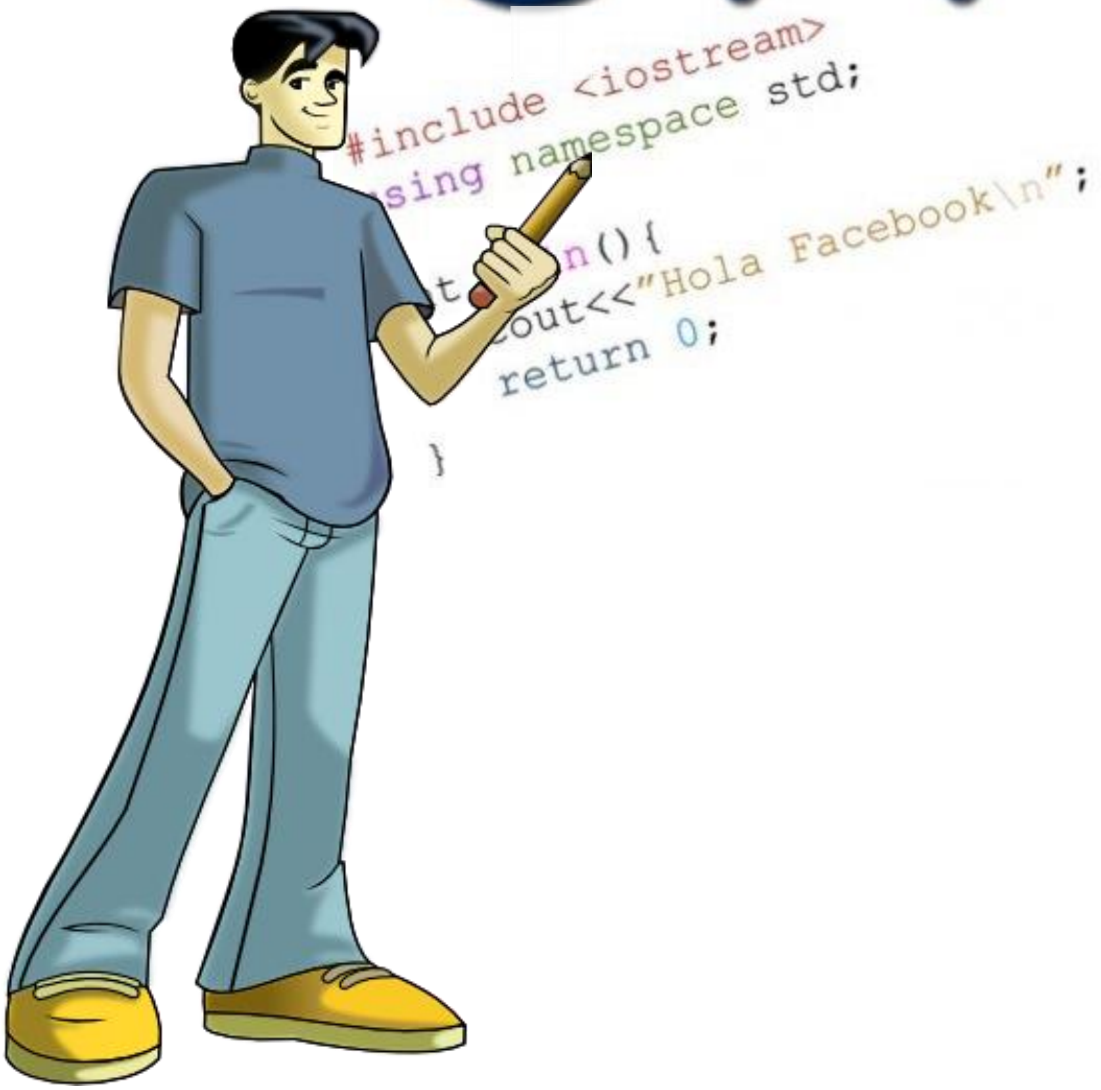
OMI
Querétaro
Olimpiada Mexicana de Informática

C++



Contenido

Algoritmos y Pseudocódigo

Algoritmos

Un algoritmo nace en respuesta a la aparición de un determinado problema.

Un algoritmo está compuesto de una serie finita de pasos que convergen en la solución de un problema, pero además estos pasos tienen un orden específico.

Entenderemos como problema a cualquier acción o evento que necesite cierto grado de análisis, desde la simpleza de cepillarse los dientes hasta la complejidad del ensamblado de un automóvil. **En general, cualquier problema puede ser solucionado utilizando un algoritmo, en este sentido podemos utilizar los algoritmos para resolver problemas de computo.**

Un algoritmo para un programador es una herramienta que le permite resaltar los aspectos más importantes de una situación y descartar los menos relevantes. Todo problema de cómputo se puede resolver ejecutando una serie de acciones en un orden específico.

Por ejemplo considere el algoritmo que se elaboraría para el problema o situación de levantarse todas las mañanas para ir al trabajo:

1. Salir de la cama
2. quitarse el pijama
3. ducharse
4. vestirse
5. desayunar
6. arrancar el automóvil para ir al trabajo o tomar transporte.

Nótese que en el algoritmo anterior se ha llegado a la solución del problema en 6 pasos, y no se resaltan aspectos como: colocarse los zapatos después de salir de la cama, o abrir la llave de la regadera antes de ducharse. Estos aspectos han sido descartados, pues no tienen mayor trascendencia, en otras palabras los estamos suponiendo, en cambio existen aspectos que no podemos obviarlos o suponerlos, de lo contrario nuestro algoritmo perdería lógica, un buen programador deberá reconocer esos aspectos importantes y tratar de simplificar al mínimo su problema.

Es importante recalcar que los pasos de un algoritmo no son conmutativos pues, no daría solución al mismo problema a tratar.

Los algoritmos se pueden expresar de diversas formas: lenguaje natural, pseudocódigo y diagramas de flujo, lenguaje de programación.

Eficiencia y Eficacia de un Algoritmo

Un algoritmo es **eficiente** cuando logra llegar a sus objetivos planteados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria, de pasos y de esfuerzo humano.

Un algoritmo es **eficaz** cuando alcanza el objetivo primordial, el análisis de resolución del problema se lo realiza prioritariamente.

Puede darse el caso de que exista un algoritmo eficaz pero no eficiente, en lo posible debemos de manejar estos dos conceptos conjuntamente.

Resolución de Problemas

Para lograr resolver cualquier problema se deben seguir básicamente los siguientes pasos:

Análisis del Problema. en este paso se define el problema, se lo comprende y se lo analiza con todo detalle.

Diseño del Algoritmo. se debe elaborar una algoritmo que refleje paso a paso la resolución del problema.

Resolución del Algoritmo en la computadora. se debe codificar el algoritmo.

Pseudocódigo

Pseudocódigo Es un lenguaje artificial e informal que ayuda a los programadores a desarrollar algoritmos.

El Pseudocódigo es similar al lenguaje cotidiano; es cómodo y amable con el usuario, aunque no es realmente in verdadero lenguaje de computadora. No se ejecutan en las computadoras mas bien sirven para ayudar al programadora razonar un programa antes de intentar escribirlo en algún lenguaje.

Un programa ejecutado en Pseudocódigo puede ser fácilmente convertido en un programa en C++, si es que esta bien elaborado. Por ejemplo supongamos que la nota para aprobar un examen es de 60. El enunciado en Pseudocódigo sería:

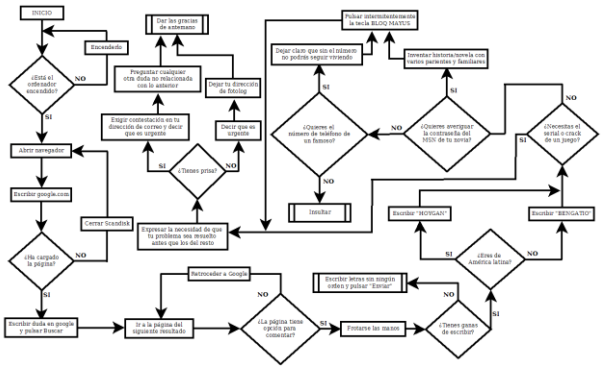
```
Si calificación >= 60 entonces
Mostrar              "Aprobado"
FinSi
```

El mismo enunciado se puede escribir en C++ como:

```
if ( calif >= 60 )
cout << "Aprobado";
```

Nótese que la operación de trasladar el Pseudocódigo a código fuente, se lo realiza con el mínimo esfuerzo, no se necesita de un mayor análisis.

Algoritmos VS Fuerza Bruta



Solución óptima, desarrollo de un procedimiento o algoritmo adecuado, si habláramos de puntaje, nos brindaría un 100%

VS

Solución parcial, desarrollo de un procedimiento en el cual desconozco el algoritmo de solución, si habláramos de puntaje, nos daría puntos sin lograr dar una solución perfecta

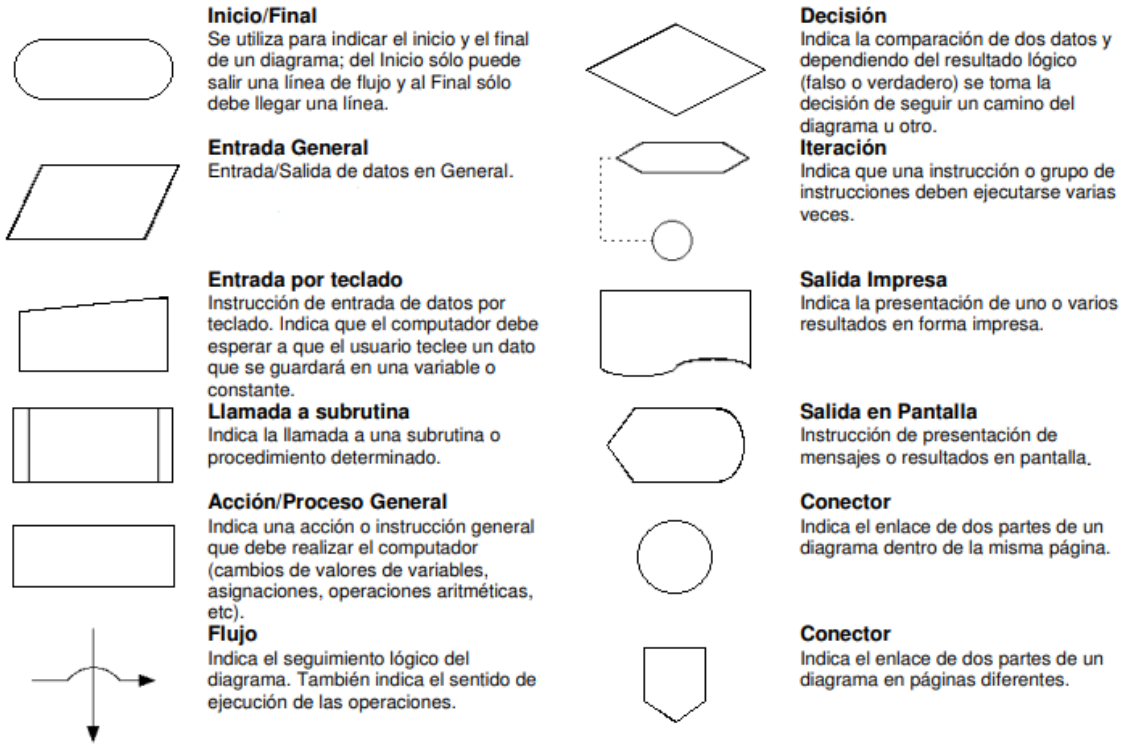
Diagramas de flujo

Un diagrama de flujo es una representación gráfica de un algoritmo o de una parte del mismo.

La ventaja de utilizar un algoritmo es que se lo puede construir independiente mente de un lenguaje de programación, pues al momento de llevarlo a código se lo puede hacer en cualquier lenguaje.

Dichos diagramas se construyen utilizando ciertos símbolos de uso especial como son rectángulos, diamantes, óvalos, y pequeños círculos, estos símbolos están conectados entre sí por flechas, conocidas como *líneas de flujo*. A continuación se detallarán estos símbolos.

Simbología básica



Inicio/Final
Se utiliza para indicar el inicio y el final de un diagrama; del Inicio sólo puede salir una línea de flujo y al Final sólo debe llegar una línea.

Entrada General
Entrada/Salida de datos en General.

Entrada por teclado
Instrucción de entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante.

Llamada a subrutina
Indica la llamada a una subrutina o procedimiento determinado.

Acción/Proceso General
Indica una acción o instrucción general que debe realizar el computador (cambios de valores de variables, asignaciones, operaciones aritméticas, etc).

Flujo
Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones.

Decisión
Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro.

Iteración
Indica que una instrucción o grupo de instrucciones deben ejecutarse varias veces.

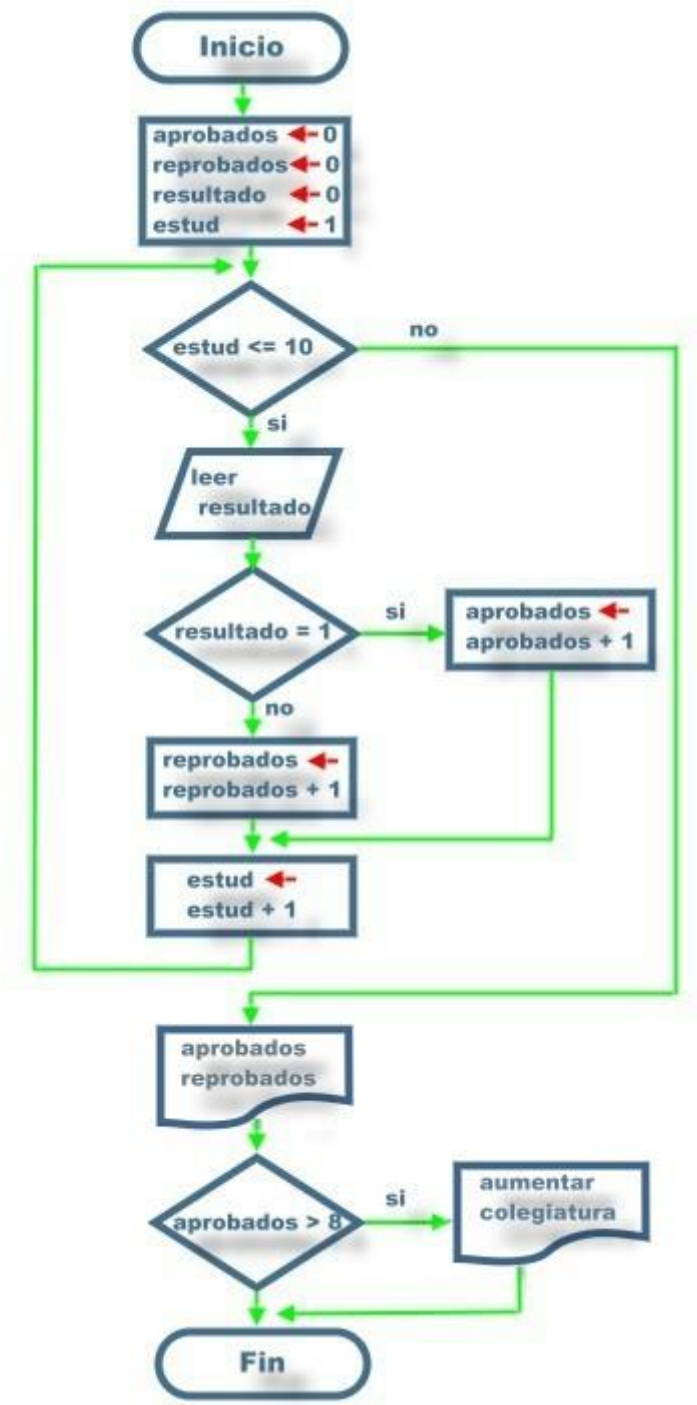
Salida Impresa
Indica la presentación de uno o varios resultados en forma impresa.

Salida en Pantalla
Instrucción de presentación de mensajes o resultados en pantalla.

Conector
Indica el enlace de dos partes de un diagrama dentro de la misma página.

Conector
Indica el enlace de dos partes de un diagrama en páginas diferentes.

Ejemplos de diagramas de Flujo



Lenguajes de Programación

- En informática, cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones para su procesamiento por un ordenador o computadora.
- Se conoce como lenguaje de programación a un programa destinado a la construcción de otros programas informáticos.

Clasificación de los Lenguajes de programación

Los programadores escriben instrucciones en diversos lenguajes de programación. La computadora puede entender directamente algunos de ellos, pero otros requieren pasos de traducción intermedios. Hoy día se utilizan cientos de lenguajes de computadora, los cuales pueden dividirse en tres tipos generales:

1. Lenguaje máquina

Una computadora sólo puede entender el lenguaje máquina. El lenguaje de máquina ordena a la computadora realizar sus operaciones fundamentales una por una. Dicho lenguaje es difícil de usar para lar persona porque trabajar con números no es muy cómodo además estos números están en formato binario.

2. Lenguajes de bajo nivel (ensamblador)

Para facilitar y agilizar su labor a los programadores, se buscaron nuevos lenguajes. El lenguaje ensamblador consiste en pequeñas abreviaturas de palabras en ingles. Se crearon los programar traductores para convertir los programas escritos en lenguaje ensamblador a lenguaje máquina a velocidades de computadora. Estos lenguajes aun requerían muchas instrucciones para realizar simples operaciones.

3. Lenguajes de alto nivel

Para acelerar, aún más, el proceso de programación se desarrollan los lenguajes de alto nivel en los que se podía escribir un enunciado para realizar tareas sustanciales. Los lenguajes de alto nivel permiten a los programadores escribir instrucciones que asemejan el ingles cotidiano y contiene notaciones matemáticas de uso común.

Ambientes de Programación

Los programadores necesitan un ambiente de programación, es decir, una lugar en donde puedan plasmar sus ideas, un lugar en donde puedan escribir sus programas, en otras palabras donde puedan programar.

Los ambientes de programación vienen a ser los diferentes leguajes de programación que existen, son muy variados, con muchas cualidades propias pero se puede realizar una misma tarea, muchas veces, con cualquiera de ellos. Existen lenguajes de programación de Alto y Bajo nivel; entre los más conocidos de Alto nivel podemos mencionar a C, C++, JAVA, Fortran, T. Pascal, etc.

Tipos de ambientes de Programación: Compiladores e Intérpretes

Compilador

Un compilador es un programa que lee el código escrito en un lenguaje (lenguaje origen), y lo traduce o traduce en un programa equivalente escrito en otro lenguaje (lenguaje objetivo). **Como una parte fundamental de este proceso de traducción, el compilador le hace notar al usuario la presencia de errores en el código fuente del programa.** Vea la figura de abajo.

El C++ es un lenguaje que utiliza un compilador y su trabajo es el de llevar el código fuente escrito en C++ a un programa escrito en lenguaje máquina.

Entrando en más detalle un programa en código fuente es compilado obteniendo un archivo parcial (un objeto) que tiene extensión obj luego el compilador invoca al linker que convierte al archivo objeto en un ejecutable con extensión exe que como ya sabemos es un archivo que esta en formato binario (ceros y unos) y que puede funcionar por si solo.

Además el compilador de C++ al realizar su tarea realiza una comprobación de errores en el programa, es decir, revisa que todo este en orden por ejemplo variables y funciones bien definidas, revisa todo lo referente a cuestiones sintácticas, esta fuera del alcance del compilador que por ejemplo el algoritmo utilizado en el problema funcione bien.

Interprete

Los interpretes en lugar de producir un Lenguaje objetivo, como en los compiladores, lo que hacen es realizar la operación que debería realizar el Lenguaje origen. Un interprete lee el código como esta escrito y luego lo convierte en acciones, es decir, lo ejecuta en ese instante.

Existen lenguajes que utilizan un Interprete, como por ejemplo JAVA, y su interprete traduce en el instante mismo de lectura, el código en lenguaje máquina para que pueda ser ejecutado.

La siguiente figura muestra el funcionamiento de un interprete.

Diferencia entre Compilador e Interprete

Los compiladores difieren de los intérpretes en varios aspectos:
Un programa que ha sido compilado puede correr por sí solo, pues en el proceso de compilación se lo transformo en otro lenguaje (lenguaje máquina).
Un intérprete traduce el programa cuando lo lee, convirtiendo el código del programa directamente en acciones.

La ventaja del intérprete es que dado cualquier programa se puede interpretarlo en cualquier plataforma (sistema operativo), en cambio el archivo generado por el compilador solo funciona en la plataforma en donde se lo ha creado. Pero por otro lado un archivo compilado puede ser distribuido fácilmente conociendo la plataforma, mientras que un archivo interpretado no funciona si no se tiene el intérprete.

Hablando de la velocidad de ejecución un archivo compilado es de 10 a 20 veces más rápido que un archivo interpretado.

Porque  ?

El Lenguaje C, fue desarrollado por Kernighan y Ritchie en 1972.

El lenguaje C++ se comenzó a desarrollar en 1980. Su autor fue Bjarne Stroustrup.

Al comienzo era una extensión del lenguaje C.

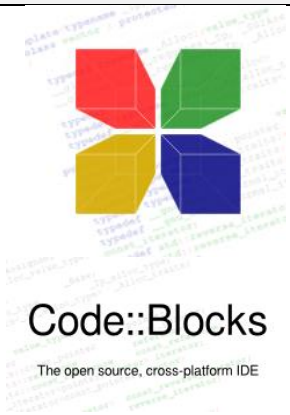
El nombre C++ establecido en1983, hace referencia al carácter del operador incremento de C (++).

Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, se estandarizo a nivel americano e internacional.

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones.

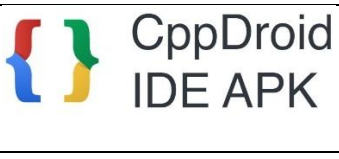
El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. La evolución de C++ ha continuado con la aparición de Java, un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras, que se utiliza para realizar aplicaciones en Internet.

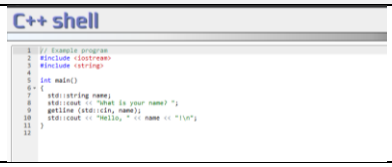
Algunos Compiladores para C++

Para computadoras	
	Uno de los lenguajes más utilizados para programar, o aprender a programar es C++ y Code::Blocks es un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C++. Está basado en la plataforma de interfaces gráficas WxWidgets, lo cual quiere decir que puede usarse libremente en diversos sistemas operativos, y está licenciado bajo la Licencia pública general de GNU. Es una herramienta para desarrollar programas en C++ muy potente, proporcionando a los usuarios una interfaz que permite trabajar con facilidad.

	Es un IDE para crear aplicaciones utilizando el lenguaje de programación C++, que ocupa muy poco tamaño en el disco duro, ideal para crear programas pequeños en las que solo sea necesario demostrar el uso de estructuras de control y estructuras de datos, estas aplicaciones se pueden compilar rápidamente y ejecutar en forma de consola.

Para celular	
	Cxxdroid es el IDE educativo C y C ++ más fácil de usar para Android. **Compilador C / C ++ sin conexión: no se requiere Internet para ejecutar programas.**

	CppDroid es simple C / C ++ IDE centrado en el aprendizaje de lenguajes de programación y las bibliotecas. **Compilador C / C ++ sin conexión: no se requiere Internet para ejecutar programas.**

Compiladores online	
	**http://cpp.sh/**
	**https://www.onlinegdb.com/online_c++_compiler**

Programa

Un programa de computadora es una serie de instrucciones, órdenes a la máquina, que producirán la ejecución de una determinada tarea. Es un medio para satisfacer una necesidad o cumplir un objetivo de una manera automatizada.

Comúnmente, la palabra programa es usada de dos maneras: para describir instrucciones individuales, **código fuente**, creado por el programador y también describe una pieza entera de software ejecutable. Esta distinción puede causar confusión, por lo que vamos a tratar de distinguir entre el código fuente por un lado y un ejecutable por otro.

Para tener un programa ejecutable primero tenemos que tener su código fuente, es decir, del código fuente se deriva el programa ejecutable.

El Código fuente puede ser convertido en un programa ejecutable de dos formas: **Interpretes** convierten el código fuente en instrucciones de computadora (lenguaje máquina), y la computadora actúa con esas instrucciones inmediatamente. El JAVA es un lenguaje interpretado.

Alternativamente, los **compiladores** trasladan el código fuente en programas, los cuales pueden ejecutarse tiempo después.

A pesar de que se puede trabajar fácilmente con los intérpretes, la mayor parte de la programación es hecha con compiladores porque el código compilado se ejecuta más rápido. **C++ es un lenguaje de compilación**.

Estructura de un Programa en C++

Un programa está formado por la cabecera y el cuerpo del programa.

Cabecera

En la cabecera se incluyen a nuestro programa algunas rutinas predefinidas que hacen a la programación más sencilla, pues no tenemos que crear todo desde cero o "tratar de inventar la rueda", es muy bueno que conozcamos la mayor cantidad de librerías disponibles para que tengamos un trabajo más que sencillo y estandarizado. Un programa puede no tener cabecera pero sería demasiado simple, he aquí un ejemplo de una cabecera para un programa sencillo.

Componentes

1.- Librerías Básicas de C++ y sus funciones iostream math stdio stdlib string etc

#include<iostream> es un componente de la biblioteca estándar (STL) del lenguaje de programación C++ que es utilizado para operaciones de entrada/salida. Su nombre es un acrónimo de Input/Output Stream.

2.- Es necesario agregar nuevas palabras reservadas por medio de "**using namespace std**" (espacio de nombres). Las palabras reservadas cout y cin están el namespace std (standard).

En caso de que no declaremos el uso del namespace std cada vez que quisieramos usar cout, tendríamos que escribir std::cout << "Hola mundo";

Ejemplo_

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

Cuerpo

El cuerpo del programa contiene la función principal, las funciones adicionales y las clases que se necesiten en el programa.

La mejor forma de aprender un lenguaje es programando con él. El programa más sencillo que se puede escribir en C++ es el siguiente:

```
int  main()
{

}
```

Estructura Básica de un programa en C++

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

int  main()
{

}
```

Como nos podemos imaginar, este programa no hace nada, pero contiene la parte más importante de cualquier programa C++ y además, es el más pequeño que se puede escribir y que se compile correctamente.

En el se define la función int main, que es la que ejecuta el sistema operativo al llamar a un programa C++. La función principal (void main o int main) o cualquier otra función siempre va seguida de paréntesis.

La definición del cuerpo de la función está formada por un bloque de sentencias o instrucciones, que esta encerrado entre llaves { }.

Un programa C++ puede estar formado por diferentes módulos de código fuente. Es conveniente mantener los código fuente de un tamaño no muy grande, para que la compilación sea rápida. También, al dividirse un programa en partes, puede facilitar la legibilidad del programa y su estructuración. Los diferentes códigos fuentes son compilados de forma separada, únicamente el código fuente que han sido modificados desde la última compilación, y después combinados con las librerías necesarias para formar el programa en su versión ejecutable.

Mecanismos de Salida (cout)

Los mecanismos de Salida son aquellos mensajes que el programa utiliza para comunicarse con el mundo exterior o con el usuario. Por ejemplo yo quiero un programa que me salude cordialmente:

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

int main()
{
cout<<" Hola amigo! \n" ;
}
```

Para usar cout se debe colocar esta palabra seguida del operador < que se lo coloca dos veces, luego entre comillas dobles "" se coloca el texto que se quiere mostrar por pantalla. El símbolo \n, colocado al final del texto, indica un cambio de línea.

cout<<" Hola amigo! \n" **;** **//imprimir mensajes entre comillas**

cout<<x; **//imprimir valores de variables**

cout<<a+b; **//imprimir operaciones directas**

Algunas Opciones.

Salto de Línea, o un enter,

- ✓ se agrega en el cout <<"/n";
- ✓ su equivalente es cout<<endl;

// Asignar Comentarios por línea

/* */ Asignar comentarios de párrafos

Mecanismos de Entrada (cin)

Los mecanismos de Entrada nos permiten interacción entre el mundo exterior (Usuarios) y el programa, así el programa puede recabar información necesaria para cumplir con su meta.

Ejemplo:
Un ejemplo sencillo sería que el programa nos pregunte nuestra edad:

```
#include <iostream>

using namespace std;
int main()
{
int edad;
cout<<" ¿Qué edad tienes? \n" ;
cin>>edad;
cout<<"Tienes "<<edad<<" años";
}
```

Para usar cin se debe colocar esta palabra seguida del operador >que se lo coloca dos veces, luego la variable a la cual le asignaremos un valor asignado desde el teclado.

cin>>x; **//asignar un valor a una variable**

cin>>x;

cin>>y; **//asignar un valor a dos variables x , y**

cin>>x>>y; **//podemos reducir la instrucción cin anterior**

Variables

Un programa necesita un medio de grabar los datos que usa. Las variables y Constantes ofrecen varias maneras para representar y manipular los datos.

Definición de variable

Una variable es un espacio para guardar información. Entrando más a detalle una variable es una ubicación en la memoria de la computadora en la cual se puede grabar un valor y por la cual se puede recuperar ese valor más tarde.

La memoria RAM de la computadora puede ser vista como una serie de pequeñas casillas, cada una de las casillas esta numerada secuencialmente, este número que se le asigna representa su dirección de memoria y su objetivo es identificarla.

Una variable reserva uno o más casillas en las cuales es posible grabar datos. Los nombres de las variables (por ejemplo, myVariable) es una etiqueta en una sola casilla, para que se pueda encontrarla fácilmente sin saber su actual dirección de memoria.

Tipos de Variables Memoria

Las variables en C++ pueden ser de varios tipos y serán utilizadas en función del tipo de datos que queramos almacenar en ellas

Las variables NOMBRE, nombre, Nombre son tres variables totalmente distintas. (Lenguaje Case Sensitive) y el nombre de una variable no puede comenzar por número (pero puede contener varios) ni tener caracteres especiales (se admite el guión bajo).

Según dónde estén declaradas, **las variables pueden ser globales** (declaradas fuera de todo procedimiento o función) **o locales** (declaradas dentro de un procedimiento o función). Las primeras serán accesibles desde todo el código fuente y las segundas sólo en la función donde estén definidas.

Por otra parte, según el tipo de datos a almacenar, las variables serán:

TIPO	NOMBRE	TAMAÑO (byte)	ALMACENA
Enteros	**int**	4	Numeros enteros sin decimales entre -2.147.483.648 y 2.147.483.647
Enteros Largos	**long long int**	8	Numero entero sin decimales entre -9.223.372.775.808 y 9.223.375.775.807.
Entero Largo sin signo	**unsigned long long int**	8	Numero entero sin decimales 0 … 18,446,744,073,709,551,615
Decimales	**float**	4	Numero decimal, es usado comúnmente en números con 6 o menos cifras decimales
Numero	**double**	8	Numero decimal, es usado comúnmente en números menos de 15 cifras decimales
Caracter	**char**	1	Un carácter (Almacena caracteres ASCII) , -128 … 127
Booleano	**bool**	1	Este tipo de dato, es comúnmente usado en condicionales o variables que solo pueden tomar el valor de 0 y 1 **(false , true).**
Cadena	**string**		**Descontinuado** (almacenaba cadena de caracteres)
Vacio	**void**		La palabra reservada void (Null) define en C++ el concepto de no existencia o no atribución de un tipo en una variable o declaración. Aplicada a funciones, void no devolverá ningún valor.

El tipo int se utiliza para guardar números enteros, es decir, que no tienen decimales. El rango de valores admitidos varía según la CPU utilizada.

Ejemplo:

```
int x=8;
cout<< x;
```

siendo el resultado por pantalla: *8*

Además, toda variable puede cambiar su valor durante la ejecución del programa y ser desplegada las veces que creamos oportunas.

Tras estas pequeñas aclaraciones, seguiremos comentando el uso de variables con el tipo float. Este tipo sirve para almacenar números decimales o reales, así como el double. El rango de valores admitidos en tan amplio que rara vez se nos quedará obsoletos.

El tipo char, capaz de almacenar un único carácter. Internamente, un carácter es almacenado como un número

Reservando Memoria

Se reserva memoria en el momento de definición de las variables, en este momento es donde se debe de especificar al compilador que clase de variable es: un entero (int), un caracter (char), etc. Esta información le dice al compilador cuanto de espacio debe separar o reservar, y que tipo de valor se va ha guardar en la variable.

Cada casilla de memoria tiene un byte de capacidad. Si el tipo de variable que se crea es de dos bytes te tamaño, este necesita de dos bytes de memoria, o de dos casillas. El tipo de variable (por ejemplo, entero) le dice al compilador cuanta memoria (o cuantas casillas) tiene que reservar para la variable.

Porque las computadores usan los bits y los bytes para representar los valores, y porque la memoria es medida en bytes, es importante entender y sentirse cómodo con este concepto.

Definir una Variable

Para crear una variable es preciso definirla. En la definición de una variable se manifestando su tipo, seguida de uno o más espacios, luego se escribe el nombre de la variable y para finalizar punto y coma.

El nombre de la variable puede ser cualquier combinación de letras, claro que sin espacios. Nombres de variables aceptadas son: x, jap007, miedad.

Importante. Los nombre buenos de variables nos dice para que la variable es utilizada, usando buenos nombres se nos hace más fácil la compresión del programa. La siguiente sentencia define una variable entera llamada **miedad**.

```
int miedad;
```

Como practica general de programación, se debe evitar los nombres horroríficos como j23qrs o xxx y restringir los nombres de variables de una sola letra como x ó y, para valores que sean de uso rápido y no perduren en todo el programa. Se debe tratar de usar nombres extensos como **miedad** o **contador**. Algunos nombres son fáciles de entender tres semanas después en lugar de romperse la cabeza imaginándose que significan nombres cortos.

Inicializar una variable

Una vez definida una variable se debe proceder a darle un valor, es cierto que este valor puede cambiar a lo largo del programa, pero es bueno acostumbrarse a dar siempre un valor inicial a nuestras variables. Por ejemplo:

```
miedad = 0;
notaFinal = 0;
```

Asignación de un Valor

Se le puede asignar valores a una variable cuantas veces se quiera durante el programa, se le asigna un valor utilizando el operador de igualdad "=".

Palabras Reservadas o Claves

Existen en todos los lenguajes nombres o palabras que ya están siendo usadas, y por eso se les da el nombre de palabras reservadas o claves.

Por ejemplo en el caso de C++ palabras reservadas son: int, if, const, main ...., etc. No podemos nombrar por ejemplo una variable definida por nosotros con ninguna palabra reservada porque el compilador encontraría un error.

Constantes

Las constantes son variables que contienen un valor que no cambia durante todo el programa. Una constante simbólica al igual que cualquier variable tiene un tipo y un nombre. Existen dos formas de declarar constantes en C++.

La primera es utilizando una instrucción, generalmente en la cabecera, que es como sigue:

```
#define Estudiantes 50
```

Es la forma tradicional de definir constantes, pero nótese que Estudiantes no tiene un tipo de dato. Lo que hace #define es simplemente sustituir 50 en todas las ocurrencias del programa donde aparezca Estudiantes.

La segunda forma es mucho más específica y mucho más útil y es así:

```
const int Estudiantes = 50;
```

Esta forma es mucho más ventajosa porque la constante Estudiantes tiene un tipo de dato lo que hace al código mucho más mantenible y lo previene de errores.

Expresiones y Operadores

Una expresión es todo aquello que se evalúa y devuelve un valor. Existen varios tipos de expresiones de acuerdo lo que contienen.

Las **expresiones** consisten de una secuencia de operadores y operandos que especifican una operación determinada.

Los operandos pueden ser variables, constantes.

Los operadores son de diversos tipos

- **Operadores aritméticos son (+ - * / %).**

Es más sencillo pensar que una expresión aritmética es como una ecuación o una fórmula matemática.
Una expresión aritmética sencilla es: area = base * altura ;

- **Operadores Relacionales (<, >, >=, <=, ==, !=)**

Este tipo de expresiones es evaluado y devuelve un valor, la diferencia esta en que este valor sólo puede ser verdadero o falso.

Una expresión sencilla es: if(a==0) ;

- **Operadores Lógicos o Booleanos (&&, ||, !) es lo mismo (and, or, not)**

Este tipo realiza múltiples expresiones y es evaluado y devuelve un valor, la diferencia está en que este valor sólo puede ser verdadero o falso.

Un ejemplo de expresiones lógicas es el siguiente:

if ((nota > 70) && (nota < 90))

En la anterior instrucción el segmento (nota > 70) && (nota < 90) devolverá 1 (verdadero) ó 0 (falso).

Tablas de Verdad

Una *tabla* de *verdad* es un diagrama que permite determinar claramente cuando una proposición compuesta es verdadera, falsa o variada

Operador &&	
V && V	**Verdadero**
V && F	**Falso**
F && V	**Falso**
F && F	**Falso**

Operador ||	
V || V	**Verdadero**
V || F	**Verdadero**
F || V	**Verdadero**
F || F	**Falso**

Operador !	
! V	**Falso**
! F	**Verdadero**

Problemas sugeridos en OMEGAUP

Entrada/ Salida – Matemáticos

- Problema SB01 Bienvenido a C++
- A+B Envío de Soluciones
- Área de un rectángulo
- Area de un triangulo
- Area del circulo
- Promedios
- Promedio de 5 calificaciones
- Foraneo hambriento
- Bolitas de Queso
- Suma de Enteros

Estructuras de Control

Existen tres clases de estructuras de control:

1. **Secuenciales**
2. **Condicionales**
3. **Iterativas**

Los programas que escribamos pueden definirse en base a las tres estructuras de control ya mencionadas.

Las estructuras secuenciales son , donde las insctrucciones se ejecutan una tras otra.
Las estructuras condicionales que C++ nos ofrece son: **if**, **if / else**, **switch**.
Las estructuras iterativas son: **for**, **while**, **Do** / **while.**

Partes de una estructura de Control

Diferenciaremos dos partes en una estructura de control:

1. La definición de dicha estructura
2. El cuerpo de la estructura.

En la definición es donde se coloca el nombre de la estructura que se va ha utilizar y en el cuerpo de la misma se ubican todas las sentencias o instrucciones que pertenecen o hacen referencia a dicha estructura. Si es cuerpo de tiene más de una instrucción va entre llaves (**{ }**).

Sentencias o Instrucciones

Una **sentencia** es la unidad ejecutable más pequeña de un programa en C++, en otras palabras una línea de código escrita es una sentencia. Las sentencias controlan el flujo y orden de ejecución. Una sentencia de C++ consta de palabras clave o reservadas como (cout, cin, for, while, if ... else,etc.), expresiones, declaraciones, o llamadas a funciones.

Toda sentencia simple termina con un punto y coma (**;**).

Dos o más sentencias pueden aparecer en una sola línea separadas por el punto y coma.

ejemplo: cout<<"hola"; cout<<"como te llamas"; cin >> nombre;

aunque esto no es recomendables por cuestion de estetica y lo dificil que seria la correccion de errores.

Una sentencia nula es simplemente un punto y coma.

Estructuras de Selección (if)

La Estructura de Selección if

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.

Fig. 5.1

La estructura de selección **if** (que se muestra en la figura 5.1) trabaja de la siguiente manera: si la evaluación de la *expresión* o *expresiones* es verdadera (1) entonces se ejecuta la *sentencia* a la cual se refiere la estructura de control if.

Si fueran varias sentencias a las que se refiere la estructura if (como se muestra en la figura 5.2) se tiene que encerrar todas las sentencias entre llaves (**{ }**) y si la evaluación de la expresión es correcta entonces se ejecuta todas las sentencias contenidas entre las llaves.

Fig. 5.2

Si la evaluación de la expresión o expresiones resultaría falsa (0), entonces no se ejecuta las 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.

```
#include <iostream>

using namespace std;

int main()
{
cout<<"¿Qué edad tienes? \n";
cin>>edad;
if ( edad > 20 )
  { cout<<"Eres mayor de edad";
    cout<<"Te estas volviendo viejo ";
  }
}
```

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

El diagrama de flujo de la estructura if se muestra en la Fig. 5.3, este diagrama contiene el símbolo diamante que es llamado el símbolo de decisión, que indica que decisión se debe tomar. El símbolo de decisión contiene una expresión, y la evaluación de ésta será verdadera o falsa. Las flechas nos indican los dos posibles caminos que se pueden tomar.

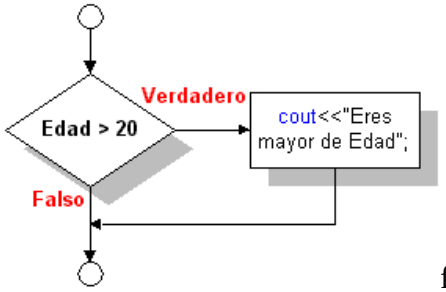


fig 5.3

El anterior diagrama de flujo funciona de la siguiente manera:

1. Si Edad es mayor 20 (verdadero) entonces se muestra "Eres mayor de Edad"

2. Si no, entonces no hace nada

Importante. Para mantener el código legible es bueno dejar espacios o sangrías en todas las líneas de código que están dentro de una estructura if / else o if, estas instrucciones se alinean un poco más a la derecha y así podemos notar claramente que forman parte de las sentencias que componen a la estructura if.

Esta práctica se puede aplicar a cualquier estructura de control. Tambien se puede unir condiciones utilizando los operadores logicos and, or y not, por ejemplo si se desea saber si 2 condiciones si han cumplido es necesario tener un formato como:

```
If ( matematicas >=60 and quimica>=80 )
{
cout << "pasaste ambas materias, felicidades;
}
```

Nota: solo si tu calificacion de matematicas es igual o mayor a 60 y la de quimica es igual o mayor a 80 se desplegara el mensaje de "pasaste...."

Estructura de Selección if / else

La estructura if / else lo que hace es ejecutar una acción si el resultado de la evaluación de la expresión es verdadera y otra acción si el resultado de la evaluación es falsa.

La diferencia con utilizar sólo la estructura if es que si la expresión evaluada es verdadera sólo en ese caso se ejecuta una acción de otro modo se pasa de largo. En cambio en la estructura if / else si la expresión es falsa entonces se ejecuta otra acción.

Fig. 5.4

En síntesis lo que hace esta estructura es realizar una acción si la expresión es verdadera y otra si es falsa.

Aquí tenemos un ejemplo para ilustrar la estructura if / else.

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

int main()
{
if ( edad > 20 )
   cout<<"Eres mayor de edad" ;
else
   cout<<"No eres mayor de edad";
}
```

El diagrama de flujo correspondiente a esta estructura es el siguiente:

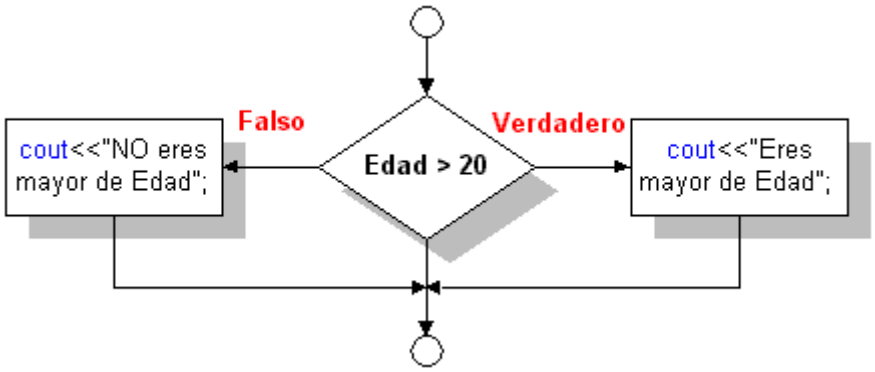


fig. 5.5

El anterior diagrama de flujo funciona de la siguiente manera:

1. Si Edad es mayor que 20 (verdadero) entonces se muestra: “Eres mayor de Edad”.
2. Si no entonces se muestra: “NO eres mayor de Edad”.

Estructuras Condicionales Anidadas

Existe el caso de estructuras if, if/else **anidadas**, que no es más que una estructura if o if/else dentro de otra.

Por **ejemplo** se desea conocer cuál es el mayor de tres números A, B ,C.

```
1: #include <iostream>
2: int main()
3: {
4: int A=0, B=0, C=0;
5: cout<<"Ingrese 3 números";
6: cin>>A>>B>>C; //Lectura de valores por teclado
7: if ( A > B )
8:    { if (A > C )
9:          cout<<"A es el número mayor";
10:     else
11:         cout<<"C es el número mayor";
12:   }
13: else
14:   { if ( B > C )
15:         cout<<"B es el número mayor";
16:     else
17:         cout<<"C es el número mayor";
18:   }
19: }
```

Para explicar el funcionamiento del problema anterior se han enumerado todas las líneas de tal programa.

Suponiendo que los valores ingresados por teclado fueran: A=2, B=1, C=5. El programa actuaría de la siguiente manera.

El programa se ejecuta secuencialmente hasta la línea 7 donde encuentra una expresión. Cuando se evalúe la expresión (A > B) **línea 7**, el resultado es **verdadero** (2 > 1), por lo tanto el programa ejecuta la **línea 8**, en dicha línea encuentra otra expresión (A > C) y el resultado de evaluar dicha expresión es **falso** (2 > 5) por lo que el programa salta hasta la **línea 11** y muestra "C es el número mayor". Luego va a la **línea 12** y verifica el cierre de llaves y finalmente salta hasta la **línea 19** siendo la última línea de ejecución.

Estructura de Selección switch

Estructuras de selección; aunque la sentencia if de C++ es muy potente, en ocasiones su escritura puede resultar tediosa, sobre todo en casos en los que el programa presenta varias elecciones después de chequear una expresión: selección múltiple o multialternativa. En situaciones donde el valor de una expresión determina que sentencias serán ejecutadas es mejor utilizar una sentencia switch en lugar de una if.

La estructura es:

```
switch (selector){
        case <opcion 1>:
                <bloque de instrucciones>
                break;
        case <opcion 2>:
                <bloque de instrucciones>
                break;
                        ⋮
        case <opcion n>:
                <bloque de instrucciones>
                break;
        default:
                <bloque de instrucciones>
}
```

Problemas sugeridos en OMEGAUP

Comparaciones if, if/else

- Es Mayor o Menor de Edad
- E/S-Terna pitagórica
- Maximo de tres
- Par o Impar
- Lada
- Selectivas-Aprobado o Reprobado
- ¿Sentados o parados?
- Nivel Óptimo

Comparaciones if, if/else - Uso de Operadores Lógicos

- Temperaturas
- Determina si es primo
- 1 Bisiesto sencillo
- Primo Facilon
- Estaciones de Radio
- Acomoda el número

Tipos de Estructuras Iterativas

1- Estructura For
2- Estructura While
3- Estructura Do....While

La Estructura de Repetición for

Esta estructura de repetición es más utilizada cuando sabemos el número de repeticiones que deseamos ejecutar. La notación de esta estructura es sencilla y se detalla a continuación

for (condición de inicio **;** expresión **;** acción después de cada iteración **)**
{
sentencia (s)**;**
}

La ***condición de inicio*** quiere decir que podemos inicializar una variable que vayamos a utilizar dentro el cuerpo de la estructura for.

La ***expresión*** nos indica que se seguirá iterando(repitiendo) mientras la condición sea verdadera.

La ***acción después de cada iteración*** viene a ser lo que queremos hacer variar después de cada iteración, esta variación podría ser un incremento en la variable definida en la condición de inicio.

Al igual que las demás estructuras de control el cuerpo de la estructura **for** lleva llaves si este contiene más de una sentencia o instrucción.

Ejemplo Un ejemplo sencillo puede ser que quiero cantar 10 veces la canción del elefante, el código sería algo así:

```
#include <iostream>
using namespace std;
int main()
{
int i;
for( i = 1 ; i<=10 ; i++)
{
cout<<i<<" elefante(s) se balanceaba(n) sobre la tela de una araña\n";
cout<<"como veía(n) que resistía(n) fueron a llamar a otro elefante\n ";
}
}
```

El código anterior emitirá por pantalla 10 veces el mensaje de 1 elefante ..... hasta 10 elefantes.....
El ejemplo anterior es muy sencillo pero nos muestra el funcionamiento de la estructura **for**.

Estructura de repetición while

Dicha estructura repite una serie de acciones mientras se cumpla una condición.

while (expresión)
{
sentencia (s);
}

La estructura while trabaja de la siguiente manera:

1. Evalúa la expresión o condición
2. Si el resultado de esta evaluación es **verdadero** la sentencia o sentencias se ejecutan, es decir, se ejecuta el cuerpo de la estructura.

Luego se vuelve a evaluar la expresión

3. Si el resultado de esta evaluación es **falso** no se ejecuta la sentencia o sentencias y sale del ciclo while.

Por **ejemplo** tengo que apagar 10 velas cuando cumpla 10 años, es decir, tengo que soplar 10 veces, entonces el problema escrito en Pseudocódigo sería:

Inicio
edad **<-** 0
mientras edad **!=** 10 años
    soplar vela
    edad = edad + 1
Fin Mientras
Fin

Ejemplo

Otro **ejemplo** que nos demostrará iteraciones con **límite conocido** es:
Tengo que mostrar la tabla de multiplicar del 9 por pantalla

```
#include <iostream>
using namespace std;
int main()
{
  int nro=1;
  while(nro <= 10 )
  {
    cout<<"9 * "<<nro<<" = "<<nro*9;
    nro++;
  }
}
```

Este programa nos mostrará lo siguiente:

```
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
```

Se dice que tiene un límite conocido porque nunca irá más allá del 10, realizará exactamente 10 iteraciones.

Ejemplo

Un ejemplo con **límite desconocido** sería invertir un número entero ingresado por teclado.

```
#include <iostream>
using namespace std;
int main()
{
int nro=0, aux=0, rpta=0;
cout<<"Ingrese un numero entero";
cin>>nro;
while(nro > 0)
      { aux = nro % 10;
        nro = nro / 10;
        rpta = (rpta * 10) + aux;
      }
cout<<"El numero invertido es: "<<rpta;
}
```

Este problema tiene un límite desconocido porque no puedo decir con exactitud el número de iteraciones que hará, eso depende del número de cifras que tenga el número ingresado por teclado. Si el número ingresado tiene 3 cifras se harán 3 iteraciones y si tiene 6 se harán 6 iteraciones. En otras palabras el límite depende de la expresión que se evalúa.

Estructura de repetición do while

Esta estructura de control es muy parecida a la estructura while lo que la hace diferente es que siempre ejecuta por lo menos una vez el cuerpo de la estructura, por eso el **do**, y luego valida una expresión y en función a este resultado vuelve a iterar o no.
La notación de esta estructura es como sigue:

```
do
{
sentencias o instrucciones
}
while ( expresión );
```

Las estructura do/while lleva punto y coma a diferencia de la estructura while.

Ejemplo

Un ejemplo para este caso es el siguiente:
Se desea ingresar por lo menos un nombre de un estudiante por teclado hasta que el usuario presione '0' para salir o cualquier otro número para continuar.

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

int main()
{
char nom[20]; //Cadena que puede contener 20 caracteres
int rpta=0;
do
{
cout<<"Ingrese el nombre de un estudiante \n";
cin>>nom;
cout<<"Desea continuar ingresando nombres: para salir '0'";
cin>>rpta;
}
while(rpta != 0); }
```

Problemas sugeridos en OMEGAUP

Ciclo for - while

- Concurso
- Asteriscos
- 1 Obtener promedio
- N Veces N
- Rango
- Repeticiones
- Mensaje de Amor
- A contar lápices
- Bolsas de dulces
- Naranjas y Limas
- ULAM
- Carrera de Larga Distancia
- Encuentra el mayor
- Acomoda el número
- Ciclo mientras no cero
- Si te da

Estructuras de Datos

Una estructura de datos es un colección de datos que se caracterizan por su forma de organización y las operaciones que se pueden definir de dicha estructura. Nosotros ya estamos familiarizados con los datos simples, como enteros (int), caracteres (char), etc. Una estructura de datos es una colección de datos simples, como por ejemplo un vector o arreglo.

La clasificación de las estructuras de datos se la puede hacer de la siguiente forma:

Arreglo o vectores

Un arreglo es un conjunto o colección finita de datos de un mismo tipo. Los elementos de un arreglo pueden ser accedidos por medio de un subíndice **i**.

Podemos considerar a un arreglo desde el punto de vista matemático como un vector, y a un arreglo bidimensional una matriz.

Declaración de un arreglo

Un arreglo se define indicando el tipo de arreglo, es decir, el tipo de datos de todos los elementos del arreglo, luego se le da un nombre al arreglo y finalmente se le da un tamaño.

<tipo> nombreArreglo**[**Tamamaño**]**

Por ejemplo tengo un arreglo de números enteros:

int arreglo[4];

En el caso anterior el tipo del arreglo es entero (**int**). Se le da una dimensión al arreglo que va entre los caracteres '[' y ']' , en el caso anterior la dimensión es 4, esto quiere decir que en la memoria se reservaron 4 posiciones para almacenar 4 valores enteros.

Inicializar un Arreglo

Existen varias maneras de inicializar un arreglo, una manera muy sencilla es poner entre llaves (**{ }**), los elementos del arreglo separados por comas.

arreglo = {51, 60, 70, 95};

También podemos utilizar la estructura de control **for** para inicializar los elementos del arreglo como se ve en el Ejemplo 7.1

Acceso a los elementos de un Arreglo

Puedo acceder a un elemento por medio de un **subíndice**, por ejemplo si yo quiero acceder al primer elemento tendré que hacerlo de esta manera

```
int nro = arreglo[0];
```

En la variable nro se almacenara el valor de 51, para acceder al segundo valor:

```
nro = arreglo[1];
```

En la variable nro se almacenará el valor de 60, y así sucesivamente con los demás elementos.

arreglo[0]	51
arreglo[1]	60
arreglo[2]	70
arreglo[3]	95

Si nos damos cuenta tener un arreglo es mucho más ventajoso que tener definidas 4 variables.

Desventajas. En ocasiones, no podemos predecir con precisión el tamaño que un arreglo tendrá por lo que podemos definir un arreglo muy grande, lo que nos lleva a desperdiciar memoria, por ejemplo si defino un arreglo de 100 elemento (arreglo[100]), y sólo utilizo 5 posiciones de memoria, las demás 95 estarán desperdiciadas, la solución a este problema la veremos más adelante en lo que se denomina punteros.

Ejemplo

Se desea ingresar las notas finales de 10 alumnos de la materia de Introducción a la programación, para luego emitir un reporte del promedio de todas las notas.

```
#include <iostream>
int main()
{

const int TAM = 10;
int i, promedio=0;
int arreglo[TAM];
for(i=0; i < TAM ; i++)
{

cout<<"Ingrese la nota del estudiante #"<<i+1<<" :\n";
cin>>arreglo[i];

}

for(i=0; i < TAM ; i++)

promedio = promedio + arreglo[i];

promedio = promedio / TAM ;
cout<<"El promedio de las notas es: "<<promedio;



}
```

No hubiera sido muy práctico manejar 10 variables diferentes para guardar las notas de los 10 alumnos, utilizando un arreglo se nos simplifican mucho las cosas.

N**ota:** Se puede declarar un arreglo dándole la dimensión o tamaño por un valor constante

const int TAM=10;
int arreglo[TAM];

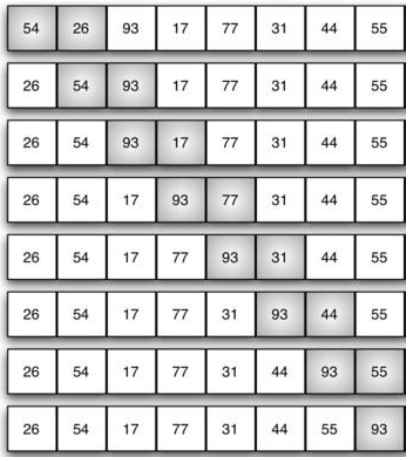
si TAM no fuera constante el programa devolvería mensaje de error.

Ordenamiento Basicos en C++

1. **Ordenación de Burbuja** (Bubble Sort en inglés)

Este es un método de ordenación elemental, que hace todas las comparaciones necesarias para colocar un elemento en la posición que le corresponde.

Dado un vector a1, a2, a3, ... an
1) Comparar a1 con a2 e intercambiarlos si a1>a2 (o a12)
2) Seguir hasta que todo se haya comparado an-1 con an
3) Repetir el proceso anterior n-1 pasadas



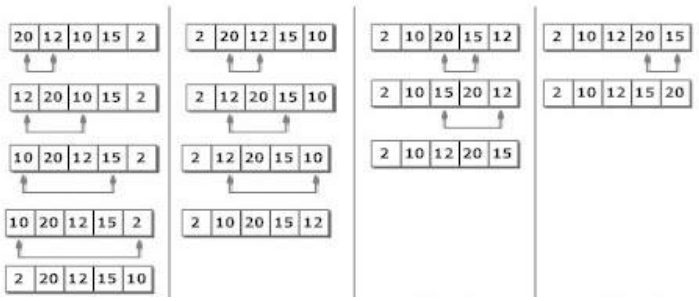

```
for (pasada=1;pasada<=n-1;pasada++) /*pasadas*/
        for(j=0; j < n-pasada; j++)
        {
          if(vec[j] > vec[j+1]) /*comparación */
           {
            /*intercambio*/
            aux=vec[j];
            vec[j]=vec[j+1];
            vec[j+1]=aux;}
            }
        }
```

2. **Procedimiento Sort**

Es una función genérica en la biblioteca estándar de C ++ para realizar una ordenación de comparación. La forma en que esto funciona es que sort internamente compara pares de enteros para decidir cuál debería ir antes que el otro.

Se incluye en la libreria <algorithm> y lleva tres argumentos :
RandomAccessIterator primero, RandomAccessIterator último, comparar comp .

```
int vec[N];
  for(int i=0;i<N;i++){
      cin >> vec[i];
                  }

  sort(vec,vec+N);
for(int i=0;i<N;i++){
      cout<<vec[i]<<" ";

         }
```



Nota: Revisar reverse(vec,vec+N);

Arreglos Bidimensionales (Matrices)

Un arreglos bidimensional esta compuesto, por un conjunto de elementos homogéneos y se puede acceder a los datos utilizando dos subíndices, este tipo de arreglo es también conocido como matriz.

Declaración

Un arreglo bidimensional se define así:

```
int arreglo[10][10];
float matriz[10][10];
```

también podemos utilizar constantes para definir la dimensión del arreglo de dos dimensiones:

```
const int N = 10;

int arreglo[N][N];
```

Inicialización

Una matriz o arreglo bidimensional se puede inicializar de este modo:

```
int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
```

Con la anterior asignación se crea en memoria una matriz igual a la de abajo

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Fig. 8.1

También podemos utilizar una estructura **for** dentro de otra estructura **for** para inicializar los valores de un arreglo de dos dimensiones como se muestra a continuación:

Ejemplo

Leer desde teclado una matriz de números enteros de dimensión 3x3.

```
#include <iostream>
int main()
{
const int TAM=3;
int matriz[TAM][TAM];
for( int i=0; i<TAM ; i++)
{
for( int j=0; j<TAM; j++)
{
cout<<"Ingrese el elemento ["<<i<<","<<j<<"] ";
cin>>matriz[I][j];
}
}
}
```

Acceso a los elementos de un arreglo bidimensional

En un arreglo de dos dimensiones necesitamos también dos índices para acceder a sus elementos.

Si utilizamos: matriz[i][j], entonces **i** se refiere a la fila y **j** a la columna.

Para acceder al elemento de la segunda fila y segunda columna de la matriz de la Fig. 8.1 hacemos:

int nro = matriz[1][1];

En la variable nro se guardara el número 5.

Las matrices o arreglos bidimensionales se suelen utilizar en cálculos matemáticos, operaciones con matrices, recorridos por matrices, y cualquier uso que nosotros le podamos dar.

Se pueden definir arreglos de más de 2 dimensiones, pero su manejo se dificultaría enormemente.

Manejo de Caracteres

Un carácter es el átomo de los programas de computadora, un carácter es un símbolo que puede ser una letra del alfabeto o un carácter especial.

Un carácter ocupa 8 bits en memoria, y existen 256 caracteres diferentes. Cada carácter se lo codifica en un número entero, es decir, que cada carácter tiene su correspondiente representación entera, el código ASCII es precisamente esto. Por ejemplo el carácter ‘a’ tiene como código ASCII el entero 97. Para ver todos códigos ASCII véase el anexo A.

Cadenas

Una cadena o cadena de caracteres nos es más que una serie de caracteres manipulados como una unidad. Si asemejamos una cadena al lenguaje castellano sería como una palabra, que es un conjunto de sílabas y vocales en donde cada una de estas viene a ser una carácter.

Visto desde otro punto vendría a ser un arreglo de caracteres.

Una cadena puede contener cualquier carácter, puede almacenar un nombre propio una dirección, es decir, lo que nosotros precisemos.

Declaración

Una cadena se la **define** de la siguiente manera

```
char cadena[20];
```

La cadena anterior puede contener un máximo de 20 caracteres.

Inicialización

Se puede inicializar una cadena de la siguiente manera:

```
cadena = "Hola" ;
```

Cualquier valor que se le asigne a una cadena va entre comillas dobles " ", como en el ejemplo anterior "Hola" esta entre comillas dobles.

Una cadena siempre finaliza con el carácter de fin de cadena ‘\0’, que siempre se añade al final automáticamente, en el ejemplo anterior se añade al final de “Hola” el carácter de fin de cadena.

También podemos considerar a una cadena como un arreglo de caracteres, y se puede inicializar de la siguiente manera:

cadena = { ‘H’, ‘o’, ‘l’, ‘a’ } ;

El arreglo de caracteres se vería de esta forma:

‘H’	‘o’	‘l’	‘a’	‘\0’
0	1	2	3	4

Nótese que en la posición 4 se aumenta el fin de cadena

Ejemplo

Se desea tener un programa que sea amable con el usuario, el programa deberá conocer el nombre del usuario y responderle con un mensaje amigable.

```
#include <iostream>

int main()
{
   char nombre[30];
   cout<<"¿Cuál es tu nombre?";
   cin>>nombre;
   cout<<"Que tengas un buen día "<<nombre;
}
```

En el ejemplo anterior el mensaje "¿Cuál es tu nombre?" es una cadena pues esta entre comillas. También es una cadena la variable nombre que recibirá un valor desde teclado.

Operaciones con Cadenas

Existen muchas operaciones que se pueden realizar utilizando cadenas, la mayoría de la operación que podemos requerir se encuentran ya a nuestra disposición dentro de la librería *string.h*

Longitud

La longitud de una cadena la podemos conocer utilizando la función strlen.

Sintaxis

strlen(cadena) ;

Ejemplo

```
#include <iostream>
#include <string.h>

int main()
{
   char nombre[30];
   int tamano;
  cout<<"¿Cuál es tu nombre?\n";
  cin>>nombre;
  tamano = strlen( nombre );
   cout<<"Tu nombre tiene "<<tamano<<”letras”;
}
```

Comparación

Para saber si dos cadenas son exactamente iguales utilizamos la función strcmp.

Sintaxis strcmp (cadena1, cadena2);

Esta función devuelve un valor de acuerdo al resultado de la comparación.

Devuelve:

0	si la dos cadenas son exactamente iguales
Mayor a 0	si la cadena1 es mayor a la cadena2
Menor a 0	si la cadena1 es menor que la cadena2

Ejemplo

```
#include <iostream>
#include <string.h>

int main()
{
   char contrasena[30], reContrasena[30];
   int resultado;
   cout<<"Escribe tu contraseña\n";
   cin>>contrasena;
   cout<<"Re escribe tu contraseña\n";
   cin>>reContrasena;
   resultado = strcmp(contrasena, reContrasena);
   if ( resultado == 0 )
      cout<<"La contraseña es aceptada";
   else

      cout<<"La contraseña no coincide";
}
```

Copia

Podemos reflejar todo el contenido de una cadena a otra, en otras palabras la copiamos tal cual, para esto utilizamos la función strcpy.

Sintaxis strcpy(cadenaDestino, cadenaOrigen);

Todo el contenido de la cadenaOrigen se copia a la cadenaDestino, si esta última tuviera algún valor este se borra.

Ejemplo

```
#include <iostream>
#include <string.h>

int main()
{
char nombre[30], apellido[30];
cout<<"¿Cuál es tu nombre? \n";
cin>>nombre;
cout<<"¿Cuál es tu apellido paterno\n";
cin>>apellido;
strcat(nombre, " "); //Se le añade un espacio en blanco
strcat(nombre, apellido);
cout<<"Tu nombre completo es "<<nombre;
}
```

Concatenación

Podemos juntar o concatenar dos cadenas una a continuación de la otra. Utilizamos la función strcat.

Sintaxis

strcat(cadenaDestino, cadenaOrigen);

Todo el contenido de la cadenaOrigen se añade a continuación de la cadenaDestino, si esta última contiene algo entonces al final contendrá lo que contenía más el contenido de la cadenaOrigen.

Ejemplo

```
#include <iostream>
#include <string.h>

void main()
{
   char origen[30], copia[30];
   cout<<"¿Qué día es hoy? \n";
   cin>>origen;
   strcpy(copia, origen);
   cout<<”Hoy es “<<copia;
}
```

Ejemplo

Escriba una función que permita conocer la longitud de una cadena. La función deberá llamarse longitud

```
#include <iostream>
#include <string.h>

int longitud(char cadena[])
{
  int acum = 0;
  while( cadena[acum] != '\0' ) //mientras no sea fin de cadena          acum++;
  return acum;
}

void main()
{
   char nombre[30];
   cout<<"¿Cuál es tu nombre?\n";
   cin>>nombre;
   cout<<"Tu nombre tiene "<<longitud(nombre)<<" letras";
}
```

Cadenas de Caracteres

Como usar cin.getline en C++

Una cadena siempre finaliza con el carácter de fin de cadena '\0', y eso es lo que se almacena; pero si deseamos guardar una gran cadena de caracteres que contenga varios espacios es conveniente utilizar la función cin.getline(); Esta función necesita tres datos o parámetros:

1. Nombre. El nombre de la variable que va a contener el string
2. Longitud. La cantidad de caracteres que queremos que se puedan introducir (nunca mayor que la longitud del string).
3. Caracter de fin. El caracter que el usuario va usar como final de la cadena. Por lo general es el 'enter' que se representa como '\n'.

Por ejemplo, supongamos que tenemos un arreglo char de 500 elementos llamado vector (nuestra string) y queremos pedirle al usuario que la "llene", la función cin.getline quedaría así:

cin.getline(vector, 500, '\n');

Como ven, los parámetros van separados por comas (,), y el caracter de fin está entre comillas simples ('). Pero bueno, dejemos de suponer y empecemos a programar.

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

int main()
{
   char vector[500];
   cout << "Introduce una frase: ";
   cin.getline(vector, 500, '\n');
   cout << "Tu frase es: " << vector;
}
```

Nota: Tomar en cuenta cin.get(); y cin.ignore();

Problemas sugeridos en OMEGAUP

Vectores

- 1 Ordena a los alumnos
- Num Mayor en arreglo
- 1 Escribir al reves
- Aprendiendo a leer cadenas

Matrices

- Matriz Diagonal
- pb Sumando
- pb Cuadrado
- Cuadro Magico UP
- Monumento
- Buscaminas
- El Misterioso Asesino de la Celda 5

Implementaciones al código

Algunas librerías ejemplo..

<iostream>	Contiene los algoritmos estándar
<algorithm>	Sort , swap, reverse
<math.h>	Pow
<string>	Strcpy, strcat, strrev, strstr, cin.getline, cin.ignore, cin.get
<iomanip>	fixed setprecision;

De no conocer todas las librerías, te recomendamos utilizar

#include <bits/stdc++.h> /// llama a todas las librerias que necesita el programa

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   long long int N;
   cin>>N;
   int vec[N];
   for(int i=0;i<N;i++){
      cin >> vec[i];
   }

   sort(vec,vec+N);
  for(int i=0;i<N;i++){
      cout<<vec[i]<<" ";  }
   }
```

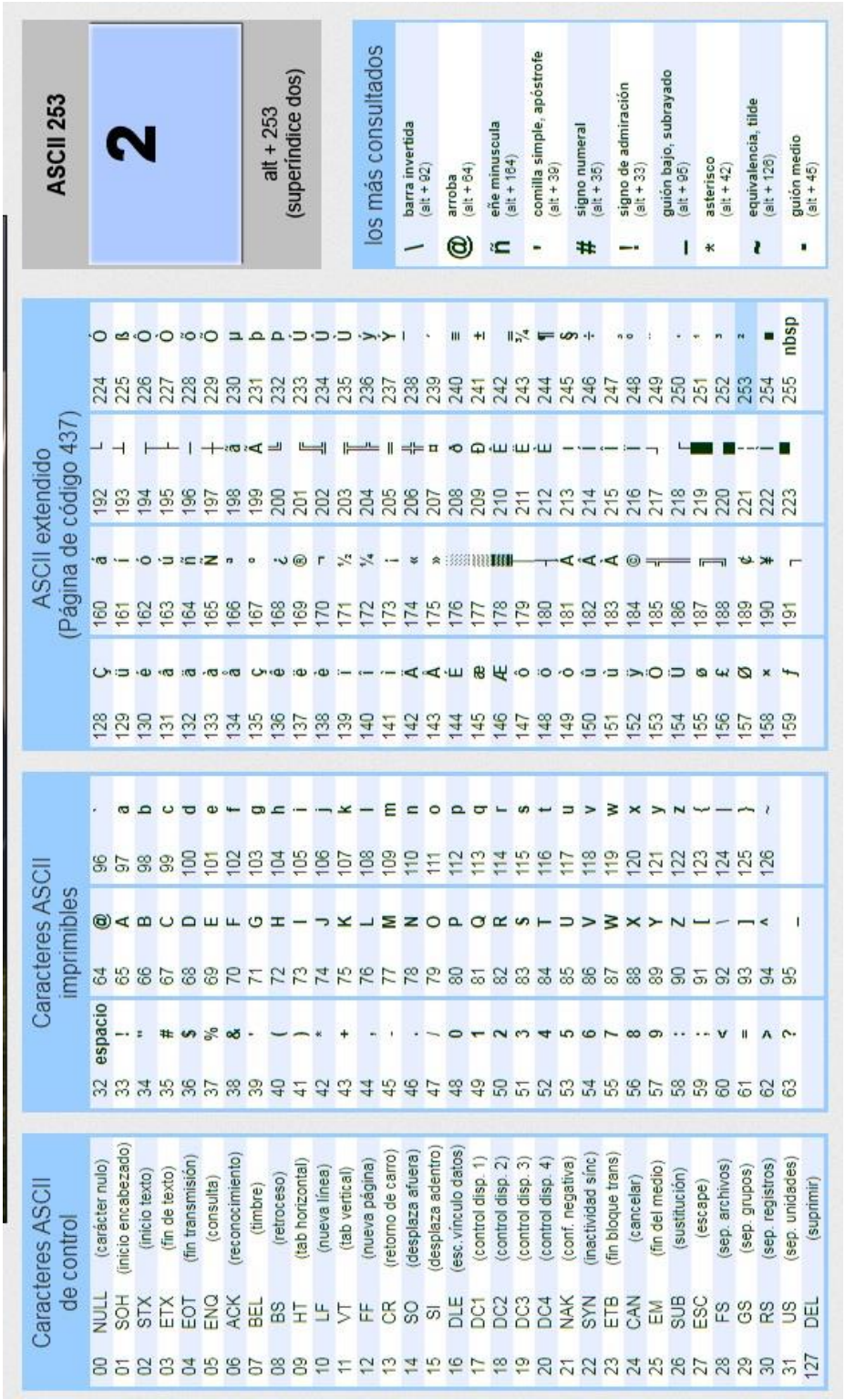
Si deseas agilizar la entrada y salida de los datos te recomendamos utilizar

```
/// acelera cin y cout
/// se debe acompañar con '\n' en cout.
ios_base::sync_with_stdio(0);
 cin.tie(0);
```

```
#include <bits/stdc++.h> /// llama a todas las librerias que necesita el programa
using namespace std;

int main()
{
   /// acelera cin y cout
   /// se debe acompañar con '\n' en cout.
   ios_base::sync_with_stdio(0);
   cin.tie(0);

   int N;
   int numeros[1000];

   cin>>N;
```

El codigo ASCII

Sigla en inglés de American Standard Code for Information Interchange (Código Estadounidense Estándar para el Intercambio de Información)



Plataformas de Practica ONLINE

omegaUp

*https://**omegaup**.com/*

omegaUp es un proyecto web enfocado a elevar el nivel de competitividad de desarrolladores de software en América Latina mediante la resolución de problemas de algoritmos, con un enfoque competitivo y divertido a la vez.

COJ Thinking Better! Caribbean Online Judge

https://**coj**.uci.cu/index.xhtml?lang=es

Juez online de programación