# Instituto Politécnico Nacional

## Escuela Superior de Computo

Fundamentos De Programación

Pérez García Edgar Israel.

Variables y operadores.

16 de septiembre de 2025.

## Introducción

La programación, en su esencia, es el arte de dar instrucciones a una computadora para que realice tareas específicas. Para lograrlo, los programadores utilizan lenguajes que permiten comunicarse con el hardware de la máquina. Entre estos lenguajes, el lenguaje de programación C se ha consolidado como una herramienta fundamental y atemporal. Desarrollado entre 1969 y 1973 por Dennis Ritchie en los Laboratorios Bell, C surgió como un lenguaje de propósito general con el objetivo de crear el sistema operativo UNIX. Su diseño se centró en la eficiencia, la portabilidad y la capacidad de manipulación directa de la memoria, lo que lo convirtió en una elección ideal para el desarrollo de sistemas operativos, compiladores y software embebido.

El paradigma de programación estructurada, en el cual C se enmarca de forma natural, revolucionó la forma en que se escriben los programas. Antes de su surgimiento, el código era a menudo desorganizado, con saltos incondicionales (goto) que creaban una maraña de instrucciones difíciles de seguir, un fenómeno conocido como "código espagueti". La programación estructurada introdujo la idea de organizar el código en estructuras de control lógicas y predecibles, como la secuencia, la selección (condicionales if/else) y la repetición (bucles for, while, do/while). Esta metodología mejoró drásticamente la legibilidad, mantenibilidad y depuración del código, permitiendo la creación de programas más complejos y robustos.

Dentro de este marco, dos conceptos son el pilar de cualquier programa en C: las variables y los operadores. Las variables son contenedores con nombre que se utilizan para almacenar datos en la memoria de la computadora. Sirven como un espacio de almacenamiento temporal para la información que el programa necesita procesar, ya sean números, caracteres o valores lógicos. Cada variable debe tener un tipo de dato asociado, el cual determina el tipo de información que puede almacenar y la cantidad de memoria que ocupará. Esta estricta tipificación es una característica fundamental de C, que contribuye a la eficiencia y seguridad del lenguaje.

Por otro lado, los operadores son símbolos especiales que permiten realizar operaciones con las variables y los valores. Son las herramientas que nos permiten manipular los datos, realizar cálculos matemáticos, comparar valores, asignar resultados y controlar el flujo del programa. La combinación de variables y operadores es lo que permite a un programa de C tomar datos, procesarlos y producir un resultado significativo. La correcta comprensión y uso de estos elementos es el primer paso para dominar el lenguaje C y, por extensión, para entender los fundamentos de la programación en un nivel más profundo.

A continuación, se presenta una tabla que resume los tipos de variables más comunes en C y otra que detalla los diferentes tipos de operadores.

| Tipo de Dato | Tamaño (bits) | Rango de Valores | Uso Común |
|---|---|---|---|
| char | 8 | -128 a 127 o 0 a 255 | Almacenar caracteres individuales |
| int | 16 o 32 | Depende de la arquitectura | Almacenar números enteros |
| float | 32 | Aprox. ±3.4×10−38 a ±3.4×1038 | Almacenar números de punto flotante de precisión simple |
| double | 64 | Aprox. ±1.7×10−308 a ±1.7×10308 | Almacenar números de punto flotante de precisión doble |
| long | 32 o 64 | Depende de la arquitectura | Almacenar números enteros grandes |
| short | 16 | -32,768 a 32,767 | Almacenar números enteros pequeños |

## Tipos de operadores.

| Categoría | Símbolos de Ejemplo | Descripción |
|---|---|---|
| Aritméticos | +, -, *, /, % | Realizan operaciones matemáticas básicas |
| Relacionales | #ERROR! | Comparan dos valores y devuelven un valor booleano (verdadero/falso) |
| Lógicos | &&, \|\|, ! | Combinan o niegan expresiones booleanas |
| De Asignación | #ERROR! | Asignan un valor a una variable |
| De Incremento/Decremento | ++, -- | Aumentan o disminuyen el valor de una variable en 1 |
| A nivel de Bits | &, \|, ^, ~, <<, >> | Realizan operaciones directamente sobre los bits de un número |
| Otros | sizeof, &, *, ? : | Operador de tamaño, dirección, desreferencia y ternario, respectivamente |

## Desarrollo.

1. Realice un programa que se llame **aleatoriocaracter.c** para generar letras o caracteres aleatorios (dados los ejemplos de las figuras 1 y 2).

```c
#include <stdio.h>
// las dos librerias necesarias
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
    int numeroUno, numeroDos;
    int limite = 100;
    // Crea la semilla de los numeros aleatorios
    srand(time(NULL));
    // Genera dos numeros aleatorios
    numeroUno = rand() % limite;
    numeroDos = rand() % limite;
    // imprime los dos numeros aleatoorios
    printf("numeroUno = %d\n", numeroUno);
    printf("numeroDos = %d\n", numeroDos);
    return 0;
}
```

**Figura 1** Programa que genera dos numeros enteros aleatorios (aleatorioentero.c).

```c
#include <stdio.h>
// las dos librerias necesarias
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
    float numeroFloatUno, numeroFloatDos;
    int numeroUno, numeroDos;
    int numeroTres, numeroCuatro;
    int limite = 100;
    // Crea la semilla de los numeros aleatorios
    srand(time(NULL));
    // Genera dos numeros aleatorios
    numeroUno = rand() % limite;
    numeroDos = rand() % limite;
    numeroTres = rand() % limite;
    numeroCuatro = rand() % limite;
    numeroFloatUno = (float)numeroUno + numeroTres / 100.0;
    numeroFloatDos = (float)numeroDos + numeroCuatro / 100.0;
    // imprime los cuatro numeros aleatorios enteros
    printf("numeroUno    = %d\n", numeroUno);
    printf("numeroDos    = %d\n", numeroDos);
    printf("numeroTres   = %d\n", numeroTres);
    printf("numeroCuatro = %d\n", numeroCuatro);
    // imprime los dos numeros flotantes aleatorios
    printf("numeroFloatUno = %f\n", numeroFloatUno);
    printf("numeroFloatDos = %f\n", numeroFloatDos);
    return 0;
}
```

**Figura 2** Generador de numeros aleatorios.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (aleatoriocaracter.c)***

2. Del código mostrado en la figura 3, cópielo y sálvelo con el nombre del programa 1. Vea como trabaja y realice el mismo procedimiento para todos los tipos de datos.

| Programa | Código fuente | Tipo VS todos los operadores |
|---|---|---|
| 1 | operadorconchar.c | char para cx y cy |
| 2 | operadorconshort.c | short para cx y cy |
| 3 | operadorconint.c | int para cx y cy |
| 4 | operadorconlong.c | long para cx, cy |
| 5 | operadorconfloat.c | float cx y cy |
| 6 | operadorcondouble.c | double para cx y cy |

Incluya el código necesario que le de a las variables de entrada su valor aleatorio. Vea el ejemplo de la figura 3, recuerde que le falta código, no está completo porque no muestra la salida de cada operación con cada operador, eso le toca a usted. Solo se muestra para que vea como se asignan los valores aleatorios a las variables con las que se hará todo el proceso con cara uno de los operadores. Recuerde que sus variables deben estar declaradas al comienzo después de declarar la función principal y su llave de inicio.

***Figura 3 anexa en la carpeta que contiene este documento cómo: figura3.c***

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorconchar.c)***

Se detectó que en el código proporcionado tenía números (4, 5, 6) los cuales impedían que el código compilara. Se eliminaron para que compilara.

Se detectó que se estaba usando **unsigned char** y solo iba a dar vuelta por la derecha, se cambió a **signed char**.

Por último en el código original solo operaba con dos variables tipo **char** y no complía con la instrucción de usar números enteros y flotantes tan grandes cómo se pueda. Se agregó una sección final llamada "DEMOSTRACION ADICIONAL CON VALORES GRANDES", donde se opera un **char** con un **int** para demostrar de manera más explícita las “vueltas” que da el valor.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorconshort.c)***

Se repiten los mismos errores de sintaxis del código proporcionado.

A su vez estaba diseñado para operar con **char** por lo que se cambió a **short**.

Se agregó el mismo bloque final para demostrar el número de vueltas.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorconint.c)***

Se repiten los mismos errores de sintaxis del código proporcionado.

A su vez estaba diseñado para operar con **char** por lo que se cambió a **int**.

Se agregó el mismo bloque final para demostrar el número de vueltas.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorconlong.c)***

Se repiten los mismos errores de sintaxis del código proporcionado.

A su vez estaba diseñado para operar con **char** por lo que se cambió a **long**.

Se agregó el mismo bloque final para demostrar el número de vueltas.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorconfloat.c)***

Se repiten los mismos errores de sintaxis del código proporcionado.

A su vez estaba diseñado para operar con **char** por lo que se cambió a **float**.

Se agregó el mismo bloque final para demostrar el número de vueltas.

***Se realiza el código fuente anexo en la carpeta que contiene a este documento (operadorcondouble.c)***

Se repiten los mismos errores de sintaxis del código proporcionado.

A su vez estaba diseñado para operar con **char** por lo que se cambió a **double**.

Se agregó el mismo bloque final para demostrar el número de vueltas.

## ¿Qué pasa cuando un tipo sale de su rango?

**Para Tipos Enteros (char, short, int, long).**

Los tipos de dato enteros "dan la vuelta", un comportamiento conocido como **aritmética modular**. Piensa en el odómetro de un coche antiguo que solo tiene 6 dígitos.

- **Desbordamiento por arriba (Overflow)**: Si el odómetro marca 999999 y avanzas un kilómetro más, no puede mostrar 1000000. En su lugar, se reinicia y da la vuelta a 000000. En la programación, si llegas al valor máximo (INT_MAX) y sumas 1, el valor da la vuelta y se convierte en el valor mínimo (INT_MIN).
- **Desbordamiento por abajo (Underflow)**: Si pudieras ir en reversa desde 000000, el odómetro daría la vuelta hacia atrás para marcar 999999. De la misma forma, si estás en el valor mínimo (INT_MIN) y restas 1, el valor da la vuelta por abajo y se convierte en el valor máximo (INT_MAX).

**Para Tipos de Punto Flotante (float, double).**

Estos tipos de dato **no dan la vuelta**. Su comportamiento es más complejo y manejan los límites convirtiéndose en valores especiales.

- **Cuando un valor es demasiado grande**: Si el resultado de una operación supera el máximo valor representable (como DBL_MAX), se convierte en **infinito** (inf). No se reinicia ni se vuelve negativo.
- **Cuando un valor es demasiado pequeño**: Si el resultado es un número positivo más cercano a cero que el mínimo representable (como DBL_MIN), pierde precisión y finalmente se redondea a **cero** (0.0).
- **Cuando una operación es inválida**: Si realizas una operación matemáticamente indeterminada, como dividir 0.0 / 0.0, el resultado se convierte en **NaN** (Not a Number o "No es un Número") para indicar que el resultado no es un valor real.

## Conclusiones.

Basado en los códigos generados, la conclusión fundamental es que los tipos de datos en C se comportan de dos maneras drásticamente diferentes al salir de su rango:

- Los **tipos enteros** (char, short, int, long) exhiben un comportamiento cíclico o de "vuelta" (aritmética modular), donde superar el límite máximo resulta en el valor mínimo y viceversa. Este comportamiento, aunque común, es técnicamente indefinido para los enteros con signo.
- Los **tipos de punto flotante** (float, double), regidos por el estándar IEEE 754, no dan la vuelta. En su lugar, se transforman en representaciones especiales: **infinito**

(inf) si el valor es demasiado grande, **cero** (0.0) si es demasiado pequeño (subdesbordamiento), o **NaN** ("No es un Número") para operaciones matemáticamente inválidas.

## Referencias.


Downey, A. B. (2016). *Think C: Like a computer scientist*. Green Tea Press. Recuperado de https://greenteapress.com/wp/think-c/

Goldschlager, L., & Lister, A. (1988). *Computer science: A modern introduction* (2nd ed.). Prentice-Hall.

Kernighan, B. W., & Ritchie, D. M. (2006). *The C programming language* (2nd ed.). Prentice Hall.

Oualline, S. (1997). *Practical C programming* (3rd ed.). O'Reilly & Associates.