ANTOLOGIA

ELEMENTOS DE PROGRAMACIÓN ESTRUCTURADA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

CUARTO CUATRIMESTRE

Marco Estratégico de Referencia

ANTECEDENTES HISTORICOS

Nuestra Universidad tiene sus antecedentes de formación en el año de 1979 con el inicio de actividades de la normal de educadoras "Edgar Robledo Santiago", que en su momento marcó un nuevo rumbo para la educación de Comitán y del estado de Chiapas. Nuestra escuela fue fundada por el Profesor de Primaria Manuel Albores Salazar con la idea de traer Educación a Comitán, ya que esto representaba una forma de apoyar a muchas familias de la región para que siguieran estudiando.

En el año 1984 inicia actividades el CBTiS Moctezuma Ilhuicamina, que fue el primer bachillerato tecnológico particular del estado de Chiapas, manteniendo con esto la visión en grande de traer Educación a nuestro municipio, esta institución fue creada para que la gente que trabajaba por la mañana tuviera la opción de estudiar por las tarde.

La Maestra Martha Ruth Alcázar Mellanes es la madre de los tres integrantes de la familia Albores Alcázar que se fueron integrando poco a poco a la escuela formada por su padre, el Profesor Manuel Albores Salazar; Víctor Manuel Albores Alcázar en septiembre de 1996 como chofer de transporte escolar, Karla Fabiola Albores Alcázar se integró como Profesora en 1998, Martha Patricia Albores Alcázar en el departamento de finanzas en 1999.

En el año 2002, Víctor Manuel Albores Alcázar formó el Grupo Educativo Albores Alcázar S.C. para darle un nuevo rumbo y sentido empresarial al negocio familiar y en el año 2004 funda la Universidad Del Sureste.

La formación de nuestra Universidad se da principalmente porque en Comitán y en toda la región no existía una verdadera oferta Educativa, por lo que se veía urgente la creación de una institución de Educación superior, pero que estuviera a la altura de las exigencias de los jóvenes que tenían intención de seguir estudiando o de los profesionistas para seguir preparándose a través de estudios de posgrado.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de cuarenta

alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el Corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y Educativos de los diferentes Campus, Sedes y Centros de Enlace Educativo, así como de crear los diferentes planes estratégicos de expansión de la marca a nivel nacional e internacional.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de cuarenta alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y educativos de los diferentes campus, así como de crear los diferentes planes estratégicos de expansión de la marca.

MISIÓN

Satisfacer la necesidad de Educación que promueva el espíritu emprendedor, aplicando altos estándares de calidad Académica, que propicien el desarrollo de nuestros alumnos, Profesores, colaboradores y la sociedad, a través de la incorporación de tecnologías en el proceso de enseñanza-aprendizaje.

VISIÓN

Ser la mejor oferta académica en cada región de influencia, y a través de nuestra Plataforma Virtual tener una cobertura Global, con un crecimiento sostenible y las ofertas académicas innovadoras con pertinencia para la sociedad.

VALORES

- Disciplina
- Honestidad
- Equidad
- Libertad

ESCUDO

El escudo de la UDS, está constituido por tres líneas curvas que nacen de izquierda a derecha formando los escalones al éxito. En la parte superior está situado un cuadro motivo de la abstracción de la forma de un libro abierto.

ESLOGAN

"Mi Universidad"

ALBORES

Es nuestra mascota, un Jaguar. Su piel es negra y se distingue por ser líder, trabaja en equipo y obtiene lo que desea. El ímpetu, extremo valor y fortaleza son los rasgos que distinguen.

ELEMENTOS DE PROGRAMACIÓN ESTRUCTUADA

Objetivo de la materia:

Especificar sistemas secuenciales síncronos como autómatas de estados finitos. Implementar sistemas secuenciales síncronos utilizando bloques funcionales secuenciales (biestables, registros y contadores) y componentes combinacionales (puertas lógicas, memorias no volátiles y circuitos programables). Calcular el retardo de propagación y la frecuencia máxima de funcionamiento de una implementación dada de un sistema secuencial síncrono. Especificar circuitos combinacionales y secuenciales mediante el lenguaje de descripción de hardware VHDL.

UNIDAD I

FUNDAMENTOS DE PROGRAMACIÓN

1.1Introducción

1.2 Programas y algoritmos

1.3 ¿Qué es un algoritmo?

1.4 Diagrama de flujo

1.5 El pseudocódigo

1.6 Comentarios

1.7 Las variables

1.8 Estructuras de control

1.9 Importancia de la programación de computadoras

1.10 Clasificación de los lenguajes de programación

1.11 Lenguaje de programación de bajo nivel

1.12 Diseño de algoritmos

UNIDAD II

ELEMENTOS DEL LENGUAJE DE PROGRAMACIÓN

2.1 Introducción al entorno de programación

2.2 Características principales

2.3 Estructura básica de un programa

2.4 Palabras reservadas

2.5 Variables

2.6 Constantes

2.7 Tipos de datos

2.7.1 Simples

2.7.2 Compuestos (abstractos)

2.8 Despliegue y formateo de datos

2.9 Operadores aritméticos, lógicos y relacionales

2.10 Control de flujo

2.11 Ciclos

UNIDAD III

PROGRAMACIÓN MODULAR

3.1 Declaración de funciones

3.2 Paso de parámetros por valor o por referencia

3.3 Simples

3.4 Con parámetros

3.5 Uso de bibliotecas de funciones

3.6 Tipos de bibliotecas de funciones

3.6.1 java.lang

3.6.2 java.io

3.6.3 java.net

3.6.4 java.until

3.6.5 java.awt

3.6.6 java.applet

3.6.7 java.math

3.6.8 java.rmi

3.6.9 java.text

3.6.10 java.sql

3.6.11 javax.swing

3.7 Entrada y salida

3.8 Archivos

3.9 Cadenas

3.10 Definición e importancia de los arreglos en la programación

3.11 Declaración de arreglos unidimensionales y multidimensionales

3.12 Arreglos unidimensionales y multidimensionales

3.13 Estructuras de selección

3.14 Estructuras de repetición

3.15 Estructura de múltiple selección

UNIDAD IV

APLICACIÓN DE PUERTOS DE COMUNICACIÓN

4.1 Puertos de comunicación

4.2 Tipos de puertos de una computadora

4.3 Puertos de audio analógico. rca

4.4 Puerto vga

4.5 Puerto dvi (interfaz de vídeo digital)

4.6 Puerto hdmi

4.7 Puerto s/pdfi

4.8 Puerto firewire

4.9 Puerto de módem

4.10 Especificaciones de los puertos rs-232 y paralelo

4.11 Envío y recepción de datos

Índice

UNIDAD I FUNDAMENTOS DE PROGRAMACIÓN

El alumno conocerá los fundamentos necesarios para realizar programación de estructuras y clases.

1.1INTRODUCCIÓN

Los fundamentos de programación son las bases **comunes** a todos los programas. Es lo primero que tendrás que aprender incluso antes de elegir el programa con el que quieres programar.

Lo primero que tienes que saber es que el ordenador es una máquina eléctrica y solo entiende el llamado código binario (1 y 0).

1 = hay corriente
0 = No hay corriente

Este es su lenguaje. Entendernos con él mediante este código es muy difícil, por eso los lenguajes de programación se dividen en dos tipos diferentes dependiendo de su cercanía al lenguaje del ordenador.

El código del ordenador se basa en asignar a cada caracter (letra, signo, número, etc) una combinación de 8 ceros y unos (8 bits = byte) mediante un código que se llama ASCII. Por ejemplo, la letra A se representa con lo combinación siguiente: 01100001.

Los lenguajes más cercanos al idioma del ordenador, llamados de bajo nivel, son muy complicados (casi como el código del ordenador) y poco usados. El más conocido es el código o lenguaje máquina, un código que el ordenador puede interpretar directamente. Aquí tienes un ejemplo:

8B542408 83FA0077 06B80000 0000C383

De este tipo de lenguajes, que solo suelen utilizar los que programan los ordenadores para su uso, no vamos hablar aquí. Hablaremos de los conocimientos comunes a los lenguajes de alto nivel.

Los **lenguajes de programación de alto nivel** permiten dar órdenes al ordenador con un lenguaje parecido al nuestro (Visual Basic, Pascal, Logo, C++, JavaScript, etc.) y siempre o casi siempre en inglés.

Hay programas de alto nivel como el GML o el Java que son programas interpretados, es decir, se analizan y ejecutan las instrucciones por el propio programa directamente. Otros necesitan un **compilador**, pero eso no es un problema, solo es un programa (software) que **se encarga de traducir el programa hecho en lenguaje de programación al código del ordenador** para que lo entienda.

Con un tipo u otro es igual, lo importante es que los lenguajes, como todo, hay que aprendérselos, pero tienen una ventaja, y es que tienen muchos puntos en común. Estos puntos son lo que vamos a estudiar aquí, los **fundamentos de programación** común a cualquier lenguaje de alto nivel.

Una vez aprendidos los fundamentos, tendrás que elegir el lenguaje que quieras usar, pero con estos conocimientos, todos te resultarán muy fáciles de aprender, solo tendrás que aprender unas cuantas instrucciones en inglés.

Aquí te dejamos un enlace con los lenguajes más comunes y para qué se usan. Te puede servir de guía para elegir el lenguaje que quieres aprender. Eso sí primero estúdiate los fundamentos que aquí presentamos. Lenguajes de Programación.

También te recomendamos este libro explicado muy bien y fácil, con el que por solo 4 euros podrás ampliar conocimientos: Fundamentos de Programación Para Todos los Públicos

1.2 PROGRAMAS Y ALGORITMOS

Los lenguajes de programación, cuentan todos en su haber con un juego de "instrucciones". Una instrucción no es más que una orden que nosotros le damos a la máquina.

Y es que, al fin y al cabo, **un programa no es más que una secuencia de instrucciones (escritas en algún lenguaje de programación) pensado para resolver algún tipo de problema**. Hay que tener claro que, si no sabemos resolver este problema, no podremos escribir el programa. Si no sabemos que es una suma, sería casi imposible hacer un programa para que nos sume dos números, a no ser que alguien nos ayudara.

A ti se te puede ocurrir una manera de resolverlo, a tu compañero, otra, lo importante es que las dos formas de resolverlo lleven al mismo resultado, la suma.

La forma con el que resolvéis el problema, es lo que se llama **algoritmo**, y es lo que vamos a ver a continuación.

1.3 ¿QUÉ ES UN ALGORITMO?

Un algoritmo es una secuencia de PASOS a seguir para resolver un problema.

Por ejemplo, cuando quiero ver una película de vídeo, podría hacer los siguientes pasos (algoritmo):

- Elijo una película de las de mi colección.
-Compruebo SI TV y vídeo están conectados a la red (y procedo).
-SI la TV está apagada, la enciendo, SI NO, pues no. Y lo mismo con el vídeo.
-Introduzco la película en el vídeo. Dejo el estuche sobre el vídeo.
-SI la TV no está en el canal adecuado, la cambio, SI NO, pues no.

-Cojo los mandos a distancia (el del TV y el del vídeo).

-Me pongo cómodo.

-Pulso PLAY en el mando del vídeo.

Fíjate bien en unos detalles que son fundamentales y que aparecen en este algoritmo:

- La descripción de cada paso no me lleva a ambigüedades, los **pasos** son absolutamente **explícitos y no inducen a error**.

- **El número de pasos es finito**. Tienen un principio y un fin. Según lo visto, una mejor definición de algoritmo sería:

"**Un algoritmo es una sucesión finita de pasos (no instrucciones como en los programas) no ambiguos que se pueden llevar a cabo en un tiempo finito**."

Este "lenguaje" el algoritmo está escrito en nuestro idioma, pero ahora necesitamos acercarnos a un poco más al lenguaje del ordenador. Pero **el primer paso para realizar un programa es sacar su algoritmo**, es como explicar lo que queremos que haga nuestro programa. Ahora entiendes por qué decíamos antes que, si no sabemos nosotros resolver el problema, no podríamos crear el programa. No seríamos capaces de hacer su algoritmo.

Ahora que ya tenemos el algoritmo, para el siguiente paso se puede utilizar dos formas: Sacar el Diagrama de Flujo del algoritmo o su pseudocódigo. Algunos programadores hacen los dos.

1.4 DIAGRAMA DE FLUJO

Un diagrama de flujo es una representación gráfica del algoritmo. Expresamos los pasos del algoritmo mediante un esquema con unos símbolos establecidos.

Un diagrama de flujo debe proporcionar una información clara, ordenada y concisa de todos los pasos a seguir.

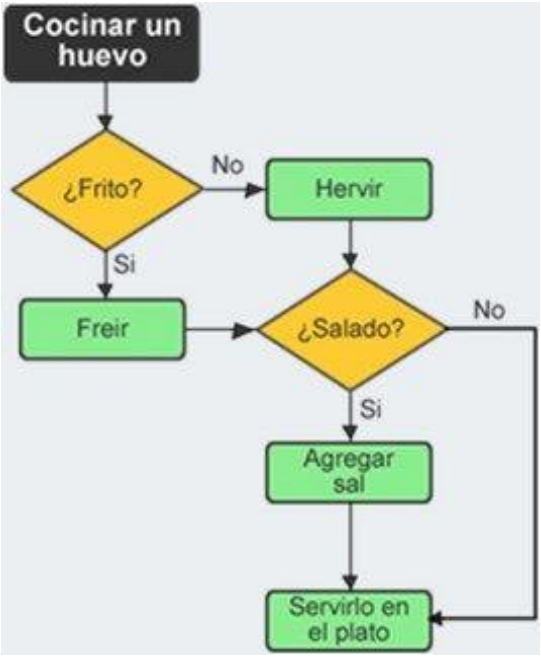
Veamos un ejemplo:

Un algoritmo para cocinar un huevo para otra persona sería:

Pregunto si quiere el huevo frito.

- Si me dice que si, lo frio, si me dice que no, lo hago hervido.
- Una vez cocinado le pregunto si quiere sal en el huevo.
- Si me dice que no lo sirvo en el Plato. Si me dice que si le hecho sal y después lo sirvo en el plato.

Ahora que ya sabemos todos los pasos, mediante el algoritmo, podemos hacer un esquema con estos pasos a seguir. Este esquema será el Diagrama de Flujo.



Si uno tiene experiencia puede prescindir del algoritmo escrito, pero siempre tendremos que tenerlo en mente para hacer el diagrama de flujo sin equivocarnos.

Como nosotros nos centraremos más en el pseudocódigo, no hablaremos más de los diagramas de flujo. Si quieres aprender hacer diagramas de flujo te recomendamos este enlace: diagramas de flujo.

1.5 EL PSEUDOCÓDIGO

El pseudocódigo es una forma de escribir los pasos, pero de la forma más cercana al lenguaje de programación que vamos a utilizar, es como un falso lenguaje, pero en nuestro idioma, en el lenguaje humano.

Una de las mayores dificultades con las que se encuentran los hispanoparlantes que empiezan a programar es el idioma. Por eso es bueno utilizar el pseudocódigo, algo así como un falso lenguaje de programación en español, que ayuda a asimilar con más facilidad las ideas básicas.

Por ejemplo, si queremos escribir algo en pantalla, en pseudocódigo podríamos poner:

Escribir "Hola" o Escribir 20+30.

También podemos usar:

mostrar por pantalla "Hola"

Realmente el pseudocódigo lo podríamos escribir como nosotros quisiéramos, ya que realmente no es el programa en sí, solo es una ayuda para posteriormente realizar el programa mediante el lenguaje de programación que utilicemos, eso sí, es de gran ayuda, tanto que es imprescindible. Pero, aunque lo podamos escribir de cualquier forma, la mayoría de los programadores suelen usar un vocabulario en común. Este vocabulario será el que veamos aquí.

Recuerda que el pseudocódigo para un programador es fundamental.

Si sabes hacer el pseudocódigo del programa, pasarlo a cualquier lenguaje de programación es muy sencillo, solo tendrás que aprender los comandos equivalentes a las instrucciones en pseudocódigo.

Además, la mayoría de los lenguajes utilizan prácticamente los mismos comandos en su lenguaje.

Sigamos hablando un poco más sobre el pseudocódigo.
Para especificar el principio y el fin del programa pondremos:

Inicio
Aquí iría el programa en pseudocódigo
Fin

Otra forma muy utilizada sería:

Proceso NombreDelPrograma
Aquí iría el programa en pseudocódigo
FinProceso

Las 3 órdenes que más utilizaremos en pseudocódigo son:
Escribir --> Escribe en pantalla el texto que pongamos entre paréntesis o también puede escribir en pantalla el valor de una variable. Esta instrucción en casi todos los programas suele escribirse con la palabra write o document.write('Hola ');.
Leer Edad- -> nos lee desde lo que el usuario marque desde el teclado y guarda el valor, por ejemplo dentro de una variable, en este caso la variable Edad (luego veremos lo que son las variables).

En programación real suele utilizarse la instrucción input.

Calcular 3 x 5 -->Calcula valores

Teniendo el pseudocódigo o el Diagrama de Flujo lo tenemos muy fácil, ya que es fácilmente traducible a cualquier lenguaje de programación.

Según avancemos en el tema irás viendo ejemplos de pseudocódigo.
Ahora vamos a empezar con lo interesante, vamos a empezar aprender a programar.

1.6 COMENTARIOS

Poner comentarios de lo que vamos haciendo es muy útil, sobre todo cuando llega la hora de revisar el programa, si no, más de una vez nos encontraremos diciendo ¿qué hacía esto aquí? No cuesta nada documentar el programa y nos ahorrará dolores de cabeza. La norma que se sigue en todos los programas es poner // delante de los comentarios, para identificarlos:

// Esto será un comentario y no hará nada en el programa

Nosotros durante las explicaciones también pondremos comentarios.

1.7 LAS VARIABLES

Una variable es como una caja donde metemos cosas (datos). Estos datos los podemos ir cambiando, ahora meto un 3, ahora lo quito y meto un 5.

Una variable tiene un nombre, que puede ser una letra, una palabra, varias palabras unidas por el guión bajo o varias palabras sin separar, pero la primera letra de cada palabra en mayúsculas ejemplo.: VidasPerdidas, vidaperdidas, vidas_perdidas. Ojo las mayúsculas y minúsculas son muy importantes en las variables, no es la misma

variable número que Numero, son dos diferentes. OJO tampoco se pueden poner acentos en el nombre de las variables.

Las variables también tienen un valor que es lo que hay dentro de ella (en la caja) en ese momento y que puede ir variando según se vaya desarrollando el programa, por eso se llama variable.

Una variable dependiendo de su valor puede ser numérica, si solo puede tener un valor numérico, de texto, si solo puede contener texto (letra, palabra o frase también llamada string).

En las variables de texto, su valor (el texto), debe ir entre comillas, para diferenciar que el texto es texto y no es el nombre de otra variable. Por ejemplos vidas = "Cinco" o vidas = "5". En los dos casos el valor es un texto, nunca el valor de 5.

Las numéricas no llevan comillas en su valor. Por ejemplo: vidas = 5. En este caso su valor si que es el número 5.

Hay otras variables que se llaman booleanas que solo pueden tener dos valores true o false. Normalmente true se puede sustituir por el valor 1 y false por el 0.

Veamos algunos ejemplos de los tipos de variables:

Edad=3; //variable numérica. Fíjate que esto en negrita es un comentario.

VariableDeTexto= “Tengo 14 años”; //fíjate que va entre comillas.

VariableNumerica= Edad + 2 ; //su valor es el valor de la variable Edad (numérica) +2; en

este caso sería = 5 (3+2).

VariableBooleana = true; en este caso sería de valor 1

¿Te has dado cuenta que hemos puesto un punto y coma (;) al acabar de definir cada variable?. En programación siempre que se acaba una instrucción o grupo de instrucciones se debe poner ";" para decir al programa que pasamos a otra instrucción diferente. Pero sigamos con las variables.

En algunos lenguajes de programación, lo normal es declarar las variables al principio de un programa. Declarar no es más que decir "mira, yo quiero tres variables, y quiero que una se llame Nombre, otra Edad y otra Apellido". Además, se tiene que especificar qué tipo de variable es al declararla (no en todos los lenguajes). La declaración de variables se suele hacer al principio del programa, por eso es importante saber cuántas variables vamos a usar antes de escribir nuestro programa. Veamos un ejemplo:

VariableNumerica: Edad; //Declaramos las variables numéricas.
VariableTexto: Nombre, Apellido; //Declaramos las variables de texto.

Nota: En la mayoría de los programas las variables se declaran poniendo "var" delante y dándoles un valor inicial:

var Edad = 15; var Texto = "Hola", etc.

A partir de este momento, podrás meter su valor en cualquier parte del programa.

Edad = 5;

Nombre = "Juan";

Apellidos = "Rodriguez"

Después podrás cambiar su valor, dentro del programa, las veces que quieras.

En los programas que no hace falta declarar el tipo de variables podríamos empezar el programa simplemente metiendo los valores iniciales de las variables o ir poniendo las variables con sus valores iniciales durante el desarrollo del programa. Si un valor no lleva comillas el programa entiende que es numérica directamente. Este último caso es el más fácil para empezar, según surga una variable la ponemos con su valor inicial directamente en la parte del programa que sea.

Podemos sumar, restar, multiplicar, dividir y hacer cualquier tipo de operación matemática con las variables.

```
numerica: Pepe, Mari ,Juan //Declaramos las variables que usaremos;
Pepe=2;
 Mari=3;
 Juan = Pepe + Mari; // Juan tiene ahora el valor numérico de 5.
```

Los operadores matemáticos más usados en todos los lenguajes de programación (se usan los mismos) son los siguientes:

Operador relacional	*Significado*	*Ejemplo*
>	Mayor que	3>2
<	Menor que	"ABC"<"abc"
=	Igual que	4=3
<=	Menor o igual que	"a"<="b"
>=	Mayor o igual que	4>=5

Hay variables ya definidas por el propio lenguaje de programación que usemos, y cuyo nombre no se lo podremos dar a ninguna de las que nosotros definamos.

Las podemos usar, pero tal y como el lenguaje las definió. Por ejemplo, en muchos lenguajes mouse_x es la variable que tiene el valor de la posición x del ratón en cada momento, hspeed es la velocidad horizontal, etc.

Variables Locales y/o Globales

En muchos lenguajes, si queremos que la variable se use en todo el programa deberemos nombrarla como una variable global, en caso contrario, si no la definimos como global, por defecto el lenguaje la considerará una variable local.

Una variable local al salir del lugar donde la hemos asignado un valor, perderá ese valor y ya no existirá (al salir de un algoritmo, de un trozo de programa, del objeto, de una estructura IF, etc.).

En la mayoría de los lenguajes se pone la palabra global, un punto y detrás el nombre de la variable, de esta forma, esta variable la podemos usar en todas las partes del programa.

Ejemplo global.pepe, que será distinta de la variable pepe.

Nosotros en todos los ejemplos las usaremos como locales, por eso no verás nunca la palabra global.

1.8 ESTRUCTURAS DE CONTROL

Las estructuras de control tienen una finalidad bastante definida: su objetivo es ir señalando el orden en que tienen que sucederse los pasos de un algoritmo o de un programa.

Las estructuras de control son de tres tipos:

- Secuenciales
- Selectivas
- Repetitivas
- Empecemos por las primeras.
- Estructuras secuenciales

Una estructura de control secuencial, en realidad, no es más que escribir un paso del algoritmo detrás de otro, el que primero que se haya escrito será el que primero se ejecute.

Veamos un ejemplo: queremos leer el radio de un círculo, calcular su área y mostrar por pantalla al usuario el resultado.

En pseudocódigo sería:

numerica: radio, area; //Declaración de variables;

inicio

Escribir "dame el radio del circulo";

Leer radio // asignación del valor de la variable radio por el usuario por medio del teclado;

area =3.14159*radio; //nosotros asignamos el valor de la variable área con su fórmula;

Escribir "el área del circulo es:" //OJO En los texto SI PODEMOS Y DEBEMOS PONER ACENTOS;

Escribir area; // nos muestra en la pantalla el valor de la variable area resultado de la fórmula anterior;

fin

Como ves las instrucciones se van ejecutando unas detrás de otra hasta llegar al final.

Estructuras selectivas

Estas estructuras se utilizan para TOMAR DECISIONES (por eso también se llaman estructuras de decisión o alternativas). Lo que se hace es EVALUAR una condición, y, a continuación, en función del resultado, se lleva a cabo una opción u otra.

Alternativas simples (condicional IF)

Son los conocidos "si... entonces". Se usan de la siguiente manera: yo quiero evaluar una condición, y si se cumple (es decir, si es cierta), entonces realizaré una serie de pasos. Un ejemplo

Es decir la palabra if seguida de la condición y seguidamente, entre corchetes, lo que se realizará si se cumple la condición.

1.9 IMPORTANCIA DE LA PROGRAMACIÓN DE COMPUTADORAS.

El rol de la programación en los últimos sesenta años ha sido crucial para comprender la evolución que ha tenido la sistematización de tareas y el manejo de la información que hoy en día damos como un hecho**.** En efecto, la misma tiene como principal función el hecho de conseguir que innumerables trabajos que antes ejercíamos de forma manual y con un alto costo sean ejecutados por un ordenador con un ahorro significativo de tiempo. Por otro lado, dada la increíble cantidad de información que hoy en día se maneja para distintas tareas, la **programación** es una herramienta de enorme valor porque permite bucear en la misma con muchísima facilidad.

A lo largo de la historia, el hombre se ha visto en la obligación de realizar un número constante de tareas para poder sobrevivir. Con el paso del tiempo y el desarrollo de la **tecnología**, estas tareas fueron ejerciéndose cada vez con un mayor grado de **productividad**. El proceso de mejora fue lento pero continuo hasta la **revolución industrial**, momento en el cual vemos como existe una rápida sustitución de actividades manuales por el trabajo llevado a cabo mediante máquinas. Así, los bienes de capital fueron aumentando cada vez más la productividad, circunstancia que todavía está en proceso de

expansión. Con el desarrollo de las primeras computadoras, ya no solo el trabajo físico pudo reemplazarse por máquinas, sino también el trabajo intelectual. En efecto, las computadoras pueden realizar cada vez con mayor poder cifras enormes de cálculos complejos que tienen la posibilidad de procesar y generar datos para el beneficio humano. La **programación**, en particular, es la adaptación de ese potencial de las computadoras a las necesidades del hombre, generando distintos procesos automáticos que generan resultados que sirven desde diversos aspectos, como por ejemplo el laboral, el estético, el lúdico, etc.

A pesar de las ventajas antedichas, todavía existe un franco escepticismo de ciertos sectores en lo que respecta a este proceso. Cualquiera sea la actitud planteada ante el fenómeno, lo cierto es que es imposible de dejar de lado, continuándose así un reemplazo continuo de actividades humanas por la que llevan adelante máquinas. La **programación** es ante todo un emergente más de este proceso. No obstante, para evitar desconsuelos, cabe señalar que el proceso sería improcedente si no sirviera en alguna medida a la sociedad. En efecto, la baja continua de costos en lo que respecta a la **producción** de bienes y servicios tiene

Desarrollar un programa involucra una serie de pasos. El programador define que un problema, planifica una solución, codifica el programa, prueba el programa y, finalmente, realiza los documentos del programa.

Con el cambio acelerado que ocurre en el área tecnológica de las computadoras, la programación es un campo excitante y siempre desafiante.

Debido a esto es que posee varios beneficios aprender programación.

1.- El conocimiento de sistemas

Los programadores de computadora tienen una comprensión completa de lo qué es y por qué de los sistemas de cómputo, incluyendo limitaciones del sistema, puede establecer las expectativas realistas y puede evitar esas limitaciones, comprender completamente a maximizar el uso del equipo y sus accesorios.

2.- La Plataforma de Creatividad

La programación es una plataforma para realzar la creatividad, especialmente en la resolución de problemas. La programación crea o desarrolla nuevos vídeos juegos , gráficos y animaciones para presentar ideas comerciales nuevas o resolver un problema particular.

La programación, especialmente en el desarrollo web, ha conseguido nuevas aplicaciones interactivas en Internet que tienen acceso a los recursos del sistema y proveen el mismo nivel de control como aplicaciones de mesa. Usadas las plataformas de aprendizaje en línea, estas
aplicaciones han permitido el despego de programas de educaciones a distancia. En la actualidad, casi todas las principales instituciones educativas tienen alguna forma de implementación de aprendizaje en línea, todo esto es gracias a la programación de computadoras.

3.- Determina el Futuro

Los principios de programación de computadoras implementados hoy, probablemente influenciarán en los avances de las distintas áreas de la tecnología como ser: el reconocimiento de voz, la inteligencia artificial y otras tecnologías sofisticadas que cambiarán el futuro y que serán aplicados a nuestras vidas cotidianas. Por ejemplo, la tendencia hacia automatizar búsquedas de la Internet, las compras de manera más localizada está en curso. Mientras las plataformas en el desarrollo hardware jugarán un papel principal, la tecnología de la computadora probablemente estará en el centro de todo y los sistemas de futuros programadores serán un aspecto importante.

4.- El Lenguaje De Máquina

Desde que las computadoras trabajan con números, la programación le permite a una persona representar lenguaje de máquina en formato legible, de tal manera el lenguaje sea

como el de un humano. Esto reduce las oportunidades de introducir errores, aprovechando más el tiempo en codificar que estar eliminando errores.

1.10 CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN.

Conocer cómo funciona el lenguaje de programación y cómo se interrelaciona con nosotros a través de software nos permite mejorar nuestra productividad y conseguir ese algo que nos diferencie de la competencia.

A lo largo de los años, los lenguajes de programación han aumentado su potencia y flexibilidad para, de esa forma, llevar a cabo las tareas complejas que la innovación y las nuevas tecnologías de información y comunicación (TIC) nos exigen.

Todas las máquinas y dispositivos requieren un lenguaje de programación para cumplir sus funciones. Si conoces cuáles permiten que estos las realicen adecuadamente, entonces tendrás un plus que te permitirá alcanzar tus objetivos en menos tiempo.

¿Qué es un lenguaje de programación?

Es un lenguaje formal que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina.

Mediante este lenguaje se comunican el programador y la máquina, permitiendo especificar, de forma precisa, aspectos como:

- cuáles datos debe operar un software específico;
- cómo deben ser almacenados o transmitidos esos datos;
- las acciones que debe tomar el software dependiendo de las circunstancias variables.

Para explicarlo mejor (en otras y con menos palabras), **el lenguaje de programación es un sistema estructurado de comunicación**, el cual está conformado por conjuntos de símbolos, palabras claves, reglas semánticas y sintácticas que permiten el entendimiento entre un programador y una máquina.

Es importante recalcar que **existe el error común de usar como sinónimos el lenguaje de programación y el lenguaje informático**, pero ¿por qué no debemos confundirlos?

Pues, es debido a que el lenguaje de programación obedece a un conjunto de reglas que permiten expresar las instrucciones que serán interpretadas por el programador. Y el lenguaje informático comprende otros lenguajes que dan formato a un texto, pero no son programación en sí mismos.

Entonces, no todos los lenguajes informáticos son de programación, pero todos los lenguajes de programación son a la vez informáticos.

¿Qué tipos de lenguaje de programación existen?

El lenguaje de programación es la base para construir todas las aplicaciones digitales que se utilizan en el día a día y se clasifican en dos tipos principales: lenguaje de bajo nivel y de alto nivel.

1.11 LENGUAJE DE PROGRAMACIÓN DE BAJO NIVEL

Son lenguajes totalmente orientados a la máquina.

Este lenguaje sirve de interfaz y crea un vínculo inseparable entre el hardware y el software.

Además, ejerce un control directo sobre el equipo y su estructura física. Para aplicarlo adecuadamente es necesario que el programador conozca sólidamente el hardware. Éste se subdivide en dos tipos:

Lenguaje máquina

Es el más primitivo de los lenguajes y es una colección de dígitos binarios o bits (0 y 1) que la computadora lee e interpreta y son los únicos idiomas que las computadoras entienden.

Ejemplo: **10110000 01100001**

No entendemos muy bien lo que dice ¿verdad? Por eso, el lenguaje ensamblador nos permite entender mejor a qué se refiere este código.

Lenguaje ensamblador

El lenguaje ensamblador es el primer intento de sustitución del lenguaje de máquina por uno más cercano al utilizado por los humanos.

Un programa escrito en éste lenguaje es almacenado como texto (tal como programas de alto nivel) y consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.

Sin embargo, dichas máquinas no comprenden el lenguaje emsamblador, por lo que se debe convertir a lenguaje máquina mediante un programa llamado Ensamblador.

Este genera códigos compactos, rápidos y eficientes creados por el programador que tiene el control total de la máquina.

Ejemplo: **MOV AL, 61h** (asigna el valor hexadecimal 61 al registro "AL")

Lenguaje de programación de alto nivel

Tienen como objetivo facilitar el trabajo del programador, ya que utilizan unas instrucciones más fáciles de entender.

Además, **el lenguaje de alto nivel permite escribir códigos mediante idiomas que conocemos** (español, inglés, etc.) y luego, para ser ejecutados, se traduce al lenguaje de máquina mediante traductores o compiladores.

Traductor

Traducen programas escritos en un lenguaje de programación al lenguaje máquina de la computadora y a medida que va siendo traducida, se ejecuta.

Compilador

Permite traducir todo un programa de una sola vez, haciendo una ejecución más rápida y puede almacenarse para usarse luego sin volver a hacer la traducción.

¿Para qué sirven los lenguajes de programación?

En general un lenguaje de programación sirve para programar. Sin embargo cada uno tiene un alcance y forma de comunicación diferente.

En resumidas cuentas, el lenguaje de bajo nivel permite la comunicación interna de la máquina, cada instrucción tiene su código único de operación.

Y el lenguaje de alto nivel facilita la captación de instrucciones que el programador le da a la máquina, mientras que éste introduce datos en el idioma conocido la máquina lo va absorbiendo en lenguaje de máquinas mediante traductores o compiladores, permitiendo así:

- reducir el tiempo de programación;
- entender más fácilmente la tarea a realizar;
- permitir al programador desvincularse del funcionamiento interno de la máquina, entre otros.

En otras palabras, **el lenguaje de bajo nivel es cercano a los idiomas de las máquinas mientras que el lenguaje de alto nivel está más cerca del entendimiento e idioma humano**.

¿Qué softwares de programación existen?

Por software de programación entendemos el conjunto de todas las herramientas que le permiten al programador, crear, escribir códigos, depurar, mantener y empaquetar los proyectos.

Algunos de los distintos programas por los que pasará el proyecto para gestionarlo son:

Editores de código o texto

Al escribir los códigos se auto-completan marcando los errores sintácticos y la refactorización.

Compiladores

Como mencionados anteriormente, éstos traducen el código ingresado a lenguaje de máquina generando un código binario ejecutable.

Depuradores

Sirven para optimizar el tiempo de desarrollo mediante el monitoreo de la ejecución de un programa, el seguimiento a los valores de ciertas variables, las referencias a objetos en memoria y por ende, nos ayuda a corregir errores.

Enlazadores

Este programa toma objetos generados en los primeros pasos del proceso de compilación y los recursos necesarios de la biblioteca, quita aquellos procesos y datos que no necesita, y enlaza el código con dicha biblioteca para así aumentar su tamaño y extensión.

Interpretadores o traductores

Como leíste en este artículo, el traductor (o intérprete) carga el código ingresado y traduce las instrucciones para que el programa pueda ser ejecutado.

IDE

El IDE (Integrated Development Environment) o **Entorno de Desarrollo Integrado**, es una aplicación informática que proporciona una serie de servicios que facilitan la programación de software, tales como:

- funciones de autocompletado;
- un editor de código fuente;
- gestión de conexiones a bases de datos;
- integración con sistemas de control de versiones;
- simuladores de dispositivos;
- un depurador para agilizar el proceso de desarrollo de software, entre otros.

1.12 DISEÑO DE ALGORITMOS

Un **algoritmo** es un conjunto de acciones que especifican la secuencia de operaciones realizar, en orden, para resolver un problema.

Los algoritmos son independientes tanto del lenguaje de programación como del ordenador que los ejecuta.

Las características de los algoritmos son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

Ejemplo tradicional de un algoritmo: Cambiar la rueda pinchada de un coche.

Etapa de diseño

Aunque en la solución de problemas sencillos parezca evidente la **codificación** en un lenguaje de programación concreto, es aconsejable realizar el **diseño** del algoritmo, a partir del cual se codifique el programa.

La solución a problemas más complejos puede requerir muchos más pasos. Las estrategias seguidas usualmente a la hora de encontrar algoritmos para problemas complejos son:

- **Partición o divide y vencerás**: consiste en dividir un problema grande en unidades más pequeñas que puedan ser resueltas individualmente.
    - Ejemplo: Podemos dividir el problema de limpiar una casa en labores más simple correspondientes a limpiar cada habitación.
- **Resolución por analogía**: Dado un problema, se trata de recordar algún problema similar que ya esté resuelto. Los dos problemas análogos pueden incluso pertenecer áreas de conocimiento totalmente distintas.
    - Ejemplo: El cálculo de la media de las temperaturas de las provincias andaluzas y la media de las notas de los alumnos e una clase se realiza del mismo modo.

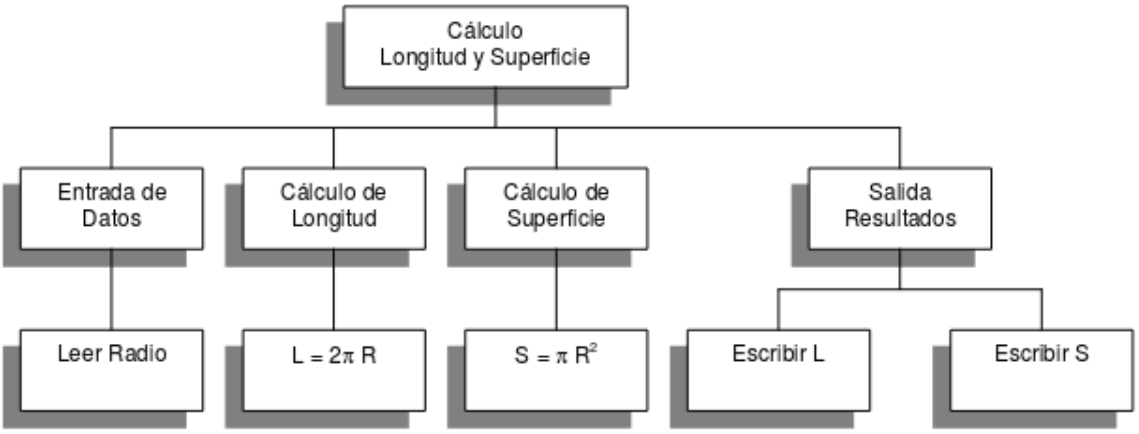
La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples se denomina **diseño descendente (top-down design).** Tras la primera descripción del problema (poco específica), se realiza una siguiente descripción más detallada con más pasos concretos. Este proceso se denomina **refinamiento del algoritmo**.

Ejemplo de diseño

Leer el radio de una circunferencia y calcular e imprimir su superficie y su circunferencia.

- Se puede dividir en tres subproblemas más sencillos:
    - Leer Radio
    - Calcular Superficie
    - Calcular Longitud
    - Escribir resultados
- Refinamiento del algoritmo:
    - Leer Radio
    - Superficie <- PI * Radio ^ 2
    - Longitud <- 2 * PI * Radio
    - Escribir Radio, Longitud, Superficie

Lo podemos ver en un **diagrama estructurado**:



Herramientas de representación de algoritmos

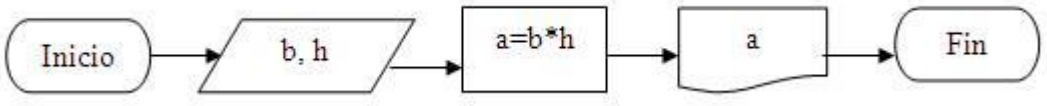
- Un **diagrama de flujo** es una de las técnicas de representación gráfica de algoritmos más antiguas. Ventajas: permite altos niveles de estructuración y modularización y es fácil de usar. Desventajas: son difíciles de actualizar y se complican cuando el algoritmo es grande.

- El **pseudocódigo**, nos permite una aproximación del algoritmo al lenguaje natural y por tanto un a redacción rápida del mismo.

El diagrama de flujo es la representación gráfica de un algoritmo; para ello se utiliza un conjunto de símbolos estándares mundialmente utilizados y desarrollados por organizaciones tales como ANSI (American National Institute) e ISO (International Standard Organization para la elaboración de diagramas de flujo;

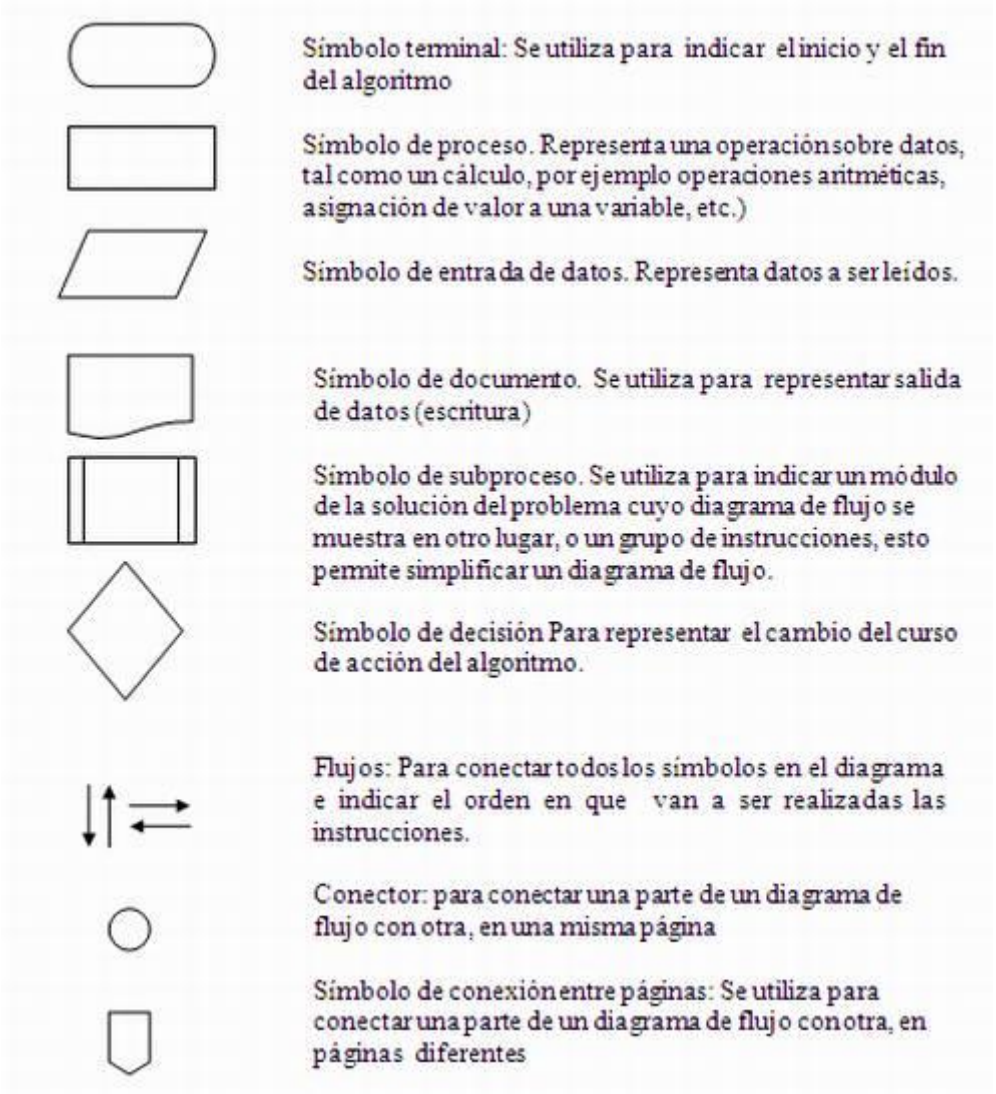
En el diagrama cada símbolo representa una acción en concreto; y cada instrucción del algoritmo se visualiza dentro del símbolo adecuado. Los símbolos se conectan con flechas para indicar el orden en que se ejecutan las instrucciones.

Por ejemplo, el siguiente diagrama de flujo corresponde al algoritmo para calcular el área del rectángulo



El ovalo rotulado con la palabra Inicio indica el comienzo del algoritmo, el paralelogramo es el símbolo de entrada de datos e indica que se lee el valor de la base (b) y el valor de la altura (h), el rectángulo es el símbolo de proceso e indica que se realiza un proceso sobre los datos de entrada para calcular el área (a) multiplicando la base por la altura, (utilizaremos el * como operador de multiplicación), el siguiente es el símbolo de salida y representa un documento e indica que se muestra el valor del área obtenido, en cualquier dispositivo de salida, finalmente el ovalo rotulado con la palabra Fin indica que se ha llegado al fin del algoritmo.

Los símbolos que se utilizaran para dibujar los diagramas de flujo son los siguientes:



Reglas para la construcción de diagramas de flujo

- 1. Todo diagrama de flujo debe tener un inicio y un fin.
- 2. Las líneas de flujo nunca deben cruzarse, para evitarlo deben utilizarse el símbolo conector.
- 3. Las líneas de flujo deben terminar siempre en un símbolo.
- 4. No puede llegar más de una línea de flujo a un símbolo.
- 5. Todos los símbolos en un diagrama deben estar conectados mediante una línea de flujo; todo símbolo debe tener una línea de flujo entrando y otra saliendo salvo el símbolo que indica inicio o fin del diagrama.

- 6. Como regla general el flujo del proceso debe mostrarse de izquierda a derecha y de arriba abajo.

Se recomienda mantener uniforme el tamaño de los símbolos, por lo que el texto que se escribe dentro no debe ser muy extenso, recuérdese que el propio símbolo indica la operación a realizar. La forma en que se capturan los datos de entrada o se muestran los datos de salida se detallarán al codificar el algoritmo en el lenguaje de programación. Esto mismo se recomienda para la representación del algoritmo en pseudocódigo.

MATERIAL WEB COMPLEMENTARIO

https://www.youtube.com/watch?v=Ptm84X-9X4o

UNIDAD II ELEMENTOS DEL LENGUAJE DE PROGRAMACIÓN

El alumno reconocerá los elementos necesarios para realizar programaciones orientadas a objetos y estructurales.

Sintaxis

Cada lenguaje de programación tiene unas reglas especiales para la construcción de programación, a esto se le denomina **sintaxis**

El compilador lee el programa y comprueba que el programa sigue las reglas de sintaxis del lenguaje de programación, el compilador traduce el código fuente de Java a un código máquina (código objeto)

Código Objeto

Consta de instrucciones máquina e información de cómo cargar el programa en la memoria antes de su ejecución

Si el compilador encuentra errores, los presentará en la pantalla, una vez corregidos los errores se vuelve a compilar sucesivamente hasta que no se produzcan más errores

Depuración

Los programas rara vez funcionan bien la primera vez que se ejecutan, por lo que los errores que se detectan deben ser corregidos

Al proceso de encontrar errores se denomina **depuración** del programa, esta tarea es de las más difíciles en el proceso de programación

Errores de sintaxis

Son aquellos que se producen cuando el programa viola la sintaxis, es decir, las reglas gramaticales del lenguaje

Errores lógicos

Son errores del programador en el diseño del algoritmo, son difíciles de encontrar y aislar, ya que no suelen ser detectados por el compilador

Errores de regresión

Son aquellos que son provocados cuando accidentalmente se producen al corregir un error lógico y se produce otro en otra parte del programa

Elementos léxicos de los programas

- Identificadores
- Palabras reservadas
- Literales
- Operadores
- Separadores

Identificador

Es una secuencia de caracteres, letras dígitos y subrayados (_)

El primer carácter debe de ser una letra, no un subrayado

Las letras mayúsculas y minúsculas son diferentes para cada identificador

Ejemplo:

Nombre

Nombre_clase

Cantidad_Total

Palabras Reservadas

Es aquella que tiene un significado especial para el lenguaje de programación

Una palabra reservada no puede ser utilizada como identificador, objeto o función

asm	case	const	do	explicit	friend
auto	catch	continue	double	extern	goto
bool	char	default	else	float	if
break	class	delete	enum	for	int

Signos de Puntuación y Separadores

Todas las sentencias deben de terminar con un punto y coma

Otros signos de puntuación son:

! % & * ( )-+= { } ~ ^ \ ; ´ : < > ? , . / “

Operadores en C

* multiplicación / división	Se evalúan primero, si hay muchas, se evaluan de izquierda a derecha
+ suma - resta	Se evalúan después, si hay muchas, se evaluan de izquierda a derecha
% módulo	Residuo de la división entera

Prioridad de Operadores Aritméticos:

Todas las expresiones con paréntesis anidados se evalúan de dentro a afuera, el paréntesis más interno se evalúa primero. Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1. ^ Exponentes
2. * Multiplicación, / División, % Módulo
3. + Suma, - Resta

Los Operadores en na misma expresión con igual nivel de prioridad, se evaluan de izquierda a derecha

Algebra

Estamos acostumbrados a representar algebraicamente una ecuación, pero en la

computadora es diferente, para ello podemos utilizar los paréntesis, de lo contrario el resultado puede ser diferente

Ejemplos:

m=(a+b+c+d+e)/5

m=a+b+c+d+e/5

Ecuación de una línea recta

y=mx+b

y=m*x+b

Ejercicio:

z=prq+w/x-y

Revisemos las siguientes ecuaciones

7+5-6

1

12-6

2

6

9+7*8-36/5

1

9+56-36/5

2

9+56-7.2

3

65-7.2

4

57.8

Operadores de igualdad

Algebraico

= igual = diferente de

En Java

== igual != diferente de

Ejemplo

x==y x!=y

igual diferente de

Operadores de relación

< menor que

> mayor que

<= menor o igual que

>= mayor o igual que

Variable

Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Constante

Es un dato numérico o alfanumérico que no cambia durante la ejecución del programa. Ejemplo: pi = 3.1416

Ejemplo:

area = pi * radio ^ 2 Las variables son : el radio, el area y la constate es pi

Clasificación de las variables

Por su contenido Variables Numéricas: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Ejemplo: iva = 0.15 pi = 3.1416 costo = 2500

Variables Lógicas: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

Variables Alfanuméricas: Esta formada por caracteres alfanuméricos (letras, números y caracteres especiales). Ejemplo: letra = 'a' apellido = 'lopez' direccion = 'Av. Libertad #190'

Por su uso Variables de Trabajo: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Ejemplo: Suma = a + b /c

Contadores: Se utilizan para llevar el control del numero de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

Acumuladores: Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

Por ejemplo:

q= a + (b + 3) / c

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas. Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en: Aritméticas Relacionales Lógicas

2.1 INTRODUCCIÓN AL ENTORNO DE PROGRAMACIÓN.

¿Qué es un programa?

Un programa es un conjunto de instrucciones u ordenes basadas en un lenguaje de programación que una computadora interpreta para resolver un problema o una función especifica.

Concepto de programación.

La programación es el camino que nos lleva a producir un programa informático, el cual discurre por una serie de reglas y principios, que constituyen una completa disciplina por sí misma.

Pese a que los lenguajes de programación, las filosofías que les dan lugar, y las técnicas empleadas con cada uno de ellos en la resolución de los problemas, son diferentes, se considera la disciplina de la programación como algo unificado, pues todos los lenguajes parten de una serie de planteamientos comunes, aunque sus técnicas de creación de programas pueden acabar siendo bastante diferentes.

etbeans es un entorno de desarrollo gratuito y de código abierto que en el momento de escribir este artículo está en su versión 7.4. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos

móviles. Da soporte a las siguientes tecnologías, entre otras: **Java**, **PHP**, **Groovy**, **C/C++**, **HTML5**,... Además puede instalarse en varios sistemas operativos: Windows, Linux, Mac OS,...

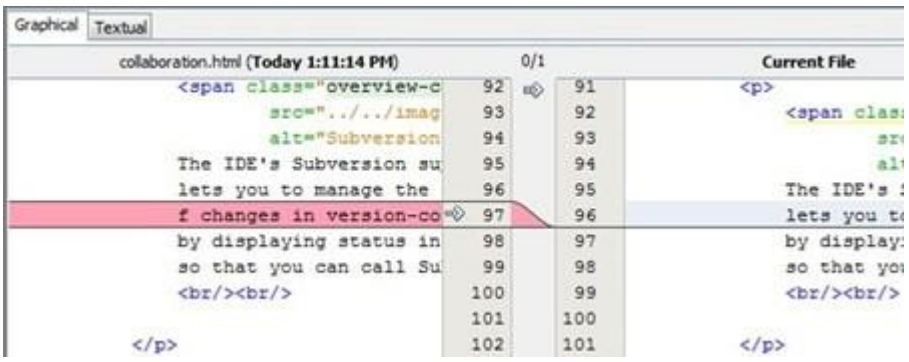
2.2 CARACTERÍSTICAS PRINCIPALES

Suele dar soporte a casi todas las novedades en el lenguaje **Java**. Cualquier preview del lenguaje es rápidamente soportada por Netbeans.

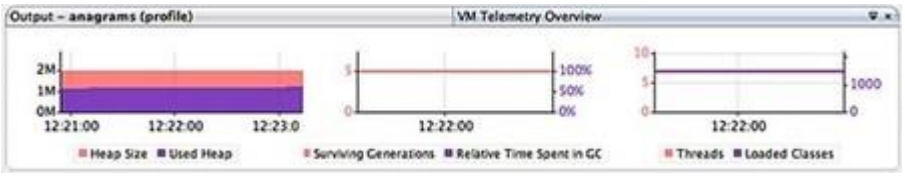
Asistentes para la creación y configuración de distintos proyectos, incluida la elección de algunos frameworks.

Buen **editor de código**, **multilenguaje**, con el habitual coloreado y sugerencias de código, acceso a clases pinchando en el código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, coding tips, herramientas de refactorización, y un largo etcétera. También hay tecnologías donde podemos usar el pulsar y arrastrar para incluir componentes en nuestro código.

Simplifica la **gestión de grandes proyectos** con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario. Una vez que nos metemos en una clase java, por poner un ejemplo, se nos mostrarán distintas ventanas con el código, su localización en el proyecto, una lista de los métodos y propiedades (ordenadas alfabéticamente), también hay una vista que nos presenta las jerarquías que tiene nuestra clase y otras muchas opciones. Por supuesto personalizable según el gusto de cada usuario.



Herramientas para **depurado de errores**: el debugger que incluye el IDE es bastante útil para encontrar dónde fallan las cosas. Podemos definir puntos de ruptura en la línea de código que nos interese, monitorizar en tiempo real los valores de propiedades y variables, se nos permite ir paso a paso, ejecutar un método de un tirón, o entrar dentro, en fin, las opciones típicas, pero que tan útiles son en el trabajo diario. Incluso podemos usar el debugger en caliente, conectándonos a él cuándo ya tenemos un proceso ejecutándose.



Optimización de código: por su parte el **Profiler** nos ayuda a optimizar nuestras aplicaciones e intentar hacer que se ejecuten más rápido y con el mínimo uso de memoria. Podemos igualmente configurarlo a nuestro gusto, aunque por defecto, nos ofrece opciones bastante útiles. Lo importante es que podemos ver el comportamiento de nuestra aplicación y obtener indicadores e información de cómo y cuantos recursos consume, cuantos objetos se crean, también podemos obtener capturas del estado del sistema en diferentes momentos (Snapshots) y compararlos entre sí.

Acceso a base de datos: desde el propio Netbeans podemos conectarnos a distintos sistemas gestores de bases de datos, como pueden ser Oracle, MySql y demás, y ver las tablas, realizar consultas y modificaciones, y todo ello integrado en el propio IDE.

Se integra con diversos **servidores de aplicaciones**, de tal manera que podemos gestionarlos desde el propio IDE: inicio, parada, arranque en modo debug, despliegues.

Entre otros podemos usar Apache Tomcat, GlassFish, JBoss, WebLogic, Sailfin, Sun Java System Application Server, ...

Es fácilmente **extensible** a través de **plugins**.

Historia

Nos tenemos que remontar a 1996 e ir hasta Praga, en la república Checa, donde nace un proyecto llamado **Xelfi** en el ámbito universitario (Facultad de Matemáticas y Física). Se pretendía escribir un **IDE** para **Java** que se pareciera al que tenía el lenguaje Delphi (de ahí el nombre de Xelfi). El código fue escrito en Java, y se convirtió en el primero escrito en dicho lenguaje con la publicación de su primera pre-release en el año 1997.

Los autores vieron suficiente interés por el proyecto, para formar una empresa y convertirlo en un proyecto comercial, con la inversión del empresario Roman Stanek. En 1999, en primavera, vería la luz Netbeans DeveloperX2, con soporte para Swing, que posteriormente se vería modificado para adaptarse al JDK 1.3, y hacerlo más modular.

Sun Microsystems se interesará por el proyecto en 1999, firmando un acuerdo en otoño de ese año.

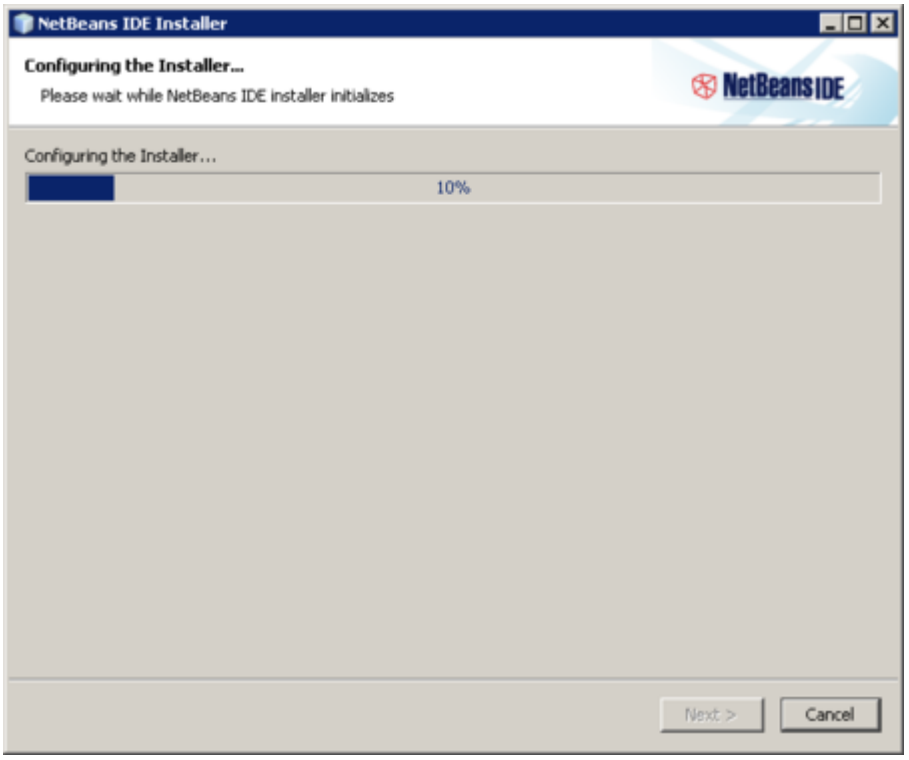
En junio del año 2000, la empresa Sun Micro Systems funda el proyecto Netbeans bajo los auspicios del software de código abierto.

El proceso de instalación

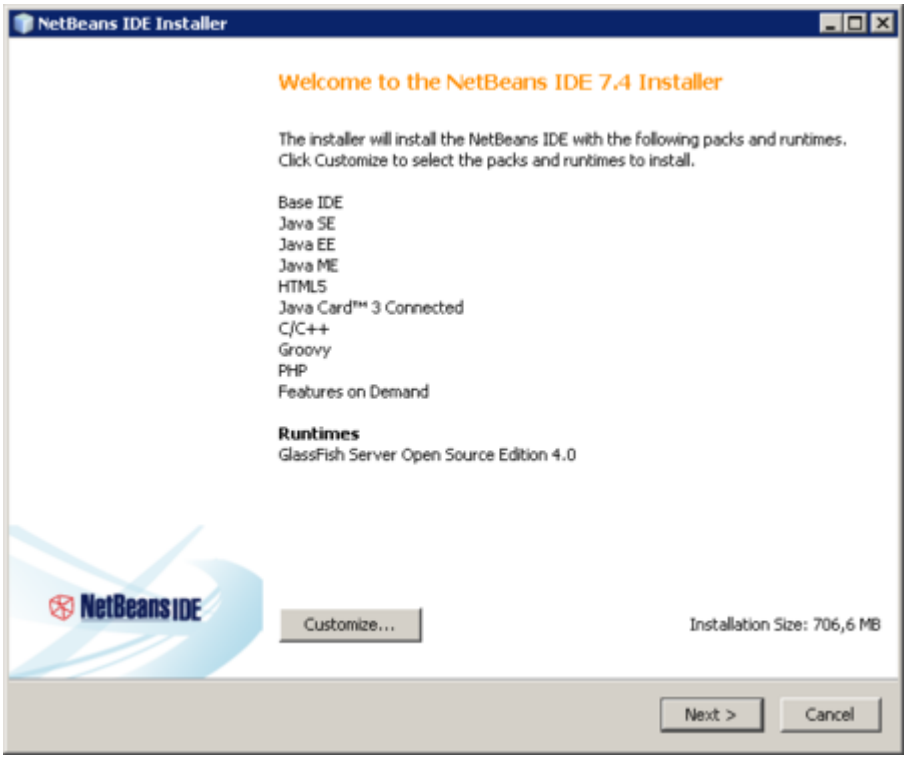
El proceso de instalación es muy muy sencillo. Vamos a ver como se instala bajo Windows, en otros sistemas operativos el proceso es similar. Como requisito previo, deberemos tener instalado un JDK de **Java**.

Lo primero que deberemos hacer es descargar el programa de instalación de la página de NetBeans y elegir el ejecutable, de las opciones que se muestran, que mejor se adapte a nuestras necesidades.

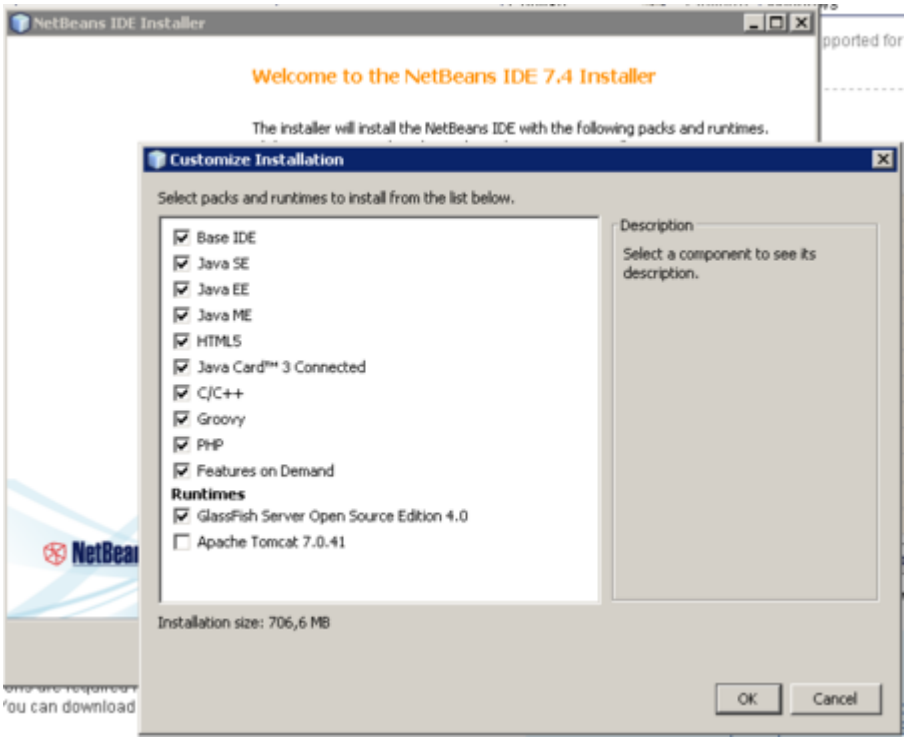
Una vez descargado ejecutamos el programa y dependiendo de la versión que hayamos elegido nos aparecerá la siguiente pantalla, para que podamos personalizar la instalación, e instalar el soporte para los lenguajes que vamos a instalar. Normalmente este paso se suele saltar, ya que, como hemos indicado, en la descarga hemos elegido el ejecutable que mejor se adapta a nuestras necesidades.



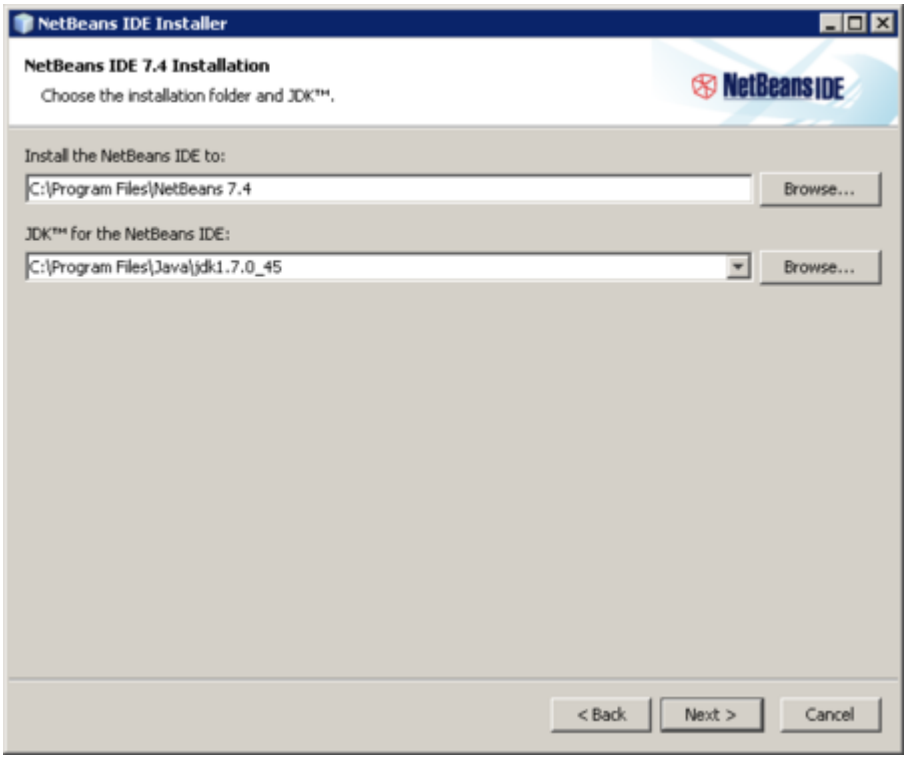
Si queremos poner o quitar algo a lo que se nos muestra en esta pantalla informativa, no hay más que pulsar en el botón "*customize*" (personalizar), y realizar los cambios oportunos. Una vez que las opciones estén como deseemos pulsaremos sobre el botón "*Next*" (siguiente).



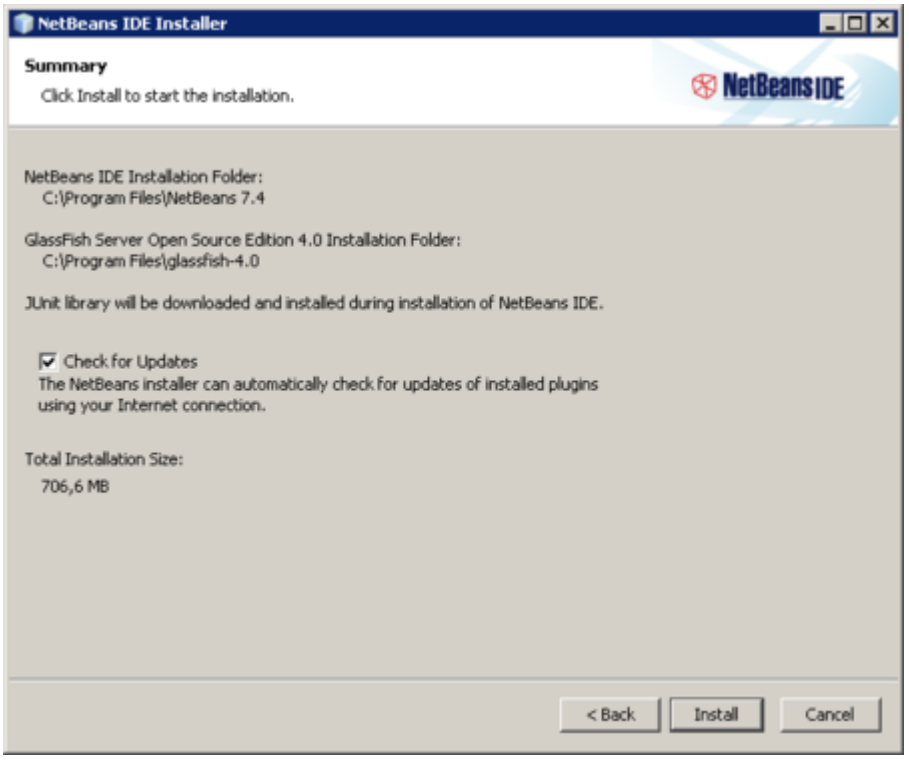
La pantalla de personalización en la sección de "*Runtimes*" nos permite especificar los servidores de aplicaciones que queremos instalar. Estos serán en los que podremos desplegar nuestras aplicaciones Web en desarrollo, componentes EJB, etc... Si no vamos a desarrollar en entornos Web, y sólo vamos a desarrollar para escritorio, por ejemplo, no sería necesario instalar ninguno. Para la mayoría de los programadores, convendrá instalar uno. Yo suelo usar Glassfish y tomcat. El propio IDE luego nos permitirá configurar estos, tener levantado uno y otro no, asignar uno u otro a cada proyecto, "*autodeploys*", etc... Una vez elegidas las opciones pulsamos sobre el botón OK.



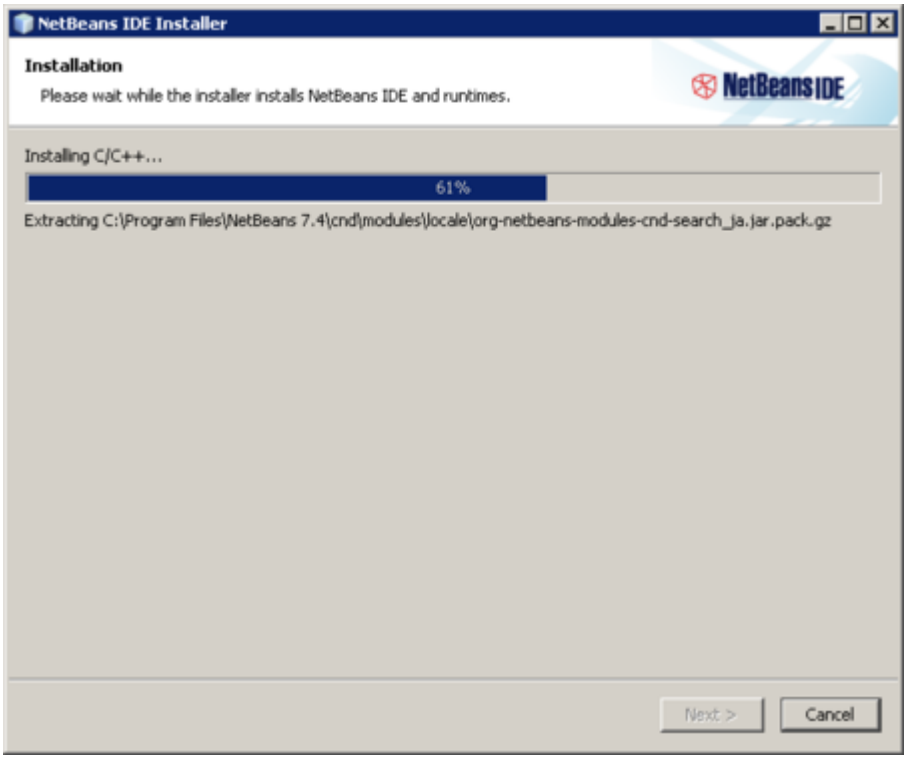
La siguiente pantalla nos pide los directorios de instalación de Netbeans, y deberemos indicarle dónde tenemos instalado el JDK de Java. Normalmente este último lo detectará automáticamente, pero si tenemos varias versiones, podemos elegir con cual de ellas queremos que trabaje Netbeans (esto luego se podrá cambiar para cada proyecto, pero al menos deberemos tener un JDK instalado). Cuando todo esté correcto, de nuevo, pulsamos el botón "*Next*" (siguiente).



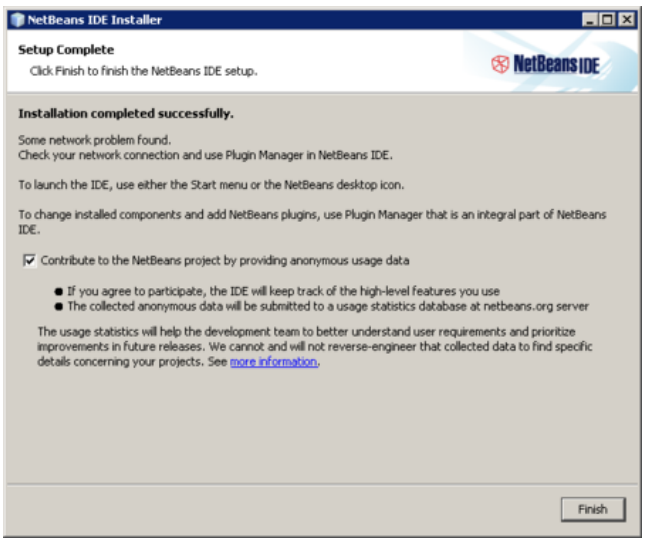
En este punto habremos configurado la instalación, y estamos listos para comenzarla de verdad. Simplemente deberemos especificar si queremos que chequee si hay actualizaciones o no (en cuyo caso se nos preguntará si queremos instalarlas), y pulsar el botón "*Install*" (instalar).



El proceso comenzará y nos mostrará esta pantalla de progreso, a la vez que nos va informando de lo que se va instalando. No tendremos que esperar mucho para que finalice completamente, tras lo cual pasaremos a la última pantalla.



Cuando nos sale esta pantalla habremos finalizado. Podremos elegir si queremos contribuir con NetBeans proporcionándole información anónima del uso, o no. Una vez hecho esto pulsaremos sobre el botón "*Finish*" (terminar).



2.3 ESTRUCTURA BÁSICA DE UN PROGRAMA.

Un programa informático (programa) es una **secuencia de acciones** (instrucciones) **que manipulan un conjunto de objetos** (datos).

Existen **dos** partes o **bloques** que componen un programa:

1. Bloque de **declaraciones**: en este se detallan todos los objetos que utiliza el programa (constantes, variables, archivos, etc.).
2. Bloque de **instrucciones**: conjunto de acciones u operaciones que se han de llevar a cabo para conseguir los resultados esperados.

El bloque de instrucciones está compuesto a su vez por tres partes, aunque en ocasiones no están perfectamente delimitadas, y aparecerán entremezcladas en la secuencia del programa, podemos localizarlas según su función. Estas son:

1. **Entrada** de datos: instrucciones que almacenan en la memoria interna datos procedentes de un dispositivo externo.
2. **Proceso** o algoritmo: instrucciones que modifican los objetos de entrada y, en ocasiones, creando otros nuevos.
3. **Salida** de resultados: conjunto de instrucciones que toman los datos finales de la memoria interna y los envían a los dispositivos externos.

Partes del bloque de instrucciones		
Entrada	**--> Algoritmo -->**	**Salida**
Inicio de programa: datos	Proceso de programa: cálculos	Fin de programa: resultados

En la siguiente tabla detallamos la estructura básica de un programa informático:

Estructura de un programa informático

Cabecera	A modo de comentarios se suele especificar: • Nombre del programa • Datos de entrada • Datos de salida
Funciones	Definición de funciones propias creadas por el programador para usarlas en varias ocasiones
Declaraciones	Definiciones y tipos de: • variables • constantes • nuevos tipos de datos
Asignaciones	Valores iniciales de los identificadores declarados previamente
Entradas	Instrucciones para almacenar en memoria los valores de algunos identificadores
Control	Instrucciones de control de flujo del programa. Pueden ser: • Alternativas • Repetitivas
Salidas	Instrucciones para devolver los resultados obtenidos

2.4 PALABRAS RESERVADAS.

En el lenguaje de programación Java, una palabra clave es cualquiera de las 57 **palabras reservadas** que tienen un significado predefinido en el lenguaje; debido a esto, los programadores no pueden usar palabras clave como nombres para variables, métodos, clases o como cualquier otro identificador. De estas 57 palabras clave, 55 están en uso y 2 no están en uso. Debido a sus funciones especiales en el lenguaje, la mayoría de los entornos de desarrollo integrados para Java utilizan el resaltado de sintaxis para mostrar palabras clave en un color diferente para una fácil identificación.

En este capítulo tenéis un listado de las **palabras reservadas** de Java orientado a objetos. Las **palabras reservadas** son identificadores, pero como su nombre indica, estas palabras están reservadas, y no se pueden usar como identificadores de usuario.

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

abstract

Se utilizan para implementar una abstracción en Java. Un método sin definición debe declararse como abstracto y la clase que lo contiene debe declararse como abstracto. Las clases abstractas no pueden ser instanciadas. Los métodos abstractos deben ser implementados en las subclases. La palabra clave abstract no se puede utilizar con variables o constructores. Tenga en cuenta que no se requiere que una clase abstracta tenga un método abstracto en absoluto.

Assert

Assert describe un predicado (una declaración de verdadero / falso) colocado en un programa Java para indicar que el desarrollador piensa que el predicado siempre es verdadero en ese lugar. Si una aserción se evalúa como falsa en tiempo de ejecución, se produce un error de aserción, que normalmente hace que la ejecución se anule. Opcionalmente habilitar por el método ClassLoader.

boolean

Define una variable booleana para los valores "true" o "false" solamente. Por defecto, el valor del tipo primitivo booleano es falso. Esta palabra clave también se usa para declarar que un método devuelve un valor del tipo primitivo booleano

break

Se utiliza para finalizar la ejecución en el cuerpo del bucle actual.

byte

La palabra clave byte se utiliza para declarar un campo que puede contener un entero de complemento de dos con signo de 8 bits. Esta palabra clave también se utiliza para declarar que un método devuelve un valor del byte tipo primitivo.

case

Una declaración en el bloque de conmutación se puede etiquetar con una o más etiquetas de case. La instrucción switch evalúa su expresión, luego ejecuta todas las declaraciones que siguen la etiqueta de caso correspondiente.

catch

Se usa junto con un bloque de prueba y un bloque finalmente opcional. Las declaraciones en el bloque catch especifican qué hacer si el bloque try lanza un tipo específico de excepción.

char

Define una variable de carácter capaz de contener cualquier carácter del conjunto de caracteres del archivo fuente Java.

class

Un tipo que define la implementación de un tipo particular de objeto. Una definición de clase define los campos de instancia y clase, los métodos y las clases internas, además de especificar las interfaces que implementa la clase y la superclase inmediata de la clase. Si la superclase no se especifica explícitamente, la superclase es implícitamente Object. La palabra clave de clase también se puede utilizar en el formulario Class.class para obtener un objeto Class sin necesidad de una instancia de esa clase. Por ejemplo, String.class se puede usar en lugar de hacer una nueva String (). GetClass ().

continue

Se utiliza para reanudar la ejecución del programa al final del cuerpo del bucle actual. Si le sigue una etiqueta, continúe con la ejecución de reanudación al final del cuerpo del bucle etiquetado adjunto.

default

La palabra clave default se puede usar opcionalmente en una instrucción de cambio para etiquetar un bloque de instrucciones que se ejecutarán si ningún caso coincide con el valor especificado; ver interruptor. Alternativamente, la palabra clave predeterminada también se puede utilizar para declarar valores predeterminados en una anotación de Java. Desde Java 8 en adelante, la palabra clave predeterminada se puede usar para permitir que una interfaz proporcione una implementación de un método.

do

La palabra clave do se usa junto con while para crear un bucle do-while, que ejecuta un bloque de sentencias asociadas con el bucle y luego prueba una expresión booleana asociada

con while. Si la expresión se evalúa como verdadera, el bloque se ejecuta de nuevo; esto continúa hasta que la expresión se evalúa como falsa.

double

La palabra clave double se usa para declarar una variable que puede contener un número de punto flotante IEEE 754 de doble precisión de 64 bits. Esta palabra clave también se usa para declarar un método que devuelve un valor del tipo primitivo double.

else

La palabra clave else se usa junto con if para crear una sentencia if-else, que pruebe una expresión booleana; si la expresión se evalúa como verdadera, se evalúa el bloque de instrucciones asociadas con el if; si se evalúa como falso, se evalúa el bloque de declaraciones asociadas con else.

enum (agregada en J2SE 5.0)

Una palabra clave de Java utilizada para declarar un tipo enumerado. Las enumeraciones amplían la clase base Enum.

exports

Se utiliza en java modular para exportar un paquete con un módulo. Esta palabra clave sólo está disponible en Java 9 y versiones posteriores.

extends

Se utiliza en una declaración de clase para especificar la superclase; utilizado en una declaración de interfaz para especificar una o más superinterfaces. La clase X extiende la clase Y para agregar funcionalidad, ya sea agregando campos o métodos a la clase Y, o reemplazando los métodos de la clase Y. Una interfaz Z extiende una o más interfaces al agregar métodos. Se dice que la clase X es una subclase de la clase Y; Se dice que la interfaz Z es una subinterfaz de las interfaces que se extiende. También se usa para especificar un límite superior en un parámetro de tipo en Genéricos.

final

Defina una entidad una vez que no se puede cambiar ni derivar de más adelante. Más específicamente: una clase final no puede ser subclasificada, un método final no puede ser anulado, y una variable final puede aparecer como máximo una vez como una expresión de la mano izquierda en un comando ejecutado. Todos los métodos en una clase final son implícitamente finales.

finally

Se utiliza para definir un bloque de instrucciones para un bloque definido previamente por la palabra clave try. El bloque finally se ejecuta después de que la ejecución sale del bloque try y de cualquier cláusula catch asociada sin importar si se lanzó o capturó una excepción, o si se ejecutó el método de izquierda en medio de los bloques try o catch usando la palabra clave return.

float

La palabra clave flotante se usa para declarar una variable que puede contener un número punto flotante IEEE 754 de precisión simple de 32 bits. Esta palabra clave también se usa para declarar que un método devuelve un valor del tipo primitivo float.

for

La palabra clave for se utiliza para crear un bucle for, que especifica una inicialización variable, una expresión booleana y un incremento. La inicialización de la variable se realiza primero y luego se evalúa la expresión booleana. Si la expresión se evalúa como verdadera, el bloque de instrucciones asociado con el bucle se ejecuta, y luego se realiza el incremento. La expresión booleana se evalúa de nuevo; esto continúa hasta que la expresión se evalúa como falsa. A partir de J2SE 5.0, la palabra clave for también se puede usar para crear el llamado "bucle mejorado para", que especifica una matriz o un objeto iterable; Cada

iteración del bucle ejecuta el bloque asociado de declaraciones utilizando un elemento diferente en la matriz o Iterable.

if

La palabra clave if se usa para crear una sentencia if, que prueba una expresión booleana; si la expresión se evalúa como verdadera, se ejecuta el bloque de instrucciones asociadas con la instrucción if. Esta palabra clave también se puede utilizar para crear una sentencia if-else; ver otra cosa

implements

Incluido en una declaración de clase para especificar una o más interfaces implementadas por la clase actual. Una clase hereda los tipos y métodos abstractos declarados por las interfaces.

import

Se usa al comienzo de un archivo fuente para especificar clases o paquetes completos de Java para consultarlos más adelante sin incluir sus nombres de paquete en la referencia. Desde J2SE 5.0, las declaraciones de importación pueden importar miembros estáticos de una clase.

instanceof

Un operador binario que toma una referencia de objeto como su primer operando y una clase o interfaz como su segundo operando y produce un resultado booleano. El instanceofoperator evalúa como verdadero si y solo si el tipo de tiempo de ejecución del objeto es compatible con la clase o la interfaz.

int

La palabra clave int se utiliza para declarar una variable que puede contener un entero de complemento a dos con signo de 32 bits. Esta palabra clave también se usa para declarar que un método devuelve un valor del tipo primitivo int.

interface

Se utiliza para declarar un tipo especial de clase que solo contiene métodos abstractos o predeterminados, campos constantes (final estático) e interfaces estáticas. Más tarde puede implementarse por clases que declaran la interfaz con la palabra clave implementa. Como la herencia múltiple no está permitida en Java, las interfaces se utilizan para evitarla. Una interfaz se puede definir dentro de otra interfaz.

long

La palabra clave larga se usa para declarar una variable que puede contener un entero de complemento a dos con signo de 64 bits. Esta palabra clave también se usa para declarar que un método devuelve un valor del tipo primitivo long.

module

La palabra clave del módulo se utiliza para declarar un módulo dentro de una aplicación Java. Esta palabra clave solo está disponible en Java 9 y versiones posteriores.

native

Se usa en declaraciones de métodos para especificar que el método no se implementa en el mismo archivo fuente de Java, sino en otro idioma

new

Se utiliza para crear una instancia de una clase o un objeto de matriz. El uso de palabras clave para este fin no es completamente necesario (como lo ejemplifica Scala), aunque sirve para dos propósitos: permite la existencia de un espacio de nombres diferente para los

métodos y nombres de clase, define estática y localmente que se crea un objeto nuevo, y de qué tipo de tiempo de ejecución es (podría decirse que introduce dependencia en el código).

package

El package Java es un grupo de clases e interfaces similares. Los paquetes se declaran con la palabra clave del paquete.

private

La palabra clave privada se usa en la declaración de un método, campo o clase interna; Los miembros privados solo pueden ser accedidos por otros miembros de su propia clase.

protected

La palabra clave protegida se usa en la declaración de un método, campo o clase interna; Los miembros protegidos solo pueden acceder a los miembros de su propia clase, las subclases de esa clase o las clases del mismo paquete.

public

La palabra clave pública se usa en la declaración de una clase, método o campo; Los miembros de cualquier clase pueden acceder a clases públicas, métodos y campos.

requires

Se utiliza para especificar las bibliotecas requeridas dentro de un módulo. Esta palabra clave solo está disponible en Java 9 y versiones posteriores.

return

Se utiliza para finalizar la ejecución de un método. Puede ir seguido de un valor requerido por la definición del método que se devuelve al llamante.

short

La palabra clave corta se usa para declarar un campo que puede contener un entero de complemento de dos con signo de 16 bits. Esta palabra clave también se usa para declarar que un método devuelve un valor del tipo primitivo short.

static

Se utiliza para declarar un campo, método o clase interna como un campo de clase. Las clases mantienen una copia de los campos de clase independientemente de cuántas instancias existen de esa clase. Staticalso se usa para definir un método como un método de clase. Los métodos de clase están vinculados a la clase en lugar de a una instancia específica, y solo pueden operar en campos de clase. (Las clases e interfaces declaradas como miembros estáticos de otra clase o interfaz son en realidad clases de nivel superior y no son clases internas).

strictfp (agregado en J2SE 1.2)

Una palabra clave de Java utilizada para restringir la precisión y el redondeo de los cálculos de punto flotante para garantizar la portabilidad.

super

Herencia básicamente utilizada para lograr la vinculación dinámica o el polimorfismo en tiempo de ejecución en java. Se utiliza para acceder a los miembros de una clase heredada por la clase en la que aparece. Permite que una subclase acceda a métodos anulados y miembros ocultos de su superclase. La palabra clave super también se usa para reenviar una llamada de un constructor a un constructor en la superclase. También se usa para especificar un límite inferior en un parámetro de tipo en Genéricos.

switch

La palabra clave switch se usa junto con case y default para crear una instrucción switch, que evalúa una variable, compara su valor con un caso específico y ejecuta el bloque de declaraciones asociadas con ese caso. Si no se compara el valor, se ejecuta el bloque opcional etiquetado de forma predeterminada, si se incluye.

synchronized

Se utiliza en la declaración de un método o bloque de código para adquirir el bloqueo mutex para un objeto mientras el hilo actual ejecuta el código. Para los métodos estáticos, el objeto bloqueado es la clase de la clase. Garantiza que, como máximo, un subproceso a la vez que opera en el mismo objeto ejecuta ese código. El bloqueo mutex se libera automáticamente cuando la ejecución sale del código sincronizado. Los campos, clases e interfaces no pueden ser declarados como sincronizados.

this

Se utiliza para representar una instancia de la clase en la que aparece. Esto se puede usar para acceder a los miembros de la clase y como referencia a la instancia actual. La palabra clave también se usa para reenviar una llamada de un constructor en una clase a otro constructor en la misma clase.

throw

Hace que se lance la instancia de excepción declarada. Esto hace que la ejecución continúe con el primer controlador de excepciones de cierre declarado por la palabra clave catch para manejar un tipo de excepción compatible de asignación. Si no se encuentra dicho controlador de excepciones en el método actual, entonces el método vuelve y el proceso se repite en el método de llamada. Si no se encuentra un controlador de excepciones en ninguna llamada de método en la pila, entonces la excepción se pasa al controlador de excepciones no capturado del subproceso.

throws

Se utiliza en las declaraciones de métodos para especificar qué excepciones no se manejan dentro del método, sino que se pasan al siguiente nivel superior del programa. Todas las excepciones no detectadas en un método que no sean instancias de RuntimeException deben declararse utilizando la palabra clave throws.

transient

Declara que un campo de instancia no es parte de la forma serializada predeterminada de un objeto. Cuando un objeto se serializa, solo los valores de sus campos de instancia no transitorios se incluyen en la representación serial predeterminada. Cuando un objeto se

deserializa, los campos transitorios se inicializan solo a su valor predeterminado. Si no se utiliza el formulario predeterminado, por ejemplo, cuando se declara una tabla serialPersistentFields en la jerarquía de clases, se ignoran todas las palabras clave transitorias.

try

Define un bloque de sentencias que tienen manejo de excepciones. Si se lanza una excepción dentro del bloque try, un bloque catch opcional puede manejar los tipos de excepción declarados. Además, se puede declarar un bloque finalmente opcional que se ejecutará cuando la ejecución salga de las cláusulas try y catch, independientemente de si se lanza una excepción o no. Un bloque try debe tener al menos una cláusula catch o un bloque finally.

void

La palabra clave void se usa para declarar que un método no devuelve ningún valor.

volatile

Se utiliza en declaraciones de campo para especificar que la variable se modifica de forma asíncrona mediante subprocesos que se ejecutan simultáneamente. Los métodos, las clases y las interfaces no se pueden declarar volátiles, ni las variables o los parámetros locales.

while

La palabra clave while se usa para crear un bucle while, que prueba una expresión booleana y ejecuta el bloque de sentencias asociadas con el bucle si la expresión se evalúa como verdadera; esto continúa hasta que la expresión se evalúa como falsa. Esta palabra clave también se puede utilizar para crear un bucle do-while; ver hacer.

Palabras reservadas para valores literales:

true, null, false

Un valor literal booleano. No son considerados palabras clave de Java, sino literales. Ahora bien, tampoco se pueden utilizar como indentificadores.

Identificadores especiales:

var

Un identificador especial que no se puede usar como nombre de tipo (desde Java 10).No usado

const

Aunque está reservado como palabra clave en Java, const no se usa y no tiene función. Para definir constantes en Java, vea la palabra clave final.

goto

Aunque reservado como una palabra clave en Java, goto no se usa y no tiene ninguna función. Los paradigmas de programación son una forma de clasificar los lenguajes de programación según sus características. Las lenguas se pueden clasificar en múltiples paradigmas. Algunos paradigmas se ocupan principalmente de las implicaciones para el modelo de ejecución del lenguaje, como permitir efectos secundarios o si la secuencia de operaciones está definida por el modelo de ejecución. Otros paradigmas se refieren principalmente a la forma en que se organiza el código, como agrupar un código en unidades junto con el estado que es modificado por el código. Sin embargo, otros están relacionados principalmente con el estilo de la sintaxis y la gramática.

2.5 VARIABLES

VARIABLES

Es un nombre que se refiere a un objeto que reside en la memoria. El objeto puede ser de alguno de los tipos vistos (número o cadena de texto), o alguno de los otros tipos existentes en Python.

Cada variable debe tener un nombre único llamado identificador. Eso es muy de ayuda pensar las variables como contenedores que contienen data el cual puede ser cambiado después a través de técnicas de programación.

Alcance de las variables

Las variables en Python son locales por defecto. Esto quiere decir que las variables definidas y utilizadas en el bloque de código de una *función*, sólo tienen existencia dentro de la misma, y no interfieren con otras variables del resto del código.

A su vez, las variables existentes fuera de una *función*, no son visibles dentro de la misma.

En caso de que sea conveniente o necesario, una variable local puede convertirse en una variable global declarándola explícitamente como tal con la sentencia *global*.

Ejemplos de variables

A continuación, se presentan algunos ejemplos del uso de *variables*:

Ejemplo de asignar valor a variable

A continuación, se creará un par de variables a modo de ejemplo. Una de tipo *cadenas de caracteres* y una de tipo *entero*:

```
>>> c = "Hola Mundo" # cadenas de caracteres
>>> type(c) # comprobar tipo de dato
<type 'str'>
>>> e = 23 # número entero
>>> type(e) # comprobar tipo de dato
<type 'int'>
```

Como puede ver en Python, a diferencia de muchos otros lenguajes, no se declara el tipo de la variable al crearla. En *Java*, por ejemplo, definir una variable seria así:

```
String c = "Hola Mundo";

int e = 23;
```

También nos ha servido el pequeño ejemplo para presentar los comentarios en linea en Python: cadenas de caracteres que comienzan con el carácter # y que Python ignora totalmente. Hay más tipos de *comentarios*, de los cuales se tratarán más adelante.

Ejemplo de cambiar valor a variable

A continuación, se cambiará el valor para una variable de tipo *cadenas de caracteres* a modo de ejemplo:

```
>>> c = "Hola Plone" # cadenas de caracteres

>>> c

'Hola Plone'
```

Ejemplo de asignar múltiples valores a a múltiples variables

A continuación, se creará múltiples variables (*entero*, *coma flotante*, *cadenas de caracteres*) asignando múltiples valores:

```
>>> a, b, c = 5, 3.2, "Hola"

>>> print a
```

```
5

>>> print b

3.2

>>> print c

'Hola'
```

Si usted quiere asignar el mismo valor a múltiples variables al mismo tiempo, usted puede hacer lo siguiente:

```
>>> x = y = z = True

>>> print x

True

>>> print y

True

>>> print z

True
```

El segundo programa asigna el mismo valor booleano a todas las tres variables x, y, z.

2.6 CONSTANTES

Una constante es un tipo de variable la cual no puede ser cambiada. Eso es muy de ayuda pensar las constantes como contenedores que contienen información el cual no puede ser cambiado después.

En Python, las constantes son usualmente declaradas y asignadas en un módulo. Aquí, el módulo significa un nuevo archivo que contiene variables, funciones, etc; el cual es importada en el archivo principal. Dentro del módulo, las constantes son escritas en letras MAYÚSCULAS y separadas las palabras con el carácter *underscore* _.

Constantes integradas

Un pequeño número de constantes vive en el espacio de nombres incorporado. Son las siguientes:

None

Más información consulte sobre *None*.

NotImplemented

Más información consulte sobre *NotImplemented*.

Ellipsis

Más información consulte sobre *Ellipsis*.

False

El valor falso del tipo *booleano*.

True

El valor verdadero del tipo *booleano*.

__debug__

Esta constante su valor es True si Python no se inició con una opción -O. Véase también la sentencia *assert*.

Nota

Los nombres *None* y __debug__ no se pueden reasignar (asignaciones a ellos, incluso como un nombre de atributo, causa una excepción *SyntaxError*), por lo que pueden considerarse constantes "verdaderas".

Ejemplo de constantes

A continuación, se presentan algunos ejemplos del uso de *constantes*:

Ejemplo de constantes desde un módulo externo

Crear un archivo llamado constantes.py con el siguiente contenido:

```
IP_DB_SERVER = "127.0.0.1"

PORT_DB_SERVER = 3307

USER_DB_SERVER = "root"

PASSWORD_DB_SERVER = "123456"

DB_NAME = "nomina"
```

Crear un archivo llamado main.py con el siguiente contenido:

```
import constantes



print "scp -v -P {0} {1}@{2}:/{3}/{4}/{4}.sql /srv/backup".format(

    str(constantes.PORT_DB_SERVER), constantes.USER_DB_SERVER,

    constantes.IP_DB_SERVER, constantes.USER_DB_SERVER,

    constantes.DB_NAME)
```

Luego ejecuta el programa de la siguiente forma:

```
python main.py
```

Cuando usted ejecuta el programa, la salida será:

```
scp -v -P 3307 root@127.0.0.1:/root/webapp/db.sql /srv/backup
```

En el programa anterior, existe un archivo de módulo constantes.py. Entonces en este se asignan los valores de constantes IP_DB_SERVER, PORT_DB_SERVER, USER_DB_SERVER, PASSWORD_DB_ SERVER y DB_NAME. Ademas, existe el archivo de módulo main.py el cual importa el módulo constante. Finalmente, se imprime una linea de conexión del comando scp de Linux usando la función integrada en la librería estándar Python llamada *format()*.

2.7 TIPOS DE DATOS

Un **tipo de datos** es la propiedad de un valor que determina su dominio (qué valores puede tomar), qué operaciones se le pueden aplicar y cómo es representado internamente por el computador.

Todos los valores que aparecen en un programa tienen un tipo.

A continuación, revisaremos los tipos de datos elementales de Python. Además de éstos, existen muchos otros, y más adelante aprenderemos a crear nuestros propios tipos de datos.

Números enteros

El tipo **int** (del inglés *integer*, que significa «entero») permite representar números enteros.

Los valores que puede tomar un `int` son todos los números enteros: ... `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...

Los números enteros literales se escriben con un signo opcional seguido por una secuencia de dígitos:

```
1570
+4591
-12
```

Números reales

El tipo **float** permite representar números reales.

El nombre float viene del término punto flotante, que es la manera en que el computador representa internamente los números reales.

Hay que tener mucho cuidado, porque los números reales no se pueden representar de manera exacta en un computador. Por ejemplo, el número decimal 0.7 es representado internamente por el computador mediante la aproximación 0.69999999999999996. Todas las operaciones entre valores float son aproximaciones. Esto puede conducir a resultados algo sorpresivos:

```
>>> 1/7 + 1/7 + 1/7 + 1/7 + 1/7 + 1/7 + 1/7
0.9999999999999998
```

Los números reales literales se escriben separando la parte entera de la decimal con un punto. Las partes entera y decimal pueden ser omitidas si alguna de ellas es cero:

```
>>> 881.9843000
881.9843
>>> -3.14159
-3.14159
>>> 1024.
1024.0
```

```
>>> .22
0.22
```

Otra representación es la notación científica, en la que se escribe un factor y una potencia de diez separados por una letra e. Por ejemplo:

```
>>> -2.45E4

-24500.0
>>> 7e-2
0.07
>>> 6.02e23
6.02e+23
>>> 9.1094E-31
9.1094e-31
```

Los dos últimos valores del ejemplo son iguales, respectivamente, a 6.02×10236.02×1023 (la constante de Avogadro) y 9.1094×10−319.1094×10−31 (la masa del electrón).

Números complejos

El tipo **complex** permite representar números complejos.

Los números complejos tienen una parte real y una imaginaria. La parte imaginaria es denotada agregando una j inmediatamente después de su valor:

```
3 + 9j
-1.4 + 2.7j
```

Valores lógicos

Los valores lógicos `True` y `False` (verdadero y falso) son de tipo **bool**, que representa valores lógicos.

El nombre `bool` viene del matemático George Boole, quien creó un sistema algebraico para la lógica binaria. Por lo mismo, a `True` y `False` también se les llama **valores booleanos**. El nombre no es muy intuitivo, pero es el que se usa en informática, así que hay que conocerlo.

Texto

A los valores que representan texto se les llama **strings**, y tienen el tipo **str**.

Los strings literales pueden ser representados con texto entre comillas simples o comillas dobles:

```
"ejemplo 1"
'ejemplo 2'
```

La ventaja de tener dos tipos de comillas es que se puede usar uno de ellos cuando el otro aparece como parte del texto:

```
"Let's go!"
'Ella dijo "hola"'
```

Es importante entender que los strings no son lo mismo que los valores que en él pueden estar representados:

```
>>> 5 == '5'
False
>>> True == 'True'
False
```

Los strings que difieren en mayúsculas y minúsculas, o en espacios también son distintos:

```
>>> 'mesa' == 'Mesa'
False
>>> ' mesa' == 'mesa '
False
```

Nulo

Existe un valor llamado **None** (en inglés, «ninguno») que es utilizado para representar casos en que ningún valor es válido, o para indicar que una variable todavía no tiene un valor que tenga sentido.

El valor `None` tiene su propio tipo, llamado `NoneType`, que es diferente al de todos los demás valores.

2.7.1 SIMPLES.

El tipo de dato representa la clase de datos con el que vamos a trabajar. Tenemos los siguientes tipos de datos simples:

- Números enteros: Nos sirve para representar números enteros.
- Números reales: Nos sirve para representar números reales.
- Cadenas de caracteres: Nos permite trabaja con cadenas de caracteres.
- Valores lógicos: Nos permite trabajar con valores lógicos.

Al escribir pseudocódigo en un programa de ordenador, tenemos que representar cada tipo de datos en binario y tenemos que decidir la memoria que va a ocupar cada uno de los tipos de datos.

En este caso PseInt representa los valores enteros con 32 bit, por lo tanto en un tipo entero se pueden representar 2^{32} valores, es decir desde el -2147483648 hasta el 2147483647.

Por lo tanto si vamos a trabajar con números más grandes o más pequeños hay que utilizar un tipo de dato Real.

Literales

Los literales nos permiten representar valores. Los literales son de distintos tipos de datos.

- **Literales enteros**: Ejemplos: 3, 0, -1, …
- **Literales reales**: Se utiliza el . para separar la parte entera y decimal. Por ejemplo: 3.0, -1.4, 2.345,…
- **Literales cadenas**: Se utiliza las " para indicar una cadena de caracteres, por ejemplo: "Hola", "123", "", …
- **Literales lógicos**: sólo pueden tener dos valores: Verdadero y Falso.

2.7.2 COMPUESTOS (ABSTRACTOS).

Los datos compuestos son el tipo opuesto a los tipos de datos atómicos. Los datos compuestos se pueden romper en subcampos que tengan significado.

Un ejemplo sencillo es el número de su teléfono celular 56 2 99110101. Realmente, este número consta de varios campos, el código del país (56, Chile), el código del área (2, Santiago) y el número propiamente dicho, que corresponde a un celular porque empieza con 9.

En algunas ocasiones los datos compuestos se conocen también como datos o tipos agregados. Los tipos agregados son tipos de datos cuyos valores constan de colecciones de

elementos de datos. Un tipo agregado se compone de tipos de datos previamente definitivos.

Existen tres tipos agregados básicos:

1. Arreglos (arrays) y Matrices (tablas)
2. Registros
3. Secuencias de texto o cadenas.

2.8 DESPLIEGUE Y FORMATEO DE DATOS.

Los problemas de cadena de formato contituyen uno de los pocos ataques realmente nuevos que surgieron en años recientes.

Al igual que con muchos problemas de seguridad, la principal causa de los errores de cadena de formato es aceptar sin validar la entrada proporcionada por el usuario. En C/C++ es posible utilizar errores de cadena de formato para escribir en ubicciones de memoria arbitrarias, y el aspecto mpas peligroso es que esto llega a suceder sin manipular bloques de memoria adyacentes. Esta capacidad de diseminación permite a un atacante eludir protecciones de pila, e incluso modificar partes my pequeñas de memoria. El problema también llega a ocurrir cuando las cadenas de formato se leen a partir de una ubicación no confiable que controla el atacante.

Este último aspecto del problema tiende a ser más frecuente en sistemas UNIX y Linux. En sistemas Windows las tablas de cadena de aplicación suelen mantenerse dentro del progrma ejecutable o de las bibliotecas de vínculo dinámico (DLL, Dynamic Link Libraries) del recurso. Si un atacante reeescribe el ejecutable principal o de las DLL, tendrá la posibilidad de realizar ataques mucho más directos que con errores de cadena de formato.

Aunque no esté trabajando con C/C++, los ataques de cadena de formato quizá conduzcan a problemas importantes; el más obvio es engañar a los usuarios, pero bajo ciertas

circustancias es posible que un atacante lance ataques de creación de script de sitio cruzado o de inyección de SQL, los cuales también se utilizan para corregir o transformar datos.

Lenguajes Afectados

El lenguaje más afectado es C/C++. Un ataque exitoso quizá conduzca de manera inmediata a la ejecución de código arbitrario y a revelación de información. Por lo general otros lenguajes no permiten la ejecución de código arbitario, pero, como ya se observó, son proporcionados por entrada del usuario, pero podría serlo si las cadenas de formato se leen a partir de datos manipulados.

Explicación del problema de cadena de formato

El formateo de datos para despliegue o almacenamiento tal vez represente una tarea un poco difícil; por tanto, en muchos lenguajes de computadora se incluyen rutinas para reformatear datos con facilidad. En casi todos los lenguajes la información de formato se describe a través de un tipo de cadena, denominada cadena de formato. En realidad, la cadena de formato se define con el uso de lenguaje de procesamiento de datos limitado que está diseñado para facilitar la descripción de formatos de salida. Sin embargo, muchos desarrolladores cometen un sencillo error: utilizan datos de usuarios no confiables como cadena de formato; el resultado es que los atacantes pueden escribir cadenas en el lenguaje de procesamiento de datos para causar muchos problemas.

El diseño de C/C++ hace que esto sea especialmente peligroso: dificulta la detección de problemas de cadena de formato, y entre las cadenas de formato se incluyen algunos comandos muy peligrosos (en particular %n) que no existen en lenguajes de cadena de formato de algunos otros lenguajes.

En C/C++ puede declararse una función para que tome un número de variable de argumentos al especificar puntos suspensivos (…) como el último (o único) argumento. El problema es que la función a la que se llama no tiene manera de saber cuántos argumentos se están pasando. El conjunto más común de funciones que toman argumentos de longitud

variable es la familia de printf: printf, sprintf, snprintf, vprintf, etc. Las extensas funciones de carácter que realizan la misma función tienen el mismo problema.

Veamos un ejemplo:

```
#include <studio.h>
int main (int argc, char* argv[])
{
if (argc > 1)
printf(argv[1]);
return 0;
}
```

Algo muy sencillo. Ahora veamos lo que está mal. El programador está esperando que el usuario introduzca algo benigno como *Hola mundo*. Si lo intenta, obtendrá *Hola mundo*. Ahora cambiemos un poco la entrada: pruebe *%x %x*. En un sistema Windows XP con la línea de comandos predeterminada (cmd. exe) , ahora obtendrá lo siguiente:

```
E:\proyectos\programación_segura\format_bug>format_bug.exe "%x %x"
12ffc0 4011e5
```

Observe que si está ejecutando un sistema operativo diferente o utilizando un interprete de línea de comandos diferente, tal vez necesite realizar algunos cambios para alimentar esta cadena exacta en su programa, y es posible que los resultados sean diferentes. Para facilitar su uso podría colocar los argumentos en una línea de comandos de shell o en un archivo de procesamiento por lotes.

¿Qúe sucedió? La función de printf tomó una cadena de entrada que hizo que esperara a que se colocaran dos argumentos en la pila antes de llamar a la función. Los especificadores *%x* le permiten leer la pila, de cuatro en cuatro bytes, hasta donde se desee. No es difícil imaginar que si tuviera una función más compleja que almacenara un secreto en una variable de pila, el atacante tendría la posibilidad de leerlo. Aquí la salida es la

dirección de la ubicación de la pila *(0×12ffc0)*, seguida de la ubicación del código en que se devolverá la función *main()* . Como se imagina, ambas son piezas de información muy importantes que se filtran al atacante.

Tal vez ahora se esté preguntando cómo utiliza el atacante un error de cadena de formato para escribir en la memoria. Uno de los especificadores de formato menos utilizados es *%n*, que escribe el número de caracteres que deben haberse escrito hasta el momento en la dirección de la variable que proporcionó como argumento correspondiente. He aquí cómo debe utilizarse:

unsigned int bytes;
printf ("%s%n\n", argv[1], &bytes);
printf (su entrada fue de %d caracteres de largo\n, bytes");

La salida sería:
E:\proyectos\programacion_segura\format_buf>format_bug2.exe "alguna salida aleatoria"
Alguna entrada aleatoria
Su entrada fue 17 caracteres de largo

En una plataforma con enteros de cuatro bytes el especificador *%n* escribirá cuatro bytes a la vez, y *%hn* escribirá dos bytes. Ahora los atacantes sólo tienen que imaginar cómo obtener la dirección que desean en la posición apropiada de la pila y ajustar los especificadores de ancho de campo hasta que el número de bytes escrito sea el que desean.

C/C++

A diferencia de muchos otros errores que examinaremos, éste es basatante fácil de localizar como defecto de código. Es muy sencillo:

printf (entrada_usuario);
Es incorrecto, y
printf ("%s", entrada_usuario);
Es correcto .

Localización de problemas de cadena de formato

Cualquier aplicación que tome entrada del usuario y la pase a una función de formateo está en riesgo. Un ejemplo muy frecuente de este problema sucede junto con aplicaciones que registran la entrada de usuario. Además, algunas funciones tal vez implanten formateo de manera interna.

En C/C++ busque funciones de la familia printf. Entre los problemas que habrá de localizar se encuentran los siguientes:

printf (entrada_usuario) ;
fprintf (STDOUT, entrada_usuario);
Si se encuentra con una función con el siguiente aspecto:
fprintf (STDOUT, msg_format, arg1, arg2);

necesitará verificar dónde se almacena la cadena a la que hace referencia msg_formato y si está bien protegida.

Hay muchas otras llamadas de sistema y API que también son vulnerables, syslog es una de ellas. Cuando vea una definición de función que incluya … en la lista de argumentos, está viendo algo que posiblemente represente un problema.

Muchos escáneres de código fuente, aun los de léxico como RATS y Flawfinder, detectan esto. Incluso PSCAN se diseñó de manera específica con este propósito. Además, existen herramientas de respuesta que se integran en el proceso de compilación; por ejemplo FormatGuard de Crispin Cowan.

Técnicas de prueba para encontrar el problema de cadena de formato

Pases especificadores de formateo a la aplicación y vea si se regresan valores hexadecimales. Por ejemplo, si tiene una aplicación que espera un nombre de archivo y regresa un mensaje

de error que contiene la entrada cuando no se encuentra el archivo, entonces trate de proporcionarle nombres de archivo como *NoProbable%x%x.txt*. Si obtiene un mensaje de error parecido a las siguientes líneas "*NoProbable12fd234104587*", acaba de encontrar una vulnerabilidad de cadena de formato.

Ejemplos de Problemas de Cadena de Formato

Entradas en la lista Common Vulnerabilities and Exposures, CVE.

CVE-2000-0573

CVE-2000-0844

Métodos de prevención

El primer paso es nunca pasar entradas de usuario directamente a una función de formateo, además de asegurarse de hacerlo en cada nivel de manipulación de salida formateada. Como nota adicional, las funciones de formateo tienen una sobrecarga de trabajo importante. Si está interesado, busque *_output* en el archivo fuente, tal vez sea conveniente escribir:

fprintf (STDOUT, buf);

la anterior línea de código no es sólo peligrosa, sino que también consume muchos ciclos de CPU adicionales.

El segundo paso que habrá de darse consiste en asegurar que las cadenas de formato que utiliza su aplicación se lean desde lugares confiables, y que las rutas a las cadenas no sean controladas por el atacante.

Medidas defensivas adicionales

Verifique y limite la ubicación a valores válidos. No utilice la familia de funciones printf, si puede evitarlas. Por ejemplo, si está empleando C++, use operadores de flujo:

```
#include <iostream>
//…
std::cout <<entrada_usuario
//…
```

2.9 OPERADORES ARITMÉTICOS, LÓGICOS Y RELACIONALES.

Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).

Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.

Operador	Nombre	Ejemplo	Significado
^	Exponenciación	x^y	x elevado a y
*	Multiplicacion	x*y	x multiplicado por y
/	División	x/y	x dividido entre y
%	Modulo o resto	x%y	resto de la división x/y
+	Suma	x+y	x más y
–	Resta	x-y	x menos y

Prioridad de los Operadores Aritméticos

Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero. Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1. ^ Exponenciación
2. *, /, mod Multiplicación, división, modulo.
3. +, – Suma y resta.

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

RELACIONALES

Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa. Por ejemplo, 8>4 (ocho mayor que cuatro) es verdadera, se representa por el valor **true** del tipo básico **boolean**, en cambio, 8<4 (ocho menor que cuatro) es falsa, **false**. En la primera columna de la tabla, se dan los símbolos de los operadores relacionales, el la segunda, el nombre de dichos operadores, y a continuación su significado mediante un ejemplo.

Operador	Nombre	Ejemplo	Significado
<	Menor que	x<y	x menor que y
>	Mayor que	x>y	x mayor que y

==	Igual a	x==y	x igual a y
!=	Distinto a	x!=y	x distinto de y
<=	Menor o igual que	x<=y	x menor o igual a y
>=	Mayor o igual que	x>=y	x mayor o igual a y

LÓGICOS

Estos operadores se utilizan para establecer relaciones entre valores lógicos. Estos valores pueden ser resultado de una expresión relacional.

Operador	Nombre	Ejemplo	Significado
&&	AND	(x<y)&&(x<z)	'x menor que y' Y 'x menor que z'
||	OR	(x<y)||(x<z)	'x menor que y' O 'x menor que z'
!	NOT	!(x<y)	x NO es menor que y

Los operadores AND y OR combinan expresiones relacionales cuyo resultado viene dado por la última columna de sus tablas de verdad. Por ejemplo:

```
(a<b) && (b<c)
```

es verdadero (**true**), si ambas son verdaderas. Si alguna o ambas son falsas el resultado es falso (**false**). En cambio, la expresión

```
(a<b) ||(b<c)
```

es verdadera si una de las dos comparaciones lo es. Si ambas, son falsas, el resultado es falso.

La expresión " NO *a* es menor que *b*"

```
!(a<b)
```

es falsa si *(a<b)* es verdadero, y es verdadera si la comparación es falsa. Por tanto, el operador NOT actuando sobre *(a<b)* es equivalente a

```
(a>=b)
```

La expresión "NO *a* es igual a *b*"

```
!(a==b)
```

es verdadera si *a* es distinto de *b*, y es falsa si *a* es igual a *b*. Esta expresión es equivalente a

```
(a!=b)
```

2.10 CONTROL DE FLUJO.

En programación, tipo de estructura de control. Ejecuta cero o más veces un grupo de instrucciones (bucle). El número de repeticiones está determinado por un número dado, o hasta que deje de cumplirse o se cumpla una condición. Las estructuras de repetición más usuales en los lenguajes de programación suelen ser WHILE, REPEAT y FOR. Los otros dos tipos de estructuras de control son: estructura de secuencia, y estructura de decisión.

Las instrucciones se siguen en una secuencia fija que normalmente viene dada por el número de renglón. Es decir que las instrucciones se ejecutan de arriba hacia abajo. Las instrucciones se ejecutan dependiendo de la condición dada dentro del algoritmo.

instrucción 1;

instrucción 2;

instrucción 3;

...

instrucción n;

La estructura secuencial es la más sencilla de todas, simplemente indica al procesador que debe ejecutar de forma consecutiva una lista de acciones (que pueden ser, a su vez, otras estructuras de control); para construir una secuencia de acciones basta con escribir cada acción en una linea diferente. A continuación se muestra una composición secuencial de acciones en notación algorítmica y su equivalente FORTRAN.

1.-leer a

2.-leer b

3.-c=a + b

4.-escribir c

PARA MATLAB

```
a=input('a=')

b=input('b=')

c = a + b

fprintf('el valor de c es:%f\n'c)
```

Existe una forma alternativa de expresar una estructura secuencial escribiendo varias acciones en la misma línea pero utilizando el punto y coma, ;, como separador. Sin embargo, esta última notación es des aconsejable puesto que puede llegar a hacer el código bastante difícil de leer.

1.-leer a

2.-leer b

3.-c=a + b

4.-escribir c

PARA MATLAB utilizando punto y coma

```
a=input('a=');

b=input('b=');
```

c = a + b;

fprintf('el valor de c es:%f\n'c);

Por último, es necesario señalar un aspecto importante de la composición secuencial y es que no es conmutativa.

Punto de entrada-Acción 1-Acción 2-Acción n-...-Punto de salida

Control selectivo.

En programación, tipo de estructura de control. También llamada estructura secuencial. Orden de ejecución de instrucciones de forma secuencial, o sea, una instrucción después de la otra. Esta es la más importante y engloba a las otras dos tipas: estructura de selección, y estructura de repetición.

La estructura selectiva permite bifurcar el "flujo" del programa en función de una expresión lógica; disponemos de tres estructuras alternativas diferentes: alternativa simple, alternativa doble y alternativa múltiple.

Estructura selectiva simple

Esta estructura permite evaluar una expresión lógica y en función de dicha evaluación ejecutar una acción (o composición de acciones) o no ejecutarla; también se la suele denominar SI-ENTONCES. A continuación se muestra la notación algorítmica y FORTRAN para la estructura alternativa simple.

si expresión lógica entonces

acciones

fin_si

if (expresión lógica) then

acciones

end if

Estructura selectiva doble

La estructura alternativa doble es similar a la anterior con la salvedad de que en este tipo de estructura se indican acciones no sólo para la rama "verdadera" sino también para la "falsa"; es decir, en caso de la expresión lógica evaluada sea cierta se ejecutan una acción o grupo de acciones y en cado de que sea falsa se ejecuta un grupo diferente. La sintáxis en la notación algorítmica y en FORTRAN son las que se muestran a continuación:

si expresión lógica entonces

acciones

si no

acciones

fin_si

if (expresión lógica) then

acciones

else

acciones

end if

Estructura selectiva multiple

Esta estructura evalúa una expresión que pueda tomar n valores (enteros, caracteres y lógicos pero nunca reales) y ejecuta una acción o grupo de acciones diferente en función del valor tomado por la expresión selectora. La sintaxis de esta estructura es la siguiente:

segun expresión

caso valor1:

acción 1

caso valor2:

acción 2

...

caso valor N:

acción n

otro caso:

acción

fin según

select case (expresión)

```
  case (valor1)
    acción 1
  case (valor2)
    acción 2
  ...
  case (valorn)
    acción n
  case default
    acción
end select
```

En el siguiente ejemplo se proporciona como salida el número de días de un mes dado:

```
segun mes
  caso 1,3,5,7,8,10,12:
    escribir '31'
  caso 4,6,9,11:
    escribir '30'
  caso 2:
    escribir '28'
```

```
  otro caso:
    escribir 'Mes incorrecto'
fin según
select case (mes)
  case (1,3,5,7,8,10,12)
    print *, '31'
  case (4,6,9,11)
    print *, '30'
  case (2)
    print *, '28'
  case default
    print *, 'Mes incorrecto'
end select
```

Obsérvese que es posible que un caso conste de múltiples valores.

Las instrucciones selectivas representan instrucciones que pueden o no ejecutarse, según el cumplimiento de una condición.

```
SI condición ENTONCES
instrucciones
```

FIN SI.

La condición es una expresión booleana. *Instrucciones* es ejecutada sólo si la condición es verdadera.

Selectiva doble

La instrucción alternativa realiza una instrucción de dos posibles, según el cumplimiento de una condición.

SI condiciones ENTONCES

instrucción 1;

SI no ENTONCES

instrucción 2;

FIN SI.

La condición es una variable booleana o una función reducible a booleana (lógica, Verdadero/Falso). Si esta condición es cierta se ejecuta *Instrucciones*$_1$, si no es así, entonces se ejecuta *Instrucciones*$_2$.

Selectiva múltiple

También es común el uso de una selección múltiple que equivaldría a anidar varias funciones de selección.

SI condición 1 ENTONCES
instrucción 1;
SI no si condición 2 ENTONCES
instrucción 2;
SI no si condición 3 ENTONCES
instrucción 3;
...
SI no ENTONCES

instrucción n;
FIN SI.

En este caso hay una serie de condiciones que tienen que ser mutuamente excluyentes, si una de ellas se cumple las demás tienen que ser falsas necesariamente, hay un caso **si no** que será cierto cuando las demás condiciones sean falsas.

En esta estructura si *Condición*$_1$ es cierta, entonces se ejecuta sólo *Instrucciones*$_1$. En general, si *Condición*$_i$ es verdadera, entonces sólo se ejecuta *Instrucciones N*.

Control repetitivo.

En programación, tipo de estructura de control. También llamada estructura de decisión. En una estructura de selección/decisión, el algoritmo al ser ejecutado toma una decisión, ejecutar o no ciertas instrucciones si se cumplen o no ciertas condiciones. Las condiciones devuelven un valor, verdadero o falso, determinado así la secuencia a seguir. Por lo general los lenguajes de programación disponen de dos estructuras de este tipo: estructura de decisión simple (if), y estructura de decisión múltiple (CASE, SWITCH). Los otros dos tipos de estructuras de control son: estructura de secuencia, y estructura de repetición.

Bucle mientras

El bucle se repite mientras la condición sea cierta, si al llegar por primera vez al bucle mientras la condición es falsa, el cuerpo del bucle no se ejecuta ninguna vez.

MIENTRAS condición HACER

instrucción;

FIN MIENTRAS.

Bucle para

Una estructura de control muy común es el ciclo *para*, la cual se usa cuando se desea iterar un número conocido de veces, empleando como índice una variable que se incrementa (o decrementa) la cual se define como:

i < x

MIENTRAS i <= n HACER

instrucción;

i < i +z

FIN MIENTRAS.

La estructura repetitiva o iterativa permite, como su propio nombre indica, repetir una acción (o grupo deEstructura repetitiva acciones); dicha repetición puede llevarse a cabo un número prefijado de veces o depender de la evaluación de una expresión lógica. Existen dos tipos de estructuras repetitivas: desde-hasta y mientras.

Estructura desde-hasta

Esta estructura permite repetir la ejecución de una acción o de un grupo de acciones un número determinado de veces; la sintáxis es la siguiente:

desde indice_inicio hasta fin [con paso valor] hacer

  acción

fin desde

do indice=inicio, fin, paso

  acción

end do

El funcionamiento de la estructura es el siguiente:

• En primer lugar, se asigna a la variable indice el valor de inicio.

• El bucle se ejecuta mientras indice no alcance el valor de fin.

• En cada iteración el valor de indice es incrementado según el paso indicado y se ejecuta la acción o grupo

de acciones encerrados en el bucle.

• En caso de que no se indique ningún paso el que se empleará sera +1.

A continuación, se muestran algunos ejemplos:

desde n_1 hasta 10 hacer

  escribir n

```
fin desde

do n=1, 10

  print *, n

end do
```

El bucle anterior imprime por pantalla los números del 1 al 10.

```
desde n!10 hasta 1 hacer

  escribir n

fin desde

do n=10, 1

  fprintf *, n

end do
```

El bucle anterior no se ejecuta nunca puesto que no se puede alcanzar 1 comenzando en 10 y avanzando con paso +1; ¡atención! Un error frecuente es pensar que el bucle se ejecuta de forma infinita.

```
desde n!10 hasta 1 con paso –2 hacer

  escribir n

fin desde

do n=10, 1, -2

 fprintf *, n
```

end do

Este bucle escribe los números pares de 10 a 2 en orden inverso.

Estructura mientras

Esta estructura repite una acción o grupo de acciones mientras una expresión lógica sea cierta; la sintáxis en la notación algorítmica y en FORTRAN es la siguiente:

mientras expresión lógica hacer

  acción

fin desde

do while (expresión lógica)

  acción

end do

Un aspecto muy importante de la presente estructura de control es que si la expresión lógica es inicialmente falsa el bucle no se ejecuta ni una sola vez; es decir, la estructura mientras supone que el bucle iterará 0 ó más veces.

A continuación, se muestra un ejemplo que solicita al usuario el radio de una circunferencia mientras el radio introducido sea incorrecto:

mientras radio<0 hacer

  escribir 'Radio?'

```
  leer radio
fin desde
do while (radio<0)
  fprintf *, 'Radio?'
  read *, radio
end do
```

2.11 CICLOS.

Un condicional, como su nombre lo indica, es una condición para discernir entre una opción u otra, y en el proceso mental normalmente se manifiesta con un "Si"; por ejemplo: Si (va a llover), coge el paraguas.

Operadores lógicos

Para crear condiciones, por muy simples que sean, se necesitan los operadores lógicos. A continuación, voy a explicarlos de modo que, después de leer este artículo, puedas escribir condiciones ajustadas a lo que necesites.

== significa "igual". If x==y, significa "si x es igual a y"

> significa "mayor que". If x>y, significa "si x es mayor que y"

< significa "menor que". If x<y, significa "si x es menor que y"

!= significa "si es distinto". If x!=y, significa "si x es distinto de y"

&& significa "Y", la conjunción copulativa; es decir: If (x==y) && (x==z), significa "si x es igual a y Y x igual a z"

|| significa "O", la conjunción adversativa; es decir, If (x==y) || (x==z), significa "si x es igual a y O x igual a z"

IF

Sintácticamente, es la palabra reservada para desencadenar el poder de los condicionales en el código.

ELSE

Expresa "en el caso contrario". Siguiendo con el ejemplo anterior de la lluvia: if(va a llover) coge el paraguas else coge el bañador.

```
1   //Condicional que los operadores == || != recordad igual a o distinto a
2   var nombre="Juan"
3   var nombre2="Pedro"
4   if(nombre=='Juan' || nombre2!='Pedro'){
5       console.log("Eres juan y no pedro")
6   }else{
7       console.log("No eres Juan y puede que pedro")
8   }
9   //Condicional que los operadores == && == recordad igual a Y  distinto a
10  if(nombre=='Juan' && nombre2=='Pedro'){
11      console.log("Eres juan y pedro")
12  }else{
13      console.log("No estais los dos juntos")
14  }
```

SWITCH

Es una estructura de control diseñada para diferentes condiciones ligadas a una decisión. Básicamente, es como poner una pila de IF, que sería más eficiente en cuanto a rendimiento, pero haría que tu código fuera ilegible, así que… ¡no es una opción!

Su sintaxis se basa en las palabras *switch* (variable) y *case*. Estos últimos plantean los casos en los que el código se dispara: si el case está a 1 y la variable contiene ese valor, se disparará el código que esté dentro de él. Cada uno de los casos se delimita con la palabra reservada *break*.

```
var opcion=2

switch(opcion){
    case 1:
    console.log("has seleccionado la opcion 1")
    break;
    case 2:
    console.log("has seleccionado la opcion 2")
    break;
    default:
    console.log("Ninguna opcion valida seleccionada")
    break;
}
```

BUCLE FOR

En mi opinión, es el "rey de los ciclos", porque con esta estructura se puede hacer cualquier cosa. Literalmente, todo lo que puedas hacer con los demás *bucles* se puede hacer con este ciclo. Ahora bien, si no quieres acabar escribiendo un código malo, será mejor que aprendas los demás bucles y que los uses cuando toque.

La sintaxis es la siguiente: for(condición inicial; condición de parada; ritmo de iteración). No te preocupes si aún no entiendes del todo lo que vas leyendo, voy a explicar poco a poco en qué consiste cada uno de los conceptos que he planteado en la definición del bucle *for*.

Condición inicial: como todo en esta vida tiene un principio y un fin, este es el principio del *bucle*. Si, por ejemplo, pones la *x=0*, el bucle empezará en 0, así que vamos completando: for(x=0;condición de parada; ritmo de iteración).

Condición de parada: ¿te acuerdas eso de que todo tiene un principio y un fin? Pues, bien, este es el fin. Si, por ejemplo, pones *x<10*, la parada se producirá cuando la x llegue a 10.

for(x=0;x<10; ritmo de iteración).

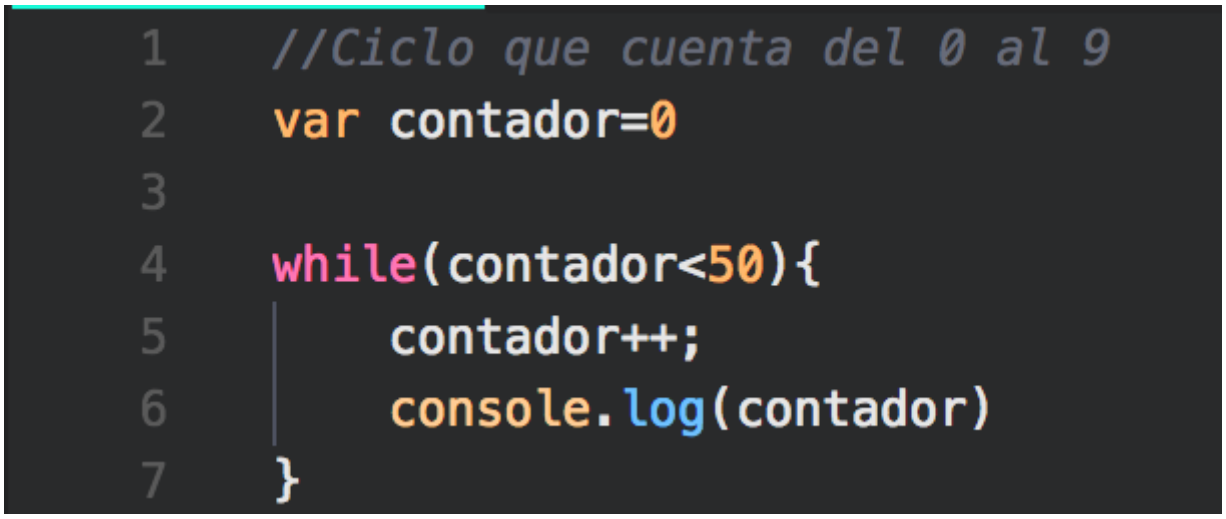
Ritmo de iteración: básicamente, es el ritmo al que se consume el *bucle*. Si, por ejemplo, pusiéramos *x=x+2*, el *bucle* itera de 2 en 2, aunque lo más común es que se haga de 1 en 1 con la siguiente sintaxis *x++*.

for(x=0;x<10;x++)

```
1   //Ciclo que cuenta del 0 al 9
2   var x
3
4   for(x=0;x<10;x++){
5       console.log(x)
6   }
```

BUCLE WHILE

El bucle *while* es más sencillo de entender. Si has entendido el bucle *for,* este también será fácil. Básicamente, esta es una estructura iterativa a la que solo hay que pasarle una condición de parada: *while(condicion de parada)*.



```
1  //Ciclo que cuenta del 0 al 9
2  var contador=0
3
4  while(contador<50){
5      contador++;
6      console.log(contador)
7  }
```

MATERIAL WEB COMPLEMENTARIO

https://www.youtube.com/watch?v=pxNYRRrJaTM

Aplicando este principio a la hora de hacer un programa, entonces habría que dividir el programa en “subprogramas” que realicen tareas específicas. Es importante considerar que para poner en práctica la modularización, es necesario un mecanismo que permita aplicarla en los lenguajes de programación

3.1.- DECLARACIÓN DE FUNCIONES.

s posible declarar una **función de usuario** para realizar tareas repetitivas y para realizar una programación más ordenar, clara e intuitiva. Las funciones te permiten crear piezas modulares de código, de forma que se puedan realizar ciertas rutinas y retornar al área de código desde donde se realizó la llamada a dicha función.

En primer lugar, se declara el tipo de la función, que será el valor retornado por la función (int, byte, long, flota, void). A continuación del tipo, se declara el nombre de la función y, entre paréntesis, los parámetros que se pasan a la función. Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”.

```
tipo nombreDeLaFuncion(parametro)
{
código;
 }
```

Para definir el principio y el final de un bloque de instrucciones, declaraciones y sentencias, se utiliza las llaves “**{}**”. Una llave de apertura “{“siempre debe ir seguida de una llave de cierre “}”. Las llaves no balanceadas provocan errores de compilación.

Cuando Arduino ejecuta una función, busca la declaración de dicha función en algún lugar del código, y le pasa el valor de la variable “parametro” como un argumento (contenido entre los paréntesis) a la función.

Los parámetros pasados a la función, deben ser definidos también con sus tipos y nombres, de igual forma a como se definen las variables. (enlace a declaracion de variables).

Dentro de la función, los parámetros y las variables utilizadas se definen como "locales" a la función, es decir se utiliza una memoria local que sólo existe cuando la función se está ejecutando.

Las variables declaradas fuera de la función, serán consideradas como "globales", es decir, que cualquier función puede acceder a ellas.

El comando "return" al final de la función es necesario si se desea devolver el valor calculado en la función y salirse de la función. El valor devuelto, debe ser del mismo tipo que el definido en la declaración de la función.

```
tipo nombreDeLaFuncion(parametro)
{
código;
return(parametro_de_vuelta);
 }
```

3.2 PASO DE PARÁMETROS POR VALOR O POR REFERENCIA

Los parámetros o argumentos pueden ser pasados por dos métodos:

- Por valor
- Por referencia

Cuando son pasados “por valor “, significa que sólo se aplica una simple operación de asignación o copiado entre las variables externas a la función y las variables locales a la función.

Cuando son pasados “por referencia “, se utiliza en el caso de que el parámetro sea de tipo estructura, como por ejemplo un vector. Con dicho método, en lugar de pasar una copia de un valor procedente del contexto de la llamada a la función, se pasa un puntero al valor de memoria que ocupa la estructura. Para indicar el paso del parámetro por referencia se utiliza el símbolo “*”.

```
tipo nombreDeLaFuncion(int *parametro)
{
código;
 }
```

3.3 SIMPLES.

En la jerga de la programación orientada a objetos, las funciones dentro de las clases se denominan funciones-miembro o métodos, y las variables dentro de clases, variables-miembro o propiedades. El sentido es el mismo que en la programación tradicional (la nomenclatura es más una cuestión de gustos), si bien referirnos a "propiedades" y "métodos" supone estar utilizando la programación orientada a objetos y que nos referimos a miembros de una clase. En C++ esta aclaración puede ser importante, porque es un lenguaje que podríamos llamar "híbrido"; en ciertas partes puede utilizarse con técnicas de programación tradicional, y en otras con técnicas de POO.

Una función de inicio

Cada programa debe tener una sola función externa denominada main(), principal, que desde la óptica del programador define el punto de entrada al programa. Las funciones se

declaran en cabeceras (estándar o específicas de usuario) o dentro de los ficheros fuente. Estas declaraciones son denominadas prototipos. En ocasiones la declaración y definición se realiza en el mismo punto (como ocurre con las variables), aunque es normal colocar al principio de la fuente los "prototipos" de las funciones que serán utilizadas en su interior, y las definiciones en cualquier otro sitio (generalmente al final). En el caso del ejemplo anterior, la declaración y definición de func1 se ha realizado en el mismo punto, mientras que la declaración de func2 se realiza dentro del cuerpo de la clase y la definición en el exterior de esta. La forma general del prototipo de una función es: valor-devuelto nombre-función (lista-de-argumentos); La forma general de la definición es:

```
valor-devuelto nombre-función (lista-de-argumentos) {
  sentencias;         // "cuerpo" de la función
}
```

Ejemplo:

```
float cuadrado (float x);                 // prototipo
float cuadrado (float x) { return x*x; } // definición
```

La comunicación entre el programa y las funciones que lo componen se realiza mediante los argumentos de llamada, los valores devueltos y las variables globales y externas.

3.4 CON PARÁMETROS.

Normalmente, las funciones operan sobre ciertos valores pasados a las mismas ya sea como constantes literales o como variables, aunque se pueden definir funciones que no reciban parámetros. Existen dos formas en de pasar parámetros a una función; por referencia o por valor. El hecho es que si en una declaración de función se declaran parámetros por

referencia, a los mismos no se les podrá pasar valores literales ya que las referencias apuntan a objetos (variables o funciones) residentes en la memoria; por otro lado, si un parámetro es declarado para ser pasado por valor, el mismo puede pasarse como una constante literal o como una variable. Los parámetros pasados por referencia pueden ser alterados por la función que los reciba, mientras que los parámetros pasados por valor o copia no pueden ser alterados por la función que los recibe, es decir, la función puede manipular a su antojo al parámetro, pero ningún cambio hecho sobre este se reflejará en el parámetro original.

Parámetros constantes

Los parámetros usados por una función pueden declararse como constantes (const) al momento de la declaración de la función. Un parámetro que ha sido declarado como constante significa que la función no podrá cambiar el valor del mismo (sin importar si dicho parámetro se recibe por valor o por referencia).

Parámetros con valor por defecto

Los parámetros usados por una función pueden declararse con un valor por defecto. Un parámetro que ha sido declarado con valor por defecto es opcional a la hora de hacer la llamada a la función

3.5 USO DE BIBLIOTECAS DE FUNCIONES.

Las bibliotecas están clasificadas por el tipo de trabajos que hacen, hay bibliotecas de entrada y salida, matemáticas, de manejo de memoria, de manejo de textos y como imaginarás existen muchísimas librerías disponibles y todas con una función específica.

Hay un conjunto de bibliotecas (o librerías) muy especiales, que se incluyen con todos los compiladores de C y de C++. Son las librerías (o bibliotecas) ANSI o estándar. También

hay librerías que no son parte del estándar, pero en esta sección sólo usaremos algunas bibliotecas (o librerías) ANSI.

3.6 TIPOS DE BIBLIOTECAS DE FUNCIONES

3.6.1 java.lang

Contiene clases esenciales para el lenguaje java y es el único paquete se importa automáticamente.

Aquí están las declaraciones de los objetos, clases, threads, excepciones, wrappers de los tipos de datos primitivos y otras clases fundamentales.

Interfaces

Clases:

- Cloneable
- Boolean
- Comparable
- Byte
- Runnable
- Character
- ClassLoader
- Compiler
- Double
- Float
- InheritableThreadLocal

- Interger
- Long
- Math
- Number
- Object
- System
- Thread
- Void
- String, etc…

3.6.2 java.io

Soporta flujos de entrada y salida java.

Interfaces

Clases:

- DataInput
- BufferedInputStream
- DataOutput
- BufferedOutputStream
- Externalizable
- BufferedReader
- FileFilter
- BufferedWrite
- FilenameFilter

- ByteArrayInputStream
- ObjectInput
- ByteArrayOutputStream
- Serializable
- DataOutputStream

3.6.3 Java.net

Soporta facilidades de red (URL, sockets TCP, sockets UDP, direcciones IP, conversiones binarias a texto).

Interfaces

Clases:

- ContentHandlerFactory Authenticator
- DatagramSocketImplFactory
- ContentHandler
- FileNameMap
- DatagramPacket
- SocketOptions
- DatagramSocketImpl
- URLStreamHandlerFactory
- HttpURLConnection
- URL, etc…

3.6.4 Java.until

Contiene diversas clases de utilidad (conjunto de bits, enumeración, contenedores, genéricos, Vectores y Hashtable, fecha, hora separación de token, generación de números aleatorios, propiedades del sistema).

Interfaces

Clases:

- Collection
- AbstractCollection
- Comparator
- AbstracList
- Enumeration
- AbstractMap
- EventListener
- AbstractSecquentialList
- Iterator
- AbstractSet
- List
- ArrayList
- Observer
- Collection
- SortedSet
- EventObject
- Random
- Stack Timer

- Vector
- Date

3.6.5 Java.awt

La librería java.awt proporciona un Abstract Window Toolkit para programación GUI (Interfaz gráfica de usuario), dibujo de gráficos e imágenes, asi como también eventos colores, tipo de letras, botones, campos de texto, etc.

Interfaces

Clases:

- ActiveEvent
- AlphaComposite
- Adjustable
- AWTEvent
- Composite AWTEventMulticaster
- LayoutManager
- BorderLayout
- Paint
- CardLayout
- Cursor
- Dialog
- Event
- Font
- Frame

- Graphics
- Image
- Label
- Menu
- Scrollbar.

3.6.6 Java.applet

El paquete java.applet permite la creación de applets a través de la clase Applet, proporciona interfaces para conectar un applet a un documento Web y para audición de audio.

Interfaces

Clases:

- AppletContext
- Applet
- AppletStub
- AudiClip

3.6.7 Java.math

Proporciona cálculos en entero grande y real grande.

Clases

- Bigdecimal
- Biginteger

3.6.8 java.rmi

Este paquete hace posible que un objeto se ejecute en una maquina virtual Java invoque métodos de otro objeto que se ejecuta en la maquina virtual distinta; dichas maquina virtuales pueden encontrarse en ordenadores diferentes conectados a través de una red TCP/IP.

Interfaces

Clases:

- Remote
- MarshalledObject
- Naming
- RMISecurityManager

3.6.9 Java.text

Contiene clases que permiten dar formato especializado a fechas, números y mensajes.

Interfaces

Clases:

- AttributedCharacterIterator Annotation
- CharacterIterator
- AttibutedCharacterIterator
- ChoceFormat
- DateFormat
- Format
- MessageFormat
- NumberFormat

- ParsePosition

Java.sound.midi

Paqueta con clases e interfaces que permiten la captura, procesamiento y reproducción de música MIDI.

Interfaces

Clases:

- ControllerEventListener
- Instrument
- MetaEventListener
- MeteMessage
- MidiChannel
- MidiDevice.Info
- MidiDevice
- MidiEvent
- Receiver
- MidiFileFormat
- Sequencer
- Midimenssage

3.6.10 Java.sql

Junto con el paqueta javax.sql, incluido en java 2 SDK Edition para la empresa, forma parte del API JDBC 2.0 (Conexión Java a Bases de Datos), y permite la conexión a bases de datos, el envío de sentencias SQL y la interpretación de los resultados de las consultas.

Interfaces

Clases:

- Array
- Date
- Blob
- DriverManager
- CallabeStatement
- DriverPropertyInfo
- Clob
- SQLPermission
- Connection
- Timer
- DatabaseMetaDate
- Timestamp
- Driver
- Types
- Ref
- SQLData
- SQLInput
- SQLOutput
- Struct

3.6.11 Javax.swing

Paquete que mejora el AWT, proporcionando un conjunto de componentes que se ejecutan uniformemente en todas las plataformas.

Interfaces

Clases:

- Action
- AbstractAction
- ComboBoxEditor
- ActonMap
- Icon
- Box.Filler
- ListModel
- CellRendererPane
- MenuElement
- DebugGraphics
- WindowsConstants
- DefaulListSelectionModel
- JApplet
- JButton
- JCheckBox
- JFrame
- JMenu
- JLabel
- JPanel

- JTextField
- JTree
- JWindows
- Timer
- UIManager, etc….

3.7 ENTRADA Y SALIDA.

Cuando nos referimos a entrada/salida estándar (E/S estándar) queremos decir que los datos o bien se están leyendo del teclado, ó bien se están escribiendo en el monitor de video.

En el lenguaje c++ tenemos varias alternativas para ingresar y/o mostrar datos, dependiendo de la librería que vamos a utilizar para desarrollar el programa.

Las operaciones de entrada y salida no forman parte del conjunto de sentencias de C++, sino que pertenecen al conjunto de funciones y clases de la biblioteca estándar de C++. Ellas se incluyen en los archivos de cabecera iostream.h por lo que siempre que queramos utilizarlas deberemos introducir la línea de código #include <iostream.h>

Esta biblioteca es una implementación orientada a objetos y está basada en el concepto de flujos. A nivel abstracto un flujo es un medio de describir la secuencia de datos de una fuente a un destino o sumidero. Así, por ejemplo, cuando se introducen caracteres desde el teclado, se puede pensar en caracteres que fluyen o se trasladan desde el teclado a las estructuras de datos del programa.

Los objetos de flujo que vienen predefinidos serán:

- cin, que toma caracteres de la entrada estándar (teclado);
- cout, pone caracteres en la salida estándar (pantalla);
- cerr y clog ponen mensajes de error en la salida estándar.

SALIDA (OUT)

El operador de insercion,

<<, inserta datos en flujo, ejemplos:

```
cout
<< 500; // envia el numero 500 a la pantalla

cout
<< " esto es una cadena " ; // visualiza Esto es una cadena
```

Es posible usar una serie de operadores

<< en cascada, ejemplo:

```
cout
<< 500
<< 600
<< 700;
```

visualiza 500, 600, 700.

De igual modo,

cout

<< 500

<<", "

<< 600

<< ", "

<< 700;

visualiza 500, 600, 700

C++ utiliza secuencias de escape para visualizar caracteres que no están representados por

simbolos tradicionales, tales como \a, \b, etc.

ejemplo:

cout << "\n" // salta a una nueva línea

cout << "Yo estoy preocupado \n no por el funcionamiento \n sino porla claridad \n";

ENTRADA (CIN)

El archivo de cabecera iostream.h de la biblioteca C++proporciona un flujo de entrada estándar cin

y un operadorde extracción, >>, para extraer valores del flujo yalmacenarlos en variables, la entrada normal es el teclado.

ejemplo:

int n, n1, n2; // se declara las variables n, n1 y n2

```
cin >> n; // el cursor aparece en pantalla esperando un
                // date de entrada por el teclado para ponerlo
               // en la variable n.


// abajo, el cursor aparece despues del mensaje entre comillas
// esperando dos valores, cada uno seguido de la tecla ENTER.


cout << " introdusca los valores de n1 y n2 " ;
cin >>  n1 >>  n2
```

3.8 ARCHIVOS.

Todos los datos que un programa utiliza durante su ejecución se encuentran en sus variables, que están almacenadas en la memoria RAM del computador.

La memoria RAM es un medio de almacenamiento **volátil**: cuando el programa termina, o cuando el computador se apaga, todos los datos se pierden para siempre.

Para que un programa pueda guardar datos de manera permanente, es necesario utilizar un medio de almacenamiento **persistente**, de los cuales el más importante es el disco duro.

Los datos en el disco duro están organizados en archivos. Un **archivo** es una secuencia de datos almacenados en un medio persistente que están disponibles para ser utilizados por un programa. Todos los archivos tienen un nombre y una ubicación dentro del sistema de archivos del sistema operativo.

Los datos en un archivo siguen estando presentes después de que termina el programa que lo ha creado. Un programa puede guardar sus datos en archivos para usarlos en una ejecución futura, e incluso puede leer datos desde archivos creados por otros programas.

Un programa no puede manipular los datos de un archivo directamente. Para usar un archivo, un programa siempre abrir el archivo y asignarlo a una variable, que llamaremos el **archivo lógico**. Todas las operaciones sobre un archivo se realizan a través del archivo lógico.

Dependiendo del contenido, hay muchos tipos de archivos. Nosotros nos preocuparemos sólo de los **archivos de texto**, que son los que contienen texto, y pueden ser abiertos y modificados usando un editor de texto como el Bloc de Notas. Los archivos de texto generalmente tienen un nombre terminado en .txt.

3.9 CADENAS.

Cadena de caracteres. (string en inglés). Es una secuencia ordenada de longitud arbitraria (aunque finita) de elementos que pertenecen a un cierto lenguaje formal o alfabeto; análogas a una frase o a una oración.

En general, una cadena de caracteres es una sucesión de **caracteres**: (**letras**, **números**, **espacio**, **signos** o **símbolos**).

En programación, si no se ponen restricciones al alfabeto, una cadena podrá estar formada por cualquier combinación finita de todo el juego de caracteres disponibles (las letras de la

'a' a la 'z' y de la 'A' a la 'Z', los números del '0' al '9', el espacio en blanco ' ', símbolos diversos '!', '@', '%', etc).

En este ámbito se utilizan normalmente como un tipo de dato predefinido, para palabras, frases o cualquier otra sucesión de caracteres. En este caso, se almacenan en un vector de datos, o matriz de datos de una sola fila (array en inglés). Las cadenas se pueden almacenar físicamente seguidas o enlazados letra a letra.

Generalmente son guardados un carácter a continuación de otro por una cuestión de eficiencia de acceso.

Un caso especial de cadena es la que contiene cero caracteres, a esta cadena se la llama cadena vacía.

Al considerar las cadenas como un tipo de datos, existen varias operaciones que se pueden hacer con ellas:

- Asignación:asignarle una cadena a otra.
- Concatenación: unir dos cadenas o más (o una cadena con un carácter) para formar una cadena de mayor tamaño.
- Búsqueda: localizar dentro de una cadena una subcadena más pequeña o un carácter.
- Extracción: sacar fuera de una cadena una porción de la misma según su posición dentro de ella.
- Comparación: comparar dos cadenas.

Una cadena suele ser representada entre comillas dobles superiores ("palabra"), mientras que un carácter de esa cadena (un char en inglés) suele ser representado entre comillas simples ('p'). Ejemplo, en el lenguaje de programación C:

- char c = 'a';
- char str[5] = "hola";

- Generalmente para acceder a un carácter en una posición determinada se suele usar la forma variable posición como cuando se accede a un vector.

Las cadenas de caracteres pueden ser:

- De naturaleza dinámica: pueden alterar su longitud durante el tiempo de ejecución,
- De naturaleza estática: su longitud es fija a lo largo del tiempo de ejecución.

3.10 DEFINICIÓN E IMPORTANCIA DE LOS ARREGLOS EN LA PROGRAMACIÓN.

Los **Arreglos** Es una colección de datos del mismo tipo, que se agrupan con un nombre.

Se clasifican en vectores y matrices.

Sintaxis

Tipo de datos nombre_vector[tamaño];

Tipo de datos nombre_arreglo[tamaño][tamaño];

Ejemplo:

float calificaciones[6];

float matrices[3][3];

3.11 DECLARACIÓN DE ARREGLOS UNIDIMENSIONALES Y MULTIDIMENSIONALES

ARREGLO UNIDIMENSIONAL:

* Un arreglo se usa para almacenar elementos del mismo tipo.
* Un arreglo es de tamaño fijo.
* Cada elemento se guarda en un espacio independiente.
* Cada espacio se referencia con un índice (0,1,2,3,…,n).

La declaración de un arreglo se hace de la siguiente forma:

TipoDeDato nombre[] = new TipoDeDato [n];
Donde n es la capacidad (tamaño) del arreglo.
Ejemplos:

String nombres[ ] = new String [4];
double notas[ ] = new double [5];
int edadEstudiantes[ ] = new int [3];

String nombres[] = new String [4];
Define un arreglo llamado nombres, que almacena cadenas de texto y puede contener máximo 4 elementos (con índices 0, 1, 2 y 3).

double notas[] = new double [5];
Define un arreglo llamado notas, que almacena números reales y puede tener máximo 5 elementos (con índices 0, 1, 2, 3 y 4).

int edadEstudiantes[] = new int [3];
Define un arreglo llamado edadEstudiantes, que almacena números enteros y puede tener máximo 3 elementos (con índices 0, 1 y 2).

TipoDeDato nombre[] = new TipoDeDato [n];
Los arreglos definidos de esta forma no están inicializados, es decir, no contienen ningún valor, lo cual se representa con null si es String o con 0 si es int o double.

ARREGLOS MULTIDIMENSIONALES:

Los arreglos multidimensionales tienen más de una dimensión. En C#, las dimensiones se manejan por medio de un par de corchetes, dentro de los que se escriben los valores de cada dimensión, separados por comas.

Operaciones.

Para manejar un arreglo, las operaciones a efectuarse son:

Declaración del arreglo

Creación del arreglo

Inicialización de de los elementos del arreglo

Acceso a los elementos del arreglo.

A continuación describiremos cada una de estas operaciones, en C#.

Declaración.

La sintaxis para declarar un arreglo multidimensional es la siguiente:

<tipo> [, …] < identificador > ;

Donde:

tipo indica el tipo correspondiente a los elementos del arreglo ,

identificador es el nombre del arreglo, y

el par de corchetes, la coma y las diéresis, [, …], representan las dimensiones del arreglo. Los corchetes encierran todas las comas necesarias para separar las dimensiones del arreglo.

Ejemplos:

double [,] bidim; // Dos dimensiones.

int [, ,] tridim ; // Tres dimensiones.

char [, , ,] enciclopedia; // Cuatro dimensiones.

3.12 ARREGLOS UNIDIMENSIONALES Y MULTIDIMENSIONALES

Arreglos Unidimensionales

Un Arreglo Unidimensional es una estructura de datos organizada y bien coordinada, que cuenta con una cantidad pequeña de datos, los cuales deben ser del mismo tipo para ser procesados en dicho Arreglo, en programación se usa mayormente en la creación de la estructura de listas en orden natural y totalmente con elementos similares en su interior.

Los datos a manipular en su interior no solo deben poseer un tipo de dato similar, sino que también deben poseer un seudónimo similar entre ellos, pero los mismos se han de diferenciar con la posición que se les otorgue dentro del Arreglo bajo el uso de sus

funciones y ordenanzas básicas estipuladas en la línea de programación con su codificación especial.

Además, un Arreglo de este tipo para ejecutar sus funciones debe primero que todo iniciar sus variables o datos en el inicio del programa que se esté llevando a cabo, como de la misma manera en la misma sección donde se está realizando dicha acción, se debe establecer tanto el nombre como el tipo de dato que ha de ejecutar dicho Arreglo en su interior.

Arreglos Multidimensionales

Los Arreglos cuya estructura sea de dos o más dimensiones se les conoce como "Arreglos Multidimensionales", en ellos el termino dimensiones se establece a los diferentes números de índices que los mismo deben llevar en su estructura para poder llevar a cabo sus funciones, el número de índices a utilizar deben ser preestablecidos al igual que los datos en la misma forma que los Arreglos unidimensionales con la diferencia que el presente contara con una estructura más robusta y con más funciones.

3.13 ESTRUCTURAS DE SELECCIÓN

Una decisión es la estructura según la cual se puede escoger uno de entre dos caminos lógicos dependiendo de una condición que al ser evaluada nos brinda la oportunidad de saber cuál de los dos caminos escoger. La evaluación de dicha condición siempre va a originar una respuesta VERDADERA (cuando la condición se cumple) o FALSA (cuando dicha condición no se cumple) y con ello se podrá saber cuál es el conjunto de instrucciones a resolver. 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.

La condición, en algoritmos técnicos, se podrá expresar en términos de dos tipos de operadores: los operadores relacionales y los operadores booleanos. Recordemos que los operadores relacionales son aquellos que nos originan una respuesta Verdadera o Falsa y que corresponden a los símbolos mayor que, menor que, mayor o igual, menor o igual, igual (de comparación) y diferente de. Los operadores booleanos son aquellos que nos permiten

establecer conexiones entre expresiones en donde aparezcan los operadores booleanos y corresponden a los operadores:

- AND: Genera Verdadero si todas las expresiones relacionales conectadas son Verdaderas
- OR: Genera Verdadero si al menos una de las expresiones conectadas es Verdadera
- NOT que invierte el sentido lógico de la expresión

Con estos elementos podemos recordar que la utilización de las decisiones como estructura básica de programación no tiene ninguna restricción y que pueden considerarse como válidos los siguientes casos:

- Una decisión dentro de otra
- Una decisión a continuación de otra
- Muchas decisiones dentro de otras
- Muchas decisiones a continuación de otras

3.14 ESTRUCTURAS DE REPETICIÓN

La estructura de repetición o bucle hace posible la ejecución repetida de una o más instrucciones.

Las estructuras de repetición nos permiten ejecutar varias veces unas mismas líneas de código

Estas estructuras describen procesos que se repiten varias veces en la solución del problema.

El conjunto de acciones que se repiten conforman el cuerpo del bucle y cada ejecución del cuerpo, del bucle se denomina iteración

REPITA PARA

Se utiliza para repetir una sentencia o grupo de sentencias un número fijo de veces.

REPITA PARA variable_de_control:= valor_inicial HASTA valor_final

Sentencia 1;

Sentencia 2;

FIN RP

• REPETIR HASTA

Se utiliza para repetir una sentencia o grupo de sentencias hasta que una condición especificada sea verdadera.

Repetir

Sentencia 1;

Sentencia 2;

.....

Sentencia n;

hasta Condición;

• REPITA MIENTRAS

Se utiliza para repetir una sentencia o grupo de sentencias mientras una condición especificada sea verdadera.

Repita Mientras Condición se cumpla haga

Sentencia 1;

Sentencia 2;

.....

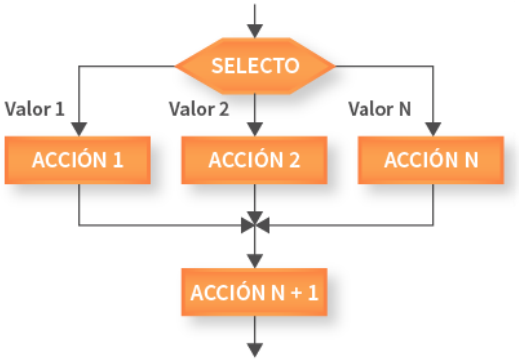
Sentencia n;

fin rm

3.15 ESTRUCTURA DE MÚLTIPLE SELECCIÓN

La estructura selectiva **si múltiple** permite que el flujo del diagrama se bifurque por varias ramas en el punto de la toma de decisión(es), esto en función del valor que tome el selector.

Así si el selector toma el valor 1 se ejecutará la acción 1, si toma el valor 2 se ejecutará la acción 2, si toma el valor N se realizará la acción N, y si toma un valor distinto de los valores comprendidos entre 1 y N, se continuará con el flujo normal del diagrama realizándose la acción N + 1.



MATERIAL WEB COMPLEMENTARIO

https://www.youtube.com/watch?v=OxAvFwYo3bw

UNIDAD IV APLICACIÓN DE PUERTOS DE COMUNICACIÓN

El alumno relacionará la programación estructura y la implementará en los sistemas de comunicación de la computadora.

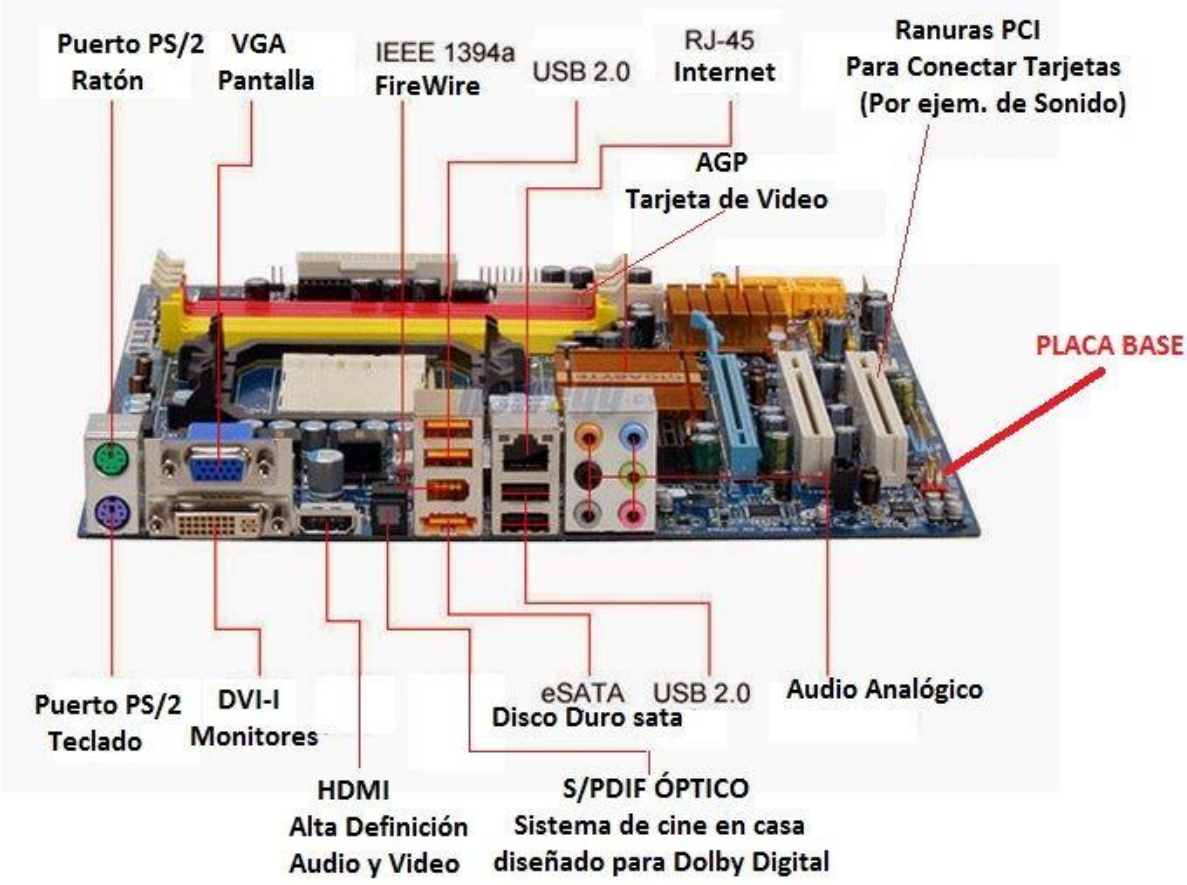
4.1 PUERTOS DE COMUNICACIÓN.

Los puertos del ordenador, también llamados puertos de comunicación, son una característica esencial de todos los dispositivos informáticos.

Los puertos de comunicación son aquellos donde se conectan los dispositivos externos al ordenador (también llamados periféricos) y por donde se recibe y/o envía información al ordenador desde el exterior. Comunicación del ordenador con el exterior.

Un puerto del ordenador es un punto de conexión con el que un dispositivo externo puede conectarse a la computadora.

Los puertos de comunicación, normalmente, son ranuras o conectores de la placa base en el cual se conecta el conector del dispositivo externo. En la siguiente imagen puedes ver los **conectores de la tarjeta madre** o placa base que luego explicaremos.



Estas ranuras o puertos suelen llamarse **conectores hembras**, ya que el macho suele ser el del extremo del cable del periférico que queremos conectar al ordenador y que se introduce en la hembra o puerto.

Ejemplos de dispositivos externos (periféricos) conectados a través de puertos son el ratón, el teclado, el monitor, el micrófono, los altavoces, etc.

La función principal de un puerto de comunicación es la de actuar como un punto de unión, donde el cable del periférico puede ser enchufado y permite el flujo de datos desde y hacia el ordenador.

4.2 TIPOS DE PUERTOS DE UNA COMPUTADORA

Puerto serie

Un puerto serie es una interfaz a través del cual los periféricos se pueden conectar mediante un protocolo serie que consiste en la **transmisión de datos de un bit detrás de otro y a través de una sola línea de comunicación**. Solo puede enviar un bit a la vez. El tipo más común de puerto serie es un D-sub o un conector D-sub que llevan señales RS-232. Normalmente, los puertos serie tienen 9 ó 25 clavijas o pins. Es el que utilizaban antiguamente los ratones y módems. **Los datos viajan a 115 kilobits por segundo**.

Puerto paralelo

Un puerto paralelo, por otro lado, es una interfaz a través del cual la comunicación entre un ordenador y su dispositivo periférico es de manera paralela, es decir **los datos se transfieren en paralelo utilizando más de una línea de comunicación (canales)**. Pueden enviar varios bits a la vez. El puerto de la impresora es un ejemplo de puerto paralelo.Se utiliza, cada vez menos, para escáneres, impresoras y los llamados **PS/2** para el teclado y el ratón del ordenador.

Hoy en día casi todos los periféricos utilizan los puertos llamados USB.

Puerto USB (Universal Serial Bus)

Se puede conectar todo tipo de dispositivos USB externos, tales como un disco duro externo, una impresora, un escáner, ratón, teclado, etc. Fue introducido en 1997 para para estandarizar cómo se conectan los periféricos, es decir que todos los periféricos se conectaran de la misma forma. Su característica principal es que son plug and play (enchufar y usar) que significa que se pueden conectar y desconectar con el ordenador encendido y ya reconoce el periférico conectado. Los antiguos puertos se tenían que apagar y reiniciar el ordenador para que los reconociera. Además, son más rápidos para transmitir la información. Los datos viajaban a 12 megabits por segundo en los primeros USB, aunque según se van mejorando se consiguen mayores velocidades.

Para saber que tipo de USB estamos utilizando se pone un número que nos dice la versión del puerto.

USB-1.0 = los datos viajan a 12Mb/s (OJO megabits por segundo, no KiloBits como dijimos en el serie y paralelo)

USB-2.0 = Los datos viajan a 480Mb/s.

USB-3.0 = Los datos viajan a 5Gb/s. (Gigabits por segundo). El 3.1 alcanza velocidades de 10Gb/s.

USB 4.0 = El futuro puerto USB.

Como ves en la imagen está el **USB A** que es el standar, el más utilizado, y luego hay otros que son más específicos para ciertos periféricos:

Los puertos y conectores de **USB del Tipo B** son pequeños y casi cuadrados, y sirven para conectar un cable USB a un dispositivo USB. En ocasiones se les denomina de "flujo ascendente", porque los datos van del dispositivo al ordenador. Se utiliza para discos duros, cámaras digitales, impresoras, etc.

El mini USB suele encontrarse con mayor frecuencia en cámaras de fotos, reproductores de música, discos duros externos y otros pequeños gadgets.

El micro USB es utilizado para dispositivos pequeños como SmartPhones, Tablets o una Phablet.

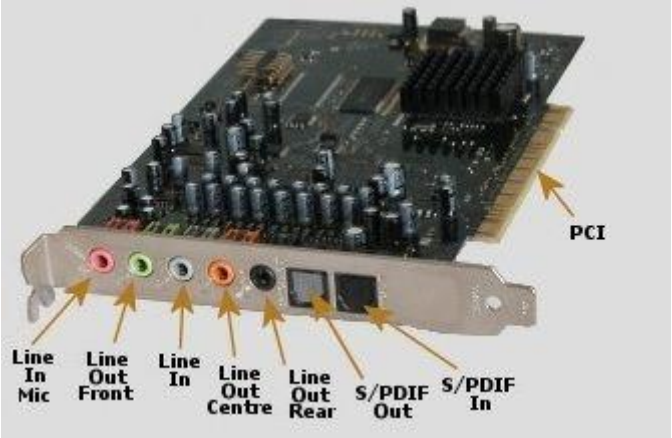
4.3 PUERTOS DE AUDIO ANALÓGICO. RCA

Permiten introducir o sacar audio del ordenador. Son clavijas y agujeros (puertos) llamados conectores RCA.

- Line In (entrada de línea) sirve para conectar con una fuente de sonido externa, como un sistema de alta fidelidad.

- Rosa = Line In Mic (entrada de micrófono) permite conectar un micrófono.

- Verde = Line Out (salida de línea) permite conectar unos altavoces o unos auriculares.



Después puedes ver que tenemos alguna más:

Gris: Salida para altavoces laterales.

Negro: Salida para altavoces traseros.

Naranja: Salida de altavoces para el canal central y subwoofer.

A veces también tenemos una entrada azul que es para conectar una entrada de audio como puede ser un reproductor casero de cd, o también puedes meter el audio de un reproductor de dvd, videocamara, etc.

Como puedes ver la tarjeta de sonido de la imagen se conecta al ordenador por medio del puerto o ranura PCI. Los otros puertos que ves los estudiaremos más adelante.

Ahora vamos hablar sobre los puertos de comunicación que se utilizan para conectar pantallas o monitores.

4.4 **PUERTO VGA**

Se conecta el monitor a la tarjeta de video de una computadora. Tiene 15 pins. Es similar al conector de puerto serie, pero el conector de puerto serie tiene pasadores metálicos (pins), y el VGA que tiene agujeros.

4.5 PUERTO DVI (INTERFAZ DE VÍDEO DIGITAL)

Donde se conecta el monitor LCD de pantalla para enviar video digital. Maximiza el rendimiento de monitores de pantalla plana. Son mejores que los VGA.

4.6 PUERTO HDMI

Para conectar Multimedia en Alta Definición (HD). Es el sustituto del llamado Euroconector que ya no se utiliza.

4.7 PUERTO S/PDFI

Diseñado por Philips y Sony para transferir audio digital comprimido y llevar la señal entre la salida de un equipo o Reproductor de DVD a un sistema de cine en casa Dolby Digital. Su principal ventaja es que es inalámbrico. En la primera imagen de los puertos de comunicación puedes verlo.

4.8 PUERTO FIREWIRE

Sirve para conectar videocámaras y equipos de vídeo al ordenador. Tiene transferencias de gran cantidad de datos a una velocidad muy rápida. Los datos viajan de 400 a 800 megabits por segundo. FueInventado por Apple.

Hay Tres variantes de puertos FireWare: 4-pin conector FireWire 400, 6-pin conector FireWire 400 y FireWire de 9 pines del conector 800.

4.9 PUERTO DE MÓDEM

Donde se conecta el módem de un PC a la red telefónica. Ya no se usa. Ahora se utiliza el puerto RJ45 o Ethernet.

Puerto RJ 45 o Ethernet

Donde se conecta a una red a Internet de alta velocidad. Se llama RJ45 (hembra) y el macho será el del cable del router. Este puerto está situado en una tarjeta Ethernet usadas para la conexión a internet.

Los datos viajan desde 10 megabits a 1000 megabits por segundo, dependiendo del ancho de banda de la red.

Puerto SATA

Es un puerto de forma especial con 7 terminales utilizado para conectar al ordenador discos duros SATA. Estos discos duros son más rápidos que los anteriores IDE o ATA.

Puerto MIDI

El protocolo MIDI fue creado para comunicar entre sí instrumentos musicales, aparatos y, hoy, también ordenadores, mediante la transmisión de información comprensible común. Ahora muchos videojuegos usan este conector para la música de los videojuegos.

El ordenador no posee este tipo de puertos, pero algunas tarjetas de sonido incluyen un puerto MIDI, aprovechando el mismo puerto que le sirve para la conexión del joystick. El cable deberá tener tanto las conexiones MIDI como la conexión propia del puerto de joystick. Gameport/MIDI (puerto de juegos/MIDI) permite conectar un joystick o un dispositivo con interfaz MIDI.

Conector de alimentación

Donde se conecta al cable de alimentación del ordenador y por el otro lado se conecta a un enchufe de la pared. Es un enchufe de tres puntas. Es el típico cable negro de alimentación del ordenador.

Ahora vamos hablar de dos ranuras que llevan internamente la placa base, mucha gente no las considera puertos, porque sirven para conectar tarjetas al ordenador.

Ranuras PCI

Peripheral Component Interconnect. Ranura para tarjetas de expansión más antiguos, tales como tarjetas de sonido, tarjetas de red, tarjetas de conexión. Se han sustituido en gran medida por las ranuras PCI-Express que son muy parecidas.

Ranuras AGP

Se dedica exclusivamente a conectar tarjetas de vídeo 3D, por lo que sólo suele haber una.

4.10 ESPECIFICACIONES DE LOS PUERTOS RS-232 Y PARALELO.

Qué es un puerto serie

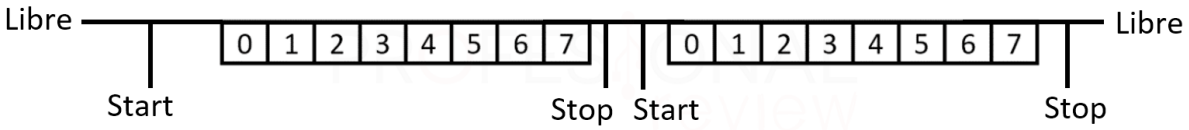
Si echas un vistazo a los cables que tienes ahora mismo en la mesa del escritorio que conectan el ratón teclado o unidad flash USB a tu equipo, estarás viendo **interfaces de comunicación serie.**

El puerto serie es una interfaz **de comunicación digital de datos en la que la información se transmite de forma secuencial bit a bit por los conductores**. De esta forma un puerto serie debe enviar toda la información en un bit detrás de otro, mientras que un puerto paralelo enviaría varios bits de forma simultánea. La interfaz de datos en serie o puerto serial trabaja **bajo el estándar RS-232.**

¿Crees entonces que un puerto serie es más lento que uno paralelo? Pues en la actualidad **tenemos puertos serie muchísimo más rápidos**. Aunque claro, estos no necesariamente se ajustan al estándar comentado, sino que son **versiones mejoradas** que dejan completamente obsoleto el puerto serie nativo. Siendo los más fáciles de implementar, con mejor compatibilidad e infinitamente más extendidos.

Funcionamiento del puerto serie y hardware

Este puerto funciona de forma **asíncrona**, gracias a un protocolo que inicia la transmisión con una señal de "**start**" que prepara el receptor para recibir la palabra (bits). Tras enviar esta palabra, que será un código ASCII para cada carácter, se envía una señal de "**stop**" para que el receptor descanse tras codificar la palabra y se mantenga a la espera para recibir otra.



Tenemos tres tipos de comunicación en serie:

Simplex: la transmisión es unidireccional, es decir, hay un solo emisor y un solo receptor, por ejemplo, en comunicaciones de radiodifusión.

Dúplex: cada extremo puede ser transmisor y receptor de forma simultánea, así que se utilizan bien cables distintos para enviar y recibir, o bien ondas con distintas frecuencias para no mezclarse.

Semi-duplex: es similar a la transmisión dúplex, pero cuando uno transmite el otro escucha, por ejemplo, dos walki talkies.

De esta forma debemos entender que en una comunicación con puerto serial **ambos dispositivos deben tener una entrada y una salida** así que los dispositivos se dividen en las categorías de **DTE** (Equipo de Terminal de Datos) y **DCE** (Equipo de Terminación de Circuito de Datos). Así que **un ordenador sería por un DTE** mientras que **el DCE sería el modem** o tarjeta programable. Para conectar dos DTE o dos DCE se debería utilizar un puente nulo que cruzara ambas señales.

Para gestionar la interfaz de comunicación tenemos un **chip UART o USART** (Transmisor y Receptor Asíncrono Universal). Su función es la de convertir las señales y tensiones de la CPU al estándar de comunicación. El chip **UART 8250** se usa para procesadores de 8 y 16 bits, mientas el que **UART 16550** para el resto a partir de los equipos IBM.

Puerto serie RS-232 y Pinout

RS-232

En la historia de la informática el puerto más utilizado ha sido el que transmite los datos en serie. **Su interfaz se estandarizó en 1962** gracias a la norma **EIA/TIA RS-232C**, para los amigos, RS-232 o "Estándar Recomendado 232". A su vez se creó la **recomendación V.24** que define los circuitos y señales de la interfaz, y la **recomendación V.28** que define los aspectos eléctricos.

El conector más extendido era el **DB-25**, después **simplificado a DB-9**, directamente denominado RS-232. Es importante **no confundir este conector con el puerto paralelo del mismo nombre, aunque se denomina D-Sub** . Estaba (y está) enfocado a su uso en conexiones entre computadoras y dispositivos externos con conexiones de tipo dúplex. Como por ejemplo un módem, switches y otros dispositivos de comunicación de automatización industrial como pueden ser **placas programables, robots y otros productos** de consumo general como lavadoras digitales.

A continuación, vamos a ver la configuración de pines del puerto RS-232 en su versión DB-9 y DB-25. En ambos casos se tenemos la misma cantidad de pines útiles.

Función (español)	Pin DB-9	Pin DB-25
DCD (Detector de datos)	1	8
RxD (Datos recibidos)	2	3
TxD (Datos transmitidos)	3	2
DTR (Terminal de datos preparado)	4	20
G (Tierra)	5	7
DSR (Conjunto de datos preparado)	6	6
RTS (Petición de envío)	7	4
CTS (Libre para envío)	8	5
RI (Indicador de llamada)	9	22

Usos actuales del puerto serie

Nuestros equipos de escritorio actuales **no cuentan ya con el puerto RS-232** implementado, ya que USB es la interfaz más actual y prácticamente compatible con todo tipo de PCB electrónicas. Pero todavía podemos encontrar este **puerto serie PCI mediante tarjeta de expansión** si nos dedicamos a la programación. Así mismo hay muchos **adaptadores RS-232 a USB.**

Estos son los **usos fundamentales** del puerto DB-9 o RS-232 en la actualidad

Módems, Switches, Routers, teléfonos por satélite o equilibradores de carga: todavía encontramos de forma interna o externa este tipo de puertos o cabeceras para modificar el microcódigo de equipos de red más antiguos y no gestionables por el usuario.

- **Lectores de códigos de barra por infrarrojos**: y otros equipos de supermercados relativamente antiguos.
- **Placas programables**, equipos de medición eléctrica y depuradoras de software.
- **Impresoras**: las más antiguas que no usan ni interfaz USB ni conector paralelo.

En general **equipos que no tengan USB** para actualizar su firmware.

Sobre todo, hablamos de dispositivos industriales y de red, en donde se prevea un uso por parte de usuarios con conocimientos técnicos.

Velocidad puerto serie (RS-232)

Antes de ver las versiones actuales del puerto serie, conviene conocer un poco cuales son las **velocidades que ha alcanzado** este tras actualizaciones de hardware y periféricos:

Velocidad (bit/s)	Tiempo por bit (µs)	Dispositivo
75	13333,30	
110	9090,90	Bell 101
134,5	7434,90	
150	6666,60	
300	3333,30	Bell 103
600	1666,70	
1200	833,30	Bell 202
1800	555,60	
2400	416,70	Modem V.27bis
4800	208,30	Modem V.27ter
7200	138,90	
9600	104,20	Modem V.32
14400	69,40	Modem V.33bis
19200	52,10	
31250	32,00	MIDI
38400	26,00	
56000	17,90	
57600	17,40	
76800	13,00	BACnet MS
115200	8,68	
128000	7,81	Adaptador ISDN
230400	4,34	LocalTalk
256000	3,91	

Estas velocidades se miden en **bits por segundo o baudios, medida habitual en los módems,** y son bastante bajas en comparación con los puertos series que tenemos actualmente como USB. Estando además gestionados directamente por software en lo que a ancho de banda y conexión con el periférico se refiere.

4.11 ENVÍO Y RECEPCIÓN DE DATOS.

La principal función del Puerto Serie es poder comunicarse con un PC para poder enviarle a éste el estado de sus sensores, información del proceso que está ejecutando, así como para recibir órdenes del PC para que modifique su comportamiento.

Una de las características del puerto serie es la velocidad de conexión (Los famosos baudios). Básicamente, lo que indica es la rapidez con la que se va a esperar la llegada de un nuevo dato al BUS serie, por lo general suele ser de 9600 bps.

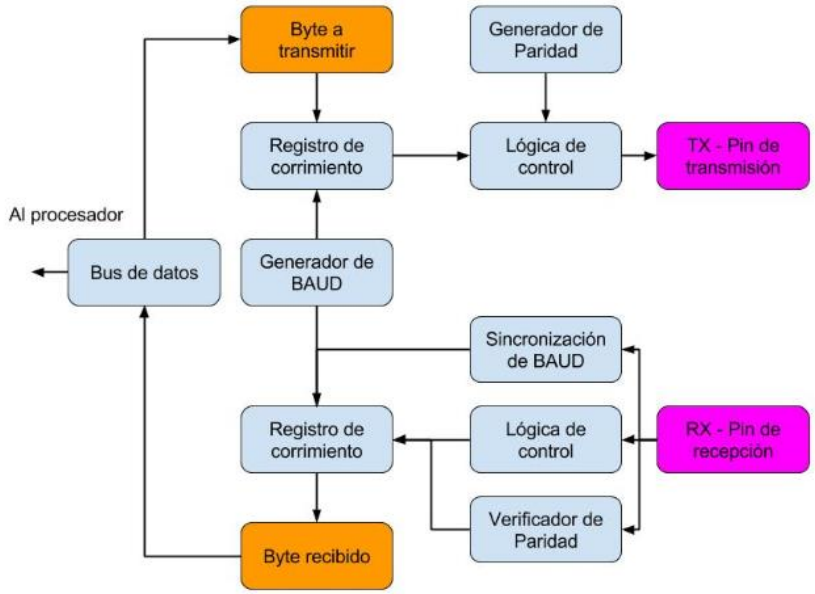
Un puerto Serial es un módulo de comunicación digital para un sistema embebido.Es decir, permite la comunicación entre dos dispositivos digitales. Cuenta con dos conexiones, RX y TX. Lo que nos indica los modos de comunicación que puede manejar, Full-duplex, Duplex y Simplex. Además podemos considerar como su principal ventaja a la sencillez de su protocolo de comunicación. Sin embargo también tiene desventajas como que sólo se puede comunicar a un puerto dos dispositivos.
Full duplex. Significa que puede recibir y enviar información digital simultáneamente.
Duplex o Half-duplex. Es cuando sólo podemos transmitir o recibir información, una cosa a la vez.
Simplex. Cuando sólo podemos ya sea recibir o transmitir.
La función principal de un puerto serial, es la de empacar y des-empacar paquetes de datos binarios seriales. Como resultado, la serialización significa convertir un dato paralelo (byte) a un conjunto de pulsos seriales que puedan ser recibidos y enviados por una línea de transmisión. En primer lugar, el protocolo serial opera mediante tres condiciones digitales básicas: inicio de transmisión (IT), paridad (P) y fin de transmisión (FT). Estas condiciones son sincronizadas mediante un oscilador interno. El generador permite controlar la

velocidad del puerto serial. Por lo tanto, la velocidad se mide en BAUD 's. Al modulo serial también se le conoce como UART ó USART o EUSART.

- UART – Universal Asyncronos Receiver and Transmitter que en español se traduciría como Transceptor Asíncrono.
- USART – Universal Syncronos and Asyncronos Receiver and Transmitter, que significa en español Transceptor Síncrono y Asíncrono.
- EUART – Enhanced Universal Asyncronos Receiver and Transmitter ó Transceptor Asíncrono Universal Mejorado.

ESTRUCTURA INTERNA Y CONFIGURACIÓN DE UN PUERTO SERIAL

Una UART contiene, en su estructura interna, un generador de paridad, registros de corrimiento, oscilador variable (para generar el BAUD), verificadores de las tres condiciones y lógica de control. Por consiguiente, la Figura-1, muestra un diagrama a bloques general para una UART. Un paquete de datos se transmite a través de un registro de corrimiento. Por lo tanto, la velocidad a la que se transmite, está controlada por el generador de BAUD. La lógica de control se encarga de agregar los bits de Inicio, Paridad y de Fin de transmisión [1]. El proceso de recepción serial es lo opuesto.



MATERIAL WEB COMPLEMENTARIO

https://www.youtube.com/watch?v=dSmV7JsPz7E

BIBLIOGRAFÍA BÁSICA Y COMPLEMENTARIA

- https://sites.google.com/a/espe.edu.ec/fundamentos-de-programacion/#:~:text=P%C3%A1gina%20Principal,-Fundamentos%20de%20Programaci%C3%B3n&text=Fundamentos%20de%20Programaci%C3%B3n%20es%20una,que%20exhiban%20un%20comportamiento%20deseado.&text=Se%20llama%20Programaci%C3%B3n%20a%20la,programaci%C3%B3n%2C%20para%20realizar%20un%20programa.
- https://programacionfacilysoftware.blogspot.com/2015/02/la-importancia-de-la-programacion-en.html#:~:text=Desde%20que%20las%20computadoras%20trabajan,codificar%20que%20estar%20eliminando%20errores.
- https://sites.google.com/site/programacionbasicajava/elementos-de-la-programacion
- https://www.genbeta.com/desarrollo/netbeans-1#:~:text=Netbeans%20es%20un%20entorno%20de,Web%2C%20o%20para%20dispositivos%20m%C3%B3viles.
- https://sites.google.com/site/portafoliodigitalguillermina/tema-2-elementos-del-lenguaje-de-programacion/2-1-introduccion-al-entorno-de-programacion
- https://plataforma.josedomingo.org/pledin/cursos/programacion/curso/u06/
- http://tic.taboadaleon.es/Unidad1-Programacion/Tema2_Lenguajes/contenido/5_estructura_de_un_programa_informtico.html#:~:text=Existen%20dos%20partes%20o%20bloques,para%20conseguir%20los%20resultados%20esperados.
- Principios de Diseño Digital /por Daniel D. Gajski, traducción de Carlos Garcia Puntonet y otros., Gajski, Daniel D.
- https://www.profesionalreview.com/2020/03/07/puerto-serie-que-es-para-que-sirve-y-tipos/
- https://concepto.de/memoria-ram/#ixzz6TG1JHn8S