

Aprende a programar

Autor: Maria Elena de Lobos

[\[Ver curso online\]](#)

Presentación del curso

Aprende a través de este curso los conceptos básicos de la lógica computacional, algoritmos, diagramas de flujo y su codificación realizado en el lenguaje de programación, el cual dará la pauta para establecer los fundamentos de programación.

Comprende y aplica los conceptos generales de la programación y lenguajes de programación, de lógica y cómo utilizarlo para deducir las tablas de verdad, además de otra serie de conceptos esenciales para ser un buen programador.

Visita más cursos como este en mailxmail:

[<http://www.mailxmail.com/cursos-informatica>]

[<http://www.mailxmail.com/cursos-programacion>]



¡Tu opinión cuenta! Lee todas las opiniones de este curso y déjanos la tuya:

[<http://www.mailxmail.com/curso-aprende-programar/opiniones>]

Cursos similares

Cursos	Valoración	Alumnos	Vídeo
Procesos en C. Sincronización (segunda parte) Curso de informática sobre sincronización de procesos en C, que, en su segunda parte, desarrolla el tema de la comunicación entre los procesos de un sistema informático; ... [21/10/08]	★★★★★	1.252	
Metodología de la Programación Este curso gratis le proporcionará, a modo de iniciación, algunos de los pasos a seguir para aprender a programar. Los bucles o los operadores lógicos son algunos de los c... [23/05/03]	★★★★★	51.893	
PHP y MySQL. Aplicaciones Web: HTML II (tercera parte) Programación de aplicaciones Web con PHP y MySQL. Ahora continuaremos con el estudio de las páginas Web HTML. Estudiaremos las listas en HTML. ... [02/12/08]	★★★★★	1.015	
Arquitectura de computadores: diseño, coste y rendimiento Nuestro curso, como su título lo indica, es una introducción a la arquitectura de computadores, en el que desarrollamos una presentación técnica sobre el diseño, coste y ... [24/07/08]	★★★★☆	2.676	
Algoritmos y lenguaje C Este curso pretende dar a conocer las explicaciones más simples del trabajo de los ordenadores. Puedes descubrir cada paso que realizan, cómo lo realizan, qué es l... [13/01/06]	★★★★★	18.227	

1. Conceptos básicos de programación.

[<http://www.mailxmail.com/curso-aprende-programar/conceptos-basicos-programacion>]

Es un proceso para convertir especificaciones generales de un sistema en instrucciones utilizables por la máquina, que produzcan los resultados deseados. Se le conoce también como desarrollo de software.

PROGRAMA

Es una lista de instrucciones que la computadora debe seguir para procesar datos y convertirlos en información. Las instrucciones se componen de enunciados usados en lenguajes de programación como Basic, Pascal o C.

CARACTERÍSTICAS DEL PROGRAMA:

- Debe ser confiable y funcional
- Advertir errores de entrada obvios y comunes
- Documentado adecuadamente
- Ser comprensible
- Codificado en el lenguaje apropiado

DATOS:

Son las características propias de cualquier entidad. Por ejemplo: los datos de una persona como su edad, fecha de nacimiento, domicilio, número de teléfono, etc.

INFORMACIÓN:

Es el conocimiento relevante producido como resultado del procesamiento de datos y adquirido por la gente para realzar el entendimiento y cumplir ciertos propósitos.

PROCESAMIENTO DE DATOS:

Consiste en la recolección de datos de entrada que son evaluados y ordenados para ser colocados de manera que produzcan información útil.

ACTIVIDADES DEL PROCESAMIENTO DE DATOS

- I. Captura de datos de entrada
- II. Manejo de los datos (incluye clasificación, ordenación, cálculo y sumariación de éstos)
- III. Administración de la salida resultante.

2. Pasos del desarrollo de software.

[<http://www.mailxmail.com/curso-aprende-programar/pasos-desarrollo-software>]

1. Especificación del programa
2. Diseño del programa
3. Codificación del programa
4. Prueba
5. Documentación
6. Mantenimiento

1. Especificación del programa

Se conoce también como definición del problema o análisis del programa. En este paso se determinan la información inicial para la elaboración del programa. Es donde se determina qué es lo que debe resolverse con el computador, de qué presupuestos se debe partir... en definitiva, el planteamiento del problema.

Se requieren cinco tareas:

a. Determinación de objetivos del programa.

Debe definirse claramente los problemas particulares que deberán ser resueltos o las tareas que hay que realizar, esto nos permitirá saber qué es lo que se pretende solucionar y nos proporcionará información útil para el planeamiento de la solución.

b. Determinación de la salida deseada.

Los datos seleccionados deben ser arreglados en una forma ordenada para producir información. Esta salida podría ser una salida de impresión o de presentación en el monitor.

c. Determinación de los datos de entrada.

Una vez identificada la salida que se desea, se pueden determinar los datos de entrada y la fuente de estos datos. Los datos deben ser recolectados y analizados.

d. Determinación de los requerimientos de procesamiento.

Aquí se definen las tareas de procesamiento que deben desempeñarse para que los datos de entrada se conviertan en una salida.

e. Documentación de las especificaciones del programa.

Es importante disponer de documentación permanente. Deben registrarse todos los datos necesarios para el procesamiento requerido. Esto conduce al siguiente paso del diseño del programa.

2 . Diseño del programa

Es diseñar cualquier sistema nuevo o las aplicaciones que se requieren para satisfacer las necesidades. Esta actividad se debe dividir en:

- Operaciones de entrada/salida
- Cálculos
- Lógica/ comparación
- Almacenamiento/ consulta

En este paso se genera una solución con técnicas de programación como diseño descendente de programas, pseudocódigos, flujogramas y estructuras lógicas.

3. Codificación del programa

Es la generación real del programa con un lenguaje de programación. En esta etapa

se hace uso de la lógica que desarrolló en el paso del diseño del programa para efectivamente generar un programa. Se debe seleccionar el lenguaje apropiado para resolver el problema.

4. Prueba y depuración del programa

Depurar es correr el programa en una computadora y corregir las partes que no funcionan. En esta fase se comprueba el funcionamiento de cada programa y esto se hace con datos reales o ficticios. Cuando los programas están depurados, se prueban. Cuando los programas se depuran, se pueden encontrar los siguientes errores:

- a) Errores de sintaxis o de compilación
- b) Errores de ejecución
- c) Errores de lógica
- d) Errores de especificación.

a) Errores de sintaxis o de compilación

Es una violación de las reglas del lenguaje de programación. Son más fáciles de corregir, ya que son detectados por el compilador (posible error de escritura), el cual dará información sobre el lugar donde está y la naturaleza de cada uno de ellos mediante un mensaje de error.

b) Errores de Ejecución

Se deben generalmente a operaciones no permitidas como dividir por cero, leer un dato no numérico en una variable numérica, exceder un rango de valores permitidos, etc. Se detectan porque se produce una parada anormal del programa durante su ejecución.

c) Errores de Lógica

Corresponden a la obtención de resultados que no son correctos y la única manera de detectarlos es realizando suficientes pruebas del programa. Son los más difíciles de corregir, no sólo por la dificultad de detectarlos, sino porque se deben a la propia concepción y diseño del programa.

d) Errores de Especificación

Es el peor tipo de error y el más difícil de corregir. Se deben a mal diseño del programa posiblemente por mala comunicación usuario programador y se detectan cuando ya se ha concluido el diseño e instalación del programa, lo cual puede implicar repetir gran parte del trabajo realizado.

Prueba : :

Consiste en verificar la funcionalidad del programa a través de varios métodos para detectar errores posibles.

Métodos de Prueba:

Chequeo de escritorio

Prueba manual de datos de muestra

Intento de traducción

Prueba de datos de muestra en la computadora

Prueba por un grupo selecto de usuarios potenciales.

a. Chequeo de Escritorio:

El programador se sienta frente a un escritorio y corrige una impresión del programa. Revisa el listado línea por línea en busca de errores de sintaxis y lógica.

b. Prueba manual de datos de muestra:

Se corre el programa en forma manual aplicando datos tanto correctos como incorrectos para comprobar que funciona correctamente.

c. Intento de Traducción:

El programa corre en una computadora usando un programa traductor para convertirlo a lenguaje de máquina. Para ello debe estar ya libre de errores de sintaxis, de lo contrario serán identificados por el programa de traducción.

d. Prueba de datos de muestra en la computadora:

Después del intento de traducción y corregidos los errores de sintaxis, se procede a buscar errores de lógica utilizando diferentes datos de muestra.

e. Prueba por un grupo selecto de usuarios potenciales:

Esto se conoce como prueba beta. Se trata por lo general del paso final en la prueba de un programa. Usuarios potenciales ponen a prueba el programa y ofrecen retroalimentación.

5. Documentación del programa

Consiste en describir por escrito a nivel técnico los procedimientos relacionados con el programa y su modo de uso. También se debe documentar el programa para que sea más entendible.

¿Para quiénes son la documentación?

- Usuarios (Digitadores)
- Operadores
- Programadores
- Analistas de sistemas

Documentos que se elaboran:

Manual de Usuario y Manual del Analista.

A los **usuarios** se les elabora un manual de referencia para que aprendan a utilizar el programa. Esto se hace a través de capacitaciones y revisión de la documentación del manual de usuario. El manual del usuario no está escrito a nivel técnico sino al de los distintos usuarios previstos y explica en detalle cómo usar el programa: descripción de las tareas que realiza el programa, instrucciones necesarias para su instalación puesta en marcha y funcionamiento, recomendaciones de uso, menús de opciones, método de entrada y salida de datos, mensajes de error, recuperación de errores, etc.

A los **operadores** por si se presentan mensajes de error, sepan cómo responder a ellos. Además que se encargan de darle soporte técnico al programa.

A los **programadores** a través del manual del analista para que recuerden aspectos de la elaboración del programa o en caso que otras personas puedan actualizarlo o modificarlo (darle mantenimiento) y no son necesariamente las personas que lo diseñaron. Es por ello, que la documentación debe contener algoritmos y flujogramas de los diferentes módulos que lo constituyen y las relaciones que se establecen entre ellos; listados del programa, corridas, descripción de variables que se emplean en cada módulo, cuáles son comunes a diferentes módulos y cuáles locales; descripción de los ficheros de cada módulo y todo lo que sea de importancia para un programador.

A los **analistas de sistemas** que son las personas que deberán proporcionar toda la información al programador. Estos se encargan de hacer una investigación previa de cómo realizar el programa y documentar con las herramientas necesarias para que

el programador pueda desarrollar el sistema en algún lenguaje de programación adecuado.

6. Mantenimiento del programa

Es el paso final del desarrollo del software. Alrededor del 75% del costo total del ciclo de vida de un programa se destina al mantenimiento. El propósito del mantenimiento es garantizar que los programas en uso estén libres de errores de operación y sean eficientes y efectivos.

3. La lógica como aspecto fundamental de la programación.

[<http://www.mailxmail.com/...so-aprende-programar/logica-como-aspecto-fundamental-programacion>]

Es la capacidad de pensar racionalmente acerca de soluciones alternativas y los resultados de aplicarlas, y por lo tanto, de hacer elecciones inteligentes.

Definiciones de Lógica:

- Es el estudio crítico del razonamiento y tiene un valor teórico y práctico.
- Es el estudio de los métodos y principios usados al distinguir entre los argumentos correctos (buenos) y los argumentos incorrectos (malos).
- En un sentido amplio, es el estudio del correcto razonamiento.

La lógica se remonta a la época de Aristóteles en el siglo IV antes de Cristo. En lógica, Aristóteles desarrolló reglas para establecer un razonamiento encadenado que, si se respetaban, no producirían nunca falsas conclusiones si la reflexión partía de premisas verdaderas (reglas de validez). En el ejemplo más famoso, "Todos los humanos son mortales" y "Todos los griegos son humanos", se llega a la conclusión válida de que "Todos los griegos son mortales"

¿Qué es Lógica?

Lógica Deductiva

Es la que se encarga de determinar la validez o invalidez de los argumentos. Permite proporcionar la simbología que nos servirá para facilitar el desarrollo de la capacidad de análisis de problemas, para obtener una solución posible.

Argumentos.

Cuando el razonamiento se expresa con palabras, recibe el nombre de "argumento". Un argumento es un grupo cualquiera de proposiciones o enunciados que forman premisas y conclusiones. Este puede constar de varias premisas pero de una sola conclusión.

Premisas y Conclusión

Las premisas de un argumento son proposiciones afirmadas como fundamento o razones para aceptar una conclusión. La conclusión es la proposición afirmada que se basa en las otras proposiciones o premisas. Una proposición puede ser premisa en un argumento y conclusión en otro. Por ejemplo: "Todos los hombres son mortales" Aparece como premisa en el siguiente argumento:

"Todos los hombres son mortales"

"Sócrates es un hombre"

"Por lo tanto, Sócrates es mortal".

Y como conclusión en el siguiente argumento:

" Todos los animales son mortales"

"Todos los hombres son animales"

"Luego, todos los hombres son mortales"

Expresiones como "por tanto", "por lo cual", "de ello se deduce", sirven para introducir la conclusión de un argumento, en tanto que "pues" y "porque" se emplean para introducir las premisas.

Otro Ejemplo:

"Todos se aburren en la conferencia".

"Ninguno de los que se aburren presta atención".

"Por consiguiente, ninguno de los asistentes está prestando atención".

Hay dos condiciones que debe satisfacer un argumento para establecer la verdad de su conclusión: Debe ser válido y todas sus premisas deben ser verdaderas.

Enunciado Simple:

Es el que no contiene otro enunciado como parte componente. Ej. "Las rosas son rojas"

Enunciado compuesto:

Es el que se compone de varios enunciados. Ej. "Las rosas son rojas y las violetas son azules". Cuando los enunciados se unen por la conjunción Y, se denominan

Enunciados conjuntos.

Cuando los enunciados se unen por el conector o, se denominan **Enunciados disyuntos**. De aquí surgen las siguientes tablas de verdad:

X	Y	And
V	V	V
V	F	F
F	V	F
F	F	F

X	Y	OR
V	V	V
V	F	V
F	V	V
F	F	F

X	NOT (X)
V	F
F	V

Ejemplos de aplicación de estas tablas de verdad:

Si A y B son valores verdaderos y P y Q son falsos, cuál es el valor de verdad de las siguientes expresiones:

1. not (A or B) and (P and not Q)

Solución: Evaluamos primero los paréntesis, comenzando con el primero. En la tabla OR buscamos a qué equivalen Verdadero or Verdadero (porque A y B son verdaderos según el enunciado). Obtenemos que es Verdadero. Como la expresión está negada, su valor opuesto es Falso. Al evaluar el segundo paréntesis, Q es falso y al negarlo nos queda verdadero. Luego, si P es falso nos queda Falso and Verdadero, y esto es igual a Falso.

Entonces:

not (A or B) and (P and not Q)

not (V or V) and (F and not F)

not (V) and (F and V)

F and F

Falso

Notación:

Not se representa con ~

And se representa con ^

y Or se representa con v

4. Concepto de lenguaje de programación.

[<http://www.mailxmail.com/curso-aprende-programar/concepto-lenguaje-programacion>]

Es un conjunto de símbolos junto a un conjunto de reglas para combinar dichos símbolos que se usan para expresar programas. Constan de un léxico, una sintaxis y una semántica.

¿Qué conoces tu por léxico, sintaxis y semántica?

Léxico : Conjunto de símbolos permitidos o vocabulario

Sintaxis : Reglas que indican cómo realizar las construcciones del lenguaje

Semántica: Reglas que permiten determinar el significado de cualquier construcción del lenguaje.

Tipos de lenguajes: Atendiendo al número de instrucciones necesarias para realizar una tarea específica podemos clasificar los lenguajes informáticos en dos grandes bloques:

- bajo nivel
- alto nivel

Lenguaje de bajo nivel

Es el tipo de lenguaje que cualquier computadora es capaz de entender. Se dice que los programas escritos en forma de ceros y unos están en lenguaje de máquina, porque esa es la versión del programa que la computadora realmente lee y sigue.

Lenguajes de alto nivel

Son lenguajes de programación que se asemejan a las lenguas humanas usando palabras y frases fáciles de entender.

- En un lenguaje de bajo nivel cada instrucción corresponde a una acción ejecutable por el ordenador, mientras que en los lenguajes de alto nivel una instrucción suele corresponder a varias acciones.

- Características de los lenguajes de alto nivel:

Son independientes de la arquitectura física de la computadora. Permiten usar los mismos programas en computadoras de diferentes arquitecturas (portabilidad), y no es necesario conocer el hardware específico de la máquina. La ejecución de un programa en lenguaje de alto nivel, requiere de una traducción del mismo al lenguaje de la computadora donde va a ser ejecutado. Una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje entendible por el computador. Utilizan notaciones cercanas a las usadas por las personas en un determinado ámbito. Se suelen incluir instrucciones potentes de uso frecuente que son ofrecidas por el lenguaje de programación.

Generaciones de Lenguajes :

1. lenguajes de máquina
2. lenguajes ensambladores
3. lenguajes de procedimientos
4. lenguajes orientados a problemas
5. lenguajes naturales

1. Lenguaje de máquina (Primera Generación)

Es el lenguaje que la computadora entiende, su estructura está totalmente adaptada a los circuitos de la máquina y la programación es tediosa porque los datos se representan por ceros y unos. Es de bajo nivel. Es un conjunto de instrucciones codificadas en binario que son capaces de relacionarse directamente con los registros y circuitería del microprocesador de la computadora y que resulta directamente ejecutable por éste, sin necesidad de otros programas intermediarios. Los datos se referencian por medio de las direcciones de memoria donde se encuentran y las instrucciones realizan operaciones simples. Estos lenguajes están íntimamente ligados a la CPU y por eso no son transferibles. (baja portabilidad). Para los programadores es posible escribir programas directamente en lenguaje de máquina, pero las instrucciones son difíciles de recordar y los programas resultan largos y laboriosos de escribir y también de corregir y depurar.

2. Lenguaje ensamblador (Segunda Generación)

Es otro lenguaje de programación de bajo nivel, pero simbólico porque las instrucciones se construyen usando códigos de tipo mnemotécnico, lo cual facilita la escritura y depuración de los programas pero no los acorta puesto que para cada acción se necesita una instrucción. El programa ensamblador va traduciendo línea a línea a la vez que comprueba la existencia de errores. Si localiza alguno da un mensaje de error. Algunas características que lo diferencian del lenguaje de máquina son que permite el uso de comentarios entre las líneas de instrucciones; en lugar de direcciones binarias usa identificadores como total, x, y, etc. Y los códigos de operación se representan por mnemotécnica siempre tienen la desventaja de repertorio reducido de instrucciones, rígido formato para las instrucciones, baja portabilidad y fuerte dependencia del hardware. Tiene la ventaja del uso óptimo de los recursos hardware, permitiendo la obtención de un código muy eficiente. Ejemplo de algunos códigos mnemónicos son: STO para guardar un dato, LOA para cargar algo en el acumulador, ADD para adicionar un dato, INP para leer un dato, STO para guardar información, MOV para mover un dato y ponerlo en un registro, END para terminar el programa, etc. Con la tercera generación avanzamos a los lenguajes de alto nivel, muchos de los cuales se consideran exportables. Esto es, pueden correr en más de un tipo de computadoras, se les puede exportar de una máquina a otra.

3. Lenguaje de procedimientos (Tercera Generación)

Son lenguajes de alto nivel similares al habla humana pero requieren cierta capacitación para su uso.

Ventajas :

a. Independencia de la arquitectura física de la computadora (portabilidad), esto significa que un mismo lenguaje puede funcionar (al menos en teoría) en distintos computadores, por lo que tanto el lenguaje como los programas escritos con él serán transportables de un computador a otro. En la práctica, esta característica resulta limitada por la gran diversidad de versiones y dialectos que se constituyen para cada lenguaje.

b. una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje máquina. Se llaman de procedimientos porque están diseñados para expresar la lógica capaz de resolver problemas generales. Entre estos tenemos:

Basic

Pascal

Cobol

C

Fortran

Para que el lenguaje de procedimientos pueda funcionar debe traducirse a lenguaje de máquina a fin de que la computadora lo entienda. Para ello se han de usar programas traductores que realicen dicho proceso. Tienen la capacidad de soportar programación estructurada.

4. Lenguajes orientados a problemas (4GL)

Resultan más eficaces para la resolución de un tipo de problemas a costa de una menor eficiencia para otros. Requieren poca capacitación especial de parte del usuario. Son considerados de muy alto nivel. Diseñados para resolver problemas específicos.

Incluye: lenguajes de consulta y generador de aplicaciones.

Lenguajes de consulta:

Permiten a no programadores usar ciertos comandos de fácil comprensión para la búsqueda y generación de reportes a partir de una base de datos.

Generador de aplicaciones:

Quiere decir que cuando se diseña uno de estos lenguajes, se tiene en cuenta que su finalidad es la resolución de problemas, prescindiendo de la arquitectura del computador. Contiene varios módulos que han sido preprogramados para cumplir varias tareas.

5. Lenguajes naturales

Lenguajes orientados a aplicaciones en inteligencia artificial, como lisp y prolog.

Dentro de este campo destacan las aplicaciones en sistemas expertos, juegos, visión artificial (Jurassic Park) y robótica. Lisp es un lenguaje para procesamiento de listas y manipulación de símbolos. Prolog es un lenguaje basado en la lógica, para aplicaciones de bases de datos e Inteligencia Artificial.

Podemos decir entonces, que los lenguajes de alto nivel, tienen las ventajas de mayor legibilidad de los programas, portabilidad, facilidad de aprendizaje y facilidad de modificación.

PARA ANALIZAR:

1. ¿Cuál es la diferencia fundamental entre los lenguajes de alto nivel y bajo nivel?
2. Investigar analogías y diferencias entre el código máquina y el lenguaje ensamblador.
3. Buscar información que permita decidir cuáles serían los lenguajes de programación más apropiados para realizar: aplicaciones para gestión de oficinas, complejos cálculos científicos, un sistema experto en medicina, un simulador de vuelo, manipulación de bases de datos, control de un robot industrial.

Sugerencias de ampliación:

Clasificación de los lenguajes de alto nivel

Lenguajes de propósito general:

- PL/1
- BASIC
- C
- PASCAL
- MODULA II
- COBOL
- LOGO
- FORTH
- ADA

Lenguajes de Cálculo científico:

- FORTRAN
- APL
- ADA
- PASCAL
- ALGOL

Lenguajes orientados a la gestión:

- COBOL
- RPG

Lenguajes de simulación en general:

- GPSS
- SIMULA
- MIMIC

En el sistema educativo:

- ASSET
- LOGO
- PASCAL
- C
- BASIC
- MODULA II
- PILOT

Lenguajes orientados a objetos:

- SMALLTALK

Lenguajes interrogativos:

- PROLOG
- DBASE

Lenguajes para Inteligencia Artificial

- LISP
- PROLOG

5. Algoritmos y diagramas de flujo- Intérpretes y compiladores.

[<http://www.mailxmail.com/...de-programar/algoritmos-diagramas-flujo-interpretes-compiladores>]

Los compiladores, los intérpretes y los ensambladores se encargan de traducir lo que haya escrito en lenguaje de alto nivel (código fuente) y lo convierten a código objeto (casi ejecutable).



Compilador

Es un programa que traduce un programa escrito en un lenguaje de alto nivel, por ejemplo C++, en un programa en lenguaje de máquina que la computadora es capaz de entender y ejecutar directamente. Un compilador es un tipo especial de programa, en cuanto a que sus entradas o datos son algún programa y su salida es otro programa. Para evitar confusiones, solemos llamar programa fuente o código fuente al programa de entrada, y programa objeto o código objeto a la versión traducida que el compilador produce. Código se usa frecuentemente para referirse a un programa o a una parte de él, sobre todo cuando se habla de programas objeto.

Ejemplo:

Pascal

Cobol

Fortran

Ada

Código Fuente

Código Objeto

Código Ensamblador

Modula 2

C , C++

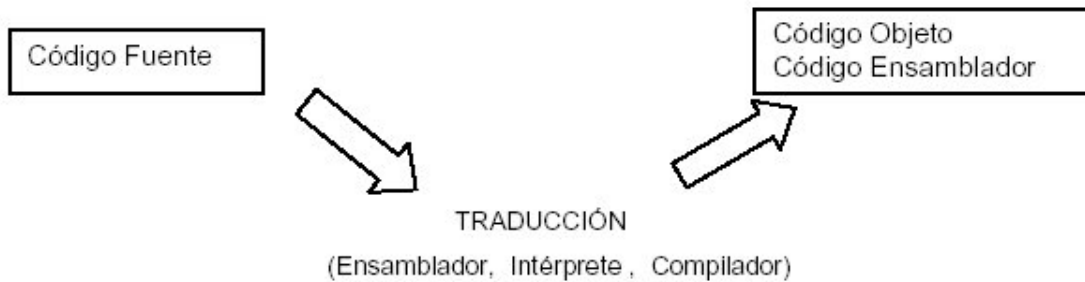
El compilador, informa al usuario de la presencia de errores en el programa fuente, pasándose a crear el programa objeto cuando está libre de errores. El código objeto puede ser ejecutado posteriormente. Una vez traducido un programa, su ejecución es independiente de su compilación. involucra dos pasos en su operación:

1. Convertir código fuente a objeto
2. Ejecutar el código objeto

Ventaja:

Al tener el código objeto, el programa se ejecuta más rápido

Fases de compilación



Análisis: Dependiente del lenguaje. Independiente de la máquina

Sintaxis: Independiente del lenguaje. Dependiente de la máquina.

Intérprete: Es el que permite que un programa fuente escrito en un lenguaje vaya traduciéndose y ejecutándose directamente sentencia a sentencia por la computadora. Convierte uno por uno los enunciados del código fuente a código objeto antes de ser ejecutados. Convierte y ejecuta el programa en línea al mismo tiempo. Ejemplo: Basic estándar.

Ventaja:

Las ventajas de los intérpretes son:

- Resulta más fácil localizar y corregir errores (depuración de programas)
- son más pedagógicos para aprender a programar.
- El programa es más fácil de desarrollar.

Traducen programas de alto nivel. No se genera en la mayoría de los ficheros.

Programa

Fuente

Código

Intermedio

Programa

Objeto

Para cada una de las líneas se ejecuta el siguiente proceso:

1. Análisis de la instrucción de esa línea
2. Traducción de esa línea (si ya está correcta) a código objeto
3. Ejecución de esa línea

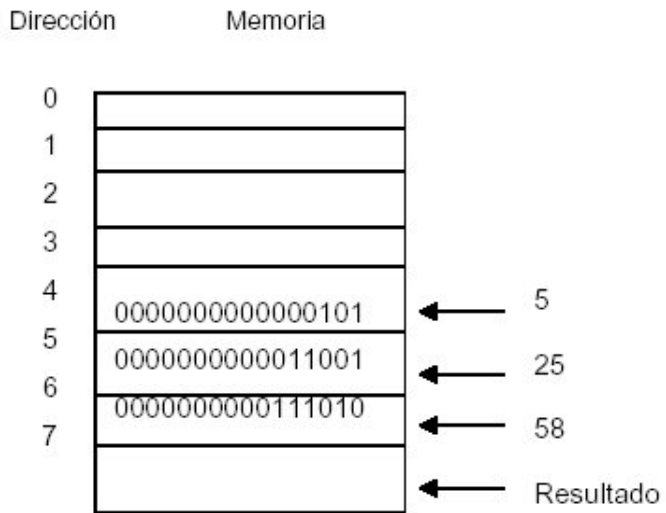
Con el intérprete, cada vez que necesitamos ejecutar el programa tenemos que volver a analizarlo porque no hay código objeto.

Con el compilador, aunque más lenta la traducción, sólo se realiza una vez.

Ejemplo 4

Supongamos que se han almacenado tres valores $5 = 01012$, $25 = 110012$ y $58 = 1110102$ en las posiciones de memoria con direcciones 4,5 y 6.

Queremos multiplicar los dos primeros valores, sumar el tercero y almacenar el resultado en la palabra de memoria 7.



Para llevar a cabo este cálculo, se deben ejecutar las siguientes instrucciones:

Recuperar el contenido de la palabra de memoria 4 y cargarlo en el registro acumulador de la unidad aritmético lógica.

Recuperar el contenido de la palabra de memoria 5 y calcular el producto de este valor y el valor situado en el acumulador.

Recuperar el contenido de la palabra de memoria 6 y sumar su valor con el valor situado en el registro acumulador.

Almacenar el contenido del registro acumulador en la palabra de memoria 7. Para almacenar estas instrucciones en la memoria de la computadora, deben estar representadas en forma binaria. Las direcciones de los datos no presentan problemas, puesto que pueden ser convertidos fácilmente a direcciones binarias:

4 = 100

5 = 101

6 = 110

7 = 111

Las operaciones de cargar, multiplicar, sumar, almacenar y otras instrucciones máquina básicas se representan mediante códigos numéricos, llamados códigos de operación, por ejemplo:

LOAD = 16 = 10000

STORE = 17 = 10001

ADD = 35 = 100011

MULTIPLY = 36 = 100100

SUB = 37 = 100101

DIV = 38 = 100110

Usando parte de una palabra para almacenar el código de operación y otra para la dirección del operando, podemos representar nuestra secuencia de instrucciones en lenguaje máquina como sigue:

1. 0001000000000100

2. 0010010000000101

3. 0010001100000110

4. 0001000100000111

Estas pueden ser almacenadas en cuatro palabras consecutivas de memoria. Cuando se ejecuta el programa, la unidad de control recuperará cada una de las instrucciones, la decodificará para determinar la operación y la dirección del operando, recuperará el operando, y entonces ejecutará la operación requerida, usando la unidad aritmético lógica cuando sea necesario. Los programas para las primeras computadoras tuvieron que ser escritos en lenguaje de máquina. Posteriormente fue posible escribirlos en lenguaje

ensamblador usando códigos nemotécnicos en lugar de códigos de operación numéricos y nombres de variables en lugar de direcciones numéricas.

Por ejemplo la secuencia de instrucciones anteriores se escribiría así:

1. LOAD A
2. MULT B
3. ADD C
4. STORE X

Luego que se crearon los lenguajes de alto nivel, las instrucciones se escribían en forma más entendible para el programador. El ejemplo anterior podría ser como lo siguiente usando C++:

$X = A * B + C$

El producto en la programación se representa por asterisco. En cada uno de estos casos (ensamblador y lenguajes de alto nivel), el compilador traduce cada instrucción del programa en una secuencia de cuatro instrucciones máquina y genera un programa objeto.

Ejercicios Propuestos:

Usando mnemónicos de instrucción y códigos de operación, escriba una secuencia de instrucciones en a) Lenguaje ensamblador b) Lenguaje de máquina, equivalente a las instrucciones de C++:

1. $X = (A - B) * C$
 $X = (A+B) / (C+D)$
 $X = (A + B) - C$

Para las instrucciones máquina, suponga que los valores de A, B, C y D son almacenados en las palabras de memoria 15,16, 17 y 18 respectivamente, y que los valores de X e Y se almacenarán en la palabra de memoria 23 y 24 respectivamente.

6. Conceptualización de los tipos de datos.

[<http://www.mailxmail.com/curso-aprende-programar/conceptualizacion-tipos-datos>]

DATO. Es la expresión general que describe los objetos con los cuales opera el programa. Por ejemplo, la edad y el domicilio de una persona, forman parte de sus datos. Los datos se sitúan en objetos llamados variables.

Las variables son zonas de memoria cuyo contenido cambia durante la fase de procesamiento de información. Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa. Las variables llevan un nombre llamado identificador. Este puede ser una cadena de letras y dígitos, empezando siempre con una letra. Por ejemplo: Pi, curso99, nom_alum, etc.

Los Identificadores son palabras creadas por los programadores para dar nombre a los objetos y demás elementos que necesitamos declarar en un programa: variables, constantes, tipos, estructuras de datos, archivos, subprogramas, etc. En C++ las letras mayúsculas se tratan como diferentes y distintas unas de otras. Por ejemplo, contador, Contador y CONTADOR son tres nombres de identificadores distintos. Un identificador no puede ser igual a una palabra reservada, y no debe tener el mismo nombre que una función, ya sea definida por el usuario o de la biblioteca de C.

Constantes : Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo. Por ejemplo, si se desea obtener un reporte para cada uno de los empleados de una empresa, con sus datos generales, la fecha y cantidad de dinero que recibieron la última semana, el dato fecha puede ser una constante ya que es el mismo para todos.

Expresiones . Son representaciones de un cálculo necesario para la obtención de un resultado. Estas pueden ser valores constantes, funciones o combinaciones de valores, variables, funciones y operadores que cumplen determinadas reglas de construcción de una expresión. Son un conjunto de operadores y operandos que producen un valor. Por ejemplo:

$\text{Cos}(\pi * X) + 12.56 * \text{SQR}(100)$

Un operador es un símbolo o palabra que significa que se ha de realizar cierta acción entre uno o dos valores que son llamados operandos.

Operación de Asignación:

Es el modo de darle valores a una variable. Se representa con el símbolo u operador !, el cual se conoce como instrucción o sentencia de asignación.

Sintaxis: Nombre de la variable \rightarrow ! expresión.

Ejemplo: cociente ! \rightarrow cal1/cal2

Tipo : es el conjunto de valores que puede tomar una variable. Es el conjunto de transformaciones y funciones internas y externas definidas sobre el conjunto de datos. Se tienen dos tipos de datos: Simples como numéricos y alfanuméricos y Estructuras de datos que pueden ser internas o externas.

Tipos de datos Numéricos son aquellos cuyo contenido es una serie de dígitos (0-9) que en conjunto nos proporcionan un valor numérico ya sea entero o real y pueden

ser precedidos de un signo + ó -. Tipos de datos Alfanuméricos son aquellos cuyo contenido son letras del abecedario, números o caracteres especiales o bien una combinación de ellos.

Tipos de Instrucciones:

- | | |
|---------------------------------------|------------------------|
| a) Instrucciones de inicio/fin | (ejemplo inicio - fin) |
| b) Instrucciones de asignación | (ejemplo B ! 7) |
| c) Instrucciones de lectura (entrada) | (ejemplo leer) |
| d) Instrucciones de escritura (salir) | (ejemplo escribir) |
| e) Instrucciones de bifurcación | (ejemplo Goto fin) |

Tipos de Operadores:

- a) Aritméticos (su resultado es un número): potencia, *, / , mod, div, + , -
- b) Relacionales (su resultado es un valor de verdad): =, <, >, <=, >=, <>
- c) Lógicos o Booleanos (su resultado es un valor de verdad): not, and, or
- d) Alfanuméricos : + (concatenación)
- e) Asociativos. El único operador asociativo es el paréntesis () , el cual permite indicar en qué orden deben realizarse las operaciones. Cuando una expresión se encuentra entre paréntesis, indica que las operaciones que están dentro de ellos debe realizarse primero. Si en una expresión se utilizan más de un paréntesis se deberá proceder primero con los que se encuentren más hacia el centro de la expresión.

Jerarquía de Operaciones:

()
signo
Potencia
Producto y división
Div
Mod
Suma y resta
Concatenación
Relacionales
Negación
And
Or

Datos de tipo entero tienen los operadores +, -, *, / , div, mod, abs, sqr, sqrt, ln, exp, sin, cos, tan, pow, etc. Los datos de tipo real tienen los mismos operadores enteros y además trunc, round, int, y otros. La suma y multiplicación de datos de tipo real cumplen la propiedad conmutativa, pero no siempre la asociativa y la distributiva.

Para resolver una expresión aritmética se deben seguir las siguientes reglas:

- Primero se resuelven las expresiones que se encuentran entre paréntesis.
- Se procede aplicando la jerarquía de operadores.
- Al evaluar una expresión, si hay dos operadores con la misma jerarquía, se procede a evaluar de izquierda a derecha.
- Si hay expresiones relacionales, se resuelven primero paréntesis, luego se encuentran los valores de verdad de las expresiones relacionales y por último se aplica jerarquía de operadores lógicos. En caso de haber iguales, proceder de izquierda a derecha.

EJERCICIOS.

Aplicando la jerarquía de los operadores, encontrar el valor de cada una de las siguientes expresiones:

- 1) $4 + 1 * 5 ^ 2 - 1$
- 2) $9 / 3 + 4 ^ 2 - 5 * 1 + 9 / -2 + 3$
- 3) $5 / 2 + 3 - 4 * 5 / 2$
- 4) $(4 + 1) * 5 ^ 2 - 1$
- 5) $17 / 2 + 3 ^ 2 ^ 2 - 2 * 2 / 2$

Aplicando la jerarquía de operadores, encontrar el valor de verdad de cada una de las siguientes expresiones:

1. Not ((M > N and R > S) or (NOT(T < V and S > M))) para M=8, N=9, R=5, S=5 , T=4 y V= 2
2. $(3 * 2 ^ 2 - 4 / 2 * 1) > (3 * 2 ^ -4 / 2 * 1)$ and $(5 > 9 / 3)$

7. Herramientas de programación.

[<http://www.mailxmail.com/curso-aprende-programar/herramientas-programacion>]

Algoritmo : es una serie de operaciones detalladas a ejecutar paso a paso, que conducen a la resolución de problemas. Es un conjunto de reglas para resolver determinado problema describiendo de forma lógica su solución. Cada una de las acciones de que consta un algoritmo es denominada sentencia y éstas deben ser escritas en términos de cierto lenguaje comprensible para el computador, que es el lenguaje de programación. Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en que han de ser ejecutadas.

Criterios que debe satisfacer un algoritmo (características):

1. Entrada. Son cero o más cantidades las cuales son externamente sustituidas.
2. Salida. Al menos una cantidad es producida.
3. Exactitud/precisión. Cada instrucción debe ser clara y sin ambigüedad.
4. Finito. Terminará después de un número finito de pasos.
5. Eficiente. Cada instrucción puede ser verificada por una persona con una prueba manual que satisfaga los requerimientos planteados por el problema.

Partes de un Algoritmo



8. Representación gráfica de algoritmos.

[<http://www.mailxmail.com/curso-aprende-programar/representacion-grafica-algoritmos>]

- a) Descripción Narrada
- b) Pseudocódigo
- c) Diagramas de Flujo
- d) Diagramas N- S (Nassi-Schneiderman o de Chapin)

1 Descripción Narrada

Este algoritmo es caracterizado porque sigue un proceso de ejecución común y lógico, describiendo textualmente paso a paso cada una de las actividades a realizar dentro de una actividad determinada.

Ejemplo 1 Algoritmo para asistir a clases:

1. Levantarse
2. Bañarse
3. Vestirse
4. Desayunar
5. Cepillarse los dientes
6. Salir de casa
7. Tomar el autobús
8. Llegar al ITCA
9. Buscar el aula
10. Ubicarse en un asiento

2.Descripción en Pseudocódigo

Pseudo = falso. El pseudo código no es realmente un código sino una imitación y una versión abreviada de instrucciones reales para las computadoras. Es una técnica para diseño de programas que permite definir las estructuras de datos, las operaciones que se aplicarán a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema. Utiliza un pseudolenguaje muy parecido a nuestro idioma, pero que respeta las directrices y los elementos de los lenguajes de programación. Se concibió para superar las dos principales desventajas de los flujogramas: lento de crear y difícil de modificar sin un nuevo redibujo.

Ejemplo 1

Diseñar un algoritmo que lea cuatro variables y calcule e imprima su producto, suma y media aritmética.

```
inicio
leer (a, b, c, d)
producto <-- (a * b * c * d)
suma <-- (a + b + c + d)
media <-- (a + b + c + d) / 4
escribir (producto, suma, media)
fin
```

3. Diagramas N-S

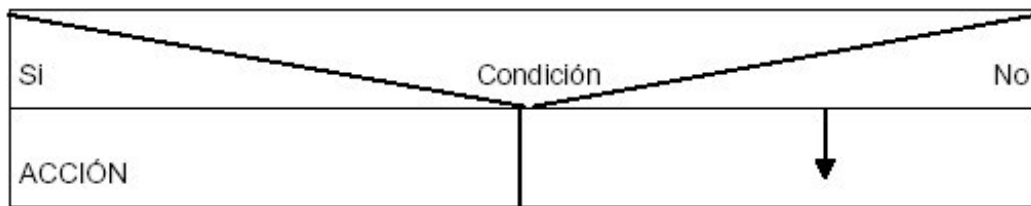
Son una herramienta que favorece la programación estructurada y reúne características gráficas propias de diagramas de flujo y lingüísticas propias de pseudocódigos. Constan de una serie de cajas contiguas que se leerán siempre de

arriba-abajo y sus estructuras lógicas son las siguientes:

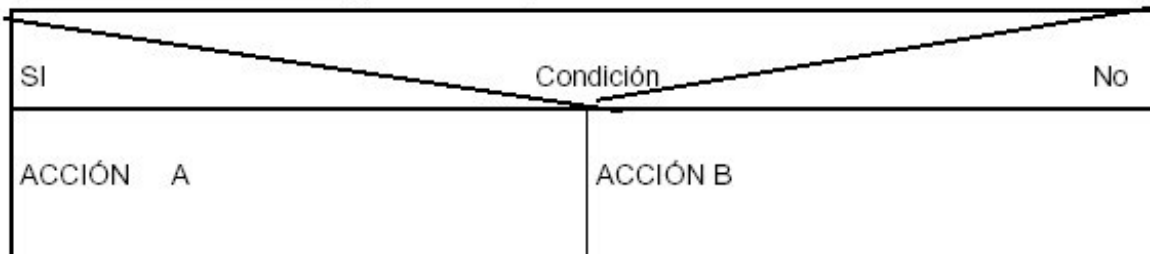
Estructura Secuencial

Acción A
Acción B
.....
.....
Acción N

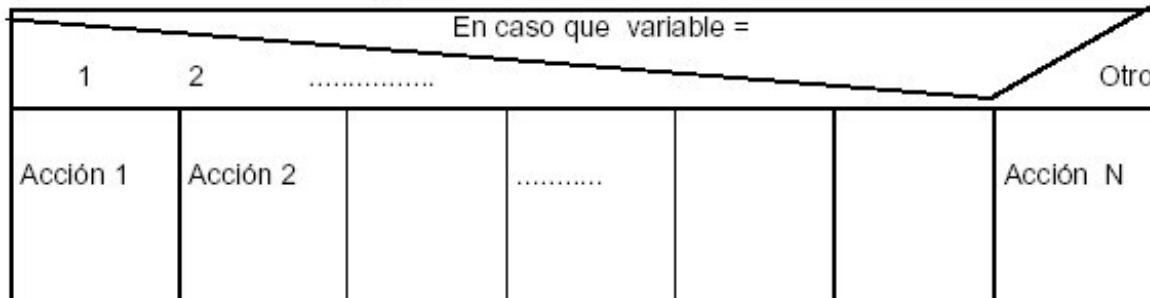
Estructura Selectiva Simple (If-Then)

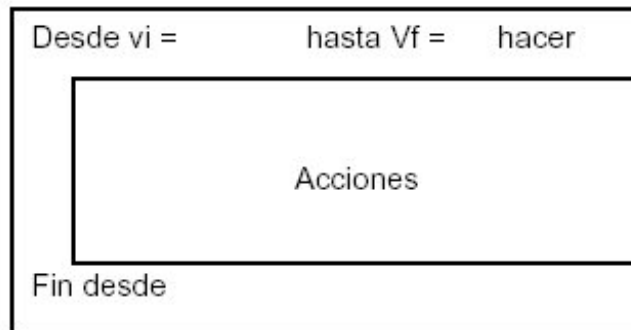


Estructura Selectiva Doble (If-Then-Else)

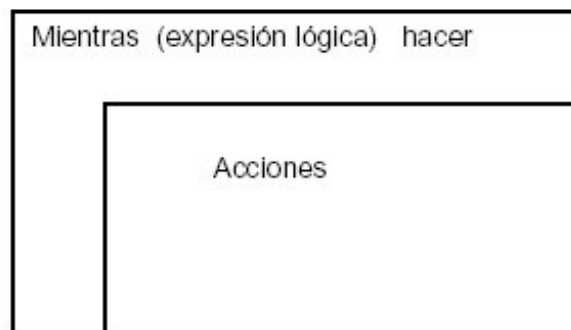
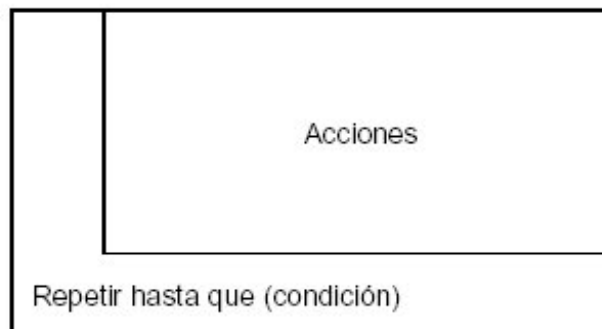


Estructura Selectiva Múltiple (Select- Case Endcase)



Estructuras Repetitivas (Desde/Para)

Estructura Iterativa Mientras

Estructura Iterativa Repetir

Por ejemplo, el ejercicio anterior se puede representar así:

Inicio
Leer (a,b,c,d)
Producto $\leftarrow (a*b*c*d)$
Suma $\leftarrow (a+b+c+d)$
Media $\leftarrow suma/4$
Escribir (producto, suma, media)
Fin

4. Diagramas de Flujo.

Son la representación gráfica de la solución algorítmica de un problema. Para diseñarlos se utilizan determinados símbolos o figuras que representan una acción dentro del procedimiento. Utilizan unos símbolos normalizados, con los pasos del algoritmo escritos en el símbolo adecuado y los símbolos unidos con flechas, denominadas líneas de flujo, que indican el orden en que los pasos deben ser ejecutados.

Para su elaboración se siguen ciertas reglas:

Se escribe de arriba hacia abajo y de izquierda a derecha

Siempre se usan flechas verticales u horizontales, jamás curvas

Evitar cruce de flujos

En cada paso expresar una acción concreta

Secuencia de flujo normal en una solución de problema

Tiene un inicio

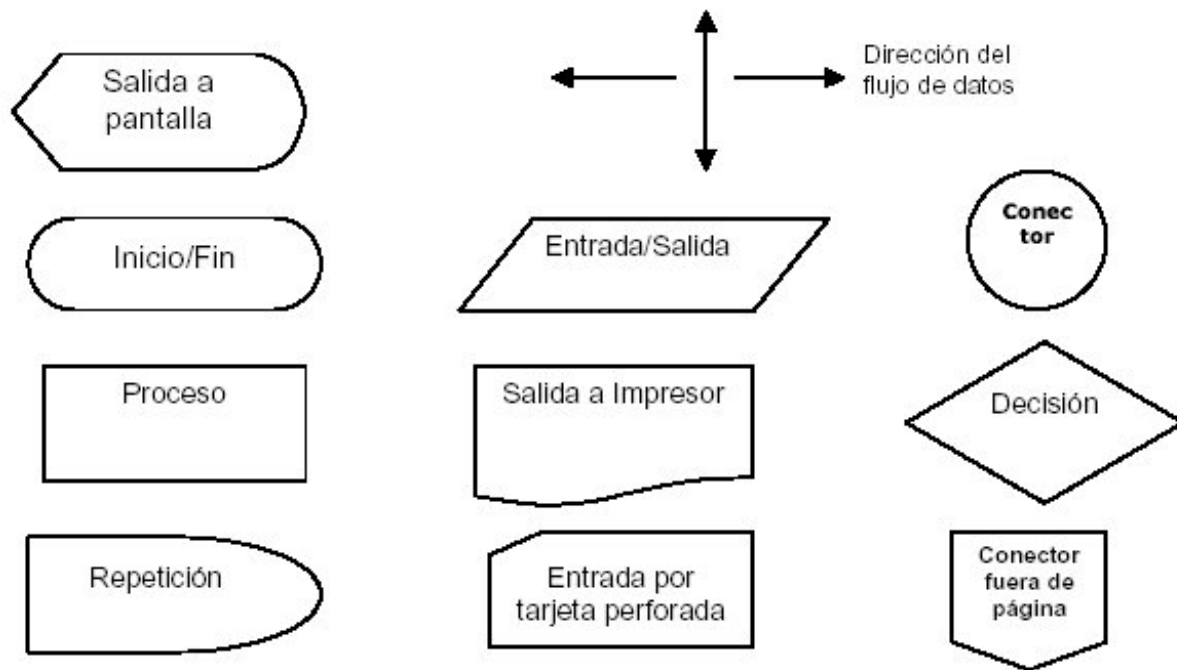
Una lectura o entrada de datos

El proceso de datos

Una salida de información

Un final

Simbología para diseñar flujogramas.



VENTAJAS DE USAR FLUJOGRAMAS

Rápida comprensión de las relaciones

Análisis efectivo de las diferentes secciones del programa

Pueden usarse como modelos de trabajo en el diseño de nuevos programas o sistemas

Comunicación con el usuario

Documentación adecuada de los programas

Codificación eficaz de los programas

Depuración y pruebas ordenadas de programas

DESVENTAJAS DE LOS FLUJOGRAMAS

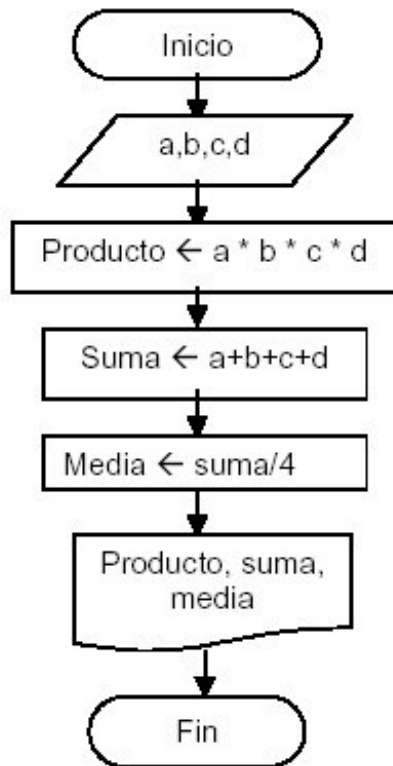
Diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño

Acciones a seguir tras la salida de un símbolo de decisión, pueden ser difíciles de seguir si existen diferentes caminos

No existen normas fijas para la elaboración de los diagramas de flujo que permitan incluir todos los detalles que el usuario desee introducir.

Representando el ejemplo como flujograma tenemos:





9. Tipos de estructuras de programación. Estructuras básicas y secuencial.

[<http://www.mailxmail.com/...mar/tipos-estructuras-programacion-estructuras-basicas-secuencial>]

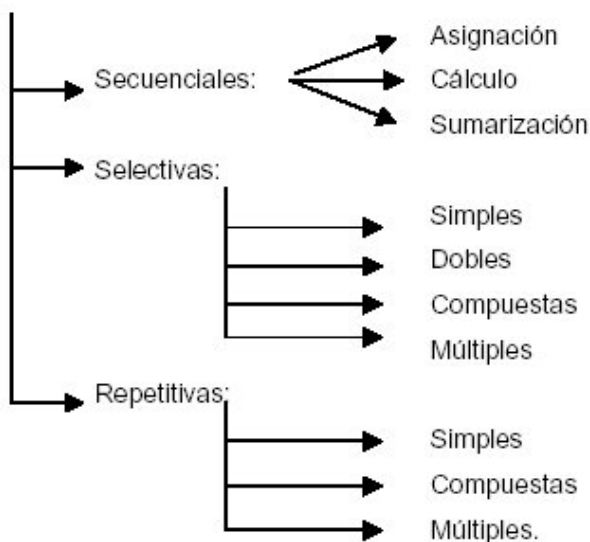
Un problema se puede dividir en acciones elementales o instrucciones, usando un número limitado de estructuras de control (básicas) y sus combinaciones que pueden servir para resolver dicho problema.

Las Estructuras Básicas pueden ser:

Secuenciales: cuando una instrucción del programa sigue a otra.

Selección o decisión: acciones en las que la ejecución de alguna dependerá de que se cumplan una o varias condiciones. Repetición, Iteración: cuando un proceso se repite en tanto cierta condición sea establecida para finalizar ese proceso.

ESTRUCTURAS BÁSICAS.



Estructura Secuencial.

Se caracteriza porque una acción se ejecuta detrás de otra. El flujo del programa coincide con el orden físico en el que se han ido poniendo las instrucciones. Dentro de este tipo podemos encontrar operaciones de inicio/fin, inicialización de variables, operaciones de asignación, cálculo, sumarización, etc. Este tipo de estructura se basa en las 5 fases de que consta todo algoritmo o programa:

Definición de variables (Declaración)

Inicialización de variables.

Lectura de datos

Cálculo

Salida

Ejemplo 1.

Se desea encontrar la longitud y el área de un círculo de radio 5.

Solución.

El objetivo del ejercicio es encontrar la longitud y el área de un círculo con un radio conocido y de valor 5. Las salidas serán entonces la longitud y el área. (Fase 5 del algoritmo) Sabemos que la longitud de un círculo viene dada por la fórmula $2 * \pi * \text{radio}$ y que el área viene dada por $\pi * \text{radio al cuadrado}$. (Fase 4 del algoritmo) Si definimos las variables como: (fase 1 del algoritmo)

L = Longitud A = área R = radio $\pi = 3.1416$ hagamos el algoritmo:

Inicio

$\pi \leftarrow 3.1416$ (definición de un valor constante)

$R \leftarrow 5$ (radio constante ya que es conocido su valor)

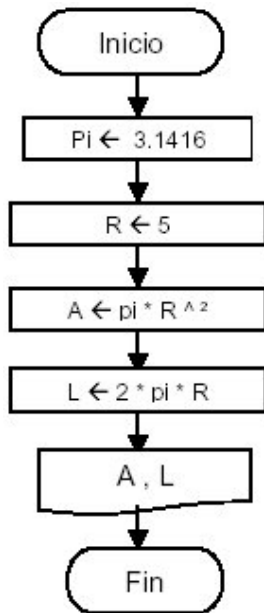
$A \leftarrow \pi * R^2$ (asignación del valor del área)

$L \leftarrow 2 * \pi * R$ (asignación del valor de la longitud)

Escribir (A, L) (salida del algoritmo)

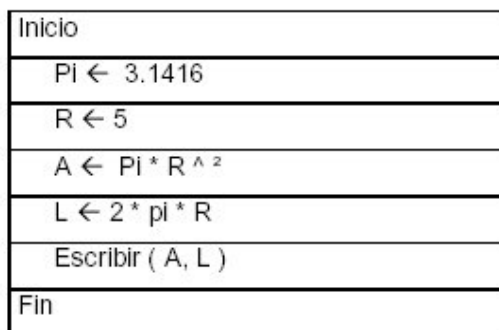
Fin

Representación en Diagrama de Flujo para el ejemplo:



Representación en Diagrama Nassi Schneiderman:

Los problemas secuenciales en diagramas N-S se representan solamente por cajas con líneas horizontales



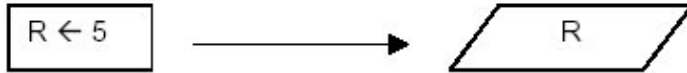
En este ejercicio no existen datos de entrada ya que para calcular el área y la longitud necesitamos únicamente el radio y el valor de π los cuales ya son dados en el problema. Modificar el problema anterior para que sea capaz de calcular el área y la longitud de un círculo de cualquier radio requerido. Solución.

El problema es el mismo con la variante de que ahora ya existe un dato de entrada, puesto que el radio puede ser cualquiera y será necesario que el usuario sea quien lo introduzca de teclado. Usando las misma definición de variables tenemos:

Algoritmo:

Inicio
Pi \rightarrow 3.1416 (fase de inicialización)
Leer (R) (fase de lectura)
Area \rightarrow pi * R ² (fase de cálculos)
L \rightarrow 2 * pi * R
Escribir (A, L) (fase de salida)
Fin

Note que la instrucción de asignación fue cambiada por la instrucción leer. En el flujograma deberán cambiarse también los símbolos que los representan:



Ejemplo 3.

Leer el sueldo de tres empleados y aplicarles un aumento del 10, 12 y 15% respectivamente. Desplegar el resultado.

Salidas: Sueldos finales

Entradas: Salarios de los empleados

Datos adicionales: aumentos del 10, 12 y 15%

Cálculos:

Sueldo final = sueldo inicial + aumento

Aumento = sueldo inicial * porcentaje/100

Definición de variables:

Sf1, Sf2, Sf3 = los sueldos finales

S1, S2, S3 = salarios de los empleados

Aum1, aum2, aum3 = aumentos

ALGORITMO

Inicio
Leer (S1,S2,S3)
Aum1 \rightarrow S1 * 0.10
Aum2 \rightarrow S2 * 0.12
Aum3 \rightarrow S3 * 0.15
Sf1 \rightarrow S1 + Aum1
Sf2 \rightarrow S2 + Aum2
Sf3 \rightarrow S3 + Aum3
Escribir (SF1,SF2,SF3)
Fin

FLUJOGRAMA

FLUJOGRAMA:

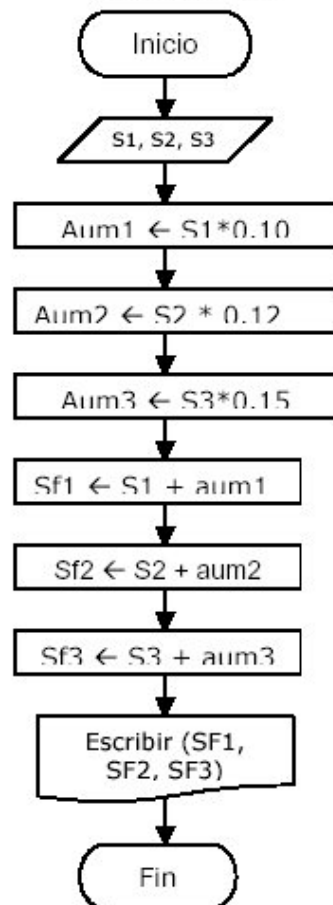
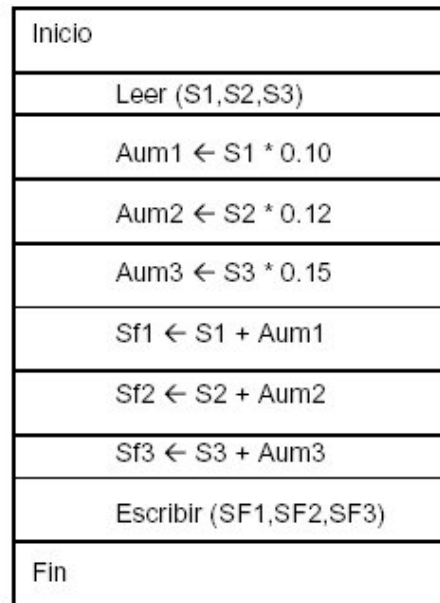


DIAGRAMA N-S



10. Tipos de estructuras selectivas. Estructura simple.

[<http://www.mailxmail.com/...-aprende-programar/tipos-estructuras-selectivas-estructura-simple>]

La especificación formal de algoritmos tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición.

Estas estructuras se identifican porque en la fase de solución del problema existe algún punto en el cual es necesario establecer una pregunta, para decidir si ciertas acciones deben realizarse o no.

Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (if - then - else o en español si - entonces - sino) y en flujograma con una figura geométrica en forma de rombo.

Las estructuras selectivas o alternativas se clasifican en:

- a) Simples
- b) Dobles
- c) Compuestas
- d) Múltiples

ESTRUCTURAS SELECTIVAS SIMPLES.

Se identifican porque están compuestos únicamente de una condición. La estructura si - entonces evalúa la condición y en tal caso:

Si la condición es verdadera, entonces ejecuta la acción Si (o acciones si son varias).

Si la condición es falsa, entonces no se hace nada.

Español	Inglés
Si <condición>	If <condición>
Entonces	then
<acción Si>	<acción Si>
fin_si	endif

Representación en Flujograma:

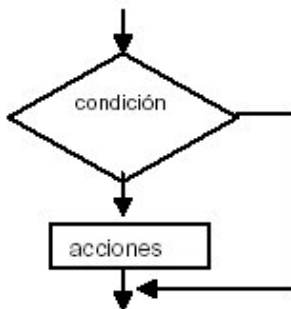
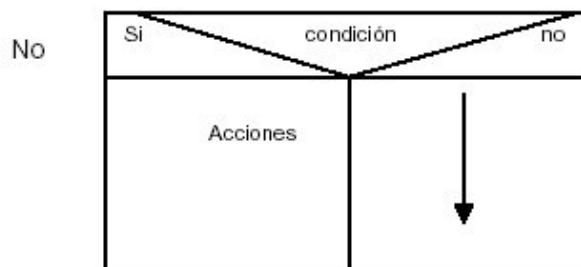


Diagrama N-S



Ejemplo 1.

Construir un algoritmo tal, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" en caso que esa calificación fuese mayor que 8.

Salidas: mensaje de aprobado si se cumple la condición.

Entradas: calificación

Datos adicionales: un alumno aprueba si la calificación es mayor que 8

Variables:

Cal = calificación

Algoritmo:
Inicio
Leer (cal)
Si $cal > 8$ entonces
Escribir ("aprobado")
Fin_si
Fin

Flujograma:

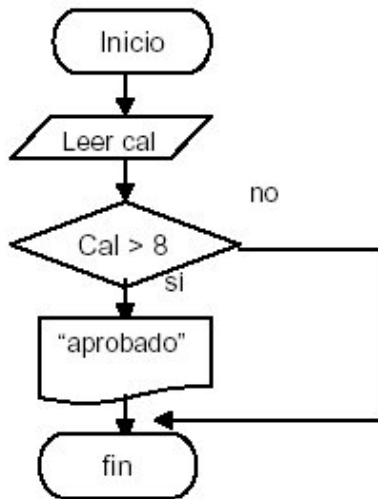
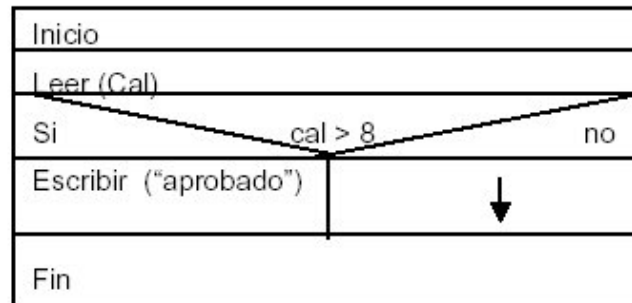


Diagrama N-S



11. Estructura de selección doble.

[<http://www.mailxmail.com/curso-aprende-programar/estructura-seleccion-doble>]

Son estructuras lógicas que permiten controlar la ejecución de varias acciones y se utilizan cuando se tienen dos opciones de acción, por la naturaleza de estas se debe ejecutar una o la otra, pero no ambas a la vez, es decir, son mutuamente excluyentes.

Representación pseudocodificada.

Español	Inglés
Si <condición> entonces	If <condición> then
<acción S1>	<acción S1>
sino	else
<acción S2>	<acción S2>
Fin_si	End_if

Entonces, si una condición C es verdadera, se ejecuta la acción S1 y si es falsa, se ejecuta la acción S2.

Flujograma:

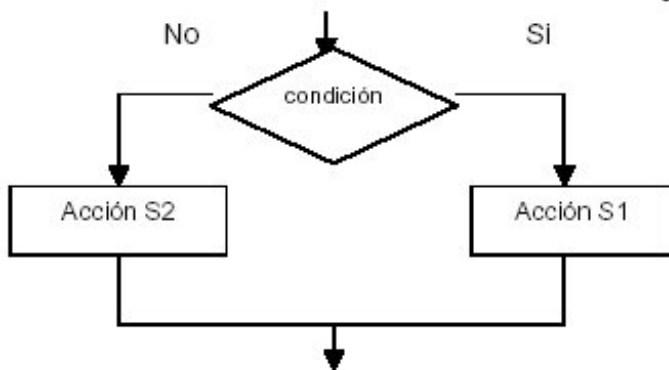
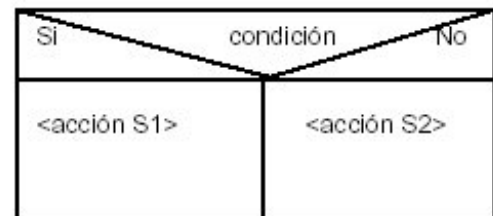


Diagrama N-S



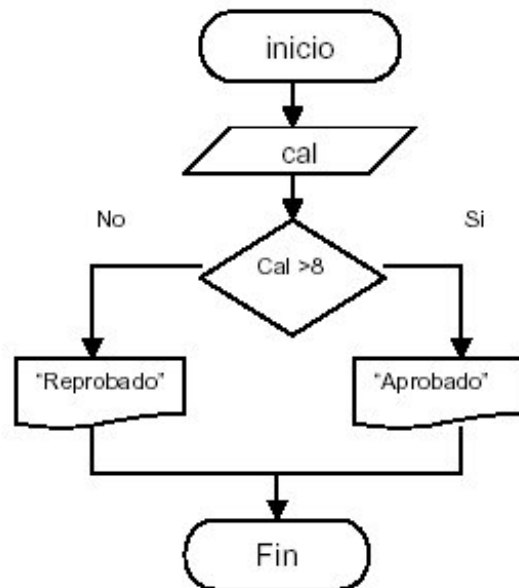
Ejemplo 1

Dado como dato la calificación de un alumno en un examen, escriba "aprobado" si su calificación es mayor que 8 y "Reprobado" en caso contrario.

Algoritmo:

```

Inicio
Leer (cal)
Si cal > 8 entonces
Escribir ("aprobado")
Sino
Escribir ("reprobado")
Fin_si
Fin
  
```



Ejemplo 2.

Dado como dato el sueldo de un trabajador, aplicar un aumento del 15% si su sueldo es inferior a \$1000 y 12% en caso contrario, luego imprimir el nuevo sueldo del trabajador.

Algoritmo:

Inicio

Leer (sue)

Si sue < 1000 entonces

$Nsue \leftarrow sue * 1.15$

Sino

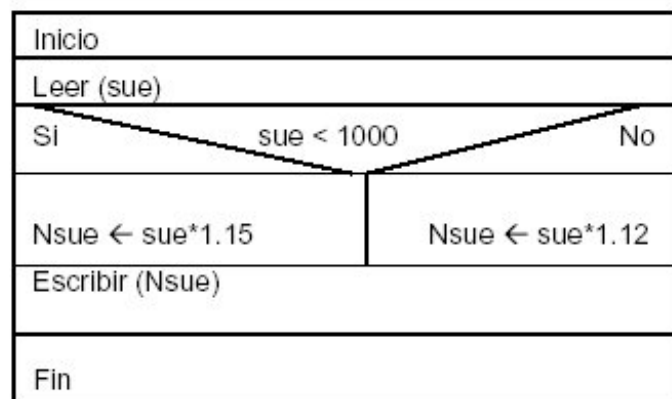
$Nsue \leftarrow sue * 1.12$

Fin_si

Escribir (Nsue)

Fin

N - S



EXPRESIONES LÓGICAS

Sirven para plantear condiciones o comparaciones y dan como resultado un valor booleano verdadero o falso, es decir, se cumple o no se cumple la condición. Se pueden clasificar en simples y complejas. Las simples son las que usan operadores relacionales y las complejas las que usan operadores lógicos.

Ejemplos:

Un ejemplo en el cual usamos el operador lógico AND sería:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como C1 y C2. El aspirante que obtenga calificaciones mayores que 80 en ambos exámenes es aceptado; en caso contrario

es rechazado.

En este ejemplo se dan las condiciones siguientes:

Si $(C1 \geq 80)$ y $(C2 \geq 80)$ entonces

Escribir ("aceptado")

Sino

Escribir ("rechazado")

Fin_si

Note que también usa operadores relacionales. Por lo general cuando hay operadores lógicos, éstos van acompañados de operadores relacionales. Un ejemplo usando el operador lógico OR sería:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como C1 y C2. El aspirante que obtenga una calificación mayor que 90 en cualquiera de los exámenes es aceptado; en caso contrario es rechazado.

En este caso se dan las condiciones siguientes:

Si $(C1 \geq 90)$ or $(C2 \geq 90)$ entonces

Escribir ("aceptado")

Sino

Escribir ("rechazado")

Fin_si

La instrucción equivale a OR ya que nos dice que puede ser en cualquiera de los exámenes no necesariamente en los dos. En el ejemplo 1 la palabra ambos equivalía a seleccionar la instrucción AND. Si la instrucción nos dijera que obtenga una nota en cualquiera de los exámenes pero no en ambos, nos estaría indicando una instrucción XOR que es un tipo de OR pero exclusivo. Es decir, no puede considerarse el caso en que tenga la misma nota en los dos exámenes, solo en uno de los dos.

12. Estructuras selectivas compuestas.

[<http://www.mailxmail.com/curso-aprende-programar/estructuras-selectivas-compuestas>]

En la solución de problemas encontramos numerosos casos en los que luego de tomar una decisión y marcar el camino correspondiente a seguir, es necesario tomar otra decisión. Dicho proceso puede repetirse numerosas veces. En aquellos problemas en donde un bloque condicional incluye otro bloque condicional se dice que un bloque está anidado dentro del otro.

Ejemplo 1.

Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que éstas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple.

Solución.

Lo primero que hay que determinar es si el trabajador trabajó horas extras o no.

Encontrar las horas extras de la siguiente forma:

Horas extras = horas trabajadas - 40

En caso que sí trabajó horas extras:

Si horas extras > 8 entonces a horas extras excedentes de 8 = horas extras - 8 y

pago por horas extras = pago por hora normal * 2 * 8 + pago por hora normal * 3 *

horas extras excedentes de 8

De otra forma (solo horas al doble) pago por horas extras = pago por hora normal * 2 * horas extras.

Finalmente, pago total que recibirá el trabajador será:

Pago = pago * hora normal * 40 + pago por horas extras.

Si no trabajó horas extras tendremos:

Pago = pago por hora normal * horas trabajadas.

Datos de salida: Pago.

Datos de entrada: número de horas trabajadas y pago por hora normal.

Definición de variables:

ht = horas trabajadas het = horas extras que exceden de 8

ph = pago por hora normal phe = pago por horas extras

he = horas extras pt = pago que recibe el trabajador

Algoritmo:

Inicio

Leer (ht, ph)

Si ht > 40 entonces

He ← ht - 40

Si he > 8 entonces

Het ← he - 8

Phe ← ph * 2 * 8 + ph * 3 * het

Sino

Phe ← ph * 2 * he

Fin_si

Pt ← ph * 40 + phe

Sino

Pt \leftarrow ph * ht
Fin_si
Escribir (pt)
Fin

Ejemplo 2.

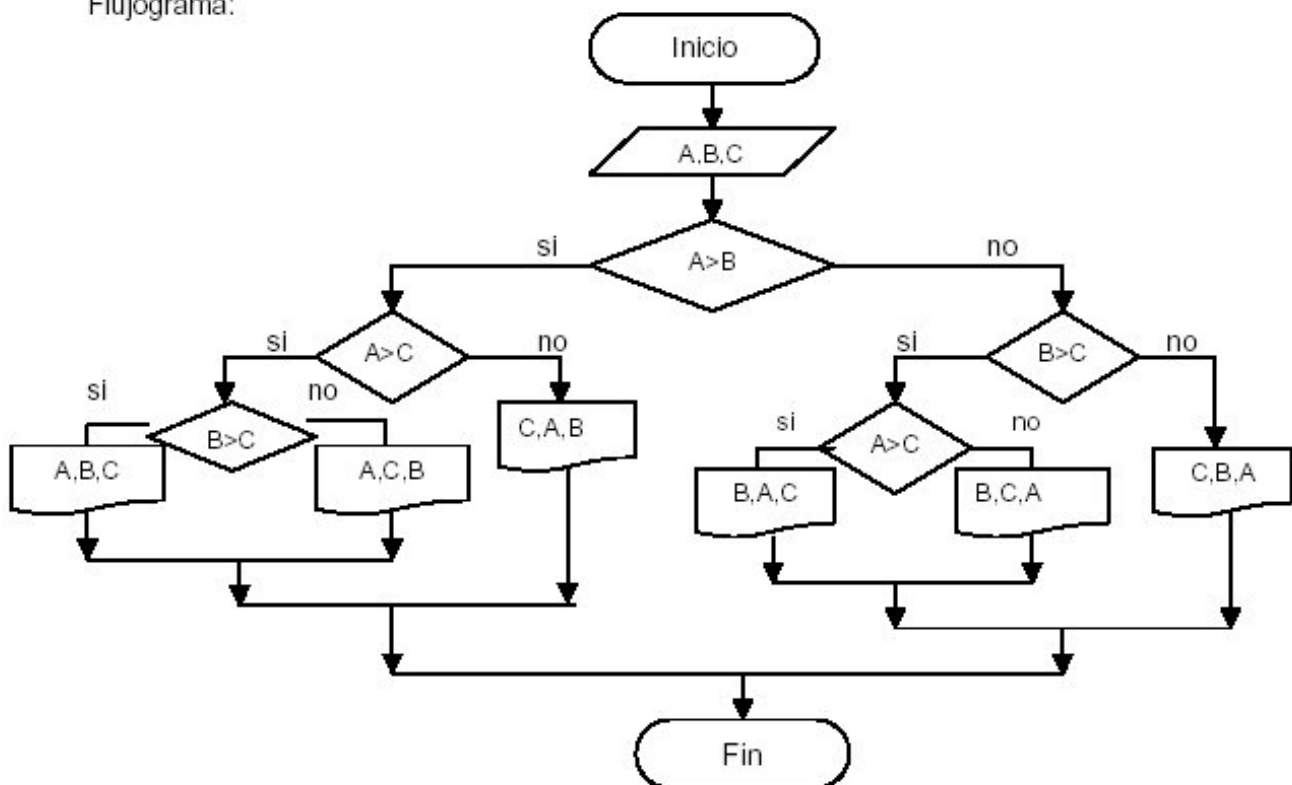
Dados los datos A, B y C que representan números enteros diferentes, construir un algoritmo para escribir estos números en forma descendente. Este es un ejemplo de los algoritmos conocidos como de Lógica Pura, ya que poseen muchas decisiones y muchas bifurcaciones.

Salida: A, B y C ordenados descendientemente.

Entradas: A, B y C.

La dinámica del problema es comparar dos números a la vez para conocer cuál es el mayor.

Flujograma:



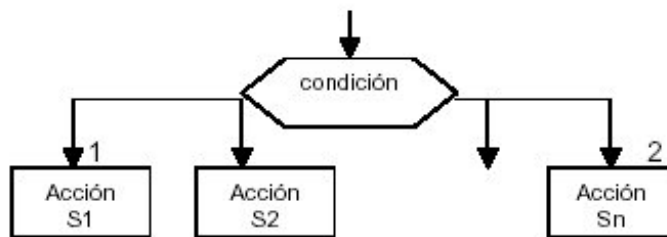
13. Estructura selectiva múltiple.

[<http://www.mailxmail.com/curso-aprende-programar/estructura-selectiva-multiple>]

Con frecuencia es necesario que existan más de dos elecciones posibles. Este problema se podría resolver por estructuras selectivas simples o dobles, anidadas o en cascada, pero si el número de alternativas es grande puede plantear serios problemas de escritura y de legibilidad.

Usando la estructura de decisión múltiple se evaluará una expresión que podrá tomar n valores distintos, 1, 2, 3, ..., n y según que elija uno de estos valores en la condición, se realizará una de las n acciones o lo que es igual, el flujo del algoritmo seguirá sólo un determinado camino entre los n posibles.

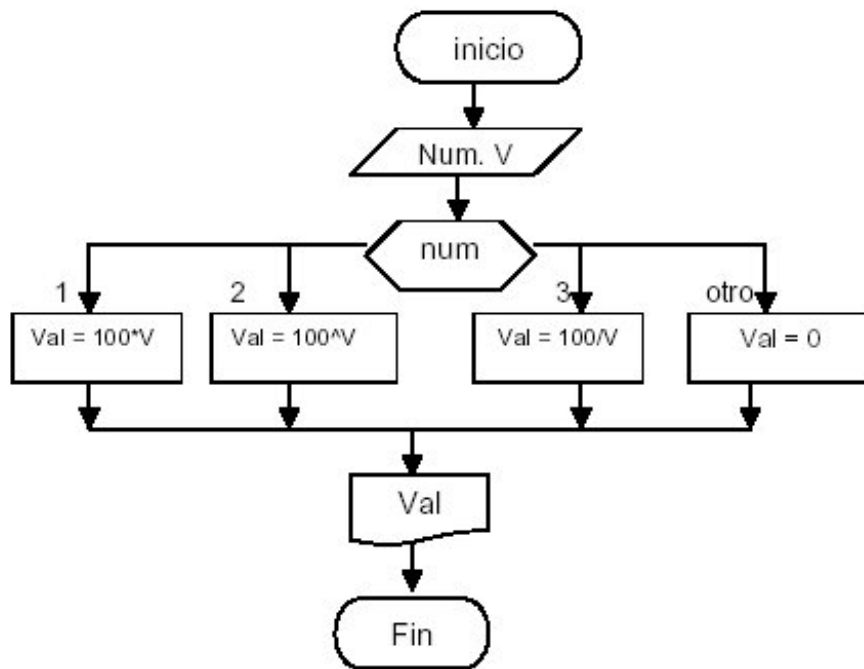
Esta estructura se representa por un selector el cual si toma el valor 1 ejecutará la acción 1, si toma el valor 2 ejecutará la acción 2, si toma el valor N realizará la acción N .



Ejemplo 1:

Diseñar un algoritmo tal que dados como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

$$\text{Val} = \begin{cases} 100 * V & \text{si Num} = 1 \\ 100 \wedge V & \text{si num} = 2 \\ 100 / V & \text{si num} = 3 \\ 0 & \text{cualquier otro valor de num} \end{cases}$$



Inicio			
Leer (num,v)			
En caso de que num sea			
1	2	3	otro
Val= 100*V	Val= 100^V	Val=100/V	Val = 0
Escribir (val)			
Fin			

Inicio

Leer (num,V)

En caso que Num sea

1: hacer val = 100*V

2: hacer val = 100 ^V

3: hacer val = 100/V

De otra forma:

Val = 0

Fin_caso_que

Escribir (val)

Fin

Ejemplo 2.

Dados como datos la categoría y el sueldo de un trabajador, calcule el aumento correspondiente teniendo en cuenta la siguiente tabla. Imprimir la categoría del trabajador y el nuevo sueldo.

INCREMENTOS	
CATEGORÍA	AUMENTO
1	15%
2	10%
3	8%
4	7%

Definición de variables:

Cate = categoría

Sue = sueldo

Nsue = nuevo sueldo

ALGORITMO

Inicio

Leer (cate, sue)

En caso que cate sea

1: hacer nsue \leftarrow sue * 1.15

2: hacer nsue \leftarrow sue * 1.10

3: hacer nsue \leftarrow sue * 1.08

4: hacer nsue \leftarrow sue * 1.07

Fin_caso_que

Escribir (cate, nsue)

Fin



14. Estructuras repetitivas e iterativas.

[<http://www.mailxmail.com/curso-aprende-programar/estructuras-repetitivas-iterativas>]

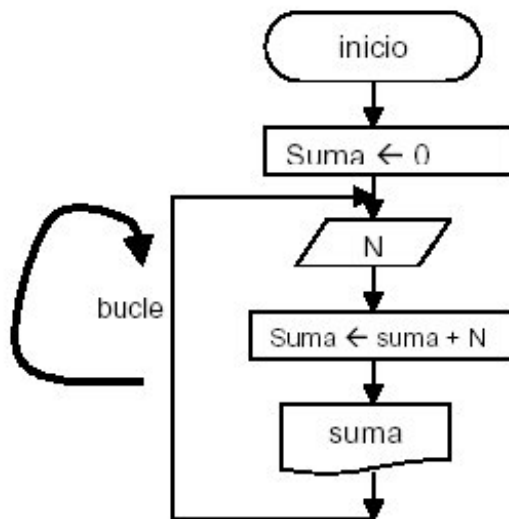
Son operaciones que se deben ejecutar un número repetido de veces. El conjunto de instrucciones que se ejecuta repetidamente cierto número de veces, se llama Ciclo, Bucle o Lazo.

Iteración es cada una de las diferentes pasadas o ejecuciones de todas las instrucciones contenidas en el bucle.

Fases de un Programa Cíclico :

1. Entrada de datos e instrucciones previas
2. Lazo o bucle
3. Instrucciones finales o resto del proceso
4. Salida de resultado

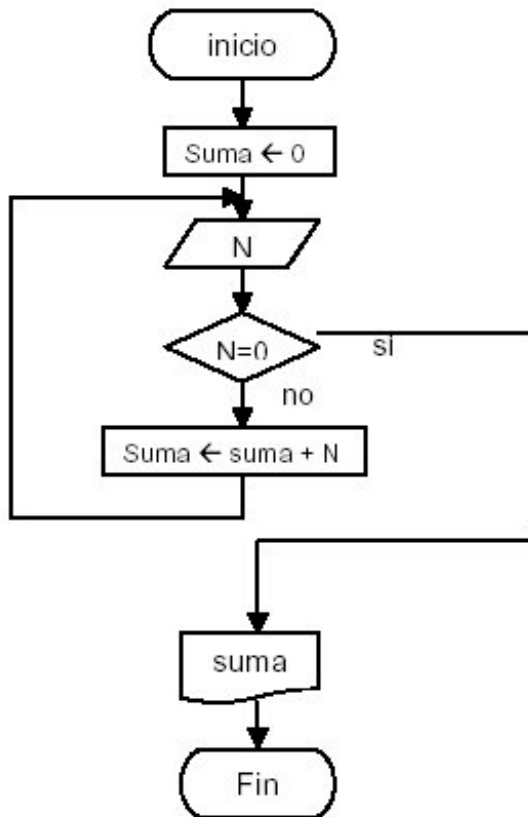
Ejemplo de bucle infinito:



En este flujograma, el bucle se estará repitiendo indefinidamente ya que no existe ninguna condición que nos permita finalizar en algún momento.

En el flujograma anterior, observa que la flecha que se regresa hacia arriba nos está indicando que hay que volver a evaluar la expresión. En ese caso como el bucle es infinito, no se tiene una condición para terminar y se estará haciendo siempre. En el siguiente ejemplo, ya se agregó una condición, la cual nos permitirá finalizar la ejecución del bucle en el caso en que la condición se cumpla.

Ejemplo de bucle finito:



En este ejemplo, el bucle finalizará cuando se cumpla la condición de que N sea igual a cero.

Bucles Repetitivos:

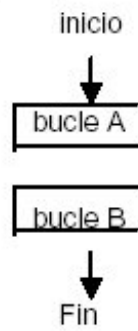
A continuación, te muestro tres diseños de estructuras cíclicas: las independientes son cuando los bucles se realiza uno primero hasta que se cumple la condición y solo en ese caso se entra al bucle B.

En los ciclos anidados, al entrar a una estructura de repetición, dentro de ella se encuentra otra. La más interna se termina de realizar y se continúa con la externa hasta que la condición se cumple.

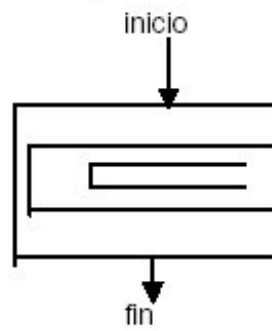
En los bucles cruzados, los cuales no son convenientes de utilizar, se tiene que iniciamos un bucle y no se ha terminado cuando empezamos otro, luego utilizamos estructuras goto (saltos) para pasar al bucle externo y se quedan entrelazados.

Esto puede ocasionar que el programa pierda el control de cuál proceso se está ejecutando y podamos obtener resultados erróneos. Veamos gráficamente el diseño de estas tres formas cíclicas:

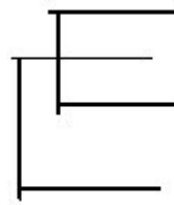
a) Independientes



b) Anidados

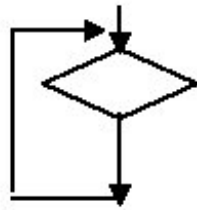


c) Cruzados

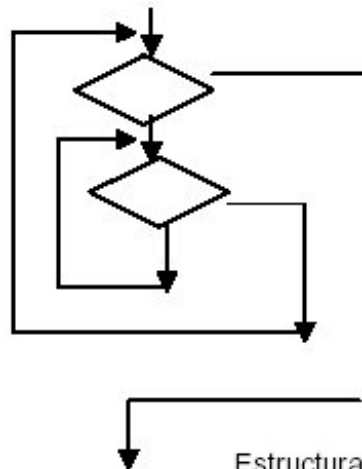


No es correcto su diseño

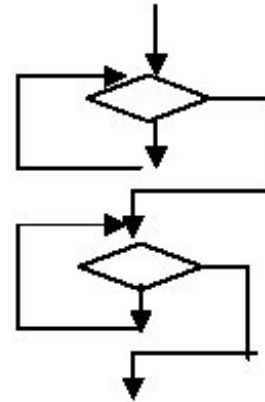
Diseño de las estructuras en flujograma:



Estructura repetitiva simple



Estructura repetitiva Anidada



Estructura repetitiva Compuesta múltiple

15. Estructuras básicas.

[<http://www.mailxmail.com/curso-aprende-programar/estructuras-basicas>]

Durante las siguientes lecciones estaremos estudiando tres estructuras básicas que son:

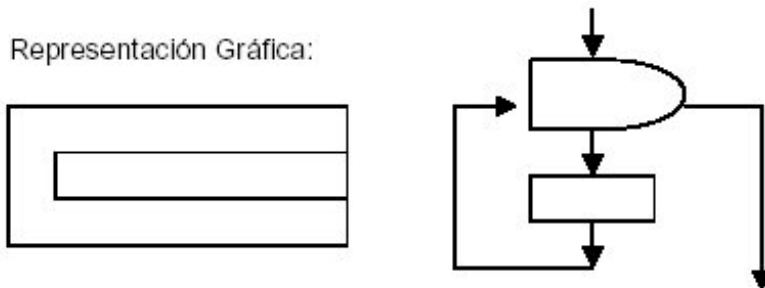
Estructura Desde/Para
Estructura Mientras
Estructura Repetir

En esta lección estudiaremos la forma general de la estructura Desde/Para, su uso y ejemplos.

Estructura Desde/Para:

Se usa frecuentemente cuando se conoce de antemano el número de veces que se ejecutarán las acciones de un bucle. Esta es una de sus características.

Representación Gráfica:



Representación pseudocodificada:

Español

Desde var = valor inicial hasta valor final hacer
valor final do
Acciones
Fin_desde

Inglés

For var=valor inicial to
valor final
acciones
end_for

A la estructura Desde/Para se le conoce como Repetitiva. Para utilizar esta estructura en algoritmos, debemos hacer uso de contadores y algunas veces de acumuladores, cuyos conceptos se describen a continuación:

CONTADOR:

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción. Los contadores se utilizan con la finalidad de contar sucesos o acciones internas de un bucle; deben realizar una operación de inicialización y posteriormente las sucesivas de incremento o decremento del mismo. La inicialización consiste en asignarle al contador un valor. Se situará antes y fuera del bucle.

Representación:

<nombre del contador> ← nombre del contador + <valor constante>

Si en vez de incremento es decremento se coloca un menos en lugar del más.

Ejemplo: $i = i + 1$

ACUMULADOR O TOTALIZADOR :

Es una variable que suma sobre sí misma un conjunto de valores para de esta manera tener la suma de todos ellos en una sola variable. La diferencia entre un contador y un acumulador es que mientras el primero va aumentando de uno en uno, el acumulador va aumentando en una cantidad variable.

Representación: <Nombre del acumulador> \leftarrow <nombre del acumulador> + <valor variable>

Ejemplo:

Calcular la suma de los cuadrados de los primeros 100 enteros y escribir el resultado. Se desea resolver el problema usando estructura Desde, Mientras y luego Repetir.

1. Usando la estructura Desde:

Inicio

Suma \leftarrow 0

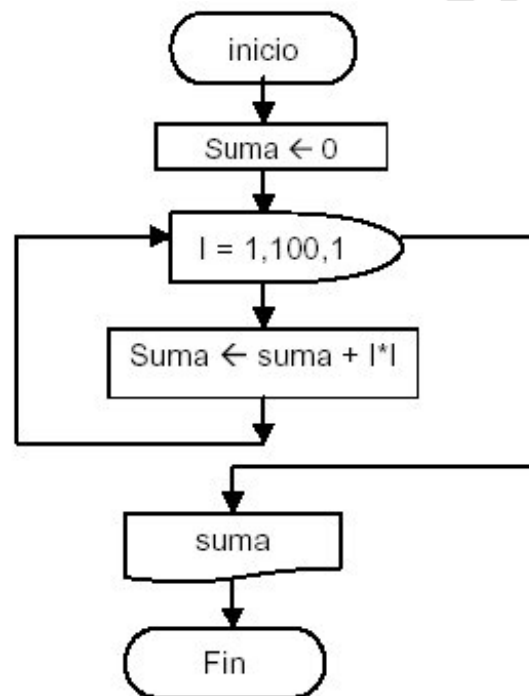
Desde I = 1 hasta 100 hacer

Suma \leftarrow suma + I * I

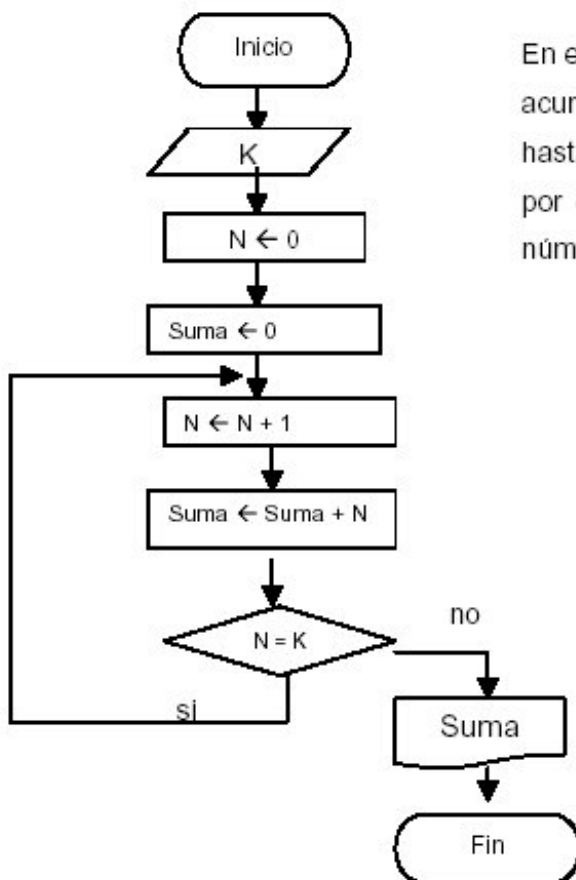
Fin_desde

Escribir (suma)

Fin

**Ejemplo 2.**

Elaborar un flujograma para encontrar la suma de los K primeros números enteros.



En este caso, utilizamos un contador (N) y un acumulador (suma). N cuenta cada número hasta llegar a K que es el valor máximo dado por el problema. Suma lleva la suma de los números enteros que se van generando

En este ejemplo hemos utilizado un bucle repetir, el cual estudiaremos en otra lección. Lo que queremos hacer notar por el momento, es cómo funcionan el contador y el acumulador. Nota que N es el contador, el cual se inicializa en este caso, con cero, antes de entrar al bucle. Dentro del bucle podrás notar que N se incrementa en 1.

También observa la variable suma, la cual es un acumulador que lleva la suma de los números generados. También debe inicializarse con cero, ya que para sumar valores debemos partir de cero, es decir, que al inicio no tenemos nada. Dentro del bucle, suma se incrementa en un número N, pero la diferencia con el contador N, es que a suma le sumamos N más ella misma.

EJERCICIO:

Trata de elaborar un flujograma para encontrar el cuadrado de los primeros 25 números naturales, usando la estructura Desde/Para.

¿Qué necesitas para resolver el problema contadores o acumuladores? Modifica el flujograma del ejercicio anterior para que también te muestre la suma de dichos cuadrados.

¿Qué necesitas agregar ahora?

RESUMEN

En esta lección aprendimos un poco del uso de contadores y acumuladores. También aprendimos a elaborar flujogramas o algoritmos usando la estructura

Desde. Hay un número importante de reglas que deben seguirse cuando se utilizan instrucciones

Desde:

Los valores inicial y final de la variable de control se determinan antes de que empiece la repetición y no pueden cambiarse durante la ejecución de la instrucción Desde. Dentro del cuerpo del bucle Desde, los valores de las variables que especifican los valores inicial y final pueden cambiar, pero esto no va a afectar al número de repeticiones. La instrucción del cuerpo del bucle de una instrucción Desde puede utilizar el valor de la variable de control, pero no debe modificar este valor. Esta estructura se puede usar únicamente en aquellos casos en que conocemos el número de veces que se va a realizar el ciclo.

Esta estructura hace el incremento automáticamente y se inicializa en la instrucción desde.

16. Estructuras iterativas. Estructura mientras.

[<http://www.mailxmail.com/...urso-aprende-programar/estructuras-iterativas-estructura-mientras>]

Se llama Mientras a la estructura algorítmica que se ejecuta mientras la condición evaluada resulte verdadera. Se evalúa la expresión booleana y, si es cierta, se ejecuta la instrucción especificada, llamada el cuerpo del bucle. Entonces se vuelve a evaluar la expresión booleana, y si todavía es cierta se ejecuta de nuevo el cuerpo. Este proceso de evaluación de la expresión booleana y ejecución del cuerpo se repite mientras la expresión sea cierta.

Cuando se hace falsa, finaliza la repetición. En la lección anterior iniciamos con las estructuras repetitivas. La estructura While y la estructura Repeat, se conocen como iterativas. Se usan cuando no se conoce con anticipación el número de veces que se ejecutará la acción.

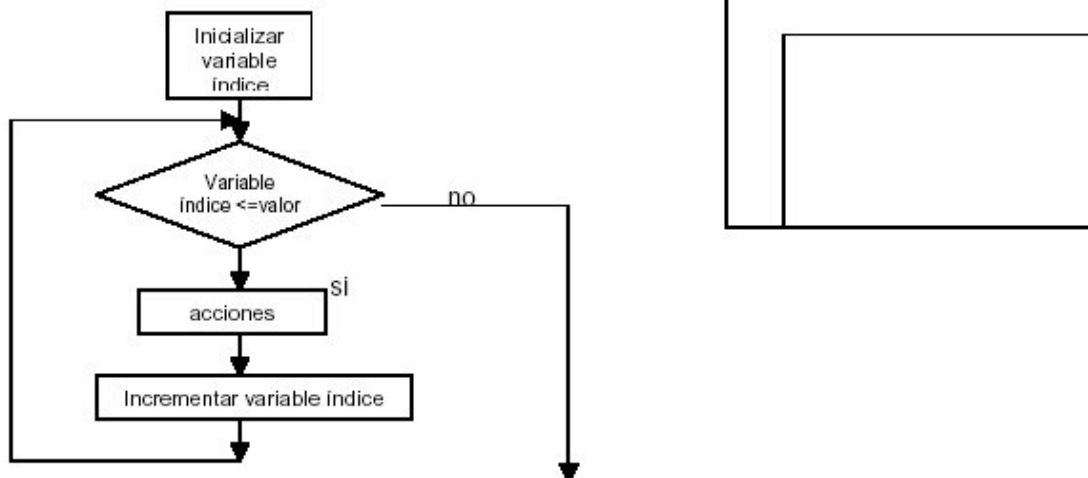
La diferencia entre ambas es que la condición se sitúa al principio (Mientras) o al final (Repetir) de la secuencia de instrucciones. Entonces, en el primero, el bucle continúa mientras la condición es verdadera (la cual se comprueba antes de ejecutar la acción) y en el segundo, el bucle continúa hasta que la condición se hace verdadera (la condición se comprueba después de ejecutar la acción, es decir, se ejecutará al menos una vez).

La estructura Desde/Para suele utilizarse cuando se conoce con anterioridad el número de veces que se ejecutará la acción y se le conoce como Estructura Repetitiva en lugar de iterativa, para diferenciarla de las dos anteriores.

Las estructuras Mientras y Para/Desde suelen en ciertos casos, no realizar ninguna iteración en el bucle, mientras que Repetir ejecutará el bucle al menos una vez.

Existe otro caso de estructura conocida como Salto (Goto), la cual no es muy recomendable de usar ya que su uso dificulta la legibilidad de un programa y tiende a confundir por el hecho de recurrir a numerosas etiquetas o números de línea.

Representación gráfica:



Observa en el flujograma, que se necesita una variable contadora (un índice), para llevar la cuenta de las veces que entramos al cuerpo del ciclo. También es importante notar que esta variable se inicializa antes de entrar al cuerpo del ciclo y dentro del cuerpo se incrementa en una cantidad constante, por lo general en uno.

Esta variable a la vez, nos sirve para compararla con el valor dado en la condición, cuando se cumple la condición, se sale del ciclo.

Representación pseudocodificada:

Español	Inglés
Mientras <condición>	While <condición> do
Acciones	Acciones
Fin_mientras	end_while

EJEMPLO:

Calcular la suma de los cuadrados de los primeros 100 números enteros y escribir el resultado.

Solución.

Como recordarás, resolvimos este ejercicio en la lección anterior pero utilizando la estructura Desde. Hoy lo haremos con la estructura Mientras. Que tendremos de diferente?

Hagamos el algoritmo:

Inicio

Suma \leftarrow 0

$i \leftarrow 1$

Mientras $i \leq 100$ hacer

Suma \leftarrow suma + $i * i$

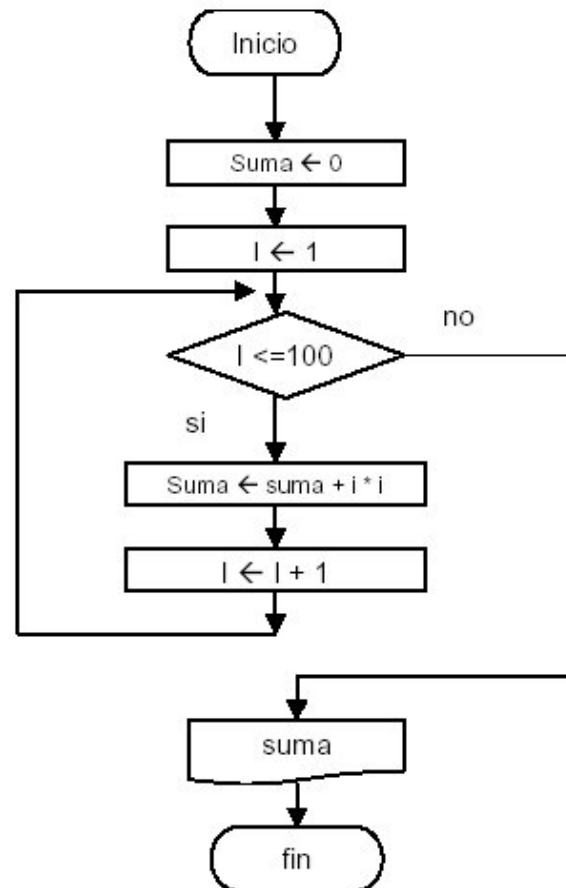
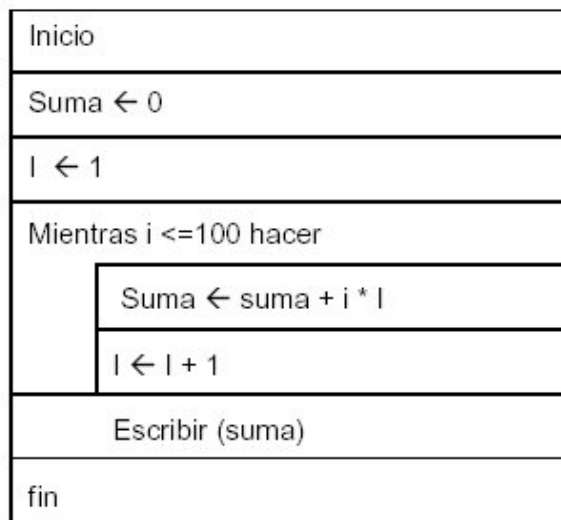
$i \leftarrow i + 1$

Fin_mientras

Escribir (suma)

Fin

Flujograma:



CENTINELAS Y BANDERAS.

Cuando no se conoce a priori el número de iteraciones que se van a realizar, el ciclo puede ser controlado por centinelas.

CENTINELAS.

En un ciclo While controlado por tarea, la condición de While especifica que el cuerpo del ciclo debe continuar ejecutándose mientras la tarea no haya sido completada.

En un ciclo controlado por centinela el usuario puede suspender la introducción de datos cuando lo desee, introduciendo una señal adecuada llamada centinela. Un ciclo Repetir controlado por centinela es cuando el usuario digita una letra para salir como por ejemplo S

o N para indicar si desea continuar o no. El bucle debe repetirse hasta que la respuesta del usuario sea "n" o "N".

Cuando una decisión toma los valores de -1 o algún posible valor que no esté dentro del rango válido en un momento determinado, se le denomina centinela y su función primordial es detener el proceso de entrada de datos en una corrida de programa.

Por ejemplo, si se tienen las calificaciones de un test (comprendida entre 0 y 100); un valor centinela en esta lista puede ser -999, ya que nunca será una calificación válida y cuando aparezca este valor se terminará de ejecutar el bucle.

Si la lista de datos son números positivos, un valor centinela puede ser un número negativo. Los centinelas solamente pueden usarse con las estructuras Mientras y Repetir, no con estructuras Desde/Para. ¿PODRÍAS DECIR POR QUÉ?

Ejemplo:

Suponga que debemos obtener la suma de los gastos que hicimos en nuestro último viaje, pero no sabemos exactamente cuántos fueron.

Si definimos gasto1, gasto2, gasto3, ..., -1 donde gastoi: real es el gasto número i y sumgas: real es el acumulador de gastos efectuados. -1 es el centinela de fin de datos.

Algoritmo:

```
Inicio
Sumgas . 0
Leer (gasto)
Mientras gasto <> -1 hacer
Sumgas . sumgas + gasto
Leer (gasto)
Fin_mientras
Escribir (sumgas)
Fin
```

BANDERAS.

Conocidas también como interruptores, switch, flags o conmutadores, son variables que pueden tomar solamente dos valores durante la ejecución del programa, los cuales pueden ser 0 ó 1, o bien los valores booleanos True o False. Se les suele llamar interruptores porque cuando toman los valores 0 ó 1 están simulando un interruptor abierto/cerrado o encendido/apagado.

Ejemplo 1:

Leer un número entero N y calcular el resultado de la siguiente serie: $1 - 1/2 + 1/3 - 1/4 + \dots + (-1)^N / N$.

Algoritmo:

```
Inicio
Serie . 0
I . 1
Leer (N)
Band . "T"
Mientras I <= N hacer
Si band = "T" entonces
Serie . serie + (1/I)
Band . "F"
Sino
Serie . serie - (1/I)
Band . "T"
Fin_si
I . I + 1
Fin_mientras
```

Escribir (serie)
Fin

Ejemplo 2.

Obtener suma de los términos de la serie: 2, 5, 7, 10, 12, 15, 17, 1800.

Sumser de tipo entero, es el acumulador de términos de la serie

Band de tipo carácter, es variable auxiliar que indica si al siguiente término de la serie hay que sumarle 3 ó 2.

Algoritmo:

```
Inicio
I ← 2
Sumser ← 0
Band ← "T"
Mientras (I <= 1800) hacer
Sumser ← sumser + I
Escribir (I)
Si band = "T" entonces
I ← I + 3
Band ← "F"
Sino
I ← I + 2
Band ← "T"
Fin_si
Fin_mientras
Escribir (sumser)
Fin
```

RESUMEN

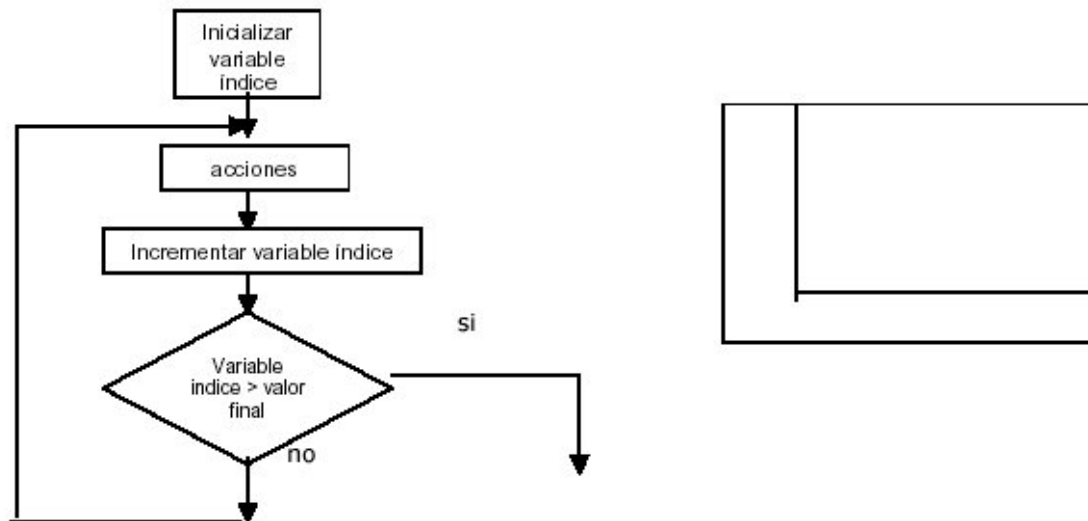
Hemos estudiado en esta lección que la estructura Mientras tiene una pequeña variante a la estructura Desde en cuanto a la representación algorítmica. Recuerda que la estructura Desde, se inicializa automáticamente en su sintaxis y el incremento también es automático. En cambio, la estructura Mientras usa un contador que es inicializado antes de entrar al ciclo y dentro del ciclo es incrementado. También estudiamos que los centinelas son valores que le damos a la condición para forzar a que un ciclo pueda terminar. También decíamos que los centinelas solamente los podemos usar en estructuras Mientras y Repetir, ya que sirven para finalizar el ciclo cuando no sabemos las veces que lo vamos a realizar, y la estructura Desde es usada cuando ya conocemos a priori el número de veces que se va a realizar el ciclo.

17. Estructuras iterativas. Estructura repetir.

[<http://www.mailxmail.com/...curso-aprende-programar/estructuras-iterativas-estructura-repetir>]

Se llama Repetir a la estructura algorítmica que se ejecuta un número definido de veces hasta que la condición se torna verdadera:

Representación gráfica:



Representación pseudocodificada :

Español

Repetir
Acciones
Hasta que <condición>

Inglés

Repeat
Acciones
until <condición>

EJEMPLO:

Calcular la suma de los cuadrados de los primeros 100 números enteros y escribir el resultado.

Solución.

Nuevamente resolveremos el ejercicio de las dos lecciones anteriores, ahora utilizando la estructura Repetir. ¿Podrás decir cuál será ahora la diferencia? Las reglas para construcción de esta estructura usando Repetir, nos dicen que debemos declarar una variable contador que debe inicializarse antes del ciclo e incrementarse dentro del ciclo. A diferencia de la estructura Mientras, la condición ahora estará colocada al final del bucle para que primero ejecutamos la instrucción y luego preguntamos si la condición se cumple. Esto quiere decir, que en esta estructura el bucle se realizará por lo menos una vez. También podrás observar que la condición está al revés, porque el bucle se repite hasta que la condición se cumpla. En el bucle Mientras, la condición se evaluaba mientras era cierta.

Hagamos el algoritmo:

Inicio

Suma \leftarrow 0

$i \leftarrow$ 1

Repetir

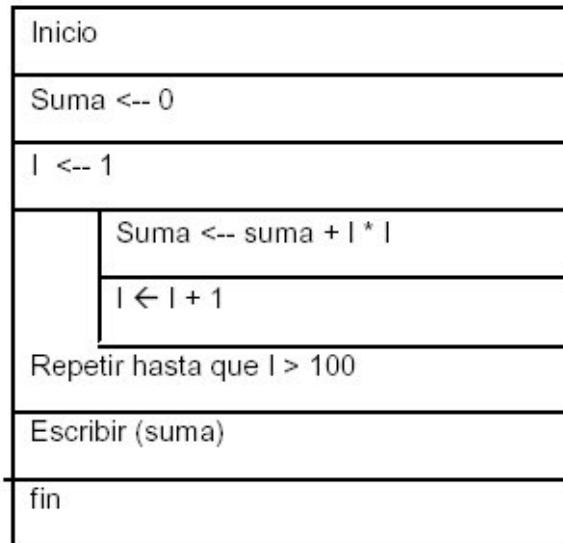
Suma \leftarrow suma + $i * i$

$i \leftarrow i + 1$

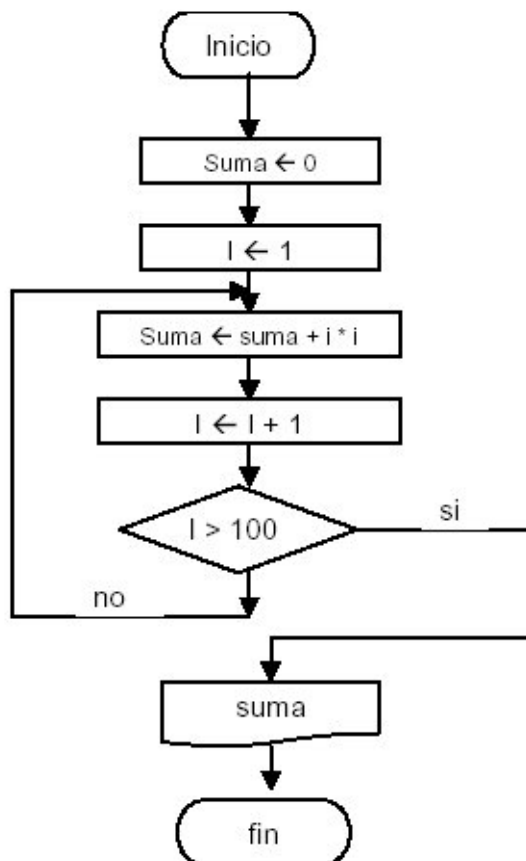
Hasta que $i > 100$

Escribir (suma)

Fin



Flujograma:



ACTIVIDAD PARA DESARROLLAR EN CLASE.

El ejercicio que resolvimos en la lección anterior:

Obtener suma de los términos de la serie: 2, 5, 7, 10, 12, 15, 17, 1800.

Resuélvelo usando la estructura Repetir. Elabora una tabla de seguimiento para unos cuantos datos. Escribe las características de cada una de las estructuras.

Escribe semejanzas y diferencias entre las estructuras:

Desde - Mientras y Repetir.



18. Estructuras de datos y arreglos.

[<http://www.mailxmail.com/curso-aprende-programar/estructuras-datos-arreglos>]

Introducción

Todas las variables que se han considerado hasta ahora son de tipo simple. Una variable de tipo simple consiste de una sola caja de memoria y sólo puede contener un valor cada vez. Una variable de tipo estructurado consiste en toda una colección de casillas de memoria. Los tipos de datos estudiados: entero, real, alfabético son considerados como datos de tipo simple, puesto que una variable que se define con alguno de estos tipos sólo puede almacenar un valor a la vez, es decir, existe una relación de uno a uno entre la variable y el número de elementos (valores) que es capaz de almacenar. En cambio un dato de tipo estructurado, como el arreglo, puede almacenar más de un elemento (valor) a la vez, con la condición de que todos los elementos deben ser del mismo tipo, es decir, que se puede tener un conjunto de datos enteros, reales, etc.

Datos estructurados:

Estructura de Datos es una colección de datos que se caracterizan por su organización y las operaciones que se definen en ella. Los datos de tipo estándar pueden ser organizados en diferentes estructuras de datos: estáticas y dinámicas.

Estructura de Datos estáticas:

Son aquellas en las que el espacio ocupado en memoria se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa. Corresponden a este tipo los arrays y registros

Estructuras de Datos Dinámicas:

Son aquellas en las que el espacio ocupado en memoria puede ser modificado en tiempo de ejecución. Corresponden a este tipo las listas, árboles y grafos. Estas estructuras no son soportadas en todos los lenguajes. La elección de la estructura de datos idónea dependerá de la naturaleza del problema a resolver y, en menor medida, del lenguaje. Las estructuras de datos tienen en común que un identificador, nombre, puede representar a múltiples datos individuales.

Arrays:

Un arreglo (array) es una colección de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común. Para referirse a un determinado elemento de un array se deberá utilizar un índice, que especifique su posición relativa en el array. Un arreglo es una colección finita, homogénea y ordenada de elementos. *Finita*: Todo arreglo tiene un límite; es decir, debe determinarse cuál será el número máximo de elementos que podrán formar parte del arreglo. *Homogénea*: Todos los elementos del arreglo deben ser del mismo tipo. *Ordenada* Se puede determinar cuál es el primer elemento, el segundo, el tercero,.... y el n-ésimo elemento.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así se tienen los:

- Unidimensionales (vectores)
- Bidimensionales (tablas o matrices)
- Multidimensionales (tres o más dimensiones)

Elemento 1
Elemento 2
Elemento 3
.....
Elemento n

Array Unidimensional

Elemento 1,1	Elemento 1,n
Elemento 2,1	Elemento 2,n
Elemento 3,1	Elemento 3,n
.....
Elemento m,1	Elemento m,n

Array Bidimensional

Elemento 1,1,1	Elemento 1,n,1
Elemento 2,1,1	Elemento 2,n,1
Elemento 3,1,1	Elemento 3,n,1
.....
Elemento m,1,1	Elemento m,n,1

Array Multidimensional

PROBLEMA.

Suponga que se desea desarrollar un programa para:

1. Leer una lista de calificaciones de un examen
2. Encontrar su media
3. Escribir una lista de las calificaciones mayores que la media
4. Ordenar la lista de las calificaciones en orden ascendente.

Supongamos también que hay 100 calificaciones. Debemos utilizar 100 variables diferentes nota1, nota2, ..., nota100, de ese modo son 100 direcciones diferentes de memoria para almacenar las calificaciones del examen. Se imagina declarar las 100 variables, ¿cuántas instrucciones involucra?

Var Nota1,nota2,nota3,.....nota100: entero

(En la declaración real de un programa no pueden usarse puntos suspensivos, por lo tanto serán 100 veces) . En la fase de lectura de datos, serán también 100 veces las instrucciones para ir leyendo cada valor. Leer (nota1, nota2,nota3,....., nota100)

Para calcular la media:

Media $\rightarrow (nota1 + nota2 + \dots + nota100) / 100$

Para la lista de calificaciones mayores que la media, deberá también irse comparando una por una:

Si $nota1 > media$ entonces

escribir (nota1)

Fin_si

Si $nota2 > media$ entonces

escribir (nota2)

Fin_si

Si $nota100 > media$ entonces

escribir (nota100)

Fin_si

Y después de más de 450 líneas de código..... ¡Falta ordenar la lista de calificaciones en orden ascendente!

Después que aprendas a usar arreglos verás cómo se ahorra instrucciones porque es fácil recorrer toda la lista de notas con unas pocas instrucciones. En el caso anterior, cuando el acceso a la información es secuencial, sólo se puede acceder a un elemento buscando desde el principio de la lista, y esto es algo lento. Lo que se necesita es una estructura de acceso directo que permita almacenar y recuperar los datos directamente especificando su posición en la estructura, de esa manera se requerirá el mismo tiempo para acceder al elemento de la posición 100 que el de la posición 5.

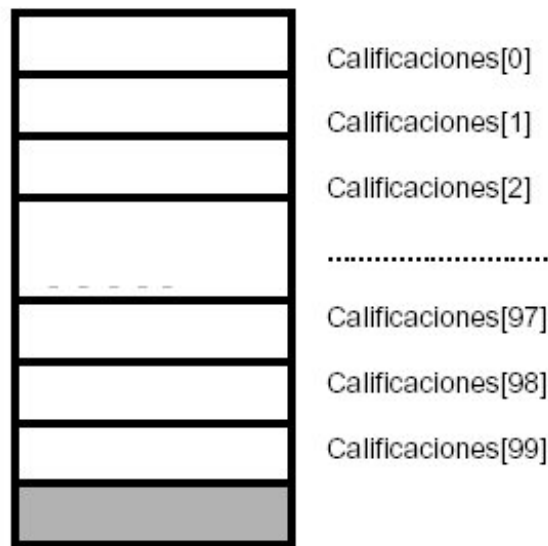
También preferiremos que esta estructura se almacene en memoria principal para que su almacenaje y recuperación sea más rápida. Es por ello que existen los arreglos, que están organizados en una secuencia de elementos, todos del mismo tipo y se puede acceder a cada elemento directamente especificando su posición en esta secuencia.

Arreglos Unidimensionales:

Están formados por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos. Al declarar un arreglo, se debe inicializar sus elementos antes de utilizarlos. Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener. Por ejemplo, las siguientes instrucciones declaran tres arreglos distintos:

```
Float costo_partes[50];
```

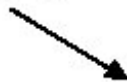
```
Int edad_empleados[100];  
Float precios_acciones[25];
```



```
Int calificaciones[100]
```



Tipo del arreglo



Tamaño del arreglo

Para acceder a valores específicos del arreglo, use un valor de índice que apunte al elemento deseado. Por ejemplo, para acceder al primer elemento del arreglo calificaciones debe utilizar el valor de índice 0 (calificaciones[0]). Los programas en C++ siempre indizan el primer elemento de un arreglo con 0 y el último con un valor menor en una unidad al tamaño del arreglo.

Inicialización y asignación de valores

Como se decía anteriormente, antes de utilizar un arreglo es necesario inicializarlo: Calificaciones[0];

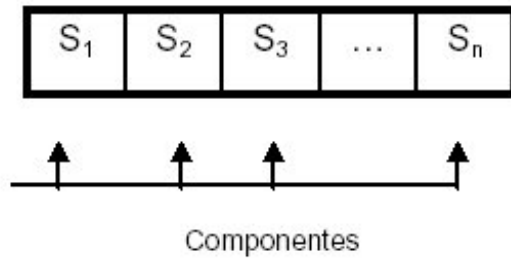
Para inicializar todos los elementos de una vez, se colocan dentro de una estructura for que va del primer elemento al último que contiene el arreglo. Para asignar un valor a un elemento del arreglo se hace por ejemplo: Calificaciones[0] = 100;

Cuando se usan arreglos, una operación común es usar una variable índice para acceder a los elementos de un arreglo. Suponiendo que la variable índice i contiene el valor 3, la siguiente instrucción asigna el valor 400 a valores[3]: valores[i] = 400;

Partes de un arreglo:

Los componentes. Hacen referencia a los elementos que forman el arreglo, es decir, a los valores que se almacenan en cada una de las casillas del mismo. Los índices. Permiten hacer referencia a los componentes del arreglo en forma individual, especifican cuántos elementos tendrá el arreglo y además, de qué modo podrán

accesarse esos componentes.



Definición de Arreglos:

$\text{ident_arreglo} = \text{arreglo}[\text{liminf} \dots \text{limsup}]$ de tipo Operaciones con Vectores:

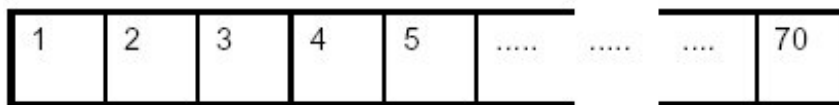
Las operaciones que se pueden realizar con vectores durante el proceso de resolución de un problema son:

- Lectura/ escritura
- Asignación
- Actualización (inserción, eliminación, modificación)
- Recorrido (acceso secuencial)
- Ordenación
- Búsqueda

Ejemplos:

Sea *arre* un arreglo de 70 elementos enteros con índices enteros. Su representación nos queda:

Arre = arreglo[1..70] de enteros

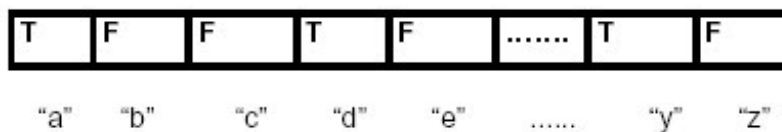


$\text{NTE} = 70 - 1 + 1 = 70$ elementos

Ejemplos:

Sea *bool* un arreglo de 26 elementos booleanos con índices de tipo caracter. Su representación nos queda:

Bool = arreglo["a".. "z"] de booleanos



Número total de elementos:

$\text{NTE} = (\text{ord}(\text{"z"}) - \text{ord}(\text{"a"})) + 1 = 122 - 97 + 1 = 26$ elementos

Lectura

El proceso de lectura de un arreglo consiste en leer y asignar un valor a cada uno de sus elementos. Normalmente se realizan con estructuras repetitivas, aunque pueden usarse estructuras selectivas. Usamos los índices para recorrer los elementos del arreglo:


```
desde i = 1 hasta 70 hacer
  leer ( arre[i])
fin_desde
```

Escritura:

Es similar al caso de lectura, sólo que en vez de leer el componente del arreglo, lo escribimos.

```
leer (N)
desde i = 1 hasta N hacer
  escribir (arre[i])
fin_desde
```

Asignación:

No es posible asignar directamente un valor a todo el arreglo; sino que se debe asignar el valor deseado en cada componente. Con una estructura repetitiva se puede asignar un valor a todos los elementos del vector.

Por ejemplo:

```
arre[1] ← 120 (asignación de un valor constante único a una casilla del vector)
arre[3] ← arre[1] / 4 (asignar una operación)
```

Se puede asignar un valor constante a todos los elementos del vector: desde i = 1 hasta 5 hacer

```
  arre[i] ← 3
fin_desde
```

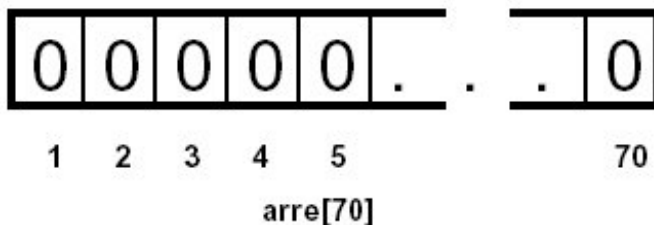
O bien

```
arre ← 3 (con arre del tipo arreglo)
```

Inicialización

Para inicializar con cero todos los elementos del arreglo:

```
desde i = 1 hasta 70 hacer
  arre[i] ← 0
fin_desde
```

**Acceso Secuencial.** (Recorrido)

El acceso a los elementos de un vector puede ser para leer en él o para escribir (visualizar su contenido). Recorrido del vector es la acción de efectuar una acción general sobre todos los elementos de ese vector.

Actualización.

Incluye añadir (insertar), borrar o modificar algunos de los ya existentes. Se debe tener en cuenta si el arreglo está o no ordenado. Añadir datos a un vector consiste en agregar un nuevo elemento al final del vector, siempre que haya espacio en memoria.

Investigue cómo insertar o eliminar elementos en un arreglo:

a) Ordenado

b) No ordenado

19. Matrices y cadenas de caracteres.

[<http://www.mailxmail.com/curso-aprende-programar/matrices-cadenas-caracteres>]

Arreglo Bidimensional:

Es un conjunto de datos homogéneo, finito y ordenado, donde se hace referencia a cada elemento por medio de dos índices. El primero se utiliza para los renglones (filas) y el segundo para las columnas. También puede definirse como un arreglo de arreglos. Internamente en memoria se reservan $M \times N$ posiciones consecutivas para almacenar todos los elementos del arreglo.

Declaración de una matriz:

Id_arreglo=arreglo[liminf..limsup,liminfc..limsupc] de tipo

Filas ↑ ↑ columnas

Por ejemplo: $1 \leq I \leq M$

$1 \leq J \leq N$

Se representa en forma de una tabla:

	1	2	J	N
1				
2				
.....				
I				
M				

Pseudocódigo para el recorrido por filas:

Const

M=valor1

N= valor2

Tipo

Array[1..M,1..N] de real:matriz

Var

Matriz:A

Desde i = 1 hasta M hacer

Desde j = 1 hasta N hacer

Escribir (A[i,j])

Fin_desde

Fin_desde

El recorrido por columnas se hace de manera similar, invirtiendo el sentido de los

índices.

```
Desde j = 1 hasta N hacer
    Desde i = 1 hasta M hacer
        Escribir (A[i,j])
    Fin_desde
```

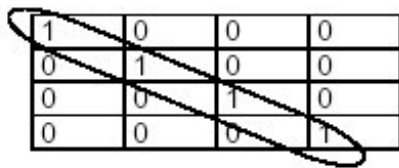
Fin_desde

El número de elementos que contendrá una fila viene dado por $U1-L1+1$ (Valor mayor - valor menor +1). Igualmente, el número de elementos para la columna es $U2-L2+1$. Así, el número total de elementos de la tabla es $(U2-L2+1)*(U1-L1+1)$

Ejemplos.

1) Rellenar una matriz identidad de 4 por 4 elementos.

Una matriz identidad es aquella en la que la diagonal principal está llena de unos y el resto de los elementos son cero. Para llenar la matriz identidad se debe verificar que cuando los índices i y j sean iguales, la posición vale 1, en caso contrario se asigna cero al elemento i,j .



1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Algoritmo

Inicio

```
Desde i = 1 hasta 4 hacer
    Desde j = 1 hasta 4 hacer
        Si i=j entonces
            Matriz[i,j] =1
        Sino
            Matriz[i,j] =0
    Fin_si
Fin_desde
```

Fin

Cadenas de Caracteres.

Una cadena es un conjunto de caracteres incluido el espacio en blanco, que se almacena en un área contigua de la memoria central. La longitud de una cadena es el número de caracteres que contiene. Una cadena vacía es la que no tiene ningún carácter. Una constante de tipo cadena es un conjunto de caracteres válidos encerrados entre comillas. Una variable de cadena es aquella cuyo contenido es una cadena de caracteres. El último carácter de la cadena marca el fin de la cadena.

Las variables de cadena se dividen en:

- Estáticas. Su longitud se define antes de ejecutar el programa y no puede cambiarse a lo largo de este.
- Semiestáticas. Su longitud puede variar durante la ejecución del programa, pero sin sobrepasar un límite máximo declarado al principio.
- Dinámicas. Su longitud puede variar sin limitación dentro del programa.

Operaciones básicas con cadenas:

- Asignación.

Nombre → "Luis Humberto"

- Entrada/ Salida

Leer(nombre, estado_civil)
Escribir(nombre, apellido)
Escribir(nombre, apellido)

· Cálculo de la longitud de una cadena. Es dar el número de caracteres que hay en una cadena que está entre comillas, incluyendo los espacios en blanco.
Comparación de cadenas: es comparar caracteres o cadenas de caracteres para ver si son iguales o no.

Según el código ASCII hay un orden de caracteres, así "A" es menor que "C". El valor de A es 65 y el de C es 67 Concatenación: es unir varias cadenas de caracteres en una sola, conservando el orden de los caracteres de cada una de ellas.

Cuando se combinan dos cadenas con el operador de concatenación, la segunda cadena se agregará directamente al final de la primera. En códigos postales y números telefónicos se suele usar caracteres ya que no se necesita operar los números y así podemos usar el guión.

Strtelefono = "1-515-555-1212"

Extracción de subcadenas. Subcadena es una porción de una cadena. Para extraer una subcadena se deben tener:

La cadena fuente de la cual se va a extraer la subcadena.

Pos que es un número que indica la posición inicial de la subcadena Long que indica el tamaño que tendrá la subcadena.

Búsqueda de información. Consiste en buscar una subcadena o cadena dentro de otra mayor. Nos devuelve el número de la posición donde inicia la cadena buscada, cero si no la encuentra. Dicho número se puede asignar a una variable entera (numérica).

Encontrar el punto medio

Truncar cadenas

Convertir cadenas a números o viceversa

Insertar una cadena dentro de otra

Borrar cadenas

Sustituir una cadena por otra.

Invertir el orden de una cadena.

Ejemplo.

El siguiente algoritmo sustituye las e por *.

Inicio

 Escribir ("escriba una palabra")

 Leer (str)

 Desde i=1 hasta len(str) hacer

 Si str[i] = 'e' entonces

 Str[i] = '*'

 Fin_si

 Fin_desde

 Escribir (str)

Fin

20. Modularidad. Procedimientos y funciones.

[<http://www.mailxmail.com/curso-aprende-programar/modularidad-procedimientos-funciones>]

Habíamos visto la programación estructurada que permite la escritura de programas fáciles de leer y modificar. En esta programación, el flujo lógico se gobierna por las estructuras de control básicas vista hasta hoy: secuenciales, repetitivas y de selección. La programación modular permite la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de resolver y que pueden ser tratados separadamente unos de otros. Gracias a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurándose sus errores antes de su inclusión en el programa principal y almacenarse para su posterior utilización cuantas veces se precise.

Módulo

Uno de los elementos principales de programación utilizados en la representación de cada módulo es la subrutina. Una subrutina es un conjunto de instrucciones de cómputo que realizan una tarea. Un programa principal llama a estos módulos a medida que se necesitan. Un módulo es un segmento, rutina, subrutina, subalgoritmo o procedimiento, que puede definirse dentro de un algoritmo con el fin de ejecutar una tarea específica y puede ser llamado o invocado desde el algoritmo principal cuando sea necesario. Los módulos son independientes en el sentido de que ningún módulo puede tener acceso directo a cualquier otro módulo, con excepción del módulo al que llama y sus propios submódulos. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera a ellos el control. Los módulos tienen una entrada y una salida. Se pueden tomar decisiones dentro de un módulo que tenga repercusión en todo el flujo, pero el salto debe ser únicamente hacia el programa principal. Al descomponer un programa en módulos independientes más simples se conoce también como el método de "Divide y vencerás".

¿Cuándo es útil la modularización?

Este enfoque de segmentación o modularización es útil en dos casos:

1. Cuando existe un grupo de instrucciones o una tarea específica que deba ejecutarse en más de una ocasión.
2. Cuando un problema es complejo o extenso, la solución se divide o segmenta en módulos que ejecutan partes o tareas específicas.

Ventajas de la Programación Modular:

Como los módulos son independientes, el desarrollo de un programa se puede efectuar con mayor facilidad, ya que cada módulo se puede crear aisladamente y varios programadores podrán trabajar simultáneamente en la confección de un algoritmo, repartiéndose las distintas partes del mismo. Se podrá modificar un módulo sin afectar a los demás. Las tareas, subalgoritmos, sólo se escribirán una vez, aunque se necesiten en distintas ocasiones a lo largo del algoritmo. El uso de módulos facilita la proyección y la comprensión de la lógica subyacente para el programador y el usuario. Aumenta la facilidad de depuración y búsqueda de errores en un programa ya que éstos se pueden aislar fácilmente. El mantenimiento y la modificación de la programación se facilitan. Los módulos reciben diferentes nombres:

- Funciones en C, C++
- Subrutinas en Basic
- Procedimientos y funciones en Pascal

- Subrutinas en Fortran y
- Secciones en Cobol.

Desarrollar programas de forma modular:

Significa que pueden identificarse las principales tareas a realizar por el programa y que se pueden diseñar y probar procedimientos individuales para estas tareas. Por ejemplo: ¿Qué transacciones se le hacen a una cuenta de ahorros?

Transacciones:

- depósito (cheque y efectivo)
- intereses
- retiro
- estado de cuenta
- cambio de libreta

Tiempo de vida de los datos

Según el lugar donde son declaradas puede haber dos tipos de variables.

Globales: las variables permanecen activas durante todo el programa. Se crean al iniciarse éste y se destruyen de la memoria al finalizar. Pueden ser utilizadas en cualquier procedimiento o función.

Locales: las variables son creadas cuando el programa llega a la función o procedimiento en la que están definidas. Al finalizar la función o el procedimiento, desaparecen de la memoria. Si dos variables, una global y una local, tienen el mismo nombre, la local prevalecerá sobre la global dentro del módulo en que ha sido declarada. Dos variables locales pueden tener el mismo nombre siempre que estén declaradas en funciones o procedimientos diferentes.

Parámetros Formales

Es un tipo especial de variables en un procedimiento a los que se pueden pasar valores desde el exterior del procedimiento. Se declaran en la cabecera del procedimiento.

Ejemplos:

1. Uso de variables globales en procedimientos o funciones.

Algoritmo global var x:entero

Inicio

```
    x ← 0
    cambiar
    escribir (x)
```

fin

Módulo cambiar

inicio

```
    x ← 1
```

fin

La variable X está definida como global, por lo tanto la salida será 1.

2. Uso de variables locales.

Algoritmo local

var x:entero

Inicio

```
    x ← 0
    cambiar
    escribir (x)
```

fin

Módulo cambiar

var x:entero

inicio

```

    x ← 1
fin

```

Como x es local, no tiene efecto en el programa, por lo tanto la salida será 0.

3. Variables locales y globales.

Programa en Borland C++

```
/* Variables globales y locales. */
```

```
#include stdio.h
```

```
int num1=1;
```

```
main() /* Escribe dos cifras */
```

```
{
    int num2=10;
    printf("%d\n",num1);
    printf("%d\n",num2);
}
```

Parámetros por Valor

Son los parámetros que pueden recibir valores pero que no pueden devolverlos. Es una variable global que se conecta con una variable local mediante el envío de su valor, después de lo cual ya no hay relación. Lo que le sucede a la variable local no afectará a la global. Cuando un parámetro actual se pasa por valor, el subprograma hace una copia del valor de éste en una posición de memoria idéntica en tamaño pero distinta en ubicación a la del parámetro actual y la asigna al parámetro formal correspondiente. Como el subprograma trabaja a partir de sus parámetros formales, si durante la ejecución se modifica el valor de un parámetro formal correspondiente a un paso por valor, el contenido de la posición de memoria del parámetro actual no se verá alterado.

Ejemplo:

Algoritmo parámetro valor

```
var x: entero
```

```
Inicio
```

```
    x = 0
```

```
    cambiar(x)
```

```
    escribir(X)
```

```
Fin
```

```
Módulo cambiar (y:entero)
```

```
inicio
```

```
    y ← 1
```

```
fin
```

```
Salida 0
```

Parámetros por Variable

Son los que pueden recibir y devolver valores. Son variables globales que se conectan con una local a través de su contenido; al establecerse dicha conexión las variables se convierten en sinónimos, lo que afecte a la variable local le sucederá a la variable global.

Ejemplo:

Algoritmo parámetro variable

```
var x:entero
```

```
Inicio
```

```
    x ← 0
```

```
    cambiar (x)
```

```
    escribir (x)
```



```
Fin
Módulo cambiar (var y:entero)
inicio
    y ← 1
fin
Salida: x = 1
```

PROCEDIMIENTOS

Son subprogramas, es decir, módulos que forman parte de un programa y realizan una tarea específica. Un procedimiento puede tener sus propias variables que se declaran en la sección var del propio procedimiento. Estas se llaman variables locales. La casilla de memoria para estas variables se crea cada vez que el procedimiento es llamado y se borran al salir del mismo. Así, las variables locales para un procedimiento sólo se pueden usar en el cuerpo del procedimiento y no en el cuerpo principal del programa.

FUNCIONES

La función es una estructura autónoma similar a los módulos. La diferencia radica en que la función se usa para devolver un solo valor de un tipo de dato simple a su punto de referencia. La función se relaciona especificando su nombre en una expresión, como si fuera una variable ordinaria de tipo simple. Las funciones se dividen en estándares y definidas por el usuario.

- **Estándar:** Son funciones proporcionadas por cualquier lenguaje de programación de alto nivel, y se dividen en aritméticas y alfabéticas.
- **Definidas por el usuario:** son funciones que puede definir el programador con el propósito de ejecutar alguna función específica, y que por lo general se usan cuando se trata de hacer algún cálculo que será requerido en varias ocasiones en la parte principal del algoritmo.

Ejemplos:

```
Función factorial (n:entero):entero
var i,factorial:entero
inicio
```

```
    si n <= 1 entonces
        factorial <-- 1
```

```
sino
```

```
    factorial <-- 1
```

```
desde i = 1 hasta n hacer
```

```
    factorial <-- factorial * i
```

```
fin_desde
```

```
fin_si
```

```
fin
```

Evaluar la función $f = x! / (y!(x-y)!)$

Algoritmo hallarf

```
var x,y:entero
```

```
f:real
```

```
inicio
```

```
    leer (x,y)
```

```
    f <-- factorial (x)/(factorial (y)* factorial (x-y))
```

```
    escribir ("El valor de f es:", f)
```

```
fin
```

Semejanzas entre Procedimientos y Funciones.

- La definición de ambos aparece en la sección de subprogramas de la parte de

declaraciones de un programa y en ambos casos consiste en una cabecera, una parte de declaraciones una parte de instrucciones.

- Ambos son unidades de programa independientes. Los parámetros, constantes y variables declarados en una función o procedimiento son locales a la función o al procedimiento, solamente son accesibles dentro del subprograma.
- Cuando se llama a una función o a un procedimiento, el número de los parámetros reales debe ser el mismo que el número de los parámetros formales y los tipos de los parámetros reales deben coincidir con los tipos de los correspondientes parámetros formales, con una excepción: se puede asociar un parámetro real de tipo entero con un parámetro formal por valor de tipo real.

Diferencias entre Procedimientos y Funciones.

- Mientras que a un procedimiento se le llama mediante una instrucción de llamada a procedimiento, a una función se la llama usando su nombre en una expresión.
- Puesto que se debe asociar un valor al número de una función, también se le debe asociar un tipo. Por tanto, la cabecera de una función debe incluir un identificador de tipo que especifique el tipo del resultado. Sin embargo, no se asocia ningún valor con el nombre de un procedimiento y, por tanto, tampoco ningún tipo.
- Las funciones normalmente devuelven un único valor a la unidad de programa que la llama. Los procedimientos suelen devolver más de un valor, o pueden no devolver ninguno si solamente realizan alguna tarea, como una operación de salida.
- En los procedimientos, los valores se devuelven a través de parámetros por variable, pero el valor de una función se devuelve mediante la asignación al nombre de la función de dicho valor en la parte de instrucciones de la definición de la función.

21. Métodos de ordenamiento y búsqueda.

[<http://www.mailxmail.com/curso-aprende-programar/metodos-ordenamiento-busqueda>]

ORDENAMIENTO.

Uno de los procedimientos más comunes y útiles en el procesamiento de datos, es la clasificación u ordenación de los mismos. Se considera ordenar al proceso de reorganizar un conjunto dado de objetos en una secuencia determinada. Cuando se analiza un método de ordenación, hay que determinar cuántas comparaciones e intercambios se realizan para el caso más favorable, para el caso medio y para el caso más desfavorable.

La colocación en orden de una lista de valores se llama **Ordenación**. Por ejemplo, se podría disponer una lista de valores numéricos en orden ascendente o descendente, o bien una lista de nombres en orden alfabético. La localización de un elemento de una lista se llama búsqueda.

Tal operación se puede hacer de manera más eficiente después de que la lista ha sido ordenada.

Existen varios métodos para ordenamiento, clasificados en tres formas:

Intercambio

Selección

Inserción.

En cada familia se distinguen dos versiones: un método simple y directo, fácil de comprender pero de escasa eficiencia respecto al tiempo de ejecución, y un método rápido, más sofisticado en su ejecución por la complejidad de las operaciones a realizar, pero mucho más eficiente en cuanto a tiempo de ejecución. En general, para arreglos con pocos elementos, los métodos directos son más eficientes (menor tiempo de ejecución) mientras que para grandes cantidades de datos se deben emplear los llamados métodos rápidos.

Intercambio

El método de intercambio se basa en comparar los elementos del arreglo e intercambiarlos si su posición actual o inicial es contraria inversa a la deseada. Pertenece a este método el de la burbuja clasificado como intercambio directo. Aunque no es muy eficiente para ordenar listas grandes, es fácil de entender y muy adecuado para ordenar una pequeña lista de unos 100 elementos o menos. Una pasada por la ordenación de burbujeo consiste en un recorrido completo a través del arreglo, en el que se comparan los contenidos de las casillas adyacentes, y se cambian si no están en orden. La ordenación por burbujeo completa consiste en una serie de pasadas ("burbujeo") que termina con una en la que ya no se hacen cambios porque todo está en orden.

Ejemplo:

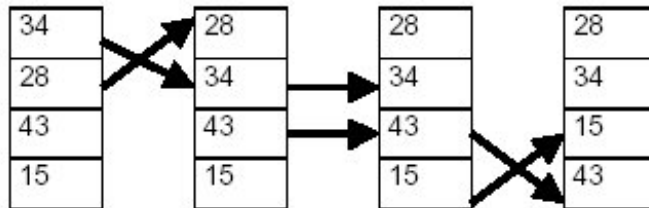
Supóngase que están almacenados cuatro números en un arreglo con casillas de memoria de $x[1]$ a $x[4]$. Se desea disponer esos números en orden creciente. La primera pasada de la ordenación por burbujeo haría lo siguiente:

Comparar el contenido de $x[1]$ con el de $x[2]$; si $x[1]$ contiene el mayor de los números, se intercambian sus contenidos.

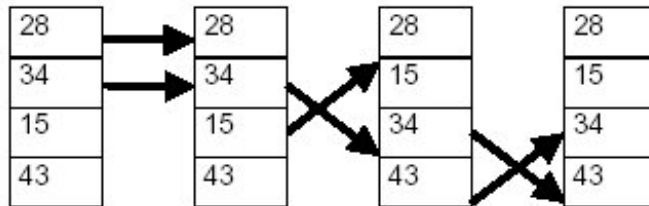
Comparar el contenido de $x[2]$ con el de $x[3]$; e intercambiarlos si fuera necesario.

Comparar el contenido de $x[3]$ con el de $x[4]$; e intercambiarlos si fuera necesario.

Al final de la primera pasada, el mayor de los números estará en $x[4]$.

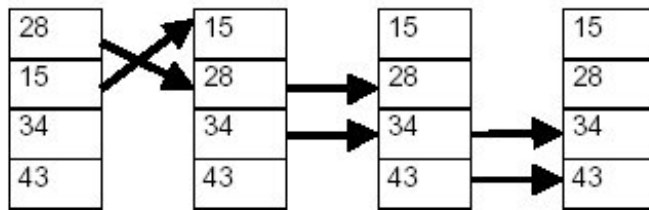


Al final de la segunda pasada, los últimos dos elementos estarán ordenados como se muestra:

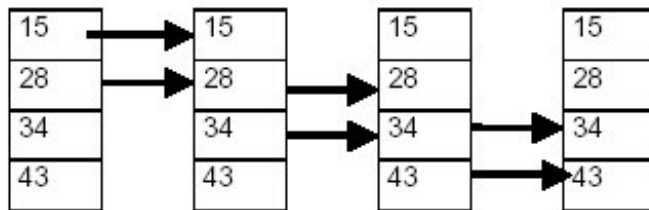


Al final de la tercera pasada, los últimos tres elementos estarán ordenados como se muestra:





Nótese que el arreglo ya quedó ordenado, sin embargo, se hace una pasada más porque la computadora no advierte que el arreglo está en orden hasta que ocurre una pasada sin intercambios.



La ordenación por burbujeo se llama así porque los números más pequeños ascienden como burbujas hasta la parte superior, mientras que los mayores se hunden y caen hasta el fondo. Está garantizado que cada pasada pone el siguiente número más grande en su lugar, aunque pueden colocarse más de ellos en su lugar por casualidad.

Quicksort.

Si bien el método de la burbuja era considerado como el peor método de ordenación simple o menos eficiente, el método Quicksort basa su estrategia en la idea intuitiva de que es más fácil ordenar una gran estructura de datos subdividiéndolas en otras más pequeñas introduciendo un orden relativo entre ellas. En otras palabras, si dividimos el array a ordenar en dos subarrays de forma que los elementos del subarray inferior sean más pequeños que los del subarray superior, y aplicamos el método reiteradamente, al final tendremos el array inicial totalmente ordenado. Existen además otros métodos conocidos, el de ordenación por montículo y el de shell.

Selección.

Los métodos de ordenación por selección se basan en dos principios básicos:

Seleccionar el elemento más pequeño (o más grande) del arreglo.

Colocarlo en la posición más baja (o más alta) del arreglo.

A diferencia del método de la burbuja, en este método el elemento más pequeño (o más grande) es el que se coloca en la posición final que le corresponde.

Inserción.

El fundamento de este método consiste en insertar los elementos no ordenados del arreglo en subarreglos del mismo que ya estén ordenados. Dependiendo del método elegido para encontrar la posición de inserción tendremos distintas versiones del método de inserción.

BÚSQUEDA.

La búsqueda es una operación que tiene por objeto la localización de un elemento

dentro de la estructura de datos. A menudo un programador estará trabajando con grandes cantidades de datos almacenados en arreglos y pudiera resultar necesario determinar si un arreglo contiene un valor que coincide con algún valor clave o buscado.

Siendo el array de una dimensión o lista una estructura de acceso directo y a su vez de acceso secuencial, encontramos dos técnicas que utilizan estos dos métodos de acceso, para encontrar elementos dentro de un array: búsqueda lineal y búsqueda binaria.

Búsqueda Secuencial:

La búsqueda secuencial es la técnica más simple para buscar un elemento en un arreglo. Consiste en recorrer el arreglo elemento a elemento e ir comparando con el valor buscado (clave). Se empieza con la primera casilla del arreglo y se observa una casilla tras otra hasta que se encuentra el elemento buscado o se han visto todas las casillas. El resultado de la búsqueda es un solo valor, y será la posición del elemento buscado o cero. Dado que el arreglo no está en ningún orden en particular, existe la misma probabilidad de que el valor se encuentra ya sea en el primer elemento, como en el último. Por lo tanto, en promedio, el programa tendrá que comparar el valor buscado con la mitad de los elementos del arreglo.

El método de búsqueda lineal funciona bien con arreglos pequeños o para arreglos no ordenados. Si el arreglo está ordenado, se puede utilizar la técnica de alta velocidad de búsqueda binaria, donde se reduce sucesivamente la operación eliminando repetidas veces la mitad de la lista restante.

Búsqueda Binaria.

La búsqueda binaria es el método más eficiente para encontrar elementos en un arreglo ordenado. El proceso comienza comparando el elemento central del arreglo con el valor buscado. Si ambos coinciden finaliza la búsqueda. Si no ocurre así, el elemento buscado será mayor o menor en sentido estricto que el central del arreglo. Si el elemento buscado es mayor se procede a hacer búsqueda binaria en el subarray superior, si el elemento buscado es menor que el contenido de la casilla central, se debe cambiar el segmento a considerar al segmento que está a la izquierda de tal sitio central.

22. Bibliografía.

[<http://www.mailxmail.com/curso-aprende-programar/bibliografia>]

" Fundamentos de Informática

Luis A. Ureña y otros. 1999 Editorial Alfaomega ra-ma

" Fundamentos de Programación. Libro de Problemas

Luis Joyanes Aguilar, Rodríguez Baena y Fernández Azuela. 1996

Editorial Mc Graw Hill 2ª edición.

" Fundamentos de Programación. Algoritmos y estructuras de datos. 2ª Edición

Luis Joyanes Aguilar 1996. Editorial Mc Graw Hill

" Pascal y Turbo Pascal enfoque práctico

Luis Joyanes Aguilar y Angel Hermoso. Editorial Mc Graw Hill

" Lógica Simbólica.

Irving M. Copi. Editorial C.E.C.S.A.

" Programación en Pascal. 4ª Edición

Sanford Leestma y Larry Nyhoff. Editorial Prentice Hall 1999

" Cómo programar en C/C++

Deitel/Deitel. Editorial Prentice Hall

" Introducción a la Ciencia de las Computadoras.

Tremblay, Jean Paul. Edit. Mc Graw Hill 1985

" Introducción a la Computación para Ingenieros

Steven C. Chapra y Raymond P. Canale

Edit. Mc Graw Hill 1993

" Curso General de Informática.

Javier Gayan y Dolores Segarra. Edit. Gustavo Gili S.A. 1988

" Aprenda Visual C++ Ya.

Mark Andrews. Editorial Mc Graw Hill

" Visual C++ 6.0 Manual de Referencia.

Chris Pappas y William H. Murray. Editorial Mc Graw Hill

" Computación. Metodología, Lógica Computacional y Programación.

Ma. del Rosario Bores Rangel y Román Rosales Becerril. 1ª Edición Editorial Mc Graw Hill 1993

" Programación en C. 1ª Edición

Byron S. Gottfried. Editorial Mc Graw Hill 1991

" Metodología de la Programación. Algoritmos, Diagramas de Flujo y programas

Tomos I y II. Osvaldo Cairó battistutti. 1a Edición

Editorial Alfaomega 1995

" Enciclopedia del Lenguaje C. 1ª Edición

Francisco Javier Ceballos. Editorial Alfaomega ra-ma 1999

DIRECCIONES DE INTERNET:

<http://www.geocities.com/joseluisdl>
<http://www.gorkasweb.com/>
<http://mundovb.net/mundoc/>
<http://lawebdelprogramador.com/cursos/>

Visita más cursos como este en mailxmail:

[<http://www.mailxmail.com/cursos-informatica>]
[<http://www.mailxmail.com/cursos-programacion>]



¡Tu opinión cuenta! Lee todas las opiniones de este curso y déjanos la tuya:
[<http://www.mailxmail.com/curso-aprende-programar/opiniones>]

Cursos similares

Cursos	Valoración	Alumnos	Vídeo
Holística informática Aprende con nuestro curso de holística informática, sobre los avances informáticos que te ayudará a actualizarte conociendo más acer... [28/04/09]		2.223	
PHP y MySQL. Aplicaciones Web: HTML II (tercera parte) Programación de aplicaciones Web con PHP y MySQL. Ahora continuaremos con el estudio de las páginas Web HTML. Estudiaremos las listas en HTML. ... [02/12/08]		1.015	
Introducción al lenguaje Pascal Pascal es un lenguaje de alto nivel y de propósito general (es aplicable a un gran número de aplicaciones diversas) desarrollado por el profesor suizo Niklaus Wirth como ... [01/03/06]		19.357	
WML. Internet para móvil (segunda parte) Internet para móvil y el lenguaje WML son las claves de este curso. En esta ocasión estudiaremos la creación del WML y las variables m&a... [07/07/09]		580	
Metodología de la Programación Este curso gratis le proporcionará, a modo de iniciación, algunos de los pasos a seguir para aprender a programar. Los bucles o los operadores lógicos son algunos de los c... [23/05/03]		51.893	