TEMA Nº 1. CONCEPTOS BÁSICOS, LIBRERÍA, VARIABLES, ENUNCIADOS, PROGRAMAS SIMPLES

CONCEPTOS BÁSICOS. ALGORITMOS

Un algoritmo es un conjunto definido, finito y ordenado de operaciones que se realizan para dar solución a un problema determinado. Cabe destacar que el mismo posee un estado inicial o entrada, pasos sucesivos y bien definidos denominados cuerpo y un estado final en el que se obtiene la solución.

Existen dos tipos de algoritmos:

Los naturales o sencillos que son todas las situaciones que enfrentamos cotidianamente, ejemplo: hacer una lista del mercado de la semana o el procedimiento que se hace para dirigirse del hogar al trabajo.

Los algoritmos que requieren una base lógica para ser resueltos, debido a que forman parte de programas de computadoras y arrojan resultados.

Las etapas de desarrollo de un algoritmo, con base en la lógica, son las siguientes:

1. Definición. En esta etapa se especifica el propósito del algoritmo y se ofrece una definición clara del problema por resolver. Además, aquí también se establece lo que se pretende lograr con su solución.

2. Análisis. En este punto se analiza el problema y sus características, y se determinan las entradas y salidas del problema. De igual modo, también se realiza una investigación sobre si ya se conoce alguna o varias soluciones de este. En el caso de que ya se conozcan varias soluciones, entonces se determina cuál es la más conveniente para el problema que estamos tratando. Si no se conoce ninguna, o no nos satisfacen las soluciones existentes, se propone una nueva.

3. Diseño. Aquí es donde se plasma la solución del problema. Con ese fi n, se emplea una herramienta de diseño, que consiste en el diagrama de flujo y el pseudocódigo.

4. Implementación. En este último paso es donde se realiza o se ve concretado el programa y, por ende, se hacen varias pruebas.
En cada una de las etapas especificadas antes, se utiliza un tipo de descripción conveniente e inteligible para cada uno de los participantes en el proceso de concepción y realización del algoritmo.

PROGRAMA

Se define Programa, a un conjunto de instrucciones que, una vez ejecutado, realiza una o varias tareas en una computadora. Está definido por su sintaxis, que establece e indica las reglas de escritura (la gramática), y por la semántica de los tipos de datos, instrucciones y definiciones que lo constituyen.

PARADIGMAS DEL PROGRAMADOR

La visión y los métodos de un programador en la construcción de un programa o subprograma. Existen diferentes paradigmas que derivan en múltiples y variados estilos de programación y en diferentes formas de solución de problemas:

1. Paradigma Imperativo: En este paradigma se impone que cualquier programa es una secuencia de instrucciones o comandos que se ejecutan siguiendo un orden de arriba hacia abajo; este único enlace del programa se interrumpe exclusivamente para ejecutar otros subprogramas o funciones, después de lo cual se regresa al punto de interrupción.

2. Paradigma Estructurado: Este paradigma es un caso particular de paradigma imperativo, por lo que se imponen únicamente algunas estructuras de código, prohibiendo una continuación del cálculo de manera caótica. Por ejemplo, se impone que las instrucciones sean agrupadas en bloques (procedimientos y funciones) que comunican; por tanto, el código que se repite tiene la forma de un ciclo (*loop*, en inglés), gobernado por una condición lógica.

3. Paradigma Declarativo: Un programa describe el problema a solucionar y la manera de resolverlo, pero no indica el orden de las acciones u operaciones que se deben seguir. En este caso, hay dos paradigmas principales:
   - 3.1. Paradigma Funcional: Conforme a este, todo se describe como una función.
   - 3.2. Paradigma Lógico: De acuerdo con este, todo se describe como un predicado lógico.

VARIABLES. TIPOS Y EXPRESIONES

Una variable es la opción que elige el programador para representar los datos dentro de un programa.

El nombre de una variable debe ser único y no ambiguo. La unicidad del nombre de la variable durante su ciclo de vida, asegura una semántica correcta de las operaciones (expresiones, órdenes o proposiciones) que implican a la variable.

De esta forma, el nombre de una variable es un identificador diferente de cualquier palabra clave utilizada en el lenguaje o nombre de una función externa. Generalmente, los nombres de las variables inician con una letra y son sucesiones de letras y cifras y el símbolo _ (guión bajo). Para la cualidad del programa, es preferible que el nombre de una variable sea sugestivo al tratamiento y de un largo de tamaño aceptable, ya que un nombre de variable muy largo puede generar errores de tecleo al momento de la edición del programa, lo que produce pérdidas de tiempo para su corrección.

En la determinación del nombre de la variable, también se sugiere utilizar únicamente letras sin acento, para una mejor portabilidad del código o porque la sintaxis del lenguaje no lo permite.

Algunos ejemplos de nombres de variables son los siguientes: a, a1, area, suma. También lo son: a1b159 y a2b158; sin embargo, la lectura de un programa con nombres de este tipo sería difícil. Por lo que respecta a la extensión del nombre, una variable llamada nueva_suma_valores_cantidades, tomaría mucho más tiempo escribirla.

Se considera que variables de nombre i, j, k, indican variables enteras usadas para índices; en tanto, las variables de nombre a, b y c, por lo general se utilizan para valores numéricos reales (punto flotante)

Las variables llamadas p y q se emplean para apuntadores; las variables llamadas n y m son variables que contienen valores de tamaños de arreglos.

En C existen cinco tipos de datos según puede verse en la tabla siguiente:

Tipo de dato	Descripción.
`char`	Carácter o entero pequeño (byte)
`int`	Entero
`float`	Punto flotante
`double`	Punto flotante (mayor rango que *float*)
`void`	Sin tipo (uso especial)

Algunos ejemplos de variables de C serían:
float a;
int b,c;
char caracter,otro_caracter;

OPERADORES ARITMÉTICOS

Los operadores aritméticos existentes en C son, ordenados de mayor a menor precedencia:

Operador		Operador		Operador	
++	Incremento	--	Decremento		
-	Menos unario				
*	Multiplicación.	/	División	%	Módulo
+	Suma	-	Resta		

Los operadores ++, -- y % solo pueden usarse con datos de tipo int o char. El operador incremento (++), incrementa en una unidad el valor de la variable sobre la que se aplica, el operador decremento (--), decrementa en una unidad el valor de la variable, y el operador módulo (%), calcula el resto de una división de dos variables de tipo entero o carácter.

Un aspecto que conviene explicar es el hecho de que los operadores incremento y decremento pueden preceder o posceder a su operando, lo cual permite escribir, si x es una variable de tipo int, las expresiones ++x o x++. Usado de forma aislada no presenta ninguna diferencia, sin embargo, cuando se usa en una expresión existe una diferencia en el orden de ejecución del mismo. Cuando el operador incremento (o decremento) precede al operando, C primero realiza el incremento (o decremento), y después usa el valor del operando, realizándose la operación al contrario si el operador poscede al operando.

OPERADORES RELACIÓNALES Y LÓGICOS.

Los operadores relaciónales y lógicos de C, ordenados de mayor a menor prioridad son:

Operador		Operador		Operador		Operador	
!	Not						
>	Mayor que	>=	Mayor o igual que	<	Menor que	<=	Menor o igual que
==	Igual	!=	No igual				
&&	And						
||	Or						

OPERADORES DE INCREMENTO Y DECREMENTO

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVALUACION
++	incremento	++i ó i++	1
--	decremento	--i ó i--	1

Para visualizar rápidamente la función de los operadores anteriores, se muestran las siguientes sentencias:
a = a + 1;
a++;

tienen una acción idéntica, de la misma forma que

a = a - 1;
a-- ;

Es decir, incrementa y decrementa a la variable en una unidad Si bien estos operadores se suelen emplear con variables int, pueden ser usados sin problemas con cualquier otro tipo de variable. Así si a es un float de valor 1.05, luego de hacer a++ adoptará el valor de 2.05 y de la misma manera si b es una variable del tipo char que contiene el carácter 'C' , luego de hacer b-- su valor será 'B' . Si bien las sentencias:
i++;
++i;

Son absolutamente equivalentes, en la mayoría de los casos la ubicación de los operadores incremento ó decremento indica CUANDO se realiza éste. Se puede ver el siguiente ejemplo:
int i = 1 , j , k;
j = i++;

k = ++i;
acá j es igualado al valor de i y POSTERIORMENTE a la asignación i es incrementado por lo que j será igual a 1 e i igual a 2, luego de ejecutada la sentencia . En la siguiente instrucción i se incrementa ANTES de efectuarse la asignación tomando el valor de 3, él que luego es copiado en k.

PROPOSICION IF - ELSE

Esta proposición sirve para ejecutar ciertas sentencias de programa, si una expresión resulta CIERTA ú otro grupo de sentencias, si aquella resulta FALSA. Su interpretación literal sería: SI es CIERTA tal cosa, haga tal otra, si no lo es salteéla.

El caso más sencillo sería:
If (expresión)
sentencia ;
ó
if (expresión) sentencia ;

Cuando la sentencia que sigue al IF es única, las dos formas de escritura expresadas arriba son equivalentes. La sentencia sólo se ejecutará si el resultado de "expresión" es distinto de cero (CIERTO), en caso contrario el programa salteará dicha sentencia, realizando la siguiente en su flujo.

Veamos unos ejemplos de las distintas formas que puede adoptar la "expresión" dentro de un IF : if(a > b)

if((a > b) != 0)

las dos expresiones son idénticas, aunque a veces resulta más claro expresarla de la segunda manera, sobre todo en los primeros contactos con el lenguaje.

if(a) if(a != 0)
if(!a) if(a == 0)

Las dos superiores son idénticas entre sí, al igual que las dos inferiores Obsérvese que (!a) dará un valor CIERTO sólo cuando a sea FALSO. (ver operador NEGACION en el capítulo anterior)

if(a == b)

if(a = b)	if(a == b)	La primera es una expresión correcta, el IF se realizará sólo si a es igual a b. En cambio la segunda es un error, ya que no se está comparando a con b, sino ASIGNANDO el valor de esta a aquella. Sin embargo, a veces puede usarse como un truco (un poco sucio) de programación , ya que primero se realiza la asignación y luego se evalúa el resultado de esta para realizar el IF, es entonces equivalente a escribir :
a = b ; if(a)	.....	con el ahorro de una línea de programa (a costa de la legibilidad del mismo).

En casos más complejos que los anteriores, la proposición IF puede estar seguida por un bloque de sentencias:

```
if(expresión) if(expresión) {
{ sentencia 1;
sentencia 1; sentencia 2;
sentencia 2; .....
..... }
}
```

Las dos maneras son equivalentes, por lo que la posición de la llave de apertura del bloque queda librada al gusto del programador. El indentado de las sentencias (sangría) es también optativo, pero sumamente recomendable, sobre todo para permitir la lectura de proposiciones muy complejas ó anidadas, como se verá luego. El bloque se ejecutará en su conjunto si la expresión resulta CIERTA. El uso del ELSE es optativo, y su aplicación resulta en la ejecución de una, ó una serie de sentencias en el caso de que la expresión del IF resulta FALSA.
Su aplicación puede verse en el ejemplo siguiente:
If (expresión) if (expresión)

```
{ {
sentencia 1; sentencia 1;
sentencia 2; sentencia 2;
} }

sentencia 3; else
sentencia 4; {
sentencia 5; sentencia 3;
sentencia 4;
}
sentencia 5;
```

En el ejemplo de la izquierda no se usa el ELSE y por lo tanto las sentencias 3 , 4 y 5 se ejecutan siempre . En el segundo caso , las sentencias 1 y 2 se ejecutan solo si la expresión es CIERTA , en ese caso las 3 y 4 NO se ejecutarán para saltarse directamente a la 5 , en el caso de que la expresión resulte FALSA se realizarán las 3 y 4 en lugar de las dos primeras y finalmente la 5 . La proposición ELSE queda siempre asociada al IF más cercano, arriba de él. Es común también, en caso de decisiones múltiples, el uso de anidamientos ELSE-IF de la forma indicada abajo:

```
If (exp.1) if (exp.1)
sentencia1; sentencia1;
else if(exp.2) else if(exp.2)
sentencia2; sentencia2;
else if(exp.3) else if(exp.3)
sentencia3; sentencia3;
else else
sentencia5; sentencia5;
```

Si bien se suele escribir según la modalidad de la izquierda, a la derecha hemos expresado las asociaciones entre los distintos ELSE é IF por medio del indentado del texto.