tabla que asocia cada caracter con un número que indica su posición en la misma. Es un código normalizado que cuenta con 128 caracteres, que incluye números, letras mayúsculas, minúsculas y caracteres especiales. Actualmente, la mayoría de los procesadores utilizan el código ASCII ampliado que cuenta de 256 caracteres, entre los que se han agregado los caracteres del alfabeto griego. Estos caracteres están ordenados de 0 a 255, característica que permite la comparación entre ellos. Los dígitos están ordenados en su propia secuencia numérica y las letras están dispuestas

acorde al orden alfabético, precediendo las mayúsculas a las minúsculas, como se muestra en la siguiente tabla.

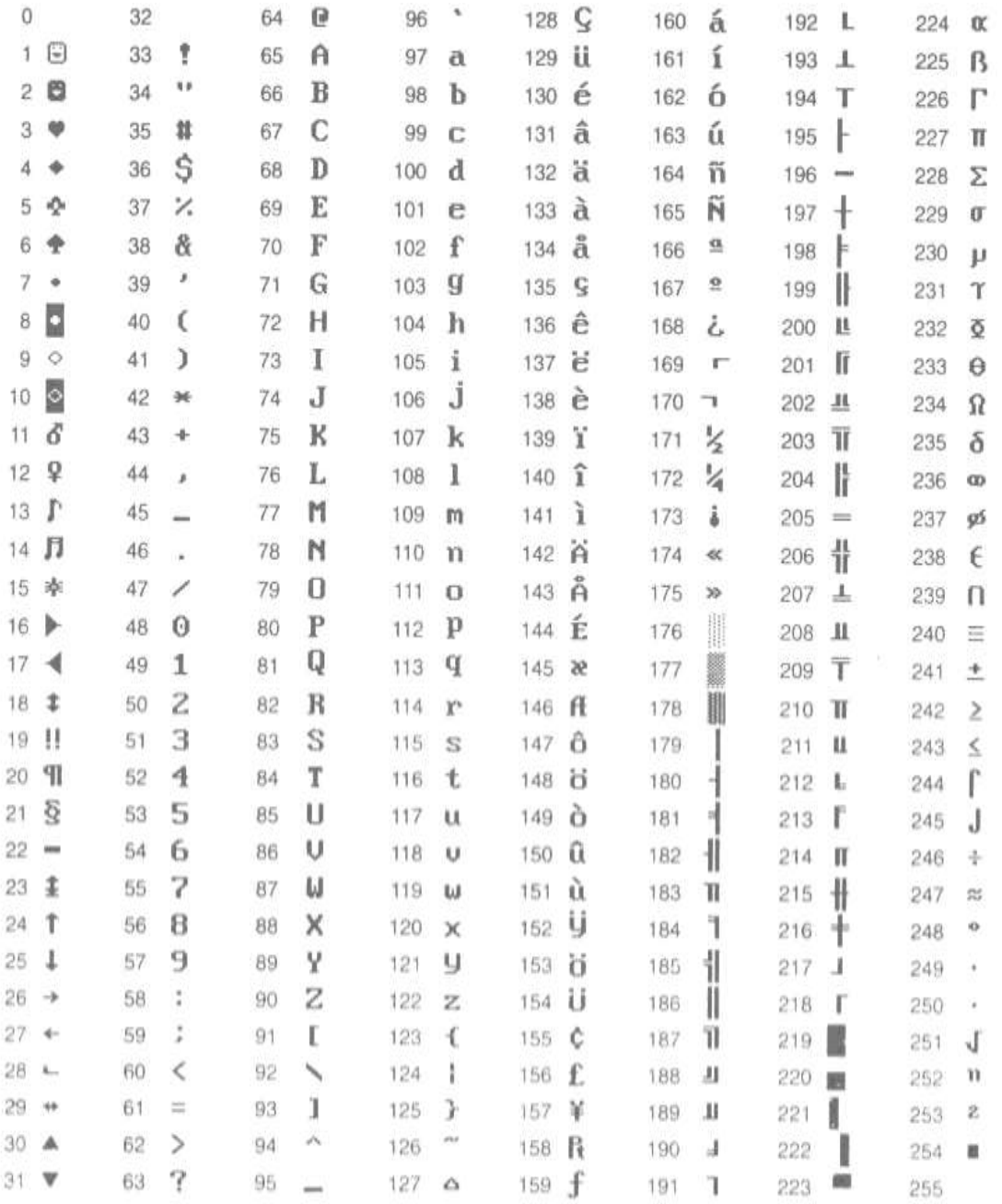


Tabla 1.2 Código ASCII

En general, todas las computadoras manipulan los siguientes datos:

Letras mayúsculas: A..Z. (excepto CH, Ñ, LL)
Letras minúsculas: a..z. (excepto ch, ñ, ll)
Dígitos decimales: 0..9.
Carácter de espacio en blanco.
Caracteres especiales: +, -, %,...@, µ. λ., α
Signos de puntuación: , ; :

Estos datos alfanuméricos son llamados caracteres alfanuméricos y se pueden clasificar en:

Caracteres alfabéticos: son las letras de la A a la Z (excepto la Ñ), ya sean mayúsculas o minúsculas.
Caracteres numéricos: son los dígitos del 0 al 9 tratados como caracteres, con ellos no se pueden realizar operaciones aritméticas porque no son números.
Caracteres especiales: guiones, paréntesis, asterisco, signos de puntuación etc.

Ejemplo 4
Si se que requiere representar el éxito o fracaso de un tratamiento para dejar de fumar, se puede declarar:

Caracter **Tratamiento_fumar**

Esto significa que la variable Tratamiento_fumar posee los siguientes atributos:

Nombre de la variable → Tratamiento_fumar
Tipo de dato → Caracter
Valor → 'E' (para el caso en que si tenga éxito) y 'F' (para el caso en que no tenga éxito)
Dirección de memoria → 1101

Para este caso:
La variable **Tratamiento_fumar** que representa la información, almacenará como valor una letra **E** si el tratamiento tuvo éxito o el carácter F para el caso que el valor a almacenar sea que el tratamiento no tuvo éxito.

Tipo de Dato Lógico

Existen datos que representan valores *Falso* o *Verdadero,* a ellos les corresponden el tipo de dato lógico o Booleano[1].

Ejemplo 5
La variable **Tratamiento_fumar** del ejemplo 4 se puede declarar también como una variable lógica cuyo valor será verdadero si el tratamiento tuvo éxito ó falso si no tuvo éxito.
De esta manera
Nombre del dato → Tratamiento_fumar
Tipo de dato → booleano
Valor → **verdadero** (para el caso en que tenga éxito) y **falso** (para el caso en que no tenga éxito)
Dirección de memoria → 10101

La variable que almacenará este dato se declara:

logico **Tratamiento_fumar**

[1] Llamado así en honor a George Boole, matemático inglés (1815-1864) que desarrolló la teoría del algebra binaria, que es la base de la representación de los circuitos lógicos.

Observación: Los tipos de datos cuyos elementos están ordenados discretamente se los denominan también **tipos ordinales**, esto es, para cada elemento, salvo el primero y el último, existe un anterior y otro posterior.
De los tipos de datos estudiados, el único que no es ordinal es el conjunto de los números reales, dado que en un número real no se puede determinar el anterior ni el siguiente, ya que entre dos números reales siempre existe otro número real.

Tipos de Datos Estructurados

Tipo de Dato Cadena

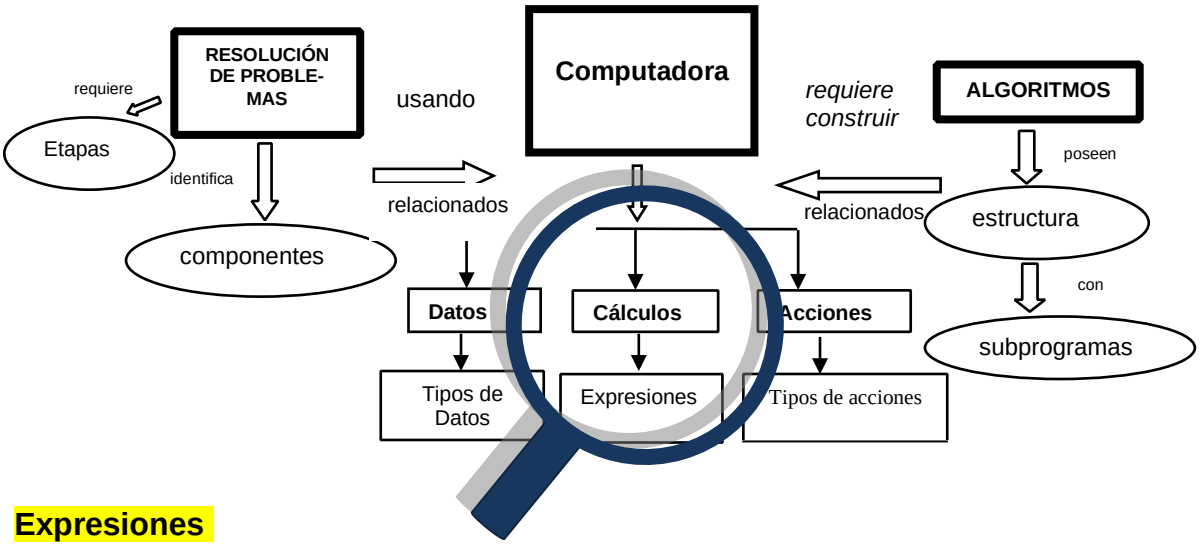
En muchos casos es necesario trabajar con datos representados por un conjunto de caracteres, por ejemplo, las que se utilizan para representar nombres. En estos casos no es posible el uso del tipo carácter, por ello la mayoría de los lenguajes de programación cuentan con el tipo de dato cadena, que permite resolver la situación planteada.

Un dato tipo cadena puede tomar como valor una cadena de caracteres, esto es, una secuencia finita de caracteres encerrada entre **comillas** **“ ”**.

Ejemplo 6
Si la variable **nomb** es utilizada para almacenar el nombre de una carrera que se cursa en la Facultad de Ciencias Exactas, se puede declarar:

cadena nomb
Nombre del dato → nomb
Tipo de dato → Cadena
Valor → Licenciatura en Ciencias de la Computación
Dirección de memoria → 10101

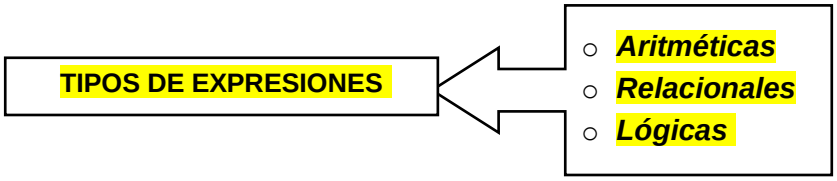
Una vez analizados los tipos de datos que puede manipular una computadora, se estudiarán los cálculos que ella puede realizar. Estos cálculos reciben también el nombre de expresiones.



Expresiones

Una **expresión** es la descripción formal de un cálculo, que al ser evaluado tiene un único resultado. Se construyen con la combinación de operandos y operadores.

Los operadores indican el tipo de cálculo a realizar y los operandos son constantes, variables u otras expresiones que se calculan con los operadores correspondientes. Según los operadores utilizados las expresiones se clasifican en:



Expresiones aritméticas

Una **expresión aritmética** es la descripción de un cálculo matemático donde los operandos son variables o constantes numéricas, los operadores son aritméticos y el resultado es un número.

Operandos	Operadores	Tipo de Resultado
Variables o Constantes numéricas	**+** suma **-** resta ***** multiplicación **/** división real **div** división entera Raiz Resto Potencia	Numérico

Tabla 1.3 Expresiones Aritméticas

- La **suma, resta y multiplicación** admiten operandos enteros y reales, por lo que su resultado puede ser entero o real.
- La **división /** admite operandos de tipo entero o real y da un resultado entero o real, dependiendo del operador utilizado.
- La división **div** admite operandos de tipo entero y da un resultado de tipo entero.
- Los operadores **Raíz y Resto** se aplican sólo a operandos enteros, siendo su resultado entero.

Si bien la función potencia no está definida en todos los lenguajes de programación, se utilizará en seudocódigo un formato similar al utilizado en lenguaje C: **Potencia (base, exponente)** siendo base y exponente números entero

Ejemplo 7
Dadas las declaraciones:
entero a, b, c, d

real r,e

Son expresiones aritméticas válidas:

- . a+b+5+d resultado entero
- a-b* (r/2) resultado real
- a/b+c-3.6 resultado real
- (a+b)*(c-d)/e resultado real

El resultado de las expresiones es único, dependiendo de los valores asignados a las variables.
Así, para a=10, b=9 y r=6, el resultado de la evaluación de la segunda expresión es –17.0.
Para a=30, b=9 y r=5, el resultado de la evaluación de la segunda expresión es 7.5.

Reglas de evaluación de una expresión aritmética
Son las mismas reglas que se utilizan en matemática. La expresión se comienza a evaluar de izquierda a derecha, si contiene un cálculo entre paréntesis, éste se resuelve primero. Si los paréntesis están anidados, un par de paréntesis dentro de otro, primero se resuelve la expresión del paréntesis interior.
Los operadores de una expresión aritmética tienen un orden de precedencia, esto es, prioridad de ejecución en caso que la expresión contenga distintos operadores sin especificación de paréntesis:

Orden de precedencia	Operador
1	* / div
2	+ -
3	Raíz, Resto, Potencia

Tabla 1.4 Orden de precedencia Operadores Aritméticos

En caso de operadores con la misma prioridad, las operaciones se realizan de izquierda a derecha. Si se desea alterar el orden natural, se deberá utilizar paréntesis.

ACTIVIDAD 3
La expresión matemática $\frac{A+B}{C+D}$ debe escribirse (A+B) / (C+D)
¿Cuál es la expresión matemática que resulta si no se colocan los paréntesis?

Ejemplo 8
En la expresión a-b*(r/2), primero se resuelve el paréntesis, luego el producto y finalmente la resta.
En cambio en la expresión a-b*r/2, se resuelve primero el producto, su resultado se divide en 2 y finalmente se realiza la resta.

En cuanto al resultado de una expresión
Si los operandos de la expresión aritmética son del mismo tipo, el resultado es del mismo tipo.

Si son de distinto tipo, se trabaja a máxima precisión. Así por ejemplo, si un operando es entero y otro real, el resultado es real.

Expresiones relacionales

Las **expresiones relacionales** describen formalmente predicados lógicos que al asignar valores a las variables, se convierten en proposiciones verdaderas o falsas.

Por ejemplo, la expresión "**Alumnos mayores de 18 años**" se puede expresar formalmente utilizando la variable edad como: **edad > 18**.
Si la variable edad toma el valor 20, se puede decir la proposición resultante es verdadera.
Por lo tanto, una expresión relacional es una comparación de dos operandos del mismo tipo, cuyo resultado es un valor lógico.
Los operadores que intervienen en la expresión son los operadores relacionales.
Los operandos deben ser del mismo tipo, por lo tanto, se puede comparar entre sí datos numéricos (constantes, variables o expresiones numéricas), caracteres o cadenas.

Operandos	Operadores	Tipo de Resultado
Datos del mismo tipo: ambos numéricos, carácter, cadena , lógico. Expresiones numéricas	**>** Mayor **<** Menor **==** Igual **<> , !=** Distinto **> =** Mayor o Igual **< =** Menor o Igual	Booleano

Tabla 1.5 Expresiones Relacionales

Ejemplo 9
Son válidas las siguientes expresiones relacionales:

```
entero m,n
caracter t, r
1.  m <= n
2.  (m+n) > (n*2)
3.  r < t
4.  t !='a'
```

Para el lote de prueba: m=-5, n=-17, r= ' d' , t='A' los resultados de las expresiones son:

Expresión 1: Falso

Expresión 2: Verdadero

Expresión 3: Falso

Expresión 4: Verdadero

Se debe recordar que la evaluación de las expresiones 3 y 4 es posible debido al orden de los caracteres en el código ASCII.

Expresiones lógicas

Una **expresión lógica** es la descripción formal de un predicado lógico compuesto con conectores lógicos, que tiene como resultado un valor lógico, verdadero o falso.

En una expresión lógica:

- Los operadores son los conectores lógicos: conjunción (y), disyunción (o) y negación (no).
- Los operandos son predicados lógicos, estos predicados pueden ser: expresiones relacionales, variables o constantes de tipo lógico.
- El resultado es un valor de tipo lógico (verdadero o falso)

Operandos	Operadores	Tipo de Resultado
Expresiones relacionales, variables o constantes de tipo lógico.	**NO** negación **Y** conjunción **O** disyunción	Booleano

Tabla 1.6 Expresiones Lógicas

Para obtener el valor booleano que resulta de la evaluación de una expresión lógica, se utilizan las tablas de verdad de la Lógica de Proposiciones. Éstas se construyen combinando los valores posibles que pueden tomar los operandos A y B:

A	B	A y B	A o B	NO A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Tabla 1.7 Tabla de Verdad

Como se puede inferir de la Tabla de verdad anterior:
La conjunción de dos operandos es verdadera únicamente en el caso en que ambos operandos lo sean.
La disyunción entre dos operandos es verdadera cuando al menos una de dichos operandos lo es.
La negación transforma un operando verdadero en falso y viceversa.

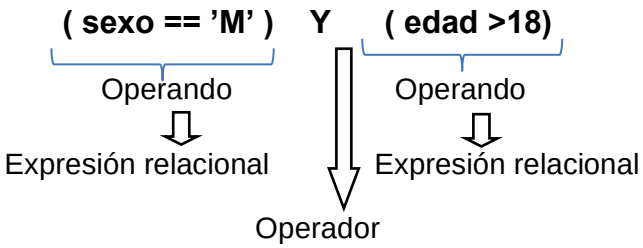
Los operadores lógicos, tienen el siguiente orden de precedencia.

Orden de Precedencia	Operador
1	NO
2	Y
3	O

Tabla 1.8 Orden de precedencia Operadores Lógicos

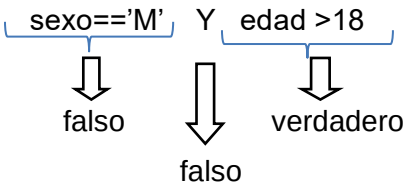
Ejemplo 10

1) Si una narrativa contiene el enunciado: **"Alumnos varones mayores de 18 años".** En lenguaje natural ésta es una forma abreviada para indicar que "los alumnos son varones y los alumnos son mayores de 18 años". En el caso de la lógica es necesario especificar en forma separada ambas propiedades como se muestra a continuación.
Si se considera, las variables **edad** de tipo entero y **sexo** de tipo carácter, se puede formalizar el enunciado anterior con la siguiente expresión lógica:



Esta expresión lógica está compuesta por dos expresiones relacionales conectadas con el operador lógico *Y.*
Si las variables toman los valores edad=20 y sexo='F'

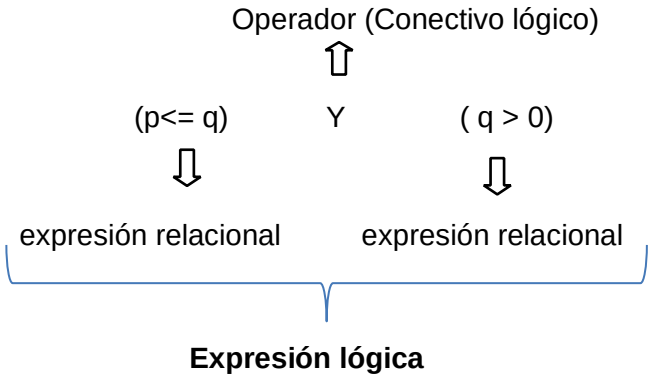
La evaluación de la expresión lógica es falsa, como se muestra a continuación:



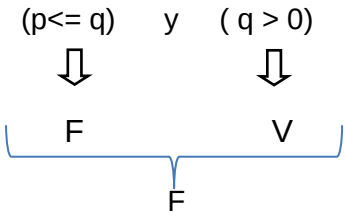
2) Son expresiones lógicas válidas para la siguiente declaración de variables y lote de prueba,

entero p, q
caracter c,d,e
booleano r, a

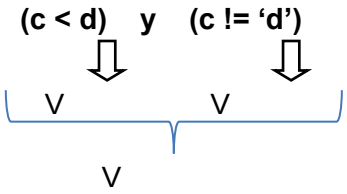
Lote de prueba p=12 ; q=3 ; c= 'b' ; d='m' ; a=Falso ; r=Verdadero



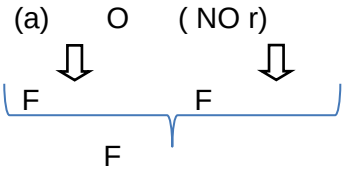
a) El resultado de la expresión para los valores del lote de prueba es **Falso**



b) El resultado de la expresión para los valores del lote es **verdadero**



c) El resultado de la expresión para los valores del lote de prueba es **falso**



Orden de precedencia entre los operadores

La precedencia de los operadores determina el orden en que se evalúan las expresiones complejas.

Precedencia	Operador	Descripción
1	`( )`	Paréntesis
2	`-`	Negación aritmética
3	`^`	Exponenciación
4	`* / %`	Multiplicación, división y módulo
5	`+ -`	Suma y resta
6	`> < >= <= = <>`	Todos los operadores relacionales
7	`no`	Negación lógica
8	`y o`	Conjunción y disyunción

Observación: Cuando los operadores tienen el mismo orden, se resuelves de izquierda a derecha.

Evaluación por cortocircuito de expresiones lógicas

La mayoría de los compiladores utilizan evaluación por cortocircuito, esto significa que detienen la evaluación cuando se tiene certeza del resultado de la expresión.
La evaluación se realiza con los operadores **Y** y **O,** de la siguiente manera:

Si la expresión lógica tiene la forma:

Si ((exp 1) **Y** (exp 2))

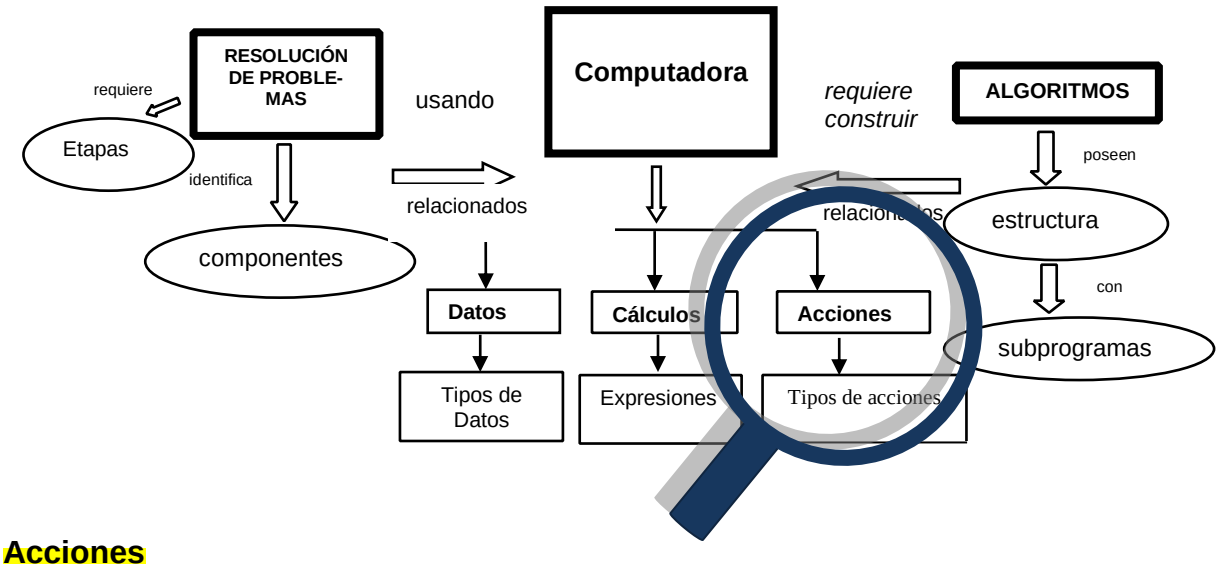
Se evalúa exp2 sólo cuando exp1 es verdadera, ya que si exp1 es falsa el resultado de la expresión lógica es falso independientemente del valor de exp 2.

Si la expresión lógica tiene la forma:

Si ((exp 1) **O** (exp 2))

Si exp1 es verdadera no evalúa exp2, ya que el resultado de la expresión lógica es verdadero, sin importar el valor de exp2.

Hasta ahora se analizó el eje referido a resolución de problemas y los datos y expresiones que pueden ser manipulados por una computadora, se profundizará ahora en las acciones que pueden ser ejecutadas por ella.



Acciones

Para construir un algoritmo son necesarias dos tareas:

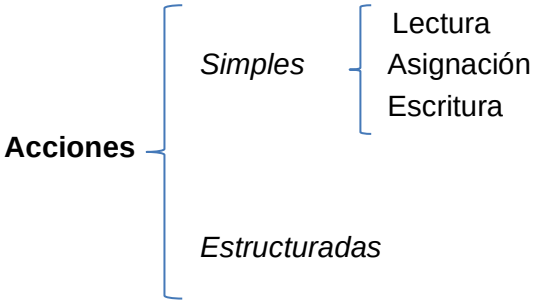
- **Definir el Ambiente**: Elegir la representación adecuada para que los datos puedan ser manipulados por la computadora.
- **Determinar el conjunto de acciones**, cuya ejecución conducirá a la solución del problema planteado.

Para determinar el ambiente, se han estudiado los tipos de datos y las operaciones permitidas para cada uno de ellos. Se necesita conocer ahora las acciones que puede ejecutar una computadora.

Se llama **acción** a cada uno de los pasos que la computadora ejecuta para resolver el problema. La acción tiene la característica de modificar los datos que intervienen en el algoritmo.

Una **acción simple** es un paso que es entendido en forma inmediata por el procesador y no puede ser descompuesto en otros pasos más sencillos.

Una **acción estructurada** es un paso compuesto por otras acciones simples.



Antes de comenzar este estudio, se indicará la convención adoptada para especificar los formatos de las acciones y otros elementos que constituyen el seudocódigo, para evitar ambigüedades en su interpretación.

- Los corchetes angulares **<>** indican que la expresión que encierran, debe ser sustituida a elección del programador.
- Los términos no encerrados entre corchetes angulares, deben ser escritos tal cual se muestran en el formato.
- Los símbolos /*…. */, indican que la expresión que encierran es un comentario que se utilizará para realizar las aclaraciones que se consideren pertinentes en el algoritmo.

A continuación, se profundizará en cada una de las acciones simples que puede ejecutar una computadora.

Acciones Simples

Acción de Lectura

La **acción de lectura** permite almacenar en una variable, un valor especificado desde un dispositivo externo, como el teclado, lector de código de barras, sensor de huellas digitales, etc. Es decir, que la lectura permite introducir al ambiente valores desde el mundo exterior.

El formato de esta acción es:

Leer <nombre de variable, ... , ... >

Donde **<nombre de variable>** representa uno o más nombres de variables del ambiente.
En caso de varias variables, sus identificadores se separan con coma.

Ejemplo 11

```
entero edad          /* Ambiente del algoritmo */
```

Leer edad /* acción de lectura */

La acción **Leer edad**, significa que la computadora tiene que leer lo que el usuario escribe desde un dispositivo de entrada, como por ejemplo el teclado, y almacenarlo en la variable edad.

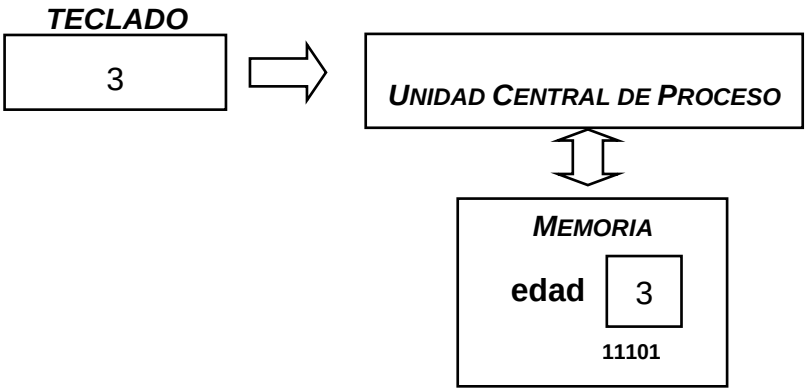


Figura 1.11 Proceso de ejecución de la acción de lectura de una variable

Los pasos que realiza una computadora para ejecutar la acción Leer son:

1. Obtener un valor a través de un dispositivo de entrada, por ejemplo el teclado.
2. Una vez que la UCP tiene el valor, lo almacena en la dirección de memoria referenciada por el <nombre_de_variable> correspondiente.

Ejemplo 12
Dada la declaración de variables:

```
/* Ambiente del algoritmo */
        cadena nomb
        entero edad
        real x,y,z
        carácter result

/*  acciones de lectura */
        Leer  nomb
        Leer  edad, result
        Leer  x,y,z
```

Estas acciones permiten que las variables nombradas se instancien, es decir almacenen los valores ingresados por teclado en sus respectivos espacios de memoria.

Acción de Asignación

La acción de asignación permite almacenar en una variable un valor que proviene de una constante, de otra variable o del resultado de una expresión.

El formato de esta acción es:

<nombre de variable> = <valor>

<nombre de variable> = <expresión>

Donde:

<nombre de variable> es el nombre de la variable a la que el procesador le asignará un valor.

= representa el símbolo de asignación.

<expresión> puede ser una constante, una variable o una expresión.

Una asignación modifica el valor almacenado en la variable que aparece a la izquierda de la asignación, sin producir ningún efecto en la expresión que se encuentra a la derecha de la misma.

Ejemplo 13

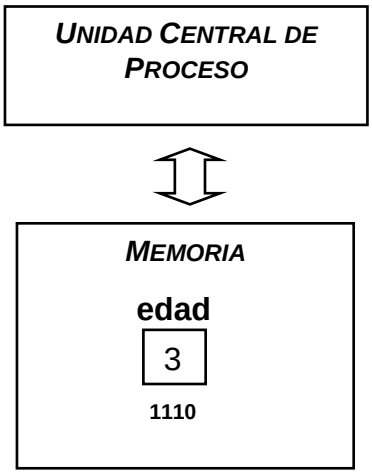
entero a, b, r;
r= 12
r= b
r= a + b 4*

Pasos que realiza una computadora para ejecutar la acción
<nombre_de_variable> = <valor>

1. Obtener el <valor> especificado, en caso de ser el contenido de otra variable, la UCP va a su dirección de memoria y lo copia.
2. Una vez que la UCP tiene el <valor> va a la dirección de memoria vinculado al <nombre de variable> y lo guarda en esa variable.

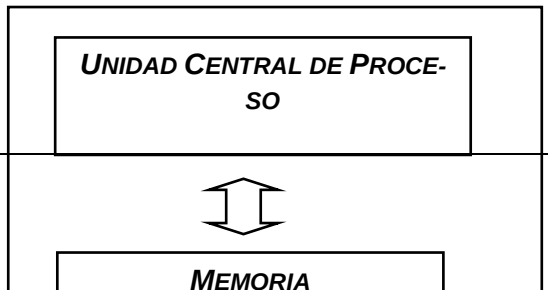
Ejemplo 14

```
entero edad          /*Declaración de variables*/
edad = 3             /* asignación*/
```



Ejemplo 15

```
/*Declaración de variables*/
entero edad
entero x
leer x          /* lectura*/
edad=x;     /* asignación*/
```



NOTA: Existen, por tanto, **dos formas de almacenar un valor en una variable,** a través de una acción de lectura desde dispositivo externo, o por medio de una asignación desde el mismo programa.

Acción de Escritura

La acción de **Escritura**, permite comunicar al exterior valores almacenados en la memoria de la computadora, a través de un dispositivo periférico tal como pantalla o impresora.

El formato de esta acción es:

Escribir <argumento 1, argumento 2, ..., argumento n>

Donde:

<argumento 1, argumento 2, ..., argumento n>, pueden ser: constantes o identificadores de variables, expresiones aritméticas, relacionales o lógicas.

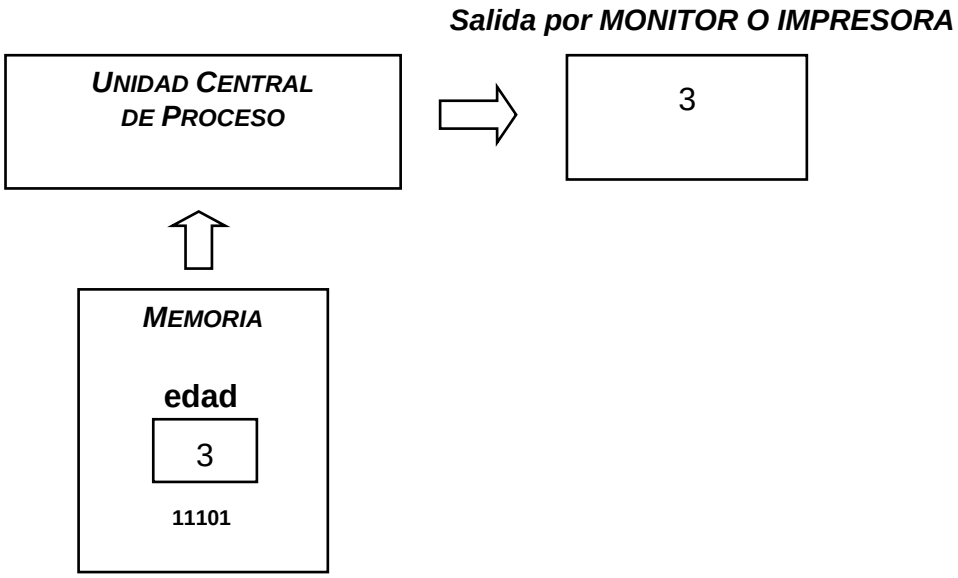
En este caso en el dispositivo externo se muestra el resultado de la expresión.

Los pasos que realiza una computadora para ejecutar la acción Escribir <argumento> son:

1. Obtener el argumento, en caso de ser el contenido de otra variable, la UCP va a su dirección de memoria y lo copia, si es una expresión obtiene su resultado.
2. Muestra el argumento en el dispositivo externo, por ejemplo una pantalla.

Ejemplo 16

```
entero edad              /*Declaración de variables*/
Escribir  edad           /* acción de escritura*/
```



Ejemplo 17

Las siguientes son distintas formas de utilizar la acción de escritura:

Escribir nomb

Escribir " Los lados del triangulo rectángulo son ", x,y,z

Escribir " Temperatura promedio ", suma/ cant

Se han analizado todos los contenidos especificados en el esquema conceptual. Ahora con el conocimiento adquirido hasta el momento, se puede abordar nuevamente, con más precisión, el eje construcción de algoritmos**.**

1.3 Resolución de problemas.

1.3.1 Construcción de algoritmos simples

A partir de las etapas para la resolución de problemas usando como herramienta la computadora, se profundizará en las dos primeras tapas.

Etapas resolución de problemas utilizando computadoras	Objetivo
Análisis del problema	Ayuda a la comprensión del problema, a comprender **¿qué se debe hacer?.** Se lo denomina también especificación del problema
Diseño del Algoritmo	Permiten explorar soluciones y seleccionar la que se considere más adecuada Se determina **¿cómo se debe hacer?** Permite indicar cómo el algoritmo realizará la tarea
Codificación: Construcción del programa	Se traduce el algoritmo a un lenguaje de programación para que pueda ser interpretado y ejecutado por una computadora.
Compilación y ejecución	
Validación	Permite comprobar la solución obtenida
Documentación	Se registra lo que se realiza durante todo el proceso, y en cada una de las distintas etapas.

Análisis del problema

El primer paso para el resolver un problema es comprenderlo. Comprender un problema implica identificar claramente los resultados llamados *datos de salida*, y los datos con que se cuenta para obtener esos resultados, llamados *datos de entrada*.

En un problema se pueden distinguir tres componentes: *los datos (entrada), el o los resultados (salida),* y *un conjunto de reglas o restricciones* que vinculan a los datos con los resultados a obtener. El análisis de un problema comienza identificando estas componentes.

Se denomina **datos de entrada** a aquellos que son conocidos y deben ser proporcionados desde un dispositivo externo tal como el teclado, lector de barra, etc..

Se denomina **datos de salida** a los resultados que debe mostrar al usuario, según lo requerido por el problema.

Algunas preguntas básicas que debe responder quien intenta resolver un problema son:

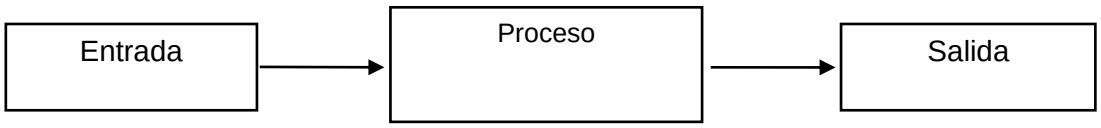
¿Qué se debe obtener?

¿Con que datos se cuenta?

¿Existen Restricciones'?

El proceso que permite determinar en forma clara y precisa el objetivo que se persigue, es decir los requisitos **del cliente,** se llama **especificación del problema.**

Un problema está definido por un estado inicial o de partida. Resolver un problema requiere transformar mediante un proceso, los datos de entrada para obtener los datos de salida.



Para representar la solución del problema se puede utilizar alguna descripción gráfica o representación simbólica, que ayude a sistematizar la información.

En síntesis, en la etapa de análisis se determina ¿**Qué Hacer?,** lo que se especifica al responder las preguntas: **¿Qué salidas se deben generar?, ¿Con qué datos de entrada se cuentan para ello?**

Esquemáticamente:

Análisis *¿Qué hacer?*	Especificación de Salidas
	Especificación de Entradas

En general, interpretar los enunciados de los problemas para reconocer sus componentes, es el mayor obstáculo con que se enfrenta el programador al momento de encontrar el algoritmo de resolución de un problema.

Una estrategia para interpretar enunciados de problemas, podría ser[2]*:*

- *Leer con detenimiento TODO el enunciado.*
- *Comprender claramente el significado de cada palabra y cada frase.*
- *Poner especial atención a los signos de puntuación.*
- *Identificar la salida o resultados del problema.*
- *Identificar datos explícitos (pueden ser relevantes o irrelevantes).*
- *Identificar datos implícitos (pueden ser relevantes o irrelevantes) y hacerlos explícitos.*
- *Detectar imprecisiones o ambigüedades antes de avanzar con la solución.*

A continuación, se verá en qué consiste cada uno de estos pasos. Para identificar las componentes se recomienda en un principio **la lectura del enunciado** del problema como un todo, sin detenerse demasiado en los detalles. Una vez que se tiene una idea global del problema, se realiza una segunda lectura determinando las partes principales del mismo, para evaluar qué se pide y con qué datos se cuenta.

[2] Adecuación estrategia propuesta por Sonia V. Rueda Alejandro J. García, Programa de Ingreso 2003. Análisis y Comprensión de Problemas. Fundamentos, Problemas Resueltos y Problemas Propuestos. UNS. Noviembre 2002

La **primera lectura** permite acercarnos al problema y conocer el contexto donde se plantea: lugar, actores, acciones. Ejemplo: Una fábrica de electrodomésticos, un supermercado posee información, una empresa de colectivos de larga distancia, entre otros.

Una **segunda lectura** permite detectar qué datos son "relevantes" para procesar. La importancia de los datos está en función de las especificaciones del enunciado.

Es necesario poner especial atención a los **signos de puntuación**, ya que de ellos depende el significado de cada frase. Y sobre todo, destacar que un cambio sintáctico leve, puede provocar una variación semántica fundamental. Por ejemplo "la mitad de: 20 más 4" respecto a " la mitad de 20, más 4".

Se ha hablado de datos **relevantes** y datos **explícitos,** ¿Qué significan estos conceptos?

A veces, se tiene la sensación de que el enunciado no brinda suficientes datos. La clave, está en obtener información útil a partir de ciertos datos que pueden estar "ocultos". Otras veces, los problemas proveen muchos datos y es importante distinguir los datos relevantes de aquellos que no lo son.

Los datos pueden estar expresados en forma explícita o implícita, y a su vez ser relevantes o no para un contexto determinado.

Un dato está expresado en forma explícita, cuando es claramente identificable en el contexto que se usa. En cambio está implícito cuando es parte del problema aunque no se lo exprese directamente, surgen de inferencias a partir de otros datos detectados y se los transforma en explícitos

1.3.2 Construcción de Algoritmos que utilizan subprogramas

Diseño: Técnica Divide y Vencerás

Una Técnica de Diseño de Algoritmos muy usada para resolver problemas es **Divide y Vencerás**.
En la cultura popular, **divide y vencerás** hace referencia a un refrán que implica resolver un problema difícil, dividiéndolo en partes más simples tantas veces como sea necesario, hasta que la resolución de las partes se torna obvia. La solución del problema principal se construye con las soluciones encontradas.
En las ciencias de la computación, el término divide y vencerás (DYV) hace referencia a uno de los más importantes paradigmas de diseño algorítmico. El método está basado en la resolución de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original.
Esta técnica que reduce un problema complejo en otros más sencillos, llamados subproblemas, facilitan su comprensión y el abordaje de su solución.

Nota: Las soluciones a los subproblemas identificados con la técnica Divide y Vencerás, pueden escribirse como "subprogramas". Luego en un Algoritmo Principal con ellos representar la solución general.

Ejemplo 18

La mutual de los empleados de la Universidad de San Juan, ha elaborado un proyecto, a llevarse a cabo en un plazo de 6 meses, con el objeto de realizar algunas refacciones en su predio ubicado en el departamento de Pocito.

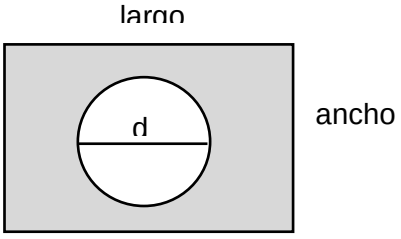
En un terreno rectangular cuyas dimensiones se conocen, está construida una piscina circular de diámetro igual a la mitad del largo del terreno.

Debido a algunos accidentes ocurridos en las últimas temporadas de verano, se desea cubrir el fondo con un material especial que evite dichas contingencias. Con el mismo objetivo se colocará una hilera de mosaico antideslizante en el contorno de la pileta.

Por otra parte, una vez finalizada la obra, se quiere parquizar nuevamente los alrededores de la piscina.

Se ha solicitado al departamento construcciones de la universidad, un informe que especifique la superficie a cubrir con cada uno de los materiales mencionados, con el objeto de realizar la licitación para la compra de los mismos.

Representación gráfica del problema:



donde d = largo/2

Estrategia de Aplicación

A partir de las preguntas básicas, se obtendrá información estratégica, según se detalla:

- **Paso 1**: Primero se responden las tres preguntas básicas de la etapa del análisis de un problema
    - *¿Qué se debe obtener?* Al responder esta pregunta se identifican los datos de salida y las tareas asociadas para obtenerlos.
    - *¿Con que datos se cuenta?* Con esta pregunta se advierten los datos de entrada y a partir de la cantidad de veces que ellos se ingresan, se determina una estructura de control adecuada.
    - *¿Existen restricciones?* Las restricciones se reflejan en estructuras de control que ajustan las entradas, el proceso y las salidas.

De este modo se interpreta el problema en general, aplicando la técnica de DYV se identifican sus partes (subproblemas) y para cada uno (si es necesario) se vuelve a repetir el paso 1.

- **Paso 2:** Identificados los subproblemas, para cada uno se busca una solución que debe tener especificado con claridad su **descripción**: un **nombre de solución** (luego será un nombre de subprograma), **que hace**, que datos recibe (**entradas**) y que resultados produce (**salidas**) tiene.
- **Paso 3:** elabora **algoritmo principal** que contenga un esquema de proceso, el cual contiene **nombre de solución, entradas y salidas.**
- **Paso 4:** A partir de lo que hace la solución, se escribe/n el/los subprograma/s correspondiente/s en seudocódigo y también se elabora el algoritmo principal., de esta forma queda escrito formalmente el algoritmo.

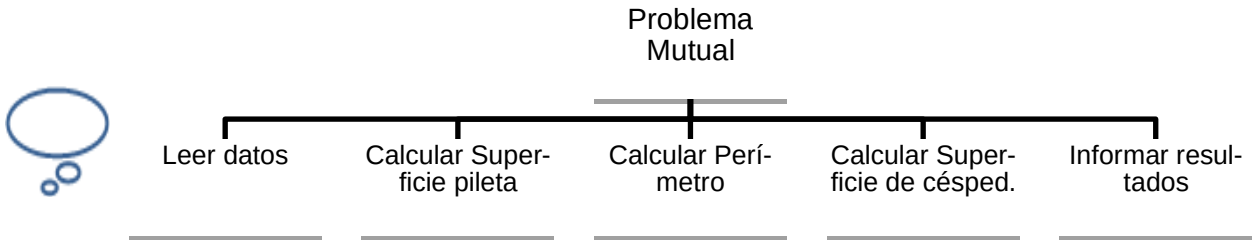
El análisis del enunciado es el siguiente:

Paso 1: Primero se responden las tres preguntas básicas de la etapa del análisis de un problema. Se interpreta el problema en general, aplicando la técnica de DYV se identifican sus partes (subproblemas) y para cada uno (si es necesario) se vuelve a repetir el paso 1.

¿Qué se debe obtener? (datos de salida)
¿Con que datos se cuenta? (datos de entradas)
¿Existen restricciones? (procesos identificados)

Datos de salida	Datos de Entrada	Proceso
1.Superficie del fondo de la pileta 2.Perímetro de la pileta 3.Superficie a cubrir de césped (superficie sombreada en la figura)	Largo y ancho del terreno	Leer datos Calcular Superficie de la pileta Calcular Perímetro de la pileta Calcular Superficie de césped. Informar resultados

Al aplicar la técnica Divide y Vencerás (DYV), este problema se puede dividir en subproblemas. Los subproblemas se pueden resolver con subprogramas invocados desde el algoritmo principal.



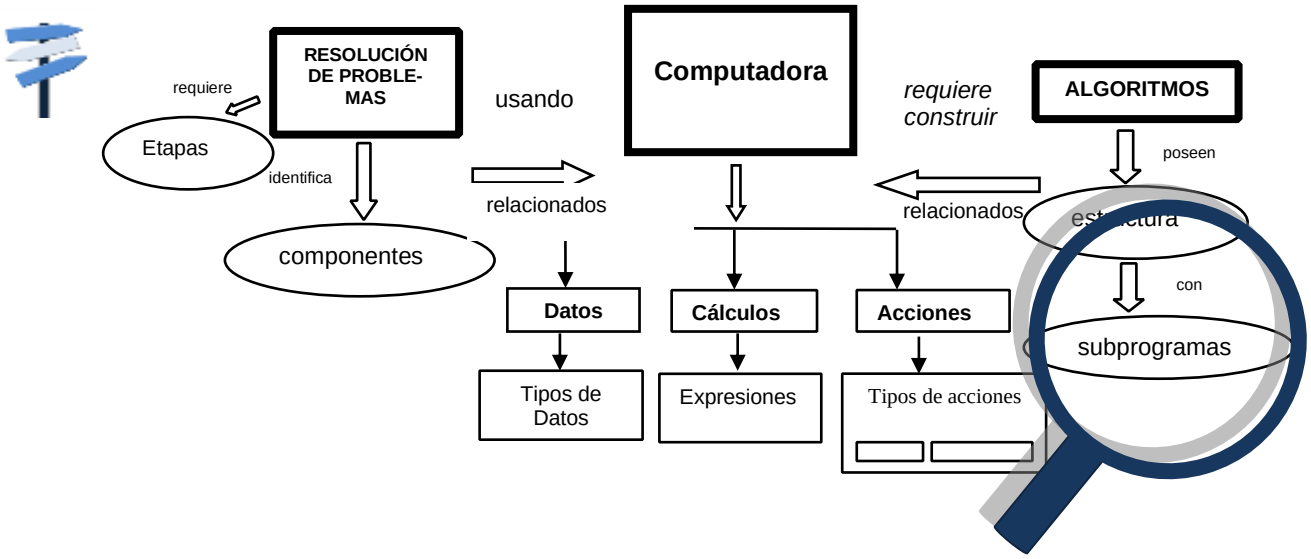
Paso 2: Identificados los subproblemas, para cada uno se busca una solución que debe tener especificado con claridad su **descripción**: un **nombre de solución** (luego será un nombre de subprograma), **qué hace**, qué datos recibe (**entradas**) y qué resultados produce (**salidas**) tiene.

Leer datos
Superficie_Pileta: Calcula la superficie de un círculo. Entrada: Largo y ancho. Salida: Superficie de la pileta.
Perimetro_Pileta: Calcula el Perímetro de un círculo. Entrada: Largo. Salida: Perimetro de la pileta.
Superficie_cesped: Calcular superficie de rectángulo y resta superficie de pileta. Entrada: Largo, ancho y Superficie de la pileta. Salida Superficie calculada.
Mostrar: Informa resultados. Entrada Perímetro de la pileta y Superficie calculada. Salida: Mensaje.

Paso 3: elabora a**lgoritmo principal** que contenga un esquema de proceso, el cual contiene **nombre de solución, entradas y salidas.**
Leer datos
Superficie de la pileta = **Superficie_Pileta** (Largo y ancho)
Perímetro de la pileta = **Perimetro_Pileta** (Largo)
Superficie calculada = **Superficie_cesped** (Largo, ancho y Superficie de la pileta)
Mostrar (Perímetro de la pileta y Superficie calculada)

- **Paso 4:** A partir de lo que hace la solución, se escribe/n el/los subprograma/s correspondiente/s en seudocódigo y también se elabora el algoritmo principal, de esta forma queda escrito formalmente.

Para continuar el paso 4 se debe formalizar algunos aspectos de sintaxis de subprogramas que se verán en desarrollo de la unidad



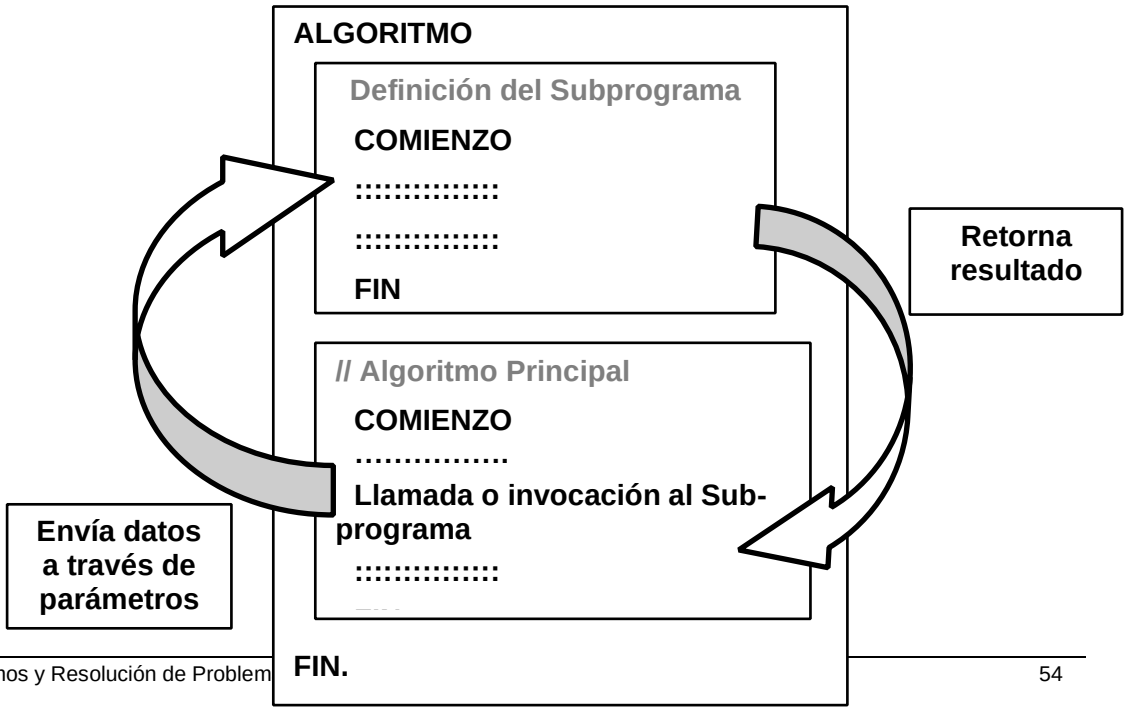
1.4 Estructura de Algoritmos que utilizan subprogramas

En la **etapa de diseño del algoritmo** se debe comenzar construyendo el algoritmo principal donde se colocan las invocaciones de los subprogramas con sus respectivas entradas y salidas.

Finalizado el diseño la estructura de un algoritmo contiene primero la definición de los subprogramas y luego su invocación como parte del algoritmo principal.

En la etapa de compilación y ejecución, al invocar un subprograma desde algún punto del algoritmo principal, se ejecutan las acciones que forman parte de él. Una vez que finaliza la ejecución del subprograma, el control se devuelve al algoritmo principal o al subprograma que lo llamó.

En forma gráfica esto sería:



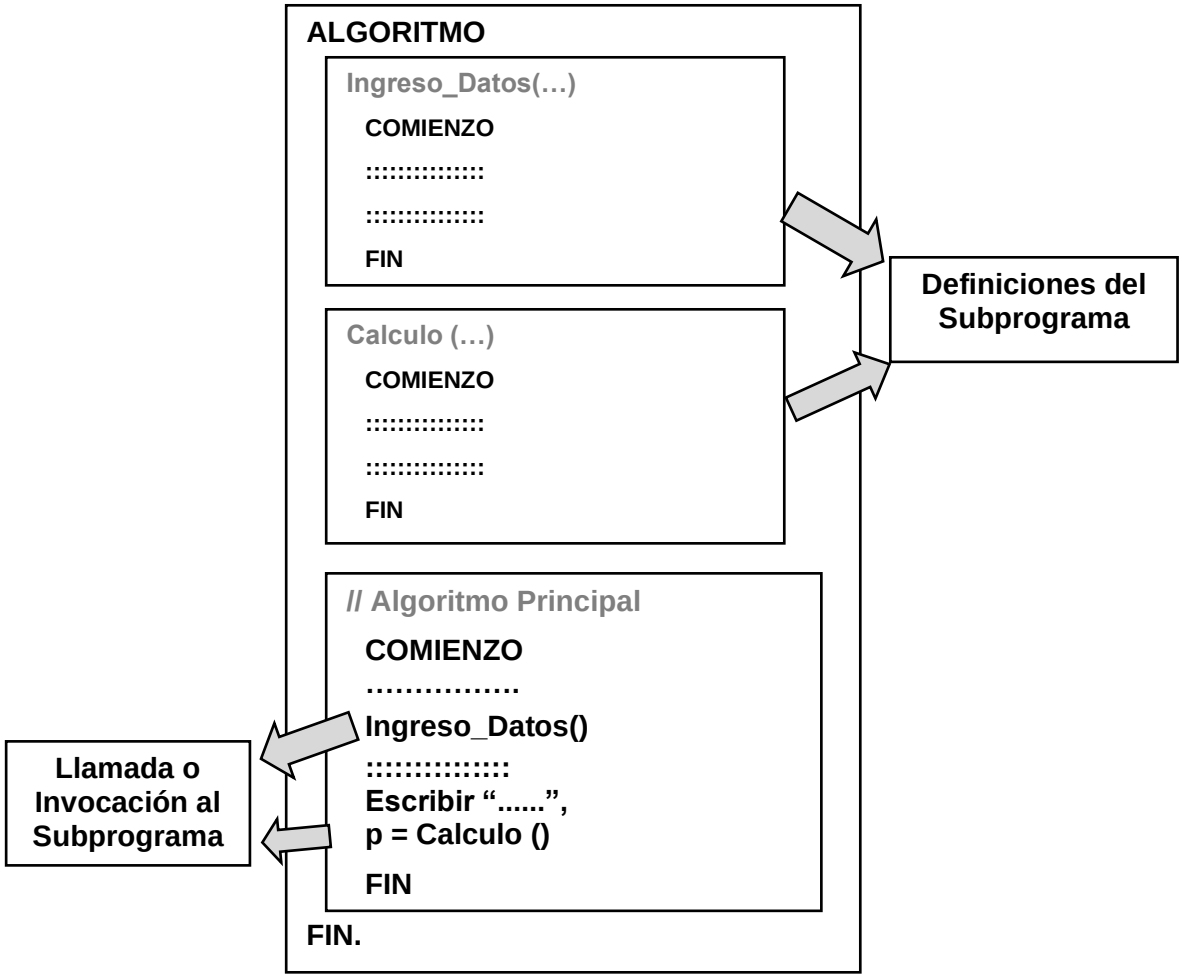


Figura 1.12 Invocación de Subprogramas

Definición e Invocación de subprogramas

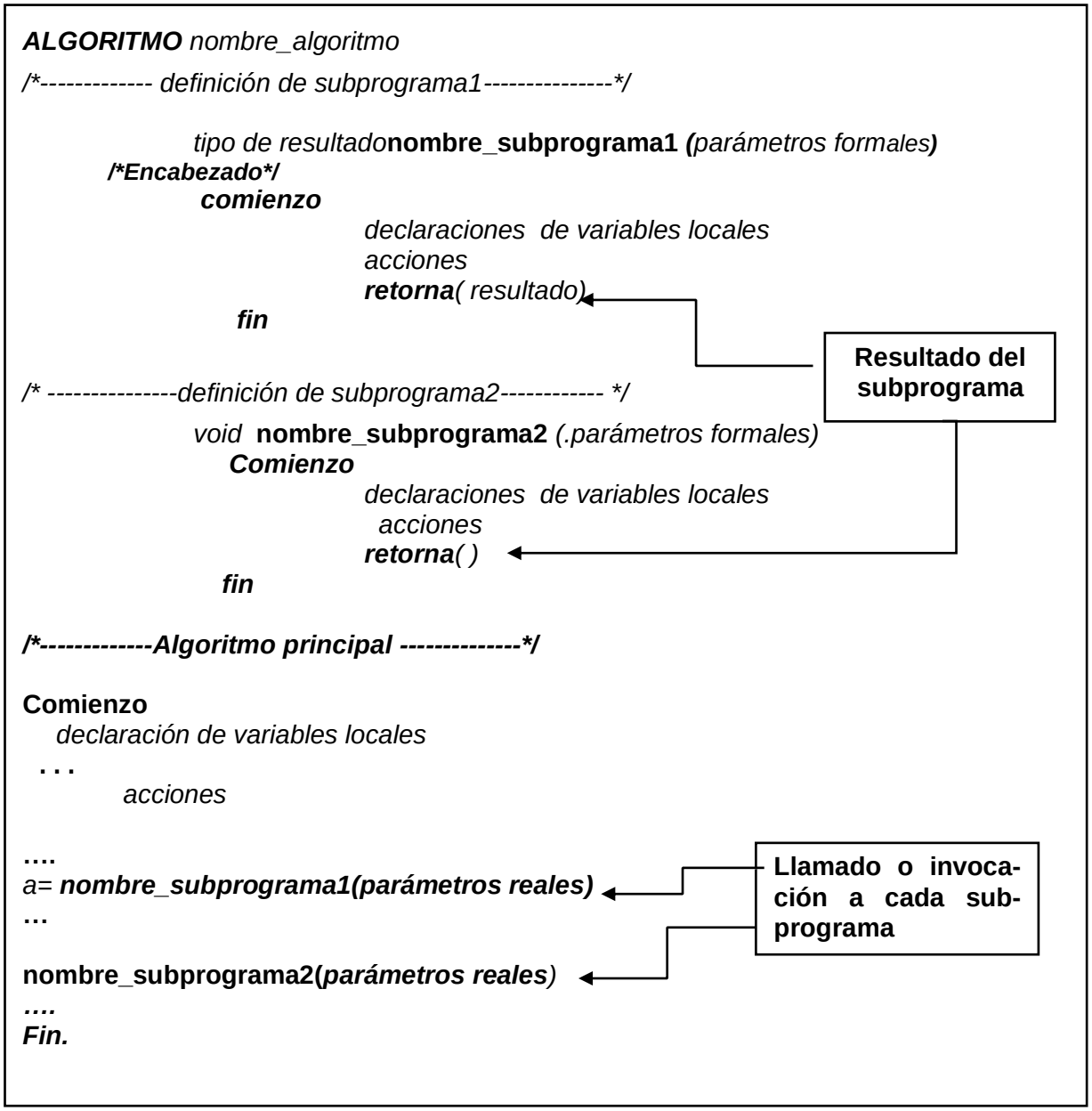
Para utilizar subprogramas en un algoritmo se debe:

Definir el subprograma. El subprograma debe ser definido antes de ser invocado. La definición de un subprograma, al igual que el programa principal, debe incluir el ambiente, donde se declaran las variables, y las acciones que el subprograma necesita para realizar la tarea. De esta manera se puede luego invocar como si fuera un subprograma predefinido por el lenguaje.

Llamar el subprograma: para usar el subprograma se realiza una invocación o llamada al mismo, dentro de las acciones del algoritmo desde otro subprograma que lo quiere usar.

Nota:
La manera de definir e invocar subprogramas depende del lenguaje de programación. En pseudocódigo se hará adecuándose al lenguaje C que se utilizará como lenguaje de referencia.

La estructura formal de un algoritmo que utiliza subprogramas es:



Definición del subprograma

La definición de un subprograma siempre precede al algoritmo principal. De esta forma, al compilar el programa, el subprograma está definido antes de la primera invocación o llamada; esto es, se compila antes de que encuentre en el programa principal una sentencia de invocación o llamada al mismo

La definición de un subprograma consta de dos partes:

- Encabezado: es la primera línea del subprograma, se llama también firma del subprograma
- Cuerpo del subprograma.

Encabezado

Es la *primera línea* de la definición del subprograma, contiene la especificación del tipo del retorno, seguido del nombre o identificador del subprograma, y opcionalmente un conjunto de argumentos o parámetros, separados por comas y encerrados entre paréntesis. Cada argumento debe ir precedido por su declaración de tipo.

Si el subprograma no incluye parámetros, el identificador del subprograma puede ir seguido de un par de paréntesis vacíos o con la palabra reservada ***void***.
Si el subprograma no devuelve resultado, se coloca la palabra reservada **void** (void significa carente de resultado), en caso de devolver un resultado debe especificarse su tipo.

La primera línea, puede indicarse con el siguiente formato:

Formato:

<tipo de retorno><identificador>(< tipo1 arg1, tipo2 arg2, . . ., tiponargn>)

arg1, arg2, …, argn, se denominan **argumentos formales** o **parámetros formales,** identifican a los datos que recibe el subprograma en el momento de su invocación o llamada.
tipo1, tipo2, …., tipon, representan los tipos de datos asociados a cada parámetro formal.

Al momento de invocar el subprograma, los datos que se envían al subprograma reciben el nombre de **parámetros actuales** o **parámetros reales**, ya que se refieren a la información que realmente se transfiere. Cabe aclarar que los nombres de los parámetros formales y los nombres de los parámetros reales no necesariamente deben coincidir.

Cuerpo del subprograma

El algoritmo del subprograma estará delimitado por las palabras ***Comienzo y Fin*** y constituido por dos partes bien diferenciadas, el **ambiente**, en donde se declaran todas las variables que se necesitan para realizar las tareas, y las **acciones** propiamente dichas.

Los parámetros formales y las variables declaradas en un subprograma, son **variables locales** al subprograma, pues no son reconocidos fuera de él.

Cuando se usan subprogramas, es importante tener en cuenta que:

- los parámetros reales deben coincidir en tipo, orden y cantidad con los parámetros formales.
- el cuerpo del subprograma debe incluir al menos una acción de retorno que denominaremos ***retorna***, para retornar el control al punto de llamada o invocación
- La acción ***retorna*** permite que se devuelva un resultado, el cual colocaremos entre paréntesis: ***retorna(resultado)***
- Si el subprograma no retorna un resultado, la acción ***retorna*** la escribiremos ***retorna ()***

Ejemplo 19

El siguiente ejemplo corresponde a la definición del subprograma *calcula*, que devuelve el perímetro de un terreno, cuyas dimensiones son recibidas en los parámetros formales.

```
real calcula( real   la, real an)              /*la y an son parámetros formales*/
comienzo
    real perim;          /* perim es una variable local al subprograma calcula*/
    perim = 2 * ( la+ an)
    retorna (perim)
fin
```

Ejemplo 20

El ejemplo muestra el código que corresponde a la definición del subprograma *superficie*, que calcula la superficie de un rectangulo
En este caso, el subprograma recibe como parámetros los valores de largo y ancho, ademas no devuelve un resultado al punto de invocación ya que el mismo se muestra dentro del cuerpo de la función.

```
void superficie( entero  largo, entero ancho )
comienzo
      entero i, sup
      sup= largo*ancho
      escribir" la superficie del rectangulo es ", sup
      retorna()
fin
```

Ejemplo 21

La siguiente definición corresponde al subprograma *cabecera*, que imprime el membrete de una factura. En este caso, no recibe información y no devuelve un resultado.

```
void cabecera (void)
comienzo
    Escribir“ Universidad Nacional de San Juan   ”
    Escribir“  Facultad de CienciasExactas”
    Escribir “       Pocito - San Juan    ”
    retorna()
fin
```

Como puede observarse, se coloca la palabra reservada ***void*** como especificador de tipo del resultado del subprograma y la acción retorna().

En estos ejemplos, al llegar al final del cuerpo del subprograma, se devuelve el control al punto de llamada del mismo, sin retornar resultado alguno.

Invocación o llamada a un Subprograma

La llamada o invocación a un subprograma produce la ejecución de las acciones del cuerpo del mismo, hasta que encuentra la acción *retorna*, que devuelve el control al punto desde donde fue invocado.

Dependiendo si devuelve o no resultados, varía el modo en que se llama o invoca a un subprograma:

Si el subprograma no devuelve resultado

En este caso la invocación se realiza colocando el nombre seguido de la lista de argumentos encerrados entre paréntesis y separados por coma**.**

Formato:

<identificador del subprograma> (<arg1,arg2, . . . ,argn>)

Donde **arg1,arg2, . . ., argn** se denominan **parámetros reales o actuales**.

Estos argumentos deben corresponderse con los parámetros formales que aparecen en la primera línea de la definición del subprograma. Si la llamada del subprograma no necesita argumentos, a continuación del identificador del subprograma se coloca un par de paréntesis vacíos.

```
void  nombre_subprograma (.parámetros formales)
   Comienzo
        declaraciones  de variables locales
          acciones
        retorna( )
   fin
```

Por ejemplo, las invocaciones de los ejemplos 3 y 4 son respectivamente:

Superficie (l,a)

Cabecera ()

Si el subprograma devuelve un resultado

En este caso, la invocación puede formar parte de una acción de escritura, de una asignación, o ser un operando en una expresión.

- ***Como parte de la acción escribir:***

Escribir <identificador del subprograma> (<arg1,arg2, . . . ,argn>)
Por ejemplo:
Escribir **superficie (largo, ancho)**

- ***En una asignación:***

<identificadorVariable> = <identificador del subprograma> (<arg1,arg2, . . . ,argn>)

Por ejemplo

r=**superficie (largo, ancho)**

- ***Como parte de una expresión***

Por ejemplo

f= **superficie (largo, ancho)** + 5

Ejemplo 22

El siguiente algoritmo invoca un subprograma superficie y este le retorna el valor de la superficie de un rectángulo.

Algoritmo numero
entero **superficie**(entero largo, entero ancho)
comienzo
entero i, sup
sup= largo*ancho
retorna(sup)
fin

Comienzo /*----Algoritmo principal----*/
entero l, a
leer (l,a)
escribir" la superficie del rectángulo es ", superficie (a,l)
Fin

Ejemplo 23

En un terreno (que es rectangular) se quiere construir una pileta cuadrada. Se quiere saber cuántos metros de césped quedaran en el mismo una vez que se haya construido la pileta
El siguiente subprograma

Algoritmo numero
entero **superficie**(entero largo, entero ancho)
comienzo
entero i, sup
sup= largo*ancho
retorna(sup)
fin

Comienzo /*----Algoritmo principal----*/
entero lt, at, lp, t, p
Escribir "Ingrese largo y ancho del terreno"
leer (lt,at)
t = superficie (lt,at)
Escribir "Ingrese lado de la pileta"
leer (lp)
p= superficie (lp,lp)
escribir" la superficie del césped es ", t-p

Fin

Nota: ¿Podría evitarse el uso de las variables t y p? ¿Cómo sería el código?

ACTIVIDAD 4
Construya el algoritmo principal y los subprogramas correspondientes al Ejemplo 18

1.5 Seguimientos de algoritmos

Una vez que se alcanza un resultado, es fundamental confrontarlo especificaciones y restricciones del problema y verificar que efectivamente constituye una solución. Para ello, nuevamente se deberá leer la especificación original del problema y asegurarse que todas las cuestiones planteadas se han resuelto consistentemente.

La *traza de un Algoritmo* se puede definir *como la ejecución manual de las acciones que lo constituyen. Consiste en seguir paso a paso las instrucciones del Algoritmo con datos concretos de manera de comprobar si la solución adoptada en la fase de diseño resuelve el problema.*

La traza permite comprobar que dicho algoritmo funciona correctamente, detectar errores o cuando se crea conveniente simplificarlo para incrementar su eficacia y velocidad.

El seguimiento se puede realizar utilizando una tabla en la que se muestra los valores que toman las distintas variables declaradas, a medida que se ejecutan las acciones.

Seguimientos de algoritmos simples sin subprogramas

Ejemplo 24

El siguiente ejemplo que usa subprogramas, se muestra cómo realizar el seguimiento del algoritmo Calculo1 correspondiente, tomando para ello el lote de prueba: largo= 20 m y ancho=16 m.

Algoritmo **Calculo1**
Comienzo
real ancho, largo, radio, supf
1. Leer ancho, largo
2. radio= largo/4
3. supf= 3.14 * radio * radio
4. Escribir “Superficie del fondo de la pileta ”, supf
5. Escribir “ Longitud a cubrir con cerámico”, 2 * 3.14 * radio
6. Escribir “Superficie a cubrir con césped”, largo * ancho - supf

Fin

Seguimiento algoritmo Calculo1

Acción		ancho	largo	radio	supf	Salida
1.		16	20			
2.				5		
3.					78.5	
4.						Superficie del fondo de la pileta 78. 5
5.						Longitud a cubrir con cerámico 31.4

6.						Superficie a cubrir con césped 241.5

Observación:
La tabla de seguimiento muestra el valor que cada variable tiene en memoria, este valor va cambiando a medida que se ejecutan acciones. En tanto que la columna Salida muestra lo que indican las acciones de escritura.
La traza debe realizarse con distintos lotes de prueba que contemplen todos los requisitos y restricciones impuestas en el problema. Esto hace más confiable el algoritmo, aunque nunca se puede tener una confiabilidad total, con este método de validación por prueba.

Aclaración:
La columna Acción que se observa en la gráfica de la tabla de seguimiento, es colocada con el propósito de hacer más sencilla su interpretación en este primer acercamiento a la temática, pero no forma parte de misma, como tampoco forma parte del algoritmo el número que se coloca en cada línea.

Seguimientos de algoritmos con subprogramas

Para realizar el seguimiento de un algoritmo que utilice subprogramas, se debe representar las variables y los parámetros en cuadros diferentes según sean del algoritmo principal o de los subprogramas .Esto es debido a que cuando se ejecute el algoritmo, el programa principal y cada subprograma se almacenan en distintos lugares de memoria.

Ejemplo 25
Realizar el seguimiento del siguiente algoritmo para el lote de prueba: 10, 5

Algoritmo rectangulo1

```
real calcula( real   la, real an)              /*la y an son parámetros formales*/
Comienzo
  real perim;   /* perim es una variable local del subprograma perímetro */
       perim = 2 * ( la + an)
       retorna(perim)
Fin

Comienzo   /*----Algoritmo principal----*/
   real largo, ancho
      Leer largo
      Leer ancho
      Escribir  "El perímetro del rectángulo es", calcula (largo, ancho)
Fin
```

Algoritmo principal		
Largo	ancho	**Salida**
10	5	El perímetro del rectángulo es 30

Subprograma Calcula			
la	an	perim	**Salida**
10	5	30	

Ejemplo 26
Realizar el seguimiento del siguiente algoritmo para el siguiente lote de prueba: 10, 5.
Observe que en este ejemplo, la acción de escritura de la salida del resultado se realiza en el mismo subprograma.

Algoritmo rectangulo2

```
        Void calcula( real   la, real an)   /*la y an son parámetros formales*/
        Comienzo
          real perim;  /* perim es una variable local a el subprograma perímetro */
                perim = 2 * ( la + an)
                Escribir "El perímetro del rectángulo es", perim
                retorna ()
        Fin
Comienzo      /*----Algoritmo principal----*/
          real largo, ancho
            Leer largo
            Leer ancho
            calcula (largo, ancho)
        Fin
```

Algoritmo principal		
Largo	ancho	**Salida**
10	5	

Subprograma Calcula			
la	an	perim	**Salida**
10	5	30	El perímetro del rectángulo es 30

ACTIVIDAD 5

1- Identificar en el algoritmo elaborado los parámetros formales con color rojo, los parámetros reales con color verde, con color negro las variables locales de cada subprograma.

2- Realizar el seguimiento del algoritmo elaborado en el Ejemplo 25 con el siguiente lote de prueba *largo del terrero: 20 metros y ancho del terreno: 10 metros.*

3- Analizar los ejemplos de seguimiento de los Ejemplos 25 y 26 respectivamente, descubrir la diferencia entre ambos algoritmos y extraer conclusiones en cuanto a la forma de representar el seguimiento en la tabla.

4- Realizar un esquema conceptual con los principales conceptos de la unidad: Problema, Subproblema, subprograma, algoritmo, parámetros formales, parámetros reales, variables locales, retorna, void.

1.6 Bibliografía

- Casanova Faus, Asunción Programación Departamento de Sistemas Informáticos y Computación- Escuela universitaria de Informática, Universidad Politécnica de Valencia .Editor. servicio de Publicaciones ISBN 84-7721-233-3 1993
- Criado Clavero, Mª Asunción.(2006) "Programación en Lenguajes Estructurados" Alfaomega.Ra-Ma
- De Giusti, Armando E., Madoz Maria y otros. Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci. LIDI. Facultad de Informática Universidad Nacional de la Plata.
- Polya George (1972) Cómo plantear y resolver problemas. México. Editorial Trillas
- Brassard, G. & Bratley, P. "Fundamentos de Algoritmia". Prentice-Hall. 1997.

- Sonia V. Rueda Alejandro J. García, Programa de Ingreso 2003. Análisis y Comprensión de Problemas. Fundamentos, Problemas Resueltos y Problemas Propuestos. Departamento de Ciencias e Ingeniería de la Computación. Universidad Nacional del Sur. Noviembre de 2002.

Práctico Unidad 1: Datos Simples

Resultados de Aprendizaje:

- Identificar datos y tipos de datos.
- Construir expresiones.
- Usar en forma adecuada acciones simples.
- Identificar las partes de un subprograma.
- Identificar entradas y la salida de un subprograma.
- Reconocer las distintas formas de invocar al subprograma.
- Capacidad para diseñar y construir subprogramas y algoritmos correctos.
- Resolver situaciones problemáticas con iniciativa, autonomía y creatividad.
- Adquirir responsabilidad y compromiso por su propio aprendizaje.
- Desarrollar capacidad para comunicar sus puntos de vista en forma oral y escrita.
- Participar en grupos de trabajo, respetando las ideas de sus compañeros.

Ejercicio 1

Definir como datos la siguiente información y clasificarlos como variables o constantes.
Ejemplo: el sueldo de un profesor universitario es una ***variable***
la cantidad de asientos del avión Boeing 747 de American Airlines es una ***constante***

1. La edad de una persona en años.
2. La capital de una provincia.
3. El río más largo del mundo.
4. La distancia entre dos planetas.
5. La cantidad de calles de una ciudad.
6. El saldo de una cuenta bancaria.
7. La quinta letra del alfabeto griego.
8. El precio de un par de zapatillas.
9. La cantidad de pasajeros de un colectivo de larga distancia.
10. El estado de una compuerta de regadío.
11. 530 en números romanos.
12. El número de patas de una araña.
13. El color de un semáforo (rojo, amarillo o verde).
14. La cantidad de ventanas de un edificio.
15. La temperatura de una habitación.

Ejercicio 2

Realice un listado indicando válidos y los que no los son, justificando en cada caso:

a) con los identificadores

1)ID.entificador
2)2019_ejercicios
3) _primero1
4) dato.color
5)**expresiones**
6)cui-da-do
7) ¡Recreo!
8) _*A*_
9)Practico_seudocódigo
10) VaLiDo
11) _P12019
12) Ejercicio-Práctico-N1

Válidos	Inválidos	Justificación
	#lista-blanca2019	NO válido porque comienza con un carácter especial
listablanca2019		Válido cumple con las condiciones de un identificador

Indique porque considera importante el nombre asignado a un identificador.

b) con las definiciones de datos

1) dato.color =”azul”
2) azul = 29
3) caracter caracter1
4) caracter1 = A
5) caracter1 = ‘A’
6) _A_ = 23 * 10 / azul
7) real IDentificador
8) entero expresiones
9) expresiones = 25.8
10) IDentificador = expresiones + 17.43
11) cadena _Pr12019
12) VaLiDo = IDentificador

Válidos	Inválidos	Justificación
	lista-blanca2019 =’678’	NO válido porque es un dato que almacena cadenas y debería llevar comillas dobles
listablanca2019 =678		Válido porque es una constante que almacena un valor entero.

Ejercicio 3

Escribir las siguientes expresiones aritméticas para que sean válidas en pseudocódigo:

A.	$15t^4 \cdot \left(\frac{q.3q-q^2}{t}\right) - 7q$	**B.**	$\frac{2x+3y^2-4z}{\frac{z}{3a}}$	**C.**	$\frac{r+\frac{12m}{2}+3m}{m.r}$
D.	$\frac{-a+\frac{r^2}{b}+b^3}{r^{2-1}}$	**E.**	$\frac{\sqrt{a}-1}{\sqrt{2.\frac{b}{c}-a}}$	**F.**	$\frac{y^2z+x}{x-y} + \frac{2z+1}{5y}$
G.	$\frac{2y \cdot x+b \cdot y}{a \cdot 4w-v \cdot y}$	**H.**	$\frac{5z+x}{x-y} - \sqrt{\frac{x}{25}}$	**I.**	$\frac{w}{t} + \frac{2m-7t}{w}\ t^3$

Nota: La operación raíz cuadrada se indicará del siguiente modo: raíz (operando) donde operando es un dato numérico o expresión aritmética.
La operación potencia se indicará del siguiente modo: potencia (base, exponente) dónde base y exponente son datos numéricos.

Ejercicio 4

Complete el cuadro, teniendo en cuenta los valores que se consignan:
a=12; b=(-5); x=5; y= 2; m=9; n=7; k=(-56); J=`t´; F=6; V=FALSO; q=74; T=10

Expresión	**Operadores** (Tipo)	**Operandos** (Tipo de dato)	**Resultado** (Tipo de dato)	**Evaluación** (Valor obtenido)
a) q/2-68+n*5-T				
b) NO V O (J <=’J’)				
c) (-58))Y(k!= 0)==((k-y)				
d) (m/3*n)+ (-6)*x*y - k				
e) (((a*y)/F)<1) **Y** V				

f) (3*a + b) > (2x - y)				
g) (k resto a) div T				
h) ((a * 3 <= 10) Y (b >= -b * a /2) Y (a+b- 7) != m))				
i) NO ((x * a) > (q / y))				

Ejercicio 5
Escribir en forma de expresiones los siguientes enunciados

Concepto	*Expresión*
a) El opuesto de un número	
b) N no es múltiplo de 9 ni divisible en 5	*no (N resto 9== 0) y (N resto 5!=0)*
c) N es positivo	
d) N es divisible en 5 y múltiplo de 4	
e) N no es mayor al doble de M	
f) N es negativo o impar	
g) N es un número par	*N resto 2 == 0*
h) N es el triple del sucesor de y	
i) El sucesor par de 2*k	
j) A es menor que B y C, y además B no es mayor que C y es mayor que A	
k) La adición de 3 números consecutivos es como mínimo igual a 3 centenas	
l) El triple de B no supera a la quinta parte de A	
m) El triple del cuadrado de un número p no es impar	
n) El resto de dividir un número por 3 es menor a la diferencia entre la mitad de la variable M y 35	
o) El doble de una variable A menos la tercera parte de una variable B no supera a 500	

Ejercicio 6
Construir la expresión correspondiente a cada uno de los siguientes enunciados lógicos

A. Datos de un artículo: *código_artículo, precio, stock*

1. Artículo que no cueste más de 180 pesos y cuyo stock sea superior a 300.
2. Código de artículo que no supere el ochocientos cincuenta, con un precio inferior a los noventa y cinco pesos.
3. Código de artículo comprendido entre 80 y 150 cuyo stock no sea menor a 25.
4. Artículo sin stock o con un precio que supere los 1000 pesos.

B. Datos del alumno: número de registro: *reg_alumno*, código de departamento (codificado `A´: Astronomía y Geofísica, `G´: Geología, `B´: Biología, `I´: Informática): *cod_dpto*, año que cursa*: cursa,* cantidad de materias que cursa: *mat_cursa*

1. Alumno que pertenece al departamento de Biología o Geofísica, con número de registro menor a veinte mil ciento cuatro.
2. Alumno que cursa entre segundo y quinto año en el departamento de Informática.
3. Alumno que esté cursando al menos dos materias en cuarto año y sea del departamento Geología.
4. Alumno de Biología inscripto como mínimo en dos materias

C. Datos del paciente: *identif_pac, edad, género* (codificada `F´: femenino `M´: masculino)**,** *esp_medica* (codificada 1: ClinicaMédica, 2:Ginecologia: 3:Pediatría 4:…. 10: Urología)**,** *nombre_doctor*

1. Paciente mujer de menos de 30 años y más de 20 años cuya identificación de paciente no sea superior a 950 y necesita atención en la especialidad de ginecología.
2. Paciente varón entre 30 y 50 años que se atiende con el doctor Pérez
3. Paciente de la especialidad pediatría atendido por el doctor Clavel y cuya edad esté comprendida entre los 0 y los 3 años.

D. Datos del usuario*: consumo-luz, dpto* y *tipo-usuario* (codificado **"**Re":residencial, **"**Ind**"**:industrial, **"**Sub**"**: subsidiado)

1. Usuario subsidiado de Capital cuyo consumo no es mayor a 150 kwv ni inferior a 55 kwv.
2. Usuario cuyo consumo no supere los 180 kwv y que viva en Chimbas o Albardón.
3. Usuario industrial que no viva en Rawson, con consumo superior 330 kwv o que sea inferior a 950 kwv.

E. Datos del empleado: Código de empleado: *Num_empl*, Edad: *E*; Provincia donde nació: *Prov*, Sexo: *S* (codificada F:femenino M:masculino), Sueldo básico: *Sueldo* y Años de antigüedad: *Antig*

1. Empleados hombres mayores de 35 años que tienen sueldo básico superior a $5000
2. Empleados de sexo femenino mayores a 25 años o sexo masculino que tengan entre 30 y 45 años.
3. Empleados nacidos en San Juan con menos de 5 años de antigüedad.
4. Empleados de sexo masculino nacidos en Mendoza con código de empleado inferior a 100.

F. Datos del aspirante a una vivienda del IPV: Número inscripción: *Num_insc*, Años de inscripto: *AI*; Estado civil: *Est_civil* (codificado S:soltero, C:casado, V:Viudo) Puntaje IPV: *Punt*, Código ingreso al sorteo: *Cod_ing* (codificado P: prioridad N: sin prioridad)

1. Aspirante vivienda IPV solteros sin prioridad.
2. Aspirante vivienda IPV con al menos 15 años de inscripción y un puntaje de IPV mayor a 350.
3. Aspirante vivienda IPV que no sean solteros y número de inscripción inferior a 650

Ejercicio 7

Realice el seguimiento o traza de los siguientes algoritmos y responda lo indicado.

A. *Lote de prueba*: *superf*: 625 *superf*: 100 *superf*: 240,25

algoritmo cuadrado Comienzo real superf, lado Escribir "Ingrese superficie del cuadrado" Leer superf lado = raíz(superf) Escribir "Cada lado del cuadrado mide", lado Fin	algoritmo cuadrado Comienzo real superf Escribir "Ingrese superficie del cuadrado" Leer superf Escribir "Cada lado del cuadrado mide", raíz(superf) Fin

- ¿Qué permite hacer la acción *Leer …..* ?
- Indique cual algoritmo considera es más óptimo, justificando su respuesta.

B. *Lote de prueba*: *lt*: 62,5 *lt:* 100 *lt:* 237,015

algoritmo convertir Comienzo real lt Escribir ”Ingrese una capacidad en litros” Leer lt Escribir “La capacidad en pintas es”, (lt *1.76) Fin	algoritmo convertir Comienzo real lt, pintas Escribir ”Ingrese una capacidad en litros” ……. lt pintas= …… Escribir “La capacidad en pintas es”, ……. Fin

- Completar según corresponda e indique si ambos algoritmos realizan lo mismo.

C. El siguiente algoritmo calcula el sueldo de un empleado teniendo como datos de entrada: el nombre, cantidad hrs. de trabajo semanales y el valor de la hora.

Lote de prueba: Juan Pérez, 30 - Eliana Álvarez, 45 - José Robledo, 25

```
algoritmo salario
Comienzo
Constante Pagohr=250.30
cadena nya-empl
entero  hrs
real Sueldo
 Escribir ”Ingrese nombre del empleado”
 Leer nya-empl
 Escribir ”Ingrese la cantidad de horas trabajadas”
 Leer hrs
 Sueldo= Pagohr*hrs
 Escribir ”El empleado:”, nya-empl , “tiene un sueldo de:”, Sueldo
Fin
```

- Realice las modificaciones necesarias para evitar el uso de la variable *Sueldo* y la constante *Pagohr*

D. El algoritmo que se muestra a continuación permite calcular la calificación final de una materia, sabiendo que dicha nota se compone de los siguientes porcentajes.
55% corresponde al promedio final de las calificaciones obtenidas en los parciales (3)
30% para la evaluación integradora
15% es lo asignado al trabajo práctico

Lote de prueba: 9, 9, 10, 8, 9 - 8, 9, 9, 8, 8 - 9, 8, 7, 9, 7

```
algoritmo notafinal
Comienzo
real P1, P2, P3, Prom, Integrador, TP, PFinal
 Escribir  ”Ingrese notas de los Parciales (3)”
 Leer P1, P2, P3
 Prom=(( P1+ P2+ P3 )/3)*0.55
 Escribir ”Ingrese nota Evaluación integradora”
 Leer Integrador
 Escribir  ”Ingrese nota Trabajo Práctico”
```

```
Leer TP
PFinal= (Prom + (Integrador*0.30) + (TP *0.15) )
Escribir "La calificación final obtenida es:",PFinal
Fin
```

- Realice las modificaciones que considere conveniente para calcular la calificación final a través de una única expresión.

E. Mediante subprogramas el algoritmo siguiente realiza la conversión de una medida en centímetros ingresada por teclado a una medida del sistema británico-americano.

Nota: Las equivalencias son:

Sistema Métrico	Sistema Británico-Americano
1 cm	2,54 pulgadas
1 cm	91,44 yardas

Lote de prueba: *cm*: 120 *cm:* 10,37

```
algoritmo calcular
real apulgadas (real xcm)
Comienzo
 real pulgada
  pulgada = xcm * 2.54
retorna (pulgada)
Fin

void ayardas (real xcm)
Comienzo
  Escribir "La medida en yardas es", (xcm * 91.44)
retorna ()
Fin

/*----Algoritmo principal----*/
Comienzo
 real cm
  Escribir "Ingrese una medida en centimetros"
  Leer cm
  Escribir "La medida en pulgadas es", apulgadas(cm)
  ayardas(cm)
Fin
```

- Complete el algoritmo anterior para que usando subprogramas se puedan realizar las restantes conversiones al sistema británico-americano.

  Nota: Las equivalencias son:

Sistema Métrico	Sistema Británico-Americano
1 cm	30,48 pies
1 cm	1,60934 millas

Ejercicio 8

Escriba un enunciado que pueda ser resuelto a través del siguiente algoritmo

Nota 10 gr = 0,3527 oz

```
algoritmo convertidor
 real gramos-a-onzas (real xgr)
 Comienzo
  real calc-onza
   calc-onza = (0.3527 * xgr) / 10
 retorna (calc-onza)
 Fin


 /*----Algoritmo principal----*/
Comienzo
 real gramos, onzas
   Escribir  ”Ingrese un valor en gramos”
   Leer gramos
   onzas = gramos-a-onzas(gramos)
   Escribir gramos,  “gramos equivale a: ”, onzas “onzas”
Fin
```

- Indicar la diferencia que existe entre “onzas” y gramos en la acción Escribir

Ejercicio 9

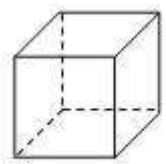
Construir un algoritmo que calcule el cuadrado de un número ingresado por teclado y la quinta parte de ese valor calculado.

```
algoritmo  calculos
entero cuadrado (entero xnum)
 Comienzo
 entero num, parte
   cuad=num*…..
 retorna (cuad)
 Fin
/*----Algoritmo principal----*/
Comienzo
entero num, parte
  Escribir ”Ingrese un numero entero”
  Leer ……..
  Escribir “El cuadrado del número  es: ”, cuadrado(num)
  parte = cuadrado(num) div 5
  Escribir “ La …….  parte del número es: ”, ……
Fin
```

- Indique la diferencia entre *“….cuadrado….”* y *cuadrado* en la acción Escribir (resaltada)

Ejercicio 10

Complete y luego escriba el enunciado que es resuelto por este algoritmo.



Nota:
Área cubo= 6 x arista^2
Volumen cubo = arista^3

```
algoritmo  cubo
 void calculoarea (entero xarista)
 Comienzo
 entero area
   area = 6*(xarista * …….)
   Escribir “El área del cubo es”, ……
 retorna ()
 Fin
 void calculovolumen (entero xarista)
 Comienzo
 entero volumen
   volumen = (xarista * ….……* …….)
   Escribir “El volumen del cubo es”, ……
 retorna ()
 Fin
 /*----Algoritmo principal----*/
 Comienzo
 real  arista
   Escribir ”Ingrese valor arista ”
   Leer ……..
  calculoarea(arista)
  calculovolumen(arista)
 Fin
```

- Modificar el algoritmo para que muestre el área del cubo en el algoritmo principal.

Ejercicio 11

Escriba el enunciado que representa lo desarrollado en el algoritmo, completando previamente.

Volumen cilindro = $\pi r^2 h$

Nota 1lt = 1000 cm^3

```
algoritmo  latas
 real calculovolumen (real xradio, real  xaltura)
 Comienzo
 entero xvolumen
   xvolumen = (3.14 *(xradio * ….……) * …….)
retorna (xvolumen)
 Fin
 entero latas (real xlt, real  xvol)
 Comienzo
 entero canti
   canti = (xlt * 1000)/ ……
 retorna (canti)
 Fin
```

```
/*----Algoritmo principal----*/
Comienzo
real  altura, radio, volumen, litros, cantidad
  Escribir ”Ingrese valor altura (en cm) ”
  Leer ……..
  Escribir ”Ingrese valor ……. .  (en cm)”
  Leer ……..
  volumen=calculovolumen (radio, altura)
  Escribir ”Ingrese cantidad de litros a envasar”
  Leer ……..
  cantidad = latas(litros,volumen)
  Escribir  “ Con ”, ……  , “ litros se pueden llenar”  cantidad  “ latas de “ ……  “cm3“
Fin
```

- ¿Se puede evitar el uso de las variables *volumen* y *cantidad*? En caso que sea afirmativa la respuesta, realice las modificaciones necesarias.

Ejercicio 12

En una estación de servicio los surtidores de combustible registran la nafta vendida en galones, pero su precio está en litros. Realice un algoritmo con subprogramas que permita calcular y mostrar lo que hay que cobrarle al cliente.

Nota: 1 litro equivale a 0.2642 galones

Ejercicio 13

Una persona recibe un préstamo de un banco por un año y desea saber cuánto pagará de interés. El banco le cobra una tasa del 1.8% mensual. Realice un algoritmo que con un subprograma permita determinar este monto.

Ejercicio 14

Escriba un algoritmo que realice a través de subprogramas haga el cálculo de la hipotenusa de un triángulo rectángulo, en función de los catetos.

Nota: Teorema de Pitágoras: “En todo triángulo rectángulo el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los catetos”.

Ejercicio 15

Un estudio biológico demostró que el número de sonidos emitidos por un grillo en un minuto, es en función de la temperatura ambiente expresada en grados Fahrenheit. Como resultado podría utilizarse al grillo como termómetro ambiental.

La fórmula que obtuvieron y determina esto es: $T=N/4+40$ (T: temperatura en grados Fahrenheit; N: número de sonidos emitidos por el grillo)

Realizar un algoritmo utilizando subprogramas en forma adecuada para que: teniendo en cuenta el número de sonidos emitidos por el grillo muestre la temperatura ambiental en grados centígrados.

Nota: °C = (°F- 32)x5/9 grados Fahrenheit a grados Celsius

Ejercicio 16

Realice un algoritmo con subprogramas que conociendo el año de nacimiento de una persona, indique cuantos meses de vida tiene hasta el año actual.

Implemente las variantes de subprogramas que permitan mostrar el resultado en el algoritmo principal y en el subprograma.

Ejercicio 17

La capacidad de almacenamiento de las computadoras se describe en kilobytes de memoria. Un kilobyte equivale a 1.024 bytes. Un carácter necesita un byte de almacenamiento (una letra, un dígito, un signo de puntuación o un símbolo).

a) Realice un algoritmo que a través de subprogramas determine la cantidad de caracteres que puede almacenar una computadora de 2048 kilobytes de memoria.

b) Investigue e indique como se debería realizar el cálculo si se usa otra medida de almacenamiento (bytes y sus múltiplos)
c) Implemente las modificaciones necesarias para una computadora cuya cantidad memoria es un valor ingresado por teclado.

Ejercicio 18
Escriba un algoritmo con subprogramas y su correspondiente programa en Lenguaje C para calcular la cantidad de latas de pintura necesaria para pintar una habitación, ingresando como datos: las medidas de la habitación (ancho, largo y alto) y la cantidad de cada tipo de aberturas que hay en ella.
Por otro lado se conoce que: las puertas son de 0,75 mts de ancho y 2,00 mts de alto y las ventanas tienen una medida de 1,20 x 1,50 mts; y la pintura se vende en latas de un litro y cada litro rinde 12 m^2

Ejercicio 19
Para realizar el cálculo de la cantidad de frigorías que un aire acondicionado necesita para ambientar un lugar, es necesario conocer el volumen en metros cúbicos de la habitación donde se va a instalar el aparato (*Nota*: el volumen de la habitación se calcula multiplicando su ancho, largo y alto).
Realizar algoritmo con subprogramas y la implementación de un programa en Lenguaje C para calcular la cantidad de frigorías necesarias para mantener un ambiente refrigerado.

Unidad 2: Acciones Estructuradas

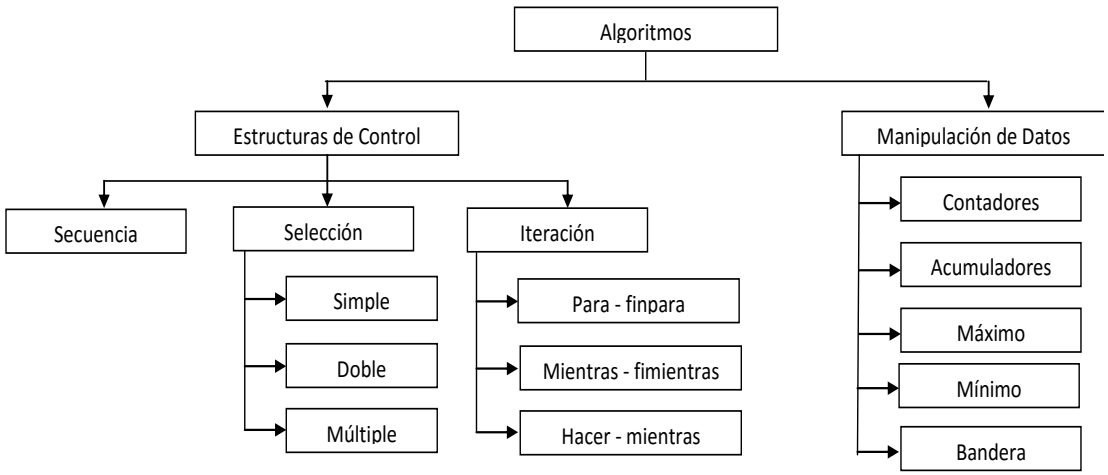
Introducción

Hasta ahora, los algoritmos han consistido en simples secuencias de instrucciones.
Existen tareas más complejas que no pueden ser resueltas así (repetir una misma acción, realizar acciones diferentes en función del valor de una expresión, etc.).
Para resolver esto existen las estructuras de control. Estas tienen un único punto de inicio y un único punto de finalización. Se componen de sentencias o de otras estructuras de control.
Estas estructuras señalan el flujo de control que tiene un algoritmo, entendiendo por **flujo de control** de un algoritmo al orden en que se ejecutan las acciones que lo constituyen.
Con las estructuras de control se pueden construir algoritmos que resuelvan problemas mas complejos, esto requiere manipulacion de datos para hacer operaciones que permitan contar, sumar, identificar el valor mayor o menor de un conjunto como asi también poder saber si se produce una situación determinada cuando se procesan datos, considerando estas operaciones como básicas en el procesamiento de datos.

En esta unidad se verán estrutucturas de control y manipulacion de datos

Cuando el algoritmo cuenta con un número reducido de acciones que se ejecutan una tras otra, en el mismo orden en el que aparecen, se habla de ejecución secuencial.
El flujo de control de un algoritmo también puede ser alternativo, cuando se deben tomar decisiones que dependen de ciertas condiciones, o iterativo cuando es necesario repetir acciones varias veces.

En esta unidad se profundiza el tema estructuras de Control y su uso, el siguiente esquema representa los contenidos que desarrollaran:



Estructuras de Control

Se llaman **acciones estructuradas** a aquellas que determinan un flujo de control específico; de ahí que también se denominan **estructuras de control**. Las acciones estructuradas, se ejecutan teniendo en cuenta los distintos tipos de flujo de control: secuencia, selección e iteración**.**

Estructuras de Control { Secuencia; Selección; Iteración

2.1 Secuencia

Una secuencia está representada por un conjunto de acciones que se ejecutan en forma consecutiva, una sola vez. En este caso el orden de ejecución coincide con el orden físico en que se representan las acciones.

El siguiente esquema representa esta estructura:

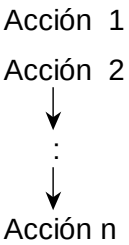
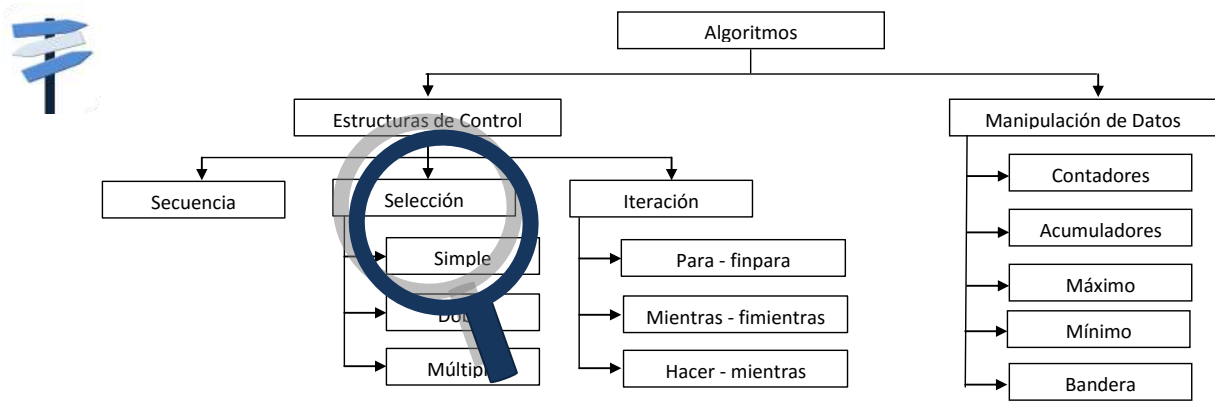


Figura 2.1 Esquema de estructura Secuencial

En el siguiente ejemplo se señalan las secuencias contenidas en el algoritmo

```
Algoritmo numero
entero superficie ( entero  largo, entero ancho )
comienzo
Secuencia {
    entero sup
    sup= largo*ancho
    retorna(sup)
fin

Comienzo //----Algoritmo principal----
Secuencia {
    entero l, a
    leer l,a
    escribir" la superficie del rectangulo es ", superficie (a,l)
Fin
```



2.2 Selección o Alternativa

Una selección es una acción estructurada que provoca la ejecución de una acción entre acciones alternativas. Las estructuras de selección o alternativa utilizadas son Alternativa Doble, Simple y Múltiple, cuyo seudocódigo se muestra a continuación:

Selección Doble

El esquema de la estructura de selección doble es el siguiente:

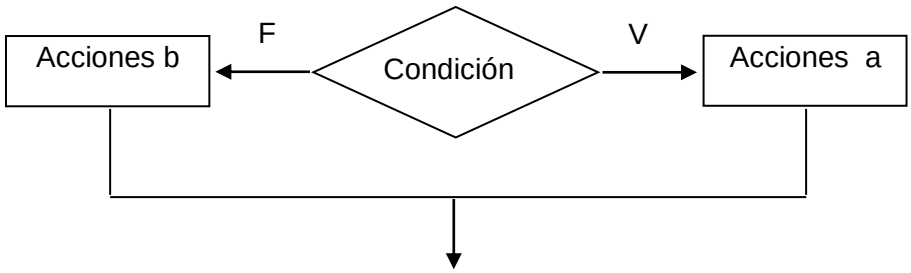


Figura 2.2 Alternativa Doble

En la figura 2.2 se grafica la selección utilizando un rombo para representar la decisión y un rectángulo para representar un bloque de acciones secuenciales

El seudocódigo para esta estructura es:

Si (<condición>)
Entonces
<acción simple o estructurada>
Sino
<acción simple o estructurada>
Finsi

Donde**:**
Condición es una variable lógica, expresión lógica o relacional, cuyo valor puede ser verdadero o falso.

Las palabras **Si**, **Entonces**, **Sino** y **FinSi**, se llaman delimitadores.

Esta estructura se interpreta de la siguiente manera, primero se evalúa la condición, si el resultado obtenido es verdadero, el procesador deberá ejecutar la acciones que están entre los delimitadores Entonces y Sino, en caso contrario ejecutará la acción simple o estructurada que está entre los delimitadores Sino y Finsi.
Las acciones que siguen a **Entonces** (alternativa verdadera) o al **Sino** (alternativa falsa), dependiendo del estado de la condición evaluada, se ejecutarán en forma excluyente una sola vez.
Como se observa en el seudocódigo, las palabras **Entonces**, **Sino** así como las acciones que están dentro de ellas, se encuentran desplazadas un poco más a la derecha que el resto de la estructura.
Este desplazamiento o sangría se denomina **indentación** y es utilizada para dar mayor claridad a los algoritmos.

Ejemplo 1

Problema

Se cuenta con el resultado del tratamiento realizado a un paciente para dejar de fumar. El resultado se representa con E si fue exitoso y F si fracasó. Mostrar un mensaje según haya sido el resultado.

Etapa 1 Análisis

Salida	Datos de Entrada	Proceso
Mensaje **Postcondición**: salida por pantalla de un mensaje	Resultado del tratamiento **Precondición** el dato de entrada puede ser representado con los caracteres 'E' y ' F'	1-Ingresar dato 2- Si el resultado es exitoso mostrar mensaje "El resultado fue exitoso" sino mostrar mensaje "El tratamiento fracasó"

Etapa 2 Diseño del algoritmo

```
Algoritmo numero
void Mensaje ( carácter Tratamiento_fumar )
comienzo
    Si (Tratamiento_fumar== 'E')
            Entonces
                Escribir "El resultado fue exitoso"
            Sino
                Escribir "El tratamiento fracasó"
    finsi
fin

Comienzo //----Algoritmo principal----
    Caracter  Tratamiento_fumar
    Leer Tratamiento_fumar
    Mensaje (Tratamiento_fumar)
Fin
```

ACTIVIDAD 1

Dado el siguiente enunciado del problema, realice la etapa de análisis y revise el algoritmo presentado con su seguimiento.

.

Problema: La emergencia energética por la que atraviesa el país debido a la falta de inversión de las empresas responsables de esa área en los últimos años, ha llevado a que en la actualidad deba tomarse medidas que afectan directamente al usuario. Energía San Juan ha implementado un sistema de castigos y ha puesto en vigencia una disposición que consiste aplicar a aquellos usuarios que incrementaron su consumo respecto al mismo bimestre del año anterior en más de 100KW, un recargo del 20% al subtotal correspondiente a energía.

El subtotal energía está constituido por la suma de un Cargo Fijo mensual de $67.13 y un Cargo variable que se obtiene de multiplicar la mitad del consumo bimestral registrado, por el precio unitario correspondiente al KW/hora, que en la actualidad es de $0.8141.

Del usuario se conocen los consumos bimestrales del año actual y del mismo periodo del año anterior.

Se necesita informar a un usuario el importe que debe pagar, tener en cuenta si fue sancionado o no y para el caso de haber sido sancionado especificar el valor del recargo aplicado.

```
Algoritmo Consumos
constante precio_KW = 0.8141
constante cargofijo = 67.13

real Cargovariable (real ca)
comienzo
    real cv
    cv =(ca / 2) * precio_KW)
    retorna(cv)
fin

real SubtEnergia (real cactual )
comienzo
    real st
    st = cargofijo + Cargovariable (cactual)
    retorna(st)
fin

void Mensaje (real actual, real anterior, real subtotal)
comienzo
real recargo
    Si  (actual > (anterior + 100))
       Entonces
          recargo= subtotal * 0.20  // calculo de porcentaje  s*20/100
          Escribir  "El usuario debe pagar un recargo de", recargo
          Escribir  "Importe Subtotal Energía", subtotal + recargo
       Sino
          Escribir "El usuario  no debe pagar recargo, Importe Subtotal Energía " , subtotal
    Finsi
fin
```

Comienzo //----Algoritmo principal----
real consumo_actual,consumo_anterior
Escribir "Ingrese consumos bimestrales actual y del mismo periodo año anterior "
Leer consumo_actual, consumo_anterior
s=SubtEnergía (consumo_actual)
Mensaje (consumo_actual, consumo_anterior, s)
Fin

El seguimiento del algoritmo para los lotes de prueba es:
a. consumo_actual = 550 KW consumo_anterior= 420 KW
b. consumo_actual = 450 KW consumo_anterior = 420 KW

Lote a):
consumo_actual = 550 KW
consumo_anterior= 420 KW

Algoritmo principal		
consumo_actual	consumo_anterior	**Salida**
550.00	420.00	Ingrese consumos bimestrales actual y del mismo periodo año anterior

subtEnergia		
cactual	st	**Salida**
550.00	291.01	

cargovariable		
ca	cv	**Salida**
550.00	223,88	

Mensaje				
consumo_actual	consumo_anterior	s	recargo	**Salida**
550.00	420.00	291.01	58.20	El usuario debe pagar un recargo de 58.2 Importe Subtotal Energía 349.21

Lote b)
consumo_actual = 450 KW
consumo_anterior = 420 KW

Algoritmo principal		
consumo_actual	consumo_anterior	**Salida**
450.00	420.00	Ingrese consumos bimestrales actual y del mismo periodo año anterior

SubtEnergia		
cactual	st	**Salida**
450.00	250.30	

Cargovariable		
ca	cv	**Salida**
450.00	183.17	

Mensaje				
consumo_actual	consumo_anterior	s	recargo	**Salida**
450.00	420.00	250.30		El usuario no debe pagar recargo, Importe Subtotal Energía 250.30

Selección Simple

Si el departamento informes de Energía San Juan, dispone que sólo debe notificar al usuario cuando deba pagar recargo, no será necesario realizar acciones cuando la condición es falsa. Éste, como muchos problemas a modelar, pueden requerir el uso de la estructura Si - Finsi sin necesidad de usar el delimitador Sino. Esta estructura se conoce como Alternativa Simple.
El seudocódigo asociado es:

Si (<condición>)
Entonces <acción simple o estructurada>
Finsi

El esquema de la estructura de selección simple es:

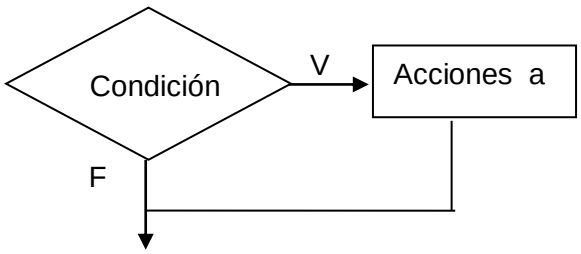


Figura 2.3 Alternativa Simple

Problema: La emergencia energética por la que atraviesa el país debido a la falta de inversión de las empresas responsables de esa área en los últimos años, ha llevado a que en la actualidad deba tomarse medidas que afectan directamente al usuario. Energía San Juan ha implementado un sistema de castigos y ha puesto en vigencia una disposición que consiste aplicar a aquellos usuarios que incrementaron su consumo respecto al mismo bimestre del año anterior en más de 100KW, un recargo del 20% al subtotal correspondiente a energía.
El subtotal energía está constituido por la suma de un Cargo Fijo mensual de $67.13 y un Cargo variable que se obtiene de multiplicar la mitad del consumo bimestral registrado, por el precio unitario correspondiente al KW/hora, que en la actualidad es de $0.8141.
Del usuario se conocen los consumos bimestrales del año actual y del mismo periodo del año anterior.
Se necesita informar a un usuario el importe que debe pagar si fue sancionado, especificar el valor del recargo aplicado.

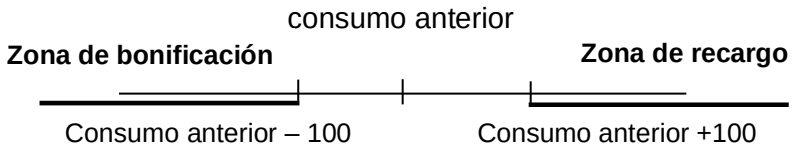
```
Algoritmo Consumos1
constante precio_KW = 0.8141
constante cargofijo = 67.13

real Cargovariable (real ca)
comienzo
   real cv
   cv =(ca / 2) * precio_KW)
   retorna(cv)
fin

real SubtEnergia (real cactual )
comienzo
   real st
   st = cargofijo + Cargovariable (cactual)
   retorna(st)
fin

void Mensaje (real actual, real anterior, real subtotal)
comienzo
real recargo
   Si  (actual > (anterior + 100))
      Entonces
         recargo= subtotal * 0.20  // calculo de porcentaje  s*20/100
         Escribir  "El usuario debe pagar un recargo de", recargo
         Escribir  "Importe Subtotal Energía", subtotal + recargo
      Finsi  // comparar con el ejemplo anterior donde figura el sino
fin

Comienzo //----Algoritmo principal----
   real consumo_actual,consumo_anterior
   Escribir "Ingrese consumos bimestrales  actual y  del mismo periodo año anterior "
   Leer consumo_actual, consumo_anterior
   s=SubtEnergía (consumo_actual)
   Mensaje (consumo_actual, consumo_anterior, s)
Fin
```

Acciones de Selección Anidadas

Problema: Después de reuniones de directorio, las autoridades de Energía San Juan decidieron modificar la disposición y poner en vigencia un nuevo sistema que consiste en otorgar a sus usuarios premios y castigos. Se premiará a aquellos usuarios que redujeron su consumo por lo menos en 100kw respecto del mismo periodo del año anterior, con una bonificación del 15% sobre el importe total. Y al igual que en la disposición anterior, aquellos usuarios que incrementaron su consumo en más del 100kw serán castigados con un recargo del 20% del importe.

Modificar el algoritmo anterior de modo que permita informar cual es su situación a un usuario, del que se conocen los consumos del mismo bimestre en los dos últimos años.

Representación del problema
Deberá verificarse en cuál de las zonas se encuentra el consumo actual, de manera de calcular recargo, bonificación en caso que corresponda.

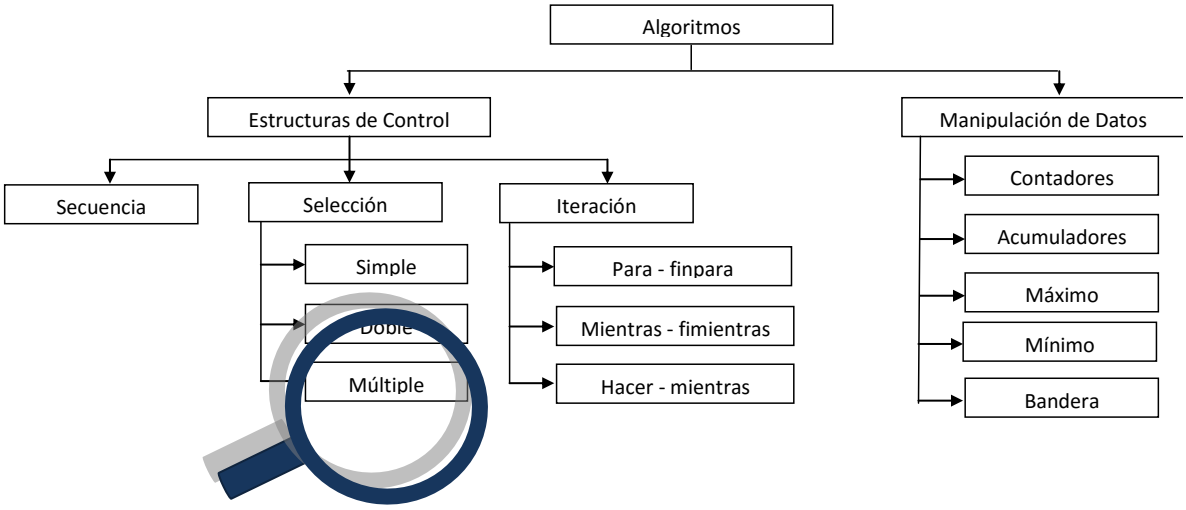


```
Algoritmo Consumos2
constante precio_KW = 0.8141
constante cargofijo = 67.13

real Cargovariable (real ca)
comienzo
    real cv
    cv =(ca / 2) * precio_KW)
    retorna(cv)
fin

real SubtEnergia (real cactual )
comienzo
    real st
    st = cargofijo + Cargovariable (cactual)
    retorna(st)
fin
```

```
void Mensaje (real actual, real anterior, real subtotal)
comienzo
real recargo
 Si  (actual > (anterior + 100))
  Entonces
          recargo= subtotal * 0.20  // calculo de porcentaje  subtotal*20/100
          Escribir  "El usuario debe pagar un recargo de ", recargo
          Escribir  "Importe Subtotal Energía ", subtotal + recargo
  Sino
     Si  (consumo_actual  < consumo_anterior - 100)
       Entonces
            descuento = subtotal * 0.15   // calculo de porcentaje  subtotal*15/100
            Escribir  "El usuario  tiene un descuento de  ",  descuento
            Escribir  "Importe Subtotal Energía ", subtotal - descuento
       Sino
         Escribir "NO corresponde bonificación ni recargo, su importe es", subtotal
     Finsi
 Finsi
fin

Comienzo //----Algoritmo principal----
   real consumo_actual,consumo_anterior
   Escribir "Ingrese consumos bimestrales  actual y  del mismo periodo año anterior "
   Leer consumo_actual, consumo_anterior
   s=SubtEnergía (consumo_actual)
   Mensaje (consumo_actual, consumo_anterior, s)
Fin
```

Observese que las llaves colocadas en las estrucuras Si - finsi de la funcion Mensaje, permiten observar graficamente si es correcto o no el anidamiento.
Es correcto si las llaves no se cortan entre si, significa que la estructura se inicia y se termina por completo en otra, de lo contrario hay un error.

Las acciones que siguen a la alternativa verdadera o falsa de una estructura Si... Finsi, pueden ser simples o estructuradas, en particular puede ser otra estructura alternativa, como la planteada en el ejemplo.
En este caso, en donde un "Si aparece dentro de otro Si", se habla de anidamiento de Si o estructuras alternativas anidadas



2.3 Selección Múltiple

Problema

Construir un algoritmo que permita determinar el importe total incluido el impuesto, que corresponde a un automóvil cuyo valor y código de marca se conocen. La agencia de venta de automotores, ha codificado las distintas marcas de automóviles para facilitar el tratamiento de la información. La tabla siguiente muestra las marcas de los automóviles con el porcentaje correspondiente a un impuesto que debe agregarse al valor sugerido por fábrica.

Marca	Código	Porcentaje
Ford	A	5%
Peugeot	B	7%
Fiat	C	8%
Toyota	D	9%
Renault	E	6%

Etapa Análisis del problema

¿Qué se debe obtener? ***DATOS DE SALIDA***	***¿Con que datos se cuenta?*** ***DATOS DE ENTRADA***	***PROCESO***
Importe que debería pagarse incluido el impuesto correspondiente.	Código de marca del automóvil Valor del automóvil Porcentaje del impuesto según el código	*1-Ingresar datos de entrada* *2- Calcular el importe adicionando al valor del automovil el porcentaje del impuesto según la marca* *3- Mostrar importe del automóvil*

Con las estructuras hasta ahora estudiadas, el algoritmo que muestra el proceso para la obtención del resultado solicitado, puede ser el siguiente:

```
real TotalAuto (carácter xcodigo, real ximp)
comienzo
real total
  Si (xcodigo==’A’)
      Entonces  total = ximp *1.05 // es equivalente a total = ximp + ximp * 5 / 100
      Sino Si (xcodigo==’B’)
                Entonces  total = ximp *1.07
                Sino Si (xcodigo==’C’)
                          Entonces  total = ximp *1.08
                          Sino Si (xcodigo==’D’)
                                    Entonces  total = ximp *1.09
                                    Sino    Si (xcodigo==’E’)
                                              Entonces  total = ximp *1.06
                                              Sino  Escribir “ Se ingreso mal el código”
                                             Finsi
                                  Finsi
                        Finsi
              Finsi
  Finsi
  retorna (total)
Fin
```

```
Algoritmo Autos
Comienzo
real imp, t
caracter codigo
 Escribir “ Ingresar importe de fábrica y código de automóvil”
 Leer  imp, codigo   // Se lee el importe del automóvil y el código correspondiente a su marca
 t = TotalAuto (imp, codigo)
 Escribir “Total a pagar por el automóvil, incluido el impuesto”, t
Fin
```

Para casos como éste, en los que, dependiendo del valor de una variable o del resultado de una expresión se debe seleccionar una de entre varias alternativas, los lenguajes proveen la estructura de selección múltiple Segun…finsegun cuyo seudocódigo es el siguiente:

```
Segun  (<e>)
  valor 1:   acción 1
  valor 2:   acción 2
    :
  valor n:   acción n
  de otro modo:  acción b
finsegun
```

Donde:

e es el nombre de *expresión de control* o *selector* y puede ser una variable o expresión ordinal, valores enteros, caracter, pero no reales.
valor 1, valor 2 son los valores (etiquetas) que puede tomar e.
de otro modo, valor distinto a la secuencia de posibles valores previstos que puede tomar la variable e.

El esquema correspondiente a la estructura de alternativa múltiple es:

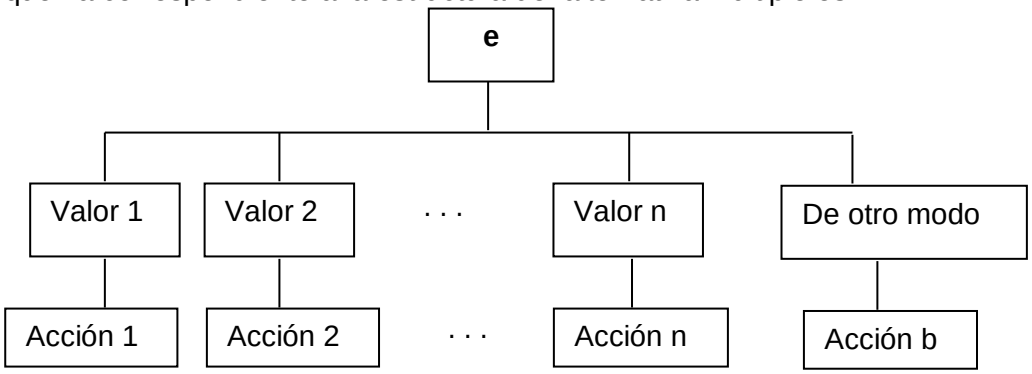


Figura 2.4 Alternativa Múltiple

Se evalúa el selector o expresión de control (e), su valor se compara con cada uno de los valores o etiquetas colocados en la estructura *Segun..finsegun*, y se ejecuta únicamente la acción- simple o estructurada-, que corresponde al valor obtenido. Una vez finalizada esta ejecución, se continúa con el código que sigue a la estructura.
En caso que el valor del selector no se corresponda con ninguna etiqueta, si no se coloca la opción *de otro modo*, no se ejecuta ninguna acción. Se recomienda el uso de esta opción para identificar posibles errores, como es el caso de ingresar un valor erróneo que debe ser detectado.
El ordenamiento de los valores o etiquetas en la estructura es arbitrario, pero no pueden aparecer más de una vez.

Ejemplo 2
El subprograma TotalAuto escrito en el algoritmo Autos, seria:

```
real TotalAuto (carácter xcodigo, real ximp)
comienzo
real total
  Segun (xcodigo)
        'A': total= ximp *1.05
        'B': total= ximp *1.07
        'C': total= ximp *1.08
        'D': total= ximp *1.09
        'E': total= ximp *1.06
        de otro modo: Escribir " Se ingreso mal el código"
  finsegun
  retorna (total)
Fin
```

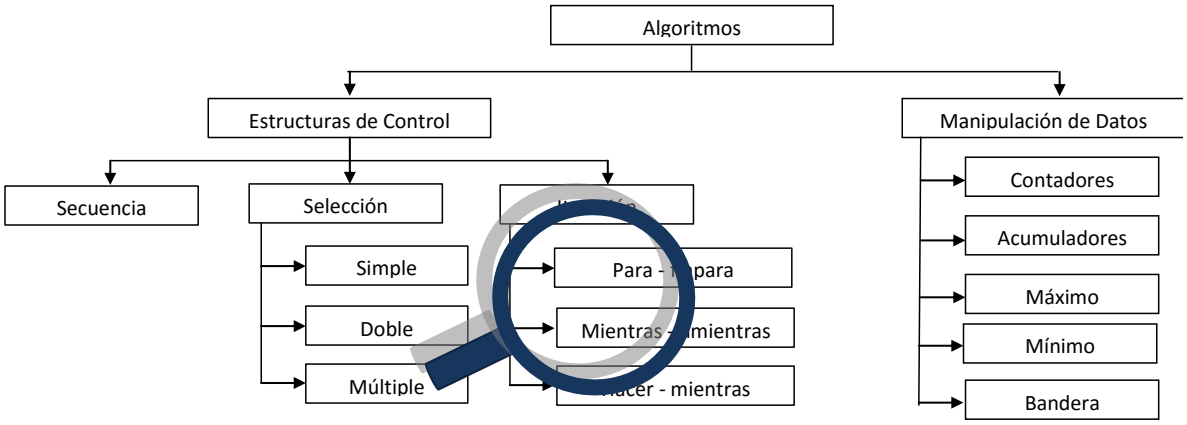
Si para diferentes valores de **e** se debe realizar la misma acción, tales valores pueden escribirse en una lista, separados por comas:

```
   Segun (mes)
       1,3,5,7,8,10,12 :  dia =31
                     2 :  dia =28
              4, 6,9,11:  dia =30
   finsegun
```

Ejemplo 3
Si un usuario de un cajero automático puede realizar tres posibles transacciones: consulta de saldo, extracción de fondos o depósito, el algoritmo que permite seleccionar la operación puede ser el siguiente

Algoritmo **Cajero**

```
void Menu()
comienzo
    Escribir " Ingrese    "
    Escribir "   C     consulta de saldo"
    Escribir "   E      extracción"
    Escribir "   D      depósito"
    Escribir "   T       finalizar transacción"
fin

void Extraer(real xsaldo)
comienzo
real importe
  Escribir " Ingrese monto a extraer "
  Leer importe
  Si (saldo ≥ importe)
     Entonces Escribir " Saldo anterior ", xsaldo
                  xsaldo= xsaldo - importe
                  Escribir " Saldo Actual", xsaldo
     Sino   Escribir "La operación no puede ser realizada, su saldo actual es de ", xsaldo
  Finsi
fin

void Deposito(real xsaldo)
comienzo
real importe
    Escribir " Ingrese  el monto a depositar "
    Leer importe
    Escribir " Saldo anterior ", xsaldo
    xsaldo= xsaldo + importe
    Escribir " Saldo Actual", xsaldo
fin

Comienzo // Algoritmo Principal
  real saldo, importe
  caracter opci
  Escribir " Ingrese Saldo:  "
  Leer  saldo
  Menu()
  Escribir " Ingrese opción"
  Leer  opci
  Según  (opci)
    'C' ,'c' : Escribir " Su saldo es ", saldo
    'E', 'e' : Extraer(saldo)
    'D', 'd':  Deposito(saldo)
    'T', 't' :  Escribir " Ud. Ingresó finalizar transacción "
    de otro modo: Escribir " Código incorrecto, intente nuevamente "
  Finsegun
Fin
```



2.4 Iteración

Las acciones estructuradas iterativas o repetitivas, se utilizan cuando se debe repetir la ejecución de una o más acciones.
Para referirse a las acciones iterativas se usan los términos ciclo o bucle. La acción o acciones que se repiten se llaman cuerpo del ciclo y cada repetición del bucle se conoce como iteración o pasada por el bucle.
Las estructuras iterativas que se conocen, difieren en el modo en que controlan el número de veces que el cuerpo del ciclo debe repetirse.
La repetición puede ser controlada por contador o por una variable de control.

Estructura Para - FinPara

La estructura Para FinPara es una estructura iterativa, por cuanto representa la ejecución repetida de una o más acciones. En esta estructura, el número de veces que se ejecuta el cuerpo del ciclo es un número fijo conocido con anticipación, por ello también se la conoce como repetición definida o bucle controlado por contador.
El seudocódigo que representa esta estructura es:

Para v **desde** vi **hasta** vf
acción 1
acción 2
⋮
acción n
Cuerpo del ciclo
FinPara

Donde: **v** variable que controla el ciclo o variable de control.
vi variable que contiene el valor inicial de la variable v
vf variable que contiene el valor final de la variable v.

El cuerpo del ciclo puede contener acciones simples o estructuradas.

La primera vez que se ejecuta la estructura Para-FinPara, v toma el valor vi y compara que no supere el valor de vf. Si el resultado de la comparación es verdadero se ejecutan las acciones del cuerpo del ciclo que están entre los delimitadores Para y FinPara. Después de la primera iteración se incrementa en 1 el valor de v, y se vuelve a controlar que sea menor o igual a vf, en este caso se vuelve a iterar, se continúa repitiendo el proceso hasta que el valor de v supere el valor de vf.

El esquema que caracteriza a esta estructura de repetición es:

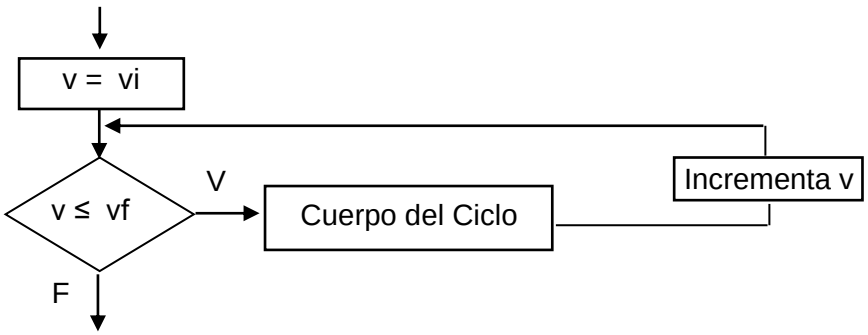


Figura 2.5 Estructura Iterativa Para FinPara

Ejemplo 4
Retomando la actividad 1, pero suponiendo que son 1000 los usuarios de Energía San Juan, a los que se debe informar su situación respecto del régimen de castigos aplicado.

Problema
La emergencia energética por la que atraviesa el país debido a la falta de inversión de las empresas responsables de esa área en los últimos años, ha llevado a que en la actualidad deba tomarse medidas que afectan directamente al usuario. Energía San Juan ha implementado un sistema de castigos y ha puesto en vigencia una disposición que consiste aplicar a aquellos usuarios que incrementaron su consumo, respecto al mismo bimestre del año anterior, en más de 100KW, un recargo del 20% al subtotal correspondiente a energía.
El total a pagar que figura en una boleta de energía, está constituido por la suma de tres subtotales: "Subtotal Energía", "Subtotal Impuestos y Contribuciones" y "Subtotal subsidio y/o Bonificación".
El subtotal energía está constituido por la suma de un Cargo Fijo mensual de $67.13 y un Cargo variable que se obtiene de multiplicar la mitad del consumo bimestral registrado, por el precio unitario correspondiente al KW/hora, que en la actualidad es de $0.8141.
Se necesita informar **a 1000 usuarios**, de los cuales se conoce los consumos bimestrales del mismo período en los dos últimos años, el importe a pagar y en caso de haber sido sancionados especificar el valor del recargo aplicado.

Análisis del problema
Como se conoce la cantidad de usuarios (1000) a los cuales se desea informar, se debe utilizar una estructura que permita ingresar los datos de todos los usuarios.
Por lo tanto, para el ingreso de los datos, será necesario utilizar una estructura que repita el proceso para los 1000 usuarios.
El algoritmo principal que corresponde a esta situación es:

```
Comienzo //----Algoritmo principal----
  real consumo_actual,consumo_anterior
  constante N=1000 // se declara la constante con la cantidad de usuarios conocida
  entero i
        Para i  desde 1 hasta N
Cuerpo    Escribir "Usuario:  ", i
del       Escribir "Ingrese consumos bimestrales  actual y del mismo periodo año anterior "
Ciclo     Leer consumo_actual, consumo_anterior
          s=SubtEnergía (consumo_actual)
          Mensaje (consumo_actual, consumo_anterior, s)
        FinPara
Fin
```

Nota: los subprogramas son los mismos que se escriben en la actividad 1.

Ejemplo 5
Retomando la actividad 1, pero suponiendo que la cantidad de usuarios es un dato que Energía San Juan conoce pero puede variar, no es una cantidad fija, constante. El enunciado del problema sería el siguiente:

Problema: La emergencia energética por la que atraviesa el país debido a la falta de inversión de las empresas responsables de esa área en los últimos años, ha llevado a que en la actualidad deba tomarse medidas que afectan directamente al usuario. Energía San Juan ha implementado un sistema de castigos y ha puesto en vigencia una disposición que consiste aplicar a aquellos usuarios que incrementaron su consumo, respecto al mismo bimestre del año anterior, en más de 100KW, un recargo del 20% al subtotal correspondiente a energía.
El total a pagar que figura en una boleta de energía, está constituido por la suma de tres subtotales: "Subtotal Energía", "Subtotal Impuestos y Contribuciones" y "Subtotal subsidio y/o Bonificación".
El subtotal energía está constituido por la suma de un Cargo Fijo mensual de $67.13 y un Cargo variable que se obtiene de multiplicar la mitad del consumo bimestral registrado, por el precio unitario correspondiente al KW/hora, que en la actualidad es de $0.8141.
Se necesita informar **a los usuarios, de los cuales se conoce la cantidad** y de cada uno de ellos el consumo bimestral del mismo período en los dos últimos años, el importe a pagar y en caso de haber sido sancionados especificar el valor del recargo aplicado

Análisis del problema
La cantidad de usuarios se conoce, pero puede variar, no es una cantidad fija, constante. Se puede utilizar una estructura Para _Fin para que permita ingresar los datos de todos los usuarios, pero ingresando previamente la cantidad de usuarios.

El algoritmo que corresponde a esta situación es:

Comienzo //----Algoritmo principal----
real consumo_actual,consumo_anterior
entero i, CantUsuarios
Escribir "Ingrese cantidad de usuarios: "
Leer CantUsuarios
Para i desde **1** hasta **CantUsuarios**
Cuerpo del Ciclo
Escribir "Usuario: ", i
Escribir "Ingrese consumos bimestrales actual y del mismo periodo año anterior "
Leer consumo_actual, consumo_anterior
s=SubtEnergía (consumo_actual)
Mensaje (consumo_actual, consumo_anterior, s)
FinPara
Fin

Nota: los subprogramas son los mismos que se escriben en la actividad 1.

Los siguientes son ejemplos del uso de esta estructura iterativa Para - FinPara

Ejemplo 6
Escribir el triple de los números pares comprendidos entre 200 y 440.

```
Algoritmo Calculo
void Triple (entero xvi, entero xvf)
Comienzo
entero c
  Para  c  desde xvi hasta xvf
    Si ((c resto 2) == 0)
      Entonces Escribir "el triple   de", c ,  "es" , c*3
    Finsi
  FinPara
Fin

Comienzo //--Algoritmo principal--
entero vi, vf
vi=200 // se inicia el valor inicial y final
vf=440
Triple (vi, vf)
fin
```

ACTIVIDAD 2
Modique el Ejemplo 6 para que pueda procesar varios rangos de direntes valores de inicio y fin.

ACTIVIDAD 3
a - Realice un subprograma que muestre la tabla de multiplicar del número 6.
b - Realice un algoritmo que muestre la tabla de multiplicar de un número cuyo valor es ingresado por teclado.

Ejemplo 7
Dado K números enteros positivos N, escribir sus divisores.

```
Algoritmo NumDivisores
void Divisor (entero XN)
Comienzo
entero c
    Para  c desde 1 hasta XN
      Si ((N resto c) == 0)
         Entonces Escribir c
      FinSi
    FinPara
Fin

Comienzo //--Algoritmo principal--
Comienzo
entero N,K
  Leer K
  Para  c desde 1 hasta K
    Leer N
    Divisor (N/2)
  finpara
```

fin

ACTIVIDAD 4
Realice la traza cuando el algoritmo del ejemplo 7 se ejecuta para K=2 y N=16 y 4

Estructura Mientras - FinMientras

La estructura Mientras FinMientras es una estructura iterativa, en la que el número de veces que se ejecuta el cuerpo del ciclo; depende del valor de verdad de una condición que se evalúa al comienzo del ciclo. La condición es una expresión relacional o lógica cuya evaluación es un valor booleano.
El seudocódigo asociado es:

Mientras (<condición>)
Cuerpo del Ciclo
acción 1
acción 2
acción 3
⋮
acción n
FinMientras

El comportamiento de esta estructura es el siguiente, las acciones del cuerpo del ciclo se repiten mientras la condición es verdadera. Como en la condición participan variables cuyos valores cambian durante la ejecución del cuerpo del ciclo, en determinado momento (estado de variables) la condición es falsa, sale del ciclo y se continúa con la acción que sigue al FinMientras.

Como la condición se evalúa antes de la ejecución de las acciones del cuerpo del ciclo, se lo denomina bucle pretest, las mismas podrán ejecutarse cero o más veces, dependiendo del resultado de la condición.

El esquema correspondiente a su funcionamiento es:

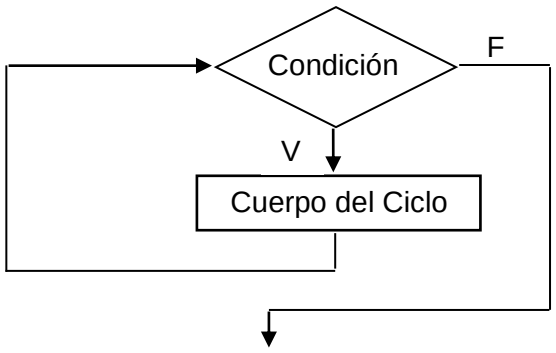


Figura 2.6 Estructura Iterativa Mientras-FinMientras

En esta estructura, no se conoce con anticipación el número de veces que se ejecutará el ciclo, por ello se la denomina estructura de repetición no definida. Se utiliza una **condición** en la que interviene una **variable centinela** para controlar la ejecución. La variable centinela que indicarà fin *de datos*, es elegida de manera que tome al final del ciclo un valor diferente a los valores de ejecución exitosa del ciclo. El valor que indica fin de datos se lo denomina **valor centinela**.

Ejemplo 8
Leer números positivos, mostrar los que son múltiplos de 5. El ingreso de valores finaliza cuando se lee el número cero.

```
Algoritmo Multiplo
Comienzo
        entero numero
        Escribir " Ingresar un número positivo, finalizar con 0"
        Leer numero          // Lectura Anticipada
        Mientras ( numero > 0 )
            Si ((numero resto 5) == 0)
                Entonces Escribir numero
            Finsi
            Escribir " Ingresar un número positivo, finalizar con 0"
            Leer numero
        FinMientras
Fin
```

Cuerpo del Ciclo

Condiciòn
(numero > 0)

Variable centinela: numero
Valor centinela : 0

Nota: en el ejemplo se resuelve el problema en el algoritmo principal.

En este ejemplo se realiza una lectura antes de entrar al Mientras - FinMientras para poder evaluar la condición la primera vez; esto se conoce como lectura anticipada. Además, antes de finalizar la estructura se realiza una nueva lectura, que permite el ingreso de un nuevo número para luego volver a evaluar la condición y así sucesivamente hasta que el número leído sea menor o igual 0, y termine la ejecución del bucle.
Es fundamental la lectura antes del Finmientras ya que de lo contrario el proceso se repetiría siempre con el primer valor leído. En este caso, si el primer número ingresado es positivo el proceso se repetirá indefinidamente, produciéndose un bucle infinito. La lectura del valor elegido como centinela (para este ejemplo el valor cero permite finalizar el ciclo.

Ejemplo 9
Considerando nuevamente el problema de la empresa Energía San Juan de la actividad 1, pero bajo el supuesto que se desconoce la cantidad de usuarios.

Problema
La emergencia energética por la que atraviesa el país debido a la falta de inversión de las empresas responsables de esa área en los últimos años, ha llevado a que en la actualidad deba tomarse medidas que afectan directamente al usuario. Energía San Juan ha implementado un sistema de castigos y ha puesto en vigencia una disposición que consiste aplicar a aquellos usuarios que incrementaron su consumo, respecto al mismo bimestre del año anterior, en más de 100KW, un recargo del 20% al subtotal correspondiente a energía.
El total a pagar que figura en una boleta de energía, está constituido por la suma de tres subtotales: "Subtotal Energía", "Subtotal Impuestos y Contribuciones" y "Subtotal subsidio y/o Bonificación".
El subtotal energía está constituido por la suma de un Cargo Fijo mensual de $67.13 y un Cargo variable que se obtiene de multiplicar la mitad del consumo bimestral registrado, por el precio unitario correspondiente al KW/hora, que en la actualidad es de $0.8141.
Se necesita informar **a un grupo de usuarios**, de los cuales se conoce los consumos bimestrales del mismo período en los dos últimos años, el importe a pagar y en caso de haber sido sancionados especificar el valor del recargo aplicado.

Análisis del problema
Como no se conoce la cantidad de usuarios del grupo al cual se desea informar, se debe utilizar una estructura que permita ir ingresando los datos mientras queden usuarios. Por lo tanto para el ingreso de los datos, será necesario determinar un variable y un valor centinela, para el problema dado, al consumo_actual puede ser la variable centinela y -1 el valor centinela que finalice el ciclo.
Esto permitirá ingresar datos de usuarios, hasta que se termine el grupo, e ingresar el valor 1 a la variable consumo_actual, para que se finalice el ciclo.
El algoritmo es:

Comienzo //----Algoritmo principal----
real consumo_actual,consumo_anterior
Escribir "Ingrese consumo bimestral actual"
Leer consumo_actual // Se elige como variable centinela
Mientras (**consumo_actual** <> -1)
Cuerpo del Ciclo
Escribir " Ingrese consumo bimestral correspondiente al mismo periodo del año anterior"
Leer consumo_anterior
s=SubtEnergía (consumo_actual)
Mensaje (consumo_actual, consumo_anterior, s)
Escribir " Ingrese consumo bimestral actual"
Leer consumo_actual //Se actualizan los datos, ingresando el consumo de otro usuario
FinMientras
Fin

Nota: los subprogramas son los mismos que se escriben en la actividad 1.

En los procesos iterativos, el control de la repetición de las acciones se realiza a partir de la información contenida en las variables. En el caso de la estructura Mientras, la condición actúa como variable de control del bucle, ésta debe tener por tanto un valor inicial y actualizarse, ya que si su valor no cambia se tendrá un bucle infinito**.**

ACTIVIDAD 5
Los siguientes algoritmos se ha escrito si usar subprograma con el objetivo de hacer diferentes analisis.
Realizar la traza de los siguientes algoritmos e indicar:

1. ¿Cuántas veces se ejecuta el cuerpo del ciclo?
2. ¿Qué valor de la condición permite que se itere el ciclo?
3. ¿Qué valor de la condición permite que finalice el ciclo?
4. ¿Cuántas veces se ejecutará, como mínimo, el cuerpo del ciclo del algoritmo, para cualquier lote de prueba?

a) Algoritmo **Receta**
Comienzo
cadena nombre
entero calorias
Escribir "Ingrese nombre del ingrediente de la receta, finalice con FIN"
Leer nombre
Mientras (nombre!="FIN")
Escribir "Ingrese calorías del ingrediente"
Leer calorias
Si (calorías > 50)
Entonces Escribir "Intente reemplazar el ingrediente ", nombre
Finsi
Escribir "Ingrese nombre del ingrediente de la receta, finalice con FIN"

```
   Leer nombre
  FinMientras
Fin
```

***Lote de prueba**:* Azúcar 30 , Manteca 80, Huevos 55, Harina 35, Leche 25, FIN

b) Algoritmo **Control**

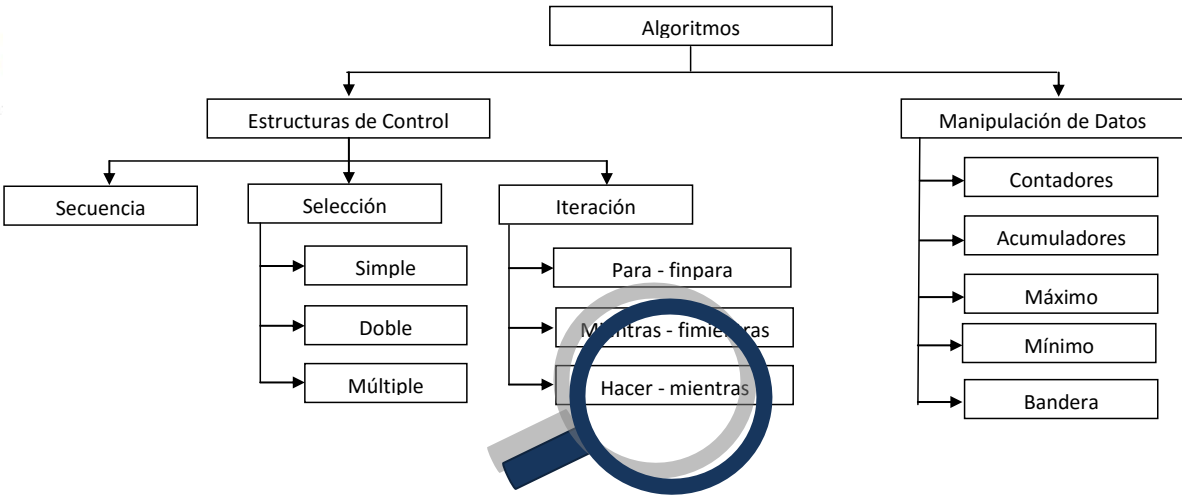
```
Comienzo
entero d, c
booleano b
b = verdadero
d = 0
Leer c
Mientras  ( (c * 5 - 15 < > 0 ) y (b==verdadero))
  d=d+2
  Si  (d  - 7 < 2)
     Entonces Escribir "El valor de d es:", d
                  Escribir  d + c
     Sino    b=falso
  finsi
  Leer c
Fin mientras
Escribir "valor de d es:", d
Fin
```

***Lote de prueba*:** 1, 0, 4, -5, 3

ACTIVIDAD 6

Construir un algoritmo que tenga subprogramas que:

a) Leer pares de números naturales. Escribir para cada par su producto e indicar si la primera componente es múltiplo de la segunda. La lectura finaliza cuando el par leído tiene ambas componentes nulas.
b) Dada la ecuación -3x +y +4=0, leer pares de números enteros e indicar cuáles son raíces. El ingreso de pares finaliza cuando ambas componentes son cero.
c) Construir el algoritmo que resuelve el problema.
d) Leer un conjunto de 150 números e indicar si está o no ordenado en forma descendente.



Estructura Hacer - Mientras

En esta estructura iterativa, el número de veces que se ejecuta el conjunto de acciones depende del valor de verdad de una condición que es evaluada al final del ciclo.
En este caso el cuerpo del ciclo se ejecuta 1 o más veces.
El seudocódigo que caracteriza esta Acción es:

```
Hacer
        acción 1
        acción 2
          ⋮
        acción n
Mientras(<condición>)
```

El esquema de esta estructura de repetición es:

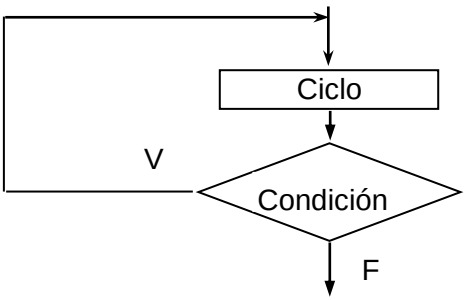


Figura 2.7 Estructura Iterativa Hacer - Mientras

Ejemplo 10
El siguiente algoritmo lee dos notas y muestra el promedio hasta que dicho promedio sea mayor a 10.

```
real Promedio (real a, real b)
Comienzo
  real resultado
  resultado= a+b/2
  retorna (resultado)
fin
Comienzo //----Algoritmo principal----
real  n1,n2
Hacer
        Leer  n1,n2
        p = Promedio (n1,n2)
        Escribir   "el promedio es: " p
Mientras  (p <= 10)
Fin
```

Realizar la traza del algoritmo para los pares de valores, (7,9) (8,10) (5,7) (11,11) (8,9) y responder:

ACTIVIDAD 7
¿Se muestra el promedio del último par de notas leídas? ¿Por qué?
¿Qué valor de la condición permite que se itere el ciclo?
¿Qué valor de la condición permite que finalice el ciclo?

¿Cuántas veces se ejecutará el ciclo?
¿Cuántas veces se ejecutará, como mínimo, el ciclo del algoritmo, para cualquier lote de prueba?

Ejemplo 11

```
Algoritmo Pares
entero Resto (entero n)
Comienzo
  real resultado
  resultado= n - (n div 2) * 2
  retorna (resultado)
fin

Comienzo //----Algoritmo principal----
entero nro
caracter opcion
Hacer
  Leer  nro
  Si  (Resto== 0)
        Entonces Escribir  "El número es par"
        Sino     Escribir  "El número es impar"
  Finsi
  Escribir "¿Desea ingresar más datos? (S/N)"
  Leer opcion
Mientras  ((opcion == 'S' )  o ( opcion == 's'))
Fin
```

2.5 Bucles Anidados

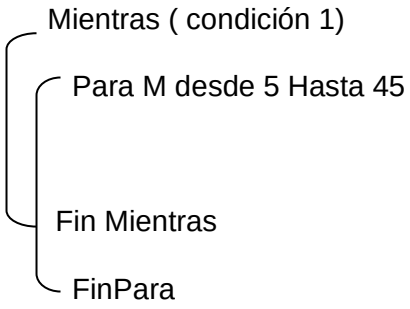
Así como es posible encontrar un Si dentro de otro Si, dando origen a las estructuras alternativas anidadas, se pueden anidar las distintas estructuras iterativas con la restricción que la estructura interna debe estar totalmente incluida en la externa, evitando solapamientos.
El funcionamiento, para el caso de dos bucles anidados es el siguiente: por cada iteración del bucle exterior, se ejecuta completamente el interno. Se continúa de la misma manera hasta que el bucle exterior completa las iteraciones.
El mismo criterio rige para más de dos bucles anidados.

Los siguientes anidamientos son válidos:

```
Mientras ( condición 1)
  Para M desde 5 hasta 45


  FinPara
Fin Mientras
```

```
Hacer
   Para  j desde 15 hasta 1 con Paso -2

      Mientras ( condición 1)

       Fin Mientras

   FinPara
Mientras (condición 2)
```

No es válido el anidamiento siguiente, debido a que el bucle interno no se ha terminado de ejecutar cuando ocurre una nueva repetición del bucle externo:



Ejemplo 12
El siguiente algoritmo permite construir la tabla de multiplicar de los números 2 al 9

```
Algoritmo Tablas
Comienzo
entero i,j
Para i desde  2 hasta 9
  Escribir “Tabla del numero”, i
     Para j desde 1 Hasta 10
              Escribir  i ,  “ * “, j “ =” , i*j
     FinPara
 Escribir “  “
FinPara
Fin
```

Para interpretar el funcionamiento de los bucles se realizará el seguimiento para la obtención de la tabla del 2 y del 3.

i	j	SALIDA
2		Tabla del número 2
	1	2*1= 2
	2	2*2= 4
	3	2*3= 26
	:	:
	10	2*10= 20

i	j	SALIDA
3		Tabla del número 3
	1	3*1= 3
	2	3*2 =6
	3	3*3 =9
	:	:
	10	3*10=30

```
Comienzo
entero i,j
Para i desde  2 hasta 3
  Escribir “Tabla del numero”, i
     Para j desde 1 Hasta 10
              Escribir  i ,  “ * “, j “ =” , i*j
     FinPara
 Escribir “  “
FinPara
Fin
```

El mismo ejemplo escrito con subprograma seria:

```
Algoritmo TablasNum
void Tabla (entero n)
Comienzo
     Para j desde 1 Hasta 10
            Escribir  n ,  " * ", j " =" , n*j
     FinPara
    Escribir "  "
fin
Comienzo //----Algoritmo principal----
entero i,j
  Para i desde  2 hasta 9
    Escribir "Tabla del numero", i
    Tabla (i)
  FinPara
Fin
```

Nota: la traza del este algoritmo produce la misma salida que el anterior.

Ejemplo 13

El INPRES (Instituto Nacional de Prevención Sísmica) ubicado en la ciudad de San Juan ha dividido el continente en 7 zonas sísmicas. Necesita realizar un estudio y para esto cuenta con la información de los sismos registrados en los últimos 3 años. Por cada zona se ingresa en forma ordenada, según la fecha de ocurrencia, el nombre de la provincia, la magnitud del sismo registrado en escala Ritcher profundidad del epicentro.

Por cada zona, el ingreso de información finaliza con nombre de provincia NADA MAS.
Se necesita realizar un algoritmo que pemita mostrar por cada zona, el nombre de la o las ciudades en donde los sismos se produjeron con epicentros a una profundidad no mayor de 200 km y que superaron los 5 grados de magnitud.

Realizar la traza del algoritmo con el lote de prueba que más adelanta se presenta.

Representación de problema:

En este tipo de problemas, para determinar la salida solicitada, conviene realizar una representación hipotética de los datos de entrada, tratando de considerar todos los casos que se pudieran presentar, aún los excepcionales.

Zona 1		
Florida	6.7	300
California	4.2	250
Carolina	5.9	180
California	3.8	400
Florida	7.1	200
Nevada	4.2	320
NADA MAS		

Zona 2		
Kobe	8	120
Kobe	5.3	90
Kobe	7.8	200
Kobe	6.1	150
Tokio	6.3	350
Yamaguchi	5.2	280
Kioto	4.9	320
Osaka	7.3	189
NADA MAS		

Zona 3
NADA MAS

Zona 7		
San Juan	5.1	622
San Juan	4.2	180
Mendoza	6	278
Córdoba	3.5	542
San Luis	3.7	345
Catamarca	3.1	453
Mendoza	4	190
NADA MAS		

Los datos hipotéticos indican que en la zona 1 se registraron 6 sismos, 8 en la 2, mientras que en la zona 3 no se registraron sismos

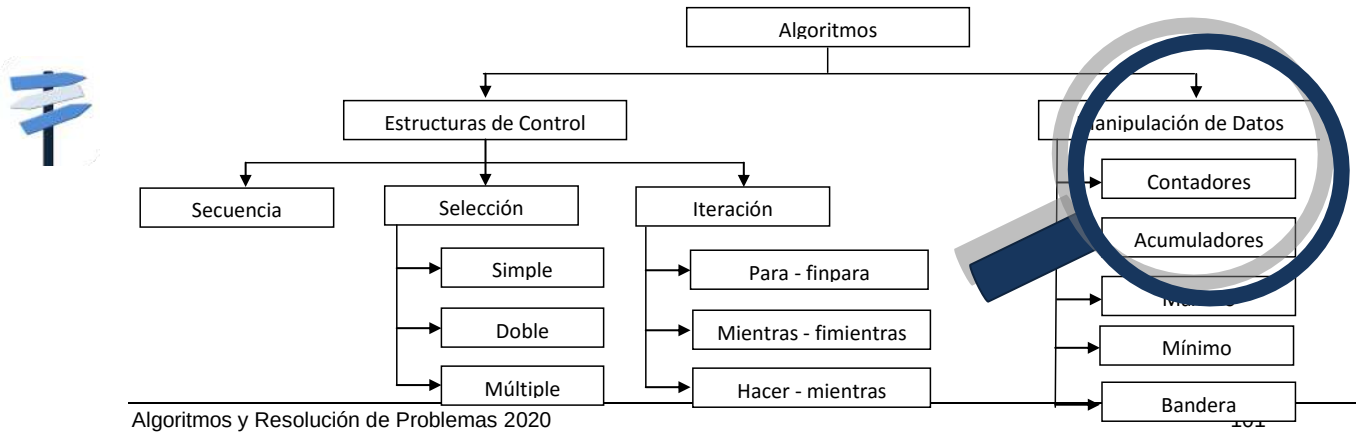
El proceso requiere el uso de dos estructuras anidadas:

- El bucle externo, para el tratamiento de cada zona, puede ser controlado por contador o variable de control que varía de 1 a 7, ya que se conoce de a priori que son 7 zonas. .. Conviene entonces, utilizar la estructura **Para**, para controlar el proceso de las zonas
- El bucle interno sin embargo, como la cantidad de sismos registrados no es conocida a priori y además es variable para las distintas zonas, se necesita el uso de una variable centinela que controle el ciclo.. Conviene entonces, utilizar la estructura **Mientras** para el tratamiento de los sismos de cada una de las zonas.

El algoritmo que puede resolver esta situación es:

Algoritmo **Sismos**

```
void Mensaje (cadena Nomprov, real m, real p)
Comienzo
Si  ((m > 5) y (p <= 200))
   Entonces
      Escribir "Provincia con sismo de magnitud > 5 y epicentro <= 200"provincia" , Nomprov
  Finsi
Fin

Comienzo //--Algoritmo principal--
entero zona, z
cadena provincia
real magni, prof
Para z desde 1 hasta 7
     Escribir " ZONA ", z
     Escribir " Ingrese  nombre de  provincia - Finalice  con NADA MAS"
     Leer  provincia
     Mientras (provincia !=" NADA MAS")
                Escribir " Ingrese  magnitud y profundidad"
                Leer magni, prof
                Mensaje (provincia, magni, prof)
                Escribir "Ingrese  nombre de  provincia - Finalice  con NADA MAS"
                Leer  provincia
     Fin Mientras
FinPara
Fin
```



2.6 Contadores y acumuladores

Muchos de los problemas que se presenta en la vida diaria aluden a situaciones en las que se debe contar o acumular valores.
Situaciones como las que se ven representadas en las siguientes tablas implican, por ejemplo, determinar la cantidad de equipos que intervienen en un campeonato (contar), calcular el total de goles realizados (acumular) ó determinar la magnitud promedio de los sismos registrados en una zona sísmica determinada (contar y acumular).

Equipo	Goles
Boca	21
River	20
Independiente	25
Racing	17
Estudiantes	23

Tabla 2.1

Provincia	Magnitud	Profundidad
San Juan	5.1	622
San Juan	4.2	180
Mendoza	6	278
Córdoba	3.5	542

Tabla 2.2

En este apartado trataremos de entender cómo realizar las operaciones de contar y acumular en un algoritmo
Un **contador** permite calcular la frecuencia de un evento. Por lo tanto, se necesita una variable entera para representarlo. Además, se debe inicializar con el valor 0 (cero) para luego ir incrementando en 1 (uno) cada vez que ocurre el evento que se desea contar.

```
entero conta          // declaración de variable contador
conta=0               // inicialización del contador
conta = conta + 1     //incrementa el valor del contador en una unidad, el resultado lo asigna nuevamente a la variable
```

Un acumulador es una variable numérica cuyo valor se va a ir incrementando en un valor que puede no ser fijo. En general se lo utiliza para ir almacenando sumas sucesivas de distintas iteraciones de un ciclo. De ahí que el acumulador se debe inicializar en 0 (cero).

Un **acumulador o sumador** es una variable de tipo entera o real, dependiendo de los valores a acumular, que permite almacenar sumas sucesivas.

El acumulador se debe inicializar en 0 (cero) y luego va incrementando su valor en una constante o variable.

En los algoritmos debe insertarse tres pasos:

```
entero v,acum         // declaración de variable
acum=0                // inicialización del acumulador antes del ciclo
acum = acum +v        // incrementa el valor del acumulador en el valor de la variable V dentro del ciclo
```

Ejemplo 14

El siguiente es el algoritmo que permite determinar la magnitud promedio de los sismos registrados en la zona sísmica de Argentina y la cantidad de sismos que se produjeron a una profundidad no mayor de 200 km.

Algoritmo **SismosProm**
Comienzo
entero C, ct // se necesitan 2 contadores
cadena provincia
real magni, prof, S // el sumador S tiene el mismo tipo de la variable que acumula magni

1. C=0 // inicialización del contador
2. S=0 // inicialización del acumulador
3. ct=0 // inicialización del contador de sismos de argentina
4. Escribir “Ingrese nombre de la provincia, finalice con NADA MAS”
5. Leer provincia
6. Mientras (provincia !=” NADA MAS”)
   7. Escribir “Ingrese magnitud y profundidad de sismos registrados en Ar gentina”
   8. Leer magni, prof
   9. S=S + magni // se acumula cada una de las magnitudes
   10. ct=ct+1 // se cuentan los sismos registrados
   11. Si (prof<= 200)
       . entonces C=C+1 // se incrementa el contador, si se cumple la condición especificada
       Finsi //
   12. Escribir “Ingrese nombre de la provincia, Finalice con NADA MAS”
   13. Leer provincia

FinMientras

14. Escribir “ Magnitud promedio de sismos registrados en Argentina”, S/ct // se muestran los resultados fuera de la estructura
15. Escribir “ Total de sismos con profundidad no mayor a 200 km” , C
16. Escribir “ Total de sismos ingresados” , ct

Fin

El siguiente es el seguimiento de este algoritmo considerando como lote de prueba el que se incluye en la tabla 1.8

C	ct	Provincia	magni	prof	S	Salida
0	0				0	Ingrese nombre de la provincia, finalice con NADA MAS
		San Juan				Ingrese magnitud y profundidad de sismos registrados en Argentina
			5.1	622		
					5.1	
	1					
						Ingrese nombre de la provincia, finalice con NADA MAS
		San Juan				Ingrese magnitud y profundidad de sismos registrados en Argentina
			4.2	180		
					9.3	
	2					
1						Ingrese nombre de la provincia, finalice con NADA MAS
		Mendoza				
			6	278		
					15.3	
	3					
						Ingrese nombre de la provincia, finalice con NADA MAS
		Córdoba				Ingrese magnitud y profundidad de sismos registrados en Argentina
			3.5	542		
					18.8	
	4					
						Ingrese nombre de la provincia, finalice con NADA MAS
		NADA MAS				
						Magnitud promedio de sismos registrados en Argentina , 4.29 Total de sismos con profundidad no mayor a 200 km 1
						Total de sismos ingresados 4

ACTIVIDAD 8
Responda

a) ¿Cuántas veces se ejecutan las acciones 7, 8, 9, 10 , 11, 12, 13? Justifique
b) ¿cuántas veces se ejecuta la acción 14? Justifique
c) ¿Porqué cuando se ejecuta la acción 11, en la tabla sólo una vez produce un cambio en el almacenamiento de una variable?.

ACTIVIDAD 9
Hacer un subprograma que reciba dos números enteros positivos **a** y **b**, calcule la potencia $\mathbf{a}^{\mathbf{b}}$ y devuelva el resultado.

ACTIVIDAD 10
Hacer un subprograma que reciba un número entero positivo, calcul su factorial y devuelva el resultado.
Recordar que el factorial se define como:

$$n! \begin{cases} 1 \text{ si } n=0 \\ 1*2*3*\ldots*n \quad \text{si } n>0 \end{cases}$$

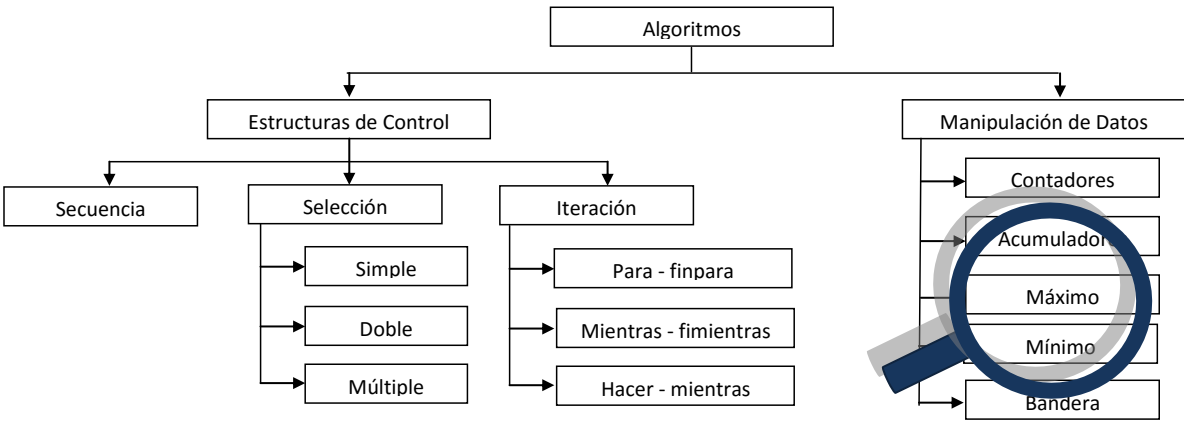
ACTIVIDAD 11
Una empresa ha realizado una encuesta de opinión para determinar la intención de voto para las próximas elecciones. Las personas fueron seleccionadas al azar.
La encuesta se realizó en los distintos departamentos de la provincia. Por cada persona encuestada se registra edad, sexo (F ó M) y código de intención de voto.
El código de intención de voto puede tomar los valores comprendidos en el rango (A..E): A: Frente para la Victoria; B: PRO; C: Frente Renovador; D: Otros; E: No sabe o no contesta.
En la sede central de la encuestadora se deben procesar los datos obtenidos, sabiendo que el ingreso de los mismos finaliza con edad =0.
Se pide informar:

- Total de encuestados.
- Porcentaje de votos para cada uno de los códigos considerados en la intención de voto (A..E).
- Edad promedio de los votantes por cada código de intención de voto.

Nota: Construir subprograma para el calculo de promedio y de porcentaje.

ACTIVIDAD 12
Una distribuidora mayorista posee 5 cajas. Se realiza un balance y como parte del mismo se desea obtener información a partir de los datos de las boletas confeccionadas a sus clientes, sabiendo que éstos se procesan ordenados por caja. En cada boleta registra tipo de cliente, importe total. El tipo de cliente puede ser: 1.Mayorista 2.Almacén 3.Minorista.
El ingreso por cada una de las cajas, finaliza con importe = 0.
Calcular los datos que permitan completar la siguiente planilla:

Caja 1		
	Importe total	Cantidad de boletas
Mayoristas		
Almacén		
Minorista		
TOTAL		
:		
Caja 5		
	Importe total	Cantidad de boletas
Mayoristas		
Almacén		
Minorista		
TOTAL		



2.7 Cálculo de máximos y mínimos de un conjunto de valores

Muchos problemas requieren determinar el menor o mayor valor de entre un conjunto de valores dados. Por ejemplo, si se desea indicar la menor nota obtenida por los alumnos que rindieron un examen de ingreso a una universidad, o encontrar la temperatura máxima registrada en una zona determinada, o el rango de valores entre los que se encuentran los sueldos de empleados de una empresa, es necesario determinar un proceso que permita generalizar este tipo de cálculos.

Para obtener el máximo, se requiere definir una variable del mismo tipo de los datos, la que puede identificarse con el nombre **max.** Esta variable debe ser inicializada con un valor pequeño, elegido de tal manera de asegurar que cualquier otro valor que se ingrese será superior a él. El cálculo consiste en un proceso iterativo en el que cada valor del conjunto es comparado con el de la variable **max,** sustituyéndolo en caso de ser superior, de esta manera al finalizar el bucle, en la variable **max** queda almacenado el mayor valor procesado.

Se debe notar que la elección del valor inicial como el menor de todos los posibles del lote, asegura que en la primera iteración, se almacene en **max** el primer valor leído, el cual será reemplazado cada vez que se procese un valor mayor.

Ejemplo 15
Una empresa de servicios realiza mensualmente el control de inasistencias de sus empleados, para lo cual cuenta con el nombre, documento y total de faltas de aquellos empleados que tuvieron inasistencias en el mes.

a) Se necesita conocer el nombre y documento del empleado que registró mayor cantidad de faltas, suponiendo único.
b) Se necesita conocer el nombre y documento del empleado que registró menor cantidad de faltas, suponiendo único.
c) Promedio de inasistencias de los empleados en ese mes.

Especificación de problema

Datos de Entrada	Salida	Proceso
Nombre Documento Cantidad de faltas	Nombre y documento del empleado con mayor cantidad de faltas.	Proceso iterativo: controlado por una condición Calcular y mostrar valor máximo

Es claro que, para procesar la información de los empleados con inasistencias, es necesario realizar un proceso iterativo. Como no se conoce el total de empleados en esta situación sería conveniente utilizar la estructura **Mientras** definiendo como condición el número de documento igual a cero, por ejemplo. Se debe notar también que para responder a la solicitud planteada se necesitan definir tres variables auxiliares, una que almacene la cantidad máxima de inasistencias, otra para almacenar el nombre y la última para guardar documento del empleado al que corresponde esa cantidad.

El algoritmo para resolver esta situación es:

```
Algoritmo Faltas
Comienzo
  cadena nomb, nomax            // declaración de variables
  entero cf,doc,documax,cfmax
  cfmax=0                       // se inicializa a variable con un valor pequeño
    Escribir "  Ingrese documento de empleado, finalice con 0"
    Leer doc
    Mientras ( doc <> 0)
         Escribir "  Ingrese nombre y cantidad de inasistencia en el mes "
         Leer nomb, cf
         Si ( cf > cfmax)
            Entonces cfmax= cf          // se modifica el valor del máximo por uno mayor
                     documax=doc        // se almacena la información del empleado
                     nomax= nomb
        Finsi
        Escribir "  Ingrese documento de empleado, finalice con 0"
        Leer doc
    FinMientras
    Escribir " El empleado ", nomax," documento", documax, " registró la mayor
              cantidad de faltas"
    Fin
```

Para los casos como el considerado, en que todos los datos son positivos, se puede seleccionar como valor centinela el cero o un número negativo. Pero si los valores a procesar pueden ser también negativos, como el caso de temperaturas, se deberá prestar atención al valor inicial seleccionado, se puede usar el primer valor leido

Es decir que para el cálculo del máximo o mínimo de un conjunto de valores, utilizando este procedimiento, es necesario conocer el rango posible de los datos a tratar.

El procedimiento para el ***cálculo del mínimo*** sigue el mismo criterio, solamente que el valor con que se inicializa la variable que contendrá el mínimo, deberá ser lo suficientemente grande de manera de asegurar que es mayor que cualquiera de los datos a procesar. El valor de esta variable se actualizará cada vez que el número procesado sea menor, de manera que al finalizar el bucle, la variable tendrá el menor de todos los valores.

ACTIVIDAD 13

Completar el algoritmo anterior resolviendo los ítems b) y c)

ACTIVIDAD 14

Realizar las modificaciones necesarias al algoritmo construido en la actividad 12 de tal manera que además se pueda indicar, el número de caja que registró la venta total máxima y el tipo de cliente que menos compras realiza a la distribuidora.

ACTIVIDAD 15

Defensa al Consumidor recibe numerosas quejas acerca del sabor a cloro en el agua que se consume en el gran San Juan. Por ello, ha decidido solicitar al departamento de química de la Universidad Nacional de San Juan, realizar un estudio del porcentaje de cloro en el agua, teniendo en cuenta 5 zonas.

Se tomaron 10.000 muestras en las que se registró número de zona y porcentaje de cloro. Se necesita conocer el rango de variación de los porcentajes encontrados y el número de zona al que corresponden dichos valores

ACTIVIDAD 16

Una ferretería mayorista tiene 5 zonas de reparto (1: “Centro”, 2:”Noreste”, 3: “Noroeste”, 4: “Sureste”, 5:”Suroeste”) y sus productos clasificados en tres rubros (1: “pinturas”, 2:”materiales de construcción”, 3: “herramientas”). Por cada zona se ingresan los siguientes datos de sus clientes: nombre, rubro de artículos y monto de la compra.

Se desea conocer por cada zona la cantidad de clientes, el nombre del cliente que realizó la mayor compra (suponiendo único) y el total recaudado por rubro.

Determinar además el total recaudado por la ferretería.

2. 8 Bandera Lógica

Hay situaciones en que solamente se quiere saber si un determinado evento sucedió. Por ejemplo, conocer si en un curso de primer año con 62 alumnos, alguno es mayor de 21 años, en ese caso, no interesa cuántos son mayores de 21, es suficiente que uno lo sea para considerar que el evento sucedió.

Para obtener esta información se deben analizar uno a uno los datos de todos los alumnos y cuando se encuentre quien cumpla con la condición se considera que el evento sucedió, y se deberá hacer una marca indicándolo. Cuando se termine de recorrer todos los datos de los alumnos se debe controlar la marca para conocer que ocurrió.

El siguiente código representa esta situación:

```
Algoritmo Edad

void Evalua (logico band) // subprograma que evalúa bandera
Comienzo
  Si band == verdadero // se evalúa la bandera para saber si el evento sucedió
  entonces
        Escribir "Hay algún alumno mayor de 21 años"
  sino
        Escribir "No hay ningún alumno mayor de 21 años")
  finsi
Fin

lógico Mayor21 (logico Bandera) // subprograma que lee e identifica una edad mayor a 21
Comienzo
  entero i, edad
  Leer edad
  Mientras (edad > 0) y (No Bandera)
    Si (edad > 21) // si sucede el evento la bandera cambia de valor
        Entonces bandera = verdadero
    FinSi
    Leer edad
  FinMientras
  retorna bandera
Fin

Comienzo //algortimo principal
   lógico ban
   band = falso // se asigna a la variable un valor inicial que señala que aun el evento no sucedió
   band = Mayor21(band) // se procesan datos para obtener el valor de bandera
   Evalua (band) // se evalúa el valor de bandera
Fin
```

El trabajo con bandera tiene tres momentos:

1) Se inicializa una variable lógica, que será utilizada como marca, previo al comenzar el proceso iterativo para analizar la ocurrencia del evento.
2) Durante la iteración, que permite recorrer todos los datos, cada vez que sucede el evento se cambia el estado de la marca.
3) Terminado el proceso iterativo, se debe analizar en qué estado quedó la marca, la cual señala si el evento sucedió o no.

2. 9 Cuadro comparativo de operaciones sobre conjuntos de datos

En un proceso iterativo, las operaciones de contar, acumular, maximo, mínimo y bandera tiene tres momentos: Inicio, Proceso, Fin, los cuales se realizan en puntos especificos respecto a las estructuras iterativas

Inicio
Comienzo Iteración
    Proceso
Fin Interación
Fin

En la siguiente tabla se generalizan estos momentos:

Inicio	Proceso	Fin
CONTADOR C=0	C=C+1	Se usa la variable C
ACUMULADOR A=0	A=A + Variable	Se usa la variable A

MAXIMO
 	
Max=0
	Si Max < Variable
	Entonces
	Max = Variable // actualiza
	[si se guardan valores relacionados se usan variables auxiliares: AUX = Variable2]
	finsi
		Se usa la variable Max [y auxiliares si se usaron]

MINIMO
 	
Min=99999
	Si Min > Variable
	Entonces
	Min = Variable // actualiza
	[si se guardan valores relacionados se usan variables auxiliares: AUX = Variable2]
	finsi
		Se usa la variable Min [y auxiliares si se usaron]

BANDERA Bandera = falso	Se identifica un evento Si CONDICION entonces bandera = verdadero finsi	Se evalua la bandera Si bandera == verdadero entonces // EL EVENTO SUCEDIÓ sino // EL EVENTO NO SUCEDIÓ finsi

Nota:
El problema de trabajar con conjuntos de datos como se ha realizado hasta ahora obliga a que la mayoria de las operaciones se realicen al momento de su lectura. Esto lleva a no poder hacer uso de subprogramas en muchos casos.
Una forma eficiente de manejar estos conjuntos es guardandolos en estructuras de datos adecuadas y despues de eso utilizarlos. Estas estructuras se llaman arreglos.

Práctico Unidad 2: Subprogramas

Resultados de Aprendizaje:

- Usar estructuras de control en forma simple y anidada.
- Capacidad para diseñar y construir subprogramas y algoritmos de manera óptima.
- Resolver situaciones problemáticas con iniciativa, autonomía y creatividad.
- Adquirir responsabilidad y compromiso por su propio aprendizaje.
- Desarrollar capacidad para comunicar sus puntos de vista en forma oral y escrita.
- Participar en grupos de trabajo, respetando las ideas de sus compañeros.

Ejercicio 1

Realizar el seguimiento de los algoritmos que se muestran a continuación.

A. *Lote de prueba* x =22, z = 12 x =39, z = - 11 x= -16, z=9

```
Algoritmo Paridad
void mensaje-paridad (entero xx , entero zz)
Comienzo
constante y=2
  SI ((xx + zz) resto y != 0 )
      entonces Escribir xx , “+”,  zz, “es  impar”
     sino  Escribir  xx , “+”,  zz, “es  Par”
  FinSi
retorna ()
Fin

/*----Algoritmo principal----*/
Comienzo
entero x, z
  Escribir “Introduzca primer valor (entero):"
  Leer  x
  Escribir “Introduzca segundo valor (entero): “
  Leer z
  mensaje-paridad (x,z)
Fin
```

B. *Lote de prueba* a =58, b = 50 a =25, b = - 50 a= - 2, b= - 5

```
Algoritmo Media
entero calculo (entero xa , entero xb)
Comienzo
entero c
   c= (a + b) / 2
retorna (c)
Fin

/*----Algoritmo principal----*/
Comienzo
entero a, b
  Escribir “Introduzca primer número (entero):"
  Leer  a
  Escribir “Introduzca segundo número (entero): “
```

```
  Leer b

  Si (( a > 0) y (b > 0 ))
       entonces  Escribir “La media aritmética de: ”, a, b,  “es”,  calculo (a, b)
       sino Escribir “uno de los dos números no es positivo ”
  Finsi
Fin
```

Ejercicio 2

Complete y responda:

A. Escribir un algoritmo que con subprogramas permita recibir un número y devuelva ‘P’ si el número es par o ‘I’ si es impar.

```
Algoritmo Calculo3
  …….  par_impar (entero xnum)
 Comienzo
  entero aux
  Si  ( xnum resto 2) == 0
      entonces aux = ´P´
      sino ….. = ´I´
  FinSi
 retorna (aux)
Fin

  /*----Algoritmo principal----*/
Comienzo
 entero n
 carácter PoI
  Escribir "Ingrese el número (sin decimales)"
  Leer n
  …  = par_impar (n)
  Escribir  “ el número ……..”, …..
Fin.
```

- ¿El valor del parámetro *xnum* se modifica con la acción *Si (xnum resto 2 == 0)*? Justifique.
- La condición de la selección en el subprograma ¿se puede expresar de otra manera para indicar que un número es par?

Ejercicio 3

Realizar un algoritmo con subprogramas que determine si alcanzan los bancos existentes en un aula, de no ser así, informar cuantos bancos es necesario agregar. Se cuenta con la cantidad de bancos del aula y la cantidad de alumnos inscriptos para ese curso.

Ejercicio 4

Un banco tiene un sistema la compra de dólares. Para la operación de compra se conoce la cantidad de dólares que se quiere comprar y el importe de venta de cada dólar. Esta operación tiene una comisión administrativa 5% sobre el importe total si la cantidad de dólares es más de 5000. Dicha comisión se adiciona al monto que el cliente debe pagar.

Realizar un algoritmo que usando subprogramas muestre un ticket indicando: la cantidad de dólares que se compraron, el valor de cada dólar, el monto de la comisión y el monto total que el cliente debe abonar por la compra.

Ejercicio 5
El dueño de una librería desea hacer una promoción, otorgando un descuento del 15% en las compras efectuadas por docentes o alumnos universitarios. Se conoce el tipo de cliente: 'Á': Alumno, 'D': Docente, 'O': Otro cliente y el importe total de la compra.
Realizar un algoritmo con subprogramas que permita mostrar el tipo de cliente y el importe a abonar por la compra.

Ejercicio 6
Complete adecuadamente los siguientes algoritmos y responda:

A. Crear un algoritmo con subprogramas que lea dos números y que escriba el menor de ellos.

```
algoritmo comparacion
 ……  resultado (entero xn1, entero xn2)
Comienzo
  Si (xn1….xn2)
      entonces Escribir "El menor es el primer número y su valor es : ",……
      sino Si (xn2….xn1)
               entonces Escribir  "El menor es el ……. número y su valor es ", xn2
               sino Escribir " Los números son ……"
           FinSi
  FinSi
retorna ( )
Fin

/*----Algoritmo principal----*/
Comienzo
real  n1, n2
  Escribir "Ingrese primer  número"
  Leer ……..
  Escribir "Ingrese segundo……."
  Leer n2
  resultado (n1,n2)
Fin
```

• ¿Que se debería modificar en el algoritmo para que calcule el mayor de dos números?

B. Realizar un algoritmo que dados la cantidad de varones y de mujeres de un curso, indique si hay mayor cantidad de mujeres y calcule el porcentaje de varones.

```
algoritmo calculo
 ……  compara (entero xm, entero xv)
Comienzo
  Si (xm….xv)
      entonces Escribir "Hay más mujeres y son",……
      sino Si (xm….xv)
               entonces Escribir  " Hay ……..…. y son  ",xv
               sino Escribir " Hay ……"
           FinSi
  FinSi
retorna ( )
```

```
Fin

entero calculo (entero xm , entero xv)
Comienzo
entero porc
   porc= ((xv *100) / (xv+xm))
retorna (porc)
Fin

/*----Algoritmo principal----*/
Comienzo
entero  m , v
 Escribir  "Ingrese cantidad de varones del curso"
 Leer v
 Escribir  "Ingrese la cantidad de mujeres del curso "
 Leer….
 compara (m, v)
 Escribir "el Porcentaje de varones es ", calculo (m,v)
Fin
```

• ¿El valor de las variables v y m se modifican con la acción *Escribir …..calculo (v, m)*?

C. Escribir un algoritmo que indique el tipo de triángulo (isósceles, equilátero, escaleno) que forman tres valoresingresados por teclado. Además debe comprobar que los números realmente formen un triángulo, y en caso contrario emitir un mensaje de error

Nota: Para que se pueda construir un triángulo los valores de cada uno de sus lados deben ser menor que la suma de los otros dos y mayor que la diferencia.
Es decir, por ejemplo si *a, b y c* son los lados del triángulo entonces: $a - b < c < a + b$

```
algoritmo triangulo
void ……. (entero xx, entero xm, entero …)
Comienzo
  Si (((xx + xm ) > z) Y (  (xx + xz)  > xm ) Y ( (xm + xz) > xx))      // Comprobación de que los
                                                                        // lados forman un triángulo
      entonces Si (xx==xm ) Y (xx==xz)
                  entonces Escribir" ……..……. "
                  sino Si (xx==xm ) O (xx==xz ) O (xm==xz)
                           entonces Escribir  "El triángulo es Isósceles"
                           sino Escribir"……..……."
                       FinSi
               FinSi
      sino Escribir "Esos números no forman un triángulo"
  FinSi
retorna ( )
Fin

/*----Algoritmo principal----*/
Comienzo
entero x, m ,z
  Escribir  "Ingrese valor del primer lado (numero positivo)"
  Leer x
```

```
  Escribir  "Ingrese valor del segundo lado (numero positivo)"
  Leer m
  Escribir  " Ingrese valor del tercer lado (numero positivo)"
  Leer …..
  comprobar (x, …, z)
Fin
```

• ¿Se modifica el valor de los parámetros xx, xm, xz; al realizar la acción: *Si (((xx+xm)>xz) Y ((xx+xz)>xm) Y ((xm+xz)>xx))* ?

D. Construir un algoritmo que muestre en forma decreciente tres números ingresados por teclado.

```
algoritmo posicionar
void  situar (entero xa, entero …, entero xc)
Comienzo
  Si ((xa < xb) Y (xa < xc))
     entonces Si (xb < xc)
                  entonces Escribir  "En orden decreciente los valores son  ",  xc, xb, xa
                  sino Escribir  "……..……",  xb, xc, xa
              FinSi
     sino  Si ((xb < xa) Y (xb < xc)
              entonces  Si (xa < xc)
                           entonces Escribir  "……", …,  …, …
                           sino Escribir  " ……..……..", …, …, …
                        FinSi
              sino  Si (xa < xb)
                        entonces Escribir  "……..……", …,  …, ….
                        sino  Escribir  "……..", …, …,  ….
                     FinSi
          FinSi
  FinSi
retorna ( )
Fin
/*----Algoritmo principal----*/
Comienzo
real   a, b, c
 //Lectura de datos
  Escribir  "Ingrese el primer número"
  Leer   a
  Escribir  "Ingrese el ……"
  Leer  ……
  Escribir  "……"
  Leer  ……
  situar ( a, b, c)    //Comparación  de datos
Fin
```

• Modificar las condiciones de las selecciones para que el algoritmo muestre los valores en orden ascendente o creciente

Ejercicio 7

Escriba un enunciado que utilice subprogramas modificando la estructura de a cada uno de los siguientes algoritmos, para ello realice las modificaciones y adecuaciones necesarias y convenientes.

A.

```
Algoritmo sensor
Comienzo
  entero temperatura
  Leer temperatura
  Si (temperatura >= 96)
      entonces Escribir “Temperatura muy alta: mal funcionamiento”
      sino Si ((temperatura > 50) y (temperatura < 95))
                entonces Escribir “Rango normal”
                sino Si (temperatura > 35)
                          entonces Escribir “Bajo el rango normal”
                          sino Escribir “Muy frío: apague equipo”
                     FinSi
            FinSi
  FinSi
Fin
```

B.

```
Algoritmo Intervalo
Comienzo
entero a, b, x
  Leer a, b, x
  Si ((x >= a) y (x <= b))
      entonces Escribir  x , “pertenece al intervalo cerrado  [ ", a, b, “] “
      sino  Si (x < a)
                entonces Escribir “el valor es menor que el límite inferior "
                sino Escribir “el valor es mayor que el límite superior "
             Finsi
  Finsi
Fin
```

C.

```
Algoritmo Ecuación
Comienzo
entero a, b, c, s
  Leer a, b, c
  Si (a <> 0)
      entonces Escribir “la ecuación es de grado dos"
                s = (b * b) - ( 4 * a * c))
                Si (s > 0)
                    entonces Escribir “las raíces son números reales y distintos"
                    sino Si (s = 0)
                              entonces Escribir “las raíces son número reales coincidentes"
                              sino Escribir “las raíces son números complejos conjugados"
                         Finsi
                Finsi
```

```
  Finsi
Fin
```

D.

```
Algoritmo calculo
Comienzo
Entero anio
booleano bisiesto
  Leer anio
  Si (anio<= 0)
     entonces  Escribir “El año ingresado no es válido"
     sino  Si (anio resto 4 == 0)
                 entonces Si (anio resto 400 == 0)
                                entonces bisiesto=Verdadero
                                sino  Si (anio resto 100 == 0)
                                            entonces bisiesto=Falso
                                            sino bisiesto=Verdadero
                                       Finsi
                            Finsi
                sino bisiesto=Falso
              Finsi
     Si bisiesto
          entonces Escribir “El año”, anio, “ SI es Bisiesto"
          sino Escribir “El año”, anio, “NO es Bisiesto"
     Finsi
  Finsi
Fin
```

Ejercicio 8

Una distribuidora de gaseosas posee 5 sucursales codificadas: 1: Zona Centro; 2: Zona Sur; 3: Zona Este; 4: Zona Oeste; 5: Zona Norte.

Se leen los datos de una factura: número de sucursal en la que se realizó la venta, importe, fecha de vencimiento y día de pago.

Realizar un algoritmo que usando subprogramas permita:

1. Si la factura es de la sucursal 2 o 5, escriba el nombre de la sucursal y el importe a cobrar, sabiendo que si la fecha de pago es posterior a la de vencimiento, se tendrá un recargo del 1,5%.
2. Si la sucursal es 1, 3 o 4, muestre el nombre de la sucursal, y el importe a cobrar, sabiendo que si la fecha de pago es igual o menor a la de vencimiento, tendrá un descuento del 2,5%.

Ejercicio 9

Realizar el seguimiento de los siguientes algoritmos.

Lote de prueba numero: 7 numero: 5

```
algoritmo Numero_del_dado
cadena num-a-letra (entero xnum )
Comienzo
cadena letras
  Segun( numero )
     1: letras = "seis"
     2: letras = "cinco"
     3: letras = "cuatro"
     4: letras = "tres"
     5: letras = "dos"
```

```
      6: letras = "uno"
  FinSegun
Retorna (letras)
Fin
/*----Algoritmo principal----*/
Comienzo
entero numero
cadena letras
Comienzo
   Escribir "Introduzca número del dado: "
   Leer  numero
   Si ((numero >= 1) y (numero <= 6))     /* Sólo si el número es válido, se ejecuta la acción*/
     entonces Escribir  "En la cara opuesta del dado está el ", num-a-letras (numero)
     sino Escribir  "ERROR: Número incorrecto."
    Finsi
Fin
```

Lote de prueba a=2, b=3, c=1, opcion=5

```
algoritmo Superficies
void Menu ( )
Comienzo
    Escribir " Ingrese    "
    Escribir "1 - Superficie del Rectángulo"
    Escribir "2 - Superficie del Cuadrado"
    Escribir "3 - Superficie del Triángulo"
    Escribir "4 - Superficie del Trapecio"
    Escribir "5 - Superficie del Rombo"
    Escribir "6 - Superficie del Círculo
Retorna ( )
Fin
void calculo-sup (entero xa, entero xb)
Comienzo
entero result
 result = xa * xb
Retorna (result)
Fin
/*----Algoritmo principal----*/
Comienzo
Constante entero Pi=3.14
entero a, b, c, opcion
  Menu ( )
  Leer opcion
  Según opcion
    1: Escribir "Ingrese valor de la base del rectángulo"
       Leer a
       Escribir "Ingrese valor de la base del rectángulo"
       Leer b
       Escribir "la superficie del rectángulo es", calculo-sup (a, b)

    2: Escribir "Ingrese valor del lado del cuadrado"
       Leer a
```

Escribir "la superficie del cuadrado es", ***calculo-sup (a, a)***

3: Escribir “Ingrese valor de la base del triángulo”
Leer a
Escribir “Ingrese valor de la altura del triángulo”
Leer b
Escribir "la superficie del triángulo es", (***calculo-sup (a, b)***) / 2
4: Escribir “Ingrese valor del lado1 del trapecio”
Leer a
Escribir “Ingrese valor del lado2 del trapecio”
Leer b
Escribir “Ingrese valor del lado3 del trapecio”
Leer c
a = a + c
Escribir "la superficie del trapecio es", (***calculo-sup (a, b)***) / 2
5: Escribir “Ingrese valor de la diagonal1”
Leer a
Escribir “Ingrese valor de la diagonal2”
Leer b
Escribir "la superficie del tombo es", (***calculo-sup (a, b)***) / 2
6: Escribir “Ingrese valor del radio”
Leer a
Escribir "la superficie del círculo es", Pi* ***calculo-sup (a, a)***
FinSegun
Fin

• Explique los beneficios de reutilizar un subprograma y porque es posible hacerlo.

Ejercicio 10
Para realizar una recarga virtual de una determinada empresa de telefonía celular, se ingresan por teclado los datos correspondientes al número de celular y el importe de la recarga. Considerar que si el importe de la recarga es mayor o igual a $50, el importe acreditado se incrementará en un 20%, si la recarga es igual o superior a $70, el incremento es del 50% y si la recarga supera los $90 el incremento será del 100%.
Realizar un algoritmo que utilizando subprogramas muestre el número de celular y el importe total acreditado de la recarga.

Ejercicio 11
En un vivero se comercializan semillas de diferentes flores. Cada una de ellas tiene una época del año en la que puede ser cultivada.
La siguiente tabla indica en qué mes se puede cultivar cada una:

Flor	N° de Mes
rosa	1..3
camelia	4
clavel	5
lirio	6..7
azucena	8
dalia	9..12

Por teclado se ingresa el número del mes y el nombre de la semilla.
Realizar un algoritmo que mediante subprogramas muestre si es posible o no cultivarla en el mes ingresado, en caso de no ser posible decir que flor se puede cultivar en ese mes.

Ejercicio 12

El encargado del planetario desea que se diseñe un algoritmo con subprogramas, que al ingresar el número de día de la semana, indique el nombre del día y el astro que da origen a ese nombre.

La siguiente tabla muestra la relación astro-día.

Día	Nombre del Astro
Domingo	Sol
Sábado	Saturno
Viernes	Venus
Jueves	Júpiter
Miércoles	Mercurio
Martes	Marte
Lunes	Luna

Ejercicio 13

Dado el siguiente algoritmo que permite calcular la potencia de un número entero:

a. Realizar su seguimiento con los siguientes lotes de prueba: b=7 e=4 y b= - 2 e=5

```
Algoritmo  Calculo1
  entero  potencia( entero base, entero exponente)
  Comienzo
  entero p, i
    p=1
    Para i desde 1 hasta exponente
       p = p * base
    FinPara
    retorna (p)
  Fin
  /*----Algoritmo principal----*/
  Comienzo
  Entero b, e
    Leer b
    Leer e
    Si  (e > 0)
      entonces Escribir “La potencia de base:”, b, “y exponente”, e, “es:”, potencia (b, e)
    FinSi
  Fin
```

b. Completar el siguiente texto:

El identificador del subprograma es ……..…….
Los parámetros formales son……
Las variables locales al subprograma son……..
La acción que llama al subprograma es ……..…… y se encuentra dentro de ……
Los parámetros actuales son……..
El subprograma retorna al algoritmo principal ……, a través de la acción…….. el tipo de dato que retorna es ……. y se declara en ……
La salida se realiza en ……..

c. Modificar el Algoritmo Calculo1 que permite calcular la potencia, para que los resultados se muestren en el mismo subprograma que realiza el cálculo de la potencia y tenga en cuenta cuando el exponente es cero o el exponente es menor que cero.
Explique mediante un texto las modificaciones realizadas.

Ejercicio 14

a. Completar el siguiente algoritmo de modo que calcule el factorial de un número natural.

```
Algoritmo Calculo2
…… factorial ( ……..)
Comienzo
Entero p, i
p=1
Para i desde 1 hasta ……..
   …….
Finpara
Escribir "El factorial de ", ……, "es" ……..)
retorna (……)
Fin

/*----Algoritmo principal----*/
Comienzo
  Entero n
  Leer n
  Si (n ….. )
     entonces …….
     sino ……
  FinSi
Fin
```

b. Realizar su seguimiento con los siguientes lotes de prueba: n=3 n=7 n= -12 n=0

Ejercicio 16

Una farmacia procesa la información de 30 facturas, de cada una ingresa el número de factura y el importe total de la misma.
Diseñe un algoritmo que usando subprogramas permita mostrar el número de cada factura cuyo importe se encuentre entre los dos valores ingresados por teclado**.**

Ejercicio 17

Se procesa la información de N apuestas realizadas en una agencia de quiniela, por cada apuesta se ingresa el número y el importe.
Se pide, realizar un algoritmo que con subprogramas permita:

1. Indicar la cantidad de apuestas con importes entre dos valores ingresados por teclado.
2. Escribir el número y el importe de las apuestas mayores a $1000.

Ejercicio 18

Se cuentan con los datos de 20 temperaturas tomadas a las 8 horas en Jáchal durante el mes de febrero. Realizar un algoritmo que usando subprogramas permita calcular la temperatura promedio del mes.

```
algoritmo temperaturas
entero sumar (entero xtemp  entero xs )
Comienzo
  xs=xs+xtemp
retorna (xs)
```

```
Fin
/*----Algoritmo principal----*/
Comienzo
entero s, c, i, temp
real prom
  s = 0                                  //inicialización sumador
  Para  i desde 1 hasta 20
     Escribir  "Ingrese Temperatura ", i
     Leer temp
     s= sumar (temp,s)                   // proceso sumador
  FinPara
  prom= s / 20                       // uso  sumador
  Escribir "El promedio de temperaturas es: ", prom
 Fin
```

• Justificar porque las acciones *s = 0* y *prom = s/20* se realizan fuera de la iteración

Ejercicio 19

Analice y complete para que el algoritmo permita: leer una serie de números naturales que finalizará cuando se ingrese "-999" y usando subprogramas pueda:

a. Indicar la cantidad de números de la serie leída que se encuentran comprendidos en un intervalo cerrado cuyos extremos son ingresados por teclado.
b. Calcular el máximo valor dentro de la serie de números leída.

```
algoritmo serie
Comienzo
entero ***calculomaxi*** ( entero xnum  entero xmax)
    Si ( ….>xmax)
        entonces xmax = xnum         //cálculo valor máximo
    FinSi
Retorna (xmax)
Fin

entero ***calculocontar*** (entero xi, entero xs)
Comienzo
entero max, min, cont
  ... = 0                                //inicialización contador
  max= -1000                             //inicialización máximo (un número muy pequeño)
  ….. = 1000000000                          //inicialización mínimo (un número muy grande)
  Escribir  "Ingrese un número (para finalizar ingrese -999) "
  Leer num
  Mientras (num!= ….. )
    Si (num > = inf) Y ( …. < = ….) .     // verificar si el número está dentro del intervalo
                          entonces cont = cont + 1       // proceso contador
    FinSi
    max=***calculomaxi***(num,max)    //cálculo valor máximo
   Escribir  "Ingrese un número (para finalizar ingrese -999) "
    Leer num
  FinMientras
Escribir "El valor máximo en la serie leída es", max
Retorna (cont)
Fin
```

```
/*----Algoritmo principal----*/
Comienzo
entero num, inf, sup
  Escribir  “Ingrese extremo inferior del intervalo “
  Leer inf
  Escribir  “Ingrese extremo superior del intervalo “
  Leer ….
  cont=calculo (inf,sup)
  Escribir “Hubieron” …  “números de la serie dentro del intervalo comprendido entre”,  inf , “ y
  “, …..
Fin
```

- Explicar ¿por qué se usa la acción *Leer num* antes de iniciar la iteración del Mientras y antes del FinMientras?
- Indicar porque la variable usada para calcular el máximo se debe inicializar en un valor pequeño e indique en que valor se inicializa el mínimo.

Ejercicio 20

En un curso de informática se han realizado dos exámenes diferentes, A y B, entre sus 50 alumnos (alumnos con registro impar, examen A; alumnos con registro par, examen B).
Se desea conocer si hubo algún aplazado en los exámenes.

```
algoritmo notas
…….. evaluar (entero xnota)
Comienzo
entero
  Si (nota< = 3)
      entonces band = Verdadero      // proceso bandera (cambia si hay al menos un aplazo)
  FinSi
Retorna (band)
Fin

/*----Algoritmo principal----*/
Comienzo
entero reg, nota, i
boolean band
  band = Falso                          //inicialización bandera para buscar aplazados
  Para  i desde 1 hasta ….
    Escribir  “Ingrese Registro del alumno“, i
    Leer reg
    Escribir  “Ingrese nota del alumno con registro“, ……
    band = evaluación (nota)
  FinPara
    Si ( band )       // es equivalente a band == Verdadero       // evaluación bandera
      entonces Escribir  “ Si Hubo algún examen con aplazo”
      sino Escribir “ ……..    examen con aplazo”
  FinSi
Fin
```

- Indicar si la variable *band* puede ser de otro tipo de dato.
- En caso de cambiar el tipo de dato de la variable *band* ¿Cuáles serían las modificaciones en el algoritmo?
- Explique bajo qué condiciones es conveniente utilizar la bandera.

Ejercicio 21
Realizar el seguimiento del algoritmo, determinando el resultado para su lote de prueba.
Lote de prueba num=5 num= 12

```
algoritmo primo
Comienzo
entero num, cont, aux
  Leer num
  cont = 0
  aux= 1
  Mientras ( aux<= num )
    Si ((num resto aux) == 0)
         entoncescont = cont + 1
    finsi
    aux = aux + 1
  fin mientras
  Si (cont = = 2)
     entonces Escribir num, “Es primo”
     sino Escribir num, “No es primo”
  FinSi
Fin
```

¿Qué otra estructura iterativa puede utilizarse para hacer este mismo proceso?
Realice la optimización del anterior algoritmo.

Ejercicio 19
Suponiendo que cada uno de los siguientes algoritmos son subprogramas, analizar y completar realizando las adecuaciones para generar el algoritmo principal.
Realizar el seguimiento determinando cuál será el resultado para su lote de prueba.
Responda las preguntas teniendo en cuenta lo efectuado.

A. *Lote de prueba*: C= 20, 14, -2, 0,10, 8,0, -1, 5

```
Comienzo
  K=falso
  Leer C
  Mientras (C != 0)
    Si (((C-10)*3)< 0)
         entonces K=no (C>22)
         sino C= 2*C+1
    Finsi
    Escribir “C”, C
    Leer C
  FinMientras
  Si (K==verdadero)
    entonces  Escribir “la evaluación es verdadera”
    sino Escribir “la evaluación es falsa”
  FinSi
Fin
```

- Si la condición (C!= 0) es falsa ¿qué acción se ejecuta inmediatamente después?
- Si la condición (((C-10)*3) < 0) es falsa ¿qué acción se ejecuta inmediatamente después?
- Indique cuántas veces se ha ejecutado el ciclo.
- Generalice lo que hace el algoritmo.

B. *Lote de prueba*: 12, 6, 9, 5, 12, 3, 4, 4, 1,3

```
Comienzo
……
  Hacer
    Leer N1, N2
    P= N1*N2
    Escribir "El producto de", N1, "*", N2," es", P
  Mientras (N1<N2)
Fin
```

- ¿Cuántas veces iterará el ciclo?
- ¿Qué condición indica que deja de iterarse el ciclo?
- Como mínimo, ¿cuántas veces, se repetirá este ciclo para cualquier lote de prueba?

C. *Lote de prueba:* 40, 20, 75, 15, 38, 22, 15, 15, 13, 1

```
Comienzo
  Leer A, B
  Mientras (( A + B ) > 30)
    S= (A - B) / 2
    Escribir "EL VALOR DE S ES", S
    Leer A, B
  FinMientras
     Escribir A, B
```

- ¿Qué valor en la condición ((A+B) > 30) determina que el ciclo deja de iterarse?
- ¿Cuántas veces iterará el ciclo?
- Para cualquier lote de prueba ¿cuántas veces, como mínimo, se repetirá este ciclo?

D. *Lote de prueba*: K= 2, 6, 3, -3, 6 , 9

```
Comienzo
Para A desde 1 hasta 4
  Escribir "Ingrese el valor de K"
  Leer K
        Si ((K-2)*A >= 4)
              entonces  Escribir  K, A
              sino Escribir 2*A + k*3
        Finsi
Finpara
Escribir "los valores finales de K y A son" K, A
Fin
```

- Si la condición ((K-2)*A>= 4) es falsa ¿qué acción se ejecuta inmediatamente después?
- ¿El valor de la variable K se modifica con la acción Si ((K-2)*A) >= 4?. Justifique
- ¿El valor de las variables A y K se modifican con la acción Escribir 2*A+k*3?. Justifique
- Generalice los aspectos comunes de las estructuras iterativas y haga una tabla comparativa

Ejercicio 20

Un comercio procesa la información de las ventas en un día, de cada una se conoce el número de la factura, la cantidad de productos y el importe total. El ingreso de información finaliza con número de factura -1
Realizar un algoritmo que permita mostrar:
1. El importe de las facturas cuya cantidad de productos vendidos está entre 50 y 100.
2. Mostrar los números de las facturas con su correspondiente cantidad de productos de aquellas ventas cuyo importe total no sea superior a $4000, ni inferior a $9500.

Ejercicio 21
Escribir un algoritmo que utilizando subprogramas permita calcular en número combinatorio C_n^m, siendo m y n dos números naturales que se ingresan por teclado.
Nota: El número combinatorio C_n^m está definido; C_n^m = m ! / (m-n)! n!
Puede ser reutilizado el subprograma que realiza el cálculo del factorial (Ejercicio 14).

Ejercicio 22
Construir un algoritmo donde con un número natural ingresado por teclado y utilizando sub-programas:
1. Muestre sus divisores.
2. Informe la cantidad de divisores.
3. Indique si es primo o no.
4. Explique mediante un texto las diferentes maneras de mostrar los resultados y especifique las modificaciones que se deben realizar.

Nota: Se sugiere reutilizar el subprograma.

Ejercicio 23:
El IPV cuenta con la siguiente información correspondiente a los adjudicatarios que adeudan cuotas: Dni, Nombre de adjudicatario, ingreso mensual, Tipo de operatoria y cantidad de cuotas que adeudan, importe de la cuota.
El tipo de operatoria 'P' . Procrear, 'S': sorteo, 'A': préstamo para ampliación.
Construir un algoritmo que permita mostrar:
1. La cantidad de adjudicatarios que adeudan cuotas
2. La cantidad de adjudicatarios por la operatoria de Procrear, con un ingreso mensual superior a los $10000 que no adeudan cuotas
3. El porcentaje respecto del total que adeudan cuotas, de adjudicatarios por la operatoria de préstamo para ampliación, con un ingreso mensual comprendido entre $10000 y $20000 que adeudan más de 2 cuotas.
4. El importe total de dinero adeudado al IPV por los adjudicatarios
5. La cantidad promedio de cuotas que adeudan los mencionados adjudicatarios
6. Nombre y el tipo de operatoria que tiene el adjudicatario de menor ingreso mensual.

Ejercicio 24
En una carrera de autos se ingresa información de N competidores. Por cada corredor se ingresa el nombre, país, código de escudería y el tiempo (en minutos) que utilizo para terminar la carrera.
La escudería se codifica del siguiente modo: '**F**': Fiat, '**R**': Renault, '**P**': Peugeot
Se pide realizar un algoritmo que usando subprogramas permita:
1. Escribir el nombre de los pilotos que demoraron más de 75 minutos en terminar la carrera.
2. Indicar el tiempo promedio de los pilotos de la escudería Renault.
3. Mostrar la cantidad de pilotos argentinos que participaron.
4. Escribir el porcentaje de pilotos argentinos respecto del total.
5. Mostrar el nombre del piloto que utilizó el mayor tiempo.

Ejercicio 25

Una estación de servicio registra las ventas de combustible realizadas. De cada una de las ventas se conoce: Tipo de combustible (‘N’: Nafta Súper, ‘P’: Nafta Premium, ‘G’: Gasoil), Importe de la venta y Cantidad de litros

Realizar un algoritmo con subprogramas que permita:

1. Escribir cuanto debe pagar cada uno de los clientes sabiendo que las ventas superiores a los $1000 tiene un descuento del 7%.
2. Mostrar el importe de la venta para aquellos clientes que compraron más de 30 litros de Gasoil
3. Decir cuántos litros de Nafta Súper se vendieron en total.
4. Escribir el importe promedio de las ventas.

Ejercicio 26

Se tiene la información de 78 deportistas que participaron en una prueba de atletismo internacional. Los datos a considerar son: Nombre, Puntaje obtenido, País que representa, Edad, Sexo.

Usando subprogramas hacer un algoritmo que:

1. Muestre el porcentaje de hombres menores de 30 años, respecto del total de hombres que ha participado.
2. Escriba el puntaje promedio de deportistas de nacionalidad italiana, cuya edad sea mayor a 25 años
3. Indicar si participaron más hombres que mujeres.

Ejercicio 27

Una zapatería procesa las compras realizadas durante el mes de Julio. Los datos que se conocen de cada una de las compras es: Clase de zapato (‘M’: Mujer, ‘H’: Hombre‘, N’: Niño,), Importe de la compra y Cantidad de pares comprados.

Construir un algoritmo que mediante subprogramas permita:

1. Sabiendo que el importe de la compra incluye el 21% de IVA, indicar cuál es el costo de cada compra.
2. Mostrar el importe de cada compra de zapato de niño donde la cantidad de pares comprados este entre 2 y 5.
3. Determinar cuántos pares de zapatos de mujer se compraron en total.
4. Escribir el importe promedio de las compras.
5. Escribir que porcentaje representa la cantidad de pares de zapato de hombre comprados respecto del total.

Ejercicio 28

Mostrar la tabla de multiplicar del 2 al 7.

```
algoritmo tablas
Comienzo
entero i, j
Para i desde 2 hasta ….
  Escribir “Las Tabla de Multiplicar del ”, i , es
  Para j desde 1 hasta 10
    Escribir …. ,” * “  i,” = “,…*j
Fin Para
FinPara
Fin
```

Construir un algoritmo donde a través de subprogramas calcule y muestre la tabla de multiplicar de un número natural dado, en un intervalo ingresado por teclado.

Ejercicio 29

Se está realizando una campaña de prevención de enfermedades respiratorias, para lo cual se ha efectuado una encuesta en los 19 departamentos de la ciudad de San Juan. Por cada encuestado, se ingresan los siguientes datos: Edad, Sexo ('F' o 'M'), Fumador ('S' o 'N' según si fuma o no respectivamente).

La información ingresa ordenada por departamento y finaliza con edad cero.

Se pide identificar precondición y postcondición. Realizar un algoritmo que permita:

Por Departamento:

1. Cantidad de mujeres que fuman
2. Edad promedio de fumadores

A nivel provincial

3. Total de encuestados.
4. Porcentaje de mujeres fumadoras respecto del total de mujeres encuestadas.
5. Edad del mayor encuestado y departamento al que pertenece indicando si es o no fumador.

Ejercicio 30

Se cuenta con información de los pedidos de medicamentos realizados por distintas farmacias a una droguería. Por cada farmacia se ingresa: CUIL, y por cada pedido de medicamento se ingresan los siguientes datos: Precio unitario, Tipo de medicamento ('c': comprimido, ´i´: inyectable, 'j': jarabe) y cantidad de unidades

El ingreso de pedidos de medicamentos finaliza con precio unitario nulo.

Se pide identificar precondición y postcondición. Realizar un algoritmo que permita:

Por *cada farmacia* muestre:

1. Cantidad de pedidos.
2. Importe total a pagar.
3. Indicar si hubo más cantidad de pedidos de inyectables que de jarabes.
4. Indicar si algún tipo de medicamento registró una pedido superior a 85 unidades

Para la *droguería*:

5. Cantidad de farmacias atendidas.
6. Importe total a cobrar en concepto de jarabes.
7. CUIL de la farmacia que realizó menor cantidad de pedidos de medicamentos

Ejercicio 31

Un Sanatorio de la ciudad de San Juan está realizando un balance para lo cual registra: Documento, Nombre, Código obra Social (`s' si posee obra social, 'n' si no posee), cantidad de días de internación y por cada día el gasto de internación.

Se pide identificar precondición y postcondición. Realizar un algoritmo que permita:

1. Por cada paciente internado: Nº de documento, Nombre e importe que debe abonar por gastos de internación.
2. Cantidad de pacientes sin obra social atendidos por el sanatorio
3. Total que el sanatorio recauda por gastos de internación con los pacientes que tienen obra social.
4. Monto promedio por afiliado que el sanatorio recauda en concepto de gastos de internación.
5. Indicar el monto máximo pagado por un afiliado en concepto de internación

Ejercicio 32

Una ferretería mayorista tiene 5 zonas de reparto (1: "Centro", 2:"Noreste", 3: "Noroeste", 4: "Sureste", 5:"Suroeste") y sus productos clasificados en tres rubros (´P´: "pinturas", ´C´:"materiales de construcción", ´H´: "herramientas"). Por cada zona se ingresan los siguientes datos de sus clientes: nombre, rubro de artículos y monto de de la compra.

Se desea conocer por cada zona: la cantidad de clientes, el nombre del cliente que realizó la mayor compra (suponiendo único) y el total recaudado por rubro.

Determinar además el total recaudado por la ferretería.

```
algoritmo ferretería
Comienzo
entero z, contz
caracter rubro
real monto, acumC, acumH, acumPmaxc, totf
cadena nomcli, maxcli

totf = 0                        //inicialización total recaudado
Para z desde 1 hasta 5
  cantcli =……                  //inicialización contador clientes por zona
  acumC = …….
  acumP = ……..
  acumH = …….
  maxc = …..                    //inicialización máxima compra por zona
  Escribir “Ingrese nombre cliente (finaliza con cliente igual Nadie)”
  Leer nomcli
  Mientas (nomcli ! = “Nadie”)
    Escribir “Ingrese rubro de la compra”
    Leer ……..
    Escribir “Ingrese …… de la compra”
    Leer ……..
    ……. = cantcli + ….          // proceso contadorclientes por zona
    Si ( ….  >maxc )
        entonces                      // proceso cálculo valor máximo
            …… = monto
    maxcli = nomcli
    FinSi
    Segun (rubro)
       ‘C’: acumC = …… + monto
       ‘H’: …… = acumH+ ……..
       ‘P’: ……= ……..   + ……..
       de otro modo: Escribir “ Se ingreso mal el código de rubro”
      finsegun
      Escribir “Ingrese nombre cliente (finaliza con cliente igual Nadie)”
      Leer nomcli
    FinMientras
    Escribir  “El total de clientes de la zona”, z, “son”, ……     // uso contador cliente por zona
    Escribir  “El nombre del cliente de la zona” , …., “que realizó la mayor compra es”, ..…
                                                          // uso maximo por zona
    Escribir  “La venta de materiales de construcción en la zona”, z, “es”, acumC
                                                          // uso acumulador por rubro y por
  zona
    Escribir  “La venta de pinturas en la zona”, …. , “es”, ……..
    Escribir  “La venta de …….. en la zona”, ……,  “es”, …….
  totf = totf + (acumC + acumH + acumP)
Fin Para
Escribir  “ El total recaudado por la ferretería es”, …….
Fin
```

- Indicar otra manera para calcular el total recaudado por la ferretería.
- Explicar qué función cumple la sentencia *de otro modo* del *Segun (rubro)*

Ejercicios Propuestos

Ejercicio 1

Analizar el algoritmo escrito y elaborar un enunciado que se ajuste a él, teniendo en cuenta que se deben usar subprogramas.
Realizar las adecuaciones especificando el ambiente y completando los mensajes en las Acciones Escribir.
Proponer un lote de prueba y realizar el seguimiento.

```
algoritmo Escuela
Comienzo
t=0, c=0, a=0, s=0
Escribir "Ingrese número de sección (termina con 7)"
Leer sec
Mientras (sec<> 7)
        Escribir "Ingrese cantidad de unidades"
        Leer cant
        t= t+1;
        Si (cant<=0)
                Entonces c=c+1
                Sino   a= a+cant
        Finsi
        Si ((sec=1) o (sec=4))
                entonces  s=s+1
        Finsi
        Escribir "Ingrese número de sección (termina con 7)"
        Leer sec
Finmientras
p=a/(t-c)
Escribir "……..",t
Escribir "……..",c
Escribir "……..",a
Escribir "……..",p
Escribir "……..",s
Fin
```

Ejercicio 2

Realizar el seguimiento de los algoritmos que se presentan, determinando cuál será el resultado para su lote de prueba. Complete el ambiente y responda las preguntas teniendo en cuenta lo efectuado. Señale y clasifique las expresiones.

A.

Lotes de prueba:

1) m = 3 n= 1 s= 3 2) m = 5 n= 7 s= 4 3) m = 4 n= 0 s= 2

```
Comienzo
 Escribir “Ingrese los valores de m, n y s”
 Leer m, n, s

 Para i desde 1 hasta s
        Si ((m>n) y (m<>0))
                entonces Escribir s*2
                        m=m -1
               sino  Escribir i*2
                     n=n+1
        Finsi
Finpara
 Escribir “Los nuevos valores de m y n son”, m, n
```

```
Fin
```
La acción Escribir s*2 ¿modifica el valor de la variable s? Justifique.
Indique los valores que toma la variable de control.

B. *Lote de prueba*: D= -3, -1, -4, -5, 6, 9, 0, 10

```
Comienzo
B=6
A=16
Hacer
  Escribir “Ingrese el valor de D”
  Leer D
        Si ((D<3) y (A-B>0))
                        Entonces Escribir “D”, D
        Finsi
  A=A - 4
Mientras A > 0
Escribir “A”, A
Fin
```
Si la condición ((D<3) y (A-B>0)) es falsa ¿qué acción se ejecuta inmediatamente después?
Indique la diferencia que existe entre “A” y A en la acción escribir
¿Necesito leer todos los valores del lote de prueba? Justifique su respuesta.

Ejercicio 3

Analizar el algoritmo escrito y elaborar un enunciado. Especificar el ambiente y completar los mensajes en las Acciones Escribir. Seleccionar un lote de prueba y realizar el seguimiento.

```
algoritmo Lluvias
Comienzo
…….. I, MesL
real ……, MinL
MinT=0
Para I desde 1 hasta 12
        SumL = 0
        Leer Lluvia
        Mientras (Lluvia <> 0)
                Si (Lluvia <MinL)
                        Entonces
                                MinL=Lluvia
                                MesL=I
                FinSi
                SumL=SumL+Lluvia
                 Leer Lluvia
        Finmientras
Escribir "Mes", I , “Cantidad de lluvia”, SumL
FinPara
Escribir "Minima toma pluvial:  ", MinL, “en el mes:  ”, MesL
Fin
```

Ejercicio 4

Se procesa la información de las apuestas realizadas en una agencia de quiniela, por cada apuesta se ingresa el número y el importe. Termina el ingreso de información con número igual a -1.
Construir el algoritmo que permita:

1. Indicar la cantidad de apuestas con importes entre $20 y $30.
2. Escribir el número al que se le hizo la mayor apuesta.
3. Escribir el total de apuestas realizadas.

Ejercicio 5

Una casa comercial tiene 40 vendedores. De cada empleado se conocen los siguientes datos: Nombre, Número de documento, Sueldo básico y Categoría (A, B, C, D)

Por cada empleado, se conoce el importe de cada una de las ventas que ha realizado (el ingreso por empleado finaliza cuando se lee un importe nulo de venta).

Se pide identificar precondición y postcondición. Realizar un algoritmo que permita:

1. Mostrar para cada empleado el sueldo a cobrar, sabiendo que el sueldo se calcula adicionando al básico el porcentaje del importe de todas las ventas realizadas de acuerdo a su categoría: Si es A el porcentaje es 3%, B el porcentaje es 5%, C el porcentaje es 7% y D cuyo porcentaje es 10%
2. Indicar el importe total que la empresa debe pagar en concepto de sueldo.
3. Indicar el importe total de todas las ventas realizadas por la empresa.
4. Escribir el DNI del empleado que registro más de 10 ventas, suponer único.
5. Escribir el nombre de los empleados que realizaron un total de ventas mayor a $1000
6. Indicar la cantidad promedio de ventas realizadas por los vendedores de la casa comercial
7. Indicar si algún empleado realizó una venta que sea superior a $3.500 y no sea mayor a $5.000 (usar bandera lógica)

Ejercicio 6

Una ferretería mayorista tiene 5 zonas de reparto (1: “Centro”, 2:”Noreste”, 3: “Noroeste”, 4: “Sureste”, 5:”Suroeste”) y sus productos clasificados en tres rubros (1: “pinturas”, 2:”materiales de construcción”, 3: “herramientas”). Por cada zona se ingresan los siguientes datos de los clientes que tiene: nombre, rubro de artículos que compra y monto de de la compra.

Se pide identificar precondición y postcondición. Realizar un algoritmo que permita:

1. La cantidad de clientes por zona
2. El total recaudado por cada rubro.
3. Si algún rubro tuvo una venta de más de $500.
4. Por zona, el nombre del cliente que realizó la mayor compra.
5. El total recaudado por la ferretería

Unidad 3: Tipo de datos estructurados Arreglos y registros

Introducción

En la unidad 1 se han estudiado los distintos tipos de datos simples que pueden utilizarse en la construcción de un algoritmo, tales como entero, real, carácter o booleano.

Pero en general la información que se utiliza en la práctica diaria no se representa en forma de datos simples aislados, sino que en general se presenta como un conjunto de datos que se organizan y se estructuran de acuerdo a determinadas reglas modo de facilitar su uso. Por ejemplo, un diccionario, una planilla de horarios de salida de colectivos son colecciones de datos organizadas de manera adecuada para su uso.

Estos conjuntos de datos, organizados de una manera determinada, se llaman **estructura de datos** y se representan por un único nombre.

Las estructuras de datos se distinguen por:

- la forma en que se organizan y procesan sus componentes y
- el tipo de dato que la constituyen.

En particular se analizarán dos tipos de datos estructurados: el arreglo unidimensional y el registro.

Comienza esta unidad con la descripción del tipo de dato arreglo, sus características, manipulación de sus componentes y sus operaciones básicas, tal es el caso de búsquedas y ordenamientos.

Luego describe las características principales del tipo de dato registro. Se analiza la conveniencia del uso de esta estructura, comparando sus características y manipulación con los arreglos. Finalmente se plantea el uso de arreglos de registros como alternativa superadora al uso de arreglos paralelos.

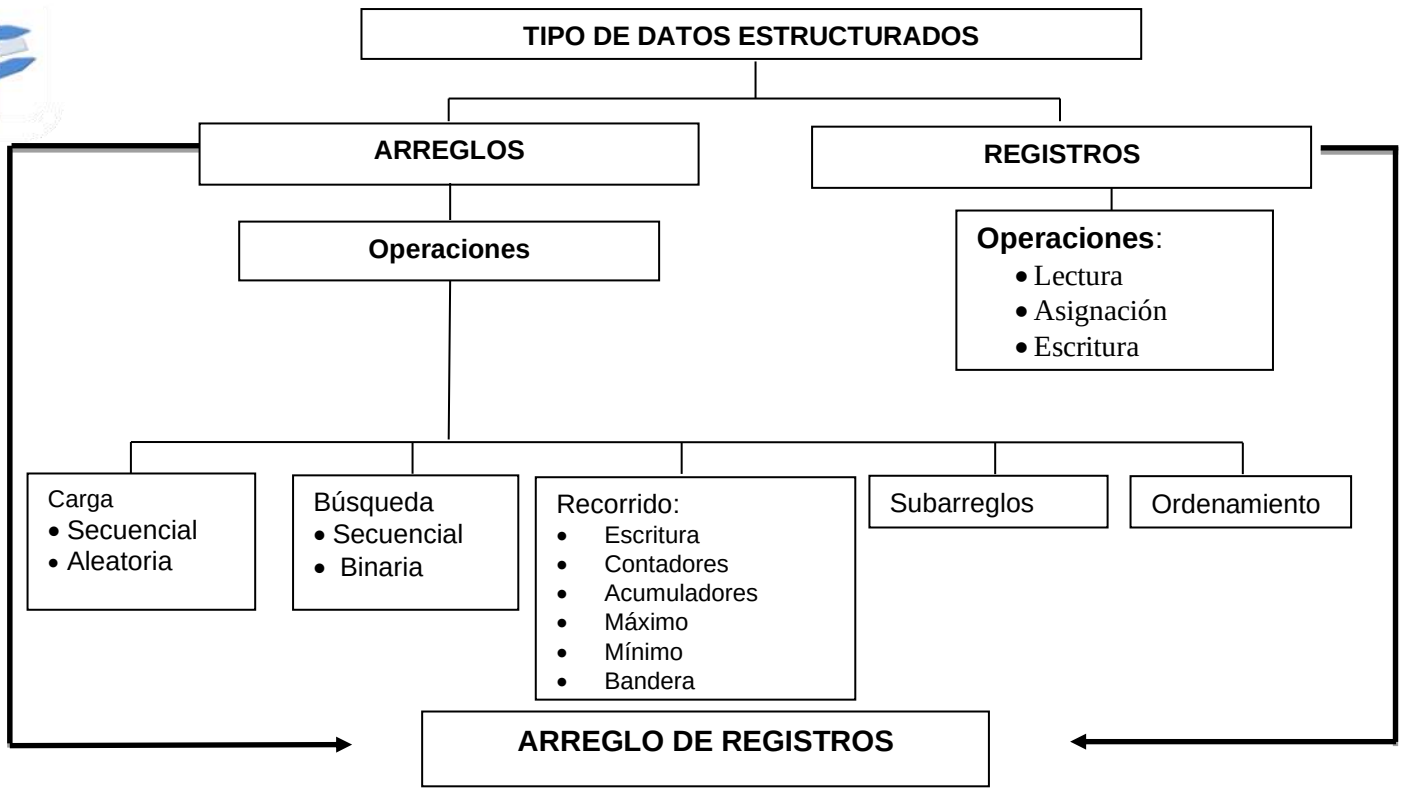


Figura 3.1 Esquema Temático de la Unidad

3.1 Arreglo Lineal

Una estructura de datos de gran utilidad a la hora de programar es el arreglo lineal o arreglo, cuya aplicación y características describiremos después de analizar el siguiente problema.

Ejemplo 1

Una empresa recibe mensualmente el importe total de ventas en cada una de sus tres sucursales. Desea determinar cuáles son las sucursales cuyo importe total supera al importe promedio de ventas de la empresa.

Con las herramientas vistas hasta ahora, las posibles soluciones serían:

Solución 1

Primer paso: Leer el importe total vendido en cada una de las tres sucursales e ir acumulándolo, y luego calcular el promedio.
Segundo Paso: Leer nuevamente las ventas de cada uno de los importes de venta de las sucursales, comparar con el promedio calculado, e indicar si la venta total de la sucursal en cuestión supera o no el promedio.

```
Algoritmo ventas 1
real promedio ( )
Comienzo
   real venta, prom
   entero i
   prom=0
   Para i Desde 1 Hasta 3
      Leer venta
      prom= prom + venta
   FinPara
   retorna (prom/3)
   Fin

   void Compara(real xprom)
   real venta
   entero i
   Para i Desde 1 Hasta 3
      Leer venta
      Si (venta >xprom)
         Entonces Escribir “La venta en la sucursal”, i,” superó la venta promedio”
      FinSi
   FinPara
   retorna()
Fin

  Comienzo /* Algoritmo Principal */
         real prom
         prom=promedio()
         Compara(prom)
Fin
```

Los datos deben ser ingresados en cada subprograma. Se debe leer dos veces el mismo conjunto de datos

Si bien este algoritmo soluciona el problema planteado, es ineficiente ya que lee dos veces el mismo conjunto de datos de entrada.

Solución 2
Para evitar la repetición de la lectura de los mismos datos, se trabajarán tres variables reales cada una de ellas representarán el monto total vendido por una sucursal.

```
Algoritmo ventas 2
real promedio (real xv1, real xv2, real xv3)
comienzo
    real prom
    entero i
    prom=(xv1+ xv2+ xv3)/3
    retorna(prom)
 Fin

void Compara (real xv, real xprom)
comienzo
    Si (xv >xprom)
          entonces Escribir "Las ventas en esta sucursal  superó la venta promedio"
    FinSi
    Retorna()
  Fin

Comienzo /* Algoritmo Principal */
  real venta1, venta2, venta3, prom
  leer venta1, venta2, venta3
  prom=promedio(venta1,venta2,venta3)
  Compara(venta1,prom)
  Compara(venta2,prom)
  Compara(venta3,prom)
Fin
```

¿Si bien este algoritmo resuelve el problema, que pasaría si en lugar de 3 son 30 las sucursales?

Evidentemente es poco práctico e ineficiente declarar 30 variables e invocar 30 veces la función Compara.

Solución 3:
Una manera eficiente es reunir las 30 ventas de sucursales en una única variable, de identificador venta, por ejemplo.

Representación gráfica de la variable venta:

venta

324	456	124	.....	765
0	1	2		29

venta, es un **arreglo lineal o arreglo**, cuyos 30 componentes se identifican como: venta[0], venta[1], ..., venta[29].
Como se observa la estructura completa se identifica con el nombre venta, mientras que para referirse a cada componente se utiliza el identificador de la estructura seguido de la posición del elemento dentro del arreglo, encerrada entre corchetes.

Es importante destacar que en la mayoría de los lenguajes de programación se considera que el índice del primer elemento del arreglo es 1. En este libro se considerará que **el índice del primer elemento es cero**, dado que así lo trabaja el lenguaje C que es el que se utilizará para la implementación de los algoritmos.

Por tanto la posición de un elemento es su índice + 1 .

Así por ejemplo la componente venta[0], (índice 0) se encuentra en la posición 1.

Cada elemento o **componente** de un arreglo puede ser de cualquiera de los tipos de datos primitivos vistos hasta ahora (entero, real, carácter, booleano); pero **todos los elementos o componentes del arreglo deben ser del mismo tipo.**
Antes de mostrar la solución del problema planteado utilizando esta estructura, se presentarán algunos conceptos y se determinará cómo se manipulan las componentes de un arreglo.

Concepto

Una variable arreglo es una estructura de datos y representa un conjunto homogéneo de datos identificados por un único nombre y almacenados en posiciones contiguas de memoria.
Cada componente del arreglo puede ser accedida de manera directa, sin necesidad de recorrer los componentes anteriores.
Un arreglo ocupa una cantidad fija de memoria, determinada por la cantidad y tipo de sus componentes. Por lo que se la considera una estructura de datos estática.

Declaración de un arreglo

Para declarar una variable de tipo arreglo, se indica el nombre de la variable, precedido del tipo de datos de sus componentes y seguido del tamaño del arreglo encerrado entre corchetes.

Para el caso que estamos considerando, la declaración es:

```
real  venta[30]
```

También puede declararse, definiendo previamente el tamaño del arreglo a través de una constante:

```
Constante N=30
real venta [N]
```

En forma general, la declaración de un arreglo lineal o unidimensional:

<tipo> <identificador-arreglo> [<tamaño>]

3.2 Operaciones básicas con arreglos lineales

Las operaciones que se pueden realizar sobre un dato tipo arreglo son:

- **Carga**: Colocar informacion en el arreglo. La carga puede ser ordenada o aleatoria.
- **Recorrido:** Una vez cargado, se recorre el arreglo para mostrar o procesar su información para obtener Contadores, Acumuladores, Máximo, Mínimo y Uso de bandera lógica como señal de aviso de un evento determinado.
- **Subarreglos:** Generar un arreglo que representa un subconjunto de un arreglo principal.
- **Busqueda:** Buscar un elemento en el arreglo.
- **Ordenamiento:** clasificar los elementos del arreglo de acuerdo a un criterio.

Carga secuencial de un arreglo

Una forma de ingresar los datos al arreglo de N componentes, es por medio de la lectura desde teclado de los N valores, de la siguiente manera:

```
Para i Desde 0 Hasta N-1
    Leer  venta[i]
FinPara
```

En este caso los valores se deben ingresar desde el primero hasta el último en forma secuencial, y se irán almacenando en forma contigua a partir de la posición cero. Esta carga recibe el nombre de **Carga Secuencial** de los elementos de un arreglo.
La carga de los elementos de un arreglo también puede realizarse por medio de asignaciones.

Así por ejemplo:

```
Para i Desde 0 Hasta N-1
   venta[ i ] = 0
FinPara
```

Este es un caso de carga secuencial, asignándo tecero a cada una de las N componentes del arreglo.

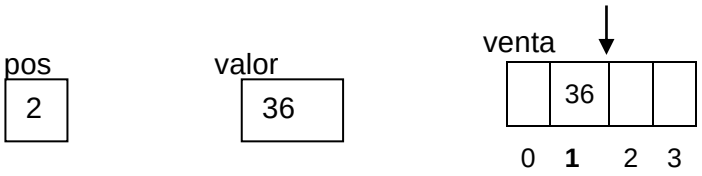
Carga aleatoria de un arreglo

También pueden cargarse las componentes de un arreglo de manera aleatoria, ya sea por lectura o asignación de un valor en una posición sin seguir un orden lineal secuencial.

Por ejemplo:

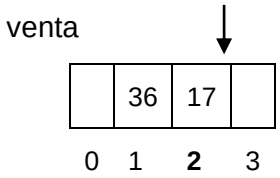
```
entero pos
real valor

Escribir “ingrese la posición de la componente y el valor”
Leer pos
Leer valor
venta [pos -1]=valor //las posiciones empiezan en 1, el índice de las componentes en 0
```

La grafica de memoria es la siguiente:

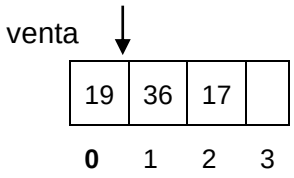


Igualmente pueden realizarse asignaciones particulares tales como:

venta [2] = 17 indica que a la terdera componente del arreglo venta se le asigna el valor 17.



Venta [0] = venta[1] - venta[2], indica que la primera componente del arreglo venta toma el valor que resulta de la resta entre las componentes ssegunda y tercera del arreglo.



Recorrido de un Arreglo: Escritura de un arreglo

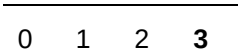
Si se desea escribir alguna componente del arreglo se debe colocar el identificador de la estructura, seguido del su índice, encerrado entre corchetes.

Así, por ejemplo, la acción:

```
Escribir venta[3]
```

mostrará la cuarta componente o sea la componente que ocupa la posición 3 del arreglo venta.

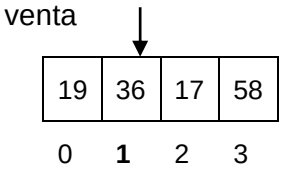
Pantalla



Pantalla

58

Si se considera j=3, entonces la acción *Escribir venta [j - 2]* mostrará la segunda componente del arreglo venta.



Pantalla

36

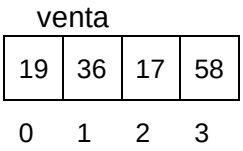
Las acciones:

```
Para i Desde 0 Hasta N -1
   Escribir  venta [ i ]
FinPara
```

Recorrido del arreglo para mostrar en pantalla todas las componentes del arreglo

Permiten mostrar todas las componentes del arreglo en el orden en que están almacenadas en memoria.

Para:



la salida en pantalla sería:

Pantalla

19
36
17
58

Con estas consideraciones, se puede resolver el problema planteado, utilizando la estructura de datos estudiada y subrogramas para resolver cada subproblema

Actividad 1 Analizar la siguiente solución planteada para resolver el Problema del Ejemplo 1

Figura 3.2 Esquema solución problema ventas

Entrada: el importe total de ventas en cada una de sus 30 sucursales.

Salida: los números de las sucursales cuyo importe total supera al importe promedio de ventas de la empresa.

Para resolver el problema planteado se ha dividido el problema en tres subproblemas, resueltos por los subprogramas **Cargar datos en un arreglo, Calcular venta promedio y Mostrar ventas que superan venta promedio.**

Cargar datos en un arreglo: recibe como datos de entrada el arreglo "vacío" y lo devuelve con los valores de las ventas de las sucursales.

Calcular venta promedio: recibe el arreglo con los valores de las ventas de las sucursales y devuelve su promedio.

Mostrar ventas que superan venta promedio: recibe como parámetros el arreglo con los valores de las ventas de las sucursales y su promedio y escribe los números de las sucursales cuyo importe de venta supera la venta promedio de las 30 sucursales, no devuelve resultado al algoritmo principal.

```
Algoritmo Ventas_sucursales
constante N=30
void  carga ( real v[N])
Comienzo
   entero i
       Para i Desde 0 Hasta N-1
              Leer  v[i]
       FinPara
       retorna
Fin
real promedio (real v[N]))
Comienzo
      entero i
      real s=0
       Para i Desde 0 Hasta N-1
              s=s + v[i]
       FinPara
       retorna (s/N )
Fin
void muestra (real v[N]), real prom)
comienzo
   entero i
   Escribir " Numero de sucursales que 'superan la venta promedio"
   Para i Desde 0 Hasta N-1
       Si (v[i] > prom)
            Entonces     Escribir i+1
       Finsi
   FinPara
   retorna ()
Fin
```

Comienzo /*---Algoritmo principal--*/
real venta[N], p
carga (venta)
p=promedio (venta)
muestra (venta, p)
Fin

En la invocación de subprogramas con arreglos como parámetros reales, debe colocarse solamente el nombre de la variable arreglo, en el ejemplo anterior las invocaciones son:

carga (**venta**)

p=promedio (**venta**)

muestra (**venta**, p)

El seguimiento para el siguiente lote de prueba es el siguiente:

Lote de prueba: Suponer 4 sucursales, N=4

Datos para las canga de las ventas 71000, 92000, 86000, 40000

El siguiente esquema muestra el estado de memoria cuando se ejecutan los distintos subprogramas.

Algoritmo principal					
Ventas				**p**	**Salida**
71000	92000	86000	40000		
100h					

Subprograma carga		
v	**i**	**Salida**
100h	0	
	1	
	2	
	3	
	4	

Subprograma promedio				
v	**i**	**s**	**prom**	**Salida**
100h	0	0	72250	
	1	71000		
	2	163000		
	3	249000		
	4	289000		

Subprograma muestra		
v	**i**	**Salida**
100h	0	“ Numero de sucursales que ‘superan la venta promedio
	1	
	2	2
	3	3
	4	

Identificar en cuál/(es) de estas problemáticas planteadas es necesario utilizar un arreglo para su resolución, justificando la respuesta.

a. Una empresa minera instalada en la ciudad de San Juan, ha realizado una convocatoria deaspirantes a ocupar puestos de trabajo. Si en total se inscribieron 50 personas y a cada una de ellas se le solicitó la edad, se desea informar la edad promedio de los aspirantes y cuántos son mayores de 21 años.

b. Una empresa minera instalada en laciudad de San Juan ha realizado una convocatoria a aspirantes para ocupar puestos de trabajo. Si en total se inscribieron 50 personas y a cada

una de ellas se le solicitó la edad, se desea informar la edad del aspirante mayor y cuántos aspirantes tienen entre 21 y 40 años.

c. Una empresa minera instalada en la ciudad de San Juan ha realizado una convocatoria a aspirantes a ocupar puestos de trabajo. Si en total se inscribieron 50 personas y para cada uno de ellos se registra la edad, se desea informar: la edad promedio de los empleados, cuántos empleados son menores a dicha edad y cuantos tienen entre 21 y 40 años (incluyendo dichos valores).

d. En una escuela que cuenta con los 3 niveles (inicial, primario y secundario), se desea procesar los siguientes datos correspondientes a sus docentes: Código de nivel (1..3), nombre, antigüedad, sueldo.

El ingreso de información finaliza con código de nivel 0.

La supervisora necesita completar una planilla con la siguiente información:

- Nombre del docente de mayor antigüedad.Suponer único
- Cantidad de docentes que se desempeñan en el nivel 3 y tienen más de 15 años de antigüedad
- Si algún docente cobra menos de $14.000de sueldo.

Seguimiento de algoritmos con Arreglos

Algoritmo Seguimiento1

```
void carga1 ( entero pf[4], entero pr[4])
    Comienzo
    entero i
    Para i Desde 0 Hasta 3
      Leer pf[i]
      pr[i]=0
    FinPara
    retorna ()
Fin

void carga2 (entero pp[4] )
    Comienzo
    entero i
    Para i Desde 3 Hasta 0 con paso-1
        Leer pp[i]
    FinPara
   retorna ()
fin

void calcula (entero pf[4], entero pr[4]), entero pp[4])
Comienzo
entero i
  Para i Desde 0 Hasta 3
    Si ( (i Resto 2) !=0)
         Entonces pr[i]= pr[i]+ pf[4-i]+ pp[i]
    FinSi
   FinPara
  retorna ()
Fin
void mostrar (entero pr[4] )
Comienzo
entero i
    Para i Desde 0 Hasta 3
```

```
      Escribir pr[i]
    FinPara
   retorna ()
Fin
```

```
Comienzo/*  Algoritmo Principal*/
entero f[4], p[4],r[4]
    carga1(f,r)
    carga2(p)
    calcula(f,r, p)
    mostrar(r)
Fin
```

Lote de prueba: f: -2, 3, 7, 8 p: 5, 4, -7, 1

Algoritmo principal — S

f				P				r			
-2	3	7	8	1	-7	4	5	0	0	0	0
									1		8
100h				200h				300h			

Subprograma carga1 — S

p f	p r	i
100h	300h	0
		1
		2
		3
		4

Subprograma carga2 — s

p p	i
200h	3
	2
	1
	0
	-1

Subprograma calcula — **Salida**

Pf	pr	pp	i
100h	300h	200h	0
			1
			2
			3
			4

Subprograma Mostrar — **Salida**

pr	i	
300h	0	0
	1	1
	2	0
	3	8
	4	

Referencia **S**: *salida*

Cuando se invoca el subprograma **carga1(f,r)** , se envían como parámetros dos arreglos f y r, que están vacíos**.** Cuando se realiza el seguimiento del subprograma **carga1**, en lugar de colocar espacio para los arreglos se colocan sólo dos variables simples **pf** y **pr.** Estas variables contienen la dirección de memoria del primer componente de cada uno de los arreglos respectivamente, y de esta manera los valores se cargan en los arreglos del algoritmo principal.

Cuando se invoca a un subprograma cuyo parámetro actual es un arreglo, para realizar el seguimiento del subprograma se utiliza el almacenamiento asignado en el algoritmo principal.

Actividad 3

Realizar la traza del siguiente algoritmo:

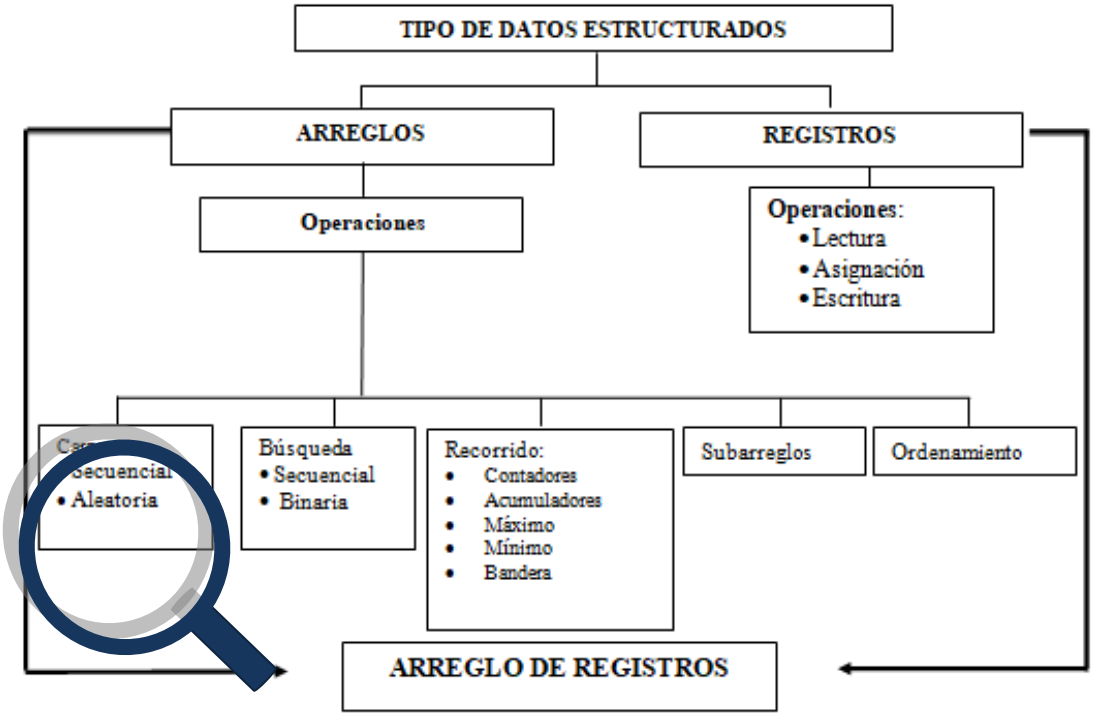
Lote de prueba: m1: 8, 1, 4, 6, 2

AlgoritmoSeguimiento2

```
void carga1 ( entero m[5], entero s[5])
    Comienzo
      Entero i
      Para i Desde 0 Hasta 4
           Leer m[ i ]
           s[ i ]= 2 + i
        FinPara
        retorna
    Fin


void calculo1 ( entero m[5])
    Comienzo
      Entero i

Para i Desde 2 Hasta 4
Si ( (i Resto 2)==0)
              Entonces m[ i ] =m[ i ] /2
           FinSi
     FinPara
        retorna
    Fin


void calculo2 ( entero m[5], entero s[5])
    Comienzo
      Entero i
        Para i Desde 0 Hasta 4
          Si ((m[ i ]  Resto 2)==0)
             Entonces s[4-i] =s[ i ]+ m[ i ]
          FinSi
        FinPara
        retorna
    Fin


void Mostrar ( entero x[5])
    Comienzo
      Entero i
        Para i Desde 0 Hasta 4
        Escribir x[ i ]
        FinPara
        retorna
    Fin



/*  Algoritmo Principal  */
Comienzo
  entero m1[5], s1[5]
```

carga1(m1,s1)
calculo1(m1)
calculo2(m1,s1)
Mostrar (m1)
Mostrar (s1)
Fin

Hasta ahora hemos visto la carga secuencial de elementos de un arreglo, ahora analizaremos la carga aleatoria.



Carga aleatoria de un arreglo

Retomando la actividad 1, se puede observar que la información se ingresa en forma secuencial y los datos se presentan de la siguiente manera:

Cajero 1: 56289.75
Cajero 2: 45325.50
Cajero3: 84567.88
:
Cajero 36: 75987.47

Esquemáticamente:

56289.75	45325.50	84567.88		75987.47
Imp[0]	Imp[1]	Imp[2]		Imp[35]

Los datos se almacenarán en memoria en forma consecutiva y en el orden en que aparecen, utilizando las acciones que constituyen el subprograma carga1:

void **carga1** (real imp[50])

Comienzo

Entero i

```
    Para i Desde 0 Hasta 36
        Leer imp[i]
    FinPara
   retorna
Fin
```

¿Cómo se realizaría la carga del arreglo si la información de las distintas cajas se ingresa en forma aleatoria? ¿Es suficiente la información que se ingresó por teclado?

Para el caso que planteamos ahora, la información con que se cuenta es por ejemplo:

Cajero 3: 84567.88

Cajero36: 75987.47

:

Cajero 2: 45325.50

Cajero 1: 56289.75

Como se observa, el orden de ingreso de los datos no coincide con el orden en que se almacenarán en memoria. Por lo tanto, para ingresar la información se necesitan dos datos, el importe total y el número del cajero que recaudó dicho valor. Este número es el que indica la posición donde se almacenará el importe que le corresponde.

Nuevamente, puede utilizarse la acción **Para** en el ingreso de los valores, ya que son 36 importes.

El algoritmo que permite el ingreso desordenado, pero de tal manera que cada importe se almacene en la celda que le corresponde, es el siguiente:

Algoritmo Importe_Aleatorio

```
void carga_aleat ( real imp[36])
Comienzo
real importe,
entero nc,i
   Para i Desde 0 Hasta 49
      Escribir "Ingrese número de cajero e importe recaudado"
      Leer  nc, importe
      imp [nc - 1] = importe
   FinPara
  retorna
Fin

  *** Algoritmo Principal **
Comienzo
  real impor[50] ,
  carga_aleat(impor)
  ......
Fin
```

Actividad 4 ¿Cómo se puede optimizar el subprograma **carga_aleat**?

Subarreglos

Carga de un arreglo a partir de los valores de otro arreglo.

En ocasiones, es necesario generar un arreglo con las componentes de otro arreglo que cumplen una determinada condición. Recordemosque cuando se declara un arreglo, previamente debe especificarse su tamaño para que el compilador pueda reservar la cantidad de memoria necesaria para su almacenamiento.

En este caso no se puede conocer a priori la cantidad de componentes que verificarán el requisito especificado, por ello se recomienda definirlo con el mismo tamaño del arreglo original, contemplándose de esta manera el máximo tamaño posible, que ocurre cuando todas las componentes de la estructura original cumplen la condición y por tanto deben almacenarse en el nuevo arreglo.

Retomando nuevamente el ejemplo de los 50 aspirantes inscriptos y suponiendo que se desea almacenar solamente las edades de aquellos que son mayores de 25 años.

La siguiente es la representación gráfica de los arreglos:

Edad	20	19	35	22	29	.....	37	43
	edad[0]	edad[1]						edad[49]

El nuevo arreglo, cuyo identificador puede ser**selec**, tendrá las componentes mayores a 25 ubicadas en forma consecutiva y adyacente, **sin dejar huecos**.

Selec	35	29	…….	37	43	.....		
	selec[0]	selec[1]						selec[49]

Para generar el nuevo arreglo, que se conoce con el nombre de ***subarreglo***, se van contando los valores que cumplen la condición prefijada. Luego, este contador se utiliza para procesar solamente los lugares en los que se han almacenado componentes, ignorando los restantes.

Entrada: 50 edades
Salida: Valores de edades, como máximo 50 valores

El algoritmo que permite realizar esta tarea es el siguiente:

```
AlgoritmoSeleccion
void carga ( real ed[50])
    Comienzo
     entero i
          Para i Desde 0 Hasta 49
              Leer  ed[i]
          FinPara
          retorna()
      Fin
entero carga_subarreglo ( real ed[50], real sel[50])
    Comienzo
     entero i,c
        c=0
        Para i Desde 0 Hasta 49
          Si (ed[ i ] >25)
                Entonces sel[ c ] = ed [ i ]
                          c=c+1
          FinSi
        FinPara
        retorna c
    Fin

void mostrar_subarreglo (  real sel[50], entero can)
    Comienzo
    entero i,
          Para i Desde 0 Hasta can-1
            Escribir sel[ i ]
         FinPara
      retorna
      Fin


Comienzo/* Algoritmo Principal /*
    entero edad[50] , selec[50], c
    carga (edad)
    c=carga_subarreglo(edad, selec)
    mostrar_subarreglo(selec, c)
Fin
```

Actividad 5

Se cuenta con un arreglo A de números enteros de tamaño N y dos valores a y b (a< b), que indican dos posiciones dentro del arreglo . Escribir un algoritmo que genere y muestre un nuevo arreglo que contenga los valores de A, comprendidos entre dichas posiciones.

Búsqueda de un elemento en un arreglo

La búsqueda de un elemento dentro de un arreglo es una de las operaciones más frecuentes e importantes en el procesamiento de la información, permitiendo la recuperación de datos previamente almacenados.
En general, todo algoritmo de búsqueda tiene como objetivos conocer si el elemento se encuentra o no en el conjunto analizado y/o si el elemento está en el conjunto, indicar su posición.

Como principales algoritmos de búsqueda en arreglos veremos:

Búsqueda Secuencial

Búsqueda Binaria

Búsqueda Secuencial o Lineal

Consiste en recorrer y examinar cada uno de los elementos del arreglo, desde el primero, hasta encontrar el elemento buscado o hasta que se hayan examinado todos los elementos del arreglo sin éxito.

Los siguientes algoritmos presentan distintas formas de realizar la búsqueda de un elemento en un arreglo de 50 componentes de tipo entero:

Forma 1: utilizando una bandera lógica

```
Algoritmo Bandera
Constante N=50
void carga ( entero ar[N])
Comienzo
entero i
    Para i Desde 0 Hasta N-1
      Leer  ar[i]
    FinPara
    retorna()
Fin

Booleanobuscar ( entero ar [N], entero el)
Comienzo
    entero i
    booleano esta
    i=0
    esta=falso
    Mientras ((i<N) y ( esta == falso))
       Si (ar[ i] == el)
          Entonces  esta = verdadero
          Sino  i = i+1
       FinSi
    FinMientras
    retorna (esta)
 Fin

Comienzo    /*Algoritmo principal/*
entero arre [N], elem
booleano p
 carga(arre)
  Leer elem
  p=buscar (arre, elem)
  Si (p ==verdadero)
     Entonces  Escribir “el elemento se encuentra en el arreglo”
     Sino Escribir “el elemento no se encontró en el arreglo”
  FinSi
Fin
```

Forma 2: sin utilizar una bandera lógica

```
Algoritmo Sin_Bandera
Constante N=50
void carga ( entero ar[N])
Comienzo
entero i
        Para i Desde 0 Hasta N-1
          Leer  ar[i]
        FinPara
        retorna()
```

```
Fin
entero  buscar ( entero ar [N], entero elem)
Comienzo
entero i
    i=0
    Mientras ((i<N) y (ar[ i] != elem))
          i=i+1
    FinMientras
    retorna (i)
Fin

Comienzo /*Algoritmo principal/*
    entero arre [N], p, elem
    carga(arre)
    Leer elem
    p=buscar (arre, elem)
    Si (p <N)
      Entonces  Escribir “el elemento se encuentra en la posición”,  p
      Sino  Escribir “el elemento no se encontró en el arreglo”
    FinSi
Fin
```

Forma 3: utilizando un elemento centinela.

Otro algoritmo de búsqueda secuencial es el que utiliza un **centinela**. Consiste en definir un arreglo con una componente más, donde en la última posición *agrega* como elemento adicional justamente el que se está buscando.

Al utilizar el elemento centinela A[N] la búsqueda siempre tendrá éxito, por lo tanto, no es necesario preguntar en cada paso si llegó al límite del arreglo. El valor del índice es el que indica si el elemento se encuentra en el arreglo original; si el índice alcanza el valor N significa que el elemento no está en el arreglo.

```
Algoritmo Centinela
Constante N=50
void carga ( entero ar[N+1])
Comienzo
entero i
    Para i Desde 0 Hasta N-1
        Leer  ar[i]
    FinPara
    retorna()
Fin

entero  buscar ( entero ar [N+1], entero elem)
Comienzo
entero i=0
    ar[N]=elem
    Mientras (ar[i]!=elem)
          i=i+1
    FinMientras
    Retornai
Fin
```

```
Comienzo   /*Algoritmo principal/*
    entero arre [N+1], p, elem
    carga(arre)
    Leer elem
    arre [N]= elem
    p=buscar (arre, elem)
    Si (p ==N)
      Entonces  Escribir “el elemento no se encontró en el arreglo”
      Sino  Escribir “el elemento se encuentra en la posición”,  p
    FinSi
Fin
```

Si al finalizar el bucle del Mientrasen la función buscar, el valor que retorna i es igual a N, indica que el elemento no se encontraba en el arreglo original.

Actividad 6
Se cuenta con los números de registros correspondientes a los 58 alumnos que aprobaron una determinada asignatura. Construir un algoritmo que permita informar a un alumno, cuyo número de registro se ingresa por teclado, si está o no aprobado.

Búsqueda binaria o dicotómica

La búsqueda lineal, por su simplicidad, es buena para arreglos pequeños. Sin embargo, para arreglos de gran cantidad de componentes y si los datos están ordenados, es conveniente usar la búsqueda binaria.
Este método es similar al que se realiza cuando se busca una palabra en el diccionario. La búsquedano comienza siempre por la primera página y se sigue secuencialmente, sino que se divide el diccionario en 2 partes:al acceder a la página se analiza si se ha acertado o en qué parte, la primera o la segunda, podría estar la palabra buscada. Se repite este proceso hasta que por divisiones sucesivas se encuentra la palabra o no se encuentra, ya sea porque está mal escrita o el diccionario no está completo.
Supongamos que el arreglo está ordenado ascendentemente. La búsqueda binaria consiste en:

- comparar el elemento a buscar con el elemento central.
- Si coinciden, la búsqueda termina.
- Sino: dividir el arreglo en dos subarreglos más pequeños.
- Si el elemento en cuestión es menor que el elemento central y está en el arreglo, debe estar en el primer subarreglo; caso contrario, debería estar en el segundo subarreglo.

Sea A un arreglo de N componentes reales y sea x el elemento que se desea buscar.
Si las componentes de ese arreglo están ordenadas en forma creciente, esto es:

A[0] < = A[1] < =..... <= A[N-1].

El método consiste en dividir el intervalo de búsqueda por la mitad, del siguiente modo:
Se determina la componente central del arreglo completo, supongamos que es A[M]. Si el valor buscado x se encuentra en esa posición, la búsqueda termina. Si es menor, es decir x<A[M], la búsqueda debe continuar en la primera parte del arreglo , dejando de lado la segunda mitad. Obteniendo un intervalo de búsqueda menor:

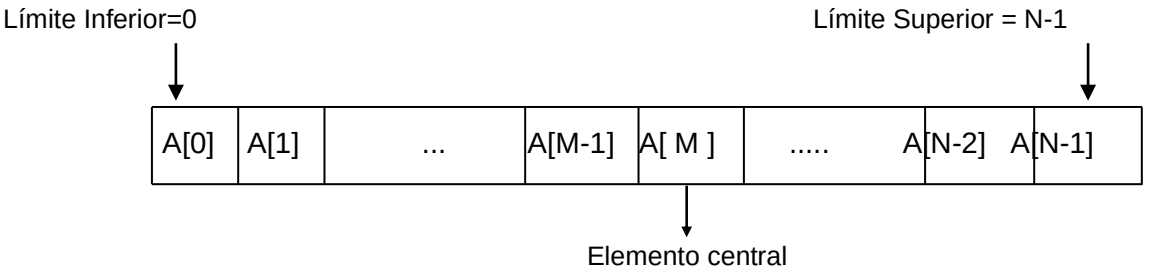
A[0].....A[M-1].

En este subarreglo se vuelve a buscar la componente central, supongamos A[P]. Si el valor es mayor que el central, es decir x > A[P] entonces se toma como nuevo subarreglo de búsqueda la segunda mitad:

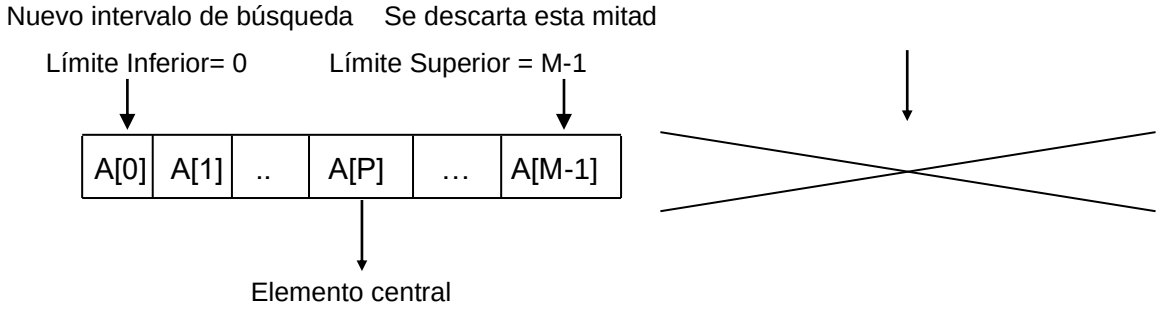
A[P+1]....A[M-1]

Se determina nuevamente la componente central y se sigue el mismo criterio hasta que se encuentra el valor buscado, o se obtiene un intervalo unitario de búsqueda que puede contener o no al elemento buscado.

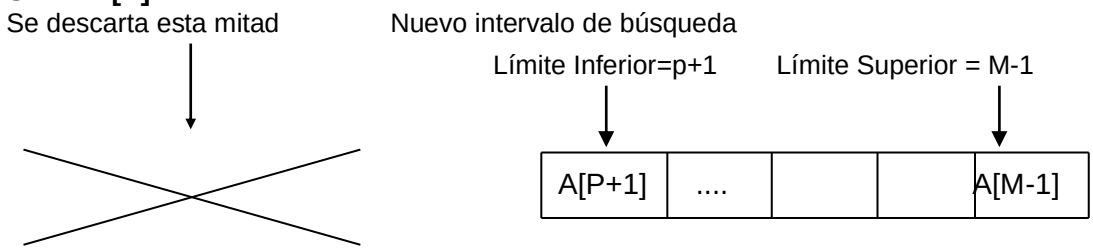
Esquemáticamente:



Si **x< A[M]**



Si **x> A[P]**



Como se ve, este método es más eficiente que la búsqueda secuencial, ya que en cada comparación si el elemento buscado no es el central, se va desechando la mitad del arreglo.
La única restricción es que el arreglo debe estar **ordenado**.
Por ejemplo, para buscar el elemento 32 en el arreglo {11, 21,32, 43, 59, 62, 71, 88, 90} se realizarían los siguientes pasos:
Se toma el elemento central y se divide el arreglo en dos. El índice del elemento central se calcula de la siguiente manera:

(límite inferior + límite superior) div 2 (siendo div la división entera)

Para el caso en cuestión, límite inferior=0; límite superior= 8, el elemento central es el que está en la posición 4, esto es el 59.
Por lo tanto el arreglo queda dividido:
{11,21,32,43} - **59** - {62,71,88,90}
Como el elemento buscado (32), es menor que el central (59), debe estar en el primer subarreglo:
{11, 21, 32, 43}. Este es ahora el nuevo arreglo de búsqueda.

En este caso límite inferior=0 ; límite superior= 3, entonces el elemento central es el que está en la posición 1, esto es el 21.
Se vuelve a dividir este arreglo en dos: {11} - **21** - {32,43}
Como el elemento buscado es mayor que el central, debe estar en el segundo subarreglo: {32,43}
En este caso límite inferior = 2 ; límite superior = 3, entonces el elemento central es el que está en la posición 2 esto es el 32. Es decir, el elemento buscado coincide con el central finalizando así la búsqueda.

El código correspondiente a este algoritmo es:

```
Algoritmo Búsqueda_Binaria
Comienzo
  Constante N=9
void carga ( entero ar[N])
Comienzo
entero i
        Para i Desde 0 Hasta N-1
            Leer  ar[i]
        FinPara
        retorna
Fin

entero  buscar ( entero ar [N], entero elem)
Comienzo
    entero inf, sup, medio
    inf=0
    sup=N-1
    medio=(inf+sup) div 2
    Mientras ((inf<=sup) y (elem!=ar[medio]))
      Si ( elem<ar[medio] )
         Entonces sup=medio-1
         Sino   inf=medio+1
       FinSi
      medio=(inf+sup) div 2
    FinMientras

    Si ( inf<=sup )
      Entonces retorna (medio)
      SiNo       retorna ( -1)
    FinSi
Fin

Comienzo/*Algoritmo principal */
    entero arre [  N], p, elem
    carga(arre)
    Leer elem
    p=buscar (arre, elem)
    Si (p != -1)
      Entonces  Escribir “el elemento se encuentra en la posición”,  p
      Sino  Escribir “el elemento no se encontró en el arreglo”
    FinSi
Fin
```

Actividad 7

Realizar el seguimiento o traza del algoritmo anterior para buscar el elemento 64, en el arreglo dado en el ejemplo.

Ejemplo 2

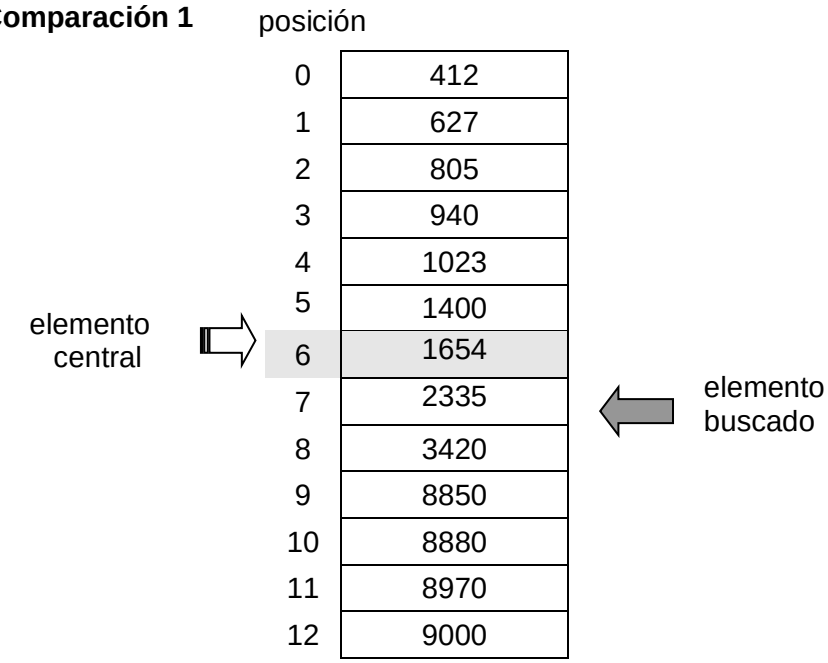
Considerar que en un banco se han almacenado, ordenados en forma creciente, los números de cuenta de aquellos clientes que en el día de la fecha han realizado extracciones en cuentas de Caja de ahorro por montos superiores a los $10000. Para ello se ha utilizado el arreglo cant, definido de la siguiente manera:

entero cant[13]

412	627	805	940	1023	1400	1654	2335	3420	8850	8880	8970	9000
0	1	2	3	4	5	6	7	8	9	10	11	12

Se desea conocer si el cliente número 2335 ha realizado una operación que se corresponda con la condición fijada.
Como los números de cuenta de los clientes están ordenados en forma creciente, se puede utilizar el algoritmo de búsqueda binaria para determinar si el número del cliente indicado está o no en el arreglo.
En el esquema siguiente se muestran las comparaciones que se realizan en esta búsqueda:

Comparación 1

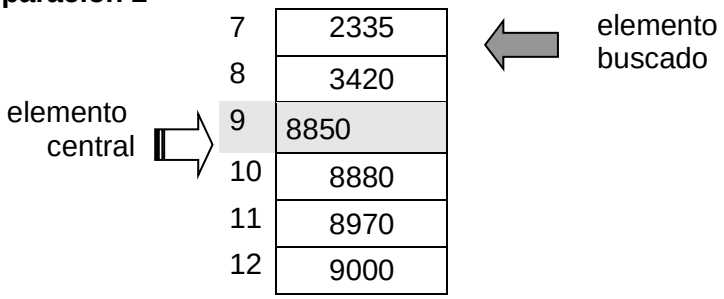


Como el elemento buscado es mayor que el elemento central 1654, significa que está en la segunda mitad del arreglo.

Recordar que el índice del elemento central se obtiene con la fórmula:
(límite inferior + límite superior) div 2

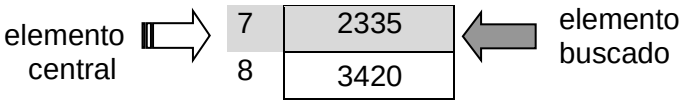
La búsqueda continúa en esta segunda mitad.

Comparación 2



Como 2335 es menor que el central de esta segunda mitad 8850, el nuevo rango a analizar es:

Comparación 3



Como el elemento central es justamente el buscado termina el proceso.
En este caso se han realizado 3 comparaciones, mientras que en la búsqueda lineal se hubieran necesitado 7 comparaciones para encontrar el valor 2335 en el arreglo dado.

Para arreglos con mayor cantidad de elementos la diferencia entre la cantidad de comparaciones se hace más notable.
Las siguientes tablas muestran el seguimiento de este algoritmo para la búsqueda de distintos números de clientes:

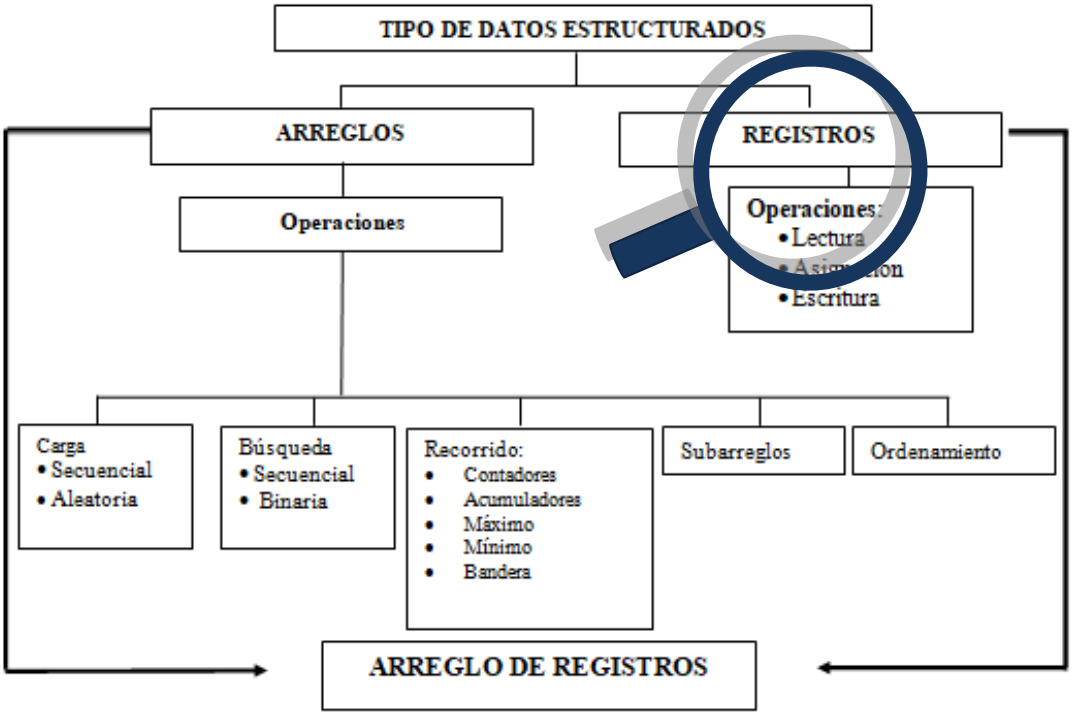
a) Si el elemento buscado es el número de cliente 8970:

Inf	Sup	Medio	A[medio]	CAUSA DE TERMINACION
0	12	6	1654	
7	12	9	8850	
10	12	11	8970	El elemento se encontró en la posición 11

b) Si el elemento buscado es el número de cliente 627

Inf	Sup	Medio	A[medio]	CAUSA DE TERMINACION
0	12	6	1654	
0	5	2	805	
0	1	0	412	
1	1	1	627	El elemento se encontró en la posición 1

Actividad 8
Realizar una tabla de seguimiento para el caso en que los números de clientes buscados son: 215, 10500, 8900, 900



3.3 Registro

Hasta ahora la estructura de datos estudiada, arreglo unidimensional, se caracteriza por ser una estructura homogénea, es decir puede contener sólo componentes del mismo tipo.
Ahora se verá una nueva estructura de datos, llamada *registro*, que permite agrupar variables que pueden ser de distinto tipo y que están relacionadas entre sí por hacer referencia a datos de un mismo objeto, entidad, evento, etc.
Por ejemplo, con el identificador **dirección** se puede nombrar a una estructura registro que tiene como datos a: nombre de calle (una cadena de caracteres), número de puerta (entero) y orientación (un carácter: N,S,E,O).
También se puede usar una estructura de registro **rega-alu** para almacenar información sobre un alumno y que contenga como datos: nombre (cadena de caracteres), matricula (entero), año de nacimiento (entero) y estado civil (caracter)

Un *Registro* es una estructura de datos que permite almacenar una colección finita de datos no necesariamente del mismo tipo.A cada una de las partes que forman la nueva estructura de datos se la denomina **campo.**

Definición del tipo y declaración de variables

Suponiendo que se desea definir una variable de tipo registro que contenga el nombre, número de documentoy nota obtenida en un parcial por un alumno.

Entonces:

Registro **alumno**
{
cadena **nom**
entero **docu**
real **nota**
}

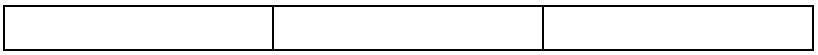
Define el tipo de dato alumno como un registro que contiene 3 campos: el campo nom de tipo cadena, el campo docu de tipo entero y el campo nota de tipo real

alumno **alu** Declara una variable **a** de tipo alumno

La variable **alu**(identificador de registro) se puede representar gráficamente en función de sus campos (**nom, docu** y **nota** son los identificadores de campo).

Esquemáticamente, la variable **alu**:

alu



nom **docu** **nota**

Un registro, al igual que un arreglo, ocupa celdas consecutivas de memoria.

En forma general la declaración del tipo Registro es:

registro <identificador registro>
{ <tipo><identificador de campo1>
<tipo><identificador de campo2>
}

Cada campo está definido por un nombre, el identificador de campo y por el tipo de dato que en él se almacenará.

Acceso a los campos de un registro

Los campos de un registro son accedidos utilizando el identificador de campo en vez de la posición relativa como sucede en los arreglos, de la siguiente forma:

<Identificador de registro> . <Identificador de campo>

Esta expresión recibe el nombre de selector de campo.

Ejemplo 3

alu.nom permite acceder al campo nombre del registro alu.

Entonces, mediante la acción **Leer alu.nom** se puede ingresar desde teclado el nombre del alumno.

La acción**alu.docu** = 28573309; permite asignar 28573309 al campo **docu** dela variable registro **alu**.
Si se realiza la asignación **c= alu.nota**, entonces la variable c debe ser declarada de tipo real.

El campo de un registro accedido por el selector de campo, es tratado como cualquier otra variable de ese tipo.

Operaciones sobre registros

Las operaciones básicas sobre un registro son: lectura, escritura y asignación.

Lectura y Escritura
Por ser un registro una estructura compuesta, se deben efectuar las operaciones de **lectura y/o escritura individualmente** para cada uno de sus campos.

Ejemplo 4

```
Leer  alu.nom,  alu.docu
Escribir alu.nota
```

Asignación
La asignación entre registros está permitida. Si **a** y **r** son variables registros del **mismo tipo**, la acción **a=r** copia todos los valores asociados con el registro **r** al registro **a**.

Ejemplo 5: si se declaran dos variables **alu** y **egresado** del tipo alumno:

```
alumno alu, egresado
```

La acción de asignación de registros: **egresado= alu;**
equivale a:

```
egresado.nom = alu.nom
egresado.docu =  alu.docu
egresado.nota = alu.nota
```

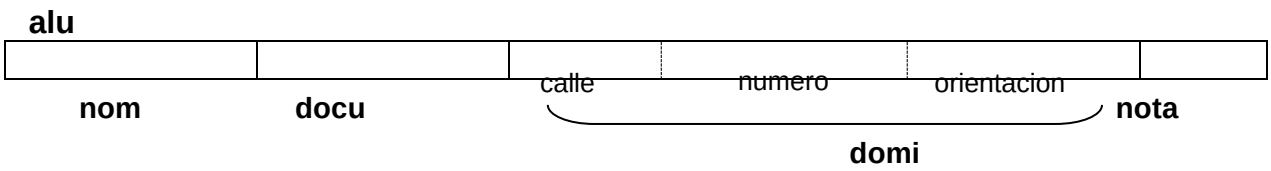
Anidamiento de Registros

Los campos de un registro pueden ser a su vez un registro, en este caso se trabajará con registros anidados

Ejemplo 6: Si se considera el campo domicilio en el registro alumno anterior

```
Registro alumno
    { cadena  nom
      entero  docu
       Registro   domi
          { cadena calle
             entero  numero
             caracter orientacion
           }
        real nota
    }
alumno alu
```

Esquemda de la variable registro **alu**:



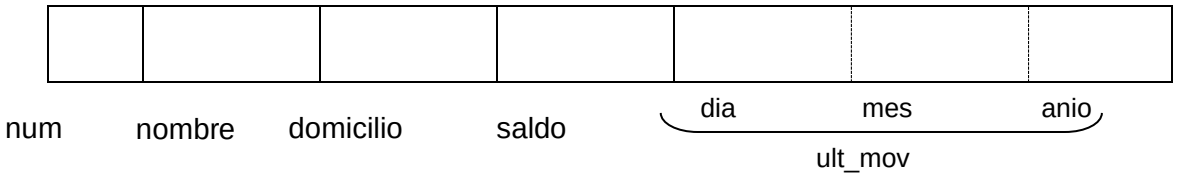
El registro **alu**tiene 4 campos**: nom, docu, domi** y **nota.**
Para referenciar un campo en registros anidados se debe indicar el camino a seguir en orden jerárquico desde el nombre del registro hasta el campo específico. Por ejemplo para hacer referencia al nombre de calle en el registro se coloca: **alu.domi.calle**

Ejemplo 7
El siguiente registro permite almacenar los datos de la cuenta corriente de un cliente de una determinada empresa.

```
registro fecha
        { entero dia
          entero  mes
 entero anio
        }
registro cliente
        { entero num
          cadena nombre
          cadena domicilio
          real saldo
        fecha ult_mov
        }
cliente cli
```

Gráficamente:

cli



En este ejemplo, el registro **ult_mov** ha sido definido como miembro de otro registro. En estas situaciones, la definición del registro interno debe preceder a la definición del registro externo que lo contiene.
Este tipo de definiciones permite que la estructura **fecha** pueda ser usada en distintos contextos.

Actividad 9
Considerando el registro del ejemplo 7, indicar como se accede al mes del último movimiento de un cliente.

Actividad 10
Una Obra Social inicia las clases de Yoga. Realizar un algoritmo que:

a) lea los datos de los 85 afiliados inscriptos a dichas clases: nombre, edad, altura, peso
b) escriba el nombre, la edad y el peso de la persona de mayor altura, suponiendo que es único.

Arreglos como campos de un registro

Los campos de un registro pueden contener arreglos.

Ejemplo 8

Suponer que en lugar de una única nota, se quiere registrar las notas correspondientes a 5 evaluaciones realizadas al alumno.
La estructura que permitirá almacenar esta información es un registro como:

```
registro alumno1
    {  cadena  nom
       entero  docu
       registro   domi
                { cadena calle
                  entero  numero
                  caracter orientacion
                 }
       real nota [5]
     }
alumno1alum
```

Para leer o mostrar en forma secuencial las notas se debe utilizar la acción Para..FinPara. Así por ejemplo si se quiere almacenar las notas en el registro:

```
        :
        Para  i desde 0 hasta 4
        Leer alum.nota [i]
              :
```

Para mostrar todos los valores almacenados en el arreglo de manera secuencial la expresión es:

```
        Para  i desde  0 hasta 4
        Escribir alum.nota [i]
              :
```

Para cambiar el valor de una nota en particular, se debe usar la sentencia de asignacióny se debe colocar el índice que corresponde. Por ejemplo si se desea cambiar la tercera nota, la expresión es la siguiente:**alum.nota[2]=7**

Ejemplo 9

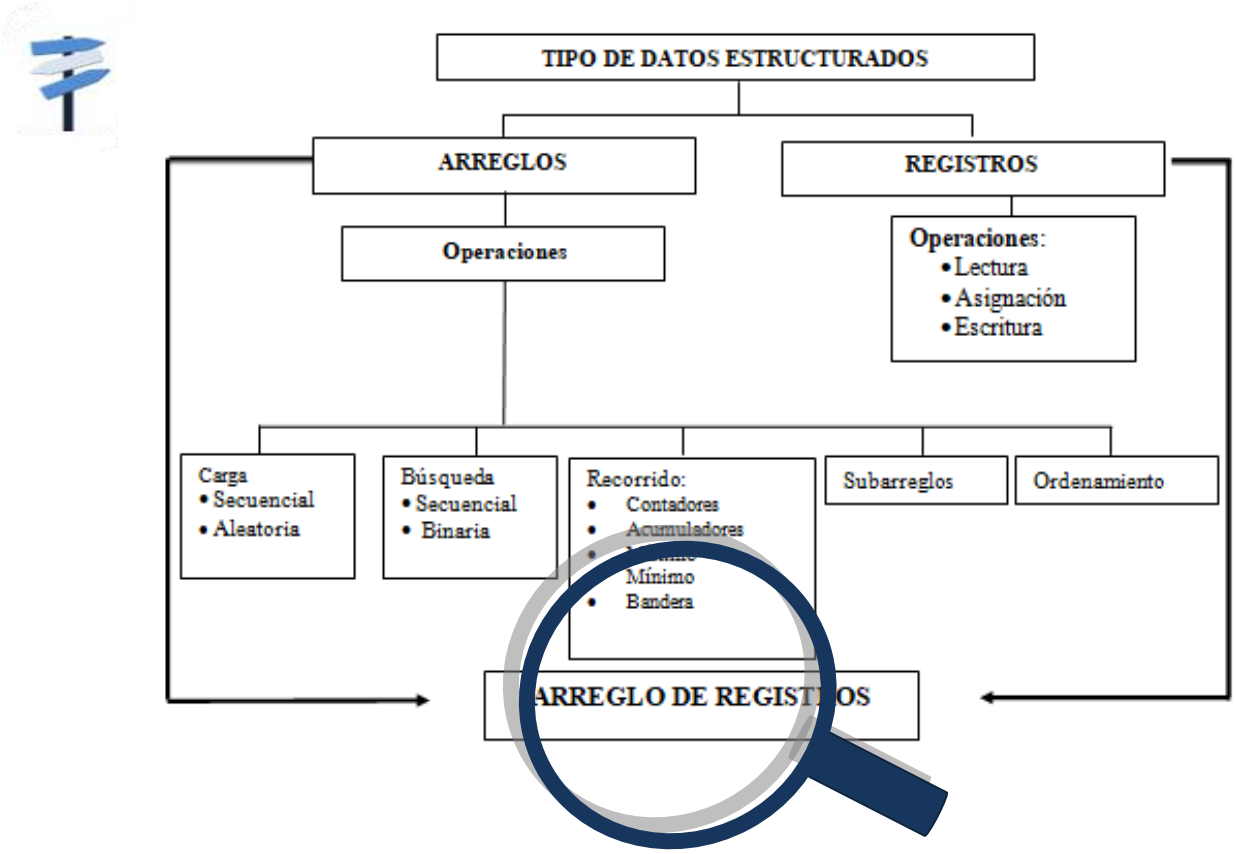
El siguiente registro contiene la información correspondiente al total de votos obtenidos por un partido político en cada uno de los 19 departamentos de la ciudad de San Juan en las últimas elecciones:

```
registro partido_politico
{  int numero;
   int votos [19];
  };
partido_politico m;
```

m. numero refiere al número del partido político.

m.votos[5] refiere al total de votos obtenidos en el sexto distrito.

Hasta ahora hemos visto arreglos cuyas componentes son tipo de datos simples, ahora veremos arreglos cuyas componentes son registros, llamados Arreglos de Registros.



3.4 Arreglo de Registros

Problema:

Una empresa desea incentivar con un importe de dinero fijo a los empleados que durante un mes determinado tuvieron asistencia perfecta. Se cuenta con la información correspondiente a 80 empleados: nombre, cantidad de faltas en el mes y sueldo.

El responsable de la sección cómputos debe adicionar el incentivo dispuesto al sueldo de los empleados que corresponda,. Deberá además generar una nueva estructura que contenga los nombres de aquellos empleados que recibirán incentivo y emitir un listado de dichos nombres. Finalmente deberá calcular, para informar a contaduría, el importe total que la empresa deberá abonar en concepto de sueldos.

Podemos resolver este problema utilizando un registro para almacenar la información de cada empleado.

nom	**cfalt**	**suel**

registro **empleado**
{ cadena **nom**
entero **cfalt**
real **sueld**
}

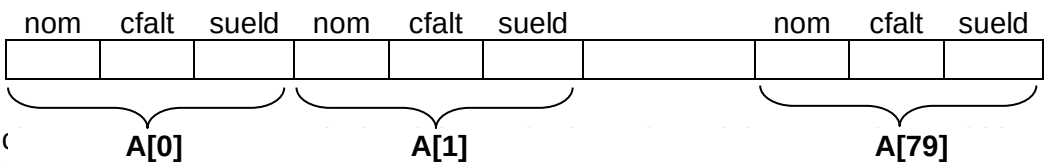
Como se debe almacenar la misma información para 80 empleados, definimos un arreglo de 80 componentes homogéneas, cada una de ellas es un registro que contiene 3 campos en los que se almacena el nombre, la cantidad de faltas y el sueldo de cada empleado.

Esta estructura llamada **ARREGLO DE REGISTROS** se declara:

empleado A[80]

Gráficamente la variable A

A



Así, para referenciar un elemento del arreglo se debe indicar el identificador de la variable A, luego indicar la componente del arreglo a referenciar mediante el índice entre [] seguido del punto para aludir al campo del registro correspondiente.
Para identificar el campo donde se almacena el nombre escribimos **A[i].nom**, el campo con la cantidad de faltas **A[i] . cfalt**, y el campo sueldo **A[i].sueld**

A continuación se muestra el subprograma de **carga** del arreglo de registro del ejemplo dado con las acciones que permiten realizar la carga de la información:

```
void carga ( empleado A[80] )
comienzo
entero i
Para i desde 0 hasta 79
    Leer  A[i].nom,  A[i] . cfalt, A[i].sueld
Finpara
   retorna()
fin
```

Actividad 11
Completar el siguiente algoritmo para que resuelva el problema anterior utilizando subprogramas

```
Algortimo Arreglo_Registro
registro empleado
  { cadena  nom
    entero  cfalt
    real      sueld
  }
void carga ( empleado Ar[80] )
  comienzo
     entero i
      Para i desde 0 hasta 79
           Leer  Ar[i].nom,  Ar[i] . cfalt, Ar[i].sueld
      Finpara
      retorna()
 fin
……
……
```

Algoritmo Principal

```
Comienzo
    empleado A[80]
    real incentivo
    carga(A)
  ……
 ……..
fin
```

Se debe notar que la estructura registro se defineinmediatamente después del nombre del algoritmo y antes (fuera) de los subprogramas. De esta manera puede ser utilizada por todos los subprogramas.

Actividad 12

Se reciben las inscripciones de los 70 alumnos aspirantes de becas, en la facultad de Ciencias Exactas.

Por cada alumno se ingresa:

- Nombre
- Calificación de asignaturas rendidas (arreglo de hasta 40 elementos conteniendo el código de asignatura y la nota correspondiente)
- Fecha de ingreso a la facultad (día, mes, año)

Se pide escribir las acciones que permitan:

a) Definir la estructura de datos más adecuada para almacenar la información anterior.

b) Escribir el nombre y el año de matriculación del alumno cuyo orden de inscripción es 17.

c) Calcular y almacenar el promedio de notas del alumno cuyo orden de inscripción es 25. Mostrar el valor obtenido

3.5 Otras operaciones sobre arreglos

En diversas situaciones surge la necesidad de usar un conjunto de contadores o acumuladores, a los cuales pueda accederse a través de un índice. En estos casos, se definen arreglos de contadores y/o acumuladores, que deben estar inicializados en cero antes de realizar la carga respectiva, esto es previo a ser usados.

Ejemplo 10:
Se ingresan las notas obtenidas por 200 alumnos en el examen de una materia determinada. Las notas van del 1 al 10 Se pide:
1) La frecuencia de alumnos por nota (para cada nota decir cuantos alumnos la obtuvieron.)
2) Decir cual es la frecuencia con mas cantidad de alumnos.
3) Decir si alguna frecuencia tuvo 0 (cero) cantidad de alumnos.

Entrada: la nota obtenida por cada uno de los 200 alumnos

Salida: la nota que registra mayor frecuencia.

la cantidad de alumnos que obtuvieron la nota de mayor frecuencia

Para resolver este problema, es necesario ir contando la cantidad de alumnos que obtuvieron 0, 1, 2, ..., 10 en su examen. Por lo tanto se define un arreglo de 11 componentes; cada componente será un contador de cada nota, el que previamente debe estar inicializado en cero.

Luego se busca un maximo y se evalua una bandera.

Solución propuesta:

```
Algoritmo frecuencia
Comienzo
    Constante N=11

void cerear ( entero con[N])
Comienzo
entero i
    Para i Desde 0 Hasta N-1
      con[i]=0
    FinPara
    retorna()
Fin

void cargar ( entero con[N])
Comienzo
entero i,
real nota
    Para i Desde 1 Hasta 200
        Escribir " ingrese  nota del alumno ",i
        Leer nota
        con[nota - 1]=con[nota -1 ] +1 // se cuentan los alumnos por nota
    FinPara
retorna()
Fin
```

```
entero mayor (entero con[N])
Comienzo // aca se busca el valor mayo
entero i
real max
max= con[0]
Para i Desde 1 Hasta N-1
Si ( con[ i ] > max)
      Entonces max=con[ i ]
FinSi
FinPara
retorna (max)
Fin
```

```
Void notas_mayor_frecuencia ( entero con[N], entero xmax)
Comienzo // como el valor mayor puede repetirse, aca se buscan esas repeticiones
entero i
Escribir "El valor de la mayor frecuencia es: " xmax
Para i Desde 0 Hasta N-1
   Si ( con[ i ] == xmax) // el valor maximo se puede repetir
        Entonces Escribir “la nota”, i+1, “registró la mayor frecuencia”
  FinSi
FinPara
retorna ()
Fin
```

```
boolean frecuencia_cero (entero con[N])
Comienzo // usa bandera para averiguar si sucede el evento
entero i
boolena band
Band= falso
Mientras (No Band) y (i <N)
   Si ( con[ i ] == 0) // pregunta por el evento
        Entonces Band= Verdadero // si toma esta opcion termina el bucle
        Sino i = i + 1 // continua recorriendo el arreglo
  FinSi
FinPara
retorna (Band)
Fin
```

```
Comienzo//Algoritmo principal
  entero cont[N], maxfrec
  cerear (cont)
  cargar(cont)
  maxfrec=mayor_frecuencia(cont)
  notas_mayor_frecuencia(cont, maxfrec)
  si (frecuencia_cero(cont)== Verdadero)
        Entonces Escribir "Alguna frecuencia tuvo cantidad 0 de alumnos"
                  // entra a esta opcion si el subprograma retorno el valor Verdadero
        Sino Escribir "Ninguna frecuencia tuvo cantidad 0 de alumnos"
  finsi
Fin
```

Cuadro comparativo de operaciones basicas sobre datos esctructurados

La generalizacion de las operaciones se ha escrito en funcion de arreglos simples, en caso de usar arreglos de registros se debe tener en cuenta el campo con el cual se trabaja.
En la siguiente tabla se generalizan los momentos habituales de la estas operaciones

Inicio	Proceso	Fin
CONTADOR Se cerea el arreglo	En un subprograma , dentro de una iteracion se cuenta: C[indice] = C[indice] +1	En un subprograma se usa el arreglo de contadores.
ACUMULADOR Se cerea el arreglo	En un subprograma , dentro de una iteracion se acumula: C[indice] = C[indice] +Variable	En un subprograma se usa el arreglo
MAXIMO Max=0	Si Max < Arre[indice] Entonces Max = Arre[indice]// actualiza [si se guardan valores relacionados se usan variables auxiliares] finsi	El valor mayor encontrado es **unico**, pero puede repetirse o no. En caso de repetirse se hace un subprograma Compara, donde se recorre el arreglo y Si Max == Arre[indice] Entonces Se hace un proceso o se escribe un mensaje. finsi:
MINIMO Min=99999	Si Min > Arre[indice] Entonces Min = Arre[indice]// actualiza [si se guardan valores relacionados se usan variables auxiliares]	El valor menor encontrado es **unico**, pero puede repetirse o no. En caso de repetirse se hace un subprograma Compara, donde se recorre el arreglo y Si Min == Arre[indice] Entonces Se hace un proceso o se escribe un mensaje. finsi:
BANDERA Bandera = falso	Se identifica un evento, analizando los datos del arreglo en la condición. Si CONDICION entonces bandera = verdadero sino indice = indice +1 [se sigue recorriendo el arreglo] finsi	Se evalua la bandera. Si sucede el evento devuelve verdadero, sino habra recorrido todo el arreglo y devuelve falso. Si bandera == verdadero entonces // EL EVENTO SUCEDIÓ sino // EL EVENTO NO SUCEDIÓ finsi

Actividad 13

El Departamento de Informática ha propuesto 10 talleres, que se realizan en forma gratuita como parte de las actividades organizadas para las Jornadas de Informática.
Para cada uno de los talleres se ha asignado un cupo. En el momento de la inscripción, se solicita a cada participante seleccionar el número del taller al que desea concurrir (1..10).
Construir un algoritmo que indique:

1. la cantidad de aspirantes a cada uno de los talleres
2. el o los talleres con mayor cantidad de inscriptos
3. el número del o los talleres en que la cantidad de inscriptos supera el cupo previsto.

Actividad 14

Procesar la información de 50 alumnos de la carrera de Geología, para los cuales se ingresa la matrícula y la nota de cada una de las materias rendidas. El ingreso de materias por alumno, finaliza con nota igual a –1.
Se pide:

1. Mostrar las matrículas de los alumnos de mayor promedio.
2. Suponiendo que la carrera tiene 25 materias, mostrar la cantidad de alumnos que rindió sólo una materia, dos materias, y así sucesivamente hasta 25 materias.

Actividad 15

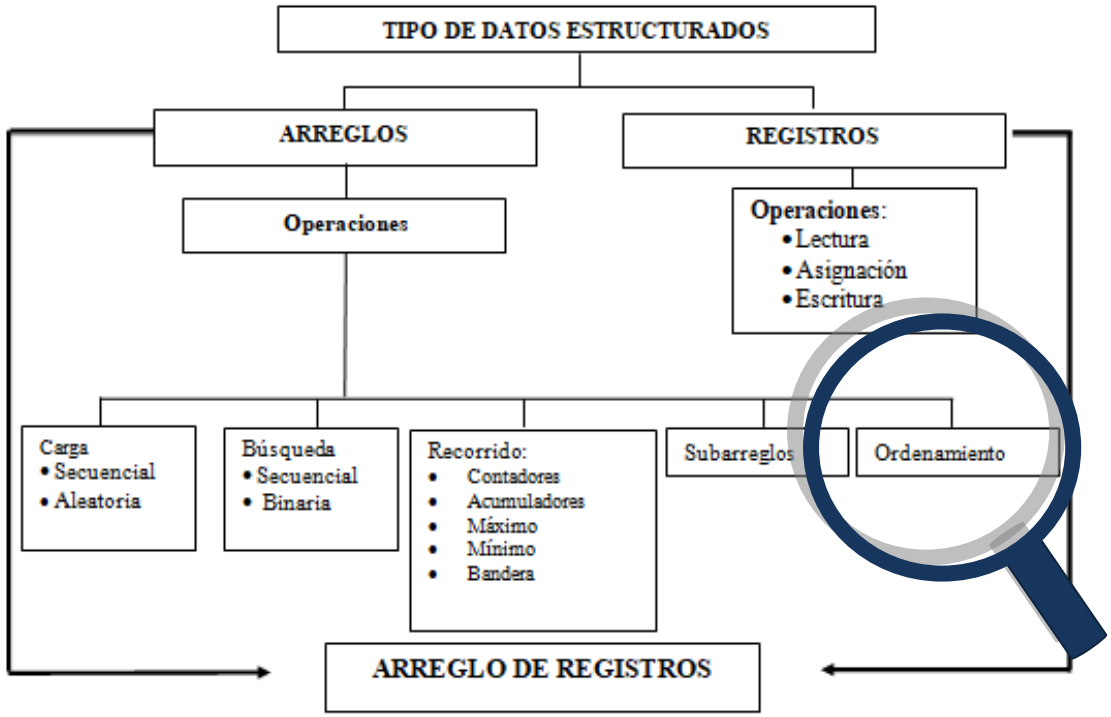
Una empresa de turismo tiene información de las reservas realizadas por 500 turistas y necesita realizar un estudio de los turistas que confirmaron sus reservas. Por cada confirmación se ingresa: documento del turista, provincia de destino (codificadas de 1 a 23) y total de días de la reserva.

1. Mostrar los documentos de los turistas que hicieron reservas por mayor cantidad de días.
2. Mostrar la o las provincias para las cuales se realizó la mayor cantidad de confirmaciones.
3. Dado el documento de un turista, indicar si corresponde a los que hicieron reserva por mayor cantidad de días

*Sugerencias para la actividad:*Para que el algoritmo resulte óptimo sólo se deben procesar las confirmaciones, que no necesariamente son 500. Se debe almacenar la información sólo de cada turista que confirmó su reserva (el documento, provincia de destino y la cantidad de días reservados). Por lo tanto, teniendo en cuenta que a lo más son 500 los turistas que confirmaron su reserva, la estructura de datos correspondiente debe tener una dimensión de 500 componentes.

Además, por cada una de las 23 provincias se deberá ir contando la cantidad de confirmaciones realizadas, para luego calcular el máximo de confirmaciones por provincias. Finalmente se deberá mostrar los números de provincias cuyo total de confirmaciones coincidan con ese máximo

A continuación analizaremos algoritmos que permiten ordenar arreglos, ascendente o descendentemente, según un criterio determinado.



3.6 Algoritmos de ordenamiento

En varias ocasiones es conveniente tener el arreglo ordenado, en un orden creciente o decreciente según un determinado criterio numérico, alfabético, alfanumérico, para facilitar la búsqueda de elementos dentro de él.

Tipos de ordenamiento: Algunos autores clasifican los ordenamientos, según donde estén almacenados los valores a ordenar, en dos tipos internos y externos.

Ordenación interna: La ordenación interna se realiza cuando los datos están en memoria principal. Como la memoria principal es de acceso directo o aleatorio, la principal ventaja, es que se puede acceder a una determinada componente sin recorrer las anteriores, por tanto el tiempo de acceso a cualquier elemento del arreglo es siempre el mismo.

Ordenación externa: La ordenación externa se lleva a cabo cuando el volumen de los datos a tratar es demasiado grande y los mismos no caben en la memoria principal de la computadora

En estos casos la ordenación es más lenta ya que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición a la que se accedió.

Otro criterio de clasificación es según el método utilizado para ordenar los elementos, en este sentido se distinguen:

- *Algoritmos de intercambio*: En este tipo de algoritmos los elementos se consideran de dos en dos, se comparan y se INTERCAMBIAN si no están en el orden requerido. Este proceso se repite hasta que se han analizado todos los elementos.
- *Algoritmos de inserción*: Se caracterizan por que los elementos que van a ser ordenados son considerados de a uno a la vez. Cada elemento es INSERTADO en la posición que le corresponde, respecto al resto de los elementos ya ordenados de acuerdo al criterio considerado.

- *Algoritmos de selección*: Estos algoritmos se caracterizan por que se busca o SELECCIONA el elemento más pequeño (o más grande), según el criterio de ordenamiento elegido, de todo el conjunto de elementos y se coloca en su posición adecuada. El proceso se repite para el resto de los elementos hasta que todos son analizados.

La siguiente tabla muestra los algoritmos más utilizados, clasificados por su tipo.

TIPO	NOMBRE
Intercambio	Burbuja Quicksort
Selección	Selección directa.
Inserción	Inserción directa. Inserción binaria Shell Hashing

Tabla 3.1 Tipos de Algoritmos de Ordenamiento

De estos algoritmos se estudiarán los más simples, atendiendo a las herramientas teóricas con las que se cuenta hasta el momento, en este curso de iniciación a la programación.

Ordenación interna

Los métodos de ordenación interna se aplican preferentemente en arreglos unidimensionales. Los principales algoritmos de ordenación interna son:

Método de la Burbuja

El método se basa en la comparación de los elementos adyacentes del arreglo, intercambiándolos si están desordenados. De este modo, si el ordenamiento es ascendente, los valores más pequeños burbujean hacia las primeras componentes del arreglo (hacia la primera posición), mientras que los valores más grandes se "hunden" hacia el fondo del arreglo.
Supongamos que se quiere ordenar en forma creciente el siguiente arreglo :
A[0], A[1],...,A[N - 1].

Las comparaciones se comienzan de izquierda a derecha, comparando A[0] con A[1], intercambiándolos si están desordenados, luego A[1] con A[2],..etc. Este proceso de comparaciones e intercambios continúa a lo largo de todo el arreglo.

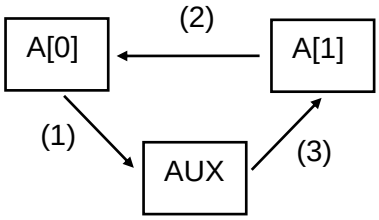
Estas operaciones constituyen una pasada o recorrida a través del arreglo.

Al terminar esta pasada, el elemento mayor queda en la última posición, y algunos de los elementos más pequeños han burbujeado hacia las primeras posiciones.

Se vuelve a explorar el arreglo, comparando elementos consecutivos, a partir del primero, intercambiándolos cuando están desordenados, teniendo en cuenta que el último elemento no debe compararse ya que está ordenado, es decir se encuentra en su posición correcta. Al terminar la 2ª pasada, quedan ordenados dos elementos. Siguen las comparaciones hasta que el arreglo quede completamente ordenado, lo que sucede cuando se hayan realizado N-1 pasadas.

Debe recordarse que para intercambiar el contenido de dos variables se necesita una variable auxiliar, de lo contrario se perderá el contenido de una de ellas.

El siguiente esquema muestra el intercambio entre los valores de dos variables, por ejemplo A[0] y A[1]:



El contenido de A[0] se almacena en AUX.

El contenido de A[1] se almacena en A[0].

El contenido de AUX se almacena en A[1].

Veamos cómo funciona el algoritmo por medio de un ejemplo. Supongamos que se desea ordenar en forma ascendente el siguiente arreglo de 6 componentes:

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Primera Pasada: si el arreglo tiene 6 elementos, se deben realizar 5 comparaciones (N-1)

21	40	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Primera comparación: Se cambia el 21 por el 40.

21	4	40	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Segunda comparación: Se cambia el 40 por el 4.

21	4	9	40	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Tercera comparación: Se cambia el 9 por el 40.

21	4	9	10	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Cuarta comparación: Se cambia el 40 por el 10.

21	4	9	10	35	**40**
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Quinta comparación Se cambia el 35 por el 40.

Elemento Ordenado

Como se observa, en esta pasada se han realizado 5 comparaciones (N-1) y ha quedado en la última posición el elemento mayor. Sin embargo el arreglo todavía no está ordenado, sólo algunos elementos se han trasladado en busca de sus posiciones. Se realiza entonces la segunda pasada.

Segunda pasada: Como hay un elemento ordenado, en esta pasada se realizan 4 comparaciones (N-2)

21	4	9	10	35	**40**
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Estado del arreglo al comenzar la segunda pasada

4	21	9	10	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Primera comparación: Se cambia el 21 por el 4.

4	9	21	10	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Segunda comparación: Se cambia el 9 por el 21.

4	9	10	21	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Tercera comparación: Se cambia el 10 por el 21.

4	9	10	21	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Cuarta comparación : No hay cambios

Como se ha dicho el algoritmo debe realizar N-1 pasadas, para el caso que estamos considerando debería realizar aún 3 pasadas innecesariamente, ya que el arreglo ha quedado ordenado en la segunda.

Si el arreglo tiene N elementos, para determinar si está ordenado, se deberían realizar N-1 pasadas, y en cada pasada i, N-i comparaciones.

En el ejemplo considerado, se tiene:

Pasada 1 se realizan 5 comparaciones (6-1)

Pasada 2 se realizan 4 comparaciones (6-2)

Pasada 3 se realizan 3 comparaciones (6-3)

.

Pasada i se realizan (N-i) comparaciones

El algoritmo del Método de la Burbuja es el siguiente:

```
Algoritmo Burbuja
Comienzo
constante n=6
void cargar ( entero a[n])
Comienzo
entero i
   Para i Desde 0 Hasta n-1
     Leer  a[i]
   FinPara
   retorna()
Fin
```

```
void ordenar ( entero a[n] )
Comienzo
entero    i , aux, j, n1
 n1=n-1
   Para j Desde 1 Hasta n1
       Para i desde 0 Hasta n1-j
          Si  (a[i] >a[i+1])
           Entonces
                    aux = a[i]
          a[i] = a[i+1]
                    a[i+1] =aux
          FinSi
       FinPara
FinPara
retorna()
Fin

void mostrar (  entero a[n] )
    Comienzo
    entero i,
         Para i Desde 0 Hasta n-1
            Escribir a[ i ]
        FinPara
        retorna()
     Fin
```

/* Algoritmo Principal /*

```
Comienzo
     entero   ar[n]
    cargar(ar)
    mostrar(ar)
    ordenar(ar)
    mostrar(ar)
Fin
```

El método de la burbuja así planteado, tiene dos inconvenientes:
1- En cada pasada, el número de comparaciones sólo disminuye en una unidad respecto de la anterior. Esto resulta poco eficiente para el caso de arreglos en donde en alguna pasada los elementos de las últimas posiciones hubieran quedado ordenados.
2- Para arreglos como los del ejemplo, que queden ordenados en pasadas intermedias es necesario que el algoritmo detecte esta situación, para evitar comparaciones y pasadas innecesarias.

Este método se puede optimizar, realizando dos consideraciones importantes que resuelven lo expuesto:
1- Detectar en cada pasada la posición del último cambio realizado. De esta manera en la pasada siguiente se llega hasta esa posición, ya que a partir de ella los elementos están ordenados. Se evita así la comparación innecesaria de las últimas posiciones, cuando están ordenadas.
2- Utilizar un indicador que registre, a lo largo de una pasada, sí se han realizado intercambios. De esta manera, si en una pasada no se realizan cambios, la ejecución debe detenerse ya que ello significa que el arreglo está ordenado.

El método conocido como Burbuja Mejorado atiende estos aspectos, como describiremos a continuación.

Método de la Burbuja Mejorado

Si A es un arreglo de dimensión N, cuyas componentes deben ser ordenadas ascendentemente, el método consiste básicamente en:

Realizar un proceso iterativo que consiste en una serie de pasadas, en cada una de las cuales se comparan pares de elementos adyacentes, ordenándolos si están desordenados.

Al finalizar la primera pasada el elemento mayor queda ubicado en la última posición, y en las distintas pasadas los elementos más grandes tienden a desplazarse hacia la derecha.

En cada pasada debe detectarse la posición del último cambio realizado, para que en la siguiente, las comparaciones- que siempre comienzan desde el primer elemento- se realicen hasta una posición anterior al último cambio realizado.

Si durante alguna pasada no se realizan cambios, el algoritmo finaliza.

El algoritmo que responde a estos requerimientos es:

```
Algoritmo Burbuja_Mejorado
Comienzo
Constante n=6
void cargar ( entero a[n])
Comienzo
entero i
        Para i Desde 0 Hasta n-1
        Leer  a[i]
        FinPara
        retorna()
     Fin
```

```
void ordenar ( entero a[n] )
Comienzo
entero    k, i , aux, cota
   cota= n-1
   k= 1
   Mientras ( k != -1)
    k=-1
    Para i Desde 0 Hasta cota-1
       Si   (a[i] >a[i+1])
                Entonces    aux = a[i]
                            a[i] = a[i+1]
                            a[i+1] =aux
                            k =i/* guarda la posición donde se realizó el cambio*/
       FinSi
     FinPara
     cota =k     /* guarda el índice del último cambio de la actual pasada como cota para la
                    siguiente*/
   FinMientras
   retorna()
Fin
```

```
void mostrar (  entero a[n] )
Comienzo
entero i,
    Para i Desde 0 Hasta n-1
      Escribir a[ i ]
   FinPara
   retorna()
Fin

Comienzo/* Algoritmo Principal /*
entero   ar[n]
  cargar(ar)
  mostrar(ar)
  ordenar(ar)
  mostrar(ar)
Fin
```

Se debe tener en cuenta que:
k: tiene doble función, por un lado detecta en cada recorrida el índice del elemento que cambió su posición, entregando a la variable cota la posición del último cambio realizado. Además, es una bandera que si conserva su valor inicial indica que en la última pasada no se detectaron cambios, es decir el arreglo **ya está ordenado.**

Cota: indica al finalizar cada pasada, hasta donde deben realizarse las comparaciones en la pasada siguiente. Es decir, hasta que posición el arreglo está todavía desordenado.

Analicemos nuevamente el ejemplo, ahora considerando los valores que toman las variables cota y k en las distintas pasadas necesarias para ordenar el siguiente arreglo:

40	21	4	9	10	35
a[0]	a[1]	a[2]	A[3]	a[4]	a[5]

Cota	K	
5	1-10 1 2 3 4	fin de la primera pasada
4	-1 0 1 2	fin de la segunda pasada
2	-1	detecta arreglo ordenado

Se puede observar que:

- En la primera pasada, se deben realizar las N-1 (5) comparaciones.
- En la segunda pasada, como la variable cota toma el valor del último cambio realizado en la pasada anterior, deberán realizarse 4 comparaciones. En esta pasada, el valor de k, que siempre es inicializado en -1 al comenzar cada pasada, toma el valor 2, que es la posición del último cambio realizado. Por lo tanto desde A[2] hasta A[5] el arreglo está ordenado.
- Cuando comienza la tercer pasada, el valor de k queda con su valor inicial -1, lo que significa que el arreglo está ya ordenado y por lo tanto debe detenerse la ejecución del algoritmo

Actividad 14

Analizar el algoritmo del Método de la Burbuja Mejorado con el siguiente arreglo. Realizar la traza correspondiente para determinar los distintos valores que toman las variables cota y k . Indicar también cuantas pasadas se realizaron.

34	14	26	15	98	90	125	429	537	650

Método de Selección

Este método consiste en buscar el elemento más pequeño del arreglo y ponerlo en la primera posición; luego, entre los restantes, se busca el elemento más pequeño y se coloca en segundo lugar, y así sucesivamente hasta colocar el último elemento.

Por ejemplo, dado el arreglo:

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Los pasos a seguir son:

4	21	40	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se coloca el más pequeño, el 4, en la primera posición

se cambia el 4 por el 40

4	9	40	21	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se coloca el 9, en la segunda posición

se cambia el 9 por el 21

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se coloca el 10, en la tercera posición

se cambia el 10 por el 40

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

El 21 continua ubicado en la cuarta posición

4	9	10	21	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se coloca el 35, en la quinta posición: se cambia el 35 por el 40, quedando ordenados 2 elementos

El algoritmo es el siguiente:

```
Algoritmo Seleccion
Comienzo
Constante n=6
void carga ( entero a[n])
Comienzo
entero i
   Para i Desde 0 Hasta n-1
        Leer  a[i]
   FinPara
retorna()
Fin
```

```
void ordenar ( entero a[n] )
Comienzo
entero  i , j, min, aux
  Para i Desde 0 Hasta  n-2
  min=i
     Para j Desde  i+1 hasta  n -1
          Si (a [ j] < a [min])          /*busca el mínimo entre los elementos a[i] .....a[n-1] */
              Entonces     min =j
           FinSi
     FinPara
     aux= a [i]                /*intercambia los valores */
     a [i] = a [min]
     a [ min] = aux
   FinPara
retorna()
Fin
```

```
void mostrar_ordenado (  entero a[n] )
Comienzo
entero i,
    Para i Desde 0 Hasta n-1
       Escribir a[ i ]
       FinPara
retorna()
Fin
```

```
/* Algoritmo Principal /*
Comienzo
entero   ar[n]
    cargar (ar)
    ordenar(ar)
    mostrar_ordenado (ar)
Fin
```

Se observa que para ordenar el arreglo son necesarias N-1 pasadas, ya que en la última se ubican los elementos N-1 y N-ésimo.

Actividad 15

Realizar la traza del algoritmo indicando los distintos valores que toman las variables para ordenar el arreglo:

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Método de Inserción directa

Sea **a** un arreglo de N componentes que debe ordenarse en forma ascendente. Este método consiste en insertar un elemento a[i], en el lugar que le corresponde entre los anteriores a[0]....a[i-1], que ya están ordenados.
Para ello se comienza con el segundo elemento del arreglo, si los dos primeros elementos están desordenados, los intercambia. Toma ahora el tercer elemento y busca la ubicación que le corresponde respecto de los 2 anteriores. Si a[2] es mayor o igual que a[1] significa que está en la posición correcta. En caso contrario, debe comparar a[2] con a[0]. Si a[2] es mayor o igual que a[0] lo inserta entre a[0] y a[1], pero si es menor que a[0], debe colocarlo en la posición a[0] y correr a[1] y a[2] una posición a la derecha.
En general, para insertar el elemento que está en la i-ésima posición debe compararlo con los i-1 elementos anteriores y ubicarlo en la posición que le corresponde.
Este proceso se repite N-1 veces, hasta que quedan todos en su posición correcta.

```
Algoritmo Insercion
Comienzo
Constante n=6
void carga ( entero a[n])
Comienzo
entero i
    Para i Desde 0 Hasta n-1
        Leer  a[i]
    FinPara
retorna()
Fin
```

```
void ordenar ( entero a[n] )
Comienzo
entero  i , j ,valor
Para i Desde 1 Hasta  n-1
valor  = a[i]
 j= i-1
     Mientras (( j ≥ 0 ) y ( valor < a [ j ] ))
         a [ j+1]= a [j]                  /* los valores se corren un lugar a la derecha */
         j=j-1
      FinMientras
      a[j+1] = valor
FinPara
retorna()
Fin
```

```
void mostrar_ordenado (  entero a[n] )
Comienzo
entero i,
 Para i Desde 0 Hasta n-1
    Escribir a[ i ]
  FinPara
retorna()
Fin
```

/* Algoritmo Principal /*
Comienzo
entero ar[n]
  cargar (ar)
  ordenar(ar)
  mostrar_ordenado (ar)
Fin

Para ordenar en forma ascendente el arreglo **a**:

40	21	4	9	10	35
0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 1:

21	40	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

En la primera pasada, el número 21 al ser menor que 40, es colocado en la posición 0, realizando antes el desplazamiento del 40 hacia la derecha.

Pasada 2:
En la segunda pasada, al ser el número 4 menor que 40, este último se corre a la posición 2 y como el número 21 también es mayor que 4 se corre a la posición 1. El 4 es colocado en la posición 0.

4	21	40	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 3:
En esta pasada el número 9 es el que se ubica en la posición que le corresponde.

4	9	21	40	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 4:

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 5:

4	9	10	21	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Actividad 16

Realizar la traza del algoritmo indicando los distintos valores que toman las variables para ordenar el arreglo

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Clasificacion de Metodos de Ordenamiento

El ordenamiento es una operación que permite clasificar los elementos de un arreglo en un orden secuencial de acuerdo a un criterio. Se efectúa en base al valor de algún dato del arreglo. Su propósito principal es el de facilitar las búsquedas de elementos en un conjunto ordenado, ademas de los listados.
El ordenar un grupo de datos significa mover los datos para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético ya sea ascendente o descendente.
Hay métodos muy simples de implementar (elementales) que son útiles en los casos en dónde el número de elementos a ordenar no es muy grande (ej. menos de 1000 elementos). Por otro lado hay métodos sofisticados, más difíciles de implementar pero que son más eficientes en cuestión de tiempo de ejecución.

Los métodos de ordenamiento se pueden clasificar en:

Ordenamientos Elementales	Ordenamiento no Elementales
1. Burbujeo 2. Selección 3. Inserción	1. Shell 2. Quick Sort 3. Fusión
Estos son los métodos estudiados en esta unidad.	Estos son métodos más complejos, algunos de ellos utilizan técnicas de programación recursiva la cual se estudia en cursos superiores.

3.6 Bibliografía

- Braunstein Silvia y A. Gioia (1987)"Introducción a la Programación y a la Estructura de Datos" Eudeba Buenos Aires. Argentina
- Brassard, G. & Bratley, P. "Fundamentos de Algoritmia". Prentice-Hall. 1997.
- Criado Clavero, MªAsunción.(2006) "Programación en Lenguajes Estructurados" Alfaomega -RaMa
- De Giusti, Armando E., Madoz Maria y otros.(2001) "Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci. "LIDI. Facultad de Informática Universidad Nacional de la Plata.
- Deitel, H M y Deitel, P J (2003) "Como programar en C/C++" Pearson Educación– Hispanoamericana Duperut,Gabriel (1996) "El computador y su entorno" Mendoza
- Joyanes Aguilar A y Zahonero Martinez, (2001)"Programación en C. Metodología, estructuras de datos y objetos" Mc Graw - Hill
- Joyanes Aguilar Luis (2004) "Algoritmos y Estructuras de Datos. Una Perspectiva en C" Segunda Edición Mc Graw Hill
- Polya George (1972) "Cómo plantear y resolver problemas". México. Editorial Trillas
- Rueda Sonia V y Alejandro J. García, Programa de Ingreso 2003. Análisis y Comprensión de Problemas.Fundamentos, Problemas Resueltos y Problemas Propuestos. Departamento de Ciencias e Ingeniería de la Computación. Universidad Nacional del Sur. Noviembre de 2002.

Práctico Unidad 3: Arreglos y Registros

Resultados de Aprendizaje:

- Comprender e implementar el concepto del tipo de dato Arreglo y Registro.
- Distinguir tipos de datos simples y estructuras de datos.
- Usar correctamente y manera óptima las operaciones básicas sobre estructuras de datos.
- Realizar seguimientos con estructura de datos.
- Capacidad para diseñar y construir subprogramas y algoritmos de manera óptima.
- Resolver situaciones problemáticas con iniciativa, autonomía y creatividad.
- Adquirir responsabilidad y compromiso por su propio aprendizaje.
- Desarrollar capacidad para comunicar sus puntos de vista en forma oral y escrita.
- Participar en grupos de trabajo, respetando las ideas de sus compañeros.

Ejercicio 1
Realizar el seguimiento de los siguientes algoritmos.
A. Lote de prueba: 25, 28, 47, 63, 6, 15, 71

Algoritmo Seguimiento

```
void carga ( entero xa[5])
Comienzo entero i
Para i Desde 0 Hasta 4
        Leer xa[ i ]
FinPara
Retorna()
Fin

void calculo ( entero xa[5])
Comienzo entero i
Para i Desde 0 Hasta 4
        Si ( (xa[ i ] resto 2) == 0)
                entonces xa[ i ] = (i * 2)+1
        FinSi
FinPara
Retorna()
Fin

void mostrar ( entero x[5])
Comienzo entero i
Para i Desde 0 Hasta 4
        Escribir x[ i ]
FinPara
Retorna()
Fin

/* Algoritmo Principal */
Comienzo
entero a[5]
carga(a)
calculo(a)
```

```
mostrar (a)
Fin
```

B. Lote de prueba**:** 90, 52, 86, 31, 94, 13, 98

```
AlgoritmoSeguimiento
void carga ( entero m[5])
Comienzo entero i
Para i Desde 0 Hasta 4
        Leer m[ i ]
FinPara
Retorna()
Fin

void calculo ( entero m[5])
Comienzo entero i
Para i Desde 0 Hasta 4
        Si ( (i resto 2)!=0)
                entonces m[ i ] =(m[ i ] div 3 ) + 2
        FinSi
FinPara Retorna()
Fin

void mostrar ( entero x[5])
Comienzo entero i
Para i Desde 0 Hasta 4
         Escribir x[ i ]
FinPara
Retorna()
Fin
/* Algoritmo Principal */
Comienzo
entero m1[5]
   carga(m1)
   calculo(m1)
   mostrar (m1)
Fin
```

Ejercicio 2

Completar para que el algoritmo resuelva el siguiente problema:

Se cuenta con las temperaturas máximas registradas durante cada día del mes de enero de 2019 en la ciudad de San Juan.

1. Mostrar la temperatura promedio mensual.
2. Indicar la cantidad de días en el mes que no fueron inferiores a la temperatura promedio mensual.
3. Mostrar el/los números de día cuya temperatura máxima varió entre dos valores ingresados por teclado, sin superar los 42°C.
4. Una vez completado el algoritmo realizar el seguimiento, indicando pre y post condiciones.

```
Algoritmo Temperaturas_enero
…… carga (       )
Comienzo entero i
Para i Desde 0 Hasta N-1
        Leer ……..
```

```
FinPara
retorna(  )
Fin

…….. promedio (      )
Comienzo
entero i
real s, prom s =…….
Para i Desde 0 Hasta .....
        s=s + ……
FinPara
prom= s/N
retorna (         )
Fin

…… cantidad (entero t[N]), entero xp)
Comienzo
entero i, ct
ct = …
Para i Desde 0 Hasta N-1
        Si (t[i] ……xp)
                entonces ct = …+1
        Finsi
FinPara
Escribir “La cantidad…….....son:”, ….
retorna ( )
Fin

……  muestra (         )
comienzo
entero i, temp1, temp2 Para i Desde 0 Hasta N-1 Leer temp1
Leer temp2
Si (temp2 > temp1) Y (temp2 <=42)
        entonces
   Si ((.... >temp1) Y (a[i]       temp2))
        entonces
            Escribir “el día”,     ,“varió su temperatura entre”, temp1 “°C y”temp2 “°C”
……
        sino Escribir “temperatura superior a 42°C”
  FinSi
FinPara
retorna ()
Fin

/*---Algoritmo principal--*/
Comienzo
constante N=....
entero temp [N], ……..
carga ()
p = promedio (          )
Escribir “el promedio mensual de temperaturas de enero de 2019 es “…….
cantidad (……, p)
muestra(temp)
Fin
```

Ejercicio 3

Escribir un algoritmo que mediante subprogramas permita:

1. Generar y mostrar un arreglo de 10 componentes enteras.
2. Sumar al tercer elemento del arreglo un valor ingresado por teclado, y mostrar el arreglo modificado.
3. Intercambiar el elemento de la décima posición con el quinto elemento y mostrar el arreglo resultante.
4. Mostrar el resultado de multiplicar las componentes impares del arreglo por un valor ingresado por teclado.

Ejercicio 4

Escribir un algoritmo que usando subprogramas permita:

1. Generar un arreglo de 25 elementos con los números pares comprendidos entre 30 y 80.
2. Calcular el promedio y mostrar cuántas componentes son mayores al promedio.
3. Mostrar las posiciones de los elementos del arreglo menores que el promedio.
4. Calcular y mostrar la suma de aquellas componentes que se encuentran en las primeras 10 posiciones y que no superan el promedio.

Ejercicio 5

Diseñar un algoritmo que usando subprogramas permita:

1. Cargar dos arreglos A y B de M componentes enteros cada uno.
2. Generar un nuevo arreglo que almacene el resultado de multiplicar el primer elemento de A con el último elemento de B, el segundo de A con el penúltimo de B y así sucesivamente.
3. Mostrar el nuevo arreglo en orden inverso.
4. Calcular y mostrar el producto escalar de los arreglos A y B.

Nota: El producto escalar se calcula: A[1]*B[1] + A[2]*B[2]+ A[3] *B[3]+…+ A[n] *B[n] = k siendo k un número entero

Ejercicio 6

Realizar un algoritmo que usando subprogramas permita:

1. Cargar un arreglo V de 12 elementos enteros.
2. Generar un arreglo N cuyas componentes serán los índices de las componentes del arreglo V cuyos valores sean múltiplos de 6.
3. Hacer el seguimiento del algoritmo con el siguiente lote de prueba: V

24	512	126	630	54	246	96	174	60	455	822	720
0	1	2	3	4	5	6	7	8	9	10	11

Ejercicio 7

Una empresa de venta de artículos para el hogar cuenta con 36 sucursales, las cuales están identificadas por un número de 1 a 36.

El ingreso de la información del importe total recaudado por sucursal se hace consecutivamente en forma ordenada por número de sucursal.

Realizar un algoritmo que usando subprogramas y estructuras de datos adecuadas permita:

1. Calcular cual fue la mayor recaudación de la empresa.
2. Indicar la o las sucursales que registraron la mayor recaudación.
3. Informar cuál es la sucursal con menor recaudación.
4. Mostrar el porcentaje que recaudó la sucursal 19 respecto del total recaudado.
5. Realizar las modificaciones necesarias para responder a todos los ítems anteriormente

planteados, suponiendo que, de cada sucursal se registra la siguiente información: número de sucursal e importe total recaudado, es decir la información viene desordenada.

Ejercicio 8

ENERGAS ha autorizado en la provincia a 15 talleres para extender obleas de habilitación de carga de GNC. Cada taller se identifica con un número del 1 al 15.
La información de los talleres puede venir en cualquier orden pero sin repetir el número de taller
Mensualmente se registra: número de taller y cantidad total de obleas emitidas. Realizar un algoritmo que utilizando subprogramas y estructuras adecuadas permita:

1. Ingresar por teclado un número de taller y mostrar la cantidad de obleas emitidas.
2. Indicar la cantidad total de obleas emitidas en la provincia.
3. Indicar el taller que menos oblea extendio.
4. Mostrar el número de taller que emitió mayor cantidad de obleas.
5. Informar si algún taller emitió más de 500 obleas en el mes.
6. Realizar las modificaciones necesarias para responder a todos los ítems anteriormente planteados, suponiendo que la información viene ordenada por número de taller. Es decir, que por cada uno de los 15 talleres se ingresa la cantidad de obleas emitidas.

Ejercicio 9

Un instituto de enseñanza ofrece 12 cursos. Por cada inscripción se ingresa el nombre del curso elegido.
Se pide construir un algoritmo que permita:

1. Generar un listado que muestre el nombre de los cursos del instituto y la cantidad total de inscriptos, ordenado por total de inscriptos.
2. Indicar el nombre del curso que tiene menos cantidad de inscriptos.
3. Mostrar la cantidad total de inscriptos del instituto.
4. Generar una nueva estructura que almacene los cursos que tuvieron menos de 30 inscriptos y mostrar los nombres de los mismos

Ejercicio 10

Complete los siguientes algoritmos para que resuelvan las situaciones problemáticas planteadas

A. Para estudiar el problema de la deserción en los primeros años de la Universidad, se realiza una encuesta a los alumnos que no aprobaron el parcial de una asignatura. De cada encuestado se tienen los siguientes datos*: código del departamento donde vive* (1: Capital, 2: Caucete, …. 19: Zonda) y *cantidad de horas dedicada al estudio*.

Se pide:

1. Mostrar por cada departamento la cantidad de horas de estudio promedio
2. Realizar el seguimiento del algoritmo una vez completado. Generar el lote de prueba.

Algoritmo Encuesta

```
N=19
Registro departamento
{ cadena nom
 entero canth
 entero cantal }

void  incializa_dep ( )
comienzo
entero i
Para i desde 0 hasta ……
::::::::::::::::::::::
::::::::::::::::::::::
```

```
Fin para
Retorna ()
fin
```

```
void carga_encuestas( departamento xa [N])
Comienzo
entero coddep, ch
Leer coddep
Mientras (coddep !=0)
    Leer ch
    xa[……].canth= ……
    xa[……]. cantal=……
    Leer coddep Fin mientras Retorna ()
Fin
```

```
void  Horas_promedio (      )
Comienzo Entero i
Para i desde 0 hasta ……
        Prom= ……..
        Escribir ("cantidad de horas promedios de estudio del departamento",      ,
                "es", ……)
Finpara
Retorna ()
Fin
```

```
/* algoritmo principal */
Comienzo
departamento a[N]
  incializa_dep ( a)
  carga_encuestas (   )
  Horas_promedio(… )
Fin
```

B. Para estudiar el problema de la deserción en los primeros años de la Universidad, se realiza una encuesta a los alumnos que no aprobaron el parcial de una asignatura. De cada encuestado se ingresa *el código de motivo de abandono* (1: "no me gusta la carrera", 2:"no tengo claro qué me gusta", 3:"no le dedico tiempo al estudio"… 20:"me falta voluntad")

Se pide:

1. Indicar cuáles son los motivos de mayor abandono.
2. Una vez completado el algoritmo, realizar el seguimiento para un lote de prueba generado.

```
Algoritmo encuesta
M = 20
registro motivo
{ cadena nommot
  entero cantal }
```

```
void  inicializar_ mot (         )
Comienzo
entero i,
Para i desde 0 hasta ……
…….
……
```

```
Fin para
Retorna ()
Fin


void Carga_encuestas (motivo xb[M] )
Comienzo
Entero i, codmot
Leer codmot
Mientras( codmot !=0 )
    xb[……..].cantal= …….
    Leer codmot
Finmientras
Retorna ()
Fin
entero Max_Mot (     )
Comienzo entero i, ……
max=xb [0].cantmot
Para i desde … hasta ……
… .
….
Finpara
Retorna (       )
Fin


void  Motivo_max (  )
Comienzo entero i, ……
Para i desde 1 hasta ……
…
Finpara
Retorna (       )
Fin

/* algoritmo principal */
Comienzo
  motivo b[M]
 entero MM
       inicializar_mot (b)
       Carga_encuestas (b)
       MM=Max_Mot (b)
       Motivo_max (MM, b)
Fin
```

Ejercicio 11

La Facultad posee la información de los N alumnos que desean rendir examen en el siguiente turno de examen. Se sabe que la cantidad máxima de alumnos que se inscriben por turno es de 280.
De cada uno de los inscriptos se conoce: código de la carrera (1 a 8) y código de la asignatura (1 a 220).
Realizar un algoritmo que usando subprogramas y estructuras de almacenamiento adecuadas, permita:

1. Ingresar un código de asignatura e indicar el total de alumnos inscriptos en ella.
2. Informar, por cada carrera, el total de alumnos que se presentan a examen.
3. Mostrar los códigos de asignaturas en las cuales no hubo inscriptos.

4. Indicar cual es la materia que más alumnos tiene para rendir.
5. Generar una estructura adecuada donde se almacene todas las materias con menos de 50 alumnos inscriptos.
6. Indicar cual es la carrera que menos alumnos tiene para rendir.

Ejercicio 12

La aduana Paso de los Libertadores desea realizar un análisis sobre cada uno de los vehículos que cruzaron la frontera Argentina-Chile en enero pasado y para cada uno de los 31 días del mes almacena en estructuras adecuadas, la siguiente información: tipo de vehículo (1: auto, 2: camioneta, 3: colectivos, 4: camiones), lugar de procedencia ('S': San Juan, 'M': Mendoza, 'C': Córdoba, 'L': La Rioja, 'T': Otra provincia Argentina, 'P': Otro país) y cantidad de ocupantes por vehículo.

Realizar un algoritmo que utilizando subprogramas, permita:

1. Determinar total de autos que cruzaron durante todo el mes.
2. Indicar el/los día/s del mes en el que pasaron la mayor cantidad de colectivos.
3. Generar y mostrar un listado que contenga la cantidad de camiones por lugar de procedencia indicando el nombre del lugar.
4. Señalar el total de personas que pasaron por la Aduana en cada tipo de vehículo.

Ejercicio 13

El EPRE ha realizado un relevamiento sobre los cortes de servicio que se produjeron en la provincia durante el año 2017. Por cada uno de los 19 departamentos, se ingresa: nombre del departamento y la duración de cada corte medido en horas. El ingreso de la información por cada departamento finaliza con corte = 0.

Construir un algoritmo que con subprogramas permita:

1. Informar la cantidad de cortes registrados por cada departamento.
2. Indicar el nombre del o los departamentos donde se registraron la menor cantidad de cortes.
3. Mostrar la duración promedio de cortes por departamento.
4. Indicar porcentaje de departamentos que no registraron cortes en relación a los 19 departamentos.
5. Emitir un listado donde se muestre el número de departamento, cantidad de cortes y cantidad de horas de cortes acumuladas por departamento.

Ejercicio 14

Una distribuidora de bebidas de nuestra provincia comercializa 50 productos. Estos productos son vendidos a 20 comercios locales. Por cada comercio se ingresan los siguientes datos: CUIL del comercio y todas las compras realizadas por ese comercio a la distribuidora de bebidas. Por cada compra, los datos ingresados son: Nombre del producto comprado y cantidad de unidades de éste.

Construya un algoritmo que usando subprogramas permita:

1. Almacenar, en estructuras adecuadas, por cada uno de los 50 productos: el nombre y el precio unitario.
2. Por cada comercio, mostrar el importe total que tiene que abonar.
3. Generar una estructura que contenga el CUIL de aquellos comercios que compraron más de 500 unidades, indicar cuántos son.
4. Dado el CUIL de un comercio indicar si se encuentra entre los comercios que compraron más de 500 unidades
5. Mostrar por cada uno de los 50 productos la cantidad total de unidades que la distribuidora debe tener para satisfacer los pedidos realizados.
6. Indicar si los productos más vendidos son los más baratos.
7. Indique la cantidad de comercios que compraron 500,501…520 unidades en total

Ejercicio 15

Una lomoteca posee 7 tipos de menú (1:pachatas carne vacuna, 1:pachatas carne cerdo, 2:lomos carne vacuna, 3: lomos carne cerdo, 4:hamburguesas, 5:piadinas carne vacuna, 6: piadinas carne cerdo, 7:pizzas) y cuenta con 5 mozos.

Se tiene almacenado en estructuras adecuadas el nombre de cada mozo ordenado alfabéticamente, de cada menú su precio unitario y denominación.

Por cada día se registran de las ventas realizadas: código de menú, cantidad de unidades, nombre del mozo que atiende el pedido y propina. La información no tiene ningún orden en particular.

Realizar un algoritmo que utilizando subprogramas óptimos permita:

1. Al finalizar el día, mostrar la cantidad de ventas realizadas y el importe total recaudado por la lomoteca.
2. Calcular y mostrar la cantidad de lomos y pachatas vendidos.
3. Indicar la denominación del menú del que más unidades se vendió y la denominación del menú del que menos se recaudó.
4. Generar un listado que contenga nombre del mozo y el total de propina recaudada por sus ventas, ordenado por total de propina.
5. Realizar las modificaciones que considere conveniente, para responder a todos los ítems anteriormente planteados, sabiendo que ahora la información viene ordenada por mozo. Es decir por cada mozo se registra; nombre del mozo, código de menú, cantidad de unidades y propina. Cada mozo puede realizar más de una venta.

Ejercicio 16

La policía de San Juan ha lanzado el “Operativo Sol” para controlar el tránsito durante el verano. Se ha dividido a la provincia en 12 zonas. Se han clasificado las infracciones en 10 tipos (1: exceso de velocidad, 2: falta de iluminación,…, 10: alcoholemia).

Por cada infracción realizada se tienen los siguientes datos: *patente del vehículo, zona y tipo de infracción*.

Para evaluar los resultados del Operativo realizar un algoritmo que con subprogramas permita:

1. Mostrar el o los tipos de infracción/es que más se cometen.
2. Decir si algún tipo de infracción no se cometió.
3. Indicar la cantidad de infracciones realizadas en cada zona.
4. Mostrar todas las zonas que tuvieron una cantidad de infracciones inferior al promedio de las infracciones realizadas en las 12 zonas de la provincia.

Ejercicio 17

La Agencia de Turismo quiere determinar el grado de aceptación en 10 destinos diferentes. Se ha realizado una encuesta a 2500 personas. Por cada encuestado se registra: *edad* y *código del destino que prefiere* (1..10).

Se han determinado las siguientes categorías de acuerdo a la edad 1:Niños (< 12 años), 2:Adolescentes (< 18 años), 3:Jóvenes (< 25 años), 4:Adultos (< 70 años), 5:Jubiliados (> 70 años).

Construir un algoritmo que con subprogramas, luego de procesar las encuestas, informe:

1. El total de encuestados por cada una de las categorías de edad.
2. El (los) destino(s) con mayor preferencia.
3. El porcentaje de encuestados que eligieron los destinos correspondientes a excursiones de fin de semana, respecto del total de personas que respondieron la encuesta. Se sabe que las excursiones de fin de semana están codificadas con los números 7, 8 y 9.

Ejercicio 18

La Secretaría de Producción de la provincia está recibiendo los siguientes datos referidos a la producción de la provincia de San Juan, registrados durante el año 2017.

Además se tiene un registro del nombre de cada productor, con 100 Productores en total.

El ingreso de datos se registra de manera ordenada por Nº de Departamento, y por cada Departamento, se ingresan los siguientes datos de cada entrega: Apellido y Nombre del productor, código del producto (1:papa, 2:ajo, 3:tomate...20:cebolla) y cantidad producida en Kg.

Construir un algoritmo con subprogramas que:

1. Informe la cantidad de kg entregados por cada producto.
2. Muestre un listado ordenado alfabéticamente que contenga nombre de cada uno de los 19 departamentos y la cantidad de productores.
3. Muestre el nombre de aquellos productores que no realizaron ninguna entrega.
4. Indique el nombre del o los productos con mayor producción tuvieron.
4. Muestre el nombre del departamento que registró la menor cantidad de entregas.
5. Dado el nombre de un producto indicar si su producción supera los 100.000 kg.
6. Generar un listado ordenado ascendentemente por total de Kg, indicando por cada uno de los 20 productos: Nº de producto y total de Kg.

Ejercicio 19

Una Empresa Constructora almacena la siguiente información de cada uno de los 110 aspirantes a un nuevo barrio a construirse: número aspirante, apellido y nombre, cantidad de Integrantes del grupo familiar, cantidad de cuotas e importe de cuota. El ingreso de la información no posee ningún orden.

Realizar un algoritmo que utilizando subprogramas permita:

1. Emitir un listado con los aspirantes que han pagado más de $ 7.500, indicando apellido y nombre, cantidad de cuotas e importe total abonado a la empresa.
2. Calcular el importe total recaudado por la empresa.
3. Dado un apellido y nombre que se ingresa por teclado, indicar si es aspirante o no.
4. La empresa ha generado un método de selección el cual consiste en otorgar una casa a los aspirantes que abonaron un importe superior a $ 7.500 y tienen familia numerosa (cantidad de integrantes del grupo familiar sea mayor a cinco). Generar una nueva estructura con los números de aspirantes seleccionados.
5. Mostrar nombre y apellido de cada uno de los aspirantes seleccionados.

Ejercicio 20

Se cuenta con los siguientes datos de 70 familias: un número de identificación de cuatro cifras, el ingreso anual y el número de miembros de una familia.

Realizar un algoritmo utilizando subprogramas permita:

1. Registrar la información detallada en una estructura de datos conveniente.
2. Clasificar las familias en alguna de las diez categorías de acuerdo al número de miembros (1: un miembro, 2: dos miembros 3: tres miembros 4: … 10: diez o más miembros).
3. Mostrar la cantidad de familias por cada una de las 10 categorías.
4. Calcular el promedio de los ingresos anuales de las familias.
5. Listar los números de identificación y los ingresos de las familias que exceden el ingreso anual promedio.
6. Almacenar en una estructura auxiliar los números de identificación de las familias cuyo ingreso anual es inferior al ingreso anual promedio y cuya cantidad de integrantes del grupo familiar es mayor que 5.
7. Dado el número de identificación de una familia, indicar si pertenece al grupo de familias cuyo ingreso anual es inferior al ingreso anual promedio y cuya cantidad de integrantes del grupo familiar es mayor que 5.

Ejercicio 21

En el Hospital Rawson se desea realizar un análisis estadístico para obtener información de los pacientes atendidos por cada una de las 12 especialidades (1:pediatría, 2:ginecología ..12::traumatología), las cuales están registradas en una estructura adecuada.

Se encuentran almacenados los nombres de los 19 departamentos ordenados alfabéticamente.

Por cada paciente que asiste al nosocomio se solicitan los siguientes datos que son ingresados ordenados por especialidad: Sexo, Edad, Tipo de atención (‘A’: ambulatorio, ‘I’: internación, ‘U’:urgencia) y nombre del departamento dónde vive

Se pide redactar un algoritmo con subprogramas que permita:

1. Realizar un listado que muestre por nombre de especialidad: la cantidad de pacientes cuya edad está comprendida 20 y 30 años, y que fueron atendidos de urgencia.
2. Indicar por nombre de especialidad la edad promedio de pacientes atendidos
3. Mostrar por nombre de departamento la edad promedio de pacientes atendidos.
4. Señalar la cantidad de mujeres por cada uno de los tipos de atención.
5. Emitir un listado indicando por cada departamento: Nombre de Departamento y cantidad total de pacientes atendidos. Este listado debe aparecer ordenado, descendentemente, por cantidad de pacientes atendidos

Ejercicio 22

Una empresa de turismo procesa información de los distintos contingentes que a lo largo del año contrataron sus servicios.

Se ingresan inicialmente los nombres de los 8 coordinadores y de las 23 provincias. Por cada contingente que viajó se ingresa: código del coordinador (1 a 8), código de la provincia (1 a 23), y cantidad de personas del contingente; finalizando el ingreso de información con código de coordinador igual a 0.

Realizar un algoritmo que usando estructuras de datos adecuadas y subprogramas permita:

1. Mostrar el/los nombre/s de coordinador/es que acompañaron mayor cantidad de contingentes.
2. Escribir el código y nombre de la provincia a la cual viajó la menor cantidad de contingentes.
3. Ingresar un código de provincia e indicar la cantidad de personas que llegaron a dicha provincia.
4. Ingresar el nombre de un coordinador e indicar si acompañó a menos de 100 personas.
5. Realizar las modificaciones que considere conveniente, para responder a todos los ítems anteriormente planteados, sabiendo que ahora la información viene ordenada por provincia. Es decir por cada uno de las 23 provincias se registra: código del coordinador (1 a 8) y cantidad de personas del contingente. Puede existir más de un contingente que viajó a la provincia.

Ejercicio 23

Una entidad bancaria posee los siguientes datos de cada uno de sus 300 clientes: número de cliente (numerados de 1000...1299), nombre y saldo.

Al finalizar una jornada de trabajo se procesan las transacciones realizadas. De cada transacción se conoce: número de cliente, código de transacción (1: Depósito, 2: Extracción) e importe de la transacción.

Realizar un algoritmo que permita:

1. Realizar un listado con los datos actualizados de los clientes.
2. Ingresar por teclado un número de cliente y decir si su saldo es inferior a $ 2.500.
3. Indicar el número de cliente con mayor importe depositado. Suponer único.
4. Indicar el número de cliente con mayor importe extraído. Suponer único.
5. Listar en forma ordenada los nombres de los clientes que no operaron ese día.
6. Mostrar el importe total registrado, en la entidad bancaria, en cada uno de los dos tipos de transacciones.

Ejercicio 24

La administración de PAMI desea procesar la siguiente información correspondiente a los 300 afiliados, mayores de 80 años que se atendieron en un sanatorio. Por cada paciente se registró: DNI, nombre del paciente y por cada tipo de prestación realizada: código y cantidad de prestaciones.

Se cuenta con un registro que almacena el costo de cada una de las 40 prestaciones. El ingreso de información por paciente finaliza con código de prestación igual a cero. Se sabe que son 40 los tipos de prestaciones distintas codificadas a partir de 1: 1Radiografía tórax, 2Electrocardiograma, …40Análisis de sangre

Realizar un algoritmo que usando subprogramas:

1. Indique a cada paciente el importe total que debe abonar por las prestaciones recibidas.
2. Muestre el nombre de la/s prestación/es que se realizaron a mayor cantidad de pacientes.
3. Permita indicar los pacientes que recibirán una bonificación, sabiendo que los bonificados por la obra social son los 20 pacientes que abonaron mayores importes por las prestaciones recibidas. Realice un listado que contenga número de documento, nombre del paciente e importe que abonan los que recibirán la bonificación. Se debe usar un subprograma que implemente un algoritmo de ordenamiento.
4. Indique si a un paciente le corresponde la bonificación, ingresando previamente su DNI por teclado.
5. Escriba el porcentaje que representa la prestación electrocardiograma en relación al total de prestaciones realizadas a los 300 afiliados.

Ejercicio 25

Un local comercial de ventas de repuestos de automotores desea obtener cierta información sobre todas las ventas registradas en un periodo de tiempo dado. Para ello el comercio cuenta con los siguientes datos referidos a los 250 artículos que están a la venta: *nombre, precio unitario, stock, código*

También, se tienen los siguientes datos referidos a cada una de las ventas efectuadas en ese periodo de tiempo: *nombre del artículo* y *cantidad de unidades vendidas.*
Nota: el **código** es un número entero, el código del primer artículo es 300. Estos datos deberán ser almacenados en una estructura adecuada.

Codificar un algoritmo que permita emitir los siguientes informes, luego de terminar de procesar todas las ventas:

1. Los nombres de aquellos artículos que quedaron con stock nulo.
2. El stock de un artículo cuyo código se ingresa previamente por teclado.
3. Los nombres de los 20 artículos que quedaron con mayor stock.
4. Para cada una de las cantidades de stock entre 1 y 10 inclusive, indicar la cantidad de artículos que la tienen.

Ejercicio 26

Una fábrica elabora 80 productos, de los que debe registrar: Nombre, precio unitario, stock y código de rubro al que pertenece (1..12).

Se sabe que los rubros son 12: 1: dulces, 2: lácteos ,..,12:aceites.

Por otra parte, se tienen los siguientes datos de cada venta efectuada por la fábrica a 7 hipermercados. Por cada hipermercado se conoce: Nombre del hipermercado y por cada artículo vendido: nombre de artículo y cantidad de unidades vendidas.

La venta por cada hipermercado finaliza con nombre de artículo Nada más Se pide construir un algoritmo que usando subprogramas óptimos permita:

1. Indicar el stock de cada artículo después de realizar las ventas.
2. Mostrar el nombre de cada uno de los 12 rubros y el importe total vendido para cada uno de ellos.
3. Realizar un listado que contenga nombre del hipermercado y el importe total que debe pagar, el listado debe estar ordenado descendentemente por importe.
4. Indicar los importes máximos y mínimos vendidos y el nombre de los supermercados a los que corresponde cada uno.
5. Realizar las modificaciones que considere conveniente, para responder a todos los ítems anteriormente planteados, sabiendo que ahora la información no tiene ningún orden. Es decir por cada uno de las ventas se conoce: Nombre del hipermercado nombre de artículo y canti-

dad de unidades vendidas.

Ejercicio 27

El IPV cuenta con la siguiente información correspondiente a los adjudicatarios que adeudan las 2 últimas cuotas: DNI, Nombre de adjudicatario y código de plan de pago (1.. 5).

Se ingresa en forma ordenada ascendente por DNI en una estructura adecuada los datos que corresponden a los 80 deudores de una cuota y en otra estructura los correspondientes a los 120 deudores de la última cuota.

Se sabe que son 5 planes de pago y por cada uno de ellos se almacena el importe de la cuota.

Utilizando subprogramas óptimos, se pide:

1. Generar una nueva estructura que contenga la información de los adjudicatarios que adeudan ambas cuotas. Indique cuántos son en total. La nueva estructura debe generarse recorriendo una sola vez cada una de las estructuras que contienen los datos de los deudores.
2. Ingresar por teclado un DNI e indicar cuantas cuotas adeuda.
3. Realizar un listado ordenado alfabéticamente por nombre que contenga, DNI, Nombre y Apellido e importe total adeudado, de los adjudicatarios que adeudan las 2 cuotas.
4. Indicar para cada código de plan cuantos adjudicatarios adeudan cuotas.

Ejercicios Propuestos

Ejercicio 1

Se cuenta con la información referida a 20 países y sus respectivas capitales:

Argentina	Buenos Aires
Canadá	Ottawa
……	…….
México	Distrito Federal de México

La información se ingresa ordenada alfabéticamente por nombre de país. Se pide:

1. Dado un nombre de una ciudad capital indicar a que país corresponde.
2. Dado un nombre de país e indicar cuál es su capital.

En ambos incisos emitir un mensaje en caso de no estar registrado el dato solicitado.

Ejercicio 2

Construir un algoritmo que permita:

1. Generar dos arreglos de 50 componentes enteras, uno guarda los números pares a partir de 2 y el otro los múltiplos de 5 a partir de 5.
2. Generar una nueva estructura de datos donde:

- en sus *posiciones impares*, se guarde el triple de los elementos que se encuentran en las posiciones pares del arreglo que guarda los números pares
- y en las *posiciones pares* de esta nueva estructura, se almacene el doble de los elementos que se encuentran en las posiciones impares del arreglo que guarda los múltiplos de 5.

3. Decir si alguna componente en el nuevo arreglo generado es superior a un número, ingresado previamente por teclado.

Ejercicio 3

Diseñar un algoritmo que permita:

1. Cargar un arreglo de 25 números enteros, de modo tal que a medida que se ingresan los datos, queden en forma ordenada ascendentemente.
Nota: Si la posición en la que debe ser ingresado un número está ocupada, tendrá que desplazar las componentes, de modo que sea posible la inserción. .
2. Mostrar el arreglo resultante.

Ejercicio 4

Una ferretería comercializa 300 productos; y cuenta con 25 empleados. Esta ferretería desea procesar las ventas realizadas durante el mes de abril pasado. Por cada día del mes de abril se ingresan todas las ventas efectuadas, teniendo en cuenta los siguientes datos por cada venta: Código de producto, Cantidad de unidades vendidas, Nombre del empleado que realizó la venta.

Realizar un algoritmo que, utilizando estructuras adecuadas, permita:

1. Almacenar el nombre de c/u de los 25 empleados. *Nota:* estos datos se encuentran ordenados alfabéticamente.
2. Almacenar el precio unitario de c/u de los 300 productos.
3. Por cada día, mostrar la cantidad de ventas realizadas y el importe total vendido por cada uno de los 25 vendedores.
4. Mostrar la cantidad de unidades vendidas, de cada uno de los 300 productos, en todo el mes de abril.
5. Indicar si los productos más caros fueron los menos vendidos.

Ejercicio 5

La Universidad Nacional de San Juan, cuenta con la información de los 150 alumnos becados. Por cada alumno becado se tiene los siguientes datos: Nombre, código de carrera (1 a 25), edad.

Además se posee los nombres de las 25 carreras de la UNSJ ordenados alfabéticamente.

Codificar un algoritmo que permita de manera óptima utilizando subprogramas y estructuras adecuadas:

1. Mostrar la cantidad de alumnos becados por carrera
2. Dado el nombre de una carrera mostrar la cantidad de alumnos becados.
3. Generar un listado con nombres de los alumnos cuya edad supera la edad promedio. Usar 2 subprogramas.
4. Dado el nombre de un alumno informar si su edad supera la edad promedio

Ejercicio 6

Una casa de venta de insumos informáticos comercializa 200 productos; y cuenta con 15 empleados. Este comercio desea procesar las ventas realizadas durante el mes de mayo pasado. Por cada día del mes de mayo se ingresan todas las ventas efectuadas, teniendo en cuenta los siguientes datos por cada venta: Código de producto, Cantidad de unidades vendidas, Nombre del vendedor.

Realizar un algoritmo que de manera óptima y utilizando estructuras de datos adecuadas permita:

1. Almacenar el nombre de c/u de los 15 empleados. *Nota*: estos datos se encuentran ordenados alfabéticamente.
2. Almacenar el precio unitario de c/u de los 200 productos.
3. Por cada día, mostrar la cantidad de ventas y el importe total vendido registrado por cada uno de los 15 vendedores.
4. Ingresar el nombre de un empleado e indicar el importe total vendido
5. Mostrar la cantidad de unidades vendidas, de cada uno de los 200 productos, en todo el mes de mayo.
6. Indicar si los productos más caros fueron los menos vendidos.
7. Indicar la cantidad de productos cuyo total vendido fue 90, 91, 92…105 unidades.

Ejercicio 7

Una compañía de materiales eléctricos que comercializa 100 productos, posee los siguientes datos de las ventas realizadas durante el mes de mayo pasado. Por cada uno de los 31 días del mes de mayo, se ingresan los siguientes datos de las ventas registradas: Número de producto y cantidad de unidades vendidas.

Se pide redactar un algoritmo que permita:

1. Registrar, en estructuras apropiadas, el nombre y el precio unitario de c/u de los 100 productos. Nota: suponer que los nombres se encuentran ordenados alfabéticamente.
2. Mostrar, por cada día, la cantidad de unidades pedidas de c/u de los 100 productos.
3. Indicar, por cada día, el nombre del producto menos vendido.
4. Mostrar el precio unitario de un producto cuyo nombre se ingresa por teclado.
5. Indicar la cantidad de productos de los que se vendieron en total 70, 71, 72…85 unidades.
6. Mostrar el importe diario recibido por la compañía.

Ejercicio 8

El Departamento Alumnos de nuestra facultad recibe las inscripciones de los 320 aspirantes a ingresar a las carreras que ofrece el Departamento de Informática. Por cada inscripción, se registra: nombre y código de colegio secundario en el que se graduó.

Además, se conocen los nombres de los colegios secundarios y a cada uno de ellos se le hace corresponder como código un número comprendido entre 1 y 82.

Realizar un algoritmo que usando subprogramas permita:

1. Listar nombre del alumno y nombre del colegio del cual proviene. Este listado debe aparecer ordenado alfabéticamente por nombre del alumno.
2. Indicar si un aspirante cuyo nombre se ingresa previamente por teclado registró inscripción.
3. Realizar un informe que contenga los nombres de los colegios y totales de inscriptos que pertenecieron a cada uno de los mismos.

Ejercicio 9

Se cuenta con los siguientes datos de cada uno de los 19 departamentos de la provincia de San Juan: número de departamento, superficie y población por km^2.

Almacenar la información, sabiendo que la misma no trae ningún orden particular. Diseñar un algoritmo que permita:

1. Dado el número de un departamento indicar cuál es su densidad (población/superficie).
2. Calcular el promedio de población por km^2 de los departamentos 3, 14 y 19.
3. Generar un listado ordenado descendentemente por superficie con el siguiente formato:

NUMERO DPTO. DENSIDAD

xxx xxxxx

xxx xxxxx

4. Dada una superficie indicar si corresponde o no a un departamento, en caso afirmativo mostrar todos los datos del mismo.

Ejercicio 10

La secretaría de Turismo de la Provincia ofrece una promoción para incentivar las visitas turísticas. Para ello, ha dividido a la provincia en 9 zonas de interés turístico y 10 categorías de grupos turísticos (categoría 1: una persona, categoría 2:dos personas,

…. categoría 10: diez personas).

Por cada turista se solicita la siguiente información: zona seleccionada (1..9), cantidad de días que visitará la zona y número de integrantes del grupo turístico.

Además se posee un tarifador con nombre de zona y costo de permanencia por día y por persona en cada una de las mismas.

Realizar un algoritmo que permita:

1. Mostrar la cantidad de turistas que visitaron cada una de las zonas.
2. Escribir el nombre de la o de las zonas donde se registró la máxima recaudación.

3. Mostrar la cantidad de grupos turísticos en cada categoría.
4. Mostrar el total de recaudación por zona

Ejercicio 11

Una compañía de telefonía celular registra las ventas de 45 empleados, por cada venta se ingresa: número de vendedor (100. 144) e importe de venta. El ingreso finaliza

con número de vendedor igual a cero. Realizar un algoritmo que permita:

1. Mostrar el total recaudado por cada empleado.
2. Indicar cuanto debe pagar la empresa en total por comisiones, sabiendo que a cada empleado se le paga una comisión del 3.5 % sobre las ventas totales realizadas.
3. Indicar número/s de/los empleados que registraron la venta mínima.
4. Dado un número de empleado, indicar si el total vendido es mayor a $ 1.675, mostrar el valor total de la venta.

Ejercicio 12

El INPRES debe realizar un estudio estadístico de los movimientos sísmicos en la provincia durante el año 2013. Por cada mes se deberán registrar las intensidades de los movimientos, dicho ingreso finaliza con intensidad nula.

Construir un algoritmo con subprogramas que permita:

1. Mostrar la intensidad promedio por mes.
2. Indicar en qué mes se registró la mayor intensidad promedio (suponerla única).
3. Mostrar el número de mes en donde la intensidad promedio fue superior a 3.5 y la cantidad de movimientos sísmicos no fue inferior a 8.

Ejercicio 13

El gobierno de la provincia de San Juan ha recabado los datos correspondientes a los productores damnificados por el granizo caído en distintos departamentos de la provincia.

Se tiene registrado el nombre y DNI de los 750 productores de la provincia.

Se cuenta con un registro, en orden alfabético, del nombre y precio por kg de los 50 cultivos que se producen en la provincia: 1ajo, 2cebolla, … 50uva de exportación. Por cada uno de los 19 departamentos se ingresa: DNI y nombre del productor, nombre de cultivo y cantidad en kg de la pérdida registrada. El ingreso de información finaliza con documento nulo.

Construir un algoritmo que usando subprogramas permita indicar:

1. Cantidad de productores damnificados en cada uno de los distintos tipos de cultivo.
2. Importe de los daños por cada tipo de cultivo.
3. DNI, nombre del productor que registraron pérdidas superiores a los 10000 kg.
4. Importe total de los daños para cada uno de los 50 cultivos.

Ejercicio 14

Se cuenta con la siguiente información correspondiente a 500 usuarios de servicios telefónicos corporativos: categoría de usuario, y por cada llamada realizada: código de llamada y cantidad de pulsos consumidos.

Se sabe que existen cinco categorias de usuarios: 1-familiar, 2-organización privada, 3-empresarial, 4: organización estatal, 5-profesional, y que tres códigos de llamada: 1urbana, 2-suburbana, 3:larga distancia.

El ingreso de información por usuario finaliza con código de llamada igual a cero. Realizar un algoritmo que mediante subprogramas permita:

1. Registrar el valor del pulso para cada código de llamada.
2. Escribir el importe total recaudado.
3. Indicar para cada una de las categorías de usuarios la cantidad de pulsos consumidos.
4. Escribir el nombre del código de llamada y el importe total recaudado por cada uno, orde-

nado por total.

Ejercicio 15

La administración de un consorcio posee los siguientes datos por cada uno de los 30 departamentos: Nombre y Apellido del titular y la cantidad de expensas que adeuda.

Construir un algoritmo que, utilizando subprogramas y estructuras de datos adecuadas:

1. Ingrese los datos indicados de cada uno de los 30 departamentos y registre el importe de cada una de las 12 expensas.
2. Indique, por cantidad de expensas, cuantos departamentos son los deudores.
3. Indique cuantos departamentos adeudan 5 expensas.
4. Genere un listado con los números de departamento que no registran deudas.
5. Muestre el importe total por deuda que debe cobrar la administración.

Ejercicio 16

Una librería tiene los siguientes datos de cada una de las 180 ventas realizadas: tipo de libro (1:Novela, 2:Documental, 3:Biografías,4:Escolares , ..., 20:Arte), precio y categoría ('A':Niños, 'B':Adolescentes, 'C':Jóvenes, 'D':Adultos).

Realizar un algoritmo que usando estructuras adecuadas y subprogramas óptimos permita:

1. Mostrar el total de libros vendidos en cada una de las cuatro categorías.
2. Mostrar para cada uno de los 20 tipos de libros el total de libros vendidos.
3. Indicar el importe total vendido, para un tipo de libro ingresado por teclado.
4. Escribir el porcentaje que representa la venta total de libros de tipo 1 y 4, respecto del total de libros vendidos.
5. Mostrar el nombre de la categoría con la máxima cantidad de libros vendidos

Ejercicio 17

Un representante de diarios y revistas registra los siguientes datos de sus 30 clientes: nombre, D.N.I., código de producto y cantidad.

Los productos están codificados de uno a cuatro (1..4) de la siguiente forma: 1:Diario Cuyo, 2:Diario Clarín, 3:Revistas Edit. Atlántida, 4:Revistas Edit. Perfil

Los clientes llevan siempre el mismo producto. Realizar un algoritmo que permita:

1. Realizar un listado ordenado alfabéticamente por nombre, donde se muestre nombre y DNI de los clientes.
2. Escribir el nombre y DNI del cliente que realizó la menor cantidad pedidos de Diario de Cuyo.
3. Escribir el porcentaje de revistas vendidas.
4. Escribir el total de pedidos realizados para cada código de producto.

Ejercicio 18

Una empresa tiene cinco sucursales provinciales codificadas de la siguiente forma: 1San Juan, 2-Mendoza, 3-San Luis, 4-La Rioja, 5-Catamarca.

En cada una de ellas comercializa cuatro rubros codificados como se señala a continuación: 1Comestible, 2-Electrónica, 3-Vestimenta, 4-Zapatería

Mensualmente se ingresa la siguiente información: código de sucursal, código de rubro e importe de cada una de las ventas realizadas diariamente.

Realizar un algoritmo que usando subprogramas permita:

1. Indicar importe vendido y cantidad de ventas realizadas para una sucursal ingresada por teclado.
2. Mostrar el importe total vendido entre todas las sucursales para un rubro ingresado previamente.

3. Indicar el importe promedio de ventas de cada una de las sucursales.
4. Generar una nueva estructura que contenga número de sucursal y cantidad de ventas realizadas.
5. Realizar las modificaciones que considere conveniente, para responder a todos los ítems anteriormente planteados, sabiendo que ahora la información viene ordenada por sucursal. Es decir por cada sucursal, se ingresa la siguiente información: código de rubro e importe de cada una de las ventas realizadas diariamente. Puede existir más de una venta por sucursal y por rubro.

Unidad 4: Análisis de Eficiencia de Algoritmos

Introducción

Cuando se resuelve un problema, en muchas oportunidades es posible seleccionar entre distintos algoritmos. ¿cómo realizar esta elección? En algunos casos y si se trata de un programa que se va a ejecutar pocas veces, es probable que el criterio de selección se centre en la facilidad para codificar, interpretar y depurar el algoritmo. Sin embargo, en la mayoría de las ocasiones, el programa se ejecutará varias veces y con entradas de gran tamaño, en este caso, el uso eficiente de los recursos y en especial el tiempo de ejecución del programa será el principal objetivo. Se seleccionará el algoritmo cuya ejecución sea más rápida aunque su código resulte más complejo.

En esta unidad se realizará el Análisis de Algoritmos con el fin de determinar su eficiencia. Este análisis permite evaluar las distintas soluciones de un mismo problema. Hasta el momento, para determinar cual algoritmo resuelve mejor una misma situación problemática, el análisis se ha limitado a comparar la cantidad de variables utilizadas y el número de comparaciones o de asignaciones efectuadas. Se realizará ahora un análisis teórico, comenzando con algoritmos muy sencillos para luego centrar la atención en algoritmos más complejos

En esta unidad se aspira a que el alumno sea capaz de:

- ✓ Adquirir capacidad para analizar teóricamente la eficiencia de los algoritmos.
- ✓ Calcular la complejidad temporal de algoritmos
- ✓ Comparar eficiencia de distintos algoritmos que resuelven la misma tarea.
- ✓ Calcular la complejidad temporal de algoritmos de búsqueda y ordenamiento

Para cumplir con estos objetivos se abordarán los temas que se muestran en el siguiente esquema:

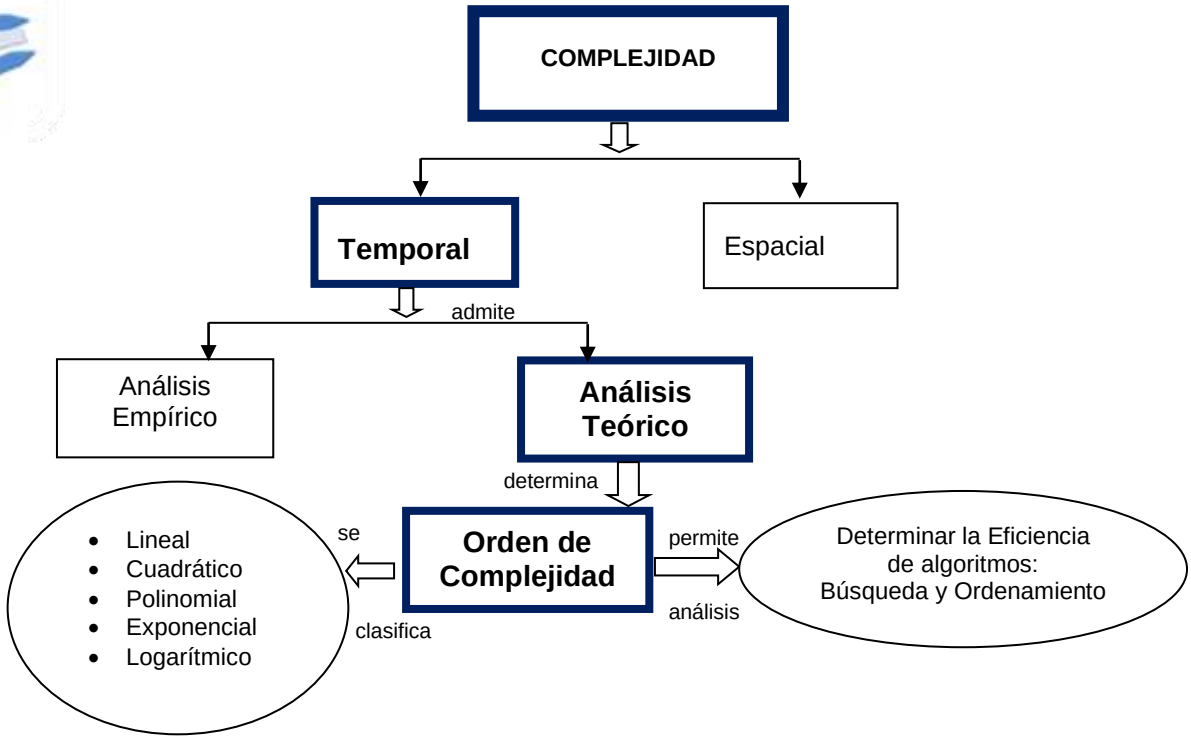


Figura 4.1. Esquema Temático de la Unidad

4.1 Eficiencia de un algoritmo

Actualmente, la velocidad de procesamiento de las computadoras ha crecido en forma considerable, esto hace pensar si es necesario "prestar atención" al estudio de la eficiencia de los algoritmos. Si bien existen problemas que son resueltos sin consumir demasiado almacenamiento y tiempo, otros dependen del tamaño de los datos considerados, pudiendo ocurrir que un pequeño aumento en su número ocasione un incremento considerable en el tiempo de ejecución.

Así por ejemplo, si para solucionar un problema se cuenta con un algoritmo que requiere de $10^{-4}.2^N$ segundos, para N elementos, si el número de elementos es tan solo 38, se necesitaría una computadora trabajando sin interrupción durante gran cantidad de tiempo para resolverlo.

Si es posible encontrar un algoritmo que en lugar de utilizar un tiempo exponencial lo haga en uno polinomial por ejemplo, se podría incluso resolver el problema para mayor cantidad de elementos en menor tiempo.

Se llama **complejidad de un algoritmo** a la medida de los recursos que necesita para su ejecución.

La Algorítmica, estudia técnicas para realizar algoritmos eficientes.

Un **algoritmo es más eficiente** que otro, cuando optimiza los recursos del sistema en el que se ejecuta en la resolución de un determinado problema.

Los principales recursos a los que hacemos referencia son **tiempo de ejecución y almacenamiento en memoria,** por ello se habla de complejidad espacial y complejidad temporal.

Complejidad espacial es una medida de la cantidad de espacio o almacenamiento ocupado por un programa.

Complejidad temporal es una medida de la cantidad de tiempo que requiere la ejecución de un programa.

Si bien esta unidad se centrará en el análisis teórico de la **Complejidad Temporal**, es importante considerar el **almacenamiento en memoria**, dado que serán más eficientes aquellos algoritmos que utilicen una estructura de datos adecuada que minimice el espacio de memoria utilizado.

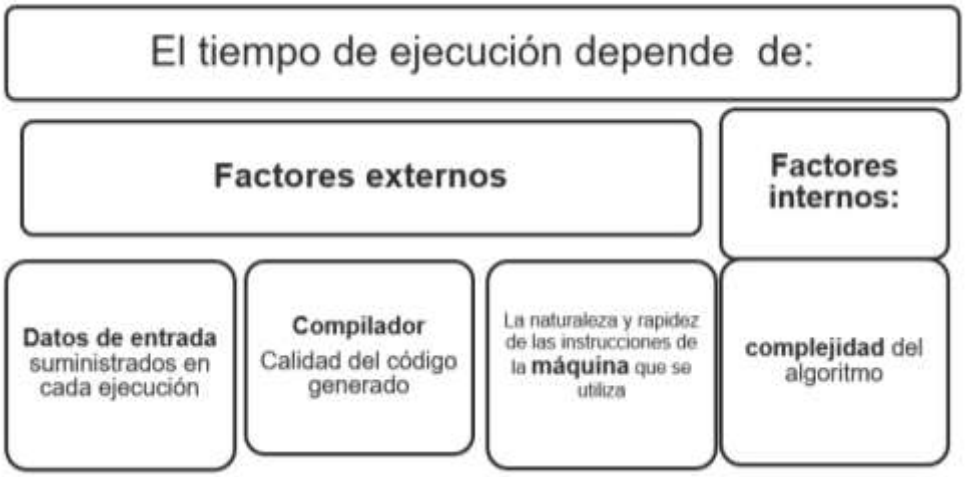
Desde el punto de vista del tiempo de ejecución, se considerarán más eficientes aquellos algoritmos que resuelvan el mismo problema en el menor tiempo posible.

El tiempo de ejecución de un programa depende de factores tales como:[3]

1. los datos de entrada al programa
2. la calidad del código generado por el compilador utilizado para crear el programa objeto
3- la naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa y
4- la complejidad de tiempo del algoritmo de base del programa

[3] Diseño y Análisis de Algoritmos Tiempo de Ejecución de un Programa. Aho Alfred Pág 16

Como se observa, algunos factores son propios del problema o factores internos (4) y otros dependen de la máquina utilizada o factores externos (1, 2 y 3).



4.2 Análisis de algoritmos según el tiempo de ejecución

Los siguientes ejemplos se refieren a problemas sencillos que pueden resolverse de distintas formas, lo que permitirá realizar un análisis somero de los aspectos a tener en cuenta para la optimización de los tiempos de ejecución.

Ejemplo 1:
Dados tres valores distintos, calcular el mayor.

Las siguientes son algunas posibilidades de solución de este problema:

Forma 1

```
Algoritmo Mayor1
entero mayor1(entero xm,xn,xp)
comienzo
  Si ((xm> xn) y (xm>x p))
     entonces  retorna(xm)
      Sino  Si (xn> xp)
              entonces retorna( xn)
              sino retorna(xp)
               FinSi
   FinSi
fin
```

Comienzo /* Algoritmo Principal */

```
 entero m, n, p, mayor
 Leer  m,n,p
 Escribir “el mayor número leído es “, mayor1(m,n,p)
```

Fin

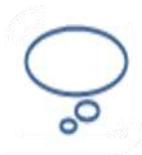
Forma 2

```
Algoritmo Mayor2
entero mayor2(entero xm,xn,xp)
comienzo
  Si (xm> xn)
        Entonces
          Si  (xm>xp)
               Entonces retorna( xm)
              Sino  retorna(xp)
            FinSi
        Sino  Si (xn>xp)
                 Entonces  retorna(xn)
                 Sino    retorna(xp)
              FinSi
  FinSi
Fin
```

Comienzo /* Algoritmo Principal */

```
  Entero m,n,p,mayor
  Leer  m,n,p
  Escribir " el mayor número leído es ", mayor2(m,n,p)
```

Fin



Forma 3

```
Algoritmo Mayor3
entero mayor3 (entero xm,xn,xp)
comienzo
  Si ((xm> xn) y  (xm>xp))
      Entonces retorna(xm)
  FinSi
  Si (( xn>xm) y (xn>xp))
       Entonces retorna(xn)
  FinSi
  Si ((xp>xm) y (xp>xn))
       Entonces retorna( xp)
  FinSi
Fin
```

Comienzo /* Algoritmo Principal */

```
  entero m,n,p,mayor
  Leer  m,n,p
  Escribir "el mayor número leído es ", mayor3(m.n.p)
```

Fin

Comparando las tres alternativas de solución, se puede concluir que en el caso de la **forma 1**, dependiendo de los valores ingresados, se puede realizar como mínimo 2 comparaciones (si m es el mayor) y como máximo 3. En cualquier caso se realiza un único retorno (que equivale a una asignación).

Para el caso de la **forma 2**, se realizan 2 comparaciones y un único retorno, independientemente del valor de las variables.
De estas 2 formas, la segunda arroja más eficiencia.

La **forma 3** es la menos óptima ya que se realizan entre 2 y 6 comparaciones dependiendo de los valores ingresados. Para el caso en que el valor m fuera el mayor, se harían 2 comparaciones sin embargo si p es el mayor se harían 4 comparaciones demás respecto del algoritmo más óptimo. Cuatro comparaciones demás para este ejemplo sencillo no es relevante. Pero ¿qué ocurre si el analista de una compañía de teléfonos de una ciudad utiliza este algoritmo para determinar, para cada uno de sus abonados, el consumo máximo de los 3 bimestres correspondientes al primer semestre del año?. Utilizar en este caso la tercera forma de este algoritmo, implicaría un derroche innecesario de tiempo.

Ejemplo 2: Otro aspecto a tener en cuenta para reducir tiempos de ejecución, consiste en no repetir cálculos, sobre todo si ellos forman parte de un ciclo, como se muestra en el ejemplo.

*Situación problemática***:** Una empresa desea realizar el control de calidad de los envases de conservas que en ella se fabrican. Se sabe que diariamente se producen 20.000 envases de forma cilíndrica y que las especificaciones son las siguientes: la superficie de la tapa debe estar comprendida en el rango 50± 0.05 cm^2 y el volumen del recipiente entre 600± 0.5 cm^3 respectivamente. Se necesita especificar las tapas o envases cilíndricos que deben ser eliminados por no cumplir los requisitos establecidos. Además se debe informar a gerencia, la cantidad de piezas defectuosas de cada tipo (tapas y recipientes) y el porcentaje que esas cantidades representan sobre el total producido. Los datos que se conocen para cada envase fabricado son el diámetro de la tapa y la altura del recipiente cilíndrico.

El algoritmo siguiente muestra una forma de resolución del problema.

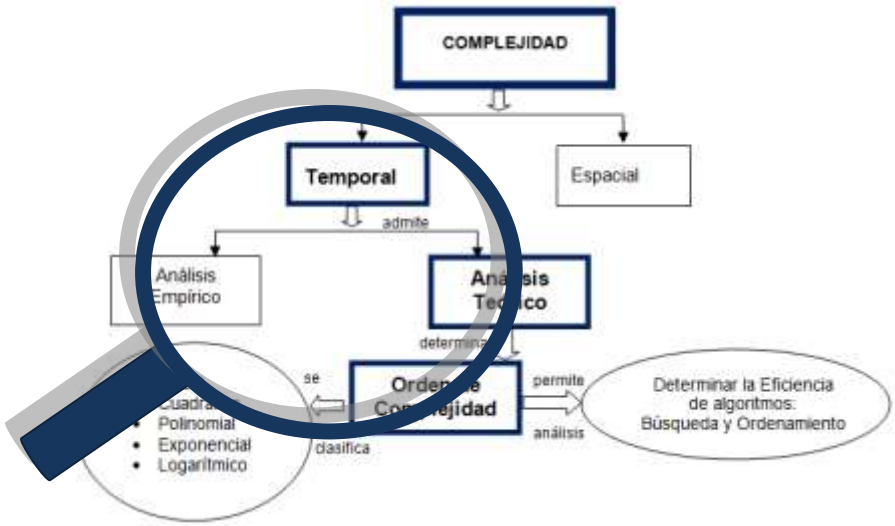
```
Algoritmo Envases
constante pi= 3.14159

real superficie (real xdiametro)
comienzo
   superficie = pi* (xdiametro/2 ) *(xdiametro/2)
fin

real volumen( real xdiametro, real xaltura)
comienzo
    volumen= pi* (xdiametro/2 ) *(xdiametro/2) *xaltura
fin


Comienzo /* Algoritmo Principal */
  entero Ct, Cc, i
  real diametro, altura, sup, Vol

  Ct=0
  Cc=0
```

```
  Para i Desde 1 Hasta 20000
     Leer diametro, altura
     sup= superficie (diametro)
     Si  ((( sup- 50)> 0.05 ) o (( sup- 50) < -0.05)))
        Entonces  Escribir " Tapa defectuosa"
                  Ct= Ct+1
     FinSi
     Vol= volumen(diametro,altura)
     Si ( ( Vol- 600)> 0.5) o ( ( Vol- 600)<- 0.5))
        Entonces  Escribir " Recipiente defectuoso"
                  Cc= Cc+1
     FinSi
  FinPara
  Escribir " Detectadas ", Ct ,"  tapas defectuosas     Porcentaje ", Ct*100/ 20000
  Escribir "Detectados ", Cc, " cilindros defectuosos  Porcentaje ", Cc*100/ 20000
Fin
```

Este algoritmo optimizará su tiempo de ejecución si las operaciones resaltadas son reemplazadas como se muestra a continuación.

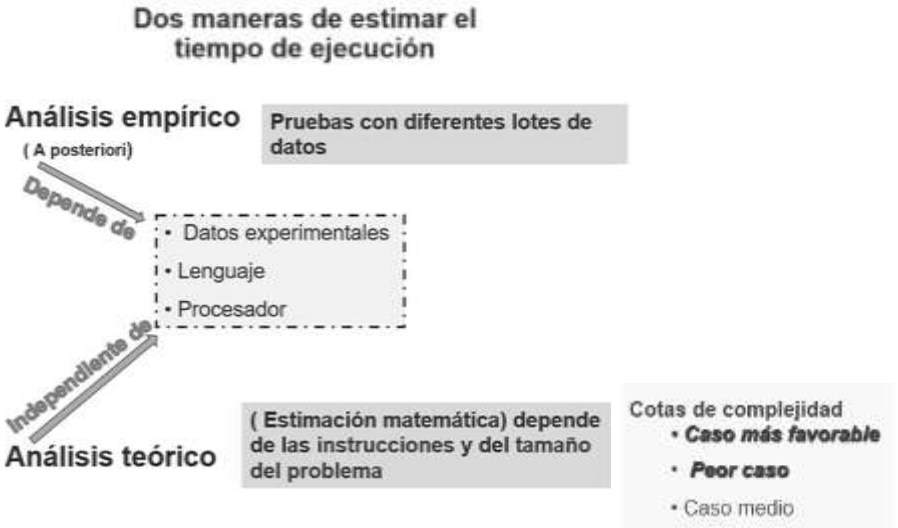
```
Algoritmo EnvasesOptimo
Pi= 3.14159

real superficie (real xradio)
comienzo
   superficie = pi* (xradio/2 ) *(xradio/2)
fin

real volumen( real xradio, real xaltura)
comienzo
   volumen= pi* (xradio ) *(xradio) *xaltura
fin

Comienzo /* Algoritmo Principal */
entero Ct, Cc, i
real  diametro, altura, sup, Vol, radio, dif
Ct=0
Cc=0
Para i Desde 1 Hasta 20000
    Leer diametro, altura
    radio = diametro/2
    sup = superficie( radio)
    dif = sup-50
   Si  (( dif)> 0.05 ) o (( dif) < -0.05)
        Entonces  Escribir " Tapa a eliminar"
                  Ct= Ct+1
    FinSi
```

```
    Vol = volumen( radio,altura)
    dif =Vol-600

    Si ( ( dif)> 0.5) o ( ( dif)<- 0.5))
        Entonces  Escribir " Cilindro a eliminar"
                  Cc= Cc+1
    FinSi
FinPara
Escribir " se han detectado ", Ct, "  tapas defectuosas  Porcentaje ", Ct/ 200
Escribir " se han detectado ", Cc, "  cilindro defectuosos Porcentaje ", Cc/ 200
Fin
```



Tiempo de ejecución de un Algoritmo: Análisis empírico y Análisis teórico

Existen **dos maneras de estimar el tiempo de ejecución** de un algoritmo, mediante un **análisis empírico o uno teórico**.



El **análisis empírico** o a posteriori, consiste en la comparación de los tiempos de ejecución de distintos algoritmos que resuelven un problema, realizando pruebas con diferentes lotes de datos.

Los algoritmos deberán codificarse en un lenguaje de programación y así se podrán calcular los tiempos, mediante la utilización de acciones que soliciten al sistema los momentos de comienzo y finalización de ejecución. Se obtiene de esta manera una medida real del tiempo en el entorno de aplicación.

En general los problemas del análisis empírico se pueden resumir en la dependencia de los resultados obtenidos del tipo de computadora, del lenguaje de programación usado, del traductor con el que se obtenga el código ejecutable, y de la habilidad del programador. Es decir que depende de los factores externos e internos.
Si se modifica alguno de estos elementos probablemente se obtengan resultados diferentes, con lo cual no se puede establecer una eficiencia empírica absoluta.
Por otra parte, existen algoritmos que pueden ser comparados con esta técnica sólo cuando el número de datos con los que se ejecutan es relativamente pequeño.

El **análisis teórico**, en cambio, favorece la independencia con todos los factores anteriores. Es válido para cualquier entrada, depende sólo de las instrucciones que componen el algoritmo y del tamaño del problema; es decir, depende solamente de factores internos.

Un análisis teórico permitirá estimar matemáticamente el orden del tiempo de respuesta, independientemente de los datos experimentales.

Del análisis teórico se obtiene una expresión matemática que indica cómo se produce el crecimiento del tiempo de ejecución a medida que aumenta el tamaño del problema. No necesita la implementación y ejecución del algoritmo.

Se denomina **tamaño de un problema** al número de datos del problema. Por ejemplo, si se desea ordenar un arreglo de N elementos, el tamaño del problema es N.

Se llama **instancia de un problema** a un caso particular del problema.

Debe tenerse en cuenta que, para un mismo tamaño, el algoritmo puede comportarse de distintas maneras. Así por ejemplo si se desea buscar un elemento en un arreglo, el tamaño del problema es la cantidad de elementos que tiene el arreglo. Sin embargo, el comportamiento de una búsqueda secuencial será distinto si el elemento se encuentra en la primera posición (el algoritmo finaliza enseguida) o si se encuentra en la última posición (el algoritmo debe recorrer todo el arreglo).

Por esto, cuando se analiza la eficiencia de un algoritmo, se podrían **estudiar tres situaciones importantes:**

- el mejor caso o cota inferior
- el peor caso o cota superior
- el caso medio o cota promedio.

Por ejemplo, si el algoritmo consiste en ordenar ascendentemente un vector, el mejor caso es cuando el arreglo está ordenado de forma creciente, el peor caso cuando esté ordenado de manera decreciente y el caso medio es cuando sus componentes están organizadas en forma aleatoria. En todos los casos el tamaño del problema es el mismo.

Aunque el análisis del término medio parece ser el más adecuado, puede ocurrir que algunos datos de entrada tengan mayor probabilidad que otros. En estos casos la media no es el promedio del peor y mejor caso, sino que debe ser ponderada, lo que implica realizar cálculos de probabilidades, situación que queda fuera del alcance de este libro.

El estudio del peor caso, es útil para problemas cuyos tiempos de respuesta son críticos, como por ejemplo los sistemas de tiempo real utilizados en medicina, home-banking, comercio electrónico, y sistemas de seguridad controlados por software.

El enfoque teórico ayudará a establecer la frontera entre lo factible y lo imposible y tiene además la ventaja de no depender de la computadora ni del lenguaje de programación utilizado.
En este libro se utilizarán técnicas elementales que permitirán determinar la eficiencia de algoritmos referidos a búsquedas y ordenamiento, para el peor de los casos. Si bien existen algoritmos más eficientes, para resolver algunos de los ejemplos que se presentan, requieren de técnicas avanzadas, cuyo abordaje está sustentado sobre una base teórica que está fuera del alcance de este libro, destinado a cursos de inicio de programación.

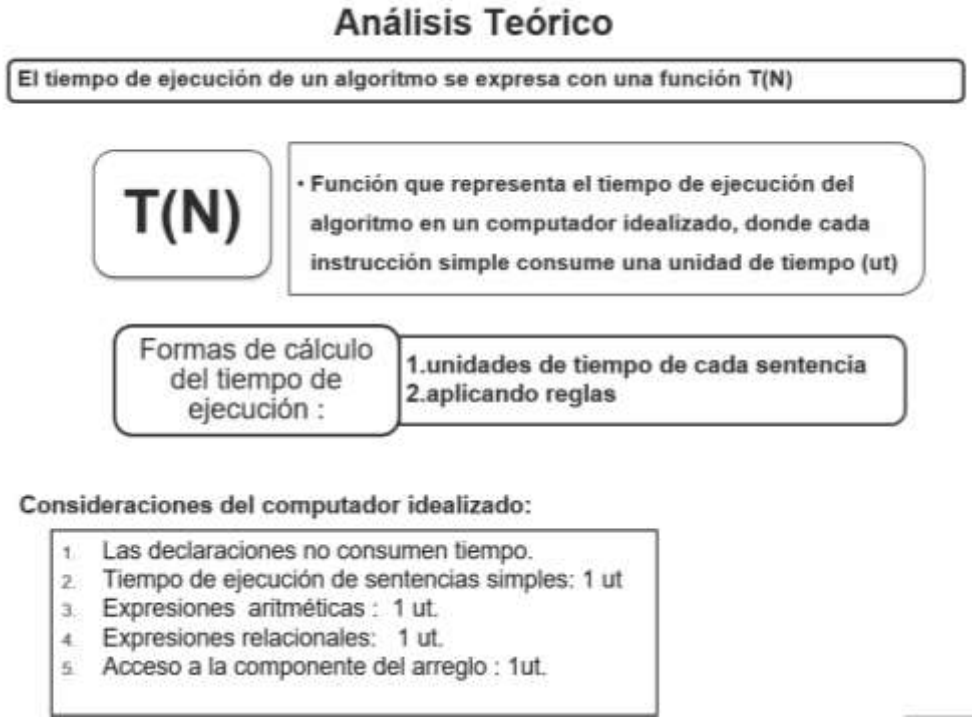
Orden de Complejidad de un algoritmo - Eficiencia asintótica

Un análisis teórico determina el orden del tiempo de respuesta u ***orden de complejidad***, que es utilizado para la comparación de diferentes soluciones algorítmicas sin depender de datos experimentales.

Para poder obtener el Orden de complejidad de un algortimo, es necesario en primer lugar calcular el Tiempo de ejecución del mismo, representado por una función T(N).

T(N) representa las unidades de tiempo (segundos, milisegundos,...) que un algoritmo tardaría en ejecutarse con unos datos de entrada de tamaño N.

T(N).) corresponde al tiempo de ejecución del algoritmo en un ordenador idealizado, donde cada una de las instrucciones simples consume una unidad de tiempo.



Al estudio de la complejidad para tamaños grandes de problema se lo conoce como **eficiencia asintótica** del algoritmo. Este análisis clasifica las funciones T(N) en clases constituidas por funciones que crecen de la misma forma. De esta forma pueden compararse fácilmente entre sí las funciones correspondientes a los algoritmos que resuelven un mismo problema.

La notación matemática que se utiliza para representar el orden de complejidad de un algoritmo cuando el tamaño del problema es grande (notación asintótica) es una **O** mayúscula, **O**(f(N).)) , conocida también como **o grande** para representar la cota superior y **Ω** (omega) para representar la cota inferior .

Eficiencia asintótica para el peor caso. Notación O

Si el tiempo de ejecución de un algoritmo se expresa con T, se dice que *T es de orden f(*N) y se simboliza T **ϵ** O(f(N)), si existe una función matemática f(N) que lo acota.

Dicho de otro modo:

T **ϵ** O(f(N).)) si existen constantes c y N_0 / T≤c.f(N).) para N≥ N).$_0$

Al decir que T **ϵ** O(f(N)) se está garantizando que la función T no crece más rápido que f(N), esto es que f(N) es un límite superior para T- La notación O-grande hace referencia al peor caso.

Cuando el tiempo de ejecución de un programa es O(f(N)), se dice que tiene velocidad de crecimiento f(N).

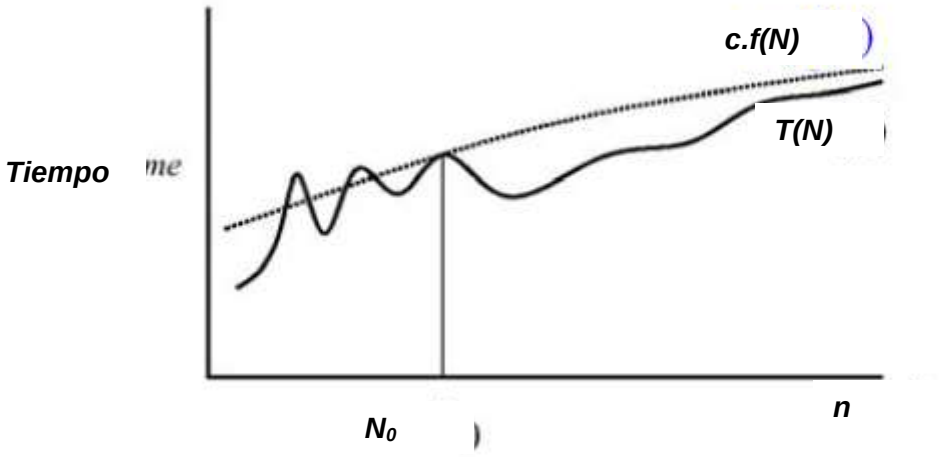
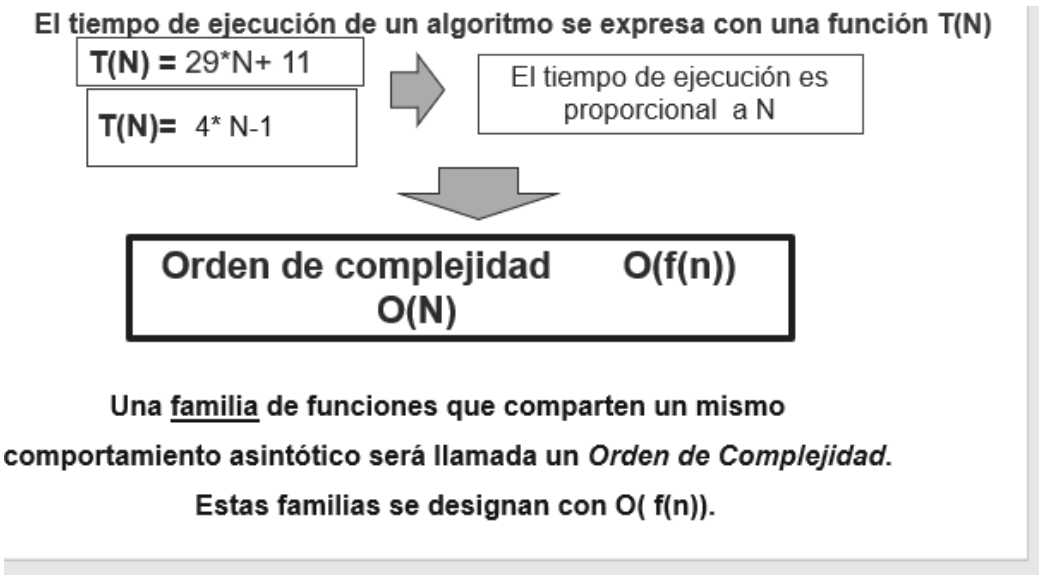


Figura 4.2 Cota superior



Ejemplo 3

- $n+5$ **ϵ O(n)** pues $n+5 \leq 2n$ para toda $\boldsymbol{n} \geq 5$
- $3n + 2$ **ϵ O(n),** pues $3n+2 \leq 4n$ para toda $\boldsymbol{n} \geq 2$
- $\boldsymbol{(n+1)^2}$ **ϵ O($\boldsymbol{n^2}$)** pues $\boldsymbol{(n+1)^2 \leq 4n^2}$ para $\boldsymbol{n} \geq 1$.
- $1000n^2 + 100n - 6$ **ϵ O($\boldsymbol{n^2}$)**, porque $1000n^2 + 100n - 6 \leq 1.001n^2$ para $n \geq 100$
- $\boldsymbol{(n+1)^2}$ **NO ϵ O(n)** pues para cualquier $\boldsymbol{c > 1}$ no se cumple que $\boldsymbol{(n+1)^2 \leq c*n}$

Entones, en general:

si $T(n) = (n + 5)^2$ se dice que **ϵ** $O(n^2)$

si $T(n) = 4n + 6n^3$ se dice que **ϵ** $O(n^3)$

Los órdenes de complejidad permiten comparar la eficiencia de algoritmos, por lo cual:

Si se determina que los tiempos para dos algoritmos que resuelven un determinado problema son $T_1 = O(3 * N)$ y $T_2 = O(2 * N^2)$ para $N>1$, se puede inferir que el primer algoritmo es más eficiente que el segundo.

Un aspecto importante a considerar al comparar los tiempos de ejecución de algoritmos es la velocidad de crecimiento de las funciones. Así por ejemplo si se deben comparar las funciones:

$T_1 = O(500 * N)$ y $T_2 = O(N^2)$

Para N pequeño el algoritmo 2 es más eficiente, pero se debe analizar que ocurre para valores de N mayores que 500.

Hay ciertos órdenes de complejidad que se producen con tanta frecuencia que se les ha dado nombre. Si el tiempo en que un algoritmo resuelve un caso de tamaño N, nunca supera a c*N segundos, siendo c una constante, se dice que el algoritmo es de orden N o que requiere un *tiempo lineal* de procesamiento.

Si el tiempo no supera a $c*N^2$ segundos, se dice que el algoritmo es de orden N^2 o que requiere un *tiempo cuadrático* de procesamiento.

Si el tiempo no supera a c*N*logN segundos, se dice que el algoritmo es de orden N* logN o que requiere un *tiempo Cuasilineal* de procesamiento

Un orden **O(N)** se llama complejidad lineal, indica que el tiempo de ejecución crece en proporción directa al crecimiento de **N**.
Un algoritmo de orden **O(1**) tiene complejidad constante, suelen ser los más eficientes y preferidos.
La mayoría de los programas tienen complejidad polinomial, **O(N^a)** donde **N** es la variable y **a** es una constante mayor o igual que 1. Son ejemplos de este tipo de complejidad, la lineal O(N), la cuadrática $O(N^2)$, y la complejidad cúbica $O(N^3)$
Para un tiempo de ejecución cuadrático, si N se duplica, el tiempo de ejecución aumenta cuatro veces.
Para un tiempo de ejecución cúbico, si N se duplica, el tiempo de ejecución se multiplica por ocho.
Existen algoritmos que tienen complejidad logarítmica **O(log N)**.
Generalmente, el tiempo de ejecución de un algoritmo es proporcional a alguno de los tiempos descriptos o una combinación de ellos.
Así, el tiempo de ejecución de un algoritmo puede ser $T = 4N^2 + 2N$. En este caso se dice que el tiempo de ejecución es proporcional a N^2 o de orden cuadrático, lo cual se simboliza $O(N^2)$

Una **FAMILIA de funciones** que comparten un mismo **comportamiento asintótico** será llamada un ***Orden de Complejidad***.

- **O(1):** Complejidad constante.
- **O(*log n*):** Complejidad logarítmica.
- **O(*n*):** Complejidad lineal.
- **O(*n log n*):** Complejidad cuasi-lineal.
- **O(n^2):** Complejidad cuadrática.
- **O(n^3):** Complejidad cúbica.
- **O(n^a)** Complejidad polinómica ($a > 3$).
- **O(2^n):** Complejidad exponencial.

Tabla 4.2 Orden de Eficiencia

La siguiente tabla[4] muestra una comparación entre diferentes complejidades.

Nº elementos	**N**	**lg n**	**n lg n**	**n^2**	**n^3**	**2^n**	**3^n**	**n!**
2	2	1	2	4	8	4	9	2
4	4	2	8	16	64	16	81	24
8	8	3	24	64	512	256	6.561	40.320
16	16	4	64	256	4.096	65.536	43.046.721	20.922.789.888.000
32	32	5	160	1.024	32.768	4.294.967.296	1.853.020.188.851.841	¿ ?
64	64	6	384	4.096	262.144	¿?	¿?	¿ ?
128	128	7	896	16.384	2.097.152	¿?	¿?	¿ ?

Tabla 4.1 Comparación de órdenes de eficiencia

NOTA: Con el símbolo **¿?** (que aparece en la tabla) se quiere reflejar tiempos extremadamente grandes.
De esta tabla se puede inferir que los algoritmos polinomiales, aquellos que son proporcionales a N^k, es decir de la forma $c*N^k$ siendo c una constante, obtienen sus resultados en un tiempo limitado. En cambio en el caso de los algoritmos exponenciales, aquellos que son proporcionales a k^N, los resultados se obtienen en tiempos desmedidamente grandes y en general se tornan poco probables de utilizar, salvo para tamaños de entrada de datos muy reducidos.

[4] Extraído y adaptado de Análisis de algoritmos *http://www.ub.edu.ar/catedras/ingenieria/sist_datos/* Fuente original D. Riley en [Ril-87].

La comparación gráfica de las funciones de complejidad más frecuentes se muestra en la figura 4.2

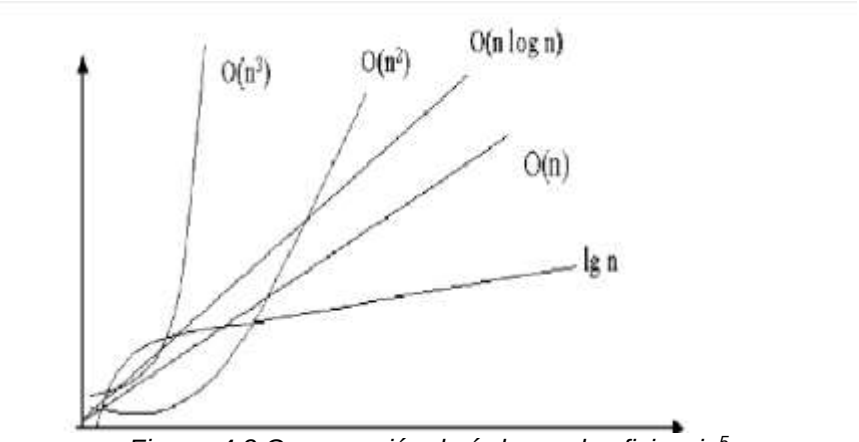


Figura 4.3 Comparación de órdenes de eficiencia[5]

El mejor orden es el logarítmico, si se duplica el tamaño del problema, no afecta al tiempo.

Para los órdenes *cuasilineal* (N.logN) y lineal, si se duplica el tamaño del problema se duplica, aproximadamente, el tiempo empleado.
Los problemas de orden polinomial necesitan mucho tiempo para resolver un problema que ha crecido relativamente poco en tamaño, sin embargo, se consideran tratables.

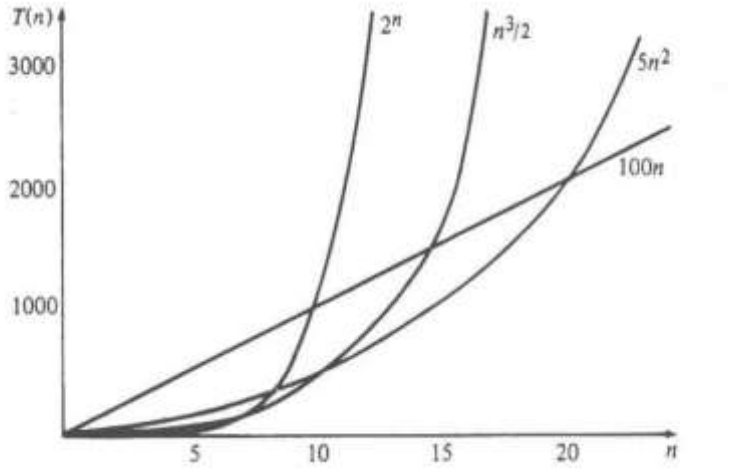


Figura 4.4 Tiempo de ejecución de cuatro programas

[5] Extraída de Diseño y Análisis de Algoritmos. Cap 2 La eficiencia de los algoritmos. Dpto Sistemas Informáticos. Universidad de Alicante

Tiempo de Ejecucion T(n)	Tamaño máximo de problema para 10^3 seg	Tamaño máximo de problema para 10^4 seg	Incremento en el tamaño máximo de problema
100n	10	100	10.0
$5n^2$	14	45	3.2
$N^3/2$	12	27	2.3
2^n	10	13	1.3

Tabla 4.3 Efecto de multiplicar por diez la velocidad del computador

En la figura 4.3 y tabla 4.3 [6], se observan los tiempos de ejecución de cuatro algoritmos de distintas complejidades de tiempo, medidas en segundos. Si se dispone de 1000 segundos (aproximadamente 17 minutos) para resolver un determinado problema, se infiere que los tres algoritmos pueden resolver problemas de un tamaño parecido (desde N=10 a N=12).

Suponga que ahora cuenta con una computadora que funciona 10 veces más rápido, entonces con el mismo costo se puede dedicar para resolver el mismo problema 10.000 seg. Como se observa ahora el tamaño del problema a resolver varía significativamente entre los algoritmos. Se advierte además que los algoritmos de orden exponencial sólo pueden resolver problemas pequeños, independientemente de la rapidez de la computadora.

En la tercera columna de la Tabla 4.3, se observa la superioridad del algoritmo con eficiencia lineal O(n), que permite un aumento de 1000% en el tamaño del problema para un incremento del 1000% en la velocidad de la computadora.

Los programas con eficiencia $O(n^3)$ y $O(n^2)$ permiten aumentos del 230% y 320% en el tamaño del problema para el mismo incremento de velocidad.

Relación entre los Órdenes de complejidad

$O(1) < O(\log(n)) < O(n) < O(n.\log(n)) < O(\log n) < O(n^2) < O(n^3) < O(2^n)$

Mientras exista la necesidad de resolver problemas cada vez más grandes, se producirá una situación casi paradójica. A medida que los computadores aumenten su rapidez y disminuyan su precio, como con toda seguridad seguirá sucediendo, también el deseo de resolver problemas más grandes y complejos seguirá creciendo. Así, la importancia del descubrimiento y empleo de algoritmos eficientes, aquellos cuya velocidad de crecimiento sean pequeñas, irá en aumento en lugar de disminuir" [7]

[6] Extraído de Estructura de datos y algoritmos Aho Alfred Pág 20

[7] Aho Alfred .Estructura de datos y algoritmos Pág. 20

Análisis teórico del tiempo de ejecución de un algoritmo

Existen dos formas de calcular el tiempo de ejecución de un algoritmo:

a) mediante el cálculo de unidades de tiempo de cada una de las sentencias que conforman un algoritmo según determinadas reglas.
b) mediante la aplicación de reglas del orden de complejidad, que optimizan el cálculo.

Formas de cálculo del tiempo de ejecución :	**1.unidades de tiempo de cada sentencia** **2.aplicando reglas**

Consideraciones del computador idealizado:

1. Las declaraciones no consumen tiempo.
2. Tiempo de ejecución de sentencias simples: 1 ut
3. Expresiones aritméticas : 1 ut.
4. Expresiones relacionales: 1 ut.
5. Acceso a la componente del arreglo : 1ut.

Cuando la definición del tiempo de ejecución de un algoritmo se realiza **en función de la cantidad de unidades de tiempo** necesarias para resolver un determinado problema, es necesario tener en cuenta las siguientes reglas

Reglas para calcular el tiempo de ejecución de un algortimo

- *Las declaraciones* no consumen tiempo.
- *Tiempo de ejecución de Sentencias simples*: cualquier Acción simple, de asignación, lectura o escritura consume una unidad de tiempo, éstas se conocen como operaciones elementales.
- *Las operaciones aritméticas* requieren una unidad de tiempo para su ejecución.
- El *acceso a una componente* de un arreglo involucra una unidad de tiempo.

De lo expuesto se infiere que:

tiempo de asignación a una variable de una expresión =
tiempo de evaluación de la expresión + tiempo de asignación

- *Bucles incondicionales*:

Si la cantidad de iteraciones es fija: el tiempo de ejecución se obtiene como el producto del tiempo de ejecución de las sentencias que están dentro del bloque por la cantidad de iteraciones que se debe realizar. A este tiempo debe agregarse el de inicialización de variables, testeos e incrementos de variable de control.

Tiempo ejecución bucle = tiempo del cuerpo* número de iteraciones + tiempo de evaluar la condición

- *Bucles condicionales*: Si la cantidad de iteraciones varía en función del valor de la variable de control, el cálculo del tiempo se expresa como una sumatoria.

- *Ciclos incondicionales anidados*: Se resuelven de adentro hacia afuera. Es decir, el tamaño de cada bloque se va multiplicando por la cantidad de iteraciones de cada ciclo. A este tiempo debe agregarse el de inicialización de variables y testeos de cada uno de los ciclos. En este caso la complejidad es polinómica.

- *Acciones alternativas : Selección doble*

```
Si (condición)
    Entonces  S1
    Sino      S2
Finsi
```

Si llamamos t_1 al tiempo de ejecución de la sentencia o bloque S1 y t_2 al tiempo de ejecución de la sentencia o bloque S2, el tiempo de ejecución máximo se obtiene sumando al tiempo de testeo de la condición, el tiempo máximo entre t_1 y t_2

T= tiempo de condición + máximo (t_1, t_2).

- *Alternativa Múltiple*

Para el caso de la alternativa múltiple, se adiciona al tiempo de evaluación de la condición el valor correspondiente al tiempo de ejecución máximo de las distintas alternativas.

Como se deduce de estas reglas, el análisis de los algoritmos se realiza desde adentro hacia afuera. Esto es, primero se determina el tiempo requerido para las instrucciones individuales y luego el que corresponde a la estructura de control que contiene a dichas instrucciones.

Con estas consideraciones, analicemos el tiempo de ejecución del siguiente segmento de código :

```
        entero j, min
1.      min= 0 ⟶ 1ut
2.      Para j desde 1 hasta n-1 ⟶ 1+ n + n-1 = 2n
3.          si (vector[j] < vector[min]) ⟶ 3ut
4.              entonces min= j ⟶ 1ut        } 4(n-1)
5.      Finpara
```

Teniendo en cuenta estas consideraciones, la estimación del tiempo de ejecución correspondiente al siguiente algoritmo es: T(N)= 1+2n+4 n-4= 6n-3

```
Algoritmo Envases1
Comienzo
entero Ct, Cc
real Pi, diametro, altura, sup, Vol
1   Pi= 3.14159
2   Ct=0
3   Cc=0
4   Para i Desde 1 Hasta 20000
5       Leer diametro, altura
6       radio= diametro/2
7       radioc=radio*radio
8       sup = pi* radioc
9       dif= sup-25
10      Si  (( dif)> 0.05 ) o (( dif) < -0.05))
11          Entonces  Escribir " Tapa a eliminar"
12                    Ct= Ct+1
13      Finsi
14      Vol= pi* radioc*altura
15      dif=Vol-50
16      Si (( dif > 0.5) o ( dif<- 0.5))
17          Entonces  Escribir " Cilindro a eliminar"
18                    Cc=Cc+1
19      FinSi
20. FinPara
21  Escribir " Detectadas ", Ct, "  tapas defectuosas      Porcentaje ", Ct/ 200
22  Escribir " Detectados ", Cc, "cilindros defectuosos    Porcentaje ", Cc/ 200
Fin
```

Las declaraciones no consumen tiempo.
Las líneas 1 a 3 consumen una unidad de tiempo (ut) cada una.
La línea 4 correspondiente al ciclo Para, utiliza 1 unidad para la inicialización, N+1 unidades para el testeo (N es la cantidad de datos y por tanto el número de veces que se realiza el ciclo) y N unidades para el incremento de la variable del ciclo. Esto hace un total de 2*N+2 unidades de tiempo.

Analicemos ahora las acciones que están dentro del ciclo:
La línea 5, lectura de 2 valores, utiliza 2 ut.
En las líneas 6, 7, 8 y 9 se realizan 1 operación y 1 asignación, 2ut cada una, siendo 8ut en total.
La líneas 10 a 13 correspondiente a la estructura de selección doble, requieren 6 ut.
La línea 14 realiza 2 operaciones aritméticas y 1 asignación, ocupa 3 ut .
En la línea 15 se efectúa 1 operación aritmética y 1 asignación, ocupa 2 ut.
Las líneas 16 a 19 correspondientes a la otra estructura alternativa también requieren 6 ut.
Por ello el cuerpo del ciclo requiere 27 unidades de tiempo que deberán multiplicarse por las veces en que el ciclo debe repetirse, esto es por N.
Finalmente para las Acciones 21y 22 que realizan 2 operaciones de escritura y 1 operación aritmética cada una, se necesitan en total 6ut

Resumiendo:

Acciones		Unidades de tiempo
Simples : líneas 1 a 3		3
Ciclo		
Línea 4		2*N+2
Cuerpo del ciclo:		(27 * N)
	Líneas 5, 6, 7, 8, 9	10
	Alternativa 10 a 13	6
	Línea 14	3
	Línea 15	2
	Alternativa 16 a 19	6
Simples : líneas 21 y 22		6

Tabla 4.3 Cálculo de unidades de tiempo

Teniendo en cuenta que el ciclo se repite N veces, el total de unidades de tiempo para este algoritmo es

$$3 + (2* N + 2) + (27 * N) + 6 = 29* N + 11.$$

Entonces se puede decir que el tiempo de ejecución de este algoritmo es de orden O(N)

Como se observa, aplicar este tipo de análisis a cada segmento de un algoritmo resulta una tarea compleja y poco práctica. Las siguientes son algunas consideraciones que pueden tenerse en cuenta, para reducir este tedioso trabajo y obtener la misma respuesta.

Reglas para determinar el orden de complejidad

Con el fin de simplificar la tarea de calcular el orden de complejidad de los algoritmos se establecerán reglas para su determinación.

Supongamos que T1(N) y T2(N) son los tiempos de ejecución de dos segmentos o bloques de programa P1 y P2 y que T1(N) ϵ O(f(N)) y T2(n) ϵO(h(N)).

Las siguientes reglas, resultan eficaces para calcular el tiempo de ejecución de un algoritmo:

1- Regla de la Suma u Orden correspondiente a bloques consecutivos con tiempo de ejecución de distinto orden

El tiempo de ejecución de P1 seguido de P2, es decir T1(N) + T2(N), es O(max(f(N), h(N))), esto es el tiempo de ejecución total tiene un orden que coincide con el mayor de los anteriores.

Simbólicamente:

Si $T_1 \epsilon O(f(N))$ y $T_2 \epsilon O(h(N))$, entonces

$$T_1 + T_2 = c * \text{máximo}(O(f(N)), O(h(N)))$$

Justificación: Esta afirmación se puede probar como sigue:

Si $T_1 \epsilon O(f(N))$ y $T_2 \epsilon O(h(N))$ entonces existen las constantes c_1, c_2, N_1 y N_2 tales que

$T_1 \leq c_1 f(N)$ para $N \geq N_1$

$T_2 \leq c_2 h(N)$ para $N \geq N_2$

Si $N_0 = \text{máximo}(N_1, N_2)$ entonces $T_1 \leq c_1 f(N)$ para $N \geq N_0$

$T_2 \leq c_2 h(N)$ para $N \geq N_0$

$\Rightarrow \quad T_1 + T_2 \leq c_1 f(N) + c_2 h(N)$ para $N \geq N_0$

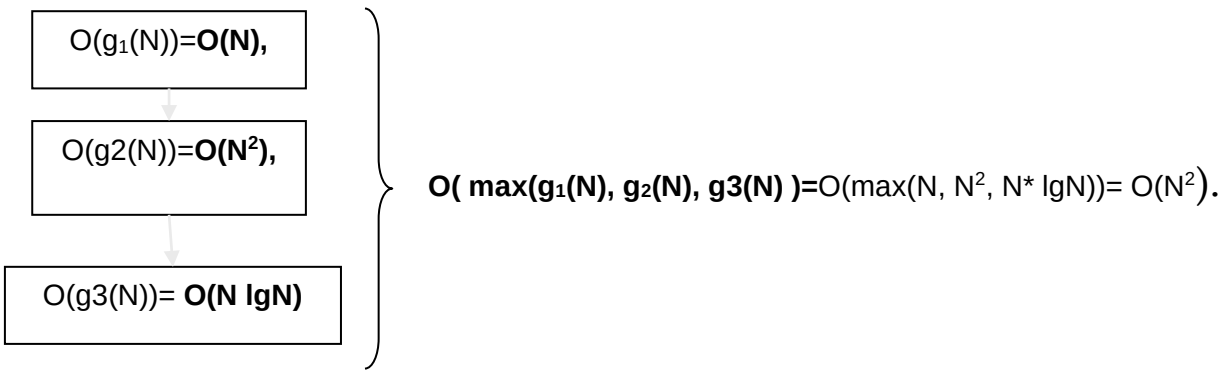
Sea $c_0 = \text{máximo}(c_1, c_2) \Rightarrow T_1 + T_2 \leq c_0 f(N) + c_0 h(N)$ para $N \geq N_0$

$T_1 + T_2 \leq c_0 f(N) + c_0 h(N)) \leq 2c_0$ máximo$(f(N), h(N))$ para $N \geq N_0$ y $c_0 = \text{máximo}(c_1, c_2)$

Llamando c a $2c_0$, se tiene:

$T_1 + T_2 \leq c$ máximo$(f(N), h(N))$ para $N \geq N_0$ y $c = 2c_0$

Aplicabilidad: La regla de la suma puede usarse para calcular el tiempo de ejecución de un algoritmo constituido por bloques. Entonces, si el algoritmo está constituido por un bloque de orden O(N), otro de orden O (N^2) y un tercero de orden O(N*logN), el tiempo completo de ejecución es de orden O(N^2). El resultado está sobreestimado, por ello se habla de tiempo máximo.



2 - Regla del producto O(f * h) = O(f) * O(h)

El orden de complejidad del producto de dos funciones es igual al producto de los órdenes de complejidad de cada una de ellas.

Justificación

Si $T_1 \epsilon O(f(N))$ y $T_2 \epsilon O(h(N))$ entonces existen las constantes c_1, c_2, N_1 y N_2 tales que

$T_1 \leq c_1 f(N)$ para $N \geq N_1$

$T_2 \leq c_2 h(N)$ para $N \geq N_2$

Entonces $T_1 * T_2 \leq c_1 f(N)) * c_2 h(N)) = c_1 * c_2 f(N) * h(N))$

Entonces $T_1 * T_2 \epsilon O(f(N) * h(N))$ para $N >= N_0$ y N_0= máximo (N_1, N_2)

Ejemplo 4: $O((20 * N) * N) = O(20 * N) * O(N) = O(N^2)$

Aplicabilidad: Si dos trozos de código anidados (no independientes), tienen eficiencias O(f(N)) y O(h(N)), la eficiencia del trozo completo es O(f(N)*h(N)).

Ejemplo 5: dados los siguientes ciclos anidados

O(N)

O(lg N)

El orden del código completo es O(N*lgN).

3 - O(C*g) = O(g)

La complejidad de una función por una constante es una O de la función.

Esto es, O (2500*N), es una O (N) , siendo C= 2500

4 - Los algoritmos sin lazos y sin recursión tienen complejidad constante. Se dice que son O(1) ya que representan una cantidad constante de tiempo.

5 - Los lazos anidados tienen complejidad polinómica.

Estas propiedades nos permiten inferir el orden de complejidad de las estructuras de control, como se muestra en la siguiente tabla:

Estructura	Orden
Secuencial (S)	**O(1):**Si el bloque está constituido de Acciones simples de orden O(1), entonces el bloque completo será de orden O(1) por aplicación de la regla de la suma.
Bloques consecutivos de distinto orden S1, S2	La **función mayor** entre **O(S1) y O(S2)**
Si (Condición) entonces S1 sino S2 Finsi	La **función mayor** entre **O(S1), O(S2)** (no se considera el orden de la condición ya que es contante)
Ciclo con un índice que se ejecuta n veces[S1] i = 1, n	**O(n)**
Ciclo con dos índices que se ejecuta n veces [S1] i = 1, nj = 1, n	**O(n^2)**

Tabla 4.4 Orden de Complejidad para distintas estructuras de control

Ejemplo 6
Analicemos el tiempo de ejecución del siguiente segmento de código

```
Si (n> m)
      entonces    n= n *m
       sino
            Para i desde 0 hasta n-1
                 m= m * n
            FinPara
```

La sentencia que corresponde al entonces tiene orden de ejecución O(1), y el orden del sino es O(n), por tanto orden del si es O(max(1,n))= O(n)

Ejemplo 7
Analicemos el tiempo de ejecución de los bucles

```
1. Para i desde 1 hasta n
2.    Para  j  desde  1 hasta  n
3.          a[i,j] = 0
4.    FinPara
5. FinPara
```

Como se observa, los límites de los lazos indican la cota superior, ya que se refieren al número de veces exacto que se repite el ciclo.

Procediendo desde adentro hacia afuera, la línea 3 consume un tiempo del orden O(1) (3 unidades de tiempo). El lazo de la línea 2 se realiza en n ocasiones. Como su cuerpo consume un tiempo O(1) y podemos despreciar los tiempos O(1) que resulta incrementar y comparar la variable de control j, entonces el tiempo de ejecución de las líneas 2 y 3 es O(n). Un análisis idéntico es válido para el ciclo exterior que, como se realiza también n veces, nos proporciona en forma conjunta un tiempo de $O(n^2)$ para este segmento de algoritmo.

Ejemplo 8
Para el caso del siguiente segmento de código:

```
1. Para  j desde  i+1 Hasta n
2. Si  (a[j] > maxi)
3.        Entonces maxi= a[j]
4. FinSi
5. FinPara
```

Como se observa, la línea 2 consume un tiempo O(1) para realizar el test y la línea 3, en caso de ejecutarse, también es de orden O(1). Por ello, como tanto el incremento, el testeo de la variable de control y el tiempo para ejecutar el cuerpo del ciclo son O(1), el tiempo total de una sola iteración es O(1).

Veamos entonces:

el número de veces que se realiza el ciclo está dado por la fórmula: Límite superior menos límite inferior más uno

Por ello, para el ejemplo que estamos analizando, el número de veces que se realiza el ciclo es, n - (i+1) + 1=n-i.

Por lo tanto el tiempo de ejecución de este ciclo es (n-i) * O(1), es decir de orden O(n-i).

Ejemplo 9
Calculemos el orden de eficiencia del siguiente segmento de algoritmo:

```
1. Si  ( a<b)
2.   Entonces  Para  m desde 1 Hasta k
3.                  Para n desde 1 Hasta k
4.                    p= m*n
5.                  FinPara
6.               FinPara
7.     Sino     Para  m Desde 1 Hasta k
8.                        s=m*m
9.               FinPara
10 .FinSi
```

El tiempo de ejecución de la línea 1 es O(1), el de las líneas 2 a 6 es de orden $O(k^2)$, y el de las líneas 7 a 9 es O(k). Como el peor caso se produce cuando las condición de la línea 1 es verdadera, se dice que el orden de este bloque de algoritmo es $O(k^2)$.

Para resolver el siguiente ejemplo, se utilizan algunas propiedades de las series cuyas demostraciones se han desarrollado en el Anexo I.

Ejemplo 10
Calculemos el orden de eficiencia del siguiente segmento de algoritmo que posee dos iteraciones anidadas.

```
Comienzo
Constante  N=10
entero i, j, may, aux, A[N]

Para i Desde  1 Hasta N-1
      may = i
      Para j Desde  i + 1 Hasta N
      Si  (A[j] > A[may])
          Entonces
                may = j
                aux = A[may]
                A[may] = A[i]
                A[i] =aux
      FinSi
      FinPara
FinPara
Fin
```

Solución
Se debe observar que si bien el bucle externo se realiza una cantidad determinada de veces(N-1), la cantidad de veces que se ejecuta el bucle interno no es constante, varía para cada valor de i. Por lo cual la cantidad de veces que se ejecutan ambos ciclos resulta de aplicar la sumatoria de la siguiente serie aritmética:

Cuando i =1 el bucle se ejecuta (N-(1+1)+1) = N-1 veces

Para i =2 el bucle se ejecuta N-2 veces

:

Para i =N -1 el bucle se ejecuta 1 vez

Entonces Cantidad total = (N-1)+(N-2)+…+1

Aplicando la propiedad de **suma de series aritméticas**, dada por la ecuación (2) del anexo 1, que expresa que la sumatoria de una serie aritmética *es igual a la suma del primer elemento más el último, multiplicado por la mitad de la cantidad de elementos de la serie,* resulta:

$$\sum_{i=1}^{N-1} i = (1+(N-1))*(N-1)/2 = N(N-1)/2$$

Por lo tanto, la eficiencia de este algoritmo $\in O(N^2)$

Ejemplo 11
Analicemos la eficiencia del siguiente segmento de código que presenta un bucle donde la evolución de la variable de control es ascendente no lineal.

```
Constante n=100
entero c = 1, vector[n], aux
Mientras(c < n)                          O(log n)
  Si(vector[c] < vector[n])
        entonces  aux = vector[n]
                  vector[n] = vector[c]
                   vector[c] = aux
   Finsi
   c = c * 2
FinMientras
```

Observemos los valores de la variable c en las distintas iteraciones del ciclo:

c	
1	Inicio
2	Finalizar iteración 1 - 2^1
4	Finalizar iteración 2- 2^2
8	Finalizar iteración 3 2^3
:	
x	Finalizar iteración k 2^x

Como se observa, el valor inicial de c es 1, al finalizar la primera iteración $c=2^1$, al cabo de x iteraciones $c= 2^x$
Si c <n luego $2^x < n$.

Aplicando $\log_2 n$ a esta desigualdad se obtiene, $\log_2 2^x < \log_2 n$.

Por propiedades de logaritmo: $x \log_2 2 < \log_2 n$

Por tanto la cantidad máxima de iteraciones $x = \log_2 n$.
Si el orden de complejidad del cuerpo del ciclo es O(1) entonces el orden de complejidad del bucle es:
O(log n) * O(1) = **O(log n),** complejidad logarítmica.

Actividad 1
Calcular el orden de eficiencia del siguiente trozo de algoritmo que posee tres iteraciones anidadas. Para mayor simplicidad, no se debe considerar el tiempo de las Acciones de orden O(1).

```
Algoritmo Calculo
Comienzo
entero i, j, k ,c
c=0
```

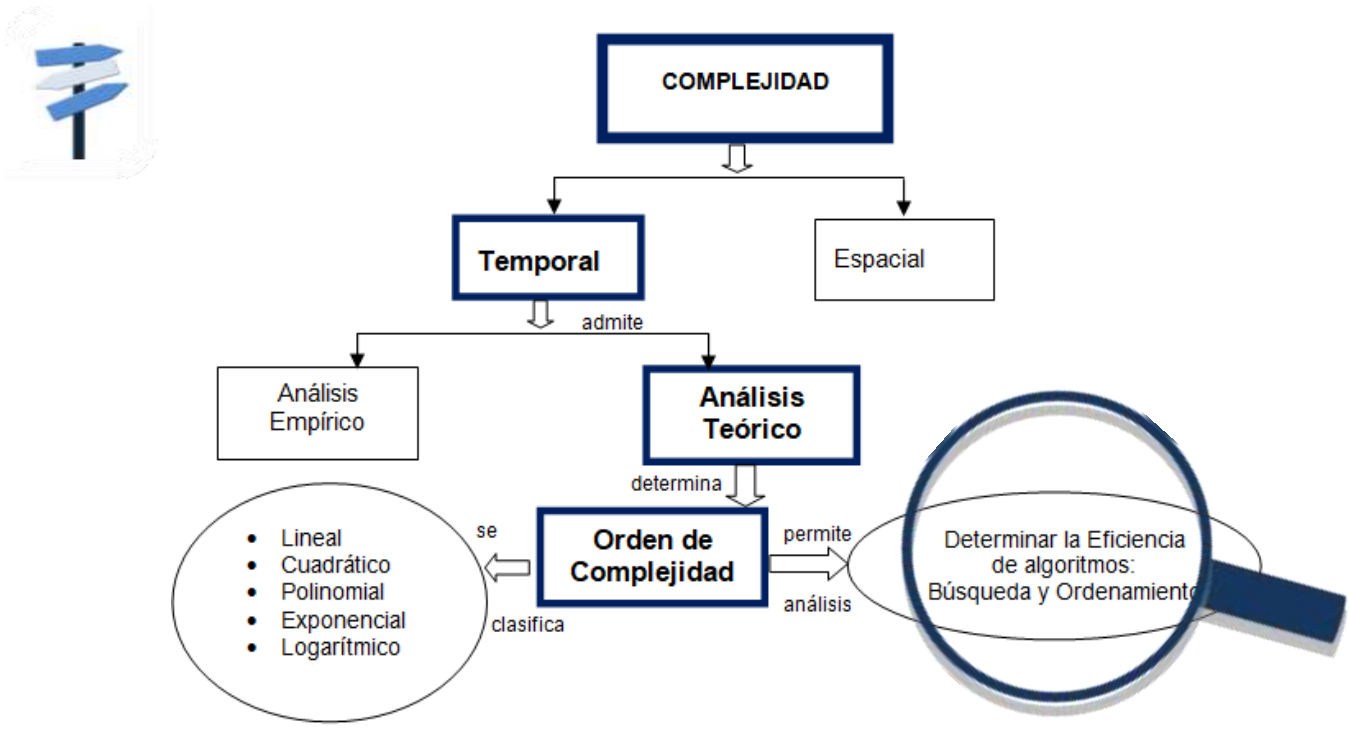
```
Para i Desde 1 Hasta N-1
   Para j Desde i+1 Hasta N + 1
      Para k Desde 1 Hasta  j
         Escribir i* j*k
          c=c+1
      FinPara
   FinPara
FinPara
Escribir c
Fin
```

Nota: Ver solucion en anexo III

Actividad 2

Analizar el orden de complejidad del siguiente segmento de código formado por un bucle donde la evolución de la variable de control no es lineal, junto a un bucle con evolución de variable lineal.

```
entero c, x, i
Para i desde  0 Hasta  N-1
     c = i
       Mientras(c > 0)
         x = x / c
         c = c / 2
       Fin Mientras
      x = x + 2
Fin Para
```



4.3 Eficiencia de Algoritmos de búsqueda

Los algoritmos de búsqueda son aquellos que permiten ubicar un dato en un conjunto determinado. En la unidad anterior se analizaron distintos algoritmos de búsqueda secuencial en arreglos. También se observó que el algoritmo a utilizar tiene dependencia directa de la forma de organización de los datos y que para el caso de arreglos ordenados es conveniente el uso de la búsqueda binaria.

Eficiencia de la búsqueda lineal: peor caso

El análisis teórico debe estar basado en contabilizar la cantidad de veces que se realiza alguna acción que resulte esencial en el algoritmo. Para el caso de la búsqueda lineal, el algoritmo consiste en comparar elementos, por lo que el análisis se realizará teniendo en cuenta la relación entre el número de elementos del arreglo y la cantidad de comparaciones necesarias para encontrar un elemento dentro de él. Esto significa que se debe buscar una función que relacione las variables: número de comparaciones, (variable dependiente) y cantidad de componentes del arreglo (variable independiente).

El método de búsqueda secuencial requiere, en el peor de los casos en que el elemento buscado está al final del arreglo o no está en él, consultar los N (o N+1) elementos que lo constituyen. En este caso el tiempo de búsqueda es directamente proporcional al número de elementos del arreglo, se dice entonces que el tiempo es de orden N, lo que se simboliza $T \epsilon O(N)$.

Si la función de eficiencia de un algoritmo se representa como f(n) y teniendo en cuenta que el tiempo de búsqueda es directamente proporcional al número de elementos, se puede expresar

$$T = k * N \quad , \text{ siendo} \quad o < k <= 1$$

Por tanto el tiempo máximo de búsqueda para arreglos de N elementos, responde a la función lineal:

$$T \in O(N)$$

Análisis comparativo de las tres funciones de búsqueda secuencial

Recordando que:

La Búsqueda Secuencial o Lineal consiste en recorrer y examinar cada uno de los elementos del arreglo, desde el primero, hasta encontrar el elemento buscado o hasta que se hayan examinado todos los elementos del arreglo sin éxito.

T(N) representa unidades de tiempo (segundos, milisegundos,...) que un algoritmo tardaría en ejecutarse con unos datos de entrada de tamaño N.
T(N) representa el tiempo de ejecución del algoritmo en un ordenador idealizado, donde cada una de las instrucciones simples consume una unidad de tiempo.

Hallemos la función T(N) para cada una de las formas de búsqueda secuencial vistas:

Forma 1: utilizando una bandera lógica

```
booleano  Bandera (arreglo arre, entero elem)
Comienzo
entero i
 booleano está
  i=0                                   1ut
esta=falso                              1ut
Mientras ((i<N) y ( esta ==falso))      3ut  (evaluación de expresión lógica)
   Si (arre[i] ==elem)                  2ut
     Entonces  esta= verdadero          1ut      ≡  4ut
     Sino        i=i+1                  2ut
   FinSi
FinMientras
retorna(esta)                           1ut
Fin
```

- En el mejor de los casos, cuando el elemento se encuentra en la primera componente del arreglo, **T(N)= 12 ut**

- En el peor de los casos, cuando el elemento no está en el arreglo, la sentencia iterativa **Mientras** se ejecuta N veces; entonces**.**

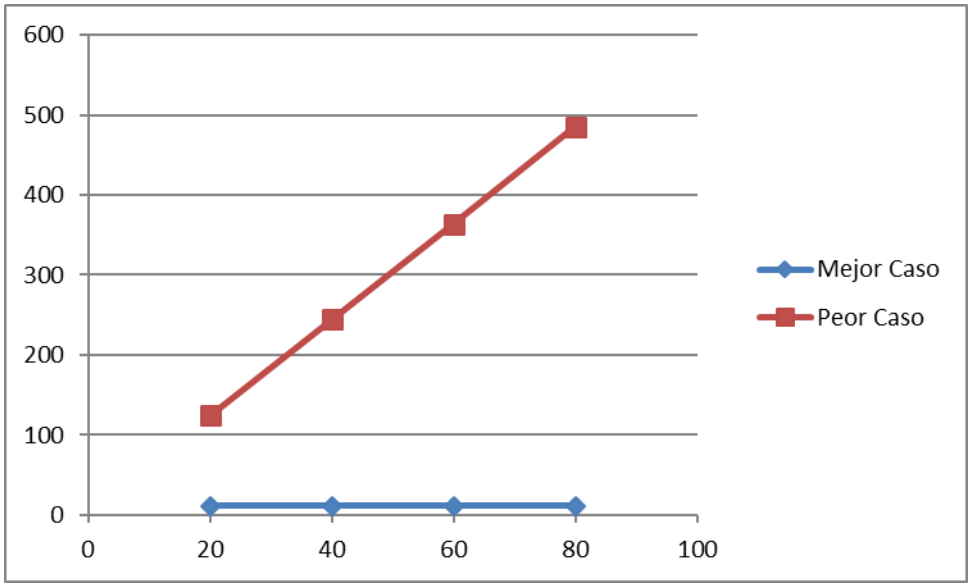
Cuerpo del ciclo 4*N

Condición 3*N +1 **Total** 7*N+1

Si sumamos las asignaciones que están fuera del ciclo obtenemos: **T(N)= 7 *N+4**

Por tanto podemos decir que el tiempo de ejecución para este algoritmo varía entre

12 ut ≤ T(N) ≤ 7*N+4 ut



.

El mejor caso es la cota inferior y el peor caso la cota superior

Analicemos ahora el tiempo para la forma 2 estudiada:

Forma 2: sin utilizar bandera lógica

```
entero SinBandera (arreglo arre, entero elem)
Comienzo
entero i
i=0                                   1ut
Mientras ((i<N) y (arre[ i] <> elem))   4ut (evaluación de expresión lógica)
     i=i+1                            2ut
FinMientras
retorna( i )                          1ut
Fin
```

- En el mejor de los casos, cuando el elemento se encuentra en la primera componente del arreglo, **T(N)= 6 ut**

- En el peor de los casos, cuando el elemento no esté en el arreglo, la sentencia iterativa **Mientras** se ejecuta N veces; entonces:

  tiempo de ejecución del cuerpo 2*N

  tiempo de ejecución de condición 4*N ut más **1ut** de evaluar por última vez la expresión relacional i<N; lo que da un total de **6*N+1 ut.**

Si sumamos todas las unidades de tiempo detalladas obtenemos: **T(N)=6*N+3**

Por tanto podemos decir que el tiempo de ejecución para este algoritmo varía entre

$$6 \text{ ut} \leq T(N) \leq 6*N+3 \text{ ut}$$

De esta manera hemos probado que la forma 2 es más eficiente que la forma 1, como habíamos inferido en la unidad anterior.

Forma 3: utilizando un elemento centinela.

```
entero Centinela (arreglo1 arre, entero elem)
Comienzo
entero i
i=0                                   1ut
Mientras (arre[i]<>elem)              2ut (evaluación de expresión relacional)
      i=i+1                           2ut
FinMientras
retorna(i)                            1ut
Fin
```

- En el mejor de los casos, cuando el elemento se encuentra en la primera componente del arreglo, **T(N)= 4ut**
- En el peor de los casos, cuando el elemento sea la N+1 ésima componente del arreglo, que está en la posición N , la sentencia iterativa **Mientras** se ejecuta N veces; por tanto el tiempo de ejecución es**: 4*N* ut** más 2**ut** de evaluar por última vez la expresión relacional arre[i]<>elem; lo que da un total de **4*N+2 ut.**

  Si sumamos todas las unidades de tiempo detalladas obtenemos: **T(N)=4*N+4**

Por tanto podemos decir que el tiempo de ejecución para este algoritmo varía entre

4 ut ≤ T(N) ≤ 4*N+4 ut

Hemos probado que la forma 3 es más eficiente que las 2 formas anteriores.

En el siguiente cuadro comparativo analizamos las funciones T(N) obtenidas:

	Mejor caso	**Peor caso**
Función Bandera	12 ut	7*N+4
Función SinBandera	6 ut	6*N+3
Función Centinela	4 ut	4*N+4

Tabla 4.5 Comparativo funciones T(N) Búsqueda Secuencial

Conclusión: los valores del cuadro muestran que tanto en el peor como en el mejor de los casos, la función correspondiente a la búsqueda secuencial Centinela es la que proporciona menor tiempo de ejecución, para N>=2.

Eficiencia de la Búsqueda binaria

Como se ha dicho, para arreglos de gran dimensión que estén ordenados resulta más eficiente la Búsqueda Binaria. Se determinará ahora la función matemática que muestre que este algoritmo de búsqueda es más eficiente que el de la búsqueda lineal.

```
Algoritmo BB
constante n=100

void carga_arreglo( entero xa[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
     Leer xa[i]
FinPara
Fin

entero Búsqueda_Binaria(entero xa[n], entero xnro)
comienzo
entero li, mi,ls
li=0
ls=n-1
mi=( li+ls) div 2

Mientras((li<=ls) y (xnro <> a[mi]))
     Si   ( xnro < a[mi] )
                Entonces ls = mi -1
                Sino        li = mi+1
    FinSi
mi =( li+ ls) div 2,
FinMientras
```

```
Si  (li > ls)
  Entonces  retorna(-1)
  Sino      retorna(mi)
Finsi
Fin

// Algoritmo Principal
Comienzo
entero a[n], nro,pos
carga_arreglo(a)
Leer nro
pos=Búsqueda_Binaria(a, nro)
si (pos==-1)
   entonces Escribir  "El número ", nro , "  no se encontró en el arreglo "
   sino Escribir "El numero ", nro , " se encontró en la posición ", pos
finsi
fin
```

Análisis del peor de los casos

Recordemos que el análisis teórico de la eficiencia de un algoritmo debe estar basado en contabilizar la cantidad de veces que se realiza alguna acción que resulte esencial en el mismo. En el caso de los algoritmos de búsqueda, la acción que se realiza es la comparación del elemento a buscar, con las distintas componentes del arreglo. Por esto, para determinar la eficiencia se analiza el número de comparaciones expresado como una función del número de elementos del arreglo.

En este algoritmo de búsqueda, con cada comparación se divide en 2 mitades el arreglo. Por lo tanto, si el tamaño del arreglo es N, los tamaños sucesivos de los subarreglos son:

$$N/2 \quad , \quad N/4 \quad , \quad N/8 \quad , \quad N/16 \quad ...$$

ó

$$N/2^1 \quad , \quad N/2^2 \quad , \quad N/2^3 \quad N/2^4 \quad .....$$

1^a comparación	2^a comparación	3^a comparación	4^a comparación	.....

Por lo tanto al terminar el proceso, el tamaño del subarreglo será mayor o igual a 1, se encuentre o no el elemento. Si se llama K al número de comparaciones, se verifica:

$$N/2^k \geq 1 \quad \Rightarrow \quad N \geq 2^k$$

Aplicando logaritmo en base 2 a ambos miembros:

$$\log 2\, N \geq K \,.\, \log_2 2 \quad \Rightarrow \quad K \leq \log_2 N$$

Esto indica que el número de comparaciones que se realizarán para buscar un elemento en un arreglo de N componentes, hasta llegar a obtener el subarreglo de menor tamaño (1 elemento) será **a lo sumo** $\log_2 N$.

Por lo tanto, la eficiencia de la búsqueda binaria es de orden:

$$T \in O(\log_2 N)$$

Comparación de eficiencia de los métodos de Búsqueda Secuencial y Binaria para el peor de los casos.

El siguiente gráfico permite comparar la eficiencia de ambos métodos de búsqueda estudiados.

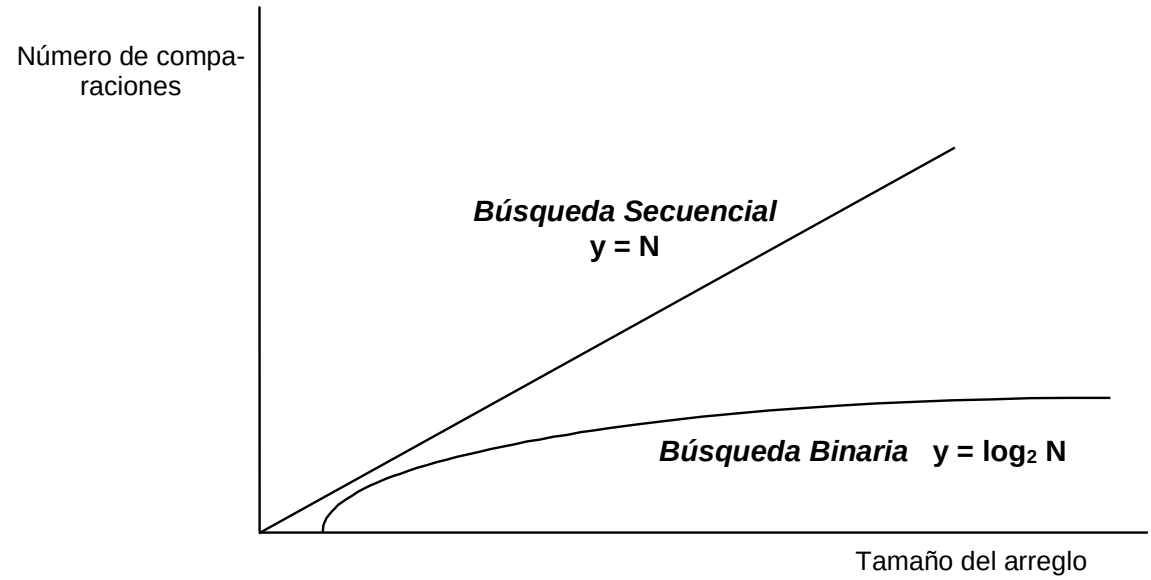


Figura 4.5 Eficiencia algoritmos de Búsqueda Secuencial Vs. Búsqueda Binaria

Sobre el eje x se considera el tamaño del arreglo y sobre el eje y la cantidad máxima de comparaciones, para los dos métodos.
Se observa que a mayor número de componentes, más eficiente es el método de búsqueda binaria.
Así, si un arreglo tiene 1.024 elementos, la búsqueda lineal en el peor de los casos requiere 1.024 comparaciones, mientras que la búsqueda binaria no requerirá más de $\log_2 1.024 = 10$, es decir alrededor de 10 comparaciones. A este tiempo deberá agregarse, si el arreglo está desordenado, el tiempo empleado en ordenarlo.

La siguiente tabla muestra la cantidad máxima de comparaciones de ambos métodos de búsqueda para arreglos de distintos tamaños.

N	B. Secuencial	B. Binaria
16	16	4
64	64	6
1024	1024	10

Tabla 4.6 Cuadro Comparativo Eficiencia de Algoritmos de Búsqueda

4.4 Eficiencia de los métodos de Ordenamiento

Como se dijo, ordenar un vector es una operación que permite disponer sus elementos en un orden secuencial de acuerdo con un criterio determinado, en orden ascendente o descendente.

Existen distintas técnicas de ordenación de vectores, la eficiencia de un método se determina en función de 2 criterios:

Uso eficiente de memoria: algunos métodos para ordenar vectores utilizan un arreglo auxiliar. Sin embargo, los que se estudiarán aquí son métodos que sólo utilizan el arreglo original, donde se intercambian los elementos que están desordenados, de modo de optimizar el almacenamiento.

Eficiencia o economía de tiempo: la rapidez del método depende fundamentalmente de:

- Número de comparaciones entre componentes.
- Número de movimientos o intercambios entre componentes.

En la siguiente sección se analizará la eficiencia de algunos de los algoritmos de ordenamiento más utilizados.

Eficiencia de los Métodos de Ordenamiento por Intercambio

Método de la burbuja mejorado ("buble sort")

Este método de ordenamiento, que fue analizado en la unidad anterior, tiene características que se resumen como sigue.
Si A es un arreglo de dimensión N, cuyas componentes deben ser ordenadas ascendentemente, el método consiste básicamente en:
Realizar un proceso iterativo mediante una serie de pasadas, en cada una de las cuales se comparan pares de elementos adyacentes, ordenándolos si están desordenados.
Al finalizar la primera pasada el elemento mayor queda ubicado en la última posición, y en las distintas pasadas los elementos más grandes tienden a desplazarse hacia la derecha.
En cada pasada debe detectarse la posición del último cambio realizado, para que en la siguiente las comparaciones, que siempre comienzan desde el primer elemento, se realicen hasta una posición anterior al último intercambio realizado.
Si durante alguna pasada no se realizan intercambios, el algoritmo finaliza por haber quedado el arreglo ordenado.

NOTA: Recordar que para realizar correctamente un intercambio se debe utilizar una variable auxiliar que permita resguardar el valor de una de las variables que se compara, para finalizar produciendo el cambio requerido.

El algoritmo que responde a estos requerimientos se muestra a continuación:

```
Algoritmo OrdenaBurbuja_M
constante n=10

void carga_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
    Leer xA[i]
FinPara
retorna()
Fin
```

```
void Ordena_Burbuja_Mejorado(entero xA[n])
comienzo
  entero  k, i , aux, cota
  cota= n-1
  k= 1
  Mientras (k <> -1)
    k= -1
    Para i desde 0 hasta  cota-1
      Si   (xA[i] > xA [i+1])
        entonces   aux = xA [i]
                   xA [i] = xA [i+1]
                   xA [i+1] =aux
                    K =i
      FinSi
    FinPara
    cota =k
  FinMientras
retorna()
Fin

void muestra_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
    Escribir xA[i]
FinPara
retorna()
Fin

// Algoritmo Principal
Comienzo
entero A[n]
carga_arreglo(A)
Ordena_Burbuja_Mejorado(A)
muestra_arreglo(A)
Fin
```

Se analizará la eficiencia de este método de ordenamiento para el peor de los casos, es decir cuando el arreglo esté invertido, para lo cual se considerarán los dos criterios mencionados: el número de comparaciones y el número de intercambios entre componentes.

Número de comparaciones

Pasada	Comparaciones
1	N-1
2	N-2
3	N-3
:	:
N -1	1

Tabla 4.7. Cantidad de comparaciones por pasada

(peor caso del método de la burbuja)

El número total de comparaciones es entonces (N-1) + (N-2) +(N-3) + … 3 +2 +1

Por suma de series aritméticas.(demostración en Anexo I)

(N-1) + (N-2) +(N-3) + … 3 +2 +1= ((N -1)+1) * (N-1) /2 =N * (N-1)/2= 1/2 (N^2 - N)

Su eficiencia es una función cuadrática de N, O(N^2) , lo que significa que si bien es muy sencillo es poco eficiente.

Este es el *peor de los casos*, que no es el caso más frecuente que se da en la práctica, en general los arreglos quedarán ordenados en pasadas intermedias, reduciendo de esta manera el número de comparaciones.

Para el *mejor de los casos*, cuando el arreglo esta ordenado inicialmente, este algoritmo debe realizar sólo una pasada y por tanto N -1 comparaciones.

(N -1)<= Comparaciones < = N* (N -1)/2

b- Número de movimientos o transferencias entre componentes

Cuando el arreglo está ordenado, este método no realiza intercambios. En el peor de los casos, en cada comparación se intercambian dos elementos del vector, es decir se realizan tres movimientos.

Tendremos que la cantidad de movimientos es:

0<= Intercambios< = 3* N (N -1)/2

Método de Selección

Para ordenar el arreglo A de N elementos en forma ascendente, primero busca el menor elemento entre los N, y lo coloca en la primera posición, busca nuevamente el menor entre los N-1 elementos restantes y los coloca en la segunda posición. Repite N –1 veces este proceso hasta colocar todos los elementos en la posición que les corresponde.

Supongamos que se quiere ordenar el siguiente arreglo:

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 1: El menor elemento de los 6 del arreglo es el 4 que se coloca en la primera posición

4	21	40	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se intercambia a[0] con a[2]

Pasada 2: El menor elemento de los 5 restantes del arreglo es el 9 que se coloca en la segunda posición

4	9	40	21	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se intercambia a[1] con a[3]

Pasada 3:

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se intercambia a[4] con a[2]

Pasada 4:

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se intercambia a[4] con a[5]

El algoritmo que trabaja el método de Selección es el que sigue:

```
Algoritmo Selección
Constante n=10

void carga_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
     Leer xA[i]
FinPara
Fin

void Ordena_Selección(entero xA[n])
comienzo
  entero i ,j, min, aux
      Para i Desde 0 Hasta  n-2
          min=i
          Para j Desde  i+1 hasta  n -1
                Si (xA [ j ] <x A [min])       /* busca el mínimo entre los elementos a[i] .....a[n-1] */
                   Entonces   min=j
               FinSi
           FinPara
         aux= xA [ i ]                                  /* intercambia los valores */
         xA [ i ] = xA [min]
         xA [ min] = aux
      FinPara
  Fin

void muestra_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
     Escribir xA[i]
FinPara
Fin
```

// Algoritmo Principal
Comienzo
Entero A[n]
Carga_arreglo(A)
Ordena_Selección(A)
Muestra_arreglo(A)
Fin

Se observa que para ordenar el vector son necesarias N -1 pasadas, ya que en la última se ubican los el elementos N-1 y N-ésimo.

Se analizará la eficiencia de este método de ordenamiento para el peor de los casos, es decir cuando el arreglo esté invertido, para lo cual se considerarán los dos criterios mencionados: el número de comparaciones y el número de intercambios entre componentes.

Recordemos que la acción estructurada iterativa Para-finpara **siempre** se ejecuta la siguiente cantidad de veces: **lím sup – lím inf + 1**

ANÁLISIS DE LA CANTIDAD DE COMPARACIONES
Observamos que en el código existe una única comparación: (**a[j] < a[min]**), la cual se encuentra ubicada en el para-finpara de adentro.
Observamos también que **en el mejor caso** (cuando el arreglo ya está ordenado) y **en el peor caso** (cuando el arreglo está ordenado de manera inversa) **coincidirán la cantidad de comparaciones.**
Hallemos esa cantidad:

El Para-finpara de afuera se ejecuta: N-2 -0 + 1= **N-1 veces**

Veamos cuantas veces se ejecuta el Para-finpara de adentro:

Cuando i=0, j varía desde 1 hasta N-1,
entonces el para-finpara se ejecuta: N-1 -1 + 1= **N-1 veces**

Cuando i=1, j varía desde 2 hasta N-1,
entonces el para-finpara se ejecuta: N-1 -2 + 1= **N-2 veces**

Cuando i=2, j varía desde 3 hasta N-1,
entonces el para-finpara se ejecuta: N-1 -3 + 1= **N-3 veces**
……
Cuando i=N-2, j varía desde N-1 hasta N-1,
entonces el para-finpara se ejecuta: N-1 –(N-1) + 1= **1 vez**

Si **sumamos las veces** que se ejecuta el para de adentro obtenemos:
(N-1) + (N-2) + (N-3) + . . . + 1 = ((N-1) + 1) * (N-1)/2

Esta última igualdad se obtiene aplicando la propiedad de series aritméticas (que es el primer miembro de esta igualdad): se suman el primer y el último término (N-1) y 1; se lo multiplica por la cantidad de términos que tiene esa serie aritmética (N-1) que es la cantidad de veces que se ejecuta el para-finpara de afuera, y se lo divide en dos. Quedando así, la siguiente fórmula: **N*(N-1)/2**

Entonces , **la cantidad de Comparaciones** que se realizan **en el mejor** y **en el peor caso** es: **N*(N-1)/2**

ANÁLISIS DE LA CANTIDAD DE INTERCAMBIOS
Observamos que en el código existen tres asignaciones, correspondientes a los intercambios; los cuales están ubicados en el para-finpara de afuera. Como hemos dicho que esta estructura **siempre** se ejecuta N-1 veces, entonces siempre se van a realizar **3* (N-1) intercambios, en el mejor y en el peor caso.**

Método de inserción directa

Sea **A** un vector de N componentes que debe ordenarse en forma ascendente. Este método consiste en insertar un elemento A[i], en el lugar que le corresponde entre los anteriores A[0]....A[i-1], que ya están ordenados .

Para ello se comienza con el segundo elemento del arreglo, si los dos primeros elementos están desordenados, los intercambia. Toma ahora el tercer elemento y busca la ubicación que le corresponde respecto de los 2 anteriores. Si A[2] es mayor o igual que A[1] significa que está en la posición correcta. En caso contrario, debe comparar A[2] con A[0]. Si A[2] es mayor o igual que A[0] lo inserta entre A[0] y A[1], pero si es menor que A[0], debe colocarlo en la posición A[0] y correr A[1] y A[2] una posición a la derecha.

En general, para insertar el elemento que está en la i-ésima posición debe compararlo con los i-1 elementos anteriores y ubicarlo en la posición que le corresponde.

Este proceso se repite N-1 veces, hasta que quedan todos en su posición correcta.

```
Algoritmo Inserción_Directa
Constante n=10

void carga_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
     Leer xA[i]
FinPara
Fin

void Ordena_Inserción_Directa(entero xA[n])
comienzo
  entero   i ,j ,valor


      Para i Desde 1 Hasta  n-1
          valor  = xA[i]
          j= i-1
         Mientras (( j >= 0 ) y ( valor < xA [ j ] ))
               xA [ j+1]= xA [j]                /* los valores se corren un lugar a la derecha */
               j=j-1
         FinMientras
         xA[j+1] = valor
      FinPara
fin
```

```
void muestra_arreglo( entero xA[n])
comienzo
entero i
Para  i Desde  0 Hasta  n-1
     Escribir xA[i]
FinPara
Fin

// Algoritmo Principal
Comienzo
entero A[n]
carga_arreglo(A)
Ordena_Inserción_Directa(A)
muestra_arreglo(A)
Fin
```

Para ordenar en forma ascendente el arreglo A:

40	21	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 1:

21	40	4	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 2:

4	21	40	9	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 3:

4	9	21	40	10	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 4:

4	9	10	21	40	35
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Pasada 5:

4	9	10	21	35	40
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Se analizará la eficiencia de este método de ordenamiento para el peor de los casos, es decir cuando el arreglo esté invertido, para lo cual se considerarán los dos criterios mencionados: el número de comparaciones y el número de intercambios entre componentes.

ANÁLISIS DE LA CANTIDAD DE COMPARACIONES

El Para-finpara se ejecuta: N-1 -1 + 1= **N-1 veces**

Observamos que en el código existe una única comparación: **valor < a[j]** , la cual se encuentra ubicada en el mientras-finmientras.
En el mejor caso (cuando el arreglo ya está ordenado), esa comparación se realiza **N-1 veces**, aunque nunca se ejecute el cuerpo del mientras, pues siempre esa comparación da Falso.
Por lo tanto **la cantidad mínima de comparaciones es: N-1**
En el peor caso (cuando el arreglo está ordenado de manera inversa) debemos analizar la cantidad de veces que se ejecuta el Mientras-finmientras, para hallar la cantidad máxima de comparaciones:
Hallemos esa cantidad:

El Para-finpara de afuera se ejecuta: N-1 -1 + 1= **N-1 veces**
Cuando i=1, j varía desde 0 hasta 0, entonces el mientras-finmientras se ejecuta: **1 vez**
Cuando i=2, j varía desde 1 hasta 0, entonces el mientras-finmientras se ejecuta: **2 veces**
Cuando i=3, j varía desde 2 hasta 0, entonces el mientras-finmientras se ejecuta: **3 veces**
……
Cuando i=N-1, j varía desde N-2 hasta 0, entonces el mientras-finmientras se ejecuta: **N-1 veces**

Si **sumamos las veces** que se ejecuta el mientras finmientras obtenemos:
1 + 2 + 3 + . . . + (N-1) = (1 + (N-1)) * (N-1)/2
Esta última igualdad se obtiene aplicando la propiedad de series aritméticas (que es el primer miembro de esta igualdad): se suman el primer y el último término 1 y (N-1); se lo multiplica por la cantidad de términos que tiene esa serie aritmética (N-1) que es la cantidad de veces que se ejecuta el para-finpara , y se lo divide en dos. Quedando así, la siguiente fórmula: **N*(N-1)/2**
Entonces, **la cantidad de Comparaciones** que se realizan **en el peor caso** es: **N*(N-1)/2**
Concluimos en que:

(N-1) <= Cantidad de comparaciones <= N* (N-1)/2

ANÁLISIS DE LA CANTIDAD DE INTERCAMBIOS

Observamos que en el código existen **tres asignaciones,** correspondientes a los intercambios:

Dos están ubicadas en el para-finpara: **Valor = a[i]** y **a[j + 1] = valor**
Y otra ubicada dentro del mientras-finmientras: a[j + 1] = a[j]

En el **mejor de los casos**, cuando el arreglo está ordenado, el mientras-finmientras **nunca** se ejecuta; por lo tanto, solo se realizan los **dos** intercambios o asignaciones ubicados en el para-finpara , estructura que se ejecuta N-1 veces. Como vemos, se realizan **2*(N-1) intercambios.**

En el **peor de los casos**, cuando el arreglo está ordenado de manera inversa; haciendo la misma deducción de la cantidad de veces que se ejecuta el mientras-finmientras y recordando la cantidad de veces que se ejecuta el para-finpara, obtenemos la siguiente cantidad de intercambios:
(1+2) + (2 + 2) + (3 + 2)+ . . . + (N-1 + 2) **(1)**
Observamos que en cada término siempre sumamos 2, pues son dos los intercambios que se hacen en el para-finpara; y vamos sumando en cada término: 1, 2, 3, . . ., N-1 correspondientes a la cantidad de intercambios, cada vez que se ejecuta el mientras-finmientras
Volviendo a la fórmula **(1),** obtenemos: 3 + 4 + 5 + . . . + (N+1) una serie aritmética, cuyo primer término es 3, el último término es N+1. Aplicando la misma propiedad de series aritmé-

ticas y teniendo en cuenta que la serie tiene N-1 términos (las veces que se ejecuta el para-finpara), obtenemos:

$$(\mathbf{3+ (N+1))* (N-1) / 2 = (4+N)* (N- 1) /2}$$

Concluimos en que: **2*(N-1) < = cantidad de intercambios < = (4+N)* (N- 1) /2**

La siguiente tabla muestra la eficiencia de cada uno de los métodos de ordenamiento analizados, para el caso de un arreglo de N componentes. Se muestran los dos criterios de eficiencia a tener en cuenta: número de comparaciones y cantidad de movimientos o intercambios realizados; para el mejor caso (el arreglo está ordenado) y el peor caso (cuando el arreglo está totalmente invertido).

Métodos	**Cantidad de Comparaciones**		**Cantidad de intercambios**	
	Mejor Caso	*Peor caso*	*Mejor Caso*	*Peor Caso*
Burbuja Mejorado	N -1	1/2 (N^2 - N)	0	3 N (N -1)/2
Selección	1/2 (N^2 - N)	1/2 (N^2 - N)	3(N -1)	3(N -1)
Inserción	N -1	1/2 (N^2 - N)	2(N -1)	(4+ N)(N -1) /2

Tabla 4.8 Comparación teórica de eficiencia de Algoritmos de Ordenamiento

Teniendo en cuenta el número de comparaciones, cuando el arreglo está ordenado, los algoritmos que funcionan de manera más óptima son los métodos de la burbuja y de inserción. El primero realiza un única pasada, N -1 comparaciones, detecta que el arreglo está ordenado y termina la ejecución. El método de inserción compara cada uno de los elementos con el anterior y determina que el elemento debe quedar en el lugar en que está.

El método de selección por su parte es independiente del orden de los elementos del arreglo, la cantidad de comparaciones es siempre la misma.
Atendiendo a la cantidad de movimientos, para el caso de un arreglo ordenado el método de la burbuja no realiza cambios, sin embargo los otros dos métodos sí.

El análisis realizado, muestra la eficiencia de los algoritmos para los casos extremos, que no son los más comunes en los casos reales. En la mayoría de las aplicaciones los vectores estarán organizados en forma aleatoria, para estos casos, el método de la burbuja tiene la ventaja de detener la ejecución si detecta que el arreglo ha quedado ordenado en pasadas intermedias.

Actividad 5:
Como se observa en la tabla anterior, en el peor caso, los métodos de inserción y de la burbuja son cuadráticos. ¿Podrías probar cual es el más óptimo e indicar para que tamaño de arreglo?

4.5 Bibliografía

- Aho Alfred V , John E. Hopcroft, Jefrey D. Ullman (1988)Estructura de datos y algoritmos versión en español de Américo Vargas y Jorge LozanoPUBLICACIÓN México, DF : Addison-Wesley. ISBN 968-6048-19-7I Iberoamericana: Sistemas Técnicos de Edición,
- Braunstein Silvia y A. Gioia (1987) Introducción a la Programación y a La Estructura de Datos. Eudeba
- Casanova Faus, A(1993).Programación. Escuela Universitaria de Informática. Universidad Politécnica de Valencia. ISBN 84-7721-233-3.
- De Giusti, Armando E, Madoz Maria y otros (2001). Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci. LIDI. Facultad de Informática Universidad Nacional de la Plata. Prentice Hall.

Páginas web consultadas

- Berzal Fernado, Análisis de algoritmos
  http://elvex.ugr.es/decsai/c/apuntes/algoritmos.pdf
- Raiger Murillo Moreno. Algoritmos de Ordenamiento:
  http://www.monografias.com/trabajos/algordenam/algordenam.sht
- Tiempo de ejecución. Notaciones para la Eficiencia de los Algoritmos
  http://decsai.ugr.es/~jmfluna/docencia/c0304/eedd/teoria/eficiencia.pdf
- Algoritmos computacionales
  http://www.monografias.com/trabajos11/alcom/alcom.shtml

Unidad 5: Verificación Formal de Algoritmos

Introducción

El diseño formal de algoritmos hace uso de métodos formales para abordar tres fases fundamentales: especificación, derivación y verificación.
La especificación formal de algoritmos es una técnica que permite explicar de una forma breve y precisa qué debe hacer un algoritmo. A partir de esta especificación formal se pueden escribir un algoritmo y verificar formalmente su corrección, o deducir formalmente (derivar) las instrucciones que debe contener el algoritmo.
La verificación formal de algoritmos tiene una gran importancia ya que permite verificar la corrección de un algoritmo antes de ser escrito en un lenguaje concreto (lenguaje de implementación). Es lo que se conoce como verificación a priori.
Tradicionalmente los algoritmos se prueban, una vez escritos, con la ayuda de herramientas de depuración, es lo que se conoce como verificación a posteriori o verificación mediante prueba. Esto quiere decir que, hasta que no se prueba el programa no aparecen los errores.
Al igual que en otra muchas actividades, cuanto más pronto aflore un error menos esfuerzo costará corregirlo, lo que da un gran valor a las técnicas formales de especificación, derivación y verificación.
En este capítulo se trabajará con los conceptos de especificación y verificación formal, (el concepto de derivación se verá en cursos más avanzados de la carrera).
Se presentará e interpretarán conceptos básicos, reglas de inferencia y la lógica de primer orden para realizar una demostración o prueba de corrección de un algoritmo en el formato de secuencia y en selección.

Objetivos de la Unidad

EL objetivo de la unidad es que el alumno sea capaz de interpretar y aplicar correctamente los conceptos de especificación y verificación formal en algoritmos en las estructuras de secuencia y selección.

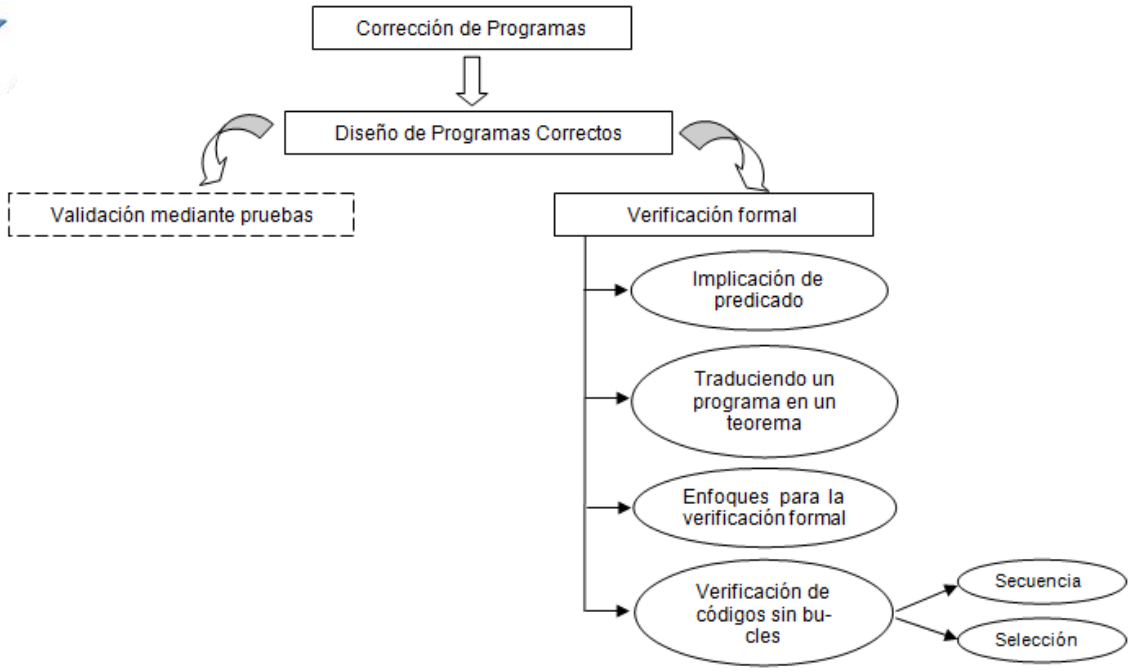
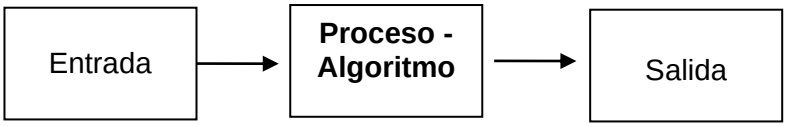


Figura 5.1 Esquema Temático de la Unidad

5.1 Conceptos Introductorios acerca de corrección de programas

En la unidad I vimos que un problema está definido por un estado inicial o de partida. Resolver un problema requiere transformar mediante un proceso (algoritmo), los datos de entrada para obtener la salida requerida.



En la etapa de análisis de un problema se determina el **"qué hacer",** es donde se debe realizar la **Especificación del problema.**
Especificar un problema significa determinar en forma clara y precisa los requerimientos de salida y los datos necesarios para obtener esa salida. Por ello se habla de Especificaciones de Entrada y Especificaciones de Salida.

Análisis **Especificación del Proceso** **(***Qué hacer?)*	Especificación de Salidas
	Especificación de Entradas

La **validación** permite comprobar que un *programa cumple con sus especificaciones*, es decir resuelve correctamente el problema para el que fue diseñado.
Los métodos de validación se pueden clasificar en dos grandes grupos:

- Validación mediante pruebas
- Validación mediante Verificación o Validación formal

La **Validación mediante pruebas** consiste en ejecutar el programa con lotes de prueba para comprobar si los resultados obtenidos coinciden con los esperados.
Los resultados anómalos deberán ser analizados para detectar su causa, la que se eliminará en la etapa de depuración del programa. El procedimiento prueba – depuración, llamado también testing, se repite hasta que no se detectan errores y se tenga confianza que el programa fue lo suficientemente probado. El problema de este tipo de validación es determinar si el conjunto (juego) de pruebas es lo suficientemente confiable. Lo ideal sería contar con el *caso de pruebas perfecto* que permita asegurar que el programa se ejecutará correctamente para cualquier entrada posible.
Algunos autores denominan a este tipo de Validación "*a posteriori*". Esto quiere decir que, hasta que no se prueba el programa no aparecen los errores.
Una prueba de un programa con juegos de ensayos exhaustivos - testing - ayuda a detectar errores, pero no puede concluir que el programa es totalmente correcto ya que sólo usa una parte de todas sus posibles entradas.

La **Validación por Verificación** consiste en demostrar formalmente que el programa es correcto, *sin ejecutarlo*. Caracteriza todas las ejecuciones. Se la conoce como verificación "a priori".
La verificación de programas es un área importante en el área de investigación de la informática teórica. Para llevar cabo la verificación y localizar las partes incorrectas de un programa se dispone tanto de **técnicas de verificación formal** aplicables manualmente como así también de la ayuda de herramientas automáticas.

5.2 Aprender a diseñar programas correctos

Una manera de programar no muy generalizada, consiste en desarrollar los programas en forma metódica a partir de especificaciones. La **lógica matemática** es la herramienta que se utiliza para la especificación y desarrollo de programas. La idea subyacente es deducir los programas a partir de una **especificación formal del problema,** entendiendo esto como un predicado sobre un universo de valores**.** Los programas serán fórmulas lógicas cuya demostración permitirá asegurar que es correcto respecto de la especificación dada**.** Los seres humanos se comunican en lenguaje natural y la primera aproximación a la descripción de cualquier problema algorítmico será siempre mejor entendida inicialmente en lenguaje natural. Sin embargo al utilizar lenguaje natural las descripciones tienden a contener ambigüedades, ser inconsistentes o ser incompletas. Es por ello que, una vez comprendida *una descripción de problema* mediante lenguaje natural, esta descripción se modela en un *lenguaje formal matemático*, lo que permitirá redondear la especificación, puliendo detalles que podrían haber quedado opacos en la descripción original.

Es importante distinguir entre verificar y derivar programas. En ambos casos los programas serán fórmulas lógicas.

- **Verificar** consiste en *demostrar que el programa construido es correcto* respecto de la especificación dada**.**

  Hoare realiza un trabajo pionero en el que establece axiomas y reglas de inferencia (utilizando lógica de primer orden, llamada también lógica de predicados) para demostrar formalmente que un programa satisface una especificación. De esta manera **es posible determinar que un programa es correcto con una certeza dada por una demostración matemática. "Este sistema de demostración permite hacer demostraciones de la corrección del programa, pero no ayuda a construirlo."**

- **Derivar o deducir** un programa permite *construir un programa a partir de su especificación*, de forma que se obtiene un algoritmo correcto por construcción, es decir el algoritmo se diseña y se verifica a la vez.

  Dijkstra, sienta las bases para la aplicación de la **lógica de Hoare** al desarrollo sistemático de programas. A través de estos métodos no sólo se construye una demostración de la corrección del programa sino que la misma demostración que se está construyendo sirve para elaborar el programa.

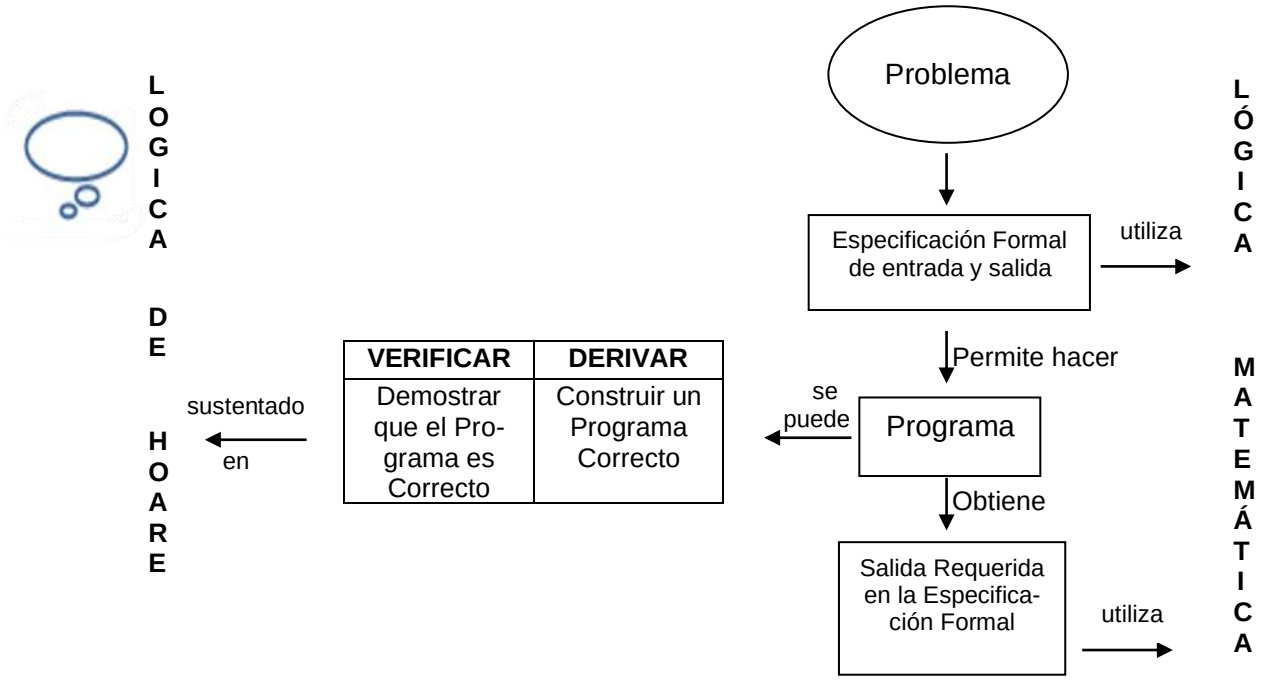


Figura 5.2 Esquema procesos de verificación/ derivación

5.3 Verificación formal de programas

La verificación formal de programas consiste en un conjunto de **técnicas de comprobación formales** que permiten demostrar si un **programa funciona correctamente**.
Esta afirmación requiere el análisis de algunos conceptos:

Técnicas de comprobación formales: La verificación consiste en un proceso de inferencia. Cada tipo de sentencia que constituye un programa, posee una regla de inferencia, las que serán analizadas más adelante.
La programación Imperativa se caracteriza por que la computación se expresa como modificación de estados. **Un programa** es una secuencia de sentencias que transforman el estado inicial en un estado final. Esto es, comienza su ejecución en un estado inicial válido, descrito por las variables de entrada y termina en un estado final en el que las variables de salida deben contener los resultados esperados.

Figura 5.3 Esquema de programa como transición de estados

Un estado está dado por el conjunto de valores que toman en un determinado momento las variables que conforman la estructura de datos del problema. En particular se puede hablar de:

El **estado inicial** es el estado anterior a la ejecución de un código o parte de un código.

El **estado final** es el estado posterior a la ejecución de dicho código.

El **estado intermedio** viene determinado por los valores de las variables en cada momento.

Ejemplo 1: dado el código

```
entero x,y
x=0
y=5
y= 2*x-1
Escribir x,y
```

El estado inicial de las variables x e y antes de ejecutar la acción y= 2*x-1, es x=0, y=5.
El estado final de las variables después de ejecutar la acción es x=0, y=-1

Las **aserciones o asertos** son expresiones lógicas que hacen referencia a un estado del programa y que se intercalan entre 2 instrucciones. Éstas son llamadas también **Fórmulas lógicas** o **Predicados** en la lógica proposicional. El aserto representa el conjunto de estados que se cumplen al llegar la ejecución del programa al punto en que se sitúa el aserto.
Una **aserción** es un predicado que se cumple (es verdadero) en determinado punto del programa.
Para indicar que una expresión es un aserto se encierra entre llaves, por ejemplo {P}.

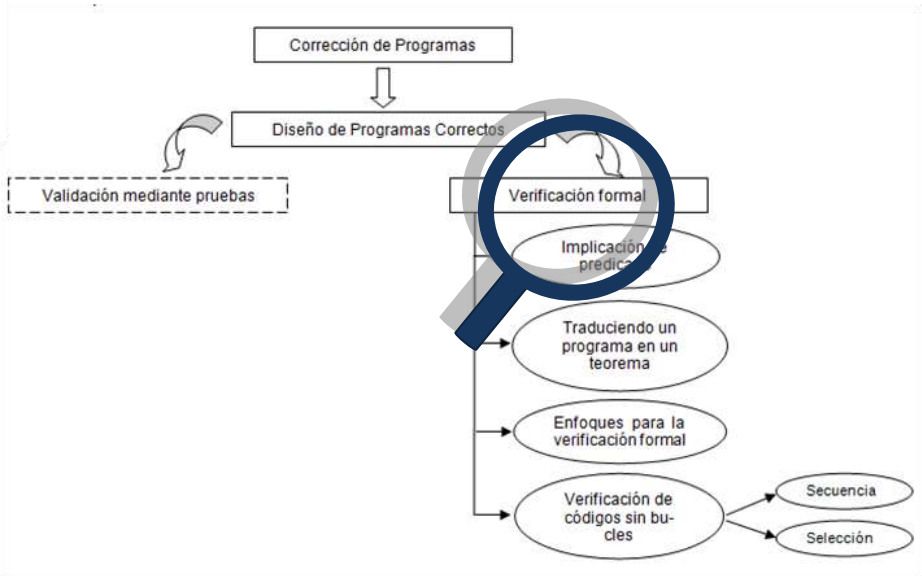
Se dice que un **Programa funciona correctamente** si cumple con especificaciones dadas. Para **especificar** un programa se utilizan los asertos que se cumplen al inicio y final del mismo, conocidos como precondición y postcondición respectivamente.
El **objetivo de toda especificación de algoritmos**, es expresar de forma correcta y sin ambigüedades qué es lo que debe hacer un algoritmo y bajo qué condiciones se puede ejecutar. Por ello se recurre a la lógica proposicional como lenguaje de especificaciones dado que es formal y cuenta con la deducción como mecanismo de análisis de propiedades.

Ejemplo 2:
Escribir un algoritmo que realice la división de un entero **a** por otro entero **b** distinto de cero, devolviendo el cociente entero en **q** y el resto en **r**.
En este ejemplo se realiza la especificación *no formal* de lo que se quiere obtener a partir del algoritmo así como y cuáles son las condiciones iniciales de los datos para su ejecución. Más adelante veremos cómo se realiza la especificación formal de estas condiciones.

El objetivo de la verificación formal es establecer asertos sobre lo que se pretende que realice el programa, basándose en las especificaciones y requerimientos, y demostrar, mediante argumentos razonados, que un determinado diseño cumple los asertos preestablecidos en lugar de esperar la ejecución del programa.
Para escribir especificaciones claras y precisas se utiliza un lenguaje formal que tenga una sintaxis y semántica perfectamente definidas, dado que el lenguaje natural suele ser impreciso o ambiguo.

¿Qué elementos intervienen en una especificación formal?

- Un espacio de estados
- Precondición
- Postcondición



Espacio de Estados

Como sabemos, la construcción de un algoritmo requiere declarar las variables con su tipo. Se llama ***Espacio de Estados*** **al universo de valores que pueden tomar las variables**.
Durante el desarrollo de un programa, es importante determinar qué condiciones deben darse al comienzo o entrada de la tarea y cuáles se espera deben ser ciertas al finalizar la misma, es decir a la salida.
Los asertos que deben cumplirse a la entrada de una tarea reciben el nombre de **Precondiciones.** Si la operación, tarea o instrucción se realiza sin que la precondición se cumpla no tendremos garantía de los resultados obtenidos.
Del mismo modo los asertos acerca de los resultados que se esperan de la salida, se llaman **postcondiciones.**

- Las **precondiciones** indican las condiciones que deben satisfacer los datos de entrada para que el programa pueda cumplir su tarea.
- Las **postcondiciones** indican las condiciones de salida que son aceptables como soluciones correctas del problema en cuestión.

Como se dijo, tanto la precondición como la postcondición se especifican mediante predicados lógicos.

Un **Predicado Lógico** es una fórmula cuya evaluación sólo puede dar dos resultados: Verdadero o Falso.

Para escribir las pre y post condición se utiliza la lógica de primer orden, llamada también lógica de predicados.

En este momento es necesario recordar algunos conceptos.

Proposición es un enunciado declarativo o afirmación del cual puede decirse si es verdadero o falso**.**
Son ejemplos de proposiciones: Julia es alta; 4 > 2, dado que se puede indicar si son verdaderas o falsas.
Afirmaciones tales como: x es alta, y > 2 no son proposiciones dado que no se puede indicar su valor de verdad, se conocen con el nombre de *predicados lógicos*.
Si se asignan valores concretos a las variables x e y, llamadas **variables proposicionales**, se transforman en proposiciones ya que se podrá indicar si son verdaderas o falsas.

Un **Predicado Lógico** es una afirmación que expresa una propiedad de variables. *Estas afirmaciones serán verdaderas o falsas cuando se reemplazan las variables por valores específicos*. Se los conoce también como fórmulas lógicas y son utilizados en las sentencias de control de los lenguajes de programación.

Las variables x e y de los predicados ejemplificados se conocen como **variables libres**. Si se les asigna valores, transformándose en una proposición, se conocen como **variable ligadas**.

Se dice que un **estado** satisface un predicado, cuando éste es verdadero para los valores de las variables de ese estado.

La especificación pre/post se basa en contemplar un algoritmo *S* como una caja negra. S actúa como una función de estados en estados, comienza su ejecución en un estado inicial válido, descrito por el valor de las variables de entrada, y termina en un estado final en el que los variables de salida contienen los resultados esperados.
Es por ello que se define un programa como una secuencia de sentencias que transforman el estado inicial en un estado final.

Ejemplo 3

Determinar las condiciones iniciales que deben cumplir las variables que intervienen en la sentencia x=2*y+1, de tal manera que al ejecutarse se verifique que x≤7 (Postcondición).

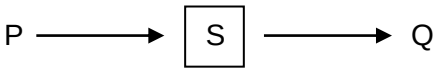
Para que se cumpla la postcondición x<=7:
Reemplazando x en la expresión resulta: 2*y+1<=7 , se obtiene y<=3.

De esto se infiere que para que se cumpla la postcondición {x<=7}, {y<=3} debe ser verdadero, por lo tanto es la precondición buscada. Obteniéndose así las aserciones requeridas.

Podemos simbolizar el proceso realizado de la siguiente manera:

{ y<=3} x=2*y+1 {x<=7}

{P} **S** **{Q}**



Esto se interpreta de la siguiente manera: Si la ejecución de la sentencia S, x=2*y+1, empieza en un estado inicial caracterizado por la precondición P {y<=3} entonces terminará en un estado final caracterizado por la postcondición Q {x<=7}.
Las sentencias de S son las que modifican los estados.

Ejemplo 4: Dado un programa que calcula la raíz cuadrada.
•Precondición: el argumento de la función es un número real no negativo
•Postcondición: La raíz cuadrada de ese argumento.

Ejemplo 5: Para un programa que realice la división de dos números enteros.
•Precondición: a y b números enteros y b distinto de cero.
•Postcondición: Cociente q entre a y b, r el resto de la división entre a y b y r<b.

Una **especificación tiene dos destinatarios**, el usuario que vaya a utilizar el algoritmo y el programador.
En la **precondición** se especifican las restricciones que tienen que cumplir las variables de entrada para que el algoritmo produzca el resultado deseado. Es donde se expresan las obligaciones que debe cumplir el **usuario**.

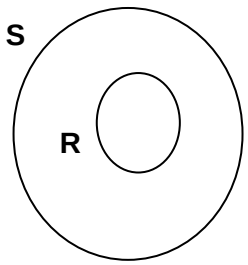
En la **postcondición** se especifica la relación que deben cumplir las variables de entrada y las de salida para que el algoritmo produzca el resultado deseado. Es donde se expresan las obligaciones del **programador**.

Bajo el esquema pre/post-condición para **especificación de problemas algorítmicos**, el comportamiento deseado de un programa imperativo a ser construido es descrito en términos de una precondición, que describe las propiedades requeridas a los datos de entrada del programa, y una postcondición, que describe las propiedades que se desea sean satisfechas por los resultados de salida del programa.

En el ejemplo 5, el usuario se tiene que asegurar que b sea distinto de cero. El programador desarrollará el algoritmo suponiendo que el usuario ingresa los valores adecuados de las variables.
En la mayoría de los códigos el estado final depende del estado inicial. Por lo tanto Precondición y Postcondición no son independientes entre sí.

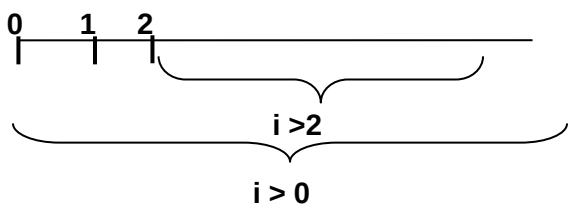
5.4 Implicación de predicado

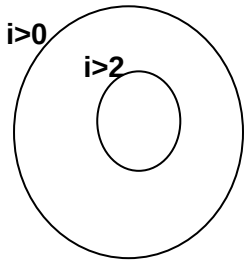
Predicado más fuerte y Predicado más débil
Un predicado R implica un predicado S, y se simboliza R⇒S , si el conjunto de estados para los que R es verdadero es subconjunto del conjunto de estados para los que S es verdadero. En este caso se dice que el R es más fuerte (más restrictivo) que S. Esto significa que siempre que se cumpla R también se cumple S. Todos los estados que satisfacen R también satisfacen S, pero pueden existir estados que satisfacen S y no R.
El siguiente diagrama de Venn grafica la relación de los predicados R y S



Ejemplo 6:
{ i>2 } es más fuerte que {i >0 } ya que todo valor de la variable i (estado) que hace verdadero el predicado {i >2}, también hace verdadero al predicado { i>0}, sin embargo no se cumple la relación inversa.
Por tanto {i>2} ⇒{ i >0}.

Gráficamente, la relación entre los predicados es la siguiente:

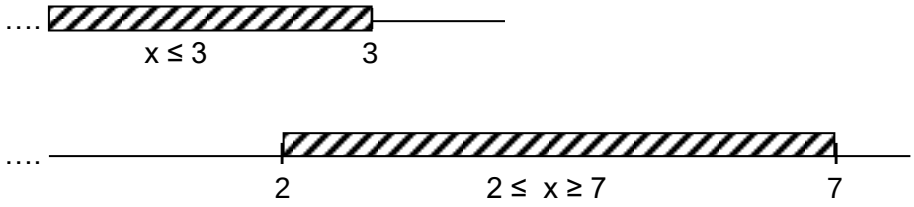




Ejemplo 7: { x ≥ 2 } ⇒ {x ≥ 1 }
{x ≥ 2} es más fuerte que {x ≥ 1} ya que todo valor de la variable x que satisface el predicado {x ≥ 2} también hace verdadero el predicado {x ≥ 1}. Sin embargo no todos los valores que verifican el estado {x ≥ 1} también verifican el estado {x ≥ 2 }.

Ejemplo 8: { 2 ≤ x ≥ 7 } ⇒ {x ≤ 3 }

Gráficamente se observa que:



De las graficas se deduce que {2 ≤ x ≥ 7} se intercepta con {x ≤ 3} pero no está contenido en el. Por lo tanto la implicación es falsa

En síntesis:
Si una aserción R es más fuerte que una aserción S entonces todos los estados que satisfacen R también satisfacen S pero no viceversa.

Más fuerte $\Longrightarrow$ más selectivo, más específico Más débil $\Longleftarrow$ menos selectivo, más general. Si R es más fuerte que S R $\Longrightarrow$ S Si S es más débil que R S $\Longleftarrow$ R

- La aserción más débil es verdadera para todos los estados posibles, es conocida como precondición vacía y se simboliza { } , corresponde a la constante lógica V (true).
  Ejemplo ($\forall x : x \in N : x > 0$).

- La aserción más fuerte será por tanto F (false), ya que representa que ningún estado satisface dicha condición.
  Ejemplo ($\forall x < 0 : \exists i \in N : i = \text{sqrt}(x)$).

Ejemplo 9: Predicados más débil y fuerte
Si x e y son variables enteras $\in$ ZxZ, establecer "más débil" ($\Longleftarrow$) o "más fuerte" ($\Rightarrow$) entre los siguientes predicados.
{ P_1 : x >0}
{P_2 : (x >0) $\wedge$ (y >0) }
{P_3 : (x >0) $\vee$ (y >0) }
{P_4 : y $\geq$0 }
{P_5 : (x $\geq$0) $\wedge$ (y $\geq$0) }

- $P_2 \Rightarrow P_1$ dado que si bien los dos predicados exigen que la variable x sea estrictamente positiva, mientras P_1 admite cualquier valor de y, P_2 sólo admite valores positivos para ella. Por tanto cualquier asignación de valores a las variables x e y (cualquier estado) que haga cierto a P_2 también hace cierto a P_1, pero no al revés.

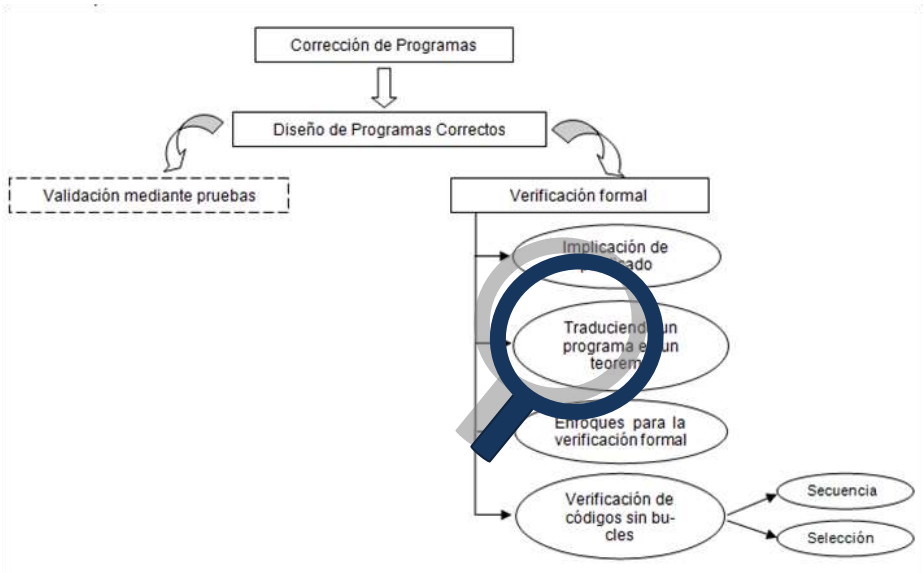
En general, una conjunción es más fuerte que cada una de sus partes.
$P \wedge Q \Rightarrow P$ cualesquiera sean los predicados P y Q

En general, una disyunción es más débil que cada una de sus partes.
$P \Rightarrow P \vee Q$ cualesquiera sean los predicados P y Q

- $P_1 \Rightarrow P_3$ por que la disyunción debilita un predicado.
- Entre P_1 y P_4 no hay relación ya que se refieren a variables disjuntas, esto es existen estados que cumplen P_1 y no P_4 (x==5 , y== -1) y estados que cumplen P_4 y no P_1 (x==0 , y== 0)
- Entre P_1 y P_5 tampoco hay relación, ya que si bien los dos predicados hacen referencia a la misma variable x , el estado que asigna los valores (x==5 , y==-5) satisface P_1 pero no P_5, y el estado que asigna los valores (x==0 , y== 5) satisface P_5 pero no P_1.

Actividad 1:
Determinar la relación entre:

- P_2 y P_3
- P_2 y P_4
- P_2 y P_5
- P_3 y P_5
- P_4 y P_5



5.5 Traduciendo un programa en un teorema

Aplicación de la lógica de Hoare a la verificación formal de Programas

Para demostrar formalmente que un programa es correcto, es necesario representarlo como una fórmula de alguna lógica.
El programa es visto como un conjunto de axiomas y reglas de inferencias**.** Para la verificación de programas imperativos se utiliza un sistema de reglas basado en la **tripla de Hoare,** que permite especificar los algoritmos basándose en lógica de predicados de primer orden. La lógica de predicados es la versión /parte /rama de la lógica que trata con proposiciones cuantificadas.

Un problema a resolver estará especificado por un **espacio de estados** (universo de valores que pueden tomar las variables), una **precondición,** esto es el conjunto de estados válidos al comienzo del programa que resuelve el problema y una **postcondición**, predicado que describe el estado de las variables al finalizar el programa.

Las especificaciones de un programa, **precondiciones y postcondiciones,** en la lógica de Hoare, son fórmulas denominadas **aserciones o asertos**.

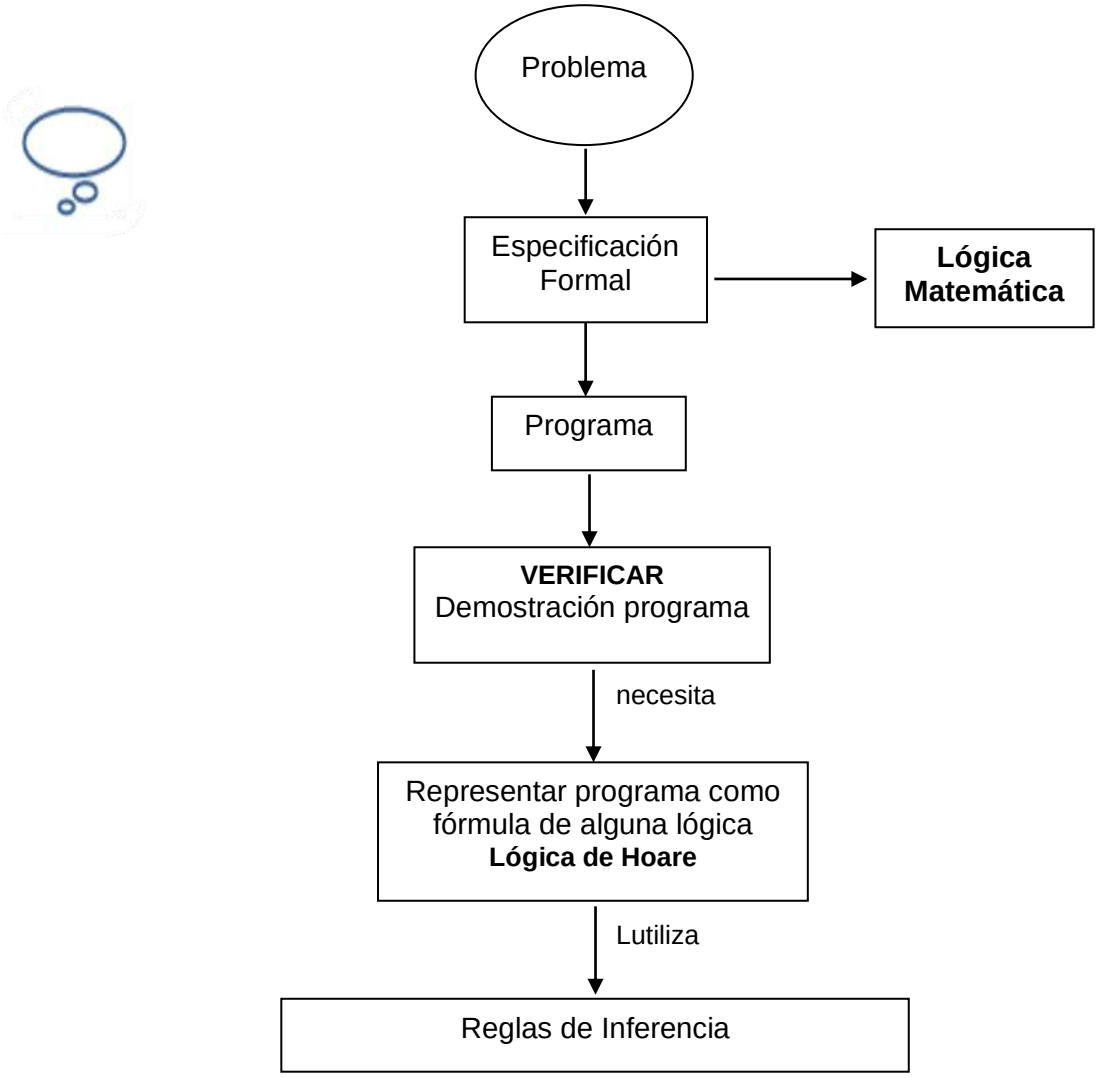


Figura 5.4 Esquema de Verificacion de Programa

La **especificación formal** de un programa S que tiene una precondición {P} y una Postcondición {Q }, se realiza mediante la siguiente terna {P} S {Q}, siendo P y Q aserciones de la lógica.

Esta expresión se interpreta, si la precondición P es cierta antes de la ejecución del programa y dicho programa termina, entonces la postcondición Q es cierta tras la ejecución de dicho programa. Es decir que si se garantiza que la entrada actual satisface las restricciones de entrada (precondiciones) la salida satisface las restricciones de salida (postcondiciones).
El programa S actúa como una función de estados en estados, comienza su ejecución en un estado inicial válido, descrito por el valor de los parámetros de entrada, y termina en un estado final en el que los parámetros de salida contienen los resultados esperados.

Como se observa la lógica de Hoare utiliza dos lenguajes formales, un lenguaje de programación imperativo para el programa S y un lenguaje lógico para los predicados.

Corrección total y parcial

Corrección parcial: se dice que {P} S {Q} es parcialmente correcto si el estado final de S, *cuando termina el programa* (aunque no se le exige esta premisa), satisface {Q} siempre que el estado inicial satisface {P}.
Es decir, si el algoritmo terminara (lo que no asegura esta corrección) se cumpliría con las especificaciones dadas para S.
Corrección total: Se da cuando un código además de ser correcto parcialmente, termina.

Los códigos sin bucles siempre terminan, por lo cual en este caso la corrección parcial implica la corrección total. Esta distinción es esencial sólo en el caso de códigos que incluyan bucles o recursiones.

Reglas de Inferencia o de Verificación

El sistema formal de Hoare es un sistema de reglas de inferencia que permiten razonar sobre las instrucciones de un algoritmo, ya sea para calcularlas o para verificar su corrección.

Existen reglas que son aplicables a cualquier sentencia, llamadas *reglas básicas*, pero además cada tipo de sentencia posee una regla de inferencia, llamada *regla específica*. Por tanto tendremos una regla de inferencia para la asignación, otra para las sentencias condicionales y otra para las sentencias iterativas.

Las reglas *de verificación* se representan formalmente con ternas de Hoare y tienen la forma:

Premisas

Conclusión

Nota: Para la resolucion de problemas de verifiacion de esta unidad se utiliza la Logica de Predicados y la Logica Proposicional. (Ver Anexo VI)

5.6 Enfoques para la verificación Formal

Existen dos enfoques de verificación: Backward y Forward, siendo el primero el más usado.La verificación Backward consiste en **proceder de atrás hacia delante**.

Conceptualmente todo el proceso de verificación se basa en la definición de la precondición más débil: "**pmd"**, la **menos restrictiva.** Esto es, vamos a intentar definir cuáles son los estados tales que desde ellos y ejecutando el programa **S** se llega a **Q**. Estos estados se pueden caracterizar por una serie de condiciones, o sea, por una aserción **pmd** que cumplirá:

{**pmd**} **S** {Q}

Esto se lee “Precondición más débil de **S** respecto de **Q”.**

Por tanto si la precondición {P} es más fuerte que la pmd, se concluye que el algoritmo es correcto para las especificaciones P y Q.

Como regla general:

1. Se debe encontrar la precondición más débil {pmd} tal que {**pmd**} **S** {**Q**} es válido
2. Cualquier precondición {P} que implique {pmd} hace que el programa S sea correcto.

Esto es:

Si {P} $\Rightarrow$ {**pmd**} entonces el programa S es correcto para la terna {**P**} **S** {**Q**}.

La precondición más débil se conoce también como **Transformador de Predicados,** ya que para cualquier fragmento de programa define una transformación de un predicado Postcondición en un predicado Precondición. Es decir que en vez de describir cómo un programa transforma un conjunto de estados iniciales en un conjunto de estados finales, describe cómo un programa transforma un predicado postcondición, que caracteriza el conjunto de estados finales, en un predicado Precondición que caracteriza el conjunto de estados iniciales. Esto significa que para probar formalmente un programa se trabaja “ hacia atrás”, partiendo de la postcondición.

5.7 Verificación de códigos sin bucles

Para demostrar la corrección de las partes de un programa se parte de la postcondición del código, es decir de las condiciones que deben satisfacer los resultados. A partir de ahí se deduce la precondición.
Recordemos que el código se verificará en sentido contrario a como se ejecuta, teniendo en cuenta que la postcondición de código 1 se transforma en precondición de código 2.

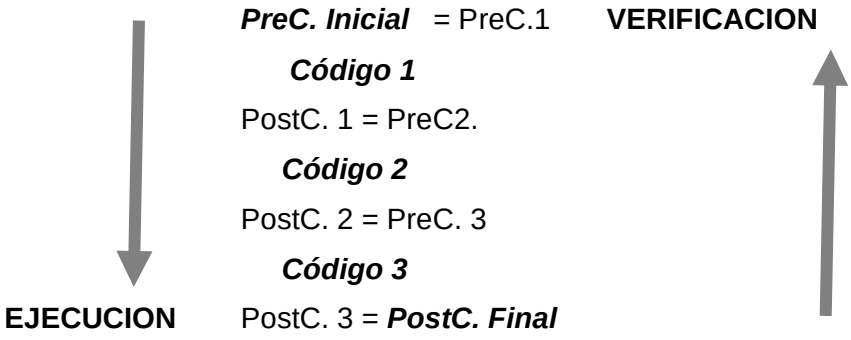


Figura 4

Reglas específicas para instrucciones del lenguaje

Se describen a continuación las reglas de verificación para las acciones en seudocódigo imperativo, que usamos para escribir los algoritmos. Se determinará además de la regla para cada instrucción, la forma de encontrar su precondición más débil con respecto a una postcondición dada.

Sentencia de asignación

Las sentencias de asignación son sentencias de la forma V = E, en donde V es una variable y E es una expresión.

$$\{\ \textbf{pmd}\ \}\quad V=E\quad \{\ Q\ \}$$

Regla de la asignación: Si S es una sentencia de la forma V=E con la postcondición {Q}, entonces la precondición más débil de S puede hallarse sustituyendo en Q todos los casos de V por E. Esto se simboliza $(Q)^{E}_{V}$

Simbólicamente lo podemos expresar:

$$\{\textbf{pmd}\}\ V=E\ \ \{Q\}\ ;\ \{\ \textbf{pmd}\ \} \equiv (Q)^{E}_{V} \Rightarrow \{(Q)^{E}_{V}\}\ V=E\ \ \{Q\}$$

$$\{pmd\} \equiv (Q)^{E}_{V}$$

Esquemáticamente:

$$\{pmd\} \equiv (Q)^{E}_{V}$$
$$\downarrow$$
$$V=E$$
$$\downarrow$$
$$\{Q\}$$

Los siguientes son ejemplos que indican cómo encontrar la precondición más débil para asignaciones dadas:

Ejemplo 10: Determinar la precondición **más débil** para que la terna siguiente sea correcta

{pmd} x=2*y+1 {x ≤15} , x, y ∈ Z

$pmd\,(Q)^{E}_{V} \Rightarrow (x \leq 15)\,{}^{2*y+1}_{x} \Rightarrow (2*y+1 \leq 15) \Rightarrow (y \leq 7)$

Esto se interpreta que para cualquier valor inicial de la variable y menor o igual a 7, la sentencia x=2*y+1 es correcto ya que cumple con la especificación de salida {x ≤15} .

Ejemplo 11: Determinar la precondición **más débil** para que la terna siguiente sea correcta:

{pmd} $i_f = 2*i_i$ {i_f <6} , i_i , i_f ∈ Z

denotaremos con $\mathbf{i_i}$ al valor de i antes de ejecutar el código y con $\mathbf{i_f}$ al valor de i después de ejecutar el código .

{pmd} ≡ $(Q)\mathbf{v}^{E} \equiv (\mathbf{i_f<6})\,{}^{2*i}_{if} \equiv (2*i_i < 6) \equiv (i_i<3)$

⇒ {pmd: i_i<3}

Esto se interpreta que la variable i tomará un valor menor que 6 después de ejecutar la instrucción i= 2*i si inicialmente (antes de la ejecución de la sentencia) toma un valor menor que 3.

Actividad 2: Indicar cuales precondiciones hacen verdadero el algoritmo del ejemplo anterior. Justificar.

1. { i_i > 0} i=2*i { $\mathbf{i_f}$ <6}
2. { i_i <2} i=2*i { $\mathbf{i_f}$ <6}
3. { i_i ==5} i=2*i { $\mathbf{i_f}$ <6}

Ejemplo 12: Determinar la precondición más débil para que la terna siguiente sea correcta
{pmd} j=i+1 {j>0} , i,j ∈ Z

La precondición más débil se obtiene sustituyendo en la postcondición j>0, la variable j por i+1.Lo expresado se formaliza:

{pmd}≡ $\{Q\}\,\mathbf{v}^{E} \equiv (j > 0)_{j}{}^{i+1} \equiv (i+1) > 0 \equiv (i > -1)$

{pmd: i >-1 }

Esto se interpreta: para que la variable j sea mayor que 0 después de ejecutarse la sentencia j=i+1, inicialmente la variable i debe ser mayor que -1.

Con este resultado indicar en qué casos el algoritmo es correcto:

1. { i> 0} j=i+1 {j>0}
2. { i<0 } j=i+1 {j>0}
3. { i> -2} j=i+1 {j>0}
4. { i<-3} j=i+1 {j>0}

Ejemplo 13: Determinar la precondición más débil para que la terna siguiente sea correcta:
{pmd} $y = x^2$ {y>1} x,y ∈ Z

{pmd}≡ {Q} $\mathbf{v^E}$ ≡ $\left(y>1\right)_y^{x2}$ ≡ (x^2>1) ≡ (x >1) V (x < -1

Actividad 3: Indicar dos precondiciones para las cuales el algoritmo del ejemplo anterior es válido y dos para las que no lo sean.

Ejemplo 14: Determinar precondición más débil para que la terna siguiente sea correcta:

{pmd} x=1/x {x>0} x ∈ Z

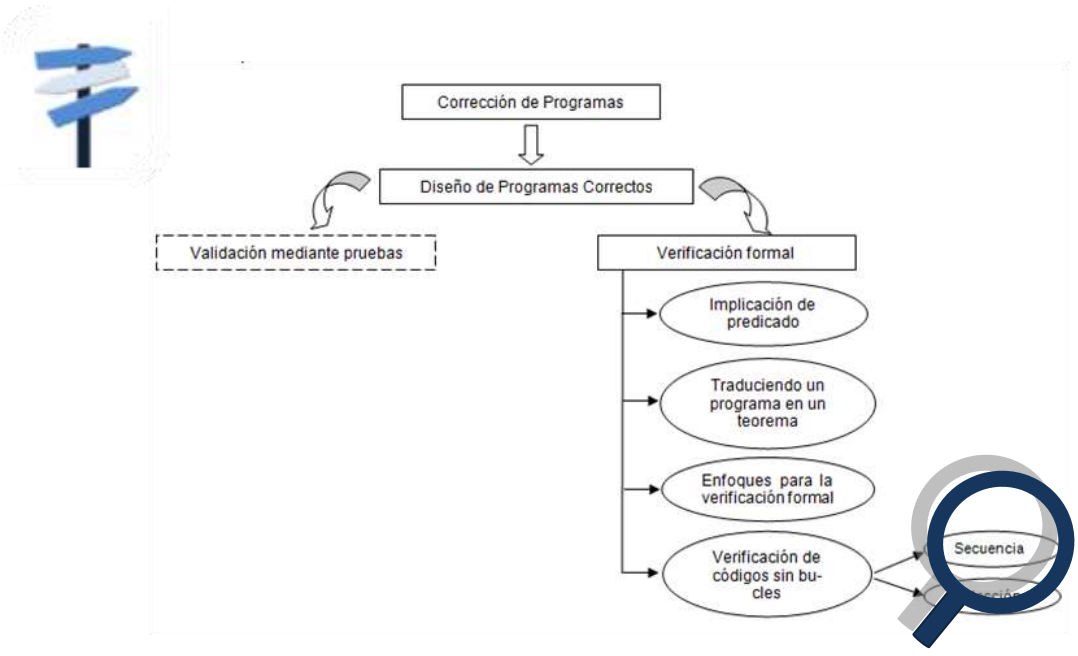
{pmd} ≡ {Q}$\mathbf{v^E}$ ≡ $(x>0)_x^{1/x}$ ⇒ (1/ x > 0) ≡ (x >0)

Actividad 4: Indicar una precondición para la que el algoritmo del ejemplo anterior es válido y otra para las que no lo es.

Actividad 5: Sea x una variable entera, indicar si el algoritmo es válido para la especificación dadas { $x_i \geq -2$ } x=x+2 {$x_f \geq 0$}. x∈ Z Justifique.

Actividad 6: Sea x una variable entera, indicar si el algoritmo es válido para las especificaciones dadas. Justifique.

1. { $x_i \leq 4$ } x=3*x { $x_f \leq 21$}
2. { $x_i > 4$ } x=3*x { $x_f \leq 21$}
3. { $x_i == 7$ } x=3*x { $x_f \leq 21$}



Sentencia compuesta: Concatenación de Código

La concatenación significa que las partes del programa se ejecutan secuencialmente de tal forma que el estado final de la primera parte de un código se convierte en el estado inicial de la segunda parte del programa.

Esquemáticamente:

{P}

C1

C2

{Q}

Regla de la concatenación

Sean C1y C2 dos partes de un código y sea C1; C2 su concatenación. Si {P} C1 {R} y {R} C2 {Q} son ternas de Hoare correctas entonces se puede afirmar que:

$$\frac{\{P\}\ C1\ \{R\} \qquad \{R\}\ C2\ \{Q\}}{\{P\}\ C1; C2\ \{Q\}}$$

Esto significa que para demostrar que la terna {P} C1; C2 {Q} es correcta, se debe encontrar el predicado intermedio {R} que es postcondición de C1 y precondición de C2

Ejemplo 15:

a) Hallar la precondición más débil para el siguiente algoritmo

{pmd} x= x*5 ; x = x-3 { x ≤ 22}, x Є Z

b) determinar si el algoritmo es correcto para cada una de las siguientes precondiciones, justificar la respuesta.

- {P1: x< 0}
- {P2: x ≥ -1}

En este caso se debe encontrar el predicado intermedio **{R}** tal que se verifique:

{pmd} x= x*5 ; **{R}** x = x-3 { x ≤ 22 }

{R} x = x-3 { x ≤ 22 }

$\{R\} \equiv (x \leq 22\ )_x^{x-3} \equiv (x-3 \leq 22\ ) \equiv (x \leq 25)$

{R: (x ≤ 25) }

Encontrar {pmd} x= x*5 {x ≤ 25}

$\{pmd\} \equiv (x \leq 25\ )_x^{x*5} \equiv (x*5 \leq 25\ ) \equiv x \leq 5$

{pmd: x ≤ 5 }

Esto se interpreta así: siempre que los valores iniciales de la variable x cumplan la precondición {x ≤ 5 } y se ejecute el algoritmo se verificará la postcondición { x ≤ 22}

- el algoritmo es correcto para {P1: x< 0} dado que {x< 0} ⇒ { x ≤ 5 }
- el algoritmo no es correcto para {P2: x ≥ -1} dado que {x ≥ -1} no implica { x ≤ 5 }. Si bien existen valores ,como x=1, que hacen válido el algoritmo; para otros como x=7 no lo es.

Nota

Verificar un código requiere realizar dos pasos especificos:

1) Calcular la pmd: $\{\textbf{pmd}\} \equiv (Q)^{E}_{V}$

2) Analizar la implicación: $P \Rightarrow$ **pmd**

Si se cumple limplicacion significa que la accion esta verificada.

Simplificando las acciones

Este cálculo también puede expresarse de manera simplificada:

{pmd} x= x*5 ; x = x-3 { x ≤ 22 , x Є Z}

$\{pmd\} \equiv (((x \leq 22)_{x}{}^{x-3})_{x}{}^{x*5}) \equiv ((x-3 \leq 22)_{x}{}^{x*5}) \equiv (x \leq 25)_{x}{}^{x*5} \equiv x*5 \leq 25) \equiv (x \leq 5)$

{pmd: x ≤ 5 }

Ejemplo 16: Indique una precondición para la cual el siguiente algoritmo sea válido y otra para la que no lo sea. En ambos casos justifique la respuesta.

{P: ?} x= x+1 ; y = y+2 { x <y } x,y Є Z

Para encontrar las precondiciones solicitadas es necesario encontrar la pmd que verifique:

{pmd} x= x+1 ; **{ R}** y = y+2 { x <y } (1)

El predicado intermedio {R} se obtiene:

$\{R\}$ y = y+2 { x <y } $\equiv$ (x <y)$_{y}{}^{y+2} \equiv$ (x <y+2)

Reemplazando en (1) {pmd} x = x+1 (x < y+2)

$\{pmd\} \equiv (x <y+2)_{x}{}^{x+1} \equiv$ x +1 < y+2 $\equiv$ x < y+1

{pmd: x <y+1}

Simplificando los pasos, esto es equivalente a realizar:

$\{pmd\} \equiv ((x <y)_{y}{}^{y+2})_{x}{}^{x+1} \equiv ((x <y+2))_{x}{}^{x+1} \equiv x +1<y+2 \equiv x <y+1$

{pmd: x <y+1}

La siguientes son precondiciones válidas dado que son más fuertes que la pmd.

{P1: x==4 ∧ y==7 }

{P2: x==0 ∧ y>5 }

{P3: x==y }

- Verificamos la precondición P1.
  {P1: x==4 ∧ y==7 } x= x+1 ; y = y+2 { x <y }

  Partiendo de los valores iniciales x==4 ∧ y==7 y ejecutando el código x=x+1 ; y=y+2 se obtiene que x==5, y==9, valores que cumplen la postcondición {x<y}

- Verificamos la precondición P2.
  {P2: x==0 ∧ y>5} x= x+1 ; y = y+2 { x<y}
  Partiendo de los valores iniciales x=0 ∧ y>5 y ejecutando el código x=x+1 ; y=y+2 se obtiene que x==1, y>7, valores que cumplen la postcondición { x <y }

- Verificamos la precondición P3.
  Partiendo de los valores iniciales x==y, al ejecutar el código x=x+1; y=y+2 se obtiene que reemplazando x por y: x+1 < x+2, lo que cumple la postcondición {x<y}

No hacen válido el código las siguientes precondiciones:

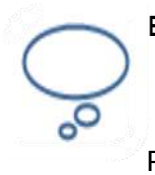
{P4: x > y+1 }

{P5: x== 5 ∧ y==0 }

Dado que no son precondiciones más fuertes que pmd.

- Verificamos la precondición P4.
  Partiendo de los valores iniciales x > y+1 y ejecutando el código x= x+1 ; y = y+2 se obtiene que x+1 > (y+1)+2 ≡ x > y+2 por lo cual no se cumple la postcondición {x<y}

- Verificamos la precondición P5.
  Partiendo de los valores iniciales x== 5 ∧ y==0 y ejecutando el código x= x+1 ; y = y+2 se obtiene que x== 6, y==2, valores que no cumplen la postcondición { x <y }



Ejemplo 17: Demostrar que el siguiente código es correcto
{ } c=a+b; c=c/2 {c==(a+b)/2}

Para esta demostración se debe:

a - Encontrar en primer lugar el predicado intermedio {R}, tal que:

{ } c=a+b; {R} c=c/2 {c==(a+b)/2}

{R} c=c/2 {c==(a+b)/2}

$\{R\} \equiv (c == \frac{a+b}{2})^{\frac{c}{2}}_{c} \equiv$ c/2==(a+b)/2

Por tanto {R} ≡ {c==(a+b)}

b - Encontrar pmd

{Pmd} c=a+b {c==a+b}

$\{Pmd\} \equiv (c==a+b)^{a+b}_{c}$ ≡ (a+b)=(a+b) dado que esta igualdad es válida para cualquier estado {Pmd }≡ { true}

c - Verificar que {P} implica la {Pmd}

{Pmd }≡ { } Esta es la aserción vacía que se puede entender como “verdadero para todos los estados”.

Dado que todo conjunto es subconjunto de sí mismo ,entonces:

{ } c=a+b {c/2==(a+b)/2} es correcto como queríamos probar.

Ejemplo 18: Demostrar que el siguiente código, que permite el intercambio de valores es correcto:

{a == X ∧ b == Y} aux=a; a=b; b=aux {a == Y ∧ b == X}

Calcular la pmd

$\{Pmd\} \equiv (((\ a == Y \wedge b == X)_{b}^{aux}\ )\ _{a}^{b}\ )\ _{aux}^{a}$

$\equiv (a == Y \wedge aux == X)_{a}^{b}\ )\ _{aux}^{a}$

$\equiv (\ b == Y \wedge aux == X)\ _{aux}^{a}$

{Pmd: b == Y ∧ a == X }

Dado que **{P}** ⇒ **{Pmd}**

siendo que {P**:** {a == X ∧ b == Y} es igual a {Pmd: b == Y ∧ a == X }

se pude decir que:

{a == X ∧ b == Y} aux=a; a=b; b=aux {a == Y ∧ b == X} es correcto

Ejemplo 19: ¿Podrías decir que realiza el siguiente algoritmo?
Sean x e y variables enteras
x= x-y
y= x+y
x= y-x

Para conocer el efecto de este segmento de algoritmo usamos la postcondición x==A ∧ y==B para indicar los valores de las variables x e y después de la ejecución de la acción.

Encontremos la precondición más débil:

pmd(x==A∧ y==B ; x= x-y , y= x+y, x= y-x)

$\equiv \left(\left((x==A \wedge y==B\ )_{x}^{y-x} \right)_{y}^{x+y} \right)_{x}^{x-y} \equiv \left(\left(y-x==A \wedge y==B \right)_{y}^{x+y} \right)_{x}^{x-y} \equiv (\ x+y-x==A \wedge x+y==B\ )_{x}^{x-y}$

$\equiv (y=A \wedge x+y=B)_{x}^{x-y} \equiv$ y=A ∧ x-y+y=B ≡ y=A ∧ x=B

Como se observa**,** el algoritmo permite el intercambio entre los valores de las variables x e y.

Actividad 7: Indicar cuáles precondiciones permiten que el algoritmo sea correcto y cuáles no. Justificar la respuesta.

x,y Є Z

{ P: ? }
y = x*2 -5;
x= y-2;
{x ≤ 23}

a. {P1: 0≤ x ≤ 20 }
b. {P2: -1≤ x ≤ 3 }
c. {P3: x ≥ 20 }

Actividad 8: Indicar una precondición que haga que el algoritmo sea correcto y otro que lo haga no válido. Justificar la respuesta

a,b Є R

{ P: ? }

a = a+2;

b = b / 3;

{b<a}

Actividad 9:

a) Hallar la precondición más débil para el siguiente algoritmo
{pmd} x= x/2 ; x = x+5 { x <= 12 ∧ x Є Z}

b) determinar si el algoritmo es correcto para cada una de las siguientes precondiciones, justificar la respuesta.

{ x< 10 x Є Z }
{x >= 6 x Є Z }

Actividad 10:

Indique una precondición para la cual el siguiente algoritmo sea válido y otra para la que no lo sea. En ambos casos justifique la respuesta.

{P: ?} x= x*3; x = x-1 { x <= 8 } x Є Z

Sentencia Skip

La instrucción Skip, " no hacer nada" no tiene efecto sobre el estado válido antes de su ejecución.

Entonces la {pmd} correspondiente a la terna {P} skip {Q} es {Q}

Pmd {P} skip {Q} ≡ {Q}

Por ejemplo para x entera es correcto el siguiente algoritmo:

{x ≥1}

skip

{x ≥0}

Dado que {x ≥1}⇒{x ≥0}

Sentencia de Selección

Sentencias de Selección con cláusula o rama sino

Si C1y C2 son dos partes de un programa y si B es una condición, entonces la acción

Si (B)
Entonces C1
Sino C2
Finsi

se interpreta de la siguiente forma: si B es verdadero se ejecuta C1, si es falso se ejecuta C2.

El siguiente es el diagrama que corresponde a esta sentencia:

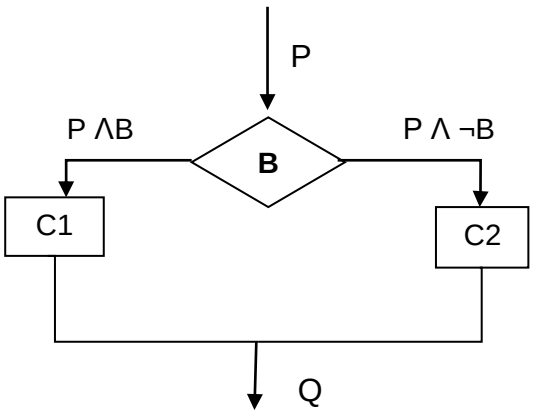


Figura 5.5 Seleccion

Para demostrar la corrección de una sentencia SI con una precondición {P} y una postcondición {Q} vamos a analizar dos casos

a) Se conoce la precondición {P} y la postcondición {Q}

b) Se debe encontrar la pmd a partir de {Q}

Caso a: Se conoce la precondición {P} y la postcondición {Q}

En este caso se debe considerar:

- Si el **estado inicial satisface B** además de P entonces se ejecutará C1 y por tanto la verificación equivaldrá a demostrar que

  {P Λ B} C1 {Q} es correcto.

- Si el **estado inicial no satisface B** entonces se ejecutará C2 y por tanto la verificación equivaldrá a demostrar que

  {P Λ¬B} C2 {Q} es correcto.

Esto es, se debe probar

{**P Λ B** } ⇒ pmd(Q, C1) Λ {**P Λ ¬ B**} ⇒ pmd(Q, C2)

La verificación de una sentencia condicional simple se apoya en la regla condicional, que establece que hay que verificar primero el camino del **si** asumiendo cierta la condición, y luego el camino de **sino**, asumiéndola falsa.

Ejemplo 20: Demostrar que el siguiente algoritmo, que calcula el máximo entre 2 números, es correcto para las especificaciones dadas.

```
{ }
  SI  (a>b)
        entonces m=a
         sino  m=b
    Finsi
{(m≥a) Λ (m ≥b)}
```

a. Primero debemos demostrar que:

{P Λ B} C1 {Q} es cierto,

esto es

{ } Λ { $a > b$} $m = a$ {($m \geq a$) Λ ($m \geq b$)} ≡ {$a > b$} $m = a$ {($m \geq a$) Λ ($m \geq b$)} es válido

Para ello se debe demostrar que {$a > b$} ⇒ {pmd: $((m \geq a) \wedge (m \geq b))_m^a$}

Encontremos la precondición pmd:

{pmd } ≡ ((m ≥a) Λ (m≥b)) $_m^a$ ≡ (a ≥a) Λ (a ≥ b)}

Como a ≥ a es verdadero entonces {pmd: (a ≥ b)}

Verificamos la implicación:

Dado que {(a >b)} ⇒ (a ≥ b), (es decir {P Λ B} implica la {pmd})

queda demostrado que

{} Λ { a >b} m=a {(m ≥ a) Λ (m ≥b)} es verdadero por lo tanto el código es correcto para la rama del entonces.

b. Ahora se debe demostrar:

{P Λ ¬B} C2 {Q} es cierto

esto es

{ } Λ ¬{a>b} m =b {(m ≥ a) Λ (m ≥ b)} ≡ {a ≤ b} m =b {(m ≥ a) Λ (m ≥ b)} es válido

Calculamos la pmd

{pmd : $((m \geq a) \wedge (m \geq b))_m^b$}

{pmd}≡ $((m \geq a) \wedge (m \geq b))_m^b \equiv$ { $(b \geq a) \wedge (b \geq b)$}

Como $b \geq b$ es verdadero ⇒ {pmd: ($b \geq a$)}

también se puede expresar como {pmd: ($a \leq b$)}

Verificamos la implicación

Dado que { ***a ≤ b***} ⇒ { ***a ≤ b***}

queda demostrado {$b \geq a$} $m = b$ {($m \geq a$) Λ ($m \geq b$)} es válido en la rama del sino.

c. Verificar la validez de ambas implicaciones

Dado que

{P Λ B } ⇒ pmd (Q,C1) (es decir en la rama del entonces es Verdadero)

y

{P Λ ¬ B} ⇒ pmd (Q,C2) (es decir en la rama del sino es Verdadero)

Se verifica que el algoritmos es correcto en ambas ramas de la Selección

Caso b: Se debe encontrar la pmd a partir de {Q}

{Pmd} Si (B) Entonces C1 Sino C2 Finsi {Q}

En este caso se deben encontrar las precondiciones {P1 } y {P2 } correspondientes a cada una de las ramas, tales que

{P1 Λ B} C1 {Q} es correcto y {P2 Λ¬B} C2 {Q} es correcto. Entonces la pmd es

{pmd } ≡ {P1 } v {P2 }

A partir de la postcondición Q se obtiene mediante un razonamiento deductivo las precondiciones P1 y P2, correspondientes a las sentencias C1 y C2 respectivamente, siendo {pmd} la disyunción de ambas.

Ejemplo 21: Sean x e y variable enteras, encontrar el predicado más débil que satisfaga la siguiente especificación:

```
{pmd}
  si (x ≥ 0)
        entonces y =x
        sino y = -x
  finsi
  {y ==4}
```

Se debe encontrar {pmd} ≡ {P1} $\vee$ {P2} , siendo:

{P1} la precondición de la rama (si ($x \geq 0$) ; $y = x$, y ==4) (rama del entonces)

y

{P2} la precondición de la rama (si (x < 0); $y = -x$, y==4) (rama del sino)

Entonces se debe probar que:

Paso 1: {P1 Λ B} C1 {Q} es correcto, verificando que

{P1} Λ (x ≥0) ⇒ pmd (y==4, y=x)

Paso 2: {P2 Λ¬B} C2 {Q} es correcto, verificando que

{P2} Λ (x <0) ⇒ pmd (y==4, y= -x ,)

Paso 3: Hallar la pmd de la Selección {P1 } v {P2 }

Resolucion de los pasos:

Paso 1:

Encontremos {pmd : $(y == 4)_y^x$ }

{pmd} ≡ $(y == 4)_y^x$ ≡ { *X*==4 }

Se debe despejar {P1} y verificar la implicación {P1} Λ (x ≥0) ⇒ pmd { *X*==4 }

Para que {P1} Λ (x ≥0) ⇒ { *X* == 4 } se deduce que **{P1: *X*==4 }**

Por lo tanto **si (x ≥ 0),** inicialmente x debe ser igual a 4 .

Paso 2

Encontremos { pmd $(y == 4)_y^{-x}$ }

{pmd } ≡ $(y == 4)_y^{-x}$ ≡ -*X* ==4 ≡ {*X* ==- 4}

Se debe despejar {P2} y verificar la implicación {P2} Λ (x <0) ⇒ pmd {*X* ==- 4}

Para que {P2} Λ (x <0) ⇒ { *X*==-4 } se deduce que **{P2: *X* == -4}**

Paso 3

Del Paso 1 y del Paso 2 se deduce que

Si x ≥0 **{P1: *x*==4 }**

y

Si x <0 **{P2: *x*==-4 }**

{pmd}≡ {P1} V {P2} ≡ { *x*==4 } V{ *x*==-4 } ≡ |x|=4

Como se observa para que y valga 4 después de la ejecución de esta instrucción, x debería valer 4 o -4 antes de su ejecución. La instrucción permite que y tome el valor absoluto de x.

Ejemplo 22: Demuestre si el siguiente algoritmo es correcto para las especificaciones dadas:

```
{ P: 4<x<10 }
  Si ( x>6 )
   entonces  x =x+1
    sino  x = x-1
   finsi
{ x !=6 }
```

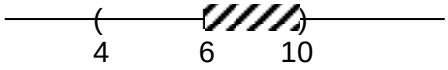
se debe demostrar que:

(1) {P: 4<x<10} Λ (x>6) ⇒ pmd (x!=6, x=x+1)

(2) {P: 4<x<10} Λ (x<=6) ⇒ pmd (x!=6, x=x-1)

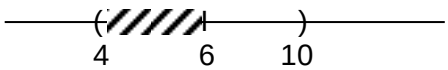
(1) pmd $(x!=6)_x^{x+1}$ ≡ x+1 ! = 6 ≡ x != 5

(4 < x < 10) Λ (x > 6) ≡ (6 < x < 10) ⇒ x != 5



(2) pmd $(x!=6)_x^{x-1}$ ≡ x - 1 ! = 6 ≡ x != 7

(4 < x < 10) Λ (x <= 6) ≡ (4 < x < 10) ⇒ x != 7



Por lo tanto el algoritmo es correcto

Ejemplo 23: Calcular la precondición más débil del siguiente fragmento de programa**:**

```
Si ( y == 0 )
   entonces  x = 0
    sino  x = y – 1
      finsi
  { x==y }
```

Se debe encontrar {P1} y {P2} y probar que

{P1} Λ (y ==0) ⇒ pmd (x==y, x=0)

Λ

{P2} Λ (y != 0) ⇒ pmd (x==y, x= y-1)

pmd (x==y, x=0) ≡ $(x==y)_x^{0}$ ≡{0==y}

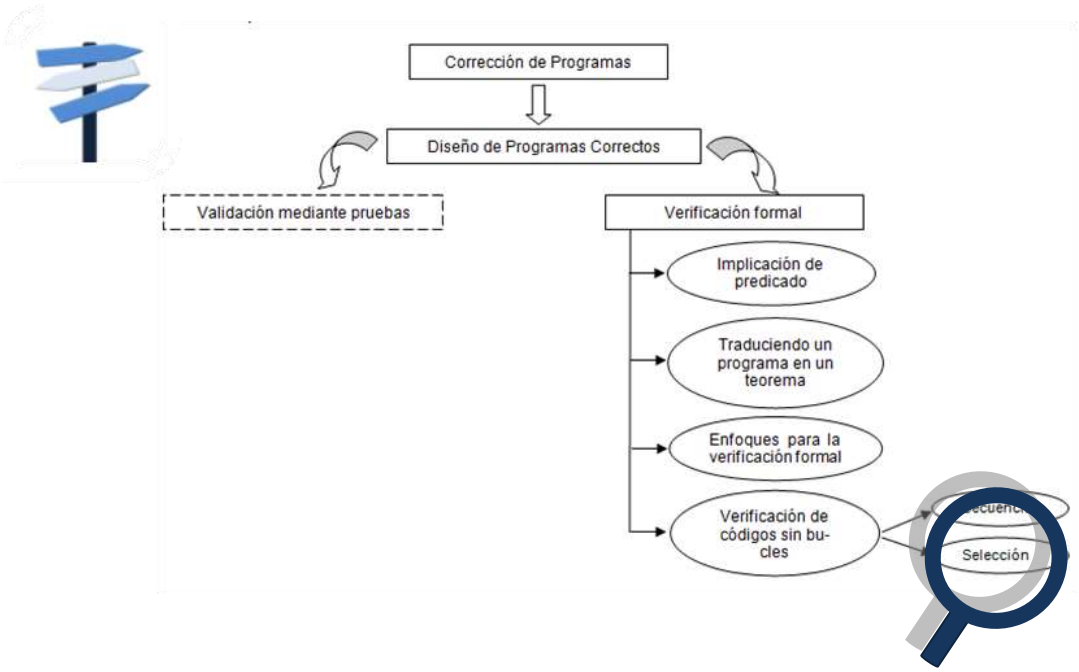
entonces {P1} Λ (y==0) ⇒ {y==0} si **{P1: y == 0}**

pmd (x==y, x= y-1) ≡ $(x==y)_x^{y-1}$ ≡{y-1==y} ≡ Falso **{P2: falso}**

⇒ pmd = **{P1} V {P2}** ≡ **{y == 0} V falso** ≡ **{y == 0}**

Esto significa que la precondición para que el fragmento del programa sea válido es

{ y== 0 }



Sentencia de Selección sin clausula o rama sino (Selección Simple)

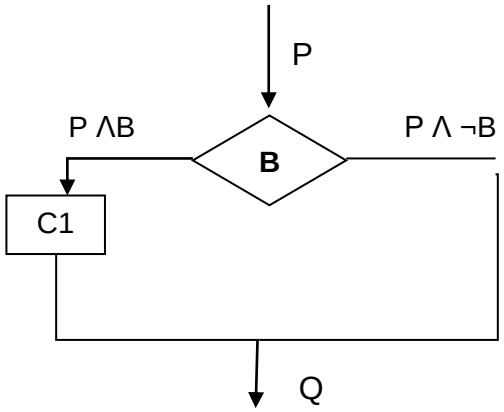


Figura 5.6 Selección Simple

En caso de que no existiera la rama del sino, no podemos dejar de verificar esa parte, ya que también debemos asegurar que si no se satisface la condición, se puede implicar la postcondición y por tanto:

Regla para condiciones sin cláusula sino

$$\frac{\{P \wedge B\}\ C1\ \{Q\} \qquad \{P \wedge \neg B\}\ skip\ \{Q\}}{\{P\}\ SI\ B\ entonces\ C1\ \{Q\}}$$

Ejemplo 24: Demostrar que el siguiente algoritmo que calcula el mínimo entre 2 números es correcto:

```
{ }
Si (a < min)
   entonces min=a
Finsi
{a ≥ min}
```

Se debe encontrar {P1} y {P2} tal que

{pmd } ≡ {P1 } v {P2 }

esto es:
{P1 Λ B} C1 {Q} es correcto y {P2 Λ¬B} skip {Q} es correcto. (1)

Equivale a demostrar

{P1} Λ (a < min) ⇒ pmd (a ≥ min, min=a) ≡ (a ≥ min) $_{min}{}^{a}$ ≡a ≥ a ≡ true
Λ
{P2} Λ (a ≥ min) ⇒ pmd (a ≥ min, Skip); {P2} ≡ a ≥ min

{True} V {a ≥ min} ≡ true como quería demostrarse.

Ejemplo 25: Encuentre la pmd para que el siguiente fragmento de seudocódigo sea correcto.

```
{ P: ? }  x,y ∈ Z
Si ( y<x)
      entonces  aux=y
                y=x
                x=aux
finsi
{ y ≥ x }
```

Se debe encontrar {P1} y {P2} tal que:
{P1 Λ (y<x) } aux=y; y=x; x=aux { y ≥ x } es correcto
y
{P2 Λ(y≥x) } skip { y ≥ x } es correcto

{pmd } ≡ {P1 } v {P2 }

Demostrar que P1 Λ (y<x) ⇒ pmd ({ y≥x; x=aux, y=x, aux=y})
≡ (((y ≥ x) $_{x}{}^{aux}$) $_{y}{}^{x}$) $_{aux}{}^{y}$) ≡ (((y ≥ aux) $_{y}{}^{x}$) $_{aux}{}^{y}$)]
≡((x ≥ aux) $_{aux}{}^{y}$) ≡ { x ≥y}
{pmd: x ≥y}

Si se elige {P1} ≡ (y < x) ; P1 Λ (y < x) ⇒ pmd

Entonces **{P1: y < x}**

Demostrar que P2 Λ (y≥ x) } ⇒ pmd (y ≥ x; SKIP)

≡ {pmd: y ≥ x}

Si {P2}≡ (y ≥ x) ; P2 Λ (y ≥ x) ⇒ pmd

{P2: y ≥ x}

La pmd de la selección es:

{pmd } ≡ {P1 } v {P2 } ≡ { true}

Esto significa que el algoritmo es válido $\forall\ x, y\ \in Z$

La identificación de la pmd de la selección sigue pasos específicos, los siguientes procedimientos señalan cada uno de ellos con el objetivo de asegurar la aplicación de los contenidos adecuadamente.

Como resumen de lo desarrollado respecto a verificación de selección se han elaborado dos procedimientos que guían la resolución de problemas:

Procedimiento para encontrar la pmd de una Selección

1 - Probar (P1 ^ B) ⇒ pmd(Q,C1)

1.a - Calcular la pmd(Q,C1).

1.b - Encontrar P1 probando la implicación.

2 - Probar (P2 ^ ~B) ⇒ pmd(Q,C2)

1.a - Calcular la pmd(Q,C2).

1.b - Encontrar P2 probando la implicación.

3 - Hallar la pmd de la Selección

{pmd} ≡ {P1} v {P2}

Procedimiento para verificar una Precondición dada en una Selección

1 - Probar (P ^ B) ⇒ pmd(Q,C1)

1.a - Calcular la pmd(Q,C1).

1.b - Evaluar la implicación.

2 - Probar (P ^ ~B) ⇒ pmd(Q,C2)

1.a - Calcular la pmd(Q,C2).

1.b - Evaluar la implicación.

Actividad 11: Mapa conceptual de la unidad

Los conceptos presentados en la unidad como así también las relaciones y datos relevantes se expresan en el siguiente mapa conceptual, el objetivo de éste es confrontar lo interpretado haciendo un recorrido por el mismo. Se propone su lectura repasando los conceptos vistos, identificar y fundamentar las relaciones que se mencionan haciendo agregados o cambios que se consideren oportunos.

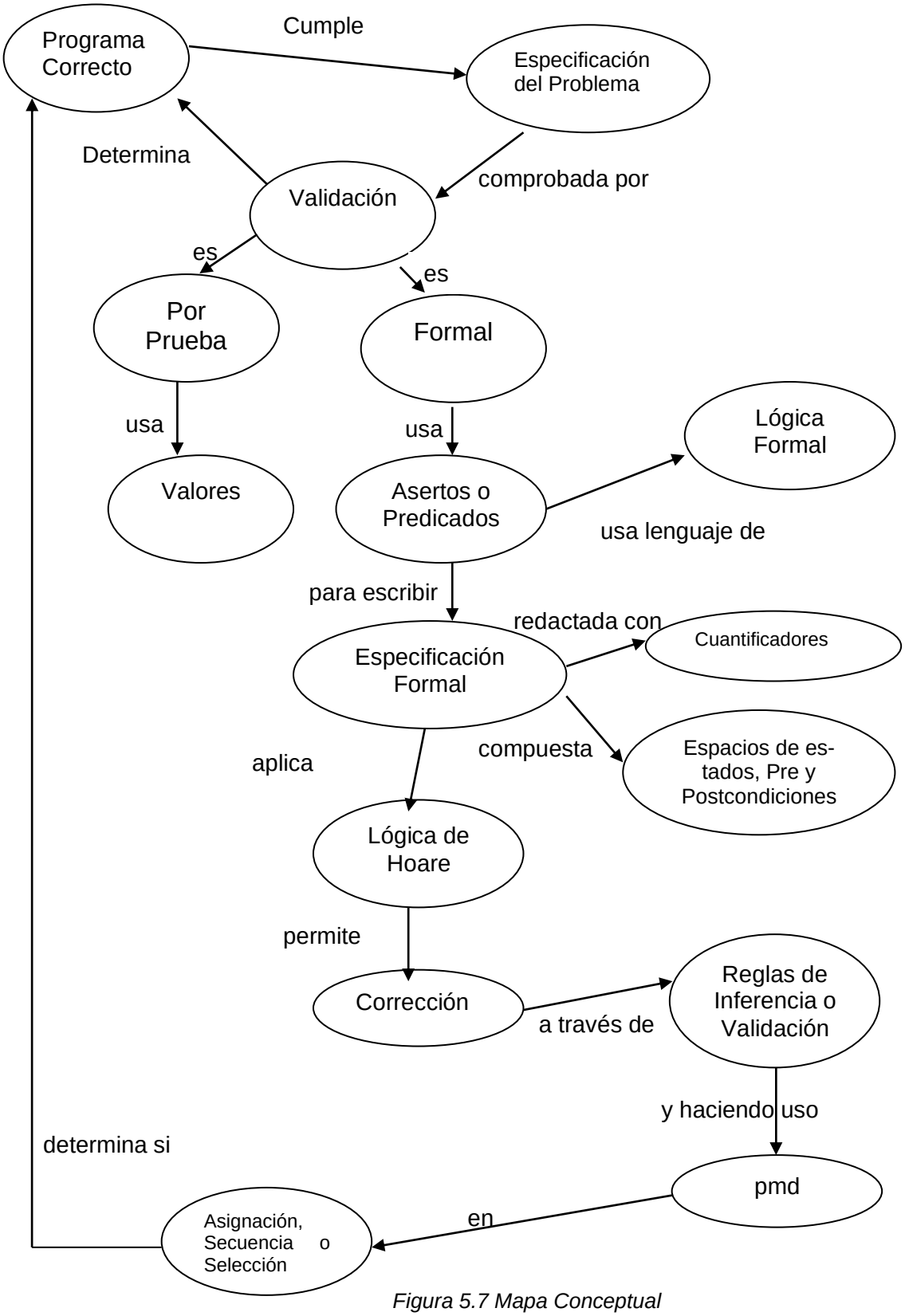


Figura 5.7 Mapa Conceptual

5.8 Bibliografía

- Backhouse Roland (2006) Algorithim Problem Solving. University of Nottingham. UK. Wiley.
- Backhouse Roland (2003) Program Construction: Calculating Implementations from specifications. University of Nottingham. UK. Wiley
- Blanco, J.; Smith, S; Barsotti. D (2008) Cálculo de Programas. Facultad de Matemática, Astronomía y Física. Universidad Nacional de Córdoba.
- Grassmann Winfried, K.; Tremblay, J. (1997) Matemática discreta y lógica. Un perspectiva desde la Ciencia de la Computación. Ed. Prentice Hall. España.
- Kaldewaij, A. (1990) The Derivation of algorithms. Prentice Hall International Series in Computer Science.
- Martí Oliet, N.; Segura Díaz, C; Verdejo López, J. (2006) Especificación, derivación y análisis de algoritmos. Ed. Pearson Prentice Hall. España.
- Martí Oliet, N. y otros (2008) Lógica matemática para informáticos. Ed. Pearson Prentice Hall. España.
- Silva Ramirez, E. (2010) Verificación Formal de Algoritmos. Ejercicios Resueltos. Universidad de Cádiz. Servicio de Publicaciones.

Documentos de Internet

- Algorithim Problem Solving. Backhouse Roland University of Nottingham.

https://www.researchgate.net/publication/228798863_Algorithmic_Problem_Solving-Three_Years_On.

Descarga: Febrero 2018.

- Verificación de Programas. Facultad de Ciencias Exactas –UNICEN.

http://www.exa.unicen.edu.ar/catedras/ccomp2/Verificacion.pdf

Descarga: Febrero 2018

- Verificación Formal de Algoritmos. Departamento de Informática Universidad de Valladolid Campus de Segvia

http://www.infor.uva.es/~jvalvarez/docencia/tema8.pdf
Descarga: Febrero 2018

- Lógica de Predicados Apuntes de Lógica Matemática. Francisco José González Gutiérrez. Escuela Superior de Ingeniería, Departamento de Matemáticas. Universidad de Cádiz.

http://www2.uca.es/matematicas/Docencia/ESI/1710040/Apuntes/Leccion1.pdf
http://www2.uca.es/matematicas/Docencia/ESI/1710040/Apuntes/Leccion2.pdf
http://www2.uca.es/matematicas/Docencia/ESI/1710040/Apuntes/Leccion3.pdf
Descarga: Febrero 2018

Práctico Unidad 5: Verificación Formal

Objetivos

- Demostrar formalmente la corrección de algoritmos secuenciales y con acciones alternativas.
- Adquirir responsabilidad y compromiso por su propio aprendizaje.
- Desarrollar capacidad para comunicar sus puntos de vista en forma oral y escrita.
- Participar en grupos de trabajo, respetando las ideas de sus compañeros.

Ejercicio 1

Establecer el orden de implicación entre los siguientes pares de predicados (x,y) $\in$ ZxZ

a) { P : x ≥ 0} , { Q : (x ≥ 0) $\wedge$ (y ≥ 0 }
b) { P : (x ≥ 0) $\vee$ (y ≥ 0) } , { Q : x + y ==0}
c) { P: x < 0} , { Q: $x^2 + y^2 == 9$}
d) { P: (x < 1) $\rightarrow$ (x ≥ 0) } , {Q : x ≥ 1}
e) {P: x>0}, {Q: (x>0) ^ (y>0)}
f) {P: x>0}, {Q: y≥0}

Ejercicio 2

En cada uno de los siguientes casos, determinar si el algoritmo es correcto para la precondición dada. Suponer que las variables *x, y* son de tipo *entero.*

a) {x ==b} x = x + 1 {x>b}
b) {x > 10} x = x - 10 {x==10}
c) {x < 10} x = x^2 {x>= 16}
d) {(3≤x ≤5) v (x ≤ -3)} x = x + 5 {x>6}
e) {VERDADERO} x = 16 {x ==16}

Ejercicio 3

a) Hallar la precondición más débil (pmd) de la siguiente asignación:

{pmd}
z = z^2 + 4
{Q: (z < 40) $\wedge$ (z resto 2 !=0) $\wedge$ (z ϵ $\mathbb{N}$)}

b) Indicar cuáles de las siguientes precondiciones hacen que el algoritmo sea correcto y cuáles no.

1) {P: z<3}
2) {P: z==4}
3) {z>5}
4) {(z==5) $\vee$ (z==1)}

Ejercicio 4

Indicar una precondición que haga que el algoritmo sea correcto y otro que lo haga no válido. Justifique la respuesta.

{?}
x = x+1
y = y+2
{x < y}

Ejercicio 5
Indicar una precondición que haga que el algoritmo sea correcto y otro que lo haga no válido. Justifique la respuesta.

```
{?}
y = x*3
x= 12 + y
{x ≤3*y}
```

Ejercicio 7
a) Dado el siguiente fragmento de código, hallar la pmd:

```
{pmd}
si  (z < 1)
  entonces  z = z^2
  sino z = z + 2
finsi
{Q: z > 0 }
```

b) Indique una precondición para la cual el algoritmo se correcto y otra que lo transforme en no válido. Justifique.

Ejercicio 8
Indique si el siguiente algoritmo es válido para las especificaciones dadas. Justifique la respuesta.

```
{P:  0  < z < 30, z ∈ R}
 si  (z < 10)
            entonces  z = z+3
    sino z = z – 2
 finsi
 {Q: z <20}
```

Ejercicio 9
Indique si el siguiente algoritmo es válido para las precondiciones dadas, sabiendo que x ∈ Z. En todos los casos justifique la respuesta.

```
si  (x <0)
   entonces  x = x-10
    sino x = x-15
finsi
 {Q: x>5}
```

a. {P: -2< x < 0}
b. {P: -2 < x < 10}
c. {P: x > 25}
d. {P: x < 25}
e. {P: -20 < x < 35)

Anexos

Anexo I: Seudocódigo y Lenguaje C

Introducción

Hasta ahora los algoritmos se escribían en seudocódigo, seudo-lenguaje que permitió mediante una sintaxis sencilla, escribir las soluciones a los problemas que se plantearon. Recordemos, que en las etapas planteadas para resolver problemas, luego de realizar el análisis del mismo y diseñar una solución, las etapas siguientes involucraban el uso de cualquier lenguaje de programación para que luego pueda ser interpretado por la computadora. El lenguaje C, es el lenguaje de programación elegido en este curso para que los algoritmos que ya fueron planteados en seudocódigo sean codificados.

En este anexo se brinda una breve información acerca de algunos elementos del lenguaje C, de tal manera, que se puedan ejecutar los algoritmos que fueron escritos y verificar que realmente hacen lo que se pretendía.

C es un lenguaje *de propósito general* ya que se utiliza para diversas aplicaciones, responde al paradigma *de programación estructurada* y es *imperativo,* es decir orientado a órdenes. No está ligado a ningún sistema operativo ni máquina, lo que permite la portabilidad de los programas entre distintas plataformas.

Es un lenguaje de alto nivel pero que provee Sentencias de bajo nivel que permiten facilidades similares a las que se encuentran en los lenguajes ensambladores.

A continuación se resume la forma en que se codifican los distintos elementos estudiados, lo que te permitirá construir tu **programa fuente**, esto es un programa escrito en un lenguaje que puede ser interpretado por la computadora.

Estos conceptos deberán ser profundizados en la bibliografía que recomendamos, la ayuda proporcionada por el lenguaje C desde su editor o consultando páginas de Internet.

Codificación Algoritmos

Datos

Un dato tiene asociado: un **Nombre**, un **Valor** y un **Tipo.**

Identificador

El nombre con que se designa un dato, se llama identificador. Los identificadores referencian constantes simbólicas, variables, funciones, etc.

Regla de Formación

Un identificador es una secuencia de 1 o más caracteres (letra, dígito o carácter subrayado). El primer carácter sólo puede ser letra o subrayado (under score).

Actualmente, el ANSI C[8] reconoce 32 caracteres para la secuencia que representa un identificador.

NOTA:

- El lenguaje C es case sensitive respecto de los identificadores, es decir que mayúsculas y minúsculas representan distintos caracteres en un identificador. Ejemplo: radio es un identificador diferente que Radio.
- Las funciones provistas por el lenguaje C usan como identificadores letras minúsculas, se denominan palabras reservadas y tienen un significado específico; por lo tanto no pueden ser utilizadas como identificadores por el programador. Ejemplo: *main, scanf, printf,* etc,

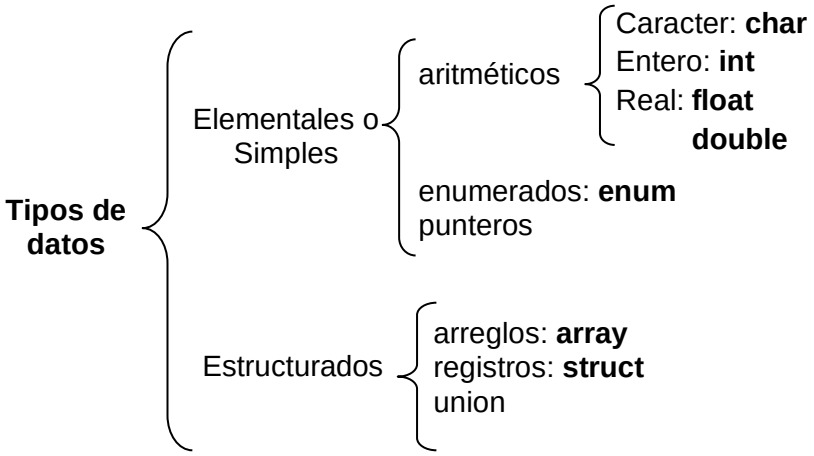
[8] ANSI (American Nationla Estándard Institute) es una organización internacional encargada de la construcción de una definición standard de C, independiente de la máquina.

Palabras reservadas en C

asm	char	goto	sizeof	volatile
auto	const	if	static	while
break	continue	int	struct	
case	default	long	switch	
do	enum	register	typedef	
do	extern	return	union	
double	float	short	unsigned	
else	for	signed	void	

Tipos de Datos

Los datos que utilizan los programas se pueden clasificar en base a diferentes criterios.
Los tipos de datos que utiliza el lenguaje C, se muestran en la siguiente clasificación:



Los tipos aritméticos y enumerados que provee el lenguaje C, cumplen con la condición de que se puede establecer la relación de orden <= (“menor o igual”) entre dos valores cualesquiera de un dato dado. Por ello, se los conoce también como *tipos escalares.*

Expresiones

Una expresión en lenguaje C es una combinación válida de operadores y operandos que representa un cálculo determinado y que al ser evaluada tiene un único resultado. Estos operandos pueden ser constantes, variables, identificadores de funciones u otras expresiones. Los operandos y operadores que intervienen, determinan el tipo de expresión.

Expresiones aritméticas

Si los operandos de una expresión son del tipo aritmético y aparecen relacionados con operadores aritméticos, se tiene una **expresión aritmética**.

Algunas consideraciones a tener en cuenta son:

- La suma, resta y multiplicación admiten operandos enteros y reales, por lo que su resultado puede ser entero o real.
- Respecto del operador cociente ***/*** :
  - La división por cero produce un error fatal, el programa finaliza de inmediato su tarea.
  - La división de dos números enteros da como resultado un entero. Así 10***/***3 da como resultado 3.
- El operador resto **(%)** se aplican sólo a operandos enteros, siendo su resultado entero.
- El operador **sizeof** recibe como argumento una variable o tipo de dato y devuelve el tamaño en bytes del argumento.

La siguiente tabla muestra los distintos operadores aritméticos permitidos por el lenguaje C.

Operandos	Operadores	Tipo de Resultado
Variables o Constantes numéricas	**+** Suma **-** Resta **-** Cambio de signo ***** Producto **/** Cociente o División **%** Módulo o resto de la división entera **++** Incremento en uno **--** Decremento en uno **sizeof** Tamaño de un tipo de dato o variable	Numérico

Tabla AII.1 Expresiones Aritméticas

Reglas de evaluación de una expresión aritmética

Son las mismas reglas que se utilizan en matemática.
Los operadores de una expresión aritmética tienen un orden de precedencia, esto es, prioridad de ejecución en caso que la expresión contenga distintos operadores sin especificación de paréntesis:

Prioridad (de mayor a menor)	Asociatividad
+,- (operadores unarios)	de izquierda a derecha
***, /, %** (operadores binarios)	de izquierda a derecha
+ , - (operadores binarios)	de izquierda a derecha

Tabla AII.2 Orden de precedencia Operadores Aritméticos

Por lo tanto, la evaluación de la expresión:
a * 2 + b % 5 da como resultado 50 para a=24 y b=12.

En este ejemplo los pasos seguidos para la evaluación son:
1. Primero se calcula el duplo de **a** (el operador ***** tiene mayor prioridad que el operador **+**).
2. Luego se realiza el cálculo de módulo *b % 5*
3. Finalmente se calcula la suma de los resultados obtenidos en punto 1 y 2.

Cuando se desea cambiar el orden de las operaciones se utilizan paréntesis.
Por ejemplo, la expresión:
a * (2 + b) % 5 da como resultado 3 para a=2 y b=12.

Así como pueden aparecer paréntesis para cambiar el orden de las operaciones, algunas expresiones aritméticas presentan anidamiento de paréntesis. En estos casos la evaluación se inicia desde los paréntesis más internos.

Reglas de asignación en expresiones aritméticas

Si los dos operandos en una expresión de asignación son de distinto tipo, entonces el valor del operando de la derecha será automáticamente convertido al tipo de la variable de la izquierda.
1. Un valor en coma flotante puede truncarse si se asigna a un identificador entero.
2. Un valor en doble precisión puede ser redondeado si se asigna a un identificador de precisión simple.
3. Una cantidad entera puede ser alterada si se asigna a un identificador de carácter.

Expresiones relacionales

En una expresión relacional aparecen dos operandos del mismo tipo. Los operadores utilizados se muestran en la siguiente tabla:

Operandos	Operadores	Tipo de Resultado
Datos del mismo tipo: ambos numéricos, carácter o lógico. Expresiones numéricas	**>** Mayor **<** Menor **==** Igual **!=** Distinto **> =** Mayor o Igual **< =** Menor o Igual	Booleano

Tabla AII.3 Expresiones Relacionales

A partir de estos operadores, se pueden construir expresiones relacionales, que al ser evaluadas pueden retornar como resultado el valor verdadero o falso.
Como el lenguaje C no trabaja con variables lógicas o booleanas, el resultado es un valor distinto de cero o cero, según que el valor de verdad de la expresión sea verdadero o falso.
Al igual que en las expresiones aritméticas, la asociatividad de los operadores lógicos es de izquierda a derecha.
Ejemplo
(a * 2) < (b % 5) da como resultado Falso (distinto de cero) para a=24 y b=12.

Expresiones lógicas

Los operandos de las expresiones lógicas son expresiones relacionales, variables o constantes de tipo lógico.

Operandos	Operadores	Tipo de Resultado
Expresiones relacionales, variables o constantes de tipo lógico.	**!** negación **&&** conjunción **‖** disyunción	Booleano

Tabla AII.4 Expresiones Lógicas

A partir de operadores lógicos es posible construir expresiones lógicas.
La evaluación de una expresión lógica da como resultado un valor booleano que se obtiene de aplicar la tabla de verdad a los operandos.

Los operadores lógicos, tienen el siguiente orden de precedencia.

Orden de Precedencia	Operador
1	**!**
2	**&&**
3	**‖**

Tabla AII.5 Orden de precedencia Operadores Lógicos

Ejemplo:
El resultado de la siguiente expresión lógica para a=24 y b=12 es:

((a * 2) > (b % 5)) && (b < 20)

La expresión relacional **((a * 2) > (b % 5))** da como resultado Falso (distinto de cero).

La expresión relacional **(b < 20)** da como resultado Verdadero (igual a cero)

Entonces la expresión lógica, según la tabla de verdad, es Falsa.

Asignación: Operadores

Como se estudió, la asignación permite almacenar en una variable un valor que proviene de una constante, de otra variable o del resultado de una expresión.
El formato general de la asignación es:

<Nombre de variable> **=** <valor>

Donde.
<Nombre de variable> es el identificador de la variable a la que el procesador le asignará <valor>.
<valor> puede ser una constante, otra variable o el resultado de una expresión.

La Tabla 1.6 muestra los operadores de asignación permitidos por el Lenguaje C.

Operador	Acción
=	Asignación
+=	Asigna la suma
***=**	Asigna producto
/=	Asigna cociente
%=	Asigna módulo

Tabla AII.6: Operadores de asignación permitidos por el lenguaje C

Ejemplo:
int i, j, k, result;
result = (i * i) + 120 – k;

Para los otros operadores de asignación **+=, -=, *=. /=, %=,** antes de efectuar la asignación, se realiza la operación que precede al símbolo =.

Por ejemplo:

a+=b es equivalente a a =a+b
a*=c es equivalente a a =a*c
a+=1 es equivalente a a++

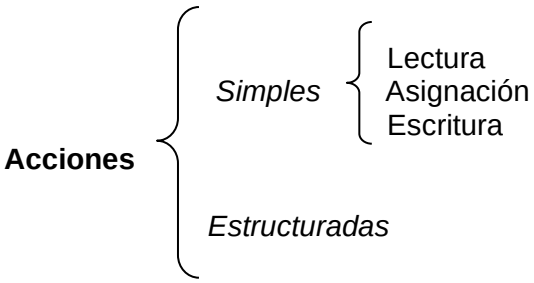
Precedencia de operadores
La siguiente tabla muestra el orden de precedencia de los operadores analizados.

Categoría del operador	Operadores	Asociatividad
Operadores monarios	**- ++ -- ! sizeof**(tipo)	Derecha a izquierda
Multiplicación, división y resto aritmético	*** / %**	Izquierda a derecha
Suma y sustracción aritméticas	**+ -**	Izquierda a derecha
Operadores relacionales	**< <= > >=**	Derecha a izquierda
Operadores de igualdad	**= = !=**	Izquierda a derecha
Operador y,o, no lógico	**&& || NO**	Izquierda a derecha
Operador condicional	**? :**	Derecha a izquierda
Operadores de asignación	**= += -=** ***= /= %=**	Derecha a izquierda

Tabla AII.7: Precedencia y asociatividad de los operados analizados

Acciones

Las acciones estudiadas en seudocódigo, son las Acciones de los lenguajes de programación. Las acciones se clasificaron en:



Veremos cuales son las Acciones correspondientes en Lenguaje C.

Acciones Simples

Son las Acciones de Entrada y Salida y la sentencia Asignación. Las Acciones de Entrada y Salida permiten ingresar valores a los datos desde un dispositivo externo como el teclado y mostrarlos en la pantalla respectivamente. La sentencia de asignación permite otorgar valor a las variables desde el mismo programa.

SEUDOCÓDIGO	**LENGUAJE C**
ACCIONES SIMPLES	
Acciones de Entrada Salida	
Leer <nombre de variable, .. , > **Escribir** <nombre de variable . >	**scanf** ("*especificador de formato*",**&**nombre de variable, ..) **printf** ("cadena de caracteres **%** *especificador de formato*", *nombre de variable*, .. , ..)

Estas Sentencias son funciones provistas por C e incluidas en el archivo de cabecera **stdio.h**. Por lo tanto, todo programa que utilice alguna de estas Sentencias de I/O, debe incluir la directiva:

 # include <stdio.h>

Lectura: scanf

Esta función permite la entrada formateada de valores numéricos, caracteres y cadena de caracteres.

Formato:

scanf ("especificador de formato", arg1,arg2, ...,argn);

La sentencia **scanf** posee dos argumentos, el primero que aparece encerrado entre comillas es el especificador de formato, que indica el tipo del valor que se desea leer. El segundo argumento es el nombre de la/s variables precedidos por el símbolo **&**. Esta sentencia se interpreta así: *lea un valor del tipo indicado en el formato y guárdelo en el lugar de memoria reservado para la variable cuyo nombre se indica.*

especificador de formato	El especificador de formato comienza con el símbolo **%** seguido de un carácter de conversión.
arg1, arg2, ...	El nombre de cada variable, precedido por el carácter **&**(ampersand)

El uso del carácter **&** se debe a que los nombre son en realidad direcciones de memoria de las variables.
Por ejemplo en la sentencia **scanf ("%f", &radio);**

&radio indica la dirección de memoria de la variable en donde se almacenará el valor que el usuario ingrese por teclado para el radio.

Escritura: printf
Esta sentencia permite mostrar textos y/o valores de variables, constantes, expresiones, en la pantalla.
Formato:

printf (“cadena de caracteres”, arg1,arg2, ..., argn)

La sentencia **printf** posee dos argumentos, el primero es el especificador de formato que aparece encerrado entre comillas con un texto optativo pero generalmente necesario, el segundo argumento es el nombre de la variable que se desea mostrar.
Observar que ahora no es necesario colocar el símbolo &

cadena de caracteres	conjunto de caracteres que lleva además de cualquier texto, las especificaciones de formato de cada uno de los argumentos arg1,,..., argn. Para especificar el formato de cada argumento se debe utilizar el signo de porcentaje (%) seguido del carácter de conversión.
arg1,arg2,...	son los argumentos que representan los datos a mostrar.

Para cada variable que se lee o escribe debe especificarse su formato, el que depende del tipo de dato.
Los especificadores más utilizados son:

%d para variables enteras
%f para variables reales
%c para variables de tipo carácter

Como se observa un especificador de formato está compuesto por el símbolo % seguido de un *carácter de conversión* asociado al tipo de dato. Los siguientes son algunos caracteres de conversión provistos por el lenguaje:

Carácter	Significado
c	El dato es un carácter
d	El dato es un entero decimal
e	El dato es un valor en coma flotante visualizado como exponente
f	El dato es un valor en coma flotante de simple precisión
g	El dato es un valor en coma flotante de doble precisión
h	El dato es un entero corto
o	El dato es un entero octal
s	El dato es una cadena de caracteres
u	El dato es un entero decimal sin signo
x	El dato es un entero hexadecimal
hd	El dato es un entero decimal corto
ld	El dato es un entero decimal largo

Tabla AII.8: Caracteres de Conversión

Especificadores de campo: Cada formato puede incluir además especificadores que permitan salidas más presentables, tales como:

Especificador	Significado
w	Cantidad de posiciones en la pantalla. Se utiliza para dar formato a argumentos de tipo Entero o Carácter.
w.d	w: cantidad de posiciones en la pantalla d: cantidad de posiciones decimales Se utiliza para dar formato a argumentos de tipo Real.

Tabla AII.9 Especificadores de campo

Ejemplo: El siguiente es un programa sencillo en lenguaje C, que contiene algunas características importantes que analizaremos:

```
#include <stdio.h>
 main()
  {
    float r=45.68;
    int a=10,b=30;
    printf("%f", 2*r);
    printf("\n La suma de %d + %d es %d",a, b, a+b);
  }
```

 # include <stdio.h> es una directiva que indica al preprocesador de C que debe incluir el archivo stdio.h. Este archivo contiene información que el compilador necesita para reconocer como válidas las funciones de entrada/salida estándar scanf y printf.

La línea **main ()** forma parte de todo programa **C.** Las Sentencias que siguen a main y que están encerradas entre llaves, constituyen un bloque llamado función. Todo programa en Lenguaje C comienza a ejecutarse desde la función main .

Las Sentencias `float r=45.68;`
`int a=10,b=30;`
permiten declarar el tipo de las variables **r**, **a** y **b** y asignarles valores.

Observar que toda sentencia de C finaliza con un punto y coma**.**

La línea **`printf("%f", 2*r );`**

Indica a la computadora que debe mostrar en pantalla el doble del valor r.

printf tiene en este caso dos argumentos: el primero es el especificador de formato y el segundo lo que debe ser impreso.

El primer argumento contiene una cadena de caracteres entre comillas que contiene el especificador de formato **%f** ya que la variable **r** es real.

Para especificar el formato de cada argumento se debe utilizar el signo de porcentaje (%) seguido de carácter de conversión que corresponde al tipo de dato.

printf("\n La suma de **%d** + **%d** es **%d",**a, b, a+b);

En esta línea se puede observar la correspondencia uno a uno entre especificadores de formato y argumentos a imprimir.
Como son tres valores enteros a escribir, deben aparecer tres especificadores de formato. Los valores de a, b y su suma reemplazarán a cada uno de sus especificadores y se obtendrá la siguiente salida:

Salida: 91.360001

La suma de 10 + 30 es 40

Si la se escribe la sentencia printf("%**10.2f**", 2*r) haciendo uso del especificador de formato para reales; la salida en pantalla sería 91.36

El carácter **\n** que aparece en la segunda sentencia printf, es el que permite que el cursor salte al comienzo de la siguiente línea antes de escribir la salida.

Sentencia gets

Permite la lectura de una cadena de caracteres. Esta cadena puede contener espacios en blancos.

Formato:

gets (<variable cadena de caracteres>)

Sentencia puts

Permite escribir una cadena por pantalla.

Formato:

puts (<variable cadena de caracteres>)

NOTA: Si bien scanf también permite la lectura de una cadena utilizando el especificador de formato "%s", esta función no admite palabras separadas por espacios en blanco, pues cuando detecta un espacio, corta el ingreso de datos.

Sentencias Estructuradas

Investigaciones realizadas por C. Böhm y G Jacopini demostraron que los programas podían codificarse utilizando sólo las tres estructuras de control estudiadas en seudocódigo: la estructura de secuencia, la de selección y la de repetición o iteración[9].

Estructuras de Control { Secuencia; Selección; Iteración

La siguiente tabla muestra la correspondencia entre las estructuras de control estudiadas en seudocodigo:

Secuencia	Selección o Alternativa	Iteración
S.Compuesta **{ }**	if	For
	switch	While
		do..while

[9] Programación en C. Metodología, estructura de datos y objetos. Aguila y Martínez.

Secuencia
Una secuencia está representada por un conjunto de dos o más sentencias que se ejecutan en forma consecutiva, una sola vez. En este caso el orden de ejecución coincide con el orden físico en que se representan las órdenes.
Toda sentencia compuesta comienza con" { " y termina con " } ".

Sentencias Alternativas
Una sentencia alternativa está formada por dos Sentencias simples o estructuradas, de las cuales sólo una se ejecuta en un determinado momento, dependiendo del resultado de la evaluación de una condición.

Alternativa doble
Formato:

En seudocódigo:	**En lenguaje C:**
Si (expresión)	**if** (expresión)
Entonces sentencia1	sentencia1
Sino sentencia2	**else**
FinSi	sentencia2

expresión	puede ser una variable lógica o una expresión relacional o lógica
sentencia1, sentencia2	pueden ser simples o estructuradas
cláusula else	es optativa, si no aparece es una alternativa simple ya que la acción se realiza o se ignora, de acuerdo al valor de la expresión

Alternativa simple
Formato:

En seudocódigo:	**En lenguaje C:**
Si (expresión)	**if** (expresión)
entonces sentencia1	sentencia1;
FinSi	

Ejemplos:
3.1.

```
if (i!=0)
      printf("\n i es distinto de 0");
  else
      printf("\n i es 0");
```

Si por alguna de las ramas deben ejecutarse más de una sentencia, éstas deben ir entre llaves (sentencia compuesta)
3.2

```
if (a > b)
        {
        a=a+1;
        b=c+a;
        }
  else
```

```
        printf("\n terminar");
3.3
if ((a > b) && (a<10))
     {
     a=a+1;
     b=c+a;
     }
   printf(“ Fin “);
```

En este caso, sea verdadera o falsa la condición, se ejecuta la sentencia printf(“Fin “).

Estructura de selección múltiple: switch

La acción Según vista en seudocódigo, que permite contemplar condiciones con alternativas múltiples, se corresponde con la sentencia switch.

Formato:

```
switch (expresión)
{
        case valor1: sentencia_1
        case valor2: sentencia_2
        case valor3: case valor4: sentencia_3
        .
        .
        case valor n: sentencia_n
        default: sentencia
}
```

expresión	expresión entera o char
valor1, valor2, ..valor n	Son etiquetas que representan las distintos valores que puede tomar la expresión.

Según el valor de la expresión se ejecutan las sentencias asociadas a esa etiqueta, ejecutándose también las Sentencias de las etiquetas siguientes. Para evitar esto, se debe utilizar la sentencia **break**, que permite salir de la estructura y continuar con la sentencia posterior al switch.

Ejemplo:

En un programa para manejo de cajero automático es común que aparezca un menú para seleccionar entre una opción requerida. Si son cuatro las opciones posibles del menú: depósito, extracción, consulta de saldos o finalizar transacción, el usuario tiene la posibilidad de elegir aquella que necesita.

El siguiente extracto del programa muestra como con C, utilizando el **switch** se puede resolver lo expresado:

```
switch (toupper(op))
 {
        case 'D': printf(“Depósito”);
                  break;
        case 'E’: printf(“Extracción”);
                  break;
       case 'C’: printf(“Consulta de saldos”);
                  break;
        case 'T': printf(“Finalizar transacción”);
                  break;
        default:  printf("operación no válida\n");
```

```
}
```

toupper es una función provista por C que permite obtener la mayúscula del carácter que se coloca como argumento. Esto permite el ingreso del carácter que corresponde a la variable **op** con mayúscula o minúscula.
Esta función está definida en el archivo de cabecera **ctype.h** por lo que dicho archivo debe incluirse al principio del programa utilizando una directiva del compilador.

Sentencias iterativas
Las Sentencias estructuradas iterativas o repetitivas, se simbolizan en el cuadro siguiente, haciendo un comparativo entre seudocódigo y **C**.

SEUDOCÓDIGO	**LENGUAJE C**
Estructuras Iterativas	**Sentencias Iterativas**
Hacer acción 1 acción 2 acción n **Mientras(<condición>)**	**do** < Sentencia simple o estructurada>**;** **while(<condición>)**
Para v Desde vi Hasta vf acción 1 acción 2 acción n **FinPara**	**for (exp1; exp2;exp3)** < Sentencia simple o estructurada>**;** **exp1:** inicializa la variable de control **exp2:** condición que al tomar el valor verdadero permite la ejecución del ciclo. **exp3:** expresión aritmética que identifica el incremento o decremento de la vble de control
SEUDOCÓDIGO	**LENGUAJE C**
Estructuras Iterativas	**Sentencias Iterativas**
Mientras (<condición>) acción 1 acción 2 acción n **Fin Mientras**	**while (<condición>)** < Sentencia simple o estructurada>**;**

Estructura for
Formato:

```
for ( expresión1; expresión2; expresión3 )
   sentencia
```

Donde:

expresión 1	es el valor inicial de la variable de control
expresión 2	es una condición que comprueba si la variable de control toma el valor final
expresión 3	es un incremento (o decremento) de la variable de control cada vez que se ejecuta al finalizar un ciclo.

Los siguientes ejemplos muestran su funcionamiento:

En seudocódigo:

```
comienzo
entero i
Para i desde 1 hasta 5
      Escribir 2 * i
FinPara
fin
```

En lenguaje C:

```
void main(void)
{
 int i;
 for( i=1; i<=5; i++)
          printf("%d",2 * i);
}
```

Sentencia while

Formato:

En seudocódigo:

```
Mientras (expresión)
    sentencia
Fin Mientras
```

En lenguaje C:

```
while (expresión)
      sentencia;
```

sentencia	Puede ser simple o estructurada. Si sentencia representa un grupo de Sentencias, éstas deben ir entre llaves (sentencia compuesta).
funcionamiento	Mientras la expresión es verdadera (valor distinto de 0) se ejecuta sentencia. Cuando la expresión es falsa (valor 0), se termina la ejecución y se pasa a la sentencia siguiente.

Ejemplo: Leer números naturales y escribir su cuadrado hasta que éste sea igual o mayor que 8142.

Solución Propuesta:

En seudocódigo:

```
Comienzo
entero numero, cuadrado
Leer numero
cuadrado = numero * numero;

MIENTRAS ( cuadrado < 8142 )

  Escribir  “El cuadrado es: ” ,cuadrado
  Leer numero
      cuadrado = numero * numero;
FIN MIENTRAS
Fin
```

En lenguaje C:

```
#include <stdio.h>
#include <math.h>
void main(void)
{
int numero, cuadrado;
scanf("%d", &numero);
cuadrado=pow(numero, 2); /* función poten-
                            cia */
while (cuadrado < 8142)
  {
    printf("el cuadrado es %d", cuadrado);
    scanf("%d", &numero);
    cuadrado=pow(numero, 2);
   }
}
```

Sentencia do .. while

Formato:

En seudocódigo:

```
Hacer
  sentencia
Mientras  (expresión)
```

En lenguaje C:

```
do  sentencia  while  (expresión);
```

Donde **sentencia**, puede ser simple o estructurada.

NOTA:
A diferencia de las otras estructuras, la estructura do .. while es la única incluye el punto y coma (;) al final de su formato.

Ejemplo:
A partir del número 2, mostrar consecutivamente todos los números naturales pares cuyo cuadrado sea menor que 8142.

En seudocódigo:

```
comienzo
entero cuadrado, numero
numero=2
Hacer
  escribir numero
  numero = numero + 2
  cuadrado = numero * numero
Mientras (cuadrado < 8142)
Fin
```

En lenguaje C:

```
void main(void)
{ int cuadrado, numero;
  numero=2;
  do
  {
    printf("%d", numero);
    numero = numero + 2;
    cuadrado=numero * numero;
   } while (cuadrado < 8142);
}
```

Ejemplo:
Se cuenta con los datos correspondientes a una encuesta de opinión, acerca de la despenalización del aborto. Por cada encuestado se registra sexo (F ó M) y un código, que puede valer:

S: si el encuestado está de acuerdo con la despenalización,
N: si el encuestado no está de acuerdo
A: si el encuestado no sabe o desea abstenerse de responder

El ingreso de datos finaliza con código de sexo = X.
Se desea indicar el total de personas que respondieron la encuesta, el porcentaje de encuestados que está de acuerdo con la despenalización del aborto y de ellos el porcentaje que son mujeres.

Algoritmo	**Lenguaje C**

Comienzo
caracter sexo, codi
entero tot, totsi,totf
 	
Escribir “Ingrese sexo, finalice con X ”
Leer sexo
 	
Mientras (sexo <>’X’)
  Escribir “Ingrese codigo de respuesta ”
  Leer codi
  tot= tot +1
  si ((codi=´S’) o (codi=’s’))
  entonces totsi= totsi +1
  si ((sexo=’F’) o (sexo=’f’))
  entonces totf= totf+1
  finsi
  finsi
  Escribir “Ingrese sexo, finalice con X ”
  Leer sexo
Finmientras
 	
Escribir “En un total de“, tot, “encuestados”, totsi *100/ tot, “estan de acuerdo con la despenalización, de ellos el ”, totf*100/ totsi , “son de sexo femenino”
Fin
	# include < stdio.h>
	# include < ctype.h>
	main()
	{
	char sexo, codi;
	int tot, totsi,totf;
	
	printf(“Ingrese sexo, finalice con X ”);
	scanf(“%c”, & sexo);
	
	while (sexo <>’X’)
	{ printf(“Ingrese codigo de respuesta ”);
	  scanf(“%d”, & codi);
	  tot= tot +1;
	  if ((codi== **toupper**(’s’))
	  { totsi= totsi +1;
	  if (sexo== toupper(’f’))
	  totf= totf+1;
	
	  }
	  printf(“Ingrese sexo, finalice con X ”);
	  scanf(“%c”, & sexo);
	} /* fin del while*/
	
	printf(“En un total de %d encuestados %f estan de acuerdo con la despenalización, y de ellos el %f son de sexo femenino” , tot, totsi *100/ tot, totf*100/ totsi);
	} // fin de main

Subprogramas

El lenguaje C permite al programador definir y construir subpropramas a través de funciones.
El uso de funciones definidas por el programador permite dividir un programa grande en un cierto número de componentes más pequeñas (subprogramas), cada una de éstas con un propósito específico y determinado.

Funciones (subprogramas en Lenguaje C)

Una ***función*** es un conjunto de sentencias que realiza una determinada tarea, que retorna como resultado cero o un valor.
El siguiente programa en Lenguaje C muestra una función que modifica el valor del parámetro dentro de su cuerpo.

```
# include < stdio.h >
void modifica ( int x )      /* función modifica */
{
  x += 3;
  printf (" \n x = %d (desde la función, modificando el valor) ", x);
  return;
 }
void main (void )
{int a = 2;
  printf (" \n a = %d (desde main, antes de invocar a la función modificar) ", a);
  modificar (a);
  printf (" \n a = %d (desde main, después de invocar a la función modificar) ", a);
}
```

Al ejecutar este programa, se obtiene la siguiente salida:
a = 2 (desde main, antes de invocar a la función modificar)
a = 6 (desde la función, modificando el valor)
a = 2 (desde main, después de invocar a la función modificar)
Otro ejemplo sería este programa qe calcula el perímetro de un terrreno rectangular a través de un subprograma

```
# include <stdio.h>
float perimetro ( float xl, float xa )
 {float perim;
   perim = 2 * ( xl + xa) ;
   return ( perim );
  }

void main(void)
{float largo, ancho;
  printf("ingrese el largo");
  scanf("%f", &largo);
  printf("\n");
  printf("ingrese el y el ancho");
  scanf("%f", &ancho);
  printf("\n");
  printf("el perímetro del terreno es: %f ", perimetro( largo,ancho ));
}
```

Arreglos

Definición de un Arreglo

La definición de un arreglo lineal se realiza según el siguiente formato:

<Tipo de dato > <Nombre del Arreglo> [cantidad de componentes]

Ejemplo:

int Edades [50];

Como se puede notar, para declarar un arreglo se debe especificar el tipo y el número de componentes (tamaño del arreglo). Esta declaración permite a la computadora reservar la cantidad de memoria necesaria para almacenar los datos del arreglo.

Acceso a una componente de un arreglo

El mecanismo de acceso a una componente se representa de la siguiente manera:

<Nombre del Vector> [<posición>]

donde <posición> puede ser una constante entera, una variable entera o una expresión entera.

Es importante destacar que, en lenguaje C, las posiciones comienzan a enumerarse a partir de 0 (cero).

Carga y escritura de las componentes de un arreglo

Ejemplo

```
void main (void)
{ int   A [15];              /*  A es un arreglo de 15 componentes enteras */
  A[0]=0;
  A[14]=0;

for(i=1; i< 14; i=i +1 )
        scanf("%d", &A[i]);  /*Lee una componente del arreglo A*/
.
for(i=0; i <  14; i++)
        printf("La  edad del aspirantes %d es: %d", i, A[i]); /*Muestra una componente del arreglo A*/
getchar();
}
```

Registros

Los siguientes son algunas consideraciones Para poder implementar en lenguaje C la estructura de datos *registro.*

Definición del tipo y declaración de variables

Para la estructura:

```
struct alumno
        { int dni;
          char nom[20];
          int  legajo;
          char carrera;
        };
```

struct alumno al;

Esto significa que se ha declarado la variable **al** del tipo alumno**,** que tiene 4 campos: dni,nom, legajo y carrera;

al. legajo y **al.carrera** hacen referencia al legajo y carrera de un alumno respectivamente.

Para la estructura:

```
stuct partido_politico
{ int numero;
  int votos [19];
};
```

struct partido_politico m;

m. numero refiere al número del partido político.

m.votos[5] refiere a los votos obtenidos en el sexto distrito.

Operaciones sobre estructuras

Las operaciones de **lectura y escritura** se deben efectuar individualmente para cada uno de sus miembros.

Para asignar valores a un miembro de una struct debe tenerse en cuenta que, una vez accedido mediante el selector, es tratado como cualquier otra variable del mismo tipo.

Sea **al** una variable de tipo **struct alumno,** son correctas las siguientes expresiones:

```
struct alumno
        { int dni;
          char nom[20];
          int  legajo;
          char carrera;
        };
struct alumno al;
```

la lectura de datos es:

```
gets(cl.nom);
scanf( "%d",& al.legajo);
scanf( "%c",& al.carrera);
```

para ingresar respectivamente al saldo, día del último movimiento, nombre y domicilio de un cliente.nombre, legajo y carrera de un alumno

Tipos de datos definidos por el usuario: typedef

Se pueden definir nuevos tipos de datos equivalentes a los tipos ya existentes, utilizando la cláusula **typedef**. De esta manera, una vez definido el nuevo tipo de dato por el usuario, pueden declararse variables de dicho tipo.

En general, un nuevo tipo de datos se define:

typedef <tipo> <nuevo-tipo>;

Para el caso de los registros (struct en Lenguaje C), la definición de tipos se realiza de la siguiente manera:

```
typedef struct {
                 miembro 1;
                 .........;
                 miembro n;
               } nuevo-tipo;
```

EJEMPLO:

```
typedef  struct
    { int dni;
      char nom[20];
      int  legajo;
      char carrera;
    } alumno;
```

Para declarar una variable de este tipo:

alumno al;

En este ejemplo alumno es el nuevo tipo definido por el usuario y **al** es una variable de tipo **alumno**.

La cláusula **typedef** es útil, particularmente para el caso de definir nuevas estructuras, ya que evita escribir repetidamente **struct <identificador>.**

NOTA: los *compiladores de las versiones actuales del lenguaje C permiten definir directamente una struct de la siguiente forma:*

```
struct alumno
   { int dni;
     char nom[20];
     int  legajo;
     char carrera;
   } ;
```

y declarar **alumno al;**
Por lo tanto, en estos casos es innecesario el uso de typdef.

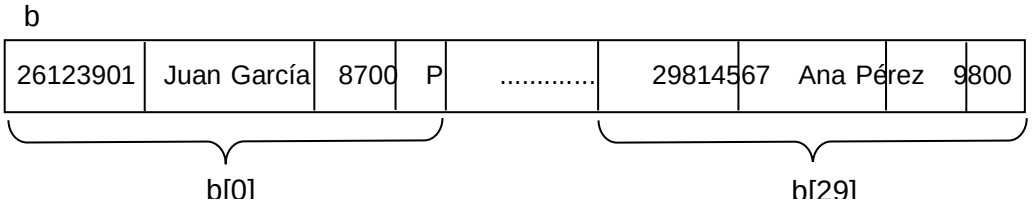
Arreglo de struct

Generalmente las variables de tipo struct se agrupan en conjuntos conocidos como **arreglo de struct**.

EJEMPLO: A continuación se presenta la definición de un arreglo que almacena la información de 30 alumnos.

```
struct alumno
   { int dni;
     char nom[20];
     int  legajo;
     char carrera;
   } ;
alumno   b[30];
```

Al cargar la información de los alumnos, en memoria se tiene por ejemplo**:**



Arreglos como parámetros de funciones

Al invocar una función que tiene como parámetro actual un arreglo debe aparecer el nombre del arreglo sin corchetes ni subíndices.
Al pasar un arreglo a una función, no se pasan los valores de las componentes de ese arreglo sino la dirección del primer elemento del arreglo.
En este ejemplo se realizan varias operaciones sobre un arreglo, el programa muestra como:

- Cargar un arreglo
- Calcular la media o promedio de los valores del arreglo
- Hacer una búsqueda secuencial de un elemento en el arreglo

```
#include <stdio.h>
#define n 20
```

```
/* Función carga */
void carga( float arre[n] )                    /*carga de los datos en del arreglo*/
{int i;
  for (i=0; i < n ; i++ )
       {printf("ingrese  valor \n");
        scanf(" %f", &arre [i]);
        }
  return ;
 }

/* Función media */
float media ( float arre[ ] )                   /* Cálculo la media con los valores del arreglo  */
      {
       int i;
       float acum=0;
       for (i=0; i < n ; i++ )
              acum=acum + arre [i];
       return ( acum/n);
       }
/* Función bus_secuencial */
int bus_secuencial (float arre[], float elem)   // Realiza la búsqueda secuencial en el arreglo
{int posi=0;
  while ( (posi!=n) && (elem != arre[posi]))
         posi++;
  if (posi==n)
          return(-1);
   else return(posi);
}

void main (void)
{int pos;
 float  valor;
 float datos [n];
  carga (datos);
  printf("\nEl promedio de valores es %f ", media (datos));
  getchar();
  printf("ingrese valor a buscar\n");
  scanf("%f",&valor);
  pos = bus_secuencial (datos,valor);
  if (pos != -1)
          printf("\n el elemento se encontró en la posición: %d", pos);
    else printf("\n el elemento no se encontró");
  getchar();
}
```

Funciones de algunas bibliotecas de uso frecuente

Veamos algunas de las funciones clasificadas según la biblioteca a la cual corresponden:

Biblioteca <string.h>

Función	Formato y Aplicación
strlen	**Size_t strlen(const char * origen)** Devuelve la longitud de la cadena

Asignación y concatenación de Cadenas	
strcpy	**char * strcpy(char * destino, const char * origen)** Copia la cadena origen en la cadena destino, el resultado es la cadena destino a la que agrega el carácter nulo.
strncpy	**char * strncpy(char * destino, const char * origen, size_t n)** Copia **n** caracteres de la cadena origen en la cadena destino, con el objeto de realizar truncamiento o rellenado se caracteres.
strcat	**char * strcat(char * destino, const char * origen)** Agrega la cadena origen al final de la cadena destino, devuelve la cadena destino.
strncat	**char * strncat(char * S1, const char * S2, size_t n)** Agrega los n primeros caracteres de S2 a la cadena S1, devuelve S1.
Comparación de Cadenas	
strcmp	**int strcmp(const char * S1, const char * S2)** Compara alfabéticamente S1 y S2. El resultado es un número Positivo si S1 >S2 Negativo si S1 <S2 Cero si S1 =S2
stricmp	**int stricmp(const char * S1, const char * S2)** Es igual a strcmp, pero sin distinguir entre letras mayúsculas y minúsculas.
strncmp	**int strncmp(const char * S1, const char * S2, size_t n)** Compara S1 con la subcadena formada por los n primeros caracteres de S2. Al igual que strcmp devuelve un valor que puede ser positivo, negativo o cero.
Búsqueda de Caracteres y cadenas	
strrchr	**char * strrchr(const char * S, int c)** Devuelve un puntero a la última ocurrencia del carácter c en S, devuelve NULL si c no está en S. La búsqueda se hace en sentido inverso, desde el último al primer carácter.
strstr	**char * strstr(const char * S1, const char * S2)** Busca la cadena S2 en S1y devuelve un puntero a los caracteres donde se encuentra S2 o NULL si no lo encuentra
Conversión de Cadenas	
strlwr	**char * strlwr(char * S)** Convierte las letras mayúsculas de la cadena S a minúsculas.

strupr	**char * strupr(char * S)** Convierte las letras minúsculas de la cadena S a mayúsculas.
Conversión de Cadenas a Números	
atoi	**int atoi(const char * S)** Convierte una cadena en un valor entero, devuelve cero en caso que no pueda realizar la conversión.
atol	**long atol(const char * S)** Convierte una cadena en un valor entero largo (long).
atof	**double atof(const char * S)** Convierte una cadena en un valor double en coma flotante.

Biblioteca <ctype.h>

Función Carácter	Argumento	Resultado	Aplicación
toascii(d)	int	Char	convierte el argumento a ASCII, en el rango 0.. 127
tolower(d)	char	Char	convierte una letra a minúscula
toupper(d)	char	Char	convierte una letra a mayúscula

Biblioteca <math.h>

Función Matemática	Argumento	Resultado	Aplicación
abs(d)	int	Int	valor absoluto de d
fabs(d)	double	double	valor absoluto de d
ceil(d)	double	double	redondeo por exceso al entero más próximo (entero mas pequeño, mayor o igual a d)
sqrt(d)	double	double	raíz cuadrada
fmod(d1,d2)	double double	double	devuelve el resto de d1/d2, con el mismo signo que d1
exp(d)	double	double	e^d
log(d)	double	double	logaritmo natural de d
sin(d)	double	double	seno de d
cos(d)	double	double	coseno de d
tan(d)	double	double	tangente de d

Resumen: Tablas comparativas

SEUDOCÓDIGO	LENGUAJE C
Comienzo y Fin del Algoritmo	
Comienzo Fin	**{** **}**
Tipos de Datos	
Entero Real Carácter Cadena Booleano →	**int** **float** **Char** Estos tipos de datos no son provistos por el lenguaje
Declaración de Variables	
entero a real p	**int a** **float p**
Operadores Aritméticos	
+, - , * , / Raiz Resto	**+, - , * , /** **sqrt** **%**
Operadores Relacionales	
> Mayor < Menor = Igual <> Distinto	**>** **<** **==** **!=**
Operadores Lógicos	
NO Negación Y Conjunción O Disyunción	**!** **&&** **||**

Sentencias simples

SEUDOCÓDIGO	LENGUAJE C
Sentencias de Entrada Salida	
Leer <nombre de variable, ., . > Escribir <nombre de variable, ., . >	scanf(" formato ",**&**nombre de variable, . , .) printf(" formato ", nombre de variable, . , .)

Estructuras Iterativas

SEUDOCÓDIGO	LENGUAJE C
Hacer acción 1 acción 2 ..... acción n **Mientras(<condición>)**	**do** < Sentencia simple o estructurada>**;** **while(<condición>)**
Para v Desde vi Hasta vf acción 1 acción 2 acción n **FinPara**	**for (exp1; exp2; exp3)** < Sentencia simple o estructurada>**;** **exp1:** inicializa la variable de control **exp2:** condición que permite que el ciclo se continúe ejecutando **exp3:** expresión aritmética que identifica el incremento o decremento de la variable de control
Mientras (<condición>) acción 1 acción 2 acción 3 ..... acción n **Fin Mientras**	**while (<condición>)** < Sentencia simple o estructurada>**;**

Ejecución de un programa

Brevemente detallaremos las etapas que se producen antes de ejecutar un programa, ya que todas ellas contribuyen a la obtención del programa ejecutable

Ellas son: edición, compilación, enlace, carga y ejecución

Edición: esta etapa consiste en la escritura de tu programa C, utilizando un programa editor. Este programa, que se conoce como **programa Fuente,** deberá ser almacenado en disco con un nombre que lo identifique para su compilación. Los programas fuentes se almacenan o graban con extensión **.c** o **.cpp** según se utilice lenguaje **C** ó **C++ (C plus plus)**

Compilación: El proceso de compilación consiste en la traducción del programa fuente a código binario, para que la computadora pueda interpretar nuestras órdenes. Se realiza dando la orden compilar del editor.

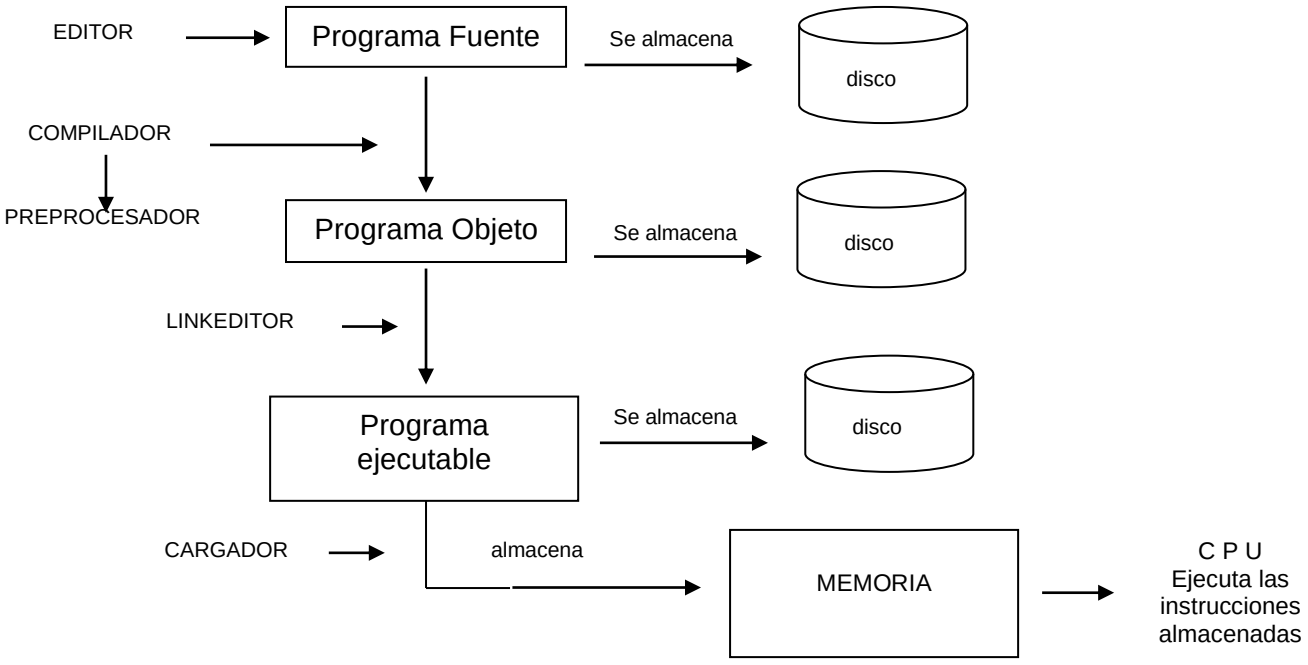
Durante esta etapa, se detectan los **errores sintácticos** cometidos por el programador, cuando no ha respetado las reglas de sintaxis del lenguaje. En este caso, es necesario volver a editar el programa, corregir los errores señalados y compilar nuevamente. Este proceso se repite hasta que el programa ha sido depurado completamente, obteniéndose el **programa objeto** que es almacenado automáticamente con la extensión **obj**.

Es importante destacar que, el compilador invoca automáticamente a un programa ***preprocesador*** que obedece directrices especiales que indican ciertas operaciones que deben realizarse antes de la compilación. En general, esas operaciones consisten en la inclusión de otros archivos en el programa a compilar o en el reemplazo de símbolos especiales con textos de programa. Este es el caso de la directiva **# include < stdio.h>**, que indica al preprocesador de C que debe incluir el archivo stdio.h. Este archivo contiene información que el compilador necesita para compilar las funciones estándar de entrada/salida scanf y printf.

Enlace: El programa objeto no es ejecutable, esto se debe a que generalmente los programas contienen referencias a funciones definidas en otro lugar, una biblioteca estándar o en bibliotecas definidas por el programador. El enlazador es el que se encarga de vincular el código objeto con las bibliotecas, obteniendo así el **programa ejecutable,** que es almacenado con extensión **exe**.

Carga: el cargador toma el programa ejecutable del disco y los transfiere a memoria.
Finalmente la computadora, bajo el control de la CPU toma cada una de las instrucciones y las ejecuta.

El siguiente es un esquema de las etapas descriptas:



Anexo II: Fundamentos matemáticos para análisis de eficiencia

Este anexo contiene algunos fundamentos e instrumentos matemáticos que permitirá realizar el análisis de la eficiencia de los diferentes algoritmos

Series Aritméticas y Geométricas

Se llama sucesión a una disposición ordenada de los elementos de un conjunto numérico.

$\{ a_1 , a_2, a_3 \ldots . a_n \}$ es una sucesión finita

$\{a_1, a_2, a_3, \ldots a_n ....\}$ una sucesión infinita.

Se llama serie finita a la siguiente expresión:

$$a_1 + a_2 + a_3 + \ldots . + a_n = \sum_{i=1}^{i=N} a_i$$

En el caso de que los elementos sucesivos de una serie difieran en una cantidad constante, la serie se llama ***aritmética.***

$a_1, \quad a_1+d, \quad a_1+2d, \ldots . , \quad a_1(N-1)d$

Corresponden a N términos de una serie aritmética.

Donde a_1 representa el primer término y d la diferencia entre términos sucesivos.

2+ 5 + 8 + 11 + 14 +17+ 20 + 23 + 26+ 29

y

50 + 45 + 40 + 35 +30 +25

son ejemplos de series aritméticas de diferencias d=3 y d= -5 respectivamente

Si el cociente entre elementos sucesivos de una serie es una constante, la serie se llama ***geométrica***:

$a_1, \quad a_1 r, \quad a_1 r^2, \ldots . , \quad a_1 r^{N-1}$

Corresponden a N términos de una serie geométrica. a_1 representa el primer término y **r** la razón entre términos sucesivos.

2+ 6 + 18 + 54 + 172 + 516

y

-1/25 + 1/5 - 1 + 5 - 25 + 125 - 625 + 3125

son ejemplos de series aritméticas de razón r=3 y r= -5 respectivamente

Algunas propiedades de las series aritméticas

1. Propiedad de los términos equidistante de los extremos de una Serie Aritmética

Sean a_1, a_1+d, a_1+2d ,…. , $a_{1+}(N-2)d$, $a_{1+}(N-1)d$ los términos de una serie aritmética.

Como se observa el último término $a_n = a_{1+}(N-1)d$, es igual al primero, sumado el producto de **d**, diferencia entre términos, por la cantidad de términos que le preceden.
Esta propiedad se puede generalizar para términos equidistantes:

Sea la serie aritmética:

a_1, a_1+d, a_1+2d , $\mathbf{a_j}$,….$\mathbf{a_k}$, $a_{1+}(N-2)d$, $a_{1+}(N-1)d$

Si $\mathbf{a_j}$ y $\mathbf{a_k}$ son dos términos equidistantes de los extremos entonces significa que si **e** términos preceden por ejemplo a $\mathbf{a_j}$ también **e** términos siguen a $\mathbf{a_k}$.

Esto puede expresarse:

$$\mathbf{a_j} = \mathbf{a_1} + e.d$$

y $$\mathbf{a_k} = \mathbf{a_n} - e.d$$

Sumando miembro a miembro estas igualdades:

$$\boxed{\mathbf{a_j + a_k = a_1 + a_n}} \qquad \mathbf{(1)}$$

La suma de los términos equidistantes de los extremos es igual a la suma de los extremos.

2. Suma de los términos de una serie aritmética

Sea $\mathbf{S_n}$ la suma de los elementos de una serie aritmética:

$$\mathbf{S_n = a_1 + a_2 + a_3 + \ldots + a_n}$$ suma que también puede expresarse:

$$\mathbf{S_n = a_n + a_{n-1} + \ldots + a_1}$$

Sumando ambos miembros de las igualdades anteriores:

$$\mathbf{2\,S_n = (a_1 + a_n) + (a_2 + a_{n-1}) + \ldots + (a_n + a_1)}$$

Cada uno de los binomios del segundo término es la suma de elementos equidistantes de los extremos, que por la propiedad **(1)** son iguales a: $\mathbf{a_1 + a_n}$.
Entonces:

$$\mathbf{S_n = (a_1 + a_n)\, n / 2} \quad \mathbf{(2)}$$

A partir de esta propiedad se puede inferir que:

$$\sum_{i=1}^{N} i = 1+2+ \ldots + N = (1+N).\ N/2$$

$$\boxed{\sum_{i=1}^{N} i = \mathbf{(1+N)\ N/2}}$$

Anexo III Solución a Actividad 1 propuesta en Unidad 4

Actividad 1: Calcular el orden de eficiencia del siguiente algoritmo que posee tres iteraciones anidadas

```
Algoritmo Iteraciones
Comienzo
 entero i, j, k,c
  1. c=0
  2. Para i desde 1 hasta N-1
  3.   Para j desde i+1 hasta N
  4.      Para k desde 1 hasta j
  5.         Escribir i* j*k
  6.           c=c+1
  7.       Finpara
  8.   Finpara
  9. Finpara
 10. Escribir c
Fin
```

El cálculo consiste en resolver las siguientes sumatorias, utilizando las propiedades de series, demostradas en el Anexo I:

$$\sum_{i=1}^{N-1} \left(\sum_{j=i+1}^{N} \left(\sum_{k=1}^{J} \right) \right)$$

↓

bucle más interno, que corresponde a las líneas 4 a 7,
las líneas 5 y 6 de este bucle son O(1) y deben ejecutarse **j** veces.

Veamos entonces cuánto vale J:
El bucle correspondiente a las líneas 3 a 8 se ejecuta: $1/2 (N^2 + N - i^2 - i)$ como se demuestra a continuación:

$$\sum_{j=i+1}^{N} j = (i+1) + (i+2) + \ldots\ldots\ldots + N =$$

por propiedad de series $S_n=(a1 + an) n / 2$ (Ver demostración en Anexo II **)** y teniendo en cuenta que son N-i términos:

$$\sum_{j=i+1}^{N} j = (i+1+ N)*(N-i) /2 = 1/2 (i*N +N+ N^2 - i^2 - i -i*N)= 1/2 (N^2 + N - i^2 - i)$$

Finalmente, el bucle exterior se ejecutará:

$$\sum_{i=1}^{N-1} i = \sum_{i=1}^{N-1} 1/2 (N^2 + N - i^2 - i) =1/2 \left(\sum_{i=1}^{N-1} N^2 + \sum_{i=1}^{N-1} N - \sum_{i=1}^{N-1} i^2 - \sum_{i=1}^{N-1} i \right)$$

Aplicando propiedades de series

$$\sum_{i=1}^{N} i = (1+N) * N/2$$

y

$$\sum_{i=1}^{N} i^2 = N*(N+1)*(2*N+1) /6$$

$$= 1/2*[N^2*(N -1) + N*(N-1) – (N *(N + 1)*(2N+1) /6) - (N + 1) *N/2] =$$

$$= 1/6*(2*N^3- 3 *N^2 – 5*N) = \mathbf{O(N^3)}$$

Anexo IV: Cuantificadores y Reglas de Verificación

Este anexo contiene algunos informacion que amplia y fortalece lo desarrollado en la Unidad 5

- **Cuantificador universal:**

$$\forall (\quad x : P(x))$$

donde *P*(*x*) representa un predicado que "depende" de *x*

Esto significa que **para toda** variable **x** correspondiente al rango especificado**,** se satisface el predicado **P**

Este cuantificador se puede trabajar de dos maneras, indicando el tipo de la variable o su rango:

• Indicando **el tipo** de la variable cuantificada

$$(\forall x : x \in T : P(x))$$

***Por ejemplo*: (∀ x : x ∈ Z : x < y+2)**

• Indicando el **rango** de la variable cuantificada

$$(\forall x : R(x) : P(x))$$

donde *R*(*x*) es un predicado que define el rango de la variable *x,* es decir representa el universo donde esa variable toma valores, P(x) es también un predicado.

***Ejemplo*: (∀ x : 0 < x < 45 : x < y+2)**

- **Cuantificador existencial**

$$(\exists x : P(x))$$

donde

$$(\exists x : x \in T : P(x)) \quad \text{o} \quad (\exists x : R(x) : P(x))$$

Esto significa que **para alguna** variable **x** correspondiente al rango especificado**,** se satisface el predicado **P.**

- **Cuantificador Sumatoria**

$$(\sum i : R(i) : e[i])$$

Esta expresión indica la suma de los elementos e(i) para valores de i que satisfacen el rango **R(i)** .
Cuando el rango de la variable i es vacío, la sumatoria es 0, el elemento neutro de la suma.

- **Cuantificador Producto**

$$(\prod i : R(i): e[i])$$

Esta expresión indica el producto de las valores e(i) para valores de i que satisfacen el rango **R(i)**
Cuando el rango de la variable i es vacío, el producto es el valor 1, el elemento neutro del producto.

- **Cuantificador Máximo y Mínimo**

$$(\max i : R(i): e[i]) \qquad (\min i : R(i): e[i])$$

Se utilizan para expresar el máximo o el mínimo sobre una serie de valores
El cuantificador evalúa e(i) para cada valor de i en el rango R(i) y devuelve el máximo o el mínimo de estas evaluaciones. Para que siempre pueda devolver un valor, el rango de este cuantificador no puede ser vacío.
Ejemplos: Dada la siguiente declaración entero v [*N*], donde v es un arreglo de componente enteras y $N \geq 1$, formalizar las siguientes aserciones

• *x* aparece como componente de *v*.

Podemos utilizar el cuantificador existencial, para decir que existe una posición del vector donde aparece el valor *x*:

$$(\exists i : 0 \leq i \leq N\text{-}1 : v[i] == x).$$

• *v* se anula en algún punto.

$$(\exists i : 0 \leq i < N : v[i] == 0).$$

• *v* tiene valores positivos en todas sus componentes.

$$(\forall i : 0 \leq i \leq N\text{-}1 : v[i] > 0).$$

• *x* aparece una sola vez como componente de *v*.

Utilizando el cuantificador existencial y el universal podemos escribir

$$(\exists i : 0 \leq i < N : v[i] == x) \wedge (\forall j : 0 \leq j < N \wedge j \neq i : v[j] \neq x)).$$

• *x* es la suma de las variables de *v*.

$$x = (\sum i : 0 \leq i < N : v[i]).$$

• *x* es el máximo de las componentes de *v*.

Esta aserción se puede expresar utilizando los operadores universal y existencial:

$$(\exists i : 0 \leq i \leq N\text{-}1 : x == v[i]) \wedge (\forall j : 0 \leq j \leq N\text{-}1 : v[j] \leq x).$$

También se puede utilizar el cuantificador max, de la siguiente forma general:

$x = (\max i : 0 \leq i \leq N\text{-}1 : v[i])$. Siendo el rango $0 \leq i \leq N\text{-}1$ no vacío

• *m* es el menor índice de *v* que contiene el valor *x*.

$$m = (\min i : 0 \leq i \leq N\text{-}1 \wedge v[i] == x : i).$$

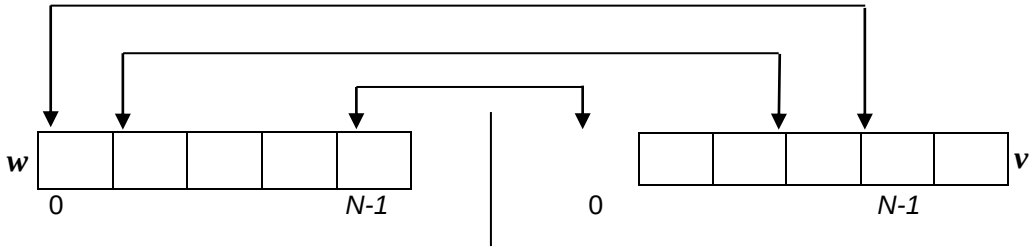
• El valor de cada componente de *v* es el doble de su índice.

Podemos expresarlo, con un cuantificador universal, de la siguiente manera:

$$(\forall i : 0 \leq i \leq N\text{-}1 : v[i] == 2 * i).$$

• *w* es la imagen especular de *v*.

La imagen especular significa que w es la imagen reflejada en un espejo. Por tanto los elementos señalados mediante las flechas deben ser iguales.



Se debe comparar la primera componente de *w* (w [0]) con la última de *v* (*v* [N-1]) , la segunda de *w* (w [1]) con la penúltima de *v* [N-2])
En general la posición *i* debe compararse con la posición *N* - *i* - 1.

Si utilizamos un cuantificador universal, la especificación es:

$(\forall i : 0 \leq i \leq N-1 : w[i] == v[N - i - 1])$

Ejercicios propuestos
Formalizar las siguientes aserciones

- Todas las componentes del vector V son pares
- Cada componente del vector V es la suma de su antecesor y su sucesor
- Todas las componentes del vector V son múltiplos de la anterior
- El vector V tiene alguna componente negativa

Reglas básicas de Verificación

Las siguientes son las reglas generales, es decir que pueden aplicarse independientemente del tipo de sentencias del programa, que se utilizarán en este libro.

Fortalecimiento de la Precondición

$$\frac{\{P\}\ S\ \{Q\} \qquad \{P_0\} \Rightarrow \{P\}}{\{P_0\}\ S\ \{Q\}}$$

Si {P} es una precondición y un conjunto de aserciones $\{P_0\}$ verifica $\{P_0\} \Rightarrow \{P\}$, se dice que $\{P_0\}$ es un fortalecimiento de la precondición {P}

La regla indica que si un código es correcto para una precondición {P}, entonces permanecerá correcto si se refuerza {P}

Recordemos que el fortalecimiento de una aserción disminuye el número de estados que la satisfacen.

Debilitamiento de la Postcondición

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\} \Rightarrow \{Q_0\}}{\{P\}\ S\ \{Q_0\}}$$

Se dice que $\{Q_0\}$ es un debilitamiento de la postcondición $\{Q\}$

Regla de la Conjunción

Conjunción de Precondiciones

Si un programa S con postcondición {Q} satisface las precondiciones {P1} y {P2} entonces es válida la regla:

$$\frac{\{P_1\}\ S\ \{Q\} \qquad \{P_2\}\ S\ \{Q\}}{\{P_1 \wedge P_2\}\ S\ \{Q\}}$$

Conjunción de Postcondiciones

Si un programa S con precondición {P} satisface las postcondiciones $\{Q_1\}$ y $\{Q_2\}$ entonces es válida la regla:

$$\frac{\{P\}\ S\ \{Q_1\} \qquad \{P\}\ S\ \{Q_2\}}{\{P\}\ S\ \{Q_1 \wedge Q_2\}}$$

Conjunción de Pre y Postcondiciones

$$\frac{\{P_1\}\ S\ \{Q_1\} \qquad \{P_2\}\ S\ \{Q_2\}}{\{P_1 \wedge P_2\}\ S\ \{Q_1 \wedge Q_2\}}$$

Caso Particular

$$\frac{\{True\}\ S\ \{Q_1\} \qquad \{P_2\}\ S\ \{Q_2\}}{\{P_2\}\ S\ \{Q_1 \wedge Q_2\}}$$

Regla de la Disyunción

$$\frac{\{P\}\ S\ \{Q_1\} \qquad \{P\}\ S\ \{Q_2\}}{\{P\}\ S\ \{Q_1 \vee Q_2\}}$$

$$\frac{\{P_1\}\ S\ \{Q\} \qquad \{P_2\}\ S\ \{Q\}}{\{P_1 \vee P_2\}\ S\ \{Q\}}$$

$$\frac{\{P_1\}\ S\ \{Q_1\} \qquad \{P_2\}\ S\ \{Q_2\}}{\{P_1 \vee P_2\}\ S\ \{Q_1 \vee Q_2\}}$$

Caso Particular

$$\frac{\{True\}\ S\ \{Q_1\} \qquad \{P_2\}\ S\ \{Q_2\}}{\{True\}\ S\ \{Q_1 \vee Q_2\}}$$