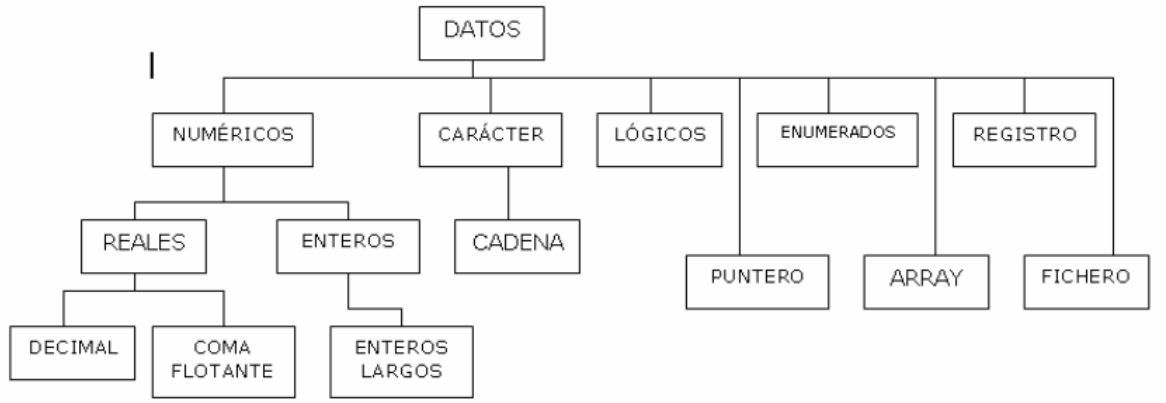
## TIPOS DE DATOS

Una clasificación de los tipos de datos existentes en los diferentes lenguajes de programación se presenta a continuación:




Por el momento nuestro interés se centrará en los siguientes tipos de datos primitivos:

- Enteros: Subconjunto finito de números enteros, cuyo tamaño o rango dependerá del lenguaje de implementación. En C existen los siguientes tipos de enteros:
  - int: entero corto, considera los valores entre -32768 a 32768 (2 bytes)
  - long: entero largo, considera los valores entre -2147483648 a 2147483648 (4 bytes)
- Reales: Subconjunto de números reales limitado no sólo en cuanto al tamaño sino también en la precisión. En C existen los siguientes tipos de reales:
  - float: real simple, considera valores entre $-3.14x10^{-38}$ a $3.14x10^{+38}$
  - double: real doble, considera valores entre $-1.7x10^{-308}$ a $1.7x10^{+308}$
- Carácter (char): Conjunto finito y ordenado de los caracteres que el sistema reconoce.
- Lógicos o Booleanos (boolean): Conjunto formado por valores “verdadero” o “falso”

## CONSTANTES Y VARIABLES

Una constante es un dato cuyo valor no cambia durante la ejecución de un programa. Según el tipo de dato las constantes se pueden clasificar en:

Constantes enteras: 12, 2003, 1522, etc.
Constantes reales: 3.1416, 2345.456, etc.
Constantes de carácter: 'a', 'B', ';', '<', '+', etc.
Constantes de cadena: "Hola", "Algoritmos Computacionales", etc.
Constantes lógicas: verdadero, falso

En C para definir una constante se utiliza la expresión:

#define <*nombre de la constante*> <*valor*>

Por su parte una variable es un dato cuyo valor puede cambiar a lo largo de la ejecución del programa. Toda variable tiene asociada dos cosas fundamentales: un identificador (nombre) único y un tipo de dato.

Para nombrar una variable en C es necesario seguir ciertas reglas:

- Pueden tener hasta 40 caracteres
- Debe empezar obligatoriamente con una letra (a-z o A-Z)
- No pueden contener espacios en blanco
- El resto de los dígitos pueden ser números
- Se pueden incluir caracteres especiales como el guión o el punto.

Ejemplos de nombres válidos de variables:

FechaNueva
C1
totalGuaranies
CONTADOR-5
H123
cantidad_de_Alumnos
Pedido.Almacen

Ejemplos de nombres NO válidos de variables:

Fecha nueva
1contador
24ABC
primer-valor N

## INSTRUCCIÓN DE ASIGNACIÓN

La forma general de esta instrucción es la siguiente:

*<variable> = <valor>*

El resultado de evaluar la expresión de la derecha se le asigna o se le da como valor a la variable que se encuentra a la izquierda.

Así por ejemplo al tener *interes = 0.5* significa que a la variable *interes* se le da un valor de *0.5* y cualquier valor anterior que pudiera tener la variable se borra.

## OPERADORES ARITMÉTICOS

Una expresión aritmética es un conjunto de variables, constantes y/o funciones unidas o relacionadas mediante paréntesis y operadores aritméticos. Los operadores aritméticos usados en el lenguaje C son:

| Operador | Significado |
|---|---|
| + | suma |
| - | resta |
| * | multiplicación |
| / | división |
| ++ | incremento |
| -- | decremento |
| mod | residuo de división entera |

Algunas consideraciones que deben tenerse en cuanta al usar operadores son:

- Si en una operación ambos operandos son enteros, entonces el resultado de la operación es un entero.
- Si en una operación uno o ambos operandos son reales, entonces el resultado de la operación es un real.
- El operador "/" produce un cociente entero sólo si los dos operandos son enteros. Esto significa que si la división no es exacta se pierde la parte decimal.
- El operador "/" produce un cociente real si uno o los dos operandos son reales.
- En los lenguajes de programación no se sobreentiende ningún símbolo. Es común en álgebra al encontrar la expresión: *AB*, entender que se pide realizar la operación de multiplicar el valor de la variable *A* por el valor de la variable *B* (*A* x *B*). En lenguaje de programación si se encuentra *AB*, se están refiriendo a un nombre de variable *AB*. Se debe colocar *A*B*.
- Igualmente *A(B+C)* se entiende en álgebra como *A X (B + C)*, en lenguaje de programación se debe colocar *A*(B+C)*
- No se pueden colocar dos operadores aritméticos seguidos, es decir que en vez de poner *A * – B* se debe colocar *A * ( - B )* colocando una de las variables entre paréntesis

Otro aspecto importante de las expresiones aritméticas es lo que se conoce como precedencia de los operadores y se refiere al orden en que se debe evaluar la expresión cuando aparecen dos o más operaciones aritméticas. En general los lenguajes de programación coinciden en evaluar primero lo que se encuentra entre paréntesis comenzando por los paréntesis más internos. Dentro de cada paréntesis o en una expresión libre de paréntesis es común evaluar de la siguiente forma:

De izquierda a derecha se hace un primer recorrido por la expresión y se evalúan:
- Funciones especiales (raíz cuadrada, valor absoluto, logaritmo, seno, coseno, tangente elevar a potencia, etcétera)
- Multiplicaciones (*) y/o divisiones (/) y/o división entera (%).
- En un tercer recorrido se evalúan también de izquierda a derecha sumas (+) y/o restas (-).

Ejemplo: Dado que al ejecutar un algoritmo este trabaja en secuencia, determinar los valores finales de las variables suponiendo que se tienen las siguientes instrucciones:

```
1       i = 3
2       j = 5
3       k = i + j
4       i = i + 1
5       m = k * 3 + i
6       n = k * (3 + i)
```

Resultado:

| Línea | i | j | k | m | n |
|---|---|---|---|---|---|
| **1** | 3 | | | | |
| **2** | | 5 | | | |
| **3** | | | 8 | | |
| **4** | 4 | | | | |
| **5** | | | | 28 | |
| **6** | | | | | 56 |

Es importante mirar como se comporta la instrucción número cuatro. En este caso la expresión a la derecha es: *i + 1*, de la instrucción uno *i* tenía un valor de 3, a ese valor se le adiciona 1 y el resultado de la expresión que es 4 se lleva a la variable de la izquierda que es *i* (el valor anterior de *i* es borrado automáticamente).

También se debe notar la diferencia de resultado entre las instrucciones cinco y seis, el orden de evaluación cambia cuando existe el paréntesis, por esto dan valores diferentes.

## INSTRUCCIONES DE LECTURA Y ESCRITURA

El valor de una variable se puede obtener mediante una instrucción de lectura, con la cual el computador espera a que por medio del teclado se ingrese un valor y luego se presione la tecla ENTER. En el lenguaje C dicha instrucción de lectura se escribe:

```
cin>>a;
```

En este caso cuando se ingrese un valor por teclado y se presione ENTER, el valor leído se asignará a la variable *a* de acuerdo al tipo de dato del cual se haya definido.

Por otro lado, cuando se desee mostrar el valor de una variable en la pantalla se utiliza la instrucción:

```
cout<<a;
```

En este caso en la pantalla de ejecución se mostrará el valor que en ese momento tenga la variable *a*.

La instrucción *cout* también puede usarse para escribir cualquier texto que se desee aparezca en la pantalla de ejecución, para lo cual simplemente debe colocarse dicho texto entre comillas dobles de la siguiente manera:

cout<<”Hola, este es mi primer programa”;

Incluso el *cout* puede usarse combinando texto con valores de variables, de la siguiente manera:

cout<<”El valor de la variable a es: “<<a;

Importante: Para poder usar las instrucciones *cin* y *cout* del lenguaje C es necesario que en el código se incluya la librería *iostream.h*

## ESQUEMA GENERAL DE UN ALGORITMO EN C

Con los elementos aprendidos hasta este momento en el curso es posible empezar a desarrollar algoritmos (por el momento sencillos), para lo cual debe tenerse siempre en cuenta que cumplan con el siguiente esquema:

1. incluir librerias
2. definir constantes
3. abrir void main
4. definir variables (entrada, salida, intermedias)
5. leer variables de entrada
6. realizar cálculos
7. imprimir resultados
8. cerrar void main

Los puntos 1,3 y 7 son absolutamente necesarios para todo algoritmo por lo que siempre deberán estar presentes, mientras que los puntos del 2 y 4 al 7 dependerán del problema que se este afrontando. En cualquiera de los casos se sugiere chequear que el algoritmo que se implemente cumpla con esta estructura.

**Ejemplo:** utilizando todos los elementos aprendidos hasta este momento en el curso diseñe un algoritmo para calcular el perímetro de cualquier círculo y luego escriba el código correspondiente en lenguaje C.

Solución:

<u>Análisis del problema:</u>

Datos de entrada: Radio del círculo
Datos de salida: Perímetro del círculo

Diseño del algoritmo:

*INICIO*
1. *Hacer* PI = 3.15159
2. *Leer* Radio
3. *Hacer* Perimetro = 2 * PI * Radio
4. *Imprimir* Perimetro

*FIN*

Verificación del algoritmo:

*INICIO*
1. PI = 3.15159
2. Si Radio = 10
3. Perimetro = 2 * PI * Radio = 2 * 3.14159 * 10
4. 62.8318

*FIN*

Implementación en C:

```
#include <iostream.h>
#define PI 3.14159

void main()
{
float radio, perimetro;

cout<<"Ingrese el radio del círculo: ";
cin>>radio;

perimetro = 2*PI*radio;

cout<<"\nEl perímetro del circulo es: "<<perimetro;
}
```