ALGORITMOS Y PROGRAMACIÓN

Julio Cesar Rodríguez Casas

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Algoritmos y Programación
Julio Cesar Rodríguez Casas
Bogotá D.C.

Fundación Universitaria del Área Andina. 2018

Catalogación en la fuente Fundación Universitaria del Área Andina (Bogotá).

Algoritmos y Programación

ISBN (impreso): **978-958-5462-95-3**

Fundación Universitaria del Área Andina
Calle 70 No. 12-55, Bogotá, Colombia
Tel: +57 (1) 7424218 Ext. 1231
Correo electrónico: publicaciones@areandina.edu.co

Director editorial: Eduardo Mora Bejarano
Coordinador editorial: Camilo Andrés Cuéllar Mejía
Corrección de estilo y diagramación: Dirección Nacional de Operaciones Virtuales
Conversión de módulos virtuales: Katherine Medina

BANDERA INSTITUCIONAL

Pablo Oliveros Marmolejo †
Gustavo Eastman Vélez
Miembros Fundadores

Diego Molano Vega
Presidente del Consejo Superior y Asamblea General

José Leonardo Valencia Molano
Rector Nacional
Representante Legal

Martha Patricia Castellanos Saavedra
Vicerrectora Nacional Académica

Jorge Andrés Rubio Peña
Vicerrector Nacional de Crecimiento y Desarrollo

Tatiana Guzmán Granados
Vicerrectora Nacional de Experiencia Areandina

Edgar Orlando Cote Rojas
Rector – Seccional Pereira

Gelca Patricia Gutiérrez Barranco
Rectora – Sede Valledupar

María Angélica Pacheco Chica
Secretaria General

Eduardo Mora Bejarano
Director Nacional de Investigación

Camilo Andrés Cuéllar Mejía
Subdirector Nacional de Publicaciones

ALGORITMOS Y PROGRAMACIÓN

Julio Cesar Rodríguez Casas

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

ÍNDICE

EJE 1

Introducción 7
Desarrollo Temático 8
Bibliografía 33

EJE 2

Introducción 36
Desarrollo Temático 37

EJE 3

Introducción 58
Desarrollo Temático 59
Bibliografía 81

EJE 4

Introducción 84
Desarrollo Temático 85
Bibliografía 101

ALGORITMOS Y PROGRAMACIÓN

Julio César Rodríguez Casas

EJE 1

Conceptualicemos

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED ILUMNO

INTRODUCCIÓN

En la actualidad las computadoras, se utilizan ampliamente en muchas áreas de los negocios, la industria, la ciencia, el hogar y la educación.

Las computadoras se han desarrollado y mejorado, según las necesidades del hombre, para realizar trabajos y cálculos más rápidos y precisos.

En este referente de pensamiento se pretende mostrar al estudiante cómo, a partir de unas especificaciones de un problema del mundo real, se diseña una solución para dicho problema (algoritmo) susceptible de ser codificada en un lenguaje de programación.

Con este objetivo en mente, se describirán las propiedades básicas de cualquier algoritmo, un conjunto de bloques básicos que permiten la construcción de algoritmos y diversas formas de representación de los algoritmos. En el tema también se mostrarán las distintas fases que se deben de seguir para buscar una solución a un problema del mundo real. En este tema no se aborda ningún lenguaje de programación particular; sino que se muestra cómo diseñar soluciones a problemas que sean fáciles de implementar en cualquier lenguaje de programación.

Antes de continuar, vamos a ver en la página principal del eje, la videocápsula 1 *Introducción a la computación* para contextualizar de manera general.

Evolución y arquitectura del computador

Para comenzar, veamos una línea de tiempo al respecto de la evolución y arquitectura del computador:

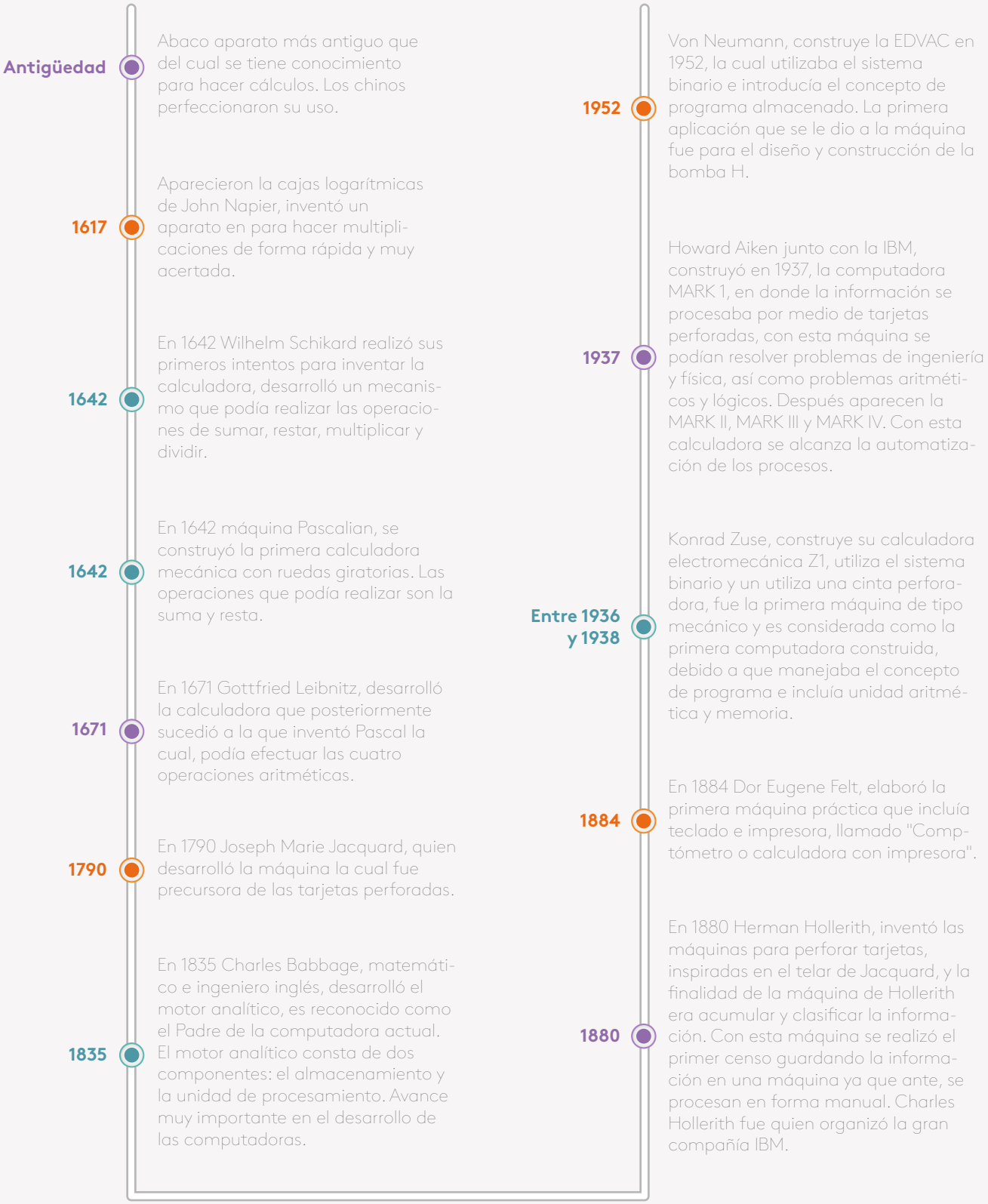


Figura 1. Línea de tiempo
Fuente: propia

Arquitectura del computador

Introducción a la arquitectura de computadores

La arquitectura de computadoras se refiere al diseño conceptual y la estructura operacional fundamental de un sistema que conforma una computadora, es decir, la disciplina dedicada a la construcción, estudio y aplicación de los computadores recibe el nombre de arquitectura de computadores y puede ser dividida en cinco partes fundamentales:

1. Entrada y salida.
2. Comunicaciones.
3. Control.
4. Procesamiento.
5. Almacenamiento.

Esta es de interés tanto para los ingenieros en electrónica y computación, dedicados al diseño de hardware, como para los científicos en computación e ingenieros de software, dedicados al diseño de programas.

Asimismo, la arquitectura de computadores es un concepto que integra software, hardware, algoritmos y lenguajes de programación para el procesamiento de datos y la generación de información.

Conceptos iniciales de la arquitectura de computadores

Un computador es un sistema secuencial síncrono complejo que procesa información, esta se trata de información binaria, utilizando solamente los dígitos de valores lógicos '1' y '0'. Estos valores lógicos binarios se corresponden con valores de tensión eléctrica, de manera que un '1' lógico corresponde a un nivel alto a 5 voltios y un '0' lógico corresponde a un nivel bajo de tensión cercano a 0 voltios; estos voltajes dependen de la tecnología que utilicen los dispositivos del computador.

Síncrono
Los circuitos secuenciales síncronos solo permiten un cambio de estado en los instantes marcados o autorizados por una señal de sincronismo de tipo oscilatorio denominada reloj.

Procesador

Figura 2. Procesador.
Fuente: http://bit.ly/2hBO4TS

Es el cerebro del sistema, encargado de procesar todos los datos e informaciones. A pesar de que es un dispositivo muy sofisticado no puede llegar a hacer nada por sí solo. Para hacer funcionar a este necesitamos algunos componentes más como lo son memorias, unidades de disco, dispositivos de entrada/salida y los programas. El procesador o núcleo central está formado por millones de transistores y componentes electrónicos de un tamaño microscópico. El procesamiento de las tareas o eventos que este realiza va en función de los nanosegundos, haciendo que los miles de transistores que contiene este trabajen en el orden de los MHz. La información binaria se introduce mediante dispositivos periféricos que sirven de interfaz entre el mundo exterior con el usuario. Estos periféricos lo que van a hacer será traducir la información que el usuario introduce en señales eléctricas, que serán interpretadas como unos y ceros, los cuales son interpretados de una manera más rápida por la computadora, ya que el lenguaje máquina utiliza el código binario para ser interpretado por el computador.

Arquitectura clásica de un computador modelo Von Neumann

La máquina Von Neumann, como todos los computadores modernos de uso general, consta de cuatro componentes principales:

1. **Dispositivo de operación (DO)**, que ejecuta instrucciones de un conjunto especificado, llamado sistema (conjunto) de instrucciones, sobre porciones de información almacenada, separada de la memoria del dispositivo operativo (aunque en la arquitectura moderna el dispositivo operativo consume más memoria -generalmente del banco de registros-), en la que los operandos son almacenados directamente en el proceso de cálculo, en un tiempo relativamente corto.

2. **Unidad de control (UC)**, que organiza la implementación consistente de algoritmos de decodificación de instrucciones que provienen de la memoria del dispositivo, responde a situaciones de emergencia y realiza funciones de dirección general de todos los nodos de computación. Por lo general, el DO y la UC conforman una estructura llamada CPU. Cabe señalar que el requisito es consistente, el orden de la memoria (el orden del cambio de dirección en el contador de programa) es fundamental a la hora de la ejecución de la instrucción. Por lo general, la arquitectura que no se adhiere a este principio no se considera von Neumann.

3. **Memoria del dispositivo,** un conjunto de celdas con identificadores únicos (direcciones), que contienen instrucciones y datos.

4. **Dispositivo de E/S (DES)**, que permite la comunicación con el mundo exterior de los computadores, son otros dispositivos que reciben los resultados y que le transmiten la información al computador para su procesamiento.

En el esquema de la figura 3, se muestra la estructura básica propuesta por Von Neumann que debe llevar una computadora para su correcta operación:

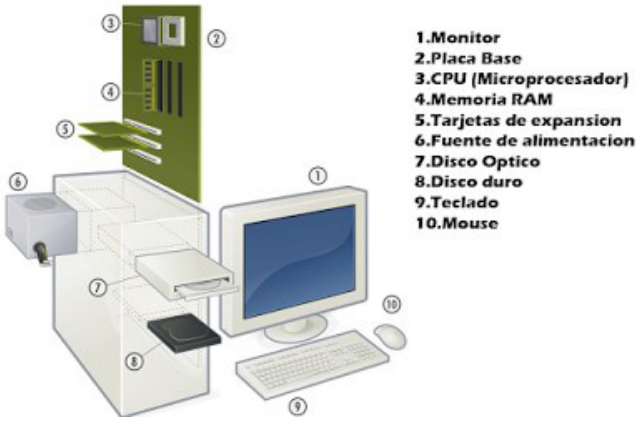


Figura 3. Estructura básica de una computadora.
Fuente: http://bit.ly/2kloZeY

La **unidad central de procesamiento** o **unidad de procesamiento central** (conocida por las siglas **CPU**, del inglés: Central Processing Unit), es el hardware dentro del computador u otros dispositivos programables, que interpreta las instrucciones de un programa mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema. El término, y su acrónimo, ha estado en uso en la industria de la Informática por lo menos desde el principio de los años 1960. La forma, el diseño de CPU y la implementación de las CPU ha cambiado drásticamente desde los primeros ejemplos, pero su operación fundamental sigue siendo la misma.

Memoria: es la responsable del almacenamiento de datos.

Entrada/Salida: transfiere datos entre el entorno exterior y el computador. En él se encuentran los controladores de periféricos que forman la interfaz entre los periféricos, la memoria y el procesador.

Sistema de interconexión: buses; es el mecanismo que permite el flujo de datos entre la CPU, la memoria y los módulos de entrada/salida. Aquí se propagan las señales eléctricas que son interpretadas como unos y ceros lógicos.

Periféricos: estos dispositivos son los que permiten la entrada de datos al computador, y la salida de información una vez procesada. Un grupo de periféricos puede entenderse como un conjunto de transductores entre la información física externa y la información binaria interpretable por el computador. Ejemplos de estos dispositivos son el teclado, el monitor, el ratón, el disco duro y las tarjetas de red.

Unidad central de procesamiento

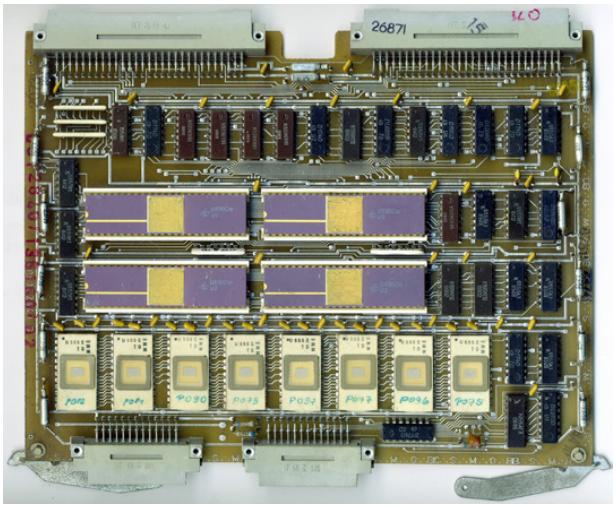


Figura 4. Unidad central de procesamiento.
Fuente: http://bit.ly/2gwSOKV

Controla el funcionamiento de los elementos de un computador. Desde que el sistema es alimentado por una corriente, este no deja de procesar información hasta que se corta dicha alimentación. La CPU es la parte más importante del procesador, debido a que es utilizado para realizar todas las operaciones y cálculos del computador.

Unidad de Control (UC): la unidad de control se encarga de leer de la memoria las instrucciones que debe de ejecutar y de secuenciar el acceso a los datos y operaciones a realizar por la unidad de proceso. La UC genera las señales de control que establecen el flujo de datos en todo el computador e interno en la CPU. Una instrucción no es más que una combinación de unos y ceros. Consta de un código de operaciones binarias para ejecutar la instrucción, la UC la almacena en un registro especial, interpreta su código de operación y ejecuta la secuencia de acciones adecuada, en pocas palabras decodifica la instrucción.

Operaciones binarias
Son aquellas en las cuales el resultado de verdadero o falso, 1 o 0.

Unidad Aritmética Lógica o ALU (por su sigla en inglés Arithmetic Logic Unit): es la parte de la CPU encargada de realizar las transformaciones de los datos. Gobernada por la UC, la ALU consta de una serie de módulos que realizan operaciones aritméticas y lógicas. La UC se encarga de seleccionar la operación a realizar habilitando los caminos de datos entre los diversos operadores de la ALU y entre los registros internos.

Registros internos: el almacenamiento de los resultados a la ejecución de las instrucciones en la memoria principal podría ser lento y excesivamente tendría muchos datos en el sistema de interconexión con la memoria, con lo que el rendimiento bajaría. De la misma manera, también se almacenan en registros internos la configuración interna del CPU o la información durante la última operación de la ALU. Los principales registros de una CPU son:

1. **Contador de programa:** se encarga de almacenar la dirección de la siguiente instrucción a ejecutar.

2. **Registro de instrucción:** se almacena la instrucción capturado en memoria y la que se está ejecutando.

3. **Registro de estado:** compuesto por una serie de bits que informan el resultado obtenido en la última operación de la ALU.

4. **Registro acumulador:** algunas CPU realizan operaciones aritméticas en un registro llamado acumulador, su función es la de almacenar los resultados de las operaciones aritméticas y lógicas.

Memoria

En la memoria se almacena el programa y los datos que va a ejecutar el CPU. Las instrucciones son códigos binarios interpretados por la unidad de control, los datos de igual manera se almacenan de forma binaria.

Las diversas tecnologías de almacenamiento dependen del tiempo de acceso a los datos; por lo tanto, se realiza un diseño jerárquico de la memoria del sistema para que esta pueda acceder rápidamente a los datos. El principio de que sea más rápida la memoria haciendo que tenga velocidades similares al CPU, sirve para diseñar el sistema de memoria. La memoria principal de los computadores tiene una estructura similar a la mostrada en el esquema de la Figura 4. Se considera como una matriz de celdas en la que la memoria puede acceder a los datos aleatoriamente.

Entrada/Salida

Como sabemos una computadora tiene dispositivos de entrada y salida como son los que contiene el gabinete, disco duro, placa madre, unidades de CD o DVD, etc. El problema principal que existe entre ellos es su tecnología y que tienen características diferentes a los del CPU, estos también necesitan una interfaz de cómo se van a entender con el CPU, al igual que el procesador y el controlador periférico para intercambiar datos entre la computadora.

Sistema de interconexión: buses

La conexión de los diversos componentes de una computadora, tales como discos duros, tarjetas madres, unidades de CD, teclados, ratones, etc. se efectúan a través de los buses. Un bus se define como un enlace de comunicación compartido que usa múltiples cables para conectar subsistemas. Cada línea es capaz de transmitir una tensión eléctrica que representa un '1' o un '0'. Cuando hay varios dispositivos en el mismo bus, habrá uno que podrá enviar una señal que será procesada por los demás módulos. Si se mandan los datos al mismo tiempo marcará un error o una contención del bus, por lo que el acceso estará denegado. Según su criterio de funcionalidad los buses se dividen en:

Buses de datos: es el que se utiliza para transmitir datos entre los diferentes dispositivos del computador.

Buses de direcciones: sirve para indicar la posición del dato que se requiere acceder.

Bus de control: sirven para seleccionar al emisor y al receptor en una transacción del bus.

Bus de alimentación: sirve para proporcionar a los dispositivos voltajes distintos.

Periféricos: se entenderán todos aquellos dispositivos que son necesarios para suministrar datos a la computadora o visualizar los resultados. Los periféricos se conectan mediante un bus especial a su controlador o al módulo de E/S.

Entre los periféricos de entrada tenemos al teclado, ratones, pantallas, digitalizadoras y más. Otros dispositivos periféricos fundamentales para la interacción del hombre con la computadora son las terminales de vídeo y las tarjetas gráficas.

A este punto, los invitamos a revisar la presentación de Jorge Chico sobre *Introducción a los computadores*.

¡Muy bien! Desarrollemos el crucigrama en la página principal del eje para afianzar los términos requeridos para la conceptualización de algoritmos.

Algoritmo, lenguaje y programa

Adicional a lo anterior, es importante tener presente tres conceptos:

Algoritmo: es una sucesión finita de pasos exento de ambigüedades que se pueden ejecutar en tiempo finito cuya razón de ser, es la resolución de problemas, cuya solución es expresable mediante un algoritmo.

Lenguaje: de programación es el responsable de que la computadora siga paso a paso las órdenes que el programador ha diseñado en el algoritmo. Con esto se entiende que el lenguaje de programación es una especie de intermediario entre el computador y el usuario, para que este último pueda darles respuesta a los problemas mediante la computadora y haciendo uso de palabras (funciones), que le interpretan dicho programa al computador para la realización de este trabajo.

Veamos en la página principal del eje, el video sobre *Lenguajes de programación* de Juan Francisco Díaz.

Programa: indicar a la computadora qué es lo que tiene que hacer. En otras palabras:

- Secuencia de instrucciones.
- Instrucciones que entiende la computadora.
- Y que persiguen un objetivo: ¡Resolver un problema!

Antes de continuar, vamos a consultar la nube de palabras en la página principal del eje para conceptualizar con algunos términos adicionales que será importante tener presentes.

Continuemos con el desglose de lenguajes.

Lenguajes de programación de alto nivel

Los lenguajes de programación de alto nivel se caracterizan por expresar los algoritmos de una manera adecuada a la capacidad de una persona para procesar información, en lugar de estar orientados a su ejecución en las máquinas.

Ventajas

- Genera un código más sencillo y comprensible.
- Escribir un código válido para diversas máquinas o sistemas operativos.
- Permite crear programas complejos en relativamente menos líneas de código.
- Más cercanos a los lenguajes natural y matemático.

resultado = dato1 + dato2;

- Mayor legibilidad, mayor facilidad de codificación
- Estructuración de datos / abstracción procedimental

Traducción
 	
Compiladores:
Compilan y enlazan
Programas completos
	Código fuente
	
	COMPILADOR
		#include <iostream>
		using namespace std;
		int main()
		{
		cout << "Hola Mundo!" << endl;
		return 0;
		}

Intérpretes: Compilan, enlazan y ejecutan instrucción a instrucción	Código objeto Enlazador	0100010100111010011100... Código objeto de biblioteca
-	Programa ejecutable	Para una arquitectura concreta y un sistema operativo

Tabla 1. Lenguajes de programación de alto nivel.
Fuente: propia

Genealogía de los lenguajes

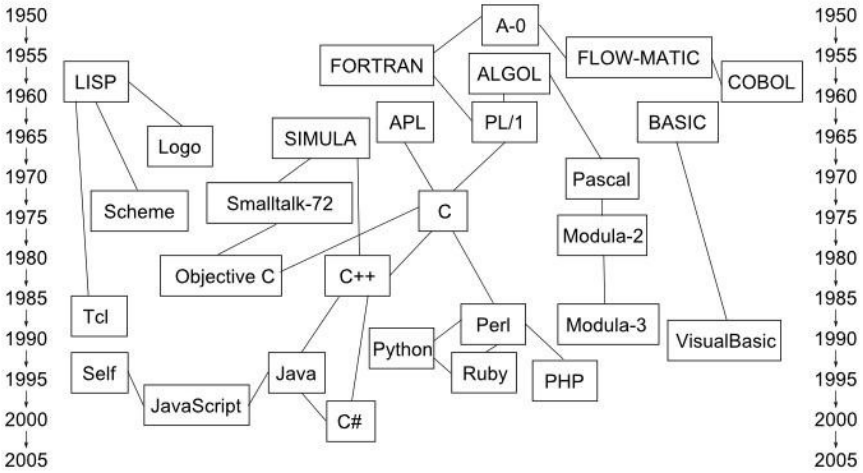


Figura 5. Genealogía de los lenguajes
Fuente: Sintes (2016).

Ahora examinaremos brevemente los grupos principales de lenguajes.

Lenguajes de programación imperativos y funcionales

Los lenguajes de programación generalmente se dividen en dos grupos principales con base al procesamiento de sus comandos: lenguajes imperativos y lenguajes funcionales.

Lenguaje de programación imperativo: un lenguaje imperativo programa mediante una serie de comandos agrupados en bloques y compuestos de órdenes condicionales, que permiten al programa retornar a un bloque de comandos si se cumple la condición. Estos fueron los primeros lenguajes de programación en uso y aún hoy muchos lenguajes modernos usan este principio.

No obstante, los lenguajes imperativos estructurados carecen de flexibilidad debido a la secuencialidad de las instrucciones.

Lenguaje de programación funcional: un lenguaje de programación funcional (a menudo llamado lenguaje procedimental) es un lenguaje que crea programas mediante funciones, devuelve un nuevo estado de resultado y recibe como entrada el resultado de otras funciones. Cuando una función se invoca a sí misma, hablamos de recursividad.

El programa: codificación del algoritmo en un lenguaje de programación. El código fuente es escrito en un lenguaje de programación.

```
1   ' Globales -----------
2   Var Variable0:Booleano
3   Var Variable1:Cadena
4   ' Fin Globales -------
5   Proc Procedimiento ' <- Procedimiento sin retorno.
6      Var Variable2:Entero ' Locales
7      Var Variable3:Real
8
9      Si Variable0 = Falso Entonces ' Condición "If"
10        Contar Variable2 = 0 a 9 ' Bucle "For"
11           Variable1 = Variable1 + "1"
12        Seguir ' "End For"
13     FinSi ' "End If"
14
15     Variable3 = 5.13
16  FinProc
```

Figura 6. Lenguaje de programación
Fuente: http://bit.ly/2yZTNKp

Instrucciones en una computadora

Una instrucción es cada paso de un algoritmo, pero que lo ejecuta la computadora.

Tipos de instrucciones

- Entrada y Salida. E/S: pasar información del exterior al interior del ordenador y al revés.
- Aritmético-lógicas: aritméticas: ^,/,mod,+,-,*; Lógicas: or, and, not. Comparación <, <=, >, >=, =, <>.
- Selectivas: permiten la selección de una alternativa en función de una condición.
- Repetitivas: repetición de un número de instrucciones un número finito de veces.

¡Recordemos que!

Un programa es un conjunto de instrucciones que ejecutadas ordenadamente resuelven un problema.

Tipos de lenguajes

Lenguaje máquina: todo se programa con 1 y 0, que es lo único que entiende el ordenador.

A continuación, encontrará una breve lista de los lenguajes de programación actuales:

Lenguaje	Principal área de aplicación	Compilado/interpretado
ADA	Tiempo real	Lenguaje compilado
BASIC	Programación para fines educativos	Lenguaje interpretado
C	Programación de sistema	Lenguaje compilado
C++	Programación de sistema orientado a objeto	Lenguaje compilado
Cobol	Administración	Lenguaje compilado
Fortran	Cálculo	Lenguaje compilado
Java	Programación orientada a Internet	Lenguaje intermediario
MATLAB	Cálculos matemáticos	Lenguaje interpretado
Cálculos matemáticos	Cálculos matemáticos	Lenguaje interpretado
LISP	Inteligencia artificial	Lenguaje intermediario
Pascal	Educación	Lenguaje compilado
PHP	Desarrollo de sitios web dinámicos	Lenguaje interpretado
Inteligencia artificial	Inteligencia artificial	Lenguaje interpretado
Perl	Procesamiento de cadenas de caracteres	Lenguaje interpretado

Tabla 2. Lenguajes de programación usados en la actualidad
Fuente: propia

Compilación del lenguaje

En un lenguaje compilado el código fuente antes de ser ejecutado es convertido a lenguaje máquina (C, C++) aunque también puede ser convertido a representación intermedia que posteriormente es interpretada y convertida a lenguaje máquina JIT (Java, C#). El compilador puede detectar una gran cantidad de errores que en un lenguaje interpretado o de tipado dinámico se descubrirán en tiempo de ejecución.

Lectura recomendada

Se invita al estudiante a realizar la lectura recomendada de *algoritmos*.

Algoritmos

Les invitamos a revisar las instrucciones de la actividad de repaso 1 e ir contrastando con el conocimiento sobre algoritmos que desarrollaremos a continuación.

Para empezar, veamos el video.

Video

¿Qué es un algoritmo?

Un algoritmo es una secuencia de pasos organizados entre sí que describe el proceso que se debe seguir, sin ambigüedades para dar solución a un problema específico.

Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX, Mohámed ben Musa, que alcanzó gran reputación al enunciar paso a paso las reglas para sumar, restar, multiplicar y dividir números con decimales.

La resolución de un problema mediante una computadora consiste en la especificación del problema, construir un programa que lo resuelva.

Los procesos necesarios para la creación de un programa son:

Figura 7. Proceso de creación de un programa
Fuente: propia

Algunos ejemplos de algoritmos son:

- Una receta de cocina.
- Un manual de uso.
- Las instrucciones para realizar un trámite.

Ahora bien, las características fundamentales que debe tener todo algoritmo son:

- Debe ser preciso, es decir, indicar el orden de realización de cada paso.
- Debe estar definido, esto es, si se ejecuta varias veces partiendo de las mismas condiciones iniciales debe obtenerse siempre el mismo resultado.
- Debe ser finito (debe tener un número finito de pasos).
- Debe ser independiente del lenguaje de programación que se emplee para implementarlo.

Metodología y construcción de algoritmos

En cualquier algoritmo se pueden distinguir tres partes:

1. La entrada de datos (la información sobre la cual se va a efectuar operaciones).
2. Procesamiento.
3. Salida del resultado (la información que debe proporcionar).

Veamos un ejemplo clásico de algoritmos: determinar el volumen de una caja de dimensiones A, B y C.

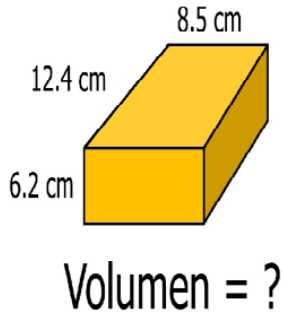


Figura 8. Volumen de una caja
Fuente: propia

Se puede establecer de la siguiente manera:

1. Inicio.
2. Leer las medidas A, B y C.
3. Realizar el producto de A * B * C y guardarlo en V. (V = A * B * C).
4. Escribir el resultado V.
5. Fin.

Como se puede apreciar, se establece de forma precisa la secuencia de los pasos por realizar; además, si se les proporciona siempre los mismos valores a las variables A, B y C, el resultado del volumen será el mismo y, por consiguiente, se cuenta con un final.

Diagramas de flujo

Los diagramas de flujo son una herramienta que permite representar visualmente qué operaciones se requieren y en qué secuencia se deben efectuar para solucionar un problema dado.

Diagramas de flujo
Un diagrama de flujo es la representación gráfica mediante símbolos especiales, de los pasos o procedimientos de manera secuencial y lógica que se deben realizar para solucionar un problema dado.

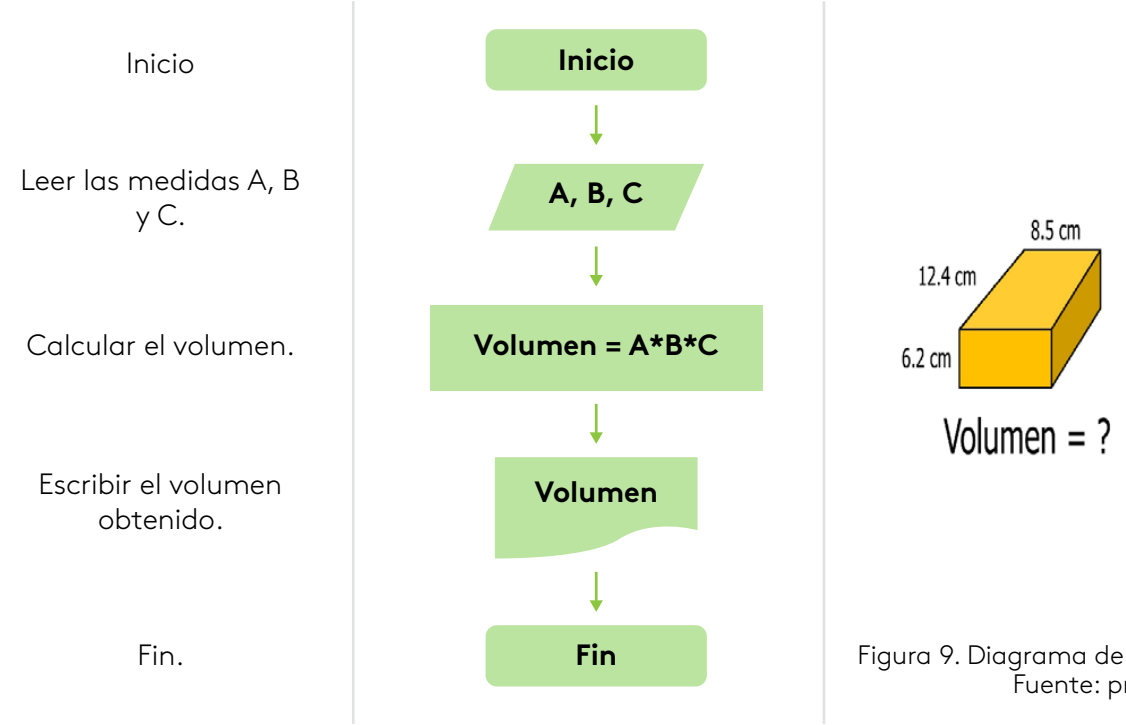


Figura 9. Diagrama de flujo
Fuente: propia

En principio, el flujo de ejecución de un programa será el orden en el cual se escriban las instrucciones (ejecución secuencial). Sin embargo, este esquema de ejecución impone serias limitaciones: a menudo hay ciertas porciones de código que sólo se deben ejecutar si se cumplen ciertas condiciones y tareas repetitivas que hay que ejecutar un gran número de veces. Por ello, se han definido una serie de estructuras de programación, independientes del lenguaje concreto que se está utilizando, que permiten crear programas cuyo flujo de ejecución sea más versátil que una ejecución secuencial.

Variables y tipos de datos

Veamos a continuación, los elementos que hacen parte de un algoritmo de programación.

Variables

Una variable es un objeto cuyo valor puede cambiar durante el desarrollo del algoritmo. Se identifica por su nombre y por su tipo, que podrá ser cualquiera, y es el que determina el conjunto de valores que podrá tomar la variable. En los algoritmos se pueden declarar variables. Cuando se traduce el algoritmo a un lenguaje de programación y se ejecuta el programa resultante, la declaración de cada una de las variables originará que se reserve un determinado espacio de memoria etiquetado con el correspondiente identificador.

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, ...

```
1570
+4591
-12
```

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

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
```

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
```

Los dos últimos valores del ejemplo son iguales, respectivamente, a 6.02×10236.02 ×1023 y 9.1094×10−319.1094×10−31.

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:

```
"ALGORITMOS Y PROGRAMACIÓN"
'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.

Revise en la página principal del eje el video.

Video

Cómo se realiza un algoritmo.

Constantes

Son datos cuyo valor no cambia durante todo el desarrollo del algoritmo. Las constantes podrán ser literales o con nombres.

Ahora bien, ya que identificamos los tipos de datos que se utilizan en el diseño de un algoritmo, a continuación, vamos a ver aquellas estructuras de tipo repetitivo que son vitales para aquellos problemas típicos que requieren ejecutar una instrucción a partir de una determinada condición que se debe cumplir. Veamos:

La estructura de control if

La sentencia if permite a un programa tomar una decisión para ejecutar una acción u otra. Diariamente tomamos decisiones, por ejemplo, evaluamos el estado del tiempo para decidir si llevamos sombrilla, verificamos si la tarjeta de Transmilenio tiene cargado dinero para viajar, sino hay que cargarle dinero.

A La sentencia if se le conoce como estructura de selección simple y su función es realizar o no una determinada acción o sentencia, basándose en el resultado de la evaluación de una expresión (verdadero o falso), en caso de ser verdadero se ejecuta la sentencia.

```
if ( expresión(es ) )
      { sen tencias  }
else
      { sentencias }
```

Por ejemplo**,** si dada la edad de una persona quiero dar un mensaje de que es o no mayor de edad, suponiendo que una persona mayor de edad tiene por lo menos 21 años, el procedimiento será el siguiente.

En pseudocódigo el anterior ejemplo se vería de la siguiente forma:

```
Inicio
Mostrar "¿Qué edad tienes?"

Leer edad

Si Edad >20 Entonces
Mostrar "Eres mayor de Edad"
Fin Si
Fin
```

La sentencia for es una sentencia que implementa un bucle, es decir, que es capaz de repetir un grupo de sentencias un número determinado de veces.

La sintaxis de un ciclo for es simple en los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3

componentes del ciclo **for** (inicio, final y tamaño de paso)

```
for (int i = valor inicial; i <= valor final; i = i + paso)
{
        ....
        ....
Bloque de Instrucciones....
        ....
        ....
}
```

Pseudocódigo
El pseudocódigo consta de un falso lenguaje. Se considera que, es un código informal que se utiliza como inicio de un algoritmo.

Sentencia While

Veamos la definición de esta sentencia, tomada de Ceballos (2007).

La sentencia while ejecuta una secuencia, simple o compuesta, cero o más veces, dependiendo de una condición:

Su sintaxis es:

```
while (condición)
        Bloque de instrucciones;
```

Si se requiere realizar más de un estatuto se deben utilizar llaves.

```
while ( condición )
    {
        Bloque de Instrucciones;
    }
```

Donde condición es cualquier expresión numérica, relacional o lógica y bloque de instrucciones es una sentencia simple o compuesta.

La ejecución de la sentencia while funciona de la siguiente forma:

1. Se evalúa la condición. El resultado es falso o verdadero.
2. Si el resultado de la evaluación es falso (0), no se ejecuta y se pasa el control a la siguiente sentencia en el programa.
3. Si el resultado de la evaluación es verdadero (1), se ejecuta la sentencia y el proceso descrito se repite desde el punto 1.
4. Un ciclo while tiene una condición del ciclo, una expresión lógica que controla la secuencia de repetición. (Ceballos, 2007).

Implementación de algoritmos

Diagramas de flujo (DFD)

¿Qué es DFD?: los diagramas de flujo de datos (DFD en inglés) son un tipo de herramienta de modelado, permiten modelar todo tipo de sistemas, concentrándose en las funciones que realiza, y los datos de entrada y salida de esas funciones.

Componentes de los DFD

PROCESOS (burbujas): representan la parte del sistema que transforma ciertas entradas en ciertas salidas.

FLUJOS: representan los datos en movimiento. Pueden ser flujos de entrada o flujos de salida. Los flujos conectan procesos entre sí y también almacenes con procesos.

ALMACENES: representan datos almacenados. Pueden ser una base de datos, un archivo físico.

TERMINADORES: representan entidades externas que se comunican con el sistema. Esas entidades pueden ser personas, organizaciones u otros sistemas, pero no pertenecen al sistema que se está modelando.

Flujos de control

Los flujos de control son señales o interrupciones, en tanto los procesos de control son burbujas que coordinan y sincronizan otros procesos; los procesos de control solo se conectan con flujos de control.
Los flujos de control de salida "despiertan" otras burbujas, en tanto los flujos de control de entrada, especifican que una tarea terminó o se presentó un evento extraordinario.

¡Importante!

Existen procesos y flujos especiales llamados procesos de control y flujos de control. Se emplean para modelar sistemas en tiempo real.

Representación de un sistema en DFD

Un sistema puede representarse empleando varios diagramas de flujos de datos, cada flujo de datos puede representar una parte "más pequeña" del sistema.

Los DFD permiten una partición por niveles del sistema. El nivel más general se representa con un DFD global llamado diagrama de contexto.

El diagrama de contexto DFD representa a todo el sistema con una simple burbuja o proceso, las entradas y salidas de todo el sistema, y las interacciones con los terminadores.

Complementos del DFD

Los DFD suelen servir para comprender fácilmente el funcionamiento de un sistema. De todas maneras, no es la única herramienta para diagramar sistemas, es más, se debe complementar con otras herramientas para agregar comprensión y exactitud al DFD.

Para la implementación de algoritmos es ideal instalar en el computador programas como PseInt.

¿Qué es PseInt?: es una herramienta para asistir a los estudiantes en sus primeros pasos en programación. Mediante un simple e intuitivo pseudolenguaje en español (complementado con un editor de diagramas de flujo), le permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos.

Se puede descargar de forma libre ingresando a Google:

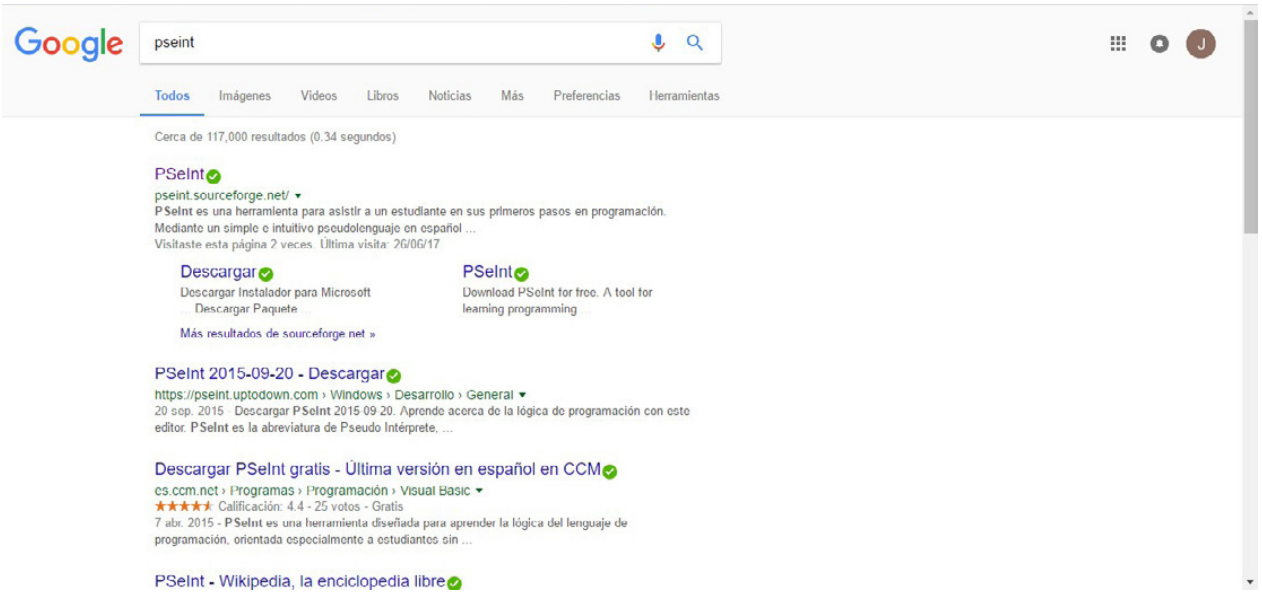


Figura 10. Búsqueda en google de descarga de PseInt
Fuente: captura de pantalla "Búsqueda de Google"

¡Importante!

Se invita al estudiante a revisar el manual PseInt del Ing. Ronald Rentería, con el fin de consolidar el concepto y desarrollo de los algoritmos. Además, será importante que se realice su instalación para diseñar los respectivos diagramas de flujo que se proponen en el módulo.

Introducción a software para interpretación e implementación (C++)

Si se desea escribir un programa en C++ se debe ejecutar como mínimo los siguientes pasos:

1. Escribir con un editor de texto plano un programa sintácticamente válido o usar un entorno de desarrollo (IDE) apropiado para tal fin.
2. Compilar el programa y asegurarse de que no ha habido errores de compilación.
3. Ejecutar el programa y comprobar que no hay errores de ejecución.

¡Importante!

Este último paso es el más costoso, porque en programas grandes, averiguar si hay o no un fallo prácticamente puede ser una tarea titánica.

Como ejemplo, si se desea escribir un archivo con el nombre hola.cpp y en él escribir un programa con emacs, por ejemplo, que es un programa de edición de textos, se puede, en GNU, ejecutar el siguiente comando:

```
$emacs hola.cpp &
```

Para otros sistemas operativos u otros entornos de desarrollo, no necesariamente se sigue este paso.

```
#include <iostream>

int main(void)

{

std::cout << "Hola Mundo" << std::endl;

std::cin.get();

//con 'std::cin.get();' lo que se hace es esperar hasta que el usuario pulse enter.

return 0;

}
```

En este punto se ha terminado este eje, por lo que es el momento de realizar los ejercicios propuestos en Actividad de repaso 2.

BIBLIOGRAFÍA

Ceballos, F. (2007). *Programación orientada a objetos con C++* (4a. ed.). Madrid, España: Editorial RA-MA. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11045993&p00=lenguaje+c%2B%2B

Ceballos, F. (2009). *Enciclopedia del lenguaje C++* (2a. ed.). Madrid, España: Editorial RA-MA Recuperado de: http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046572&p00=lenguaje+c%2B%2B

Gaxiola, C., y Flores, D. (2008). *Metodología de la programación con pseudocódigo enfocado al lenguaje C*. Ciudad de México, México: Editorial Plaza y Valdés, S.A. de C.V. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10877093&p00=algoritmos+programaci%C3%B3n

Joyanes, L. (2006). *Programación en C++: algoritmos, estructuras de datos y objetos* (2a. ed.) Madrid, España: McGraw-Hill. Recuperado de: http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491359&p00=lenguaje+c%2B%2B

Joyanes, L., Castillo, A., y Sánchez, L. (2005). *C algoritmos, programación y estructuras de datos*. Madrid, España: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491350&p00=lenguaje+c%2B%2B

Joyanes, L., Rodríguez, L., y Fernández, M. (2003). *Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos* (2a. ed.) Madrid, España: McGraw-Hill España Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498607&p00=algoritmos+programaci%C3%B3n

Joyanes, L., y Sánchez, L. (2006). *Programación en C++: un enfoque práctico*. Madrid, España: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491298&p00=lenguaje+c%2B%2B

Joyanes, L., y Zahonero, I. (2007). *Estructura de datos en C++*. Madrid, España: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491301&p00=lenguaje+c%2B%2B

BIBLIOGRAFÍA

Joyanes, L., Zahonero, I. (2005). *Programación en C: metodología, algoritmos y estructura de datos* (2a. ed.). Madrid, España: McGraw-Hill Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498360&p00=algoritmos+programaci%C3%B3n

Juganaru, M. (2014). *Introducción a la programación*. Ciudad de México, México: Grupo Editorial Patria. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11017472&p00=programaci%C3%B3n

Moreno, J. (2014). *Programación*. Madrid, España: Editorial RA-MA. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046398&p00=lenguaje+c%2B%2B

Noguera, F., y Riera, D. (2013). *Programación*. Barcelona, España: Editorial UOC. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10853420&p00=lenguaje+c%2B%2B

Schildt, H. (2009). *C++: soluciones de programación*. McGraw-Hill Interamericana. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10433927&p00=algoritmos+programaci%C3%B3n

ALGORITMOS Y PROGRAMACIÓN

Julio César Rodríguez Casas

EJE 2

Analicemos la situación

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED
ILUMNO

INTRODUCCIÓN

En el presente referente abordaremos los temas relacionados con el lenguaje C++, partiendo desde una breve reseña acerca de la historia del lenguaje C++, teniendo en cuenta los conceptos básicos de C++; así mismo, veremos cómo se puede instalar el compilador del lenguaje C++ con el propósito de desarrollar programas y compilarlos y así mismo corregir los posibles errores de sintaxis que se puedan presentar.

De igual forma, se ofrece la opción de ejecutar los programas online, con una URL para compilar, sin la necesidad de tener instalado el compilador en el computador.

Adicionalmente se realiza la explicación línea por línea del primer programa en C++ para entender cada una de las instrucciones e irse familiarizando con el código.

Para empezar, los invitamos a descargar y tener a mano las siguientes lecturas:

Lectura recomendada

Para reforzar los temas que veremos más adelante se invita al estudiante realizar la lectura complementaria Programación en C++/Lo más básico.

De igual forma se facilita el Manual de programación de C++.

¡Comencemos!

Las principales herramientas necesarias para escribir un programa en C++ son las siguientes:

Un compilador de C++.

Paciencia.

- Nociones sobre programación.
- Un editor cualquiera de texto.

Programación básica
Lenguaje C++
Historia C++

Inicialmente el lenguaje C se realizó en los laboratorios Bell de AT&T en los años 1969 y 1973, en un comienzo se le nombró como "C", la mayoría de sus características se tomaron de un lenguaje llamado "B".

¡Importante!

En 1983 se formó el comité ANSI (Instituto Nacional Americano de Estándares) con el objeto de crear un estándar. Este proceso tuvo una duración de aproximadamente de seis años y a principios de los años 90 es estándar fue reconocido como ISO (Organización Internacional de Estándares) así mismo se comienza a comercializar con el nombre ANSI C.

En forma paralela en 1980 aparece el lenguaje C++, en los laboratorios Bell de AT&T, con el señor Bjarne Strouptrup quien realiza el diseño del lenguaje C++ con el fin de adicionar nuevas características al lenguaje C, como son clases y funciones virtuales tomadas de SIMULA 67, tipos genéricos y expresiones tomadas de lenguaje ADA. Así mismo de ALGOL 68 se tomo la declaración de variables en cualquier parte del programa.

C++ comenzó su proyecto de estandarización ante el comité ANSI y su primera referencia es The Annotated C++ Reference Manual [Ellis 89]. En diciembre de 1989 se reunió el comité X3J16 del ANSI por iniciativa de Hewlett Packard.

Con base en el auge y éxito del lenguaje en el año 1990, se reunieron las organizaciones ISO y ANSI para concretar y definir un estándar con el fin de formalizar el lenguaje. En 1998 termina con la aprobación del ANSI C++.

En junio de 1991, a la estandarización de ANSI se unió ISO (International Organization for Standardization) con su propio comité (ISO-WG-21), creando un esfuerzo común ANSI/ISO para desarrollar un estándar para C++.).

C es un lenguaje de propósito general que se puede utilizar para escribir cualquier tipo de programa, pero su éxito y popularidad está especialmente relacionado con el sistema operativo UNIX. (Fue desarrollado como lenguaje de programación de sistemas, es decir, un lenguaje de programación para escribir **sistemas operativos** y utilidades (programas) del sistema.)

Sistemas operativos
Los sistemas operativos son los programas que gestionan (administran) los recursos de la computadora. Ejemplos bien conocidos de sistemas operativos además de UNIX son MS/DOS, OS/2, MVS, Linux, Windows 95/98, Windows NT, Windows 2000, OS Mac, etc.

La especificación formal del lenguaje C es un documento escrito por Ritchie, titulado *The C Reference Manual*. En 1997, Ritchie y Brian Kernighan, ampliaron ese documento y publicaron un libro referencia del lenguaje *The C Programming Language* también conocido por el K&R.

En la actualidad el lenguaje C++ se ha vuelto muy popular y en se utiliza mucho en las universidades, además es un lenguaje orientado a objetos, de igual forma se puede utilizar como un lenguaje estructurado como su antecesor el lenguaje C y si se quiere trabajar con algoritmos y estructuras de datos.

Instrucción

A este punto, le invitamos a consultar el capítulo 1 "Introducción a la ciencia de la computación y a la programación" de Joyanes, específicamente de la página 38 a 43 para ahondar en la evolución de C++.

Conceptos básicos del lenguaje C++

Funciones

El lenguaje C++ está basado en el concepto de funciones en donde cada una tiene un nombre y una lista de argumentos. En general se puede dar a una función el nombre que se quiera con excepción de MAIN que se reserva para la función que inicia la ejecución del programa.

Una función es un proceso con entradas y salidas bien definidas. Su implementación práctica también debe ir encaminada a la realización de bloques bien definidos en cuanto al proceso que realicen y los insumos y consumos que requiera. Las funciones reciben datos a través de una lista, y devuelve determinada información de un tipo específico. A la lista de datos que la función recibe, se le conoce con el nombre de lista de parámetros.

Las funciones son una de las herramientas más útiles en la programación porque permiten encerrar código con un nombre e invocarlo a través de él. Con esto los programadores evitan repetir el código cada vez que sea necesario en un programa, además de que, es posible ocultar código de forma que el usuario de la función no tenga que conocer los detalles del cómo se hacen las cosas.

Biblioteca Estándar de C++

En C++, la biblioteca estándar es una colección de Clases y funciones, escritas en el núcleo del lenguaje. La biblioteca estándar proporciona varios contenedores genéricos, funciones para utilizar y manipular esos contenedores, funciones objeto, cadenas y flujos genéricos (incluyendo E/S interactiva y de archivos) y soporte para la mayoría de las características del lenguaje. La biblioteca estándar de C++ también incorpora la ISO C90 biblioteca estándar de C. Las características de la biblioteca estándar están declaradas en el espacio de nombres (namespace) std.

La Standard Template Library es un subconjunto de la biblioteca estándar de C++ que contiene los contenedores, algoritmos, iteradores, funciones objeto, etc.; aunque algunas personas utilizan el término STL indistintamente con la biblioteca estándar de C++.

Para profundizar en el tema de programación, se recomienda realizar la lectura y la actividad de las lecturas complementarias; para que pueda instalar y compilar los ejercicios de lenguaje C++.

Otra alternativa para compilar los programas a través de la web en línea es JDoogle. Se recomienda habilitar el modo interactivo.

Conceptos básicos de un programa en C++

Analicemos el siguiente código:

```
#include <iostream>
using namespace std;

int main() {

// aquí escribiremos el código del programa . . .
cout << "Hola Areandina"; << endln;
}
```

Si ejecutamos el programa anterior, no presenta ningún error y nos mostrará el mensaje Hola Areandina.

Estructura del código fuente

Todo el código que escribamos irá siempre dentro de las llaves que tenemos en la función int main () { ...(código)... }.

Vamos ahora a analizar por encima el programa anterior:

Línea 1: #include <iostream>

Las líneas que empiezan por el carácter almohadilla (#), son leídas por lo que se conoce como el "preprocesador". estas son líneas especiales que se ponen antes del código del programa en sí, También se llaman "cabeceras". En este caso #include <iostream>, encarga al preprocesador que se incluya una sección de código C++ estándar que permite realizar operaciones estándar de entrada y salida de datos. Las secciones que se incluyan mediante la almohadilla (#) se llaman también bibliotecas o librerías.

Línea 2: using namespace std;

Video

Antes de continuar, veamos el video sobre Librerías e instrucciones básicas.

Esta línea declara un "namespace" o espacio de nombres. En concreto el namespace "std" o estándar. Dentro del namespace std están declarados todos los elementos de las bibliotecas estándar. Mediante esta línea podemos usar los elementos de la biblioteca estándar sin necesidad de tener que declararlos cada vez que se usan.

Línea 3:

Esta es una línea en blanco, no es imprescindible ponerla, pero ayuda a una mayor claridad en la comprensión del código para el programador. Veremos más adelante el tratamiento de líneas en blanco y espacios en la programación.

Líneas 4 y 7: int main() {

}

Esta estructura es la función principal que contiene el código (main significa "principal"). Lo que se escriba dentro de las llaves de esta función es el código fuente en sí, es decir el programa que se ejecutará. Es posible que pueda haber instrucciones fuera de esta función, pero siempre tendrán que estar dirigidas desde esta función.

Más adelante entraremos a explicar más en profundidad lo que son las funciones, las librerías y otros conceptos que aquí hemos dicho de pasada. De momento hay que saber que para programar en C++ lo primero que debemos hacer es escribir la plantilla que damos aquí. En nuestro ejemplo, dentro de la función int main() { } hemos puesto las siguientes líneas:

Línea 5: Este es un comentario no lo interpreta el compilador.

// aquí escribiremos el código del programa ...

Dentro de las llaves de la función int main() escribiremos el código propiamente dicho, es decir, lo que queremos que se ejecute en el programa.

En este caso hemos puesto un comentario. La doble//barra del principio de línea indica que este es un comentario introducido por el programador. Este no tiene ningún efecto en la programación. Se usan para introducir explicaciones u observacio-

nes. son útiles para documentar el programa e indicar cómo funciona.

Línea 6: cout << "Hola Areandina" << endl;

Esta línea es una sentencia C ++. una sentencia es una instrucción que produce algún efecto o cambia algo en el programa. Las sentencias se ejecutan en el mismo orden en que aparecen. A las sentencias también se les llama declaraciones.

¡Importante!

La sentencia o declaración en un lenguaje de programación, es el equivalente a la frase en el lenguaje humano.

Esta sentencia tiene varias partes:

- En primer lugar, cout, que identifica el dispositivo de salida de caracteres estándar (por lo general la pantalla del ordenador).
- En segundo lugar, el operador de inserción (<<), que indica que lo que sigue debe insertarse (mostrarse en pantalla).
- Por último, una frase entre comillas ("Hola Areandina"), que es el contenido insertado.

Estas dos últimas partes se repiten para volver a insertar otro elemento, es decir repetimos el operador de inserción (**<<**) y después indicamos lo que debe insertarse, en este caso es un salto de línea que se indica mediante la instrucción endl.

Observamos que la declaración termina con un punto y coma **(;)**. Este carácter marca el final de la declaración. Todas las declaraciones o sentencias en C ++ deben terminar con un punto y coma.

Comentarios

Línea 5: Este es un comentario no lo interpreta el compilador.

// aquí escribiremos el código del programa ...

Como hemos visto en el código anterior, los comentarios son anotaciones que hace el programador para sí mismo. El comentario no tiene ningún efecto en la programación, por lo tanto, y a efectos de programación, es ignorado, y el compilador no lo lee.

Sin embargo, puede resultar útil para el programador, como nota aclaratoria que indica lo que se está haciendo.

De una línea

Hay dos tipos de comentarios, el primero, que ya hemos visto, es el comentario de una sola línea.

Para insertarlo escribimos una doble barra, y todo lo que se escriba después de ella en la misma línea es un comentario.

Por ejemplo:

```
// Este es un comentario de una línea.
```

El comentario de una línea no tiene por qué empezar al principio de línea, sino que puede ir en la misma línea después de una declaración, por ejemplo:

```
cout << "Hola Areandina" << endl;// Hola mundo en pantalla.
```

De varias líneas

El segundo tipo de comentario es el de varias líneas, este empieza por los caracteres /*, (barra, asterisco), escribimos después el comentario en una o varias líneas, y para indicar que ha acabado escribimos después los caracteres */ (asterisco, barra). por ejemplo:

```
/* abrimos el comentario.
Sigue el comentario en esta línea...
otra línea más del comentario...
...cerramos el comentario */
```

Todo lo que escribamos entre los signos /* y */, será un comentario de varias líneas y será ignorado por el compilador.

Espacios en blanco

Al escribir el código escribimos una serie de palabras clave, declaraciones, etc.

Como hemos visto en los códigos que hemos hecho hasta ahora, se pueden poner líneas en blanco, más de un espacio entre palabras, o tabulaciones, sin que esto afecte al código.

De hecho, podríamos poner todo el código seguido, si exceptuamos algunas instrucciones que deben ir en una sola línea. Estas son: las que empiezan por #include, y los comentarios de una sola línea. Por ejemplo, podríamos poner el programa del "Hola Areandina" así:

```
//Hola mundo ...
#include <iostream>
using namespace std; int main()
{cout << "Hola Areandina" << endl;}
```

Sin embargo, para una mayor claridad en el código, se suelen utilizar líneas en blanco, tabulaciones, y otros espacios en blanco. De esta manera indicamos qué elementos son principales, y cuales dependen de otros, y los cambios entre las distintas partes del programa. Por ejemplo, el código anterior, bien organizado, quedará así:

```
//Hola mundo
#include <iostream>
using namespace std;

int main() {
  cout << "Hola Areandina" << endl;
}
```

Estructura de un programa C++

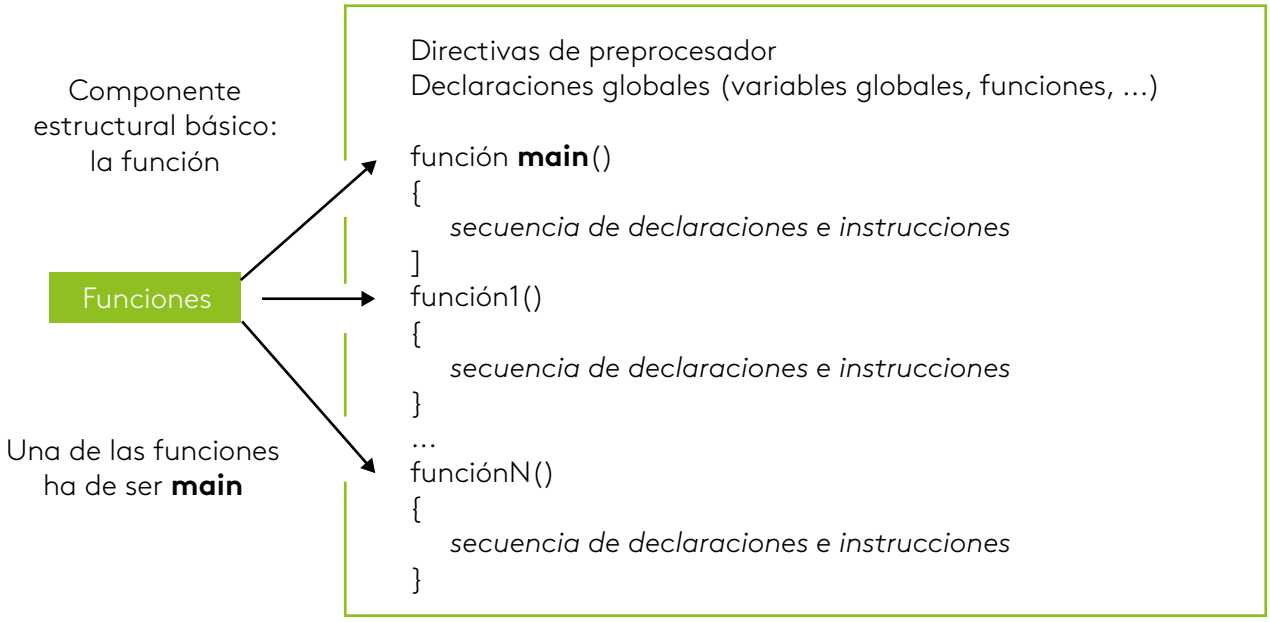


Figura 1. Estructura básica de un programa en lenguaje C++
Tomado de: https://www.yumpu.com/es/document/view/14837159/1-estructura-basica-de-un-programa-c-

return 0;

Todas las funciones deben retornar un valor. Y la función main(), retornará el valor cero (0) si no hubo ningún error. Las funciones deberán retornar el valor especificado al declararlas "**int main ()**" significa que la función **main** devolverá un número entero (a los números enteros se les denomina **int** por "integer" que significa entero en inglés).

Lectura recomendada

Para profundizar en el tema se invita al estudiante a realizar la lectura complementaria acerca de la salida de datos.

Video

Se invita al estudiante a que vea el video acerca de cómo se realiza un programa con base en el algoritmo.

Identificadores y palabras reservadas en el lenguaje C++

Las palabras reservadas son identificadores predefinidos que tienen significados especiales y no pueden usarse como identificadores creados por el usuario en los programas.

Las palabras reservadas de C++ pueden agruparse en tres (3) grupos.

- **Primer grupo:** contiene las palabras de C y que C++ como evolución de C también contiene:

auto	const	double	Break	switch
short	struct	unsigned	Signed	goto
else	For	long	Enum	char
void	case	default	volatile	static
register	sizeof	typedef	Return	
do	extern	If	Int	
union	while	float	continue	

Tabla 1. Primer grupo de palabras reservadas para C++
Fuente: propia

- **Segundo grupo:** contiene palabras que no provienen de C y que, por tanto, solo utiliza C++:

Asm	namespace	try	explicit
new	typeid	false	template
typename	friend	this	const_cast
public	virtual	mutable	true
dynamic_cast	reinterpret_cast	bool	inline
static_cast	catch	operator	wchar_t
class	private	using	
throw	delete	protected	

Tabla 2. Segundo grupo de palabras reservadas para C++
Fuente: propia

- **Tercer grupo:** las siguientes palabras reservadas se han añadido como alternativas para algunos operadores de C++ y hacen los programas más legibles y fáciles de escribir:

not_eq	or_eq	xor_eq
and_eq bitor	not	or
xor		

Tabla 3. Tercer grupo de palabras reservadas para C++
Fuente: propia

Tipos de datos y sus formatos

Los datos en C++

El primer objetivo de un computador es el manejo de información a partir de datos. Estos datos pueden ser costos, calificaciones, temperaturas, presupuestos, datos personales, signos vitales, velocidad, entre otros.

¡Importante!

Los distintos tipos de datos son representados en la memoria del computador de acuerdo con el tipo y el lenguaje de programación que use. Los números enteros en C++, por ejemplo, miden 16 bits o 2 bytes (bit=dígito binario).

El mínimo número que se puede escribir en 16 bits equivale a 16 ceros (0) que al ser convertidos a decimal representan precisamente el valor 0 (cero). El máximo número que se puede escribir en 16 bits son 16 unos que representan el número 65535 decimal. Teniendo en cuenta que hablamos de números sin signo.

La siguiente tabla indica los tipos de datos simple de C++.

TIPO	EJEMPLO	BYTES	RANGO
char	'C'	1	0 a 255
short	-15	2	-128 a 127
int	1024	2	-32768 a 32767
unsigned int	42345	2	0 a 65535
long	262144	4	-2147483648 a 2147483637
float	10.45	4	3.4*(10 e -38) a 3.4*(10 e 38)
double	0.000000000045	8	1.7*(10 e -308) a 1.7*(10 e 308)
long double	1,00E-08	8	Igual que double

Tabla 4. Tipos de datos simple de C++
Fuente: propia

Tipos de datos

Los tipos de datos que maneja C++ son:

- **Enteros:** dentro de los enteros están los tipos: short, int, long, los cuales varían en rango de acuerdo al compilador que se utilice, siendo long rango mayor y short el de menor.

- **Flotantes:** dentro de los flotantes C++ tiene los tipos: float, double y long double donde al igual que los enteros varía el rango de cada uno de acuerdo al compilador que se utilice. De igual forma el float es el de menor rango siendo long double el de rango mayor

- **Caracteres:** se utiliza el tipo char. Para representar un carácter en C++ se utilizan comillas.

  Ejemplos: 'a', 'b' , '05'

  Para representar una cadena de caracteres se utilizan las comillas.

  Ejemplo: "Areandina", "INGENIERÍA DE SISTEMAS"

Declaración de variables

Las variables son elementos clave en todo lenguaje de programación. Se deben declarar diciendo el tipo de dato, el nombre y si es necesario se inicializa a un valor. El compilador separa un espacio en la memoria física del computador para manipular la variable declarada por el usuario

¡Importante!

La declaración de una variable es un estatuto que proporciona información de la variable al compilador de C++.

La sintaxis para la declaración de una variable es:

tipo variable

- tipo: es el nombre de un tipo de dato conocido por C++.
- variable: es un identificador (nombre) válido en C++.

Ejemplo: Declaración de una variable de tipo entero llamada x e inicializada en 100;

```
int  x = 100 ;
```

Veamos un ejemplo de declaración de constante y variables:

```
#include <iostream>
int main( ) {
 const float pi=3.141592;
int radio=5;
float area;
área = pi * radio * radio;
cout << "El área del circulo es: "<< área
<< endl;
return 0; }
```

Expresiones lógicas

Las expresiones lógicas son todas aquellas expresiones que obtienen como resultado verdadero o falso.

Estos operadores unen estas expresiones devolviendo también verdadero o falso. Son:

Operador	Significado
&&	Y (AND)
||	O (OR)
!	NO (NOT)

Tabla 5. Expresiones lógicas
Fuente: propia

Por ejemplo: (18>6) **&&** (20<30) devuelve verdadero (1) ya que la primera expresión (18>6) es verdadera y la segunda (20<30) también.

El operador **Y (&&)** devuelve verdadero cuando las dos expresiones son verdaderas.

El operador **O (||)** devuelve verdadero cuando cualquiera de las dos es verdadero.

Finalmente, el operador NO (!) invierte la lógica de la expresión que le sigue; si la expresión siguiente es verdadera devuelve falso y viceversa.

Por ejemplo !(18>15) devuelve falso (0).

Operadores aritméticos

Los operadores aritméticos se usan para realizar cálculos de aritmética de números reales y de aritmética de punteros. C++ dispone de los siguientes:

Operación	Operador aritmético	Operador en C++
Suma	+	+
Resta	-	-
Multiplicación	X	*
División		/
Módulo	Mod	%

Tabla 6. Operadores aritméticos
Fuente: propia

Video

Antes de continuar, les invitamos a ver el video sobre operadores aritméticos y lógicos.

Operaciones 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.

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

Tabla 7. Operadores relacionales
Fuente: propia

Funciones de entrada y salida

La mayoría de los programas en C++ incluyen el archivo de encabezado **<iostream>,** el cual contiene la información básica requerida para todas las operaciones de entrada y salida (E/S) de flujo.

Cuando usamos la instrucción:

```
Cout <<"Mensaje a la pantalla"<< endl;
```

Estamos enviando una cadena de caracteres ("Mensaje a la pantalla") al dispositivo de salida estándar (la pantalla). Luego, el manipulador de flujo endl da el efecto de la secuencia de escape '\n', nueva línea.

Se invita al estudiante para que pruebe el siguiente programa:

```
#include <iostream>
int main() {
 cout <<"Hola estudiante Areandina" <<endl;
 cout << 2 + 2 <<endl;//imprime un entero
cout << 9/2 <<endl;//imprime un flotante
cout<<(int) (3.141592+2)<<endl;//imprime un entero
return 0;
}
```

La instrucción **cout <<** puede imprimir tanto números enteros como flotantes sin necesidad de decirle específicamente el tipo de datos del que se trata, pero, por supuesto que al enviarle una cadena de caracteres esta debe de estar entre comillas.

¡Importante!

La interacción con el usuario es algo muy importante en la programación, imaginemos que en este preciso momento y con los conocimientos que tenemos hasta ahora, necesitamos hacer un programa que calcule la distancia a la que caerá un proyectil lanzado a determinada velocidad y ángulo, o simplemente un programa que calcule las raíces de una ecuación cuadrática. Sería muy molesto estar cambiando los valores de las variables directamente en el código para cada caso que queramos calcular. Por eso debemos ver cuanto antes la forma de leer datos desde el teclado.

La principal función para leer desde el teclado es **cin >>**, pero es mejor ver un ejemplo para tener la idea clara.

```
#include <iostream>
using namespace std;
int main() {
   int numero;
   char car;
   float otroNum;
    cout << "Escribe un numero:"<<endl;
    cin >>numero;
    cout <<"\nEl número que ingresó es: «<<numero<<endl;
    cout<<"Digite una letra"<<endl;
    cin>>car;
    cout<<"\nLa letra que ingresó es: "<<car<<endl;
    cout<<"Digite un número de tipo flotante"<<endl;
     cin>>otroNum;
      cout<<"\El número que digitó es: "<<otroNum;
return 0;
}
```

En resumen, **cin** es el flujo de entrada asociado al teclado, **cout** es el flujo de salida estándar asociado a la pantalla, y existe otro, que, aunque a lo largo de este trabajo casi no lo utilicemos, es necesario nombrarlo, cerrar, que es el flujo de error estándar asociado a la pantalla.

¡Importante!

Los operadores << , >> son operadores de inserción y extracción de flujo respectivamente, y no deben confundirse con los de desplazamiento de bits. Estos operadores son muy eficaces porque no es necesario especificar formatos para presentación o lectura, ellos los presentan en función al tipo de datos de la variable. Aunque en ocasiones podría necesitar de nuestra ayuda para obtener los resultados específicos que queremos, y para eso están los modificadores de formato.

Menús, funciones y arreglos

Estructuras de control anidadas

En lenguajes de programación, las **estructuras de control** permiten modificar el flujo de ejecución de las instrucciones de un programa. Con las **estructuras de control** se puede de acuerdo con una condición, ejecutar un grupo u otro de sentencias (If-Then-Else) Si – Entonces – de lo contrario.

Video

Veamos un video sobre estructura Condicional If-Else (Si - De lo contrario)

En nuestra vida cotidiana, todos tenemos una lógica a seguir, continuamente tomamos decisiones, y estas decisiones repercuten en nuestra acción siguiente. Por ejemplo, supongamos el caso de un estudiante de nivel preparatoria que cursa el sexto semestre, él está pensando en presentar el examen para la universidad, sin embargo, sus calificaciones no le han dado mucho aliento últimamente, y está en riesgo de tener que repetir ese semestre, si ocurre eso, el resultado que tenga en el examen no importará. Lo que vaya a pasar marcará el camino a seguir en su vida.

```
#include <iostream>
using namespace std;
int main(){
   float  nota;
   cout<<"Cuál fue tu nota de Algoritmos y Programación?" << endl;
   cin>> nota;

   if (nota >= 3.0)
     cout << "Felicitaciones Aprobó";
   else
     cout << "Lo lamento, debe repetir la metería";

   cin.ignore();
   cin.get();
   return 0;

}
```

- **Instrucción switch, case y break**

Sentencia switch

Se trata de una sentencia que permite construir alternativas múltiples, cuando la estructura condicional no resulta muy cómoda de utilizar. Cuando el programa encuentra un case que es igual al valor de la expresión se ejecutan todos los ***case*** siguientes. Por eso se utiliza la sentencia break para hacer que el programa abandone el bloque switch.

Veamos la sintaxis:

Switch

```
#include <iostream>
using namespace std;
int main() {
int dia;
cout<<"Digite un número de la semana: ";
cin >> dia;
switch(dia)
{ case 1:
  cout <<"Lunes " <<endl;
  break;
  case 2:
  cout <<"Martes" <<endl;
  break;
  case 3:
  cout <<"Miércoles" <<endl;
  break;
  case 4:
  cout <<"Jueves " <<endl;
  break;
  case 5:
  cout <<"viernes" <<endl;
  break;
  case 6:
  cout <<"Sábado" <<endl;
  break;
  case 7:
  cout <<"Domingo" <<endl;
  break;
  default:
  cout<<"Dia de la semana no corresponde"<<endl;
  break;
  }
   return 0;
  }
```

While

Los ciclos while son también una estructura cíclica, que nos permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuándo se va a dar el valor final que esperamos, los ciclos while, no dependen directamente de valores numéricos, sino de valores booleanos, es decir su ejecución depende del valor de verdad de una condición dada, verdadera o falso, nada más. De este modo los ciclos while, son mucho más efectivos para condiciones indeterminadas, que no conocemos cuándo se van a dar a diferencia de los ciclos for, con los cuales se debe tener claro un principio, un final y un tamaño de paso.

```
#include “iostream”
using namespace std;
int main(){
int i = 0;
while(i <= 10)
{
cout << “Soy la variable i, mi valor en esta iteración es: “ << i <<endl;
i++;
}
   return 0;
}
```

Funciones

Definición de una función

Cuando tratamos de resolver un problema, resulta muy útil utilizar la filosofía de “divide y vencerás”. Esta estrategia consiste en dividir nuestro problema en otros más sencillos.

¡Importante!

Cuando realizamos un programa, por ejemplo, en el que se repetirán varias instrucciones, pero con distintos valores que definan los resultados, podemos construir el programa con base en funciones.

Una función es un bloque de instrucciones a las que se les asigna un nombre. Entonces, cada que necesitemos que se ejecuten esa serie de instrucciones, haremos una invocación a la función.

Prototipos de funciones

El prototipo de una función se refiere a la información contenida en la declaración de una función. Una función debe de estar definida o al menos declarada antes de hacer uso de ella.

Cuando se declara una función debe de especificarse el tipo de dato que va a devolver, el nombre de la función, y los parámetros. La siguiente declaración: **int suma(int** a, **int** b);

Especifica una función que devuelve un tipo de dato entero, tiene por nombre suma, y al invocarla, recibe 2 parámetros de tipo entero.

Esta declaración debe de escribirse antes de la función **main**, y su definición puede escribirse después de esta.

Por ejemplo:

```
#include "iostream"
using namespace std;

  int suma(int x, int y);

  int main(){
     int numero1,numero2;
     cout << "CALCULAR LA SUMA DE 2 NUMEROS"<<endl;
     cout << "Ingrese el primer dato:  " <<endl;
     cin >> numero1;
     cout << "Ingrese el segundo dato:  " <<endl;
     cin >> numero2;
     cout << "La suma es: "<< suma(numero1,numero2) <<endl;

     return 0; }
     int suma(int a, int b)
     {
        return a+b;
```

Instrucción

Para finalizar, le invitamos a realizar la actividad de repaso 1, con el propósito de poner en práctica los conceptos y procedimientos que se enunciaron en el referente de pensamiento.

ALGORITMOS Y PROGRAMACIÓN

Julio César Rodríguez Casas

EJE 3

Pongamos en práctica

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED
ILUMNO

INTRODUCCIÓN

El referente de pensamiento de este eje está basado en las estructuras de control. Veremos los tipos de estructuras como los saltos de línea que consisten en forzar el salto de punto del programa a otro que no va en orden secuencial. Pueden hacerse saliendo de un bloque de sentencias antes de acabarlo, o saltando de un punto a otro del programa mediante ciertas instrucciones comenzando por la condicional if-else, la cual nos permite tomar cierta decisión al interior de nuestro algoritmo, se trabajarán con algunos ejemplos con los cuales podrá practicar y familiarizarse con la estructura.

De igual forma, se verá la estructura switch case, que es más práctica cuando se tienen muchas condiciones que evaluar, se explicara su sintaxis y se realizarán ejemplos prácticos.

Seguidamente se analizará la estructura While y Do while y se explica con ejemplos el comportamiento de la estructura.

Por último, se mostrará el ciclo for, cada una de sus partes y ejemplos prácticos para ejecutar en el compilador de lenguaje C++.

A la par de este desarrollo temático se realizarán actividades de aprendizaje que permitirán poner en práctica estas estructuras.

Antes de continuar le recomendamos realizar la lectura complementaria acerca de las estructuras de control.

Lectura recomendada

Programación en C++/Iteraciones y decisiones

bit.ly/28Pohla

Condicionales anidados

Para la ejecución de los programas se puede utilizar los compiladores disponibles en línea, se recomienda utilizar el compilador de JDoodle.

Compilador de JDoodle
http://bit.ly/2gsosJx

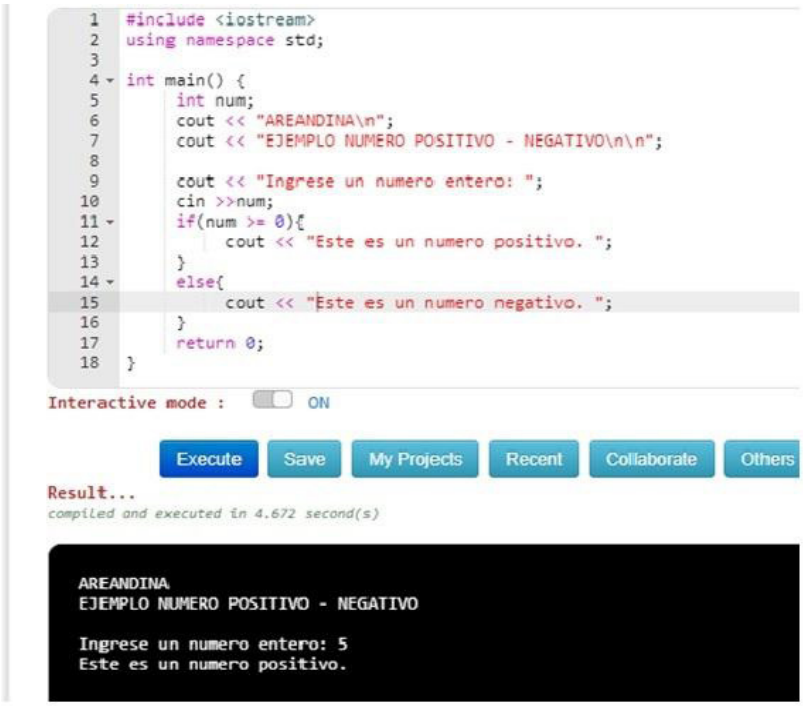


Figura 1. Compilador de Jdoodle
Fuente: Jdoodle

Se recomienda habilitar el modo interactivo (Interactive mode: ON), para interactuar con el programa; de igual forma, una vez se ejecuta el programa en la parte inferior se muestra el resultado de la compilación, bien sea errores de sintaxis o el resultado de la ejecución del programa.

Importante

Los programas presentados en el presente documento han sido escritos, aprobados y ejecutados en forma exitosa en el link anteriormente mencionado.

C++ usa todas las sentencias de control de ejecución de C. Esto incluye if-else, do-while, for, y una sentencia de selección llamada switch.

Sin embargo, lo primero que tenemos que hacer es agrupar varias sentencias en un bloque. para tratarlas todas como una unidad, a la que se le aplican unas determinadas órdenes. Para ello utilizaremos las llaves { ... } .

Dentro de las llaves escribiremos todas las sentencias que queramos agrupar, por ejemplo:

```
{
int a;
cout << "Tu edad : "; cin >> a;
cout << "tienes " << a << " años." << endl;
}
```

Estas sentencias forman un bloque al estar escritas dentro de las llaves.

La propia función int main () { ... } es un bloque, ya que todas las sentencias del programa deben escribirse dentro de las llaves.

Importante

La importancia de los bloques es fundamental para crear todo tipo de estructuras en donde queremos que afecten a más de una sentencia. Un bloque actúa de manera uniforme ante órdenes que se le den al mismo, como si fuera una única sentencia.

Tipos de estructuras

Una estructura cambia el flujo normal de ejecución del programa, de manera que este no se realice de una manera secuencial. Distinguimos entre las estructuras de control y otro tipo de estructuras como pueden ser las funciones o las clases.

Hay dos tipos de estructuras de control, que son las condicionales y los bucles. A esto se podría añadir un tercer tipo consistente en los saltos de línea, aunque este último es menos utilizado.

- Las condicionales consisten en que, llegado un punto, el programa deja de ser secuencial, y se bifurca, para elegir una opción entre dos (o entre varias) ignorando el resto, o pudiendo volver a ellas más tarde.

- Los bucles consisten en la repetición de determinadas tareas, La repetición no es exacta, en cada repetición hay alguna variable o algún elemento que cambia de manera controlada, así como una condición necesaria para que vuelva a repetirse la tarea. Cuando esta condición deja de existir la repetición se acaba y se sale del bucle.

Los saltos de línea consisten en forzar el salto de punto del programa a otro que no va en orden secuencial. Pueden hacerse saliendo de un bloque de sentencias antes de acabarlo, o saltando de un punto a otro del programa mediante ciertas instrucciones.

La estructura if

Dentro de las estructuras de control condicionales, la más importante es la estructura if. Esta consiste en que el programador indica una condición. Esta condición es evaluada como un dato booleano. Si el valor del dato es verdadero la condición se cumple, y, por lo tanto, la sentencia, o el bloque que va asociado a la condición, se ejecuta. Si el resultado de la condición es falso, la sentencia o bloque no se ejecuta.

Es decir, si la condición que hemos puesto se cumple, el programa realiza la acción, y si no es así no la realiza.

La sintaxis que se usa para esta estructura es la siguiente:

```
if (<condicion>) { <bloque de sentencias> }
```

Vamos cada uno de los componentes de esta estructura:

- if: es la palabra clave que indica el tipo de estructura. Esta palabra se pone siempre al principio.

- (<condicion>): entre paréntesis escribiremos a continuación una condición. Esta

tiene que dar un resultado booleano de “true” o “false”, por lo que en la mayoría de las veces suele ser una operación lógica o condicional, por ejemplo: (‘a’ > ‘b’).

- { <bloque de sentencias> }: después de la condición escribiremos el código al que queremos que afecte la estructura. este, si tiene más de una sentencia, debe ir en un bloque, (escrito entre llaves), ya que de otra manera solo afectaría a la primera sentencia.

Video

Se invita al estudiante a que vea el video acerca de cómo se realiza un algoritmo, en la página principal del eje.

Veamos un ejemplo de una estructura condicional con if:

```
#include<iostream>
using namespace std;
int main(){
    int a;
    cout << “Escribe un número mayor que 10 : “;
    cin >> a;
    if (a > 10) {
      cout << “Muy bien !! “ << endl;
      cout << “El numero “ << a << “ es mayor que 10.” << endl;
    }
}
```

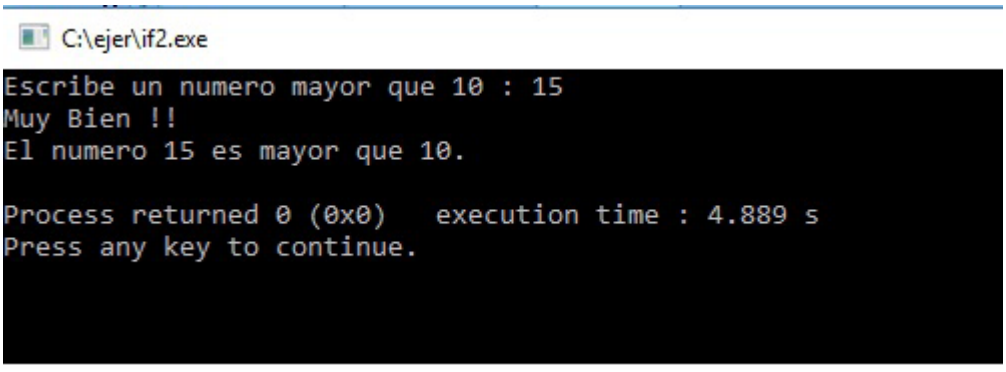


Figura 2. Resultado de la estructura condicional con if
Fuente: propia

Compilamos y ejecutamos el programa. Aquí pedimos al usuario que escriba un número mayor que 10. En la estructura condicional comprobamos que el número que nos ha dado el usuario sea mayor que 10, y solo en ese caso se ejecutará el código del bloque que está escrito entre llaves.

Observamos cómo se forma la estructura condicional. En primer lugar, la palabra clave if. Después de la condición entre paréntesis: (a > 10), y después la sentencia o bloque de sentencias.

```
    { cout << "Bien Hecho !! " << endl; cout << "El numero " << a << " es mayor que 10."
<< endl; }.
```

Aquí hemos puesto la estructura escrita en varias líneas, separando la condición, y tabulando las sentencias del bloque. Esto, no es imprescindible, ya que podríamos ponerlo todo seguido sin saltos de línea ni tabulaciones. por ejemplo:

```
  if (a > 10) { cout << "Muy Bien !! " << endl; cout << "El numero " << a << " es mayor que
10." << endl; }
```

Pero para una mayor claridad de los programadores, se suele poner siempre de la forma que lo hemos puesto antes:

```
  if (a > 10) {
   cout << "Muy Bien !! " << endl;
   cout << "El numero " << a << " es mayor que 10." << endl;
}
```

La palabra clave else

Si lo que queremos es que cuando la condición se cumpla se ejecute un código, y cuando no se cumpla se ejecute otro distinto, lo que debemos de hacer es añadir después de la estructura if la palabra clave else, seguida del bloque de sentencias que debe ejecutarse cuando la condición no se cumple. La estructura completa es:

```
  if (<condicion>) {
  <sentencias cuando SI se cumple>;
  }
else {
  <sentencias cuando NO se cumple>;
  }
```

Es decir, el programa evalúa la condición escrita después del if, y si el resultado es ver-

dadero se ejecuta el primer bloque de sentencias, pero si es falso se ejecutará el bloque que hay detrás del else.

Vamos a hacer lo mismo, pero con una estructura if ... else. El programa quedará así:

```
#include<iostream>

using namespace std;

int main(){

    int a;

    cout << "Areandina\n";

    cout << "Escribe un número mayor que 10 : ";

    cin >> a;

    if (a > 10) {

      cout << "Muy Bien  !! " << endl;

      cout << "El número " << a << " es mayor que 10." << endl;

    }

    else {

      cout << "Numero equivocado!! " << endl;

      cout << "El número " << a << " NO es mayor que 10." << endl;

    }

}
```

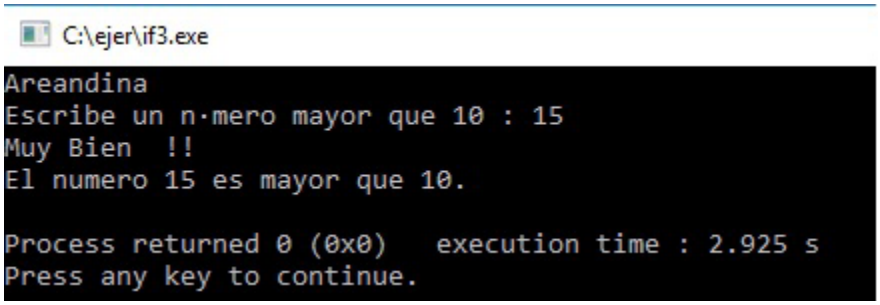


Figura 3. Resultado del ejemplo estructura if...else
Fuente: propia

Cuando el programa es ejecutado y evaluado por la parte falsa el resultado es el siguiente:

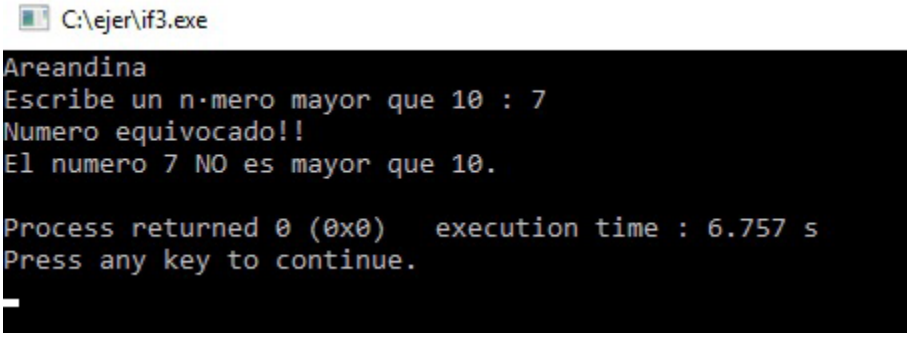


Figura 4. Resultado del ejemplo estructura if...else ejecutado
Fuente: propia

En este ejemplo pedimos dos números al usuario, el cual elige después la operación que va a realizar con ellos. Una vez elegida se realiza las operaciones aritméticas con esos dos números.

Veamos el código fuente:

```
#include<iostream>

using namespace std;

int main() {

    cout << "AREANDINA\n";

    cout << "OPERACIONES ARITMÉTICAS\n";

    cout << "OPERACIONES CON DOS NUMEROS" << endl;

    int a, b; char op;//Declaramos las variables

    cout << "Digite un numero: ";cin >> a;//pedimos el primer número

    cout << "Digite otro número: "; cin >> b;//pedimos el segundo número

    cout << "Los números son " << a << " y " << b << endl;//confirmamos números

     cout << "Elige una operación, para ello escribe su letra: " << endl;//explicación al usuario

    cout << "S = Suma; R = Resta; M = Multiplicación; D = División." << endl;
```

```
    cout << "Tu operación : "; cin >> op;//pedimos operación
    if ((op == 's') || (op =='S')) {//opción 1: Suma.
        cout << "Operación: Suma " << endl;
        cout << a << " + " << b << " = " << a+b << endl;
    }
    else if ((op == 'r') || (op =='R')) {//opción 2: Resta
        cout << "Operación: Resta " << endl;
        cout << a << " - " << b << " = " << a-b << endl;
    }
    else if ((op == 'm') || (op =='M')) {//opción 3: Multiplicación
        cout << "Operación: Multiplicación " << endl;
        cout << a << " * " << b << " = " << a*b << endl;
    }
    else if ((op == 'd') || (op =='D')) {//opción 4 División
        cout << "Operación: División " << endl;
        cout << a << " / " << b << " = " << a/b << endl;
    }
    else {//Respuesta para opción equivocada.
        cout << "La letra escrita no se corresponde con ninguna operación." << endl;
    }
}
```

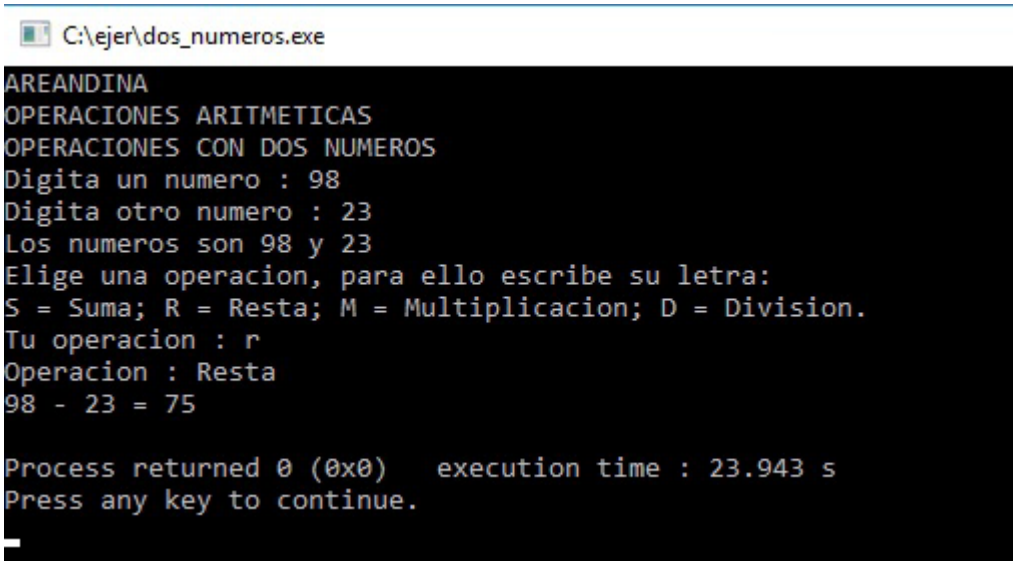


Figura 5. Ejemplo de resultado con operaciones aritméticas.
Fuente: propia

Video

Se invita al estudiante a que vea el video *Cómo se utiliza el switch case* en la página principal del eje.

Condicional switch

La estructura switch tiene también la misma función if, pero aquí todas las opciones dependen del valor de una variable.

Importante

Tenemos una variable, y cada opción consiste en comparar esta variable con otro elemento, de manera que, si el valor de la variable y el del elemento comparado son iguales, el resultado es verdadero, la condición se cumple, y se ejecuta el código asociado. Si no es así se pasa a la siguiente opción donde se vuelve a repetir el proceso. Si no se ejecuta ninguna opción puede ponerse una opción por defecto al final.

Video

Antes de continuar veamos el video con un ejemplo sencillo sobre la **sentencia switch.**

youtu.be/InDTfd4XIX8

Código fuente

El código de esta estructura es el siguiente:

```
    switch (x) {
  case a:
   <sentencias para x == a>;
   break;
  case b:
   <sentencias para x == b>;
   break;
   .....
  case n:
   <sentencias para x == n>;
   break;
  default:
   <sentencias para x == n>;
}
```

Vamos a explicar este código y ver cómo está estructurado:

- Empezamos poniendo la palabra clave switch, y después, entre paréntesis la variable con la que compararemos las demás (x).

- El resto del código forma un bloque y va escrito entre llaves. Este código se compone de varias opciones.

- Cada una de las opciones (excepto la última) empieza por la palabra clave case seguida del elemento con el que hacemos una comparación de igualdad y después dos puntos case a.

- Cada una de las opciones (excepto la última) acaba con la palabra clave break.

- En cada una de las opciones anteriores, entre el código del principio (case a:) y el del final (break;), escribiremos las sentencias que deben ejecutarse cuando la condición se cumpla.

- La opción final indica lo que debe ejecutarse en el caso en que ninguna de las anteriores se cumpla, y empieza por la palabra clave default: (acabada con dos puntos), seguida de las sentencias que deben ejecutarse cuando no se cumpla ninguna opción anterior.

Importante

Observa que podemos poner tantas opciones case .... break; como queramos. El case a: indica que iniciamos una nueva opción, la cual ejecutará el código que le sigue en el caso de que la variable principal (x) sea igual a la indicada a.

El break hace que el flujo se interrumpa y salgamos del bloque. De no ponerlo el programa no sale del bloque y el resto del código del bloque, perteneciente a otras opciones, también se ejecutaría.

La última opción default indica lo que se debe hacer cuando no hay ningún elemento anterior que coincida. Aunque no es obligatorio ponerla, es conveniente si queremos que el programa haga algo cuando no se cumplen las opciones anteriores. Al ser la última no es necesario poner el break; al final, ya que es ahí donde acabamos el bloque.

```
#include<iostream>

using namespace std;

int main() {

    cout << "AREANDINA\n";

    cout << "PROGRAMA SWITCH\n";

    cout << "OPERACIONES CON DOS NUMEROS" << endl;

    int a, b; char op;//Declaramos las variables

    cout << "Escribe un número : ";cin >> a;//pedimos el primer número

    cout << "Escribe otro número : "; cin >> b;//pedimos el segundo número

    cout << "Tus números son " << a << " y " << b << endl;//confirmamos números

    cout << "Elige una operación, para ello escribe su letra: " << endl;//explicación al
usuario

    cout << "s = Suma; r = Resta; m = Multiplicación; d = División." << endl;

    cout << "Tu operación: "; cin >> op;//pedimos operación

    switch (op){//inicio estructura.

      case 's': case 'S': case '+'://opcion 1: suma

        cout << "Operación : Suma " << endl;

        cout << a << " + " << b << " = " << a+b << endl;

        break;

      case 'r': case 'R': case '-'://opción 2: resta

        cout << "Operación: Resta " << endl;
```

```
            cout << a << " - " << b << " = " << a-b << endl;
            break;
        case 'm': case 'M': case '*'://opción 3: multiplicación
            cout << "Operación: Multiplicación " << endl;
            cout << a << " * " << b << " = " << a*b << endl;
            break;
        case 'd': case 'D': case '/'://opción 4 : división
            cout << "Operación: División " << endl;
            if (b != 0){
            cout << a << " / " << b << " = " << a/b << endl;
            }
            else {
              cout << "NO DIVIDIRÁS ENTRE CERO\n";
            }
            break;

        default: //Respuesta para opción equivocada.
            cout << "El carácter no se corresponde a ninguna operación." << endl;
    }
}
```

El siguiente es el resultado de la ejecución:

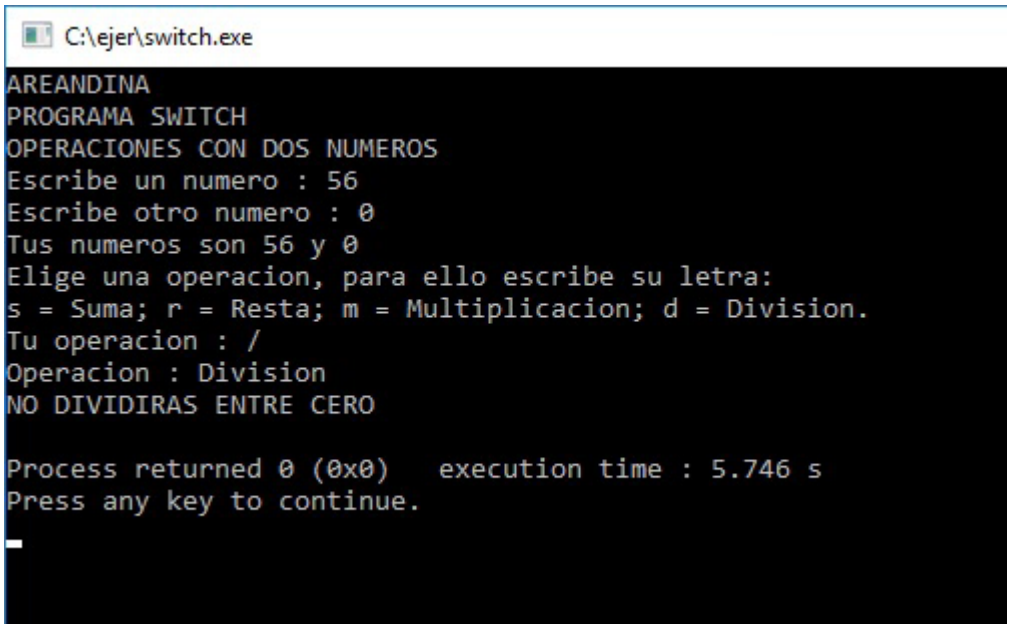


Figura 6. Resultado condicional switch
Fuente: propia

El bucle while

El tipo de bucle más sencillo es el bucle while. Para crear un bucle while tan solo se debe poner la palabra clave while seguida de la condición entre paréntesis, y después el bloque de sentencias.

```
while (<condicion>) {<Sentencias que se repiten>}
```

Sin embargo, la mayoría de las veces utilizaremos una operación condicional o lógica como condición. Por ejemplo, queremos que un determinado bloque se repita 10 veces. Para ello lo normal es utilizar una **variable de control**. Esta variable controlará el número de veces que se repite el bucle. Debemos declararla antes de empezar el bucle, y además inicializarla, normalmente con valor 0.

```
int i = 0;
```

Importante

Se utiliza una variable de tipo entero para controlar el número de vueltas (repeticiones) del bucle. Por convención suele utilizarse la letra i para definir esta variable.

Empezamos el bucle escribiendo la palabra clave while seguida de la condición entre paréntesis. Como queremos que el bucle se repita 10 veces, la condición será que la variable i se menor que 10:

```
while (i < 10)
```

Seguimos con el bloque de sentencias. Para que el bucle tenga un número definido de repeticiones, debemos variar la variable de control (i) de manera que cuente el número de repeticiones que llevamos en cada vuelta. Para ello, dentro del bloque de sentencias, debemos incrementar el valor de la variable de control una unidad en cada vuelta. Esto es lo que se llama "actualizar" la variable de control. El bloque de sentencias será como el siguiente:

```
{
<sentencias que se repiten>;
i = i+1;
}
```

Es decir, en cada vuelta el valor de i aumenta en una unidad. Cuando el valor de i sea igual a 10 la condición será falsa, y el bucle dejará de repetirse.

Veamos un ejemplo de programa utilizando while:

```
#include<iostream>
using namespace std;
int main() {
    cout << "AREANDINA\n";
    cout << "TABLA DE MULTIPLICAR\n";
    cout << "Tabla del nueve" << endl << endl;
    int i= 0;
    int tb = 9;
    while (i <=10) {
        cout << tb << " x " << i << " = " << tb * i << endl;
        i = i+1;
    }
}
```

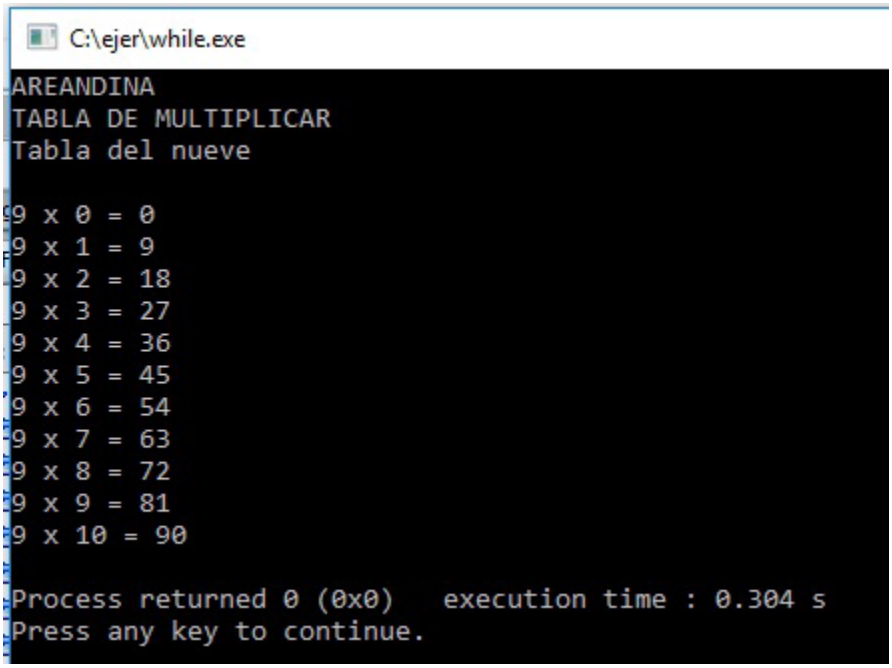


Figura 7. Ejemplo de programa utilizando while
Fuente: propia

Bucle do while

El bucle do while es muy parecido al bucle while. La diferencia es que aquí, aunque la condición no se cumpla, el código se ejecutará siempre al menos una vez.

Esto es debido a que primero se ejecuta el código, y después se comprueba la condición. Si esta es verdadera, el bucle se repite, si no es así se sale de él y continúa con el resto del programa.

Video

Antes de continuar, veamos en la página principal del eje, el video sobre **do while, cómo funciona, su sintaxis y un ejemplo básico.**

youtu.be/Z0kD_blq-Q4

Sintaxis de do while

La sintaxis es ligeramente diferente del bucle while, ya que aquí la condición se indica al final de la estructura:

```
   do {
  <Sentencias que se repiten>;
} while (<condicion>);
```

Empezamos poniendo la palabra clave do seguido del bloque de sentencias que se repiten. Después del bloque de sentencias ponemos la palabra clave while seguida de la condición entre paréntesis.

Tal como hemos indicado, al estar la condición al final del bucle, este se ejecuta siempre la primera vez, y la comprobación de la condición se hace al finalizar el bucle, con lo cual se decide si el bucle se repite o no.

Por lo demás es igual que el bucle while, es decir, si queremos repetir el bucle un número determinado de veces, necesitamos una variable de control (i).

Como con cualquier otro bucle, hay que tener en cuenta que tras un número de repeticiones tiene que llegar un momento en que la condición no se cumpla, para poder salir del bucle, y no caer en el error de hacer un bucle infinito.

Al igual que con el bucle while la variable de control la inicializamos antes del bucle, la utilizaremos también como referencia en la condición, y la actualizaremos en cada repetición del bucle. Por ejemplo, para hacer un bucle que se repita 10 veces haremos:

```
   int i=0;
do {
  <sentencias que se repiten>;
  i++;
} while (i < 10)
```

Veamos un ejemplo en el que empleamos el bucle do while. Aquí le pedimos al usuario un número y el programa repetirá una frase el número de veces que el usuario nos diga, pero si el número es cero o negativo, escribiremos igualmente la frase una vez. El archivo fuente será el siguiente:

```
        #include<iostream>

        using namespace std;

        int main() {
```

```
        int i=0, r;

        cout << "AREANDINA\n";

        cout << "PROGRAMA do while\n";

        cout << "Escribe un número para repetir la frase.";

        cin >> r;

         cout << "La frase se escribirá al menos una vez aunque el número sea cero o
negativo." << endl;

        do {

           i++;

           cout << i << ".- ALGORITMOS Y PROGRAMACIÓN. AREANDINA" << endl;

        } while(i < r);

      }
```

```
C:\ejer\do_while.exe
AREANDINA
PROGRAMA do while
Escribe un numero para repetir la frase.6
La frase se escribira al menos una vez aunque el numero sea cero o negativo.
1.- ALGORITMOS Y PROGRAMACION. AREANDINA
2.- ALGORITMOS Y PROGRAMACION. AREANDINA
3.- ALGORITMOS Y PROGRAMACION. AREANDINA
4.- ALGORITMOS Y PROGRAMACION. AREANDINA
5.- ALGORITMOS Y PROGRAMACION. AREANDINA
6.- ALGORITMOS Y PROGRAMACION. AREANDINA

Process returned 0 (0x0)   execution time : 6.327 s
Press any key to continue.
```

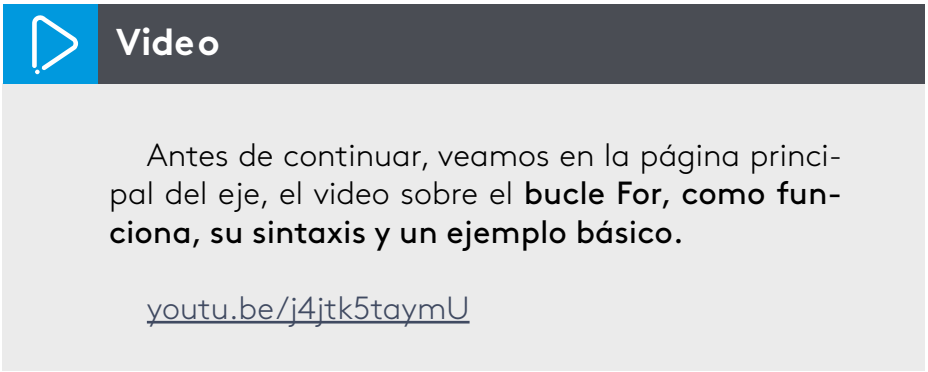
Figura 8. Ejemplo en el que empleamos el bucle do while
Fuente: propia

Bucle for

En la estructura for indicamos todo lo relacionado con la variable de control. Es decir, la estructura for no solo necesita incluir la condición, sino también el valor inicial de la variable de control (se llamará inicialización), y el cambio que se le aplica a la variable de control tras cada repetición (le llamaremos actualización).

En los bucles while tanto la inicialización como la actualización no forman parte de la misma estructura, y para ponerlos había que incluir la iniciación antes del bucle, y la actualización como una sentencia más del bloque que se repite.

En los bucles for tanto la inicialización como la actualización forman parte de la estructura, de manera que la variable de control es controlada en la misma estructura.



Sintaxis del bucle for

Entenderemos esto mejor viendo la sintaxis que utiliza este bucle. Esta es la siguiente:

```
for (<inicializacion>;<condicion>;<actualización>) {
 <sentencias que se repiten>;
}
```

Expliquemos el código de esta estructura:

- **for**: Empezamos el bucle escribiendo la palabra clave for.
- **<inicializacion>**: entre paréntesis y separados por punto y coma escribimos tres sentencias, la primera de ellas es la "inicialización", donde damos un valor inicial a la variable de control. Si esta no estaba declarada anteriormente, la podemos declarar aquí también ej.: int i=0;.
- **<condicion>**: la segunda sentencia del paréntesis es la "condición", en donde indicamos la condición que tiene que ser verdadera para que el bucle se repita. Evidentemente en la condición tiene que aparecer la variable de control. por ejemplo: i <= 10;.

- **<actualizacion>**: en esta tercera sentencia de dentro del paréntesis indicamos qué es lo que le debe ocurrir a la variable de control tras cada repetición, para que esta cambie y en un momento dado se pueda salir del bucle, en el ejemplo que estamos siguiendo, sería el incremento en una unidad: i++;.

- **{ <sentencias que se repiten>; }**: dentro de las llaves, escribimos el bloque de sentencias que se repetirán en cada vuelta.

Siguiendo con los ejemplos que hemos puesto antes podemos escribir un bucle que escriba una tabla de multiplicar, por ejemplo, la del 5:

```
for (int 1 = 0 ; i <= 10 ; i++) {
cout << "5 x " << i << " = " << 5*i << endl;
}
```

Dentro del paréntesis hemos puesto tres sentencias, la primera corresponde a la inicialización, en donde declaramos la variable i y la inicializamos a 0. En la segunda indicamos que esta variable debe ser menor o igual a 10 para que se repita el bucle. En la tercera indicamos que en cada vuelta se incrementará el valor de esta variable en una unidad.

En este programa vamos a pedirle al usuario que escriba un número entre el 1 y el 10 y el programa. El programa le devolverá la tabla de multiplicar de ese número. Si el número escrito no está entre el 1 y el 10, el programa le dirá al usuario que tiene un error y no puede escribir la tabla.

```
#include<iostream>

using namespace std;

int main() {

   cout << "AREANDINA\n";

   cout << "PROGRAMA BUCLE for\n";

  cout << "TABLAS DE MULTIPLICAR" << endl;

  int t;

  cout << "INTRODUZCA EL VALOR DE LA TABLA DE MULTIPLICAR (del 1 al 10) :";

  cin >> t;

  if (t >=1 and t<=10) {
```

```
    for (int i=1; i<=10 ; i++) {
      cout << t <<  " x " << i << " = " << t*i << endl;
    }
  }
  else {
    cout << "Error: tu número no está entre el 1 y el 10." << endl;
  }
}
```

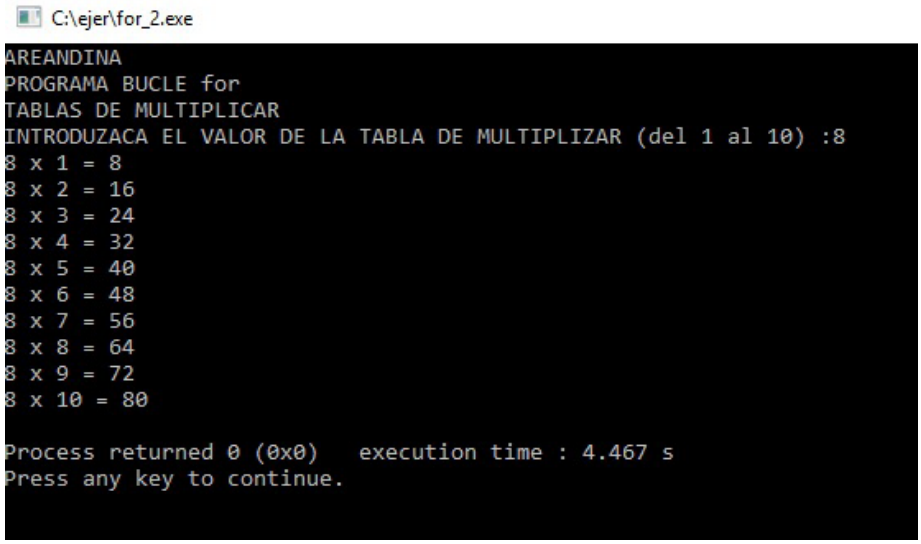


Figura 9. Ejemplo de bucle que escriba una tabla de multiplicar
Fuente: propia

Lectura recomendada

Ahora bien, para complementar este tema, le invitamos a realizar la lectura complementaria sobre el bucle for.

http://bit.ly/2i2nzuB

Y para finalizar, les invitamos a realizar la actividad de repaso 1 para el desarrollo de algunos ejercicios que permiten profundizar en los temas vistos en el presente referente.

BIBLIOGRAFÍA

Ceballos, F. (2007). Programación orientada a objetos con C++ (4a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11045993&p00=lenguaje+c%2B%2B

Ceballos, F. (2009). Enciclopedia del lenguaje C++ (2a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046572&p00=lenguaje+c%2B%2B

Gaxiola, C., y Flores, D. (2008). Metodología de la programación con pseudocódigo enfocado al lenguaje C. México: Editorial Plaza y Valdés, S.A. de C.V. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10877093&p00=algoritmos+programaci%C3%B3n

Joyanes, L. (2006). Programación en C++: algoritmos, estructuras de datos y objetos (2a. ed.) Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491359&p00=lenguaje+c%2B%2B

Joyanes, L., Castillo, A., y Sánchez, L. (2005). C algoritmos, programación y estructuras de datos. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491350&p00=lenguaje+c%2B%2B

Joyanes, L., Rodríguez, L., y Fernández, M. (2003). Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos (2a. ed.) madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498607&p00=algoritmos+programaci%C3%B3n

Joyanes, L., y Sánchez, L. (2006). Programación en C++: un enfoque práctico. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491298&p00=lenguaje+c%2B%2B

Joyanes, L., y Zahonero, I. (2007). Estructura de datos en C++. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491301&p00=lenguaje+c%2B%2B

Joyanes, L. y Zahonero, I. (2005). Programación en C: metodología, algoritmos y estructura de datos (2a. ed.). Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498360&p00=algoritmos+programaci%C3%B3n

Juganaru, M. (2014). Introducción a la programación. México: Grupo Editorial Patria. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11017472&p00=programaci%C3%B3n

BIBLIOGRAFÍA

Moreno, J. (2014). Programación. México: RA-MA. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046398&p00=lenguaje+c%2B%2B

Noguera, F., y Riera, D. (2013). Programación. Barcelona: Editorial UOC. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10853420&p00=lenguaje+c%2B%2B

Schildt, H. (2009). C++: soluciones de programación. Madrid: McGraw-Hill Interamericana. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10433927&p00=algoritmos+programaci%C3%B3n

ALGORITMOS Y PROGRAMACIÓN

Julio Cesar Rodríguez Casas

EJE 4

Propongamos

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED ILUMNO

INTRODUCCIÓN

En el siguiente eje se introducirá en el manejo de los vectores de una dimensión y los vectores multidimensionales, se presentarán ejercicios con sintaxis en lenguaje C++ que son las estructuras principales y las más útiles que tienen los lenguajes de programación; y se podrá ver cómo se asignan los valores a un vector.

Así mismo, una matriz es un vector de vector de vectores, también se le llama array bidimensional.

De igual forma, se estudiarán los arreglos multidimensionales los cuales son una estructura de datos estáticos y de un mismo tipo de datos, de longitud fija que almacena datos de forma matricial.

Cada uno de los elementos que se verán en el presente eje están acompañados con ejercicios prácticos y debidamente comprobados, para que el estudiante los prueba y analice los resultados.

Introducción a los vectores

Un array es una serie de elementos del mismo tipo al que se le puede identificar mediante un nombre, es decir, es una variable donde se guarda una lista de elementos, todos ellos del mismo tipo.

Al igual que las variables, los arrays deben ser declarados. Al declararse se debe indicar el tipo de elementos que contiene el array y también su número. Al declararse el ordenador reserva un espacio en la memoria para los elementos del array. Este espacio son una especie de casillas contiguas, cada una de ellas para un elemento del mismo.

Por lo tanto, al declarar un array indicamos el número de elementos y reservamos su espacio en la memoria, por lo que una vez declarado no podremos ampliar o reducir el número de elementos de que dispone, ni cambiar de tipo de elementos.

Posteriormente podremos acceder al array completo, como una única lista; o a cada uno de los elementos, mediante el nombre del array y un identificador para cada elemento.

Representación gráfica de un arreglo de **una dimensión**

Figura 1. Representación gráfica de un arreglo de una dimensión
Fuente: propia

Crear un vector en C++ es sencillo, a partir de la siguiente sintaxis:

o tipo nombre[tamaño];

1. int a[5];//Vector de 5 enteros
2. float b[5];//vector de 5 flotantes
3. Producto product[5];//vector de 5 objetos de tipo Producto

También se puede inicializar el vector en la declaración:

1. int a[] = {5, 15, 20, 25, 30};
2. float b[] = {10.5, 20.5, 30.5, 12.5, 50.5}
3. Producto product[] = {celular, calculadora, cámara, iPod, USB}

Como hay cinco elementos en cada array, automáticamente se le asignará 5 espacios de memoria a cada vector, pero si se trata de crear el vector de la siguiente forma int a[], el compilador mostrará un error porque no se indicó el tamaño del vector ni tampoco se inicializaron los sus elementos.

Asignar valores a los elementos de un vector indicando su posición:

1. int a[4] = 30;//se asignó el valor 30 a la posición 4 del vector.
2. product[2].setPrecio(300) //Se asignó un precio de 300 al producto en la posición 2.

El siguiente ejercicio se trata de una función simple para sumar 2 vectores A y B y poner el resultado en un tercer vector C.

```
#include <iostream>
using namespace std;

void sumar(int a[], int b[], int c[],int dim) {
        for (int i = 0; i < dim; i++) {
                c[i] = a[i] + b[i];
        }
 }

void imprimir(int v[], int dim)
  {
        for(int i = 0; i < dim; i++) {
                cout << v[i] << endl;
        }
        cout << endl << endl;
        }
         int main()
        {
        int dim;
        cout << "Ingresa la dimensión" << endl;
```

```
cin >> dim;
 int a[dim];
 int b[dim];
 int c[dim];

 for(int i = 0; i < dim; i++) {
        a[i] = i * 10;
        b[i] = i * 5;
}
cout << "Vector A"<< endl;
imprimir(a, dim);
cout << "Vector B" << endl;
imprimir(b, dim);
sumar(a, b, c, dim);
cout << «Vector C « << endl;
imprimir(c, dim);
return 0;
}
```

En la siguiente imagen se puede ver el resultado de la ejecución del programa.

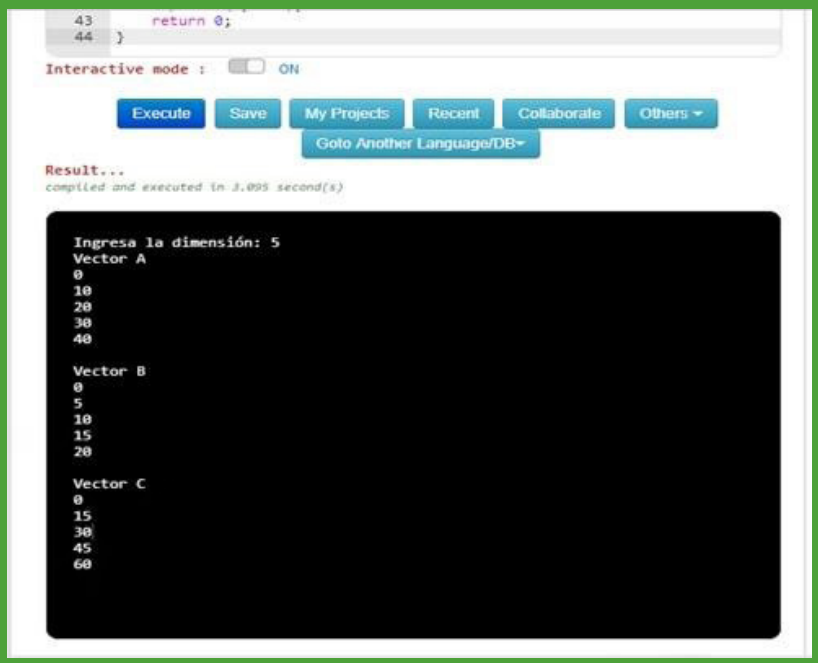


Figura 2. Resultado de ejecución de programa de función simple para sumar 2 vectores A y B y poner el resultado en un tercer vector C
Fuente: propia

Sin embargo, se debe recordar que, aunque el tamaño de a es 10, se seleccionan los elementos del vector comenzando por cero (esto se llama a veces indexado a cero, de modo que solo se pueden seleccionar los elementos del vector de 0 a 9, como sigue:

```
#include <iostream>
using namespace std;

int main() {
        int a[10];
        for(int i = 0; i < 10; i++) {
          a[i] = i * 10;
          cout << "a[" << i << "] = " << a[i] << endl;
           }
}
```

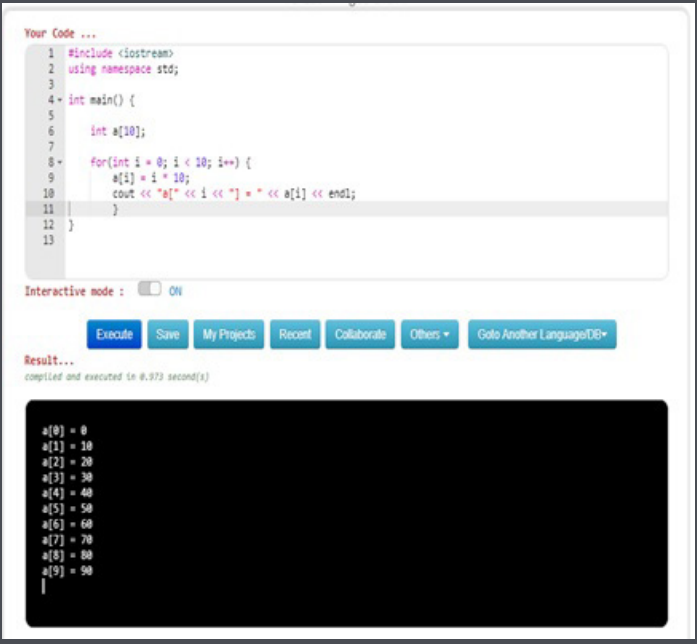


Figura 3. Ejecución de los elementos de un vector de 0 a 9
Fuente: propia

Los accesos a los vectores son extremadamente rápidos, Sin embargo, si se indexa más allá del final del vector, no hay ninguna red de seguridad y se entrará en otras variables. La otra desventaja es que se debe definir el tamaño del vector en tiempo de compilación; si se quiere cambiar el tamaño en tiempo de ejecución no se puede.

Se puede hacer un vector de cualquier tipo:

```
#include <iostream>
using namespace std;
int main() {
        int a[10];
        cout << «sizeof(int) = «<< sizeof(int) << endl;
        for(int i = 0; i < 10; i++)
        cout << "&a[" << i << "] = "
        << (long)&a[i] << endl;
 }
```

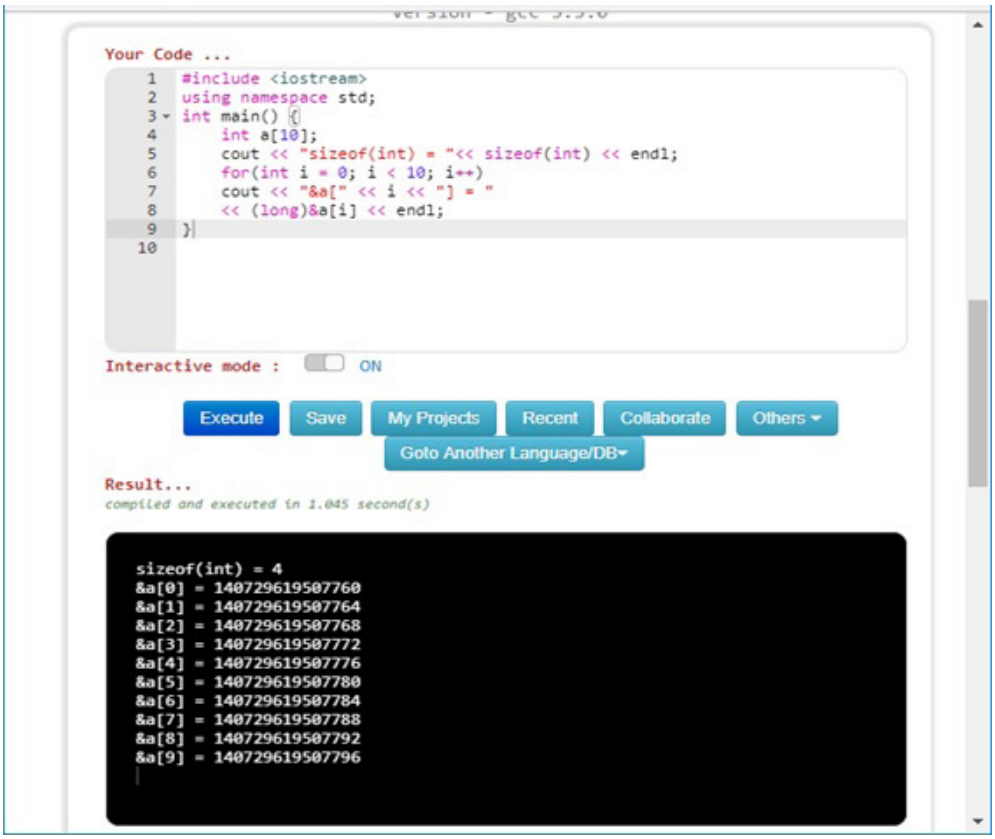


Figura 4. Resultado del programa de un vector de cualquier tipo
Fuente: propia

Cuando se ejecuta este programa, se ve que cada elemento está separado por el tamaño de un int del anterior. Esto significa, que están colocados uno a continuación del otro.

Lectura recomendada

Arrays y cadenas de texto

http://bit.ly/2ybrfQp

Video

Cómo se suman dos vectores

https://vimeo.com/235575608

Asignación de valores

En el momento de declarar un arreglo de cualquier tipo, podemos inicializarlo con los valores que queramos. Para inicializar un arreglo de enteros:

```
int MiArreglo[5] ={2,34,78,1,9};
```

Así, estos valores estarán almacenados en cada elemento del array. Es muy importante hacer notar que el primer elemento de un arreglo es el elemento 0, entonces, MiArreglo[0] contendrá el número 2, el segundo (MiArreglo[1]) contendrá el número 34 y así sucesivamente hasta MiArreglo[4] que es el último elemento del array. Si un arreglo cuenta con menos inicializadores que elementos entonces el resto se inicializará a 0.

Y en caso de que se trate de una cadena de caracteres podemos hacerlo de 2 formas:

```
char MiCadena[13]= "hola a todos";
```

o bien, declarar cada elemento

```
char MiArray[5]={'h','o','l','a','\0'};
```

Cuando se inicializa una cadena por el primer método, automáticamente se coloca un carácter de terminación de cadena (el carácter \0), en cambio, de la segunda forma debemos de ponerlo nosotros. También se puede excluir el tamaño del arreglo, el compilador lo determinará en la compilación.

Para acceder a cada elemento del arreglo debemos especificar su posición (la primera es la posición 0). En tiempo de ejecución podemos asignar valores a cada elemento cuidando siempre de no sobrepasar el tamaño del arreglo, si sobrepasamos ese tamaño se escribirán datos en un área de la memoria que no está asignada para ese array, puede escribir datos en un área en donde se almacenaba otra variable del programa o un componente del sistema, esto ocasionaría resultados no deseados.

Matrices

Una matriz es un vector de vectores o un también llamado array bidimensional. La manera de declarar una matriz es C++ es similar a un vector:

1. 1 int matrix[rows][cols];

int es el tipo de dato, matrix es el nombre del todo el conjunto de datos y debo de especificar el número de filas y columnas.

Las matrices también pueden ser de distintos tipos de datos como char, float, double, etc. Las matrices en C++ se almacenan al igual que los vectores en posiciones consecutivas de memoria.

Usualmente uno se hace la idea que una matriz es como un tablero, pero internamente el manejo es como su definición lo indica, un vector de vectores, es decir, los vectores están uno detrás del otro juntos.

La forma de acceder a los elementos de la matriz es utilizando su nombre e indicando los 2 subíndices que van en los corchetes.

Video

Cómo es el manejo de las matrices con caracteres

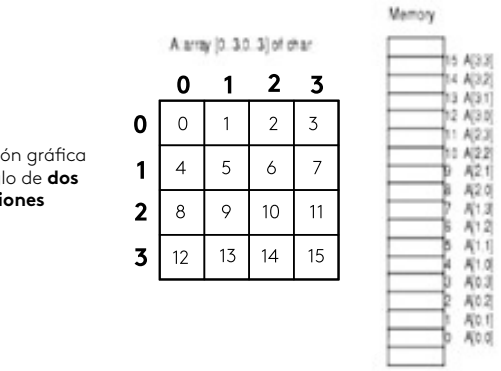
http://bit.ly/2yTgfEN

Si coloco int matriz[2][3] = 11; estoy asignando al cuarto elemento de la tercera fila el valor 11.

int arreglo [4] [4]

Representación gráfica de un arreglo de **dos dimensiones**

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15



int arreglo [4] [4] [3]
Representación gráfica de un arreglo de **tres dimensiones**

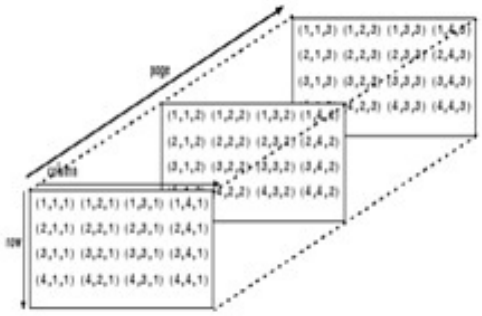


Figura 5. Arreglos de dos y tres dimensiones
Fuente: propia

Índice de un arreglo

Todo arreglo está compuesto por un número de elementos. El índice es un número correlativo que indica la posición de un elemento del arreglo. Los índices en C++ van desde la posición 0 hasta la posición tamaño –1.

Elemento de un arreglo

Un elemento de un arreglo es un valor particular dentro de la estructura del arreglo. Para acceder a un elemento del arreglo es necesario indicar la posición o índice dentro del arreglo. Ejemplo:

- *arreglo[0] //Primer elemento del arreglo*
- *arreglo[3] //Cuarto elemento del arreglo*

Arreglos unidimensionales

Un arreglo de una dimensión es una lista de variables, todas de un mismo tipo a las que se hace referencia por medio de un nombre común. Una variable individual del arreglo se llama elemento del arreglo. Para declarar un arreglo de una sola dimensión se usa el formato general:

tipo_dato identificador[tamaño];

int arreglo[3]; arreglo[0] arreglo[1] arreglo[2]

Un elemento del arreglo se accede indexando el arreglo por medio de un número del elemento. En C++ todos los arreglos empiezan en 0, esto quiere decir que si se desea acceder al primer elemento del arreglo se debe usar el índice igual a 0. Para indexar un arreglo se especifica el índice del elemento que interesa dentro de un corchete, ejemplo:

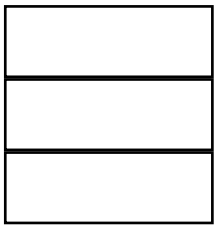
valor = arreglo[1];

Los arreglos empiezan en 0, de manera que el índice 1 se refiere al segundo elemento. Para asignar el valor a un elemento de un arreglo, ponga el elemento en el lado izquierdo de una sentencia de asignación.

mi_arreglo[0] = 100;

C++ almacena arreglos de una sola dimensión en una localización de memoria contigua con el primer elemento en la posición más baja. De esta manera, mi_arreglo[0] es adyacente a mi_arreglo[1], que es adyacente a mi_arreglo[2] y así sucesivamente. Puede usar el valor de un elemento de un arreglo donde quiera que usaría una variable sencilla o una constante.

Ejemplo. Arreglo de una dimensión.



Declaración:

int arreglo[3]; // forma un arreglo de una dimensión y de tres elementos.

Nombre del arreglo
arreglo
Nombre de los elementos
arreglo[0] → primer elemento
arreglo[1] → segundo elemento
arreglo[2] → tercer elemento

Video

Acomodar en un array,
100 números ascendentemente

http://bit.ly/2ybUdQj

El siguiente programa carga el arreglo sqrs con los cuadrados de los números del 1 al 10 y luego los visualiza.

```
1.  using namespace std;
2.  #include <iostream>
3.  int main()
4.  {
5.  int sqrs[10];
6.  int i;
7.  for (i=1;i<11;i++) {
8.          sqrs[i-1]=i*i;
9.  }
10. for (i=0;i<10;i++) {
11.         cout<<sqrs[i]<<endl;
12. }
13. return 0;
14. }
```

La forma como se almacenan los valores en el arreglo es la siguiente:

```
sqrs[0] = 1*1
sqrs[1] = 2*2
sqrs[2] = 3*3
sqrs[3] = 4*4
sqrs[4] = 5*5
sqrs[5] = 6*6
sqrs[6] = 7*7
sqrs[7] = 8*8
sqrs[8] = 9*9
sqrs[9] = 10*10
```

Figura 6. Almacenamiento de los cuadrados en un arreglo
Fuente: propia

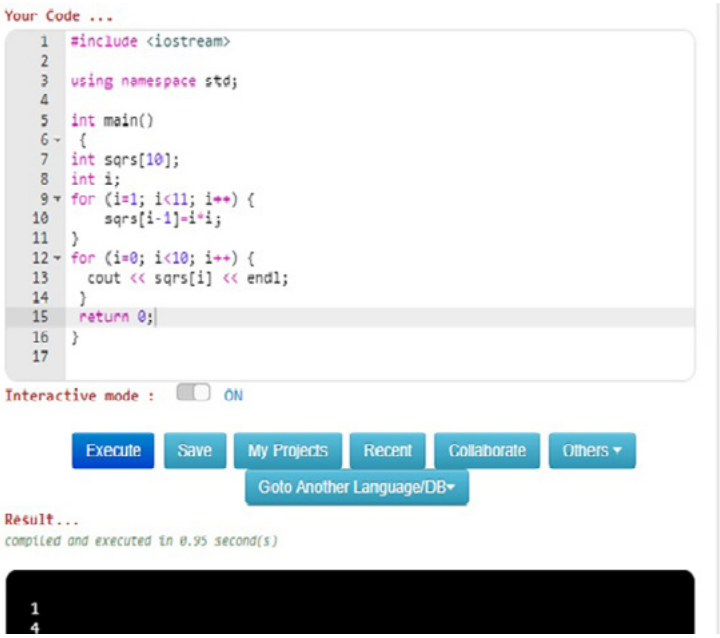


Figura 7. Resultado de la ejecución del cálculo de los cuadrados
Fuente: propia

Los arreglos prestan mucha utilidad cuando es necesario manejar lista de información. Por ejemplo, este programa lee la temperatura al mediodía, durante todos los días de un mes y luego informar la temperatura promedio mensual, así como el día más caluroso y el más frío.

```
    using namespace std;
#include <iostream>
int main()
{
int temp[31],min, max, media;
int dias;
 cout<<"Cuantos días tiene el mes"<<endl;
 cin>>dias;
  for(int i=0;i<dias;i++){
     cout<<"Introduzca la temperatura de mediodía del día"<<i+1<<":"<<endl;
     cin>>temp[i];
 }
  //Hallar la media
 media=0;
   for(int i=0;i<dias;i++){
      media=media+temp[i];
   }
 cout<<"Temperatura media: "<<media/dias<<endl;
//Hallar min y max
   min=60;//Temperatura mínima de la tierra es -90 Grados centígrados
   max=-90;//Temperatura máxima de la tierra es 60 Grados centígrados
   for(int i=0;i<dias;i++){
      if(min>temp[i]) min=temp[i];
       if(max<temp[i]) max=temp[i];
  }
   cout<<"Temperatura mínima: "<<min<<endl;
   cout<<"Temperatura máxima: "<<max<<endl;
return 0;
}
```

El resultado de la ejecución del programa es el siguiente:

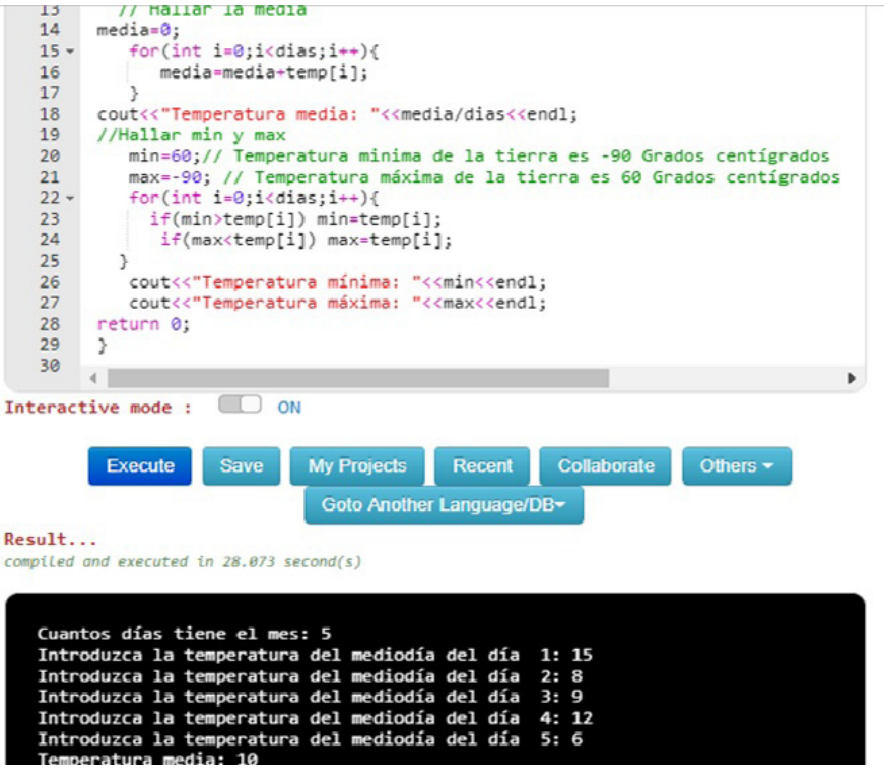


Figura 8. Resultado cálculo de temperatura máxima y mínima
Fuente: propia

Arreglos multidimensionales

De igual forma que los arreglos unidimensionales, el almacenamiento de los datos en la memoria se realiza de forma secuencial y son accedidos mediante índices. Los arreglos multidimensionales son también conocidos como matrices. Por lo tanto, se llama matriz de orden "m×n" a un conjunto rectangular de elementos dispuestos en filas "m" y en columnas "n", siendo m y n números naturales.

Las matrices se denotan con letras mayúsculas: A, B, C, … y los elementos de las mismas con letras minúsculas y subíndices que indican el lugar ocupado: a, b, c, … Un elemento genérico que ocupe la fila i y la columna j se escribe i,j. Si el elemento genérico aparece entre paréntesis también representa a toda la matriz: A (i,j).

Una matriz de orden 3x4 se muestra a continuación, siendo M una matriz de 3 filas y 4 columnas, la representación gráfica de sus posiciones sería la siguiente:

M 3x4
Filas=3, columnas= 4

Columnas

Filas	c0	c1	c2	c3
f0	m [f0,c0]	m [f0,c1]	m [f0,c2]	m [f0,c3]
f1	m [f1,c0]	m [f1,c1]	m [f1,c2]	m [f1,c3]
f2	m [f2,c0]	m [f2,c1]	m [f2,c2]	m [f2,c3]

Figura 9. Matriz de 3 filas por 4 columnas
Fuente: propia

Matrices cuadradas

Una matriz cuadrada es una matriz que tiene el mismo número de filas y columnas. La matriz que se muestra a continuación es de orden 3x3.

$$\begin{pmatrix} 1 & -3 & 8 \\ 2 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}$$

Figura 10. Matriz cuadrada de 3 filas por 3 columnas
Fuente: propia

Declaración de arreglos multidimensionales

La sintaxis es la siguiente:

tipo_dato identificador [dimensión1] [dimensión2] ... [dimensiónN] ; Donde N es un número natural positivo.

Ejemplo Arreglo de dos dimensiones de orden 2x3.

char m[2][3] ;

	c0	c1	c2
f0	a	x	w
f1	b	y	10

Figura 11. Arreglo de 2 filas por 3 columnas
Fuente: propia

Declaración

```
char m[2][3];//forma una tabla de dos filas y tres columnas
//cada fila es un arreglo de una dimensión
```

Ejemplo:

Llenado de un arreglo de enteros de dimensión 3x2. En este ejemplo el llenado lo realiza el usuario, en otros ejemplos se verá como realizar llenado de matrices mediante asignación automática, cálculos de operaciones, etcétera.

```
#include <iostream>
using namespace std;
int main()
{
int matriz [3][2];
  int valor;
  for(int i=0;i<3;i++) //Recorre las filas de la matriz
 {
 for(int j=0; j<2;j++) //Recorre las columnas de la matriz
 {
 cout<<"Ingrese el valor de la matriz en la posición ["<<i<<","<<j<<"] ";
 cin>>valor;
 matriz[i][j] = valor;
 }
 }
 //Imprimiendo el arreglo en formato matricial
 for(int i=0;i<3;i++)
 {
 cout<<"|";
 for(int j=0; j<2;j++)
 {
cout<<"\t"<<matriz[i][j]<<"\t";
 }
 cout<<"|"<<endl;
 }
 return 0;
 }
```

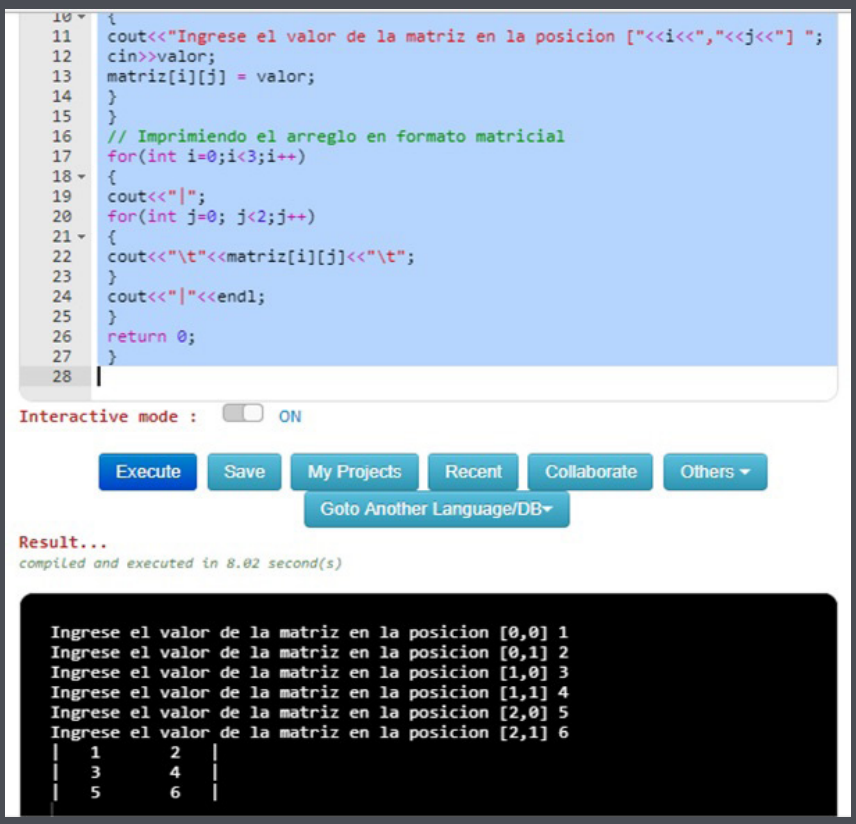


Figura 12. Resultado de una matriz de 3 por 2
Fuente: propia

Interfaz gráfica de usuario

Interfaz gráfica de usuario (en inglés Graphic User Interface, conocido pior sus siglas GUI) es un método para facilitar la interacción del usuario con la computadora a través de la utilización de un conjunto de imágenes y objetos pictóricos (iconos y ventanas), además de texto.

La GUI surge como:

> ... evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Douglas Engelbart, además de inventor del ratón de computador, desarrolló la primera interfaz gráfica en los años 1960 en EE. UU. en los laboratorios de XEROX. Fue introducida posteriormente al público en las computadoras Apple Macintosh en 1984, y a las masas hasta 1993 con la primera versión popular del sistema operativo Windows 3.0. Los GUI que no son PUI son encontrados en juegos de computadora, y los GUI avanzados basados en realidad virtual ahora son usados con más frecuencia en las investigaciones. Muchos grupos de investigación en Norteamérica y Europa están trabajando actualmente en el interfaz de enfoque del usuario o ZUI (Zooming User Interface), que es un adelanto lógico en los GUI, mezclando 3D con 2.o o "2D y medio objetos vectoriales de una D (González, 1999).

Principios para el diseño de interfaces gráficos

Existen principios relevantes para el diseño e implementación de Interfaces de usuario (IU) ya sean para IU gráficas como para la web.

Autonomía: la computadora, la IU y el entorno de trabajo deben estar a disposición del usuario. Se debe dar al usuario el ambiente flexible para que pueda aprender rápidamente a usar la aplicación.

Percepción del color: aunque se utilicen convenciones de color en la IU, se deberían usar otros mecanismos secundarios para proveer la información a aquellos usuarios con problemas en la visualización de colores.

Legibilidad

Para que la IU favorezca la usabilidad del sistema de software, la información que se exhiba en ella debe ser fácil de ubicar y leer. Es importante hacer clara la presentación visual (colocación /agrupación de objetos, evitar la presentación de excesiva información.

Leer y escribir datos en puertos seriales

Una pregunta recurrente que me suelen preguntar es la relacionada a cómo hacer X o Y a través de los puertos seriales. Es decir, cómo lograr interactuar con algún hardware conectado al puerto serial.

La respuesta a esto es sencilla. Un puerto serial no "controla" a un dispositivo. Simplemente sirve de comunicación entre la PC y el hardware. En otras palabras, tienes que enviarle bytes de información a través del puerto serial, o bien, leer esos bytes del propio puerto e interpretarlos. Pero siempre debes saber, de antemano, qué bytes está esperando, cómo se construye el búfer y cómo interpretar los bytes que te sean enviados.

Bueno, aclarado lo anterior, prosigamos. En esencia, entonces, lo que tenemos que hacer es leer y escribir bytes al puerto, ¿correcto? Pues bien, Windows trata a los puertos de la misma forma que a los archivos, aunque con un nombre establecido. El puerto "COM1" se llama "COM1:", el puerto "LPT2" se llama "LPT2:", etc. Nótese que todos terminan con dos puntos (:).

La **programación modular** es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

En la mayoría de los casos, un determinado problema complejo lo podemos (y debemos) dividir en problemas más sencillos. Estos subproblemas se conocen en el contexto de la programación como "Módulos" o subprogramas.

BIBLIOGRAFÍA

Ceballos, F. (2007). *Programación orientada a objetos con C++* (4a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11045993&p00=lenguaje+c%2B%2B

Ceballos, F. (2009). *Enciclopedia del lenguaje C++* (2a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046572&p00=lenguaje+c%2B%2B

Gaxiola, C., y Flores, D. (2008). *Metodología de la programación con pseudocódigo enfocado al lenguaje C*. México: Editorial Plaza y Valdés, S.A. de C.V. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10877093&p00=algoritmos+programaci%C3%B3n

Joyanes, L. (2006). *Programación en C++: algoritmos, estructuras de datos y objetos* (2a. ed.) Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491359&p00=lenguaje+c%2B%2B

Joyanes, L., Castillo, A., y Sánchez, L. (2005). *C algoritmos, programación y estructuras de datos*. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491350&p00=lenguaje+c%2B%2B

Joyanes, L., Rodríguez, L., y Fernández, M. (2003). *Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos* (2a. ed.) madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498607&p00=algoritmos+programaci%C3%B3n

Joyanes, L., y Sánchez, L. (2006). *Programación en C++: un enfoque práctico*. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491298&p00=lenguaje+c%2B%2B

Joyanes, L., y Zahonero, I. (2007). *Estructura de datos en C++*. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491301&p00=lenguaje+c%2B%2B

Joyanes, L. y Zahonero, I. (2005). *Programación en C: metodología, algoritmos y estructura de datos* (2a. ed.). Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498360&p00=algoritmos+programaci%C3%B3n

BIBLIOGRAFÍA

Juganaru, M. (2014). *Introducción a la programación*. México: Grupo Editorial Patria. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11017472&p00=programaci%C3%B3n

Moreno, J. (2014). *Programación*. México: RA-MA. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046398&p00=lenguaje+c%2B%2B

Noguera, F., y Riera, D. (2013). *Programación*. Barcelona: Editorial UOC. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10853420&p00=lenguaje+c%2B%2B

Schildt, H. (2009). *C++: soluciones de programación*. Madrid: McGraw-Hill Interamericana. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10433927&p00=algoritmos+programaci%C3%B3n

Esta obra se terminó de editar en el mes de Septiembre 2018
Tipografá BrownStd Light, 12 puntos
Bogotá D.C,-Colombia.

AREANDINA

Fundación Universitaria del Área Andina

MIEMBRO DE LA RED

ILUMNO