ICF 122 - Fundamentos de Programación
Ingeniería Civil Informática
Unidad 2: Fundamentos del lenguaje

Facultad de Ingeniería

Universidad Andrés Bello

Tipo de Datos y Variables

- Un **tipo de dato** define de forma explícita un conjunto de valores denominado **dominio**.
- Un **valor** es un elemento del conjunto dominio.
- Una **variable** es un espacio de memoria destinada al almacenamiento de un valor del tipo de dato, referenciada por un nombre.
- Cada lenguaje de programación ofrece una colección de tipos de datos llamados **primitivos**.

¿Cómo se crea una variable en C?

⟶ Toda variable debe ser declarada, previo a su uso.

⟶ Al **declarar una variable**, se indica al programa un **identificador** o nombre para esa variable y su **tipo de dato**.

```
tipo_d var_1;
tipo_d var_1 = valor_1;
tipo_d var_1 = valor_1, ... , var_N = valor_N;
```

⟶ `tipo_d` es el tipo de dato de la variable que se quiere crear.

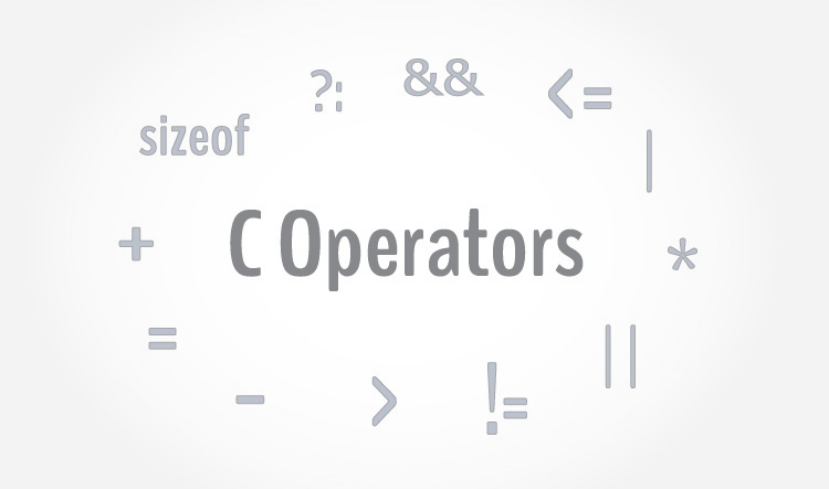
⟶ `var_1` es el nombre o identificador de la variable.

Tipo	Tamaño (bytes)	Rango de valores	
		Mínimo	Máximo
`signed char`	1	-128	127
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`unsigned short int`	2	0	65535
`signed long int`	4	-2147483648	2147483647
`unsigned long int`	4	0	4294967296
`float`	4	-3.4E+38	3.4E+38
`double`	8	-1.7E+308	1.7E+308
`long double`	10	-1.2E+4932	1.2E+4932

Tipos de datos primitivos

Operadores

- Los operadores pueden aplicarse a una sola variable, a dos variables e incluso a varias variables.
- Se denomina operación **unaria** a la que se aplica a una sola variable. Una operación unaria es, por ejemplo, el signo del valor.
- Variables `int` y `char`: signo, asignación, aritméticos, relacionales, lógicos y a nivel de bit.
- Variables `float` y `double`: asignación, artiméticos (excepto módulo), relacionales y lógicos.



- El operador **asignación** permite modificar los valores de las variables y alterar, por lo tanto, el estado de la memoria del computador. El caracter que lo representa es =.
- La forma general de este operador es:

```
nombre_variable = expresion;
```

- `expresion` es una secuencia de operandos y operadores que, unidos según ciertas reglas, producen un resultado.

Ejemplo 1

Sea la expresión

```
a = a + 1;
```

El procesador toma el valor de la variable `a`*, lo copia en un registro de la ALU donde se incrementa en una unidad, y almacena (asigna) el valor resultante en la variable* `a`*, modificando por ello el valor anterior de esa posición de memoria.*

- El operador **asignación** tiene dos extremos:
  - Izquierdo: **left value**.
  - Derecho: **right value**.
- El **left value** solamente puede ser el nombre de una variable y nunca una expresión o un literal.
- Expresiones erróneas: `a + b = c;` `3 = a;` `a + b = 9;`
- Asignación de valores enteros $\longrightarrow$ `z = 3;` (base 10)
- Asignación en hexadecimal $\longrightarrow$ `z = 0x20;` corresponde al número binario 100000 y al número decimal 32.

Tipos de datos primitivos

Operadores

Los operadores aritméticos son:

- **Suma**: su identificador es + y se aplica sobre cualquier variable primitiva de C. Si se aplica de forma unaria, entonces es el operador de signo positivo.
- **Resta**: su identificador es - y se aplica sobre cualquier variable primitiva de C. Si se aplica de forma unaria, entonces es el operador de signo negativo.

Ejemplo 2

- *Operador suma:*

```
y = y + 2;
b = 1.2E+3;
```

- *Operador resta:*

```
z = x - 9;
c = 3.4E-6;
```

- **Producto**: su identificador es `*` y se aplica sobre cualquier variable primitiva de C.
- **Cociente o División**: su identificador es `/` y se aplica sobre cualquier variable primitiva de C. Cuando se aplica a variables enteras, el resultado también será entero. El divisor no puede ser 0.
  ¿Qué sucede con la expresión `sup = (1 / 2)*b*h;`?

Ejemplo 3

- *Operador producto:*

```
z = 2 * (x + y);
b = 3 * b;
```

- *Operador cociente:*

```
z = x / 9;
c = a + (b / 6);
```

- **Módulo**: su identificador es % y calcula el resto del cociente entero.
- Por su definición, no se aplica a variables no enteras.
- Su divisor no puede ser 0.

Ejemplo 4

```
x = 15 % 4; (x = 3)

z = 24 % 9; (z = 6)

b = 35 % 7; (b = 0)
```

- **Incremento y decremento**: su identificadores son ++ y -- respectivamente y son válidos para todos los tipos de datos primitivos de C.
- La expresión `a++;` es equivalente a la expresión `a = a + 1;` y la expresión `a--;` es equivalente a la expresión `a = a - 1;`
- Estos operadores condensan, en uno sola expresión, un operador asignación, un operador suma (o resta) y un valor literal: el valor 1. Y como se puede apreciar son operadores unarios: se aplican a una sola variable.
- La ubicación del operador, con respecto a la variable, determina el comportamiento dentro de la expresión.

```
unsigned short int a, b = 2, c = 5;
a = b + c++;
```

si el operador precede a la variable, se ejecuta antes que el resto de la expresión, pero si está después, se evalúa la expresión y luego se aplica el operador. En este caso `a = 7` y `c = 6`. Si la expresión hubiera sido `a = b + ++c;`, entonces `a = 8` y `c = 6`.

Tipos de datos primitivos

Operadores

- Los operadores **relacionales** y **lógicos**, generan expresiones que se pueden evaluar como **verdaderas** o **falsas**.
- En C, el tipo de dato **booleano** no existe, es decir, no hay un tipo de dato para el valor verdadero o falso.
- Cualquier expresión falsa se evalúa con valor 0, y verdadera cualquier otro valor.
- Los operadores relacionales son seis:

Operadores relacionales		**Ejemplo**
igual que	`==`	`x == y;`
distintos	`!=`	`a != 0;`
mayor que	`>`	`x + y > 3;`
mayor o igual que	`>=`	`a + b + c >= 1;`
menor que	`<`	`x + y < a + b;`
menor o igual que	`<=`	`z <= 9;`

- Los operadores lógicos son tres:

Operadores lógicos				
a	b	a && b	a || b	!a
F	F	F	F	V
F	V	F	V	V
V	F	F	V	F
V	V	V	V	F

Supongamos el siguiente código:

```
int a = 1 , b = 3 , x = 30 , y = 10;
int resultado;
resultado = a * x == b * y;
```

El valor de la variable `resultado` sería 1.

Tipos de datos primitivos

Operadores

Evaluar las siguientes expresiones:

```
short a = 0, b = 1, c = 5;
a;
b;
a < b;
5 * (a + b) == c;
```

```
float pi = 3.141596;
long x = 0, y = 100, z =1234;
3 * pi < y && (x + y) * 10 <= z / 2;
3 * pi < y || (x + y) * 10 <= z / 2;
3 * pi < y && !((x + y) * 10 <= z / 2);
```

```
long a = 5, b = 25, c = 125, d = 625;
5 * a == b;
5 * b == c;
a + b + c + d < 1000;
a > b || a = 10;
```

- Los operadores **a nivel de bit** están definidos únicamente para las variables de tipo entero. No se puede aplicar sobre variables de tipo `float`, `double` y `long double`.
- Operador **AND a nivel de bit**: su identificador es `&`, y se aplica sobre variables del mismo tipo, con la misma longitud de bits. Compara bit a bit ambas variables, teniendo como resultado 1, si ambos bits son 1, en caso contrario el resultado es 0.

```
  11001000
& 10111000
  --------
= 10001000
```

- Operador **OR a nivel de bit**: su identificador es |, y se aplica sobre variables del mismo tipo, con la misma longitud de bits. Compara bit a bit ambas variables, teniendo como resultado 1, si alguno de los dos bits es 1, en caso contrario el resultado es 0.

```
  11001000
| 10111000
  --------
= 11111000
```

- Operador **OR EXCLUSIVO (XOR) a nivel de bit**: su identificador es ^, y se aplica sobre variables del mismo tipo, con la misma longitud de bits. Compara bit a bit ambas variables, teniendo como resultado 1, si ambos bits son distintos, en caso contrario el resultado es 0.

```
  11001000
^ 10111000
  --------
= 01110000
```

- Operador **complemento uno**: su identificador es ~, y es un operador unario. Retorna el complemento a uno del valor de la variable.

```
~ 11001000
  --------
= 00110111
```

- Operador **desplazamiento a la izquierda**: su identificador es <<. Es un operador binario que realiza un desplazamiento hacia la izquierda de tantos bits como se indique. Los bits más significativos se pierden.

```
short int var1 = 0x7654;
short int var2 = var1 << 3;
```

En binario `var1 = 0111 0110 0101 0100 = 7654`, luego se desplazan 3 bits hacia la izquierda, por lo tanto, `var2 = 1011 0010 1010 0000 = B2A0`

- Operador **desplazamiento a la derecha**: su identificador es >>. Es un operador binario que realiza un desplazamiento hacia la derecha de tantos bits como se indique. Los bits menos significativos se pierden.

```
short int var1 = 231;
short int var2 = var1 >> 1;
```

En binario `var1 = 0000 0000 1110 0111 = 231`, luego se desplaza 1 bit hacia la derecha, por lo tanto, `var2 = 0000 0000 0111 0011 = 115`

Tipos de datos primitivos

Operadores

- Cuando una variable situada al lado izquierdo de una asignación, también se encuentra la lado derecho, se puede utilizar los **operadores compuestos**.
- Todos los operadores aritméticos y los operadores a nivel de bit binarios (exceptuando, por tanto, los operadores de signo, incremento, decremento y complemento) pueden presentar otra forma.

Operador	Expresión	Equivale a
+=	`x += y`	`x = x + y`
-=	`x -= y`	`x = x - y`
*=	`x *= y`	`x = x * y`
/=	`x /= y`	`x = x / y`
%=	`x%= y`	`x = x% y`
>>=	`x >>= y`	`x = x >> y`
<<=	`x <<= y`	`x = x << y`
&=	`x &= y`	`x = x & y`
|=	`x |= y`	`x = x | y`
^=	`x ^= y`	`x = x ^ y`

- El operador `sizeof` devuelve la cantidad de bytes que ocupa una variable o un tipo de dato.
- El valor retornado es tomado como entero y puede estar presente en cualquier expresión.
- Su sintaxis es:

```
sizeof(nombre_variable);

sizeof(nombre_tipo_de_dato);
```

Ejemplo 5

```
sizeof(int);
sizeof(char);
sizeof(float);
sizeof(double);
```

- Existe un operador unario que permite forzar un cambio de tipo a una variable.
- Este cambia se denomina **cambio por promoción** o **casting**.
- La sintaxis es:

  `(tipo)nombre_variable;`
- El operador de casting, precede a la variable. Se escribe entre paréntesis el tipo de dato al cual se quiere forzar la variable su cambio.

Ejemplo 6

```
int x = 3, y = 9;
(float)x; (x pasa a tener valor 3.0)
(double)y; (y pasa a tener valor 9.0)
```

Tipos de datos primitivos

Constantes

- Se puede crear una variable, donde su valor inicial no se debe modificar en el programa.
- Se utiliza la palabra reservada `const`:

```
const tipo_dato var1 = val_1;
```

- Se puede definir cualquier constante de los diferentes tipos de datos

```
const float DOS_PI = 6.28;
```

- Otra forma de definir constantes es mediante la directiva de compilación `#define`

```
#define DOS_PI 6.28
```

Entrada y Salida de datos

- La función `printf` permite la salida de datos
- Su sintaxis es:

```
printf("%<tipo_dato>",<var>);
```

  donde <tipo_dato> es un identificador que depende del tipo de dato de la variable y <var>, es el nombre de la variable.

- Por ejemplo, si la variable a es entera:

```
printf("El valor de a es: %d\n",a);
```

Entrada y Salida de datos

Especificador	Detalle
%d	Entero con signo, base decimal.
%i	Entero con signo, base decimal.
%u	Entero sin signo, base decimal.
%o	Entero (con o sin signo), base octal.
%x	Entero (con o sin signo), base hexadecimal.
%f	Número real con signo.
%e	Número real con signo en notación científica.
%c	Un caracter.
%s	Cadena de caracteres.
%p	Dirección de memoria.

Tabla 1: Especificadores de tipo de datos.

Entrada y Salida de datos

Algunos ejemplos:

```
long int a = 123;
printf("La variable a vale ... %06li.\n",a);

La variable a vale ... 000123.
```

```
double raiz_2 = sqrt(2);
printf("A. Raiz de dos vale %lf\n",raiz_2);
printf("B. Raiz de dos vale %12.1lf\n",raiz_2);
printf("C. Raiz de dos vale %12.3lf\n",raiz_2);

A. Raiz de dos vale 1.414214
B. Raiz de dos vale          1.4
C. Raiz de dos vale        1.414
```

Entrada y Salida de datos

- La función `scanf` permite la entrada de datos
- Su sintaxis es:

```
scanf("%<tipo_dato>",&<var>);
```

donde `<tipo_dato>` es un identificador que depende del tipo de dato de la variable y `<var>`, es el nombre de la variable. El operador unario `&` indica la dirección de memoria de la variable.

```
scanf("%f\n",&b);
```

El valor ingresado por teclado, queda almacenado en la variable `b`, que es de tipo `float`.

- Se utilizan los mismos especificadores de la Tabla 1.
- Tanto la función `scanf` como `printf` están declaradas en el archivo de cabecera `stdio.h`