Repaso de Variables

Como se ha definido en los cursos anteriores, las variables son espacios de memoria identificados por un nombre en los que los programas escriben o leen los datos necesarios para resolver los problemas específicos para los cuales fue diseñado ese programa.

Variables, estructuras de datos u objetos de datos, son sinónimos. En todos los casos estamos hablando de los recipientes que vamos a utilizar para almacenar información. Y asi como los recipientes físicos se ajustan a lo que en ellos se pretende guardar, los recipientes de datos deben ser elegidos cuidadosamente, porque de esto depende en gran medida la complejidad del algoritmo a construir, o que un programa cumpla o no con la finalidad para la cual fue creado.

El conocimiento de las características de las variables que C nos proporciona es central para el desarrollo de programas. Por esta razón comenzamos la materia con un repaso de lo visto hasta el momento

Clasificación de variables

Las variables pueden clasificarse por el tipo de datos que contienen, por su ámbito o alcance, y por la cantidad de valores distintos que pueden almacenar al mismo tiempo.

Por tipo de dato:

La mayoría de los lenguajes de programación proporcionan un conjunto de tipos básicos, también llamados primitivos, a partir de los cuales declarar variables. En C, como ya sabemos, los tipos primitivos son:
int para números enteros
float para números reales
char para caracteres
bool para true o flase

Algunos de los tipos básicos tiene calificadores adicionales que permiten ajustar su funcionamiento a las necesidades del programa en desarrollo (sin signo, largo, etc.)

Por su alcance

De acuerdo al lugar donde son declaradas las variables se definirán como locales o globales. Como norma general, las variables son visibles dentro de las llaves en las cuales son declaradas, es decir son locales a la función que las contiene. Si se declaran fuera de toda función, serán visibles por todas las funciones incluídas en el fuente, y por lo tanto serán globales. Debido a las inconsistencia que pueden generar, debe hacerse un uso muy restrictivo de las variables globales. Sólo es aceptable su uso para establecer valores de entorno, tales como un color, una fecha, o la resolución de pantalla.

Por la cantidad de elementos que pueden almacenar

De acuerdo a la cantidad de valores que pueden almacenar al mismo tiempo, las variables pueden clasificarse como simples y compuestas.

Variables simples: puede almacenarse un solo valor en cada nombre de variable.
Ej:
a=10;
suma=a+3;

Tanto la variable **a**, como la **suma** pueden almacenar un solo valor al mismo tiempo. Si se le asigna un nuevo valor, éste eliminará al que tenía anteriormente la variable.

Variables compuestas: pueden almacenar más de un valor bajo un mismo nombre de variable. Las variables compuestas con la que hemos trabajando son los **vectores**, o arrays unidimensionales, y las **matrices** o arrays multidimensionales.

Un vector cuenta con *n* posiciones distintas para almacenar *n* valores de un mismo tipo, siendo *n* un número entero, que se debe definir al inicio del programa y que no se puede cambiar. Esto quiere decir que para usar un vector se debe conocer previamente cuántas posiciones serán necesarias.

Para leer o escribir en un vector se debe *identificar cuál es la posición* en la que se quiere leer o escribir. La posición se identifica mediante un subíndice de la siguiente manera:

v[5]=15;
v[0]=14;

En este caso se asignaron valores en las posiciones identificadas con el 5 y el 0.
Estas asignaciones son válidas para el lenguaje C, que empieza a contar desde la posición 0.

Se puede considerar a una posición específica de un vector como una variable simple. Esto es, al especificar la posición del vector se pueden realizar las mismas operaciones que se realizan con las variables simples.

Una matriz también cuenta con *n* posiciones distintas para almacenar *n* valores de un mismo tipo, pero como pueden tener 2 ó más dimensiones, se necesita un subíndice por cada dimensión, por lo que el valor *n* será un número entero que se obtiene de multiplicar el tamaño de cada una las dimensiones entre sí. Por ejemplo

int m[5][5], podrá almacenar 25 valores (5*5).

Para leer o escribir en una matriz, será necesario establecer un valor de subíndice para cada una de las dimensiones

m[0][0]=8
m[3][1]=10

Al declarar un vector o una matriz, se debe establecer la cantidad de elementos que se necesitan. Asi

int v[10], reserva 10 posiciones de memoria para almacenar enteros, y

int m[5][5], reserva 25 posiciones de memoria para almacenar enteros.

Al utilizar un vector o una matriz en C, el primer elemento se debe indicar con 0.
para el caso de int v[10], los subíndices válidos serán los números comprendidos entre el 0 y el 9, y

para el caso de int m[5][5], los subíndices válidos para ambas dimensiones serán los números comprendidos entre el 0 y el 4.