Estructura de datos

En programación, una **estructura de datos** es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema.

Una estructura de datos define la organización e interrelación de estos y un conjunto de operaciones que se pueden realizar sobre ellos. Las operaciones básicas son:

- Alta, adicionar un nuevo valor a la estructura.
- Baja, borrar un valor de la estructura.
- Búsqueda, encontrar un determinado valor en la estructura para realizar una operación con este valor, en forma secuencial o binario (siempre y cuando los datos estén ordenados).

Otras operaciones que se pueden realizar son:

- Ordenamiento, de los elementos pertenecientes a la estructura.
- Apareo, dadas dos estructuras originar una nueva ordenada y que contenga a las apareadas.

Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De esta forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

Tipo de dato

Tipo de dato informático es un atributo de una parte de los datos que indica al ordenador (y/o al programador) algo sobre la clase de datos sobre los que se va a procesar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar. Tipos de datos comunes son: enteros, números de coma flotante (decimales), cadenas alfanuméricas, fechas, horas, colores, etc.

Por ejemplo, por lo general el tipo "int" representa un conjunto de enteros de 32 bits cuyo rango va desde el -2.147.483.648 al 2.147.483.647, así como las operaciones que se pueden realizar con los enteros, como son la suma, la resta, y la multiplicación. Los colores, por su parte, se representan como tres bytes denotando la cantidad de rojo, verde y azul, y una cadena de caracteres representando el nombre del color; las operaciones permitidas en este caso incluyen la adición y la sustracción, pero no la multiplicación.

Éste es un concepto propio de la informática, más específicamente de los lenguajes de programación, aunque también se encuentra relacionado con nociones similares de la matemática y la lógica.

En un sentido amplio, un tipo de datos define un conjunto de valores y las operaciones sobre estos valores. Casi todos los lenguajes de programación explícitamente incluyen la notación del tipo de datos, aunque lenguajes diferentes pueden usar terminologías diferentes. La mayor parte de los lenguajes de programación permiten al programador definir tipos de datos adicionales, normalmente combinando múltiples elementos de otros tipos y definiendo las operaciones del nuevo tipo de dato. Por ejemplo, un programador puede crear un nuevo tipo de dato llamado "Persona" que específica que el dato interpretado como Persona incluirá, por ejemplo, un nombre y una fecha de nacimiento.

Un tipo de dato puede ser también visto como una limitación impuesta en la interpretación de los datos en un sistema de tipificación, describiendo la representación, la interpretación y la estructura de los valores uobjetos almacenados en la memoria del ordenador. El sistema de tipificación usa información de los tipos de datos para comprobar la verificación de los programas que acceden o manipulan los datos

Datos estructurados

La esencia de la Programación radica en lograr que la computadora resuelva un problema, utilizando un algoritmo y una técnica que depende del lenguaje a utilizar.

Variable

En programación, una variable está formada por un espacio en el sistema de almacenaje (memoria principal de un ordenador) y un nombre simbólico (un identificador) que está asociado a dicho espacio. Ese espacio contiene una cantidad o información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. El identificador, en el codigo fuente de la computadora puede estar ligado a un valor durante el tiempo de ejecucióny el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa. El concepto de variables en computación puede no corresponder directamente al concepto de variables en matemática. El valor de una variable en computación no es necesariamente parte de una ecuación o fórmula como en matemáticas. En computación una variable puede ser utilizada en un proceso repetitivo: puede asignársele un valor en un sitio, ser luego utilizada en otro, más adelante reasignársele un nuevo valor para más tarde utilizarla de la misma manera.Procedimientos de este tipo son conocidos con el nombre deiteración. En programación de computadoras, a las variables, frecuentemente se le asignan nombres largos para hacerlos relativamente descriptivas para su uso, mientras que las variables en matemáticas a menudo tienen nombres escuetos, formados por uno o dos caracteres para hacer breve en su transcripción y manipulación.

El espacio en el sistema de almacenaje puede ser referido por distintos identificadores diferentes. Esta situación es conocida entre los angloparlantes como "aliasing" y podría traducirse como "sobrenombramiento" para los hispanoparlantes. Asignarle un valor a una

variable utilizando uno de los identificadores cambiará el valor al que se puede acceder a través de los otros identificadores.

Los compiladores deben reemplazar los nombres simbólicos de las variables con la real ubicación de los datos. Mientras que el nombre, tipo y ubicación de una variable permanecen fijos, los datos almacenados en la ubicación pueden ser cambiados durante la ejecución del programa.

Las variables pueden ser de longitud:

- Fija.- Cuando el tamaño de la misma no variará a lo largo de la ejecución del programa. Todas las variables, sean del tipo que sean tienen longitud fija, salvo algunas excepciones — como las colecciones de otras variables (arrays) o las cadenas.
- Variable.- Cuando el tamaño de la misma puede variar a lo largo de la ejecución. Típicamente colecciones de datos.

Operadores Aritméticos

Los operadores aritméticos son aquellos que "manipulan" datos numéricos, tanto enteros como reales. Hay 2 tipos de operadores aritméticos: **unarios** y **binarios**. Los operadores unarios se anteponen a la expresión aritmética, y son los operadores de signo. Los operadores binarios se sitúan entre 2 expresiones aritméticas.

Operadores aritméticos unarios

Operador	Operación
+	Signo positivo
-	Signo negativo

Los operadores unarios devuelven expresiones del mismo tipo que la expresión a la que afectan.

Por ejemplo, -4 es el resultado de aplicar el operador unario - al entero 4, devolviendo otro número enterio.

Operadores aritméticos binarios

Operador	Operación
+	Suma

Operador	Operación
/	División

-	Resta		div	División entera
*	Multiplicación		mod	Módulo (resto)

Todos los operadores binarios admiten expresiones enteras y reales a excepción dediv y mod, que sólo admiten expresiones enteras, por lo que devuelven expresiones enteras. En el caso de los otros operadores, si los 2 operandos a los que afecta son enteros, la expresión resultante será entera, pero si alguno o ambos son reales, la expresión resultado es de tipo real. En esto tenemos de nuevo otra excepción: el operador / devuelve siempre expresiones de tipo real.

El operador aritmético + también se puede usar con datos de tipo string. El resultado es la concatenación de las cadenas. Por ejemplo:

```
'Esto' + 'es una' + 'concatenación de' + 'cadenas'
```

Como todos sabréis, cualquier número dividido por 0 da como resultado *infinito*. Esto se puede aceptar desde el punto de vista abstracto de las matemáticas, pero si probáis esto en un lenguaje de programación, el compilador os avisará de un error que viene a decir que la división por 0 no está permitida. Por lo tanto, el segundo argumento de los operadores /, div y mod no debe ser 0.

Como última aclaración, el operador mod devuelve el resto de realizar la división del primer operando por el segundo. Es decir, 6 mod 3 = 0 y 5 mod 2 = 1.

Instrucciones de asignación

Actualización: noviembre 2007
Las instrucciones de asignación realizan operaciones de asignación, que consisten en tomar el valor de la derecha del operador de asignación (**=**) y almacenarlo en el elemento de la izquierda, como en el ejemplo siguiente.
VB

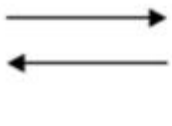
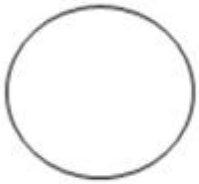
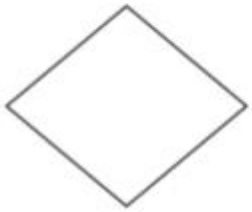
```
v = 42
```

En el ejemplo anterior, la instrucción de asignación almacena el valor literal 42 en la variable v.
Operadores de cadena

Además de los operadores de comparación, los cuales pueden ser empleados con los valores de cadena, el operador de concatenación (+) concatena dos valores de cadenas juntos, retornando otra cadena que es la unión de los dos operandos de tipo cadena. Por ejemplo, `"mi " + "cadena"` retorna la cadena`"mi cadena"`.
El operador abreviado de asignación += también puede usarse para concatenar cadenas. Por ejemplo, si la variable `micadena` tiene el valor "alfa", la expresión`micadena += "beto"` se evalúa como "alfabeto" y asigna este valor a `micadena`.

Simbolos de Diagramacion

Bueno, si leyeron la entrada anterior no es necesario que me extienda mucho, pero para esta continuacion consegui una imagen bastante explicita con ejemplos de algunos simbolos basicos que se utilizan en los diagramas de flujo, con sus respectivas caracteristicas, esto con el fin de que tengan una idea sobre se lo que se usa en un diagrama de flujo.

Símbolos	Nombre	Explicación
	Línea de flujo (Conexiones de Pasos o flechas).	Muestra la dirección y sentido del flujo del proceso, conectando los Símbolos.
	Terminador (Comienzo o final de procesos)	En su interior situamos materiales, información o acciones para comenzar el proceso o para mostrar el resultado en el final del mismo.
	Proceso (actividad)	Tarea o actividad llevada a cabo durante el Proceso. Puede tener muchas entradas, pero solo una salida.
	Conector (Conexión con Otro procesos)	Nombramos un proceso independiente que en algún momento aparece relacionado con el Proceso principal.
	Datos. Entrada/salida (Información de Apoyo)	Situamos en su interior la información necesaria para alimentar una actividad (datos para realizarla)
	Decisión (Decisión/ Bifurcación)	Indicamos puntos en que se toman decisiones: Si o no, abierto/cerrado.
	Documento	Se utiliza para hacer referencia a la gene- O consulta de un documento especifico en un punto del proceso.

pseudocódigo

En ciencias de la computación, y análisis numérico, el **pseudocódigo** (o falso lenguaje) es una descripción de alto nivel compacta e informal[1] del principio operativo de un programa informático u otro algoritmo.

Utiliza las convenciones estructurales de un lenguaje de programación real,[2] pero está diseñado para la lectura humana en lugar de la lectura mediante máquina, y con independencia de cualquier otro lenguaje de programación. Normalmente, el pseudocódigo omite detalles que no son esenciales para la comprensión humana del algoritmo, tales como declaraciones de variables, código específico del sistema y algunassubrutinas. El lenguaje de programación se complementa, donde sea conveniente, con descripciones detalladas en lenguaje natural, o con notación matemática compacta. Se utiliza pseudocódigo pues este es más fácil de entender para las personas que el código del lenguaje de programación convencional, ya que es una descripción eficiente y con un entorno independiente de los principios fundamentales de un algoritmo. Se utiliza comúnmente en los libros de texto y publicaciones científicas que se documentan varios algoritmos, y también en la planificación del desarrollo de programas informáticos, para esbozar la estructura del programa antes de realizar la efectiva codificación.

No existe una sintaxis estándar para el pseudocódigo, aunque los ocho IDE's que manejan pseudocódigo tengan su sintaxis propia. Aunque sea parecido, el pseudocódigo no debe confundirse con los programas esqueleto que incluyen código ficticio, que pueden ser compilados sin errores. Los diagramas de flujo y UML pueden ser considerados como una alternativa gráfica al pseudocódigo, aunque sean más amplios en papel.

- **asigne a x el valor de y**

$$x \leftarrow y;$$
$$x := y;$$
$$x = y;$$

Estructuras selectivas[editar]

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

$$Si\ condición\ Entonces$$
$$instrucciones;$$
$$Fin\ Si$$

Selectiva doble (alternativa)

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

$$Si\ condición\ Entonces$$
$$instrucciones_1;$$
$$Si\ no\ Entonces$$
$$instrucciones_2;$$
$$Fin\ Si$$

. Selectiva múltiple

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

$$Si\ condición_1\ Entonces$$
$$instrucciones_1;$$
$$Si\ no\ si\ condición_2\ Entonces$$
$$instrucciones_2;$$
$$Si\ no\ si\ condición_3\ Entonces$$
$$instrucciones_3;$$
$$\ldots$$
$$Si\ no\ Entonces$$
$$instrucciones_n;$$
$$Fin\ Si$$

Selectiva múltiple-Casos

Una construcción similar a la anterior (equivalente en algunos casos) es la que se muestra a continuación.

$$Según\ variable Hacer$$
$$caso\ valor_1;$$
$$instrucciones_1;$$
$$caso\ valor_2;$$
$$instrucciones_2;$$
$$caso\ valor_3;$$

$instrucciones_3;$

...

De Otro Modo

$instrucciones_n;$

Fin Según

Bucle repetir

Existen otras variantes que se derivan a partir de la anterior. La estructura de control *repetir* se utiliza cuando es necesario que el cuerpo del bucle se ejecuten al menos una vez y hasta que se cumpla la condición:

Repetir

$instrucciones;$

Hasta Que $condición$

La estructura anterior equivaldría a escribir:

$instrucciones;$

Mientras $\neg(condición)$ *Hacer*

$instrucciones;$

Fin Mientras

Bucle hacer

El Bucle hacer se utiliza para repetir un bloque de código mientras se cumpla cierta condición.

Hacer

$instrucciones;$

Mientras $condición$

Bucle para

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

la cual se define como:

$i \leftarrow x$

Mientras $i \leq n$ *Hacer*

$$
\begin{aligned}
&instrucciones;\\
&i \leftarrow i + z;\\
&Fin\ Mientras
\end{aligned}
$$

Bucle para cada

Por último, también es común usar la estructura de control *para cada*. Esta sentencia se usa cuando se tiene una lista o un conjunto L y se quiere iterar por cada uno de sus elementos:

$$
\begin{aligned}
&Para\ Cada\ x \in L\ Hacer\\
&instrucciones;\\
&Fin\ Para\ Cada
\end{aligned}
$$

Si asumimos que los elementos de L son $L_0, L_1, \ldots, L_n$, entonces esta sentencia equivaldría a:

$$
\begin{aligned}
&Para\ i \leftarrow 0\ Hasta\ n\ Con\ Paso\ 1\ Hacer\\
&x \leftarrow L_i\\
&instrucciones;\\
&Fin\ Para
\end{aligned}
$$

Que es lo mismo que:

$$
\begin{aligned}
&Para\ i \leftarrow 0\ Hasta\ n\ Hacer\\
&x \leftarrow L_i\\
&instrucciones;\\
&Fin\ Para
\end{aligned}
$$

Sin embargo, en la práctica existen mejores formas de implementar esta instrucción dependiendo del problema.

Es importante recalcar que el pseudocódigo no es un lenguaje estandarizado. Eso significa que diferentes autores podrían dar otras estructuras de control o bien usar estas mismas estructuras, pero con una notación diferente. Sin embargo, las funciones matemáticas y lógicas toman el significado usual que tienen en matemática y lógica, con las mismas expresiones.

El anidamiento

Cualquier instrucción puede ser sustituida por una estructura de control. El siguiente ejemplo muestra el pseudocódigo del ordenamiento de burbuja, que tiene varias estructuras anidadas. Este algoritmo ordena de menor a mayor los elementos de una lista L.

```
Procedimiento Ordenar (L )
    //Comentario :  L = (L_1, L_2, ..., L_n) es una lista con n el
    k ← 0;
    Repetir
        intercambio ← Falso;
        k ← k + 1;
        Para i ← 1 Hasta n − k ConPaso 1 Hacer
            Si L_i > L_{i+1} Entonces
                intercambiar (L_i, L_{i+1} )
                intercambio ← Verdadero;
            FinSi
        FinPara
    Hasta Que intercambio = Falso;
FinProcedimiento
```

En general, las estructuras anidadas se muestran indentadas, para hacer más sencilla su identificación a simple vista. En el ejemplo, además de la indentación, se ha conectado con flechas los pares de delimitadores de cada nivel de anidamiento.

Funciones y procedimientos

Muchas personas prefieren distinguir entre *funciones* y *procedimientos*. Una *función*, al igual que una función matemática, recibe uno o varios valores de *entrada* y regresa una *salida* mientras que un *procedimiento*recibe una entrada y no genera ninguna salida aunque en algún caso podría devolver resultados a través de sus parámetros de entrada si estos se han declarado por referencia (ver formas de pasar argumentos a una función o procedimiento).

En ambos casos es necesario dejar en claro cuáles son las entradas para el algoritmo, esto se hace comúnmente colocando estos valores entre paréntesis al principio o bien declarándolo explícitamente con un enunciado. En el caso de las funciones, es necesario colocar una palabra como **regresar** o **devolver** para indicar cuál es la salida generada por el algoritmo. Por ejemplo, el pseudocódigo de una función que permite calcular a^n (un número a elevado a potencia n).

```
Función potencia (a, n)
//Comentario :  Este algoritmo calcula a^n con a y n número
i ← n;
r ← 1;
```

$x \leftarrow a;$
$Mientras\ i > 0\ Hacer$
$Si\ i\ \%\ 2 \neq 0 Entonces\ //\ Si\ i\ es\ impar\ (\%\ es\ resto)$
$r \leftarrow r * x;$
$FinSi$
$x \leftarrow x * x;$
$i \leftarrow i\ /\ 2;$
$FinMientras$
$Escribir\ r;$
$FinFunción$

Un ejemplo de procedimiento seria el algoritmo de Ordenamiento de burbuja, por el que partiendo de una lista de valores estos se ordenan, nótese que en un procedimiento, no se calcula el valor de una función, sino que se realiza una acción, en este caso ordenar la lista.

$Procedimiento\ DeLaBurbuja\ (a_0, a_1, a_2, \ldots, a_{(n-1)})$
$Para\ i \leftarrow 2\ Hasta\ n\ Hacer$
$Para\ j \leftarrow 0\ Hasta\ n - i\ Hacer$
$Si\ a_{(j)} < a_{(j+1)}\ Entonces$
$aux \leftarrow a_{(j)}$
$a_{(j)} \leftarrow a_{(j+1)}$
$a_{(j+1)} \leftarrow aux$
$FinSi$
$FinPara$
$FinPara$
$FinProcedimiento$

algoritmo

En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un **algoritmo** (del griego y latín, *dixit algorithmus* y este a su vez del matemático persa Al-Juarismi[1]) es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. Los algoritmos son el objeto de estudio de la **algoritmia**.

En la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas. Algunos ejemplos son los manuales de usuario, que muestran algoritmos para usar un aparato, o las instrucciones que recibe un trabajador por parte de su patrón. Algunos ejemplos en matemática son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema lineal de ecuaciones.

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un cálculo o un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida). Sin embargo cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular. Por ejemplo, una versión modificada de la criba de Eratóstenes que nunca termine de calcular números primos no deja de ser un algoritmo.

A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos. Esto fue realizado por Alonzo Church en 1936 con el concepto de "calculabilidad efectiva" basada en su cálculo lambda y por Alan Turing basándose en la máquina de Turing. Los dos enfoques son equivalentes, en el sentido en que se pueden resolver exactamente los mismos problemas con ambos enfoques. Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o gráficas mientras que, en general, los algoritmos funcionan sobre una vasta cantidad deestructuras de datos. En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:

> **Tiempo secuencial**. Un algoritmo funciona en tiempo discretizado –paso a paso–, definiendo así una secuencia de estados "*computacionales*" por cada entrada válida (la *entrada* son los datos que se le suministran al algoritmo antes de comenzar).
>
> **Estado abstracto**. Cada estado computacional puede ser descrito formalmente utilizando una estructura de primer orden y cada algoritmo es independiente de su implementación (los algoritmos son objetos abstractos) de manera que en un algoritmo las estructuras de primer orden son invariantes bajo isomorfismo.

Exploración acotada. La transición de un estado al siguiente queda completamente determinada por una descripción fija y finita; es decir, entre cada estado y el siguiente solamente se puede tomar en cuenta una cantidad fija y limitada de términos del estado actual.

En resumen, un algoritmo es cualquier cosa que funcione paso a paso, donde cada paso se pueda describir sin ambigüedad y sin hacer referencia a una computadora en particular, y además tiene un límite fijo en cuanto a la cantidad de datos que se pueden leer/escribir en un solo paso. Esta amplia definición abarca tanto a algoritmos prácticos como aquellos que solo funcionan en teoría, por ejemplo el método de Newton y la eliminación de Gauss-Jordan funcionan, al menos en principio, con números de precisión infinita; sin embargo no es posible programar la precisión infinita en una computadora, y no por ello dejan de ser algoritmos. En particular es posible considerar una cuarta propiedad que puede ser usada para validar la tesis de Church-Turing de que toda función calculable se puede programar en una máquina de Turing (o equivalentemente, en un lenguaje de programación suficientemente general):

1. PROBLEMA: Un estudiante se encuentra en su casa (durmiendo) y debe ir a la universidad (a tomar la clase de programación!!), ¿qué debe haga el estudiante?

ALGORITMO:

Inicio
Dormir
haga 1 **hasta** que suene el despertador (o lo llame la mamá).
Mirar la hora.
¿Hay tiempo suficiente?
Si hay, **entonces**
    Bañarse.
    Vestirse.
    Desayunar.
Sino,
     Vestirse.
Cepillarse los dientes.
Despedirse de la mamá y el papá.
  ¿Hay tiempo suficiente?
Si, Caminar al paradero.
Sino, Correr al paradero.
Hasta que pase un bus para la universidad **haga** :
    Esperar el bus
    Ver a las demás personas que esperan un bus.
Tomar el bus.
Mientras no llegue a la universidad **haga** :
    Seguir en el bus.
    Pelear mentalmente con el conductor.
Timbrar.
Bajarse.
Entrar a la universidad.
Fin

Descripción de alto nivel

Dado un conjunto finito C de números, se tiene el problema de encontrar el número más grande. Sin pérdida de generalidad se puede asumir que dicho conjunto no es vacío y que sus elementos están numerados como $c_0, c_1, \ldots, c_n$.

Es decir, dado un conjunto $C = \{c_0, c_1, \ldots, c_n\}$ se pide encontrar m tal que $x \leq m$ para todo elemento x que pertenece al conjunto C.

Para encontrar el elemento máximo, se asume que el primer elemento (c_0) es el máximo; luego, se recorre el conjunto y se compara cada valor con el valor del máximo número encontrado hasta ese momento. En el caso que un elemento sea mayor que el máximo, se asigna su valor al máximo. Cuando se termina de recorrer la lista, el máximo número que se ha encontrado es el máximo de todo el conjunto.

Descripción formal

El algoritmo puede ser escrito de una manera más formal en el siguiente pseudocódigo:

Algoritmo Encontrar el máximo de un conjunto
función max(C) //C es un conjunto no vacío de números// $n \leftarrow \|C\|$ //$\|C\|$ es el número de elementos de C// $m \leftarrow c_0$ **para** $i \leftarrow 1$ **hasta** n **hacer** **si** $c_i > m$ **entonces** $m \leftarrow c_i$ **devolver** m

Implementación

En lenguaje C++:

```
int max(int c[], int n)
{
   int i, m = c[0];
   for (i = 1; i < n; i++)
      if (c[i] > m) m = c[i];
   return m;
}
```

Mas información

https://es.scribd.com/doc/4079447/9/ELEMENTOS-BASICOS-DE-UN-PROGRAMA