C

ULTIMATE C HANDBOOK

By CodeWithHarry

PREFACIO

Bienvenido al "Manual definitivo de programación en C", su guía completa para dominar la programación en C. Este manual está diseñado para principiantes y cualquier persona que busque fortalecer sus conocimientos básicos de C, un lenguaje de programación versátil y fácil de usar.

PROPÓSITO Y AUDIENCIA

Este manual tiene como objetivo hacer que la programación sea accesible y agradable para todos. Tanto si eres un estudiante nuevo en la codificación, un profesional que busca mejorar sus habilidades o un entusiasta que explora C, este manual te será sin duda útil. La simplicidad y legibilidad de C lo convierten en un punto de partida ideal para cualquier persona interesada en la programación.

ESTRUCTURA Y CONTENIDO

El manual está dividido en capítulos claros y concisos, cada uno centrado en un aspecto específico de C:

- Conceptos fundamentales: Comience con lo básico y escriba su primer programa.
- Ejemplos prácticos: Ejemplos ilustrativos y código de muestra demuestran la aplicación de conceptos.
- Ejercicios prácticos: Los ejercicios de fin de capítulo refuerzan el aprendizaje y generan confianza.

¿POR QUÉ C?

C es conocido por su eficiencia y control, lo que lo hace perfecto para la programación a nivel de sistema. Es un lenguaje compilado de bajo nivel que proporciona un control detallado sobre el hardware y la memoria, lo que admite aplicaciones en sistemas operativos, sistemas integrados, desarrollo de juegos y computación de alto rendimiento. La potencia y la flexibilidad de C lo convierten en una herramienta valiosa tanto para programadores novatos como experimentados que buscan comprender cómo funcionan las computadoras a un nivel más profundo.

AGRADECIMIENTOS

Extiendo mi gratitud a los educadores, programadores y colaboradores que han compartido sus conocimientos y perspectivas, dando forma al contenido de este manual. Un agradecimiento especial a todos los estudiantes que ven mi contenido en YouTube y a la comunidad C por mantener un entorno de apoyo e inspiración para los estudiantes de todo el mundo.

CONCLUSIÓN

Aprender a programar puede ser tanto emocionante como desafiante. El "Manual de programación en C definitivo" tiene como objetivo hacer que su viaje sea fluido y gratificante. Mire mi video junto con seguir este manual para un aprendizaje óptimo. Deja

que esta guía sea tu trampolín para tener éxito en el mundo de la programación.

TABLA DE CONTENIDOS

MANUAL DE PROGRAMACIÓN C DE CODEWITHHARRY

¿QUÉ ES LA PROGRAMACIÓN?

La programación informática es un medio para que nos comuniquemos con las computadoras. Al igual que usamos el hindi o el inglés para comunicarnos entre nosotros, la programación es una forma de entregar nuestras instrucciones a la computadora.

¿QUÉ ES C?

C es un lenguaje de programación.

C es uno de los lenguajes de programación más antiguos y finos.

C fue desarrollado por Dennis Ritchie en los laboratorios Bell de ATCT, EE.UU. en 1972.

USOS DE C

El lenguaje C se utiliza para programar una amplia variedad de sistemas. Algunos de los usos de C son los siguientes:

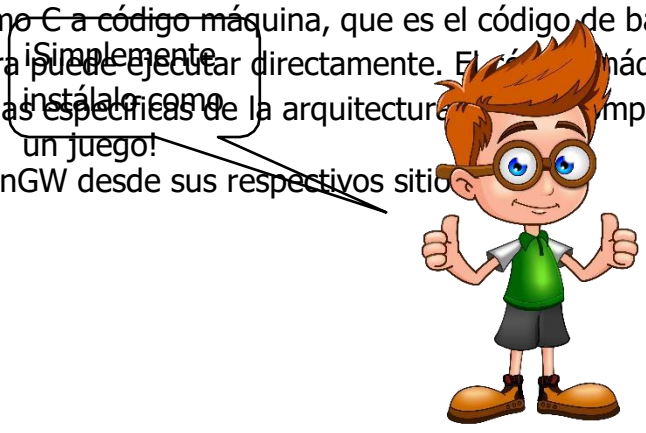
1. La mayoría de las partes de Windows, Linux y otros sistemas operativos están escritas en C.
2. C se utiliza para escribir programas de controladores para dispositivos como tabletas, impresoras, etc.
3. El lenguaje C se utiliza para programar sistemas embebidos donde los programas necesitan ejecutarse más rápido en una memoria limitada (microondas, cámaras, etc.)
4. C se utiliza para desarrollar juegos, un área en la que la latencia es muy importante, es decir, el ordenador debe reaccionar rápidamente a la entrada del usuario.

INSTALACIÓN

Usaremos VS Code como nuestro editor de código para escribir nuestro código e instalaremos el compilador MinGW gcc para compilar nuestro programa C.

La compilación es el proceso de traducir código fuente de alto nivel escrito en lenguajes de programación como C a código máquina, que es el código de bajo nivel que la CPU de una computadora puede ejecutar directamente. El código máquina consiste en instrucciones binarias específicas de la arquitectura de la computadora.

Podemos instalar VS Code y MinGW desde sus respectivos sitio



CAPÍTULO 1: VARIABLES, CONSTANTES C PALABRAS CLAVE

VARIABLES

Una variable es un contenedor que almacena un 'valor'. En la cocina, tenemos contenedores que almacenan arroz, dal, azúcar, etc. De manera similar, las variables en C almacenan el valor de una constante.

Ejemplo:

```
un = 3;      A se le asigna "3"
b = 4.7;    A b se le asigna "4.7"
c = 'A';   A c se le asigna 'A'
```

REGLAS PARA NOMBRAR VARIABLES EN C

1. El primer carácter debe ser un alfabeto o un guión bajo (_)
2. No se permiten comas, espacios en blanco.
3. No se permite ningún símbolo especial que no sea (_).
4. Los nombres de las variables distinguen entre mayúsculas y minúsculas.

Debemos crear nombres de variables significativos en nuestros programas. Esto mejora la legibilidad de nuestros programas.

CONSTANTES

Una entidad cuyo valor no cambia se llama como una

constante. Una variable es una entidad cuyo valor se puede

cambiar.

TIPOS DE CONSTANTES

Principalmente, hay tres tipos de constantes:

1. Constante entera → 1,6,7,9

2. Constante real → 322.1, 2.5 ,7.0

3. Constante de caracteres → 'a', '$', '@' (debe ir entre comillas simples)

PALABRAS CLAVE

Se trata de palabras reservadas, cuyo significado ya es conocido por el compilador. Hay 32 palabras clave disponibles en C.

Automático	doble	Int	Estructura
quebrar	largo	más	interruptor
caso	devolución	Enumeración	typedef
carbonizar	registro	Extern	unión
Const	corto	flotar	Unsigned
continuar	fichado	para	vacío
predeterminado	tamañode	Goto	volátil
hacer	estático	si	mientras

NUESTRO PRIMER PROGRAMA C

```c
#include <stdio.h>

int main() {
    printf("Hola, estoy aprendiendo C con Harry");
    devuelve 0;
}
```

ESTRUCTURA BÁSICA DE UN PROGRAMA C

Todos los programas C deben seguir una estructura básica. Un programa C comienza con una función principal y ejecuta instrucciones presentes en ella.

Cada instrucción termina con un punto y coma (;).

Hay algunas reglas que son aplicables a todos los programas C:

1. La ejecución de cada programa comienza desde la función main().
2. Todas las instrucciones terminan con un punto y coma.
3. Las instrucciones distinguen entre mayúsculas y minúsculas.
4. Las instrucciones se ejecutan en el mismo orden en que se escriben.

COMENTARIOS

Los comentarios se utilizan para aclarar algo sobre el programa en un lenguaje sencillo. Es una forma de añadir notas a nuestro programa. Hay dos tipos de comentarios en C.

1. Comentario de una sola línea: los comentarios de una sola línea comienzan con dos barras diagonales (//). Cualquier información después de las barras // que se encuentra en la misma línea se ignorará (no se ejecutará).

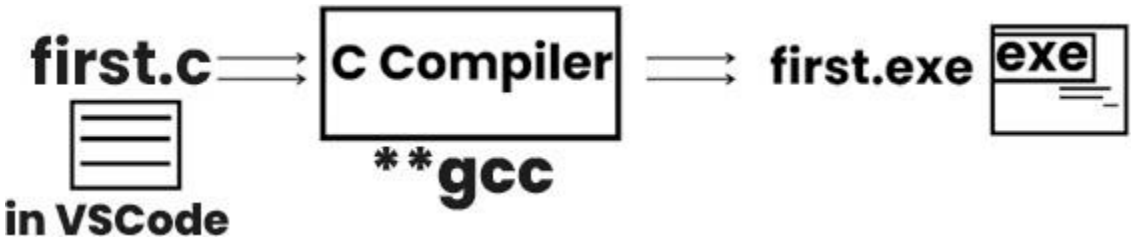
```
Este es un comentario de una sola línea.
```

2. Comentario de varias líneas: Un comentario de varias líneas comienza con /* y termina con */. Cualquier información entre /* y */ será ignorada por el compilador.

```
/*
Este es un comentario de varias líneas
*/
```

Nota: Los comentarios en un programa C no se ejecutan y se ignoran.

COMPILACIÓN Y EJECUCIÓN



Un compilador es un programa informático que convierte un programa C en lenguaje de máquina para que pueda ser fácilmente entendido por el ordenador.

Un programa C está escrito en texto plano.

Este texto sin formato es una combinación de instrucciones en una secuencia particular. El compilador realiza algunas comprobaciones básicas y finalmente convierte el programa en un ejecutable.

FUNCIONES DE LA BIBLIOTECA

El lenguaje C tiene muchas funciones de biblioteca valiosas que se utilizan para llevar a cabo ciertas tareas. Por ejemplo, la función printf() se utiliza para imprimir valores en la pantalla.

```
#include <stdio.h>
int main() {
    int i = 10;
    printf("Esto es %d\n", i);
    %d para enteros
    %f para valores reales (números de coma flotante)
     %c para los
    caracteres devuelve
    0;
```

TIPOS DE VARIABLES

1. Variables enteras → int a=3;
2. Variables reales → int a=7; float a=7.7;
3. Variables de carácter → char a= 'b';

RECEPCIÓN DE ENTRADAS DEL USUARIO

Para tomar la entrada del usuario y asignarla a una variable, usamos la función scanf()

Sintaxis:

```
scanf("%d", &i);
```

'C' es el operador "dirección de" y significa que el valor proporcionado debe copiarse en la dirección indicada por la variable i.

CAPÍTULO 1- CONJUNTO DE PRÁCTICAS

1. Escribe un programa C para calcular el área de un rectángulo:
    a. Uso de entradas codificadas de forma rígida.
    b. Utilizando entradas suministradas por el usuario.
2. Calcula el área de un círculo y modifica el mismo programa para calcular el volumen de un cilindro dado su radio y altura.
3. Escribe un programa para convertir Celsius (grados centígrados de temperatura a Fahrenheit).
4. Escriba un programa para calcular el interés simple para un conjunto de valores que representan el capital, el número de años y la tasa de interés.

CAPÍTULO 2: INSTRUCCIONES Y OPERADORES

Un programa C es un conjunto de instrucciones. Al igual que una receta, que contiene instrucciones para preparar un plato en particular.

TIPOS DE INSTRUCCIONES

1. Instrucciones para la declaración de tipos.
2. Instrucciones aritméticas
3. Instrucciones de control.

INSTRUCCIONES PARA LA DECLARACIÓN DE TIPO

Así es como se declara una variable en C

```
int a;

flotador

B;
```

OTRAS VARIACIONES:

Algunas otras variaciones de esta declaración se ven así:

```
int a;      Declarar una variable entera 'a'
flotador    Declarar una variable float 'b'
int i = 10;  Declarar e inicializar 'i' con 10
int j = i;  Declarar 'j' e inicializar con 'i'
int a = 2, b = 3, c = 4, d = 5;  Declarar e inicializar varias variables int

j1 = a + j - i;  Válido: utilizar variables previamente definidas

Inválido: se utiliza 'a' antes de la declaración
flotador b = a + 3;
flotador a = 1,1;

 Válido: Asignar el mismo valor a varias variables int a,
b, c, d;
a = b = c = d = 30;  A, B, C, D todos iguales a 30
```

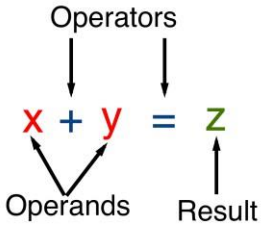
INSTRUCCIONES ARITMÉTICAS

Las instrucciones aritméticas realizan operaciones

matemáticas. Estos son algunos de los operadores más

utilizados en el lenguaje C:

- + (Adición)
- - (Resta)
- * (Multiplicación)
- / (División)
- % (módulo)



Nota:

1. Los operandos pueden ser int/float, etc. **+ - * /** son operadores aritméticos.

```
int b = 2, c = 3;
int z; z = b*c; legal
int z; b*c = z; ilegal (no permitido)
```

2. % es el operador de división modular
   - % → devuelve el resto
   - % → no se puede aplicar en flotación
   - % → signo es el mismo que el del numerador (-5%2=-1)
3. No se supone que ningún operador esté presente.

```
int i = ab  inválido
int i  = a * b válido
```

4. No hay un operador para realizar la exponenciación en C, sin embargo, podemos usar pow (x,y) de <*math.h*> (más adelante).

CONVERSIÓN DE TIPOS

Una operación aritmética entre

- int e int → int
- int y float → float
- flotar y flotar → flotar

Ejemplo:

- 5/2 se convierte en 2 ya que ambos operandos son int
- 5.0/2 se convierte en 2.5 ya que uno de los operandos es float
- 2/5 se convierte en 0 ya que ambos operandos son int

NOTA:

En programación, la compatibilidad de tipos es crucial. Para `int a = 3,5;`, el float 3.5 se degrada a 3, perdiendo la parte fraccionaria porque `a` es un número entero. Por el contrario, para `el flotador a = 8;`, el entero 8 se promueve a 8.0, coincidiendo con el tipo flotante de `a` y conservando la precisión.

```
int a = 3,5;  En este caso, 3.5 (float) se degradará a 3 (int) porque
a no puede almacenar floats.

flotador a = 8;  A Will Store 8.0 | 8 -> 8.0 (promoción a flotante)
```

Quick Quiz: int k = 3.0 /9; valor de k? ¿y por qué?

Respuesta: 3,0/9 = 0,333. Pero dado que k es un int, no puede almacenar flotantes, el valor de C 0.33 se degrada a 0.

PRECEDENCIA DEL OPERADOR EN C

Echa un vistazo a la siguiente declaración:

3*x – 8*y es (3x)-(8y) o 3(x-8y)?

En el lenguaje C, las reglas matemáticas simples como BODMAS, ya no se aplican.

La respuesta a las preguntas anteriores la proporciona la asociatividad C de precedencia del operador.

PRECEDENCIA DEL OPERADOR

En la tabla siguiente se muestra la prioridad del operador en C

Prioridad	Operadores * / %
Segundo	+ -
Tercera	=

Los operadores de mayor prioridad se evalúan primero en ausencia de paréntesis.

ASOCIATIVIDAD DEL OPERADOR

Cuando los operadores de igual prioridad están presentes en una expresión, la vinculación se soluciona mediante la asociatividad.

x*y/z →

(x*y)/z x/y*z →

(x/y)*z

*, / sigue la asociatividad de izquierda a derecha

Consejo profesional: Utilice siempre paréntesis en caso de confusión

INSTRUCCIONES DE CONTROL

Determina el flujo de control en un programa, cuatro tipos de instrucciones de control en C son:

1. Instrucciones de control de secuencia.
2. Instrucciones de control de decisiones
3. Instrucciones de control de bucle
4. Instrucciones de control de casos.

CAPÍTULO 2 – CONJUNTO DE PRÁCTICAS

1. ¿Cuál de los siguientes no es válido en C?
    a. int a=1; int b = a;
    b. int v = 3*3;
    c. char dt = '21 de diciembre de 2020';
2. ¿Qué tipo de datos devolverá 3.0/8 – 2?
3. Escribe un programa para comprobar si un número es divisible por 97 o no.
4. Explique paso a paso la evaluación de 3 * x / y - z + k, donde x = 2, y = 3, z = 3, k = 1
5. 3.0 + 1 será:
    a. Entero.
    b. Número de coma flotante.
    c. Carácter.

CAPÍTULO 3: INSTRUCCIONES CONDICIONALES

A veces queremos ver videos de comedia en YouTube si el día es domingo. A veces pedimos comida chatarra si es el cumpleaños de nuestro amigo en el albergue.

Es posible que desee comprar un paraguas si está lloviendo, y tiene el dinero. Usted pide la comida si el dal o su bhindi favorito figuran en el menú.

Todas estas son decisiones que dependen de que se cumpla una condición.

También en el lenguaje C, debemos ser capaces de ejecutar instrucciones sobre una condición que se cumple.

INSTRUCCIONES PARA LA TOMA DE DECISIONES EN C

- Instrucción if–else
- Instrucción switch

INSTRUCCIÓN IF-ELSE

La sintaxis de una instrucción if-else en C es la siguiente:

```
if (condition_to_be_checked) {
    Declaraciones si la condición es verdadera
} else {
    Declaraciones si la condición es falsa
}
```

EJEMPLO DE CÓDIGO:

```
int a = 23;
if (a > 18)
{
    printf("puedes conducir \n");
}
```

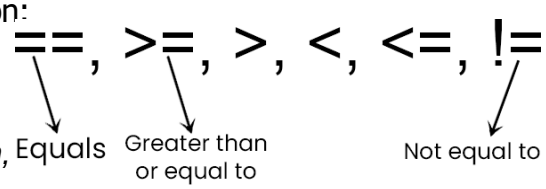
Tenga en cuenta que el bloque else no es necesario, sino opcional.

OPERADORES RELACIONALES EN C

Los operadores relacionales se utilizan para evaluar condiciones (verdaderas o falsas) dentro de las sentencias if.

Algunos ejemplos de operadores relacionales son:

==, >=, >, <, <=, !=



Nota importante: '=' se utiliza para la asignación, mientras que '==' se utiliza para la comprobación de igualdad.

La condición puede ser cualquier expresión válida. En C, un valor distinto de cero se considera verdadero.

OPERADORES LÓGICOS

CC, || y !, son tres operadores lógicos en C. Estos se leen como "Y", "O" y "NO"

Se utilizan para proporcionar lógica a nuestros programas C.

USO DE OPERADORES LÓGICOS:

1. CC (AND) → es verdadera cuando ambas condiciones son verdaderas
    a. "1 y 0" se evalúa como falso.
    b. "0 y 0" se evalúan como falsos.
    c. "1 y 1" se evalúa como verdadero.
2. || (O) → es verdadera cuando al menos una de las condiciones es verdadera. (1 o 0 → 1) (1 o 1 → 1)
3. ! (NO) → devuelve verdadero si se da falso y falso si se da verdadero
    a. ! (3==3) → se evalúa como falso
    b. ! (3>30) → se evalúa como verdadero.

A medida que aumenta el número de condiciones, aumenta el nivel de sangría. Esto reduce la legibilidad. Los operadores lógicos vienen al rescate en estos casos.

CLÁUSULA ELSE IF

En lugar de usar múltiples declaraciones if, también podemos usar else if junto con ellas, formando así una escalera if-else if-else.

EJEMPLO DE CÓDIGO

Un aspecto típico de la escalera if - else if - else es el siguiente:

```
if{
    Declaraciones
}
else if{
    Declaraciones
}
else{
    Declaraciones
}
```

NOTA IMPORTANTE

1. El uso de if-else if -else reduce las sangrías.
2. El último "si no" es opcional.
3. También puede haber cualquier número de "si no".
4. El último se ejecuta solo si se produce un error en todas las condiciones.

PRECEDENCIA DEL OPERADOR

Prioridad	Operador
primero	!
Segundo	*, /, %
Tercera	+, -
4º	<>, <=, >=
5º	==, !=
sexto	CC
séptimo	||
8º	=

OPERADORES CONDICIONALES

Una abreviatura "if – else" se puede escribir usando los operadores condicionales o ternarios

```
condición ?  expresión-si-true  : expresión-si-falso
Aquí "?" y ":" se denominan operadores ternarios
```

INSTRUCCIONES DE CONTROL DE LA CAJA DEL INTERRUPTOR

switch-case se utiliza cuando tenemos que elegir entre un número de alternativas para una variable dada.

```
switch (expresión entera)
{

Caso C1:
    código;

  caso C2:                    c1, c2 y c3 -> constantes
      código;                code -> Cualquier código C
                                          válido.

Caso C3:
    código:

predeterminado:
    código;
```

El valor de integer-expression se compara con c1, c2, c3... Si coincide con alguno de estos casos, se ejecuta ese caso junto con todas las instrucciones "case" y "default" subsiguientes.

Quick Quiz: Escriba un programa para encontrar la calificación de un estudiante dada su calificación basada en lo siguiente: 90 – 100 = > A
80 – 90 => B
70 – 80 => C
60 – 70 => D
50 – 60 => E
<50 => F

Algunas notas importantes:

- Podemos usar sentencias de mayúsculas y minúsculas incluso escribiendo las mayúsculas y minúsculas en cualquier orden de nuestra elección (no necesariamente ascendentes).
- Los valores char están permitidos, ya que se pueden evaluar fácilmente como un número entero.
- Un cambio puede ocurrir dentro de otro, pero en la práctica esto rara vez se hace.

CAPÍTULO 3 – CONJUNTO DE PRÁCTICAS

1. ¿Cuál será la salida de este programa?

```
int a = 10;
if (a = 11)
   printf("Tengo 11
años"); más
    printf("No tengo 11 años");
```

2. Escriba un programa para determinar si un estudiante ha aprobado o reprobado. Para aprobar, un estudiante requiere un total de 40% y al menos 33% en cada materia. Supongamos que hay tres sujetos y tome las notas como entrada del usuario.
3. Calcule el impuesto sobre la renta pagado por un empleado al gobierno según las tablas que se mencionan a continuación:

Losa de ingresos	Impuesto
2.5 – 5.0L	5%
5.0L - 10.0L	20%
Por encima de 10.0L	30%

Tenga en cuenta que no hay impuestos por debajo de 2.5L. Tome el monto de los ingresos como una entrada del usuario.

4. Escriba un programa para averiguar si un año ingresado por el usuario es un año bisiesto o no. Tome el año como una entrada del usuario.
5. Escriba un programa para determinar si un carácter ingresado por el usuario está en minúsculas o no.
6. Escriba un programa para encontrar el mayor de los cuatro números ingresados por el usuario.

CAPÍTULO 4: INSTRUCCIONES DE CONTROL DE BUCLE

¿POR QUÉ LOS BUCLES?

A veces queremos que nuestros programas ejecuten algunos conjuntos de instrucciones una y otra vez. Por ejemplo: Imprimir del 1 al 100, los primeros 100 números pares, etc.

Por lo tanto, los bucles hacen que sea fácil para un programador decirle a la computadora que un conjunto dado de instrucciones debe ejecutarse repetidamente.

TIPOS DE BUCLES

Principalmente hay tres tipos de bucles en el lenguaje C:

1. bucle while
2. Bucle do–while
3. bucle for

Vamos a analizarlos uno por uno:

BUCLE WHILE

```
while (la condición es verdadera) {
    Código
    El bloque se sigue ejecutando mientras la condición sea verdadera
}
```

Ejemplo:

```
int i = 0;
while (i<10) {
   printf("el valor de i es %d\n", i);
   I++;
}
```

Nota: *Si la condición nunca se convierte en falsa, el bucle while se sigue ejecutando. Este bucle se conoce como bucle infinito.*

Quick Quiz: Escribe un programa para imprimir números naturales del 10 al 20 cuando el contador de bucle inicial se inicializa en 0.

El contador de bucle no necesita ser int, también puede ser float.

OPERADORES DE INCREMENTO Y DECREMENTO

i++ → i se incrementa

en 1 i -- → i se

disminuye en 1

```
 Disminuya i primero y luego
imprima printf("--i = %d\n", --i);

 Imprime i primero y luego
disminuye printf("i-- = %d\n", i--
```

- +++ no existe.
- i + = 2 es una asignación compuesta que se traduce en i = i + 2
- Similar al operador += tenemos otros operadores como -=, *=, /=, %=.

BUCLE DO–WHILE

La sintaxis del bucle do-while es la siguiente:

```
hacer {
   código;
} while (condición);
```

El bucle do–while funciona de forma muy similar al bucle while.

- 'while' comprueba la condición C y luego ejecuta el código.
- 'do-while' ejecuta el código C y luego verifica la

condición. En términos más simples podemos decir:

do-while loop = bucle while que se ejecuta al menos una vez.

Quick Quiz: Escribe un programa para imprimir el primer número natural 'n' usando el bucle do-while.

```
Entrada: 4
```

```
Salida: 1
        2
        3
        4
```

BUCLE FOR

La sintaxis de un bucle 'for' típico es la siguiente:

```
for (inicializar; probar; incrementar o disminuir)
{
```

```
    código;
}
```

- Inicializar → establecer un contador de bucle en un valor inicial.
- Prueba → Comprobación de una condición.
- Incremento → Actualización del contador de bucles.

Ejemplo:

```
para (i=0; I<3; i++){
    printf("%d\n", i);
    printf("\n");
}
Salida:
//          0
//          1
//          2
```

Quick Quiz: Escribe un programa para imprimir primero 'n' números naturales usando el bucle for

UN CASO DE DECREMENTO PARA BUCLE

```
para (i=5; yo ; i--)
      printf("%d\n",i);
```

Este bucle for seguirá ejecutándose hasta que i

se convierta en 0. El bucle se ejecuta en los

siguientes pasos:

1. 'i' se inicializa en 5.
2. Se prueba la condición "i" (0 o ninguna).
3. Se ejecuta el código.
4. La 'i' está decrementada.
5. La condición 'i' está marcada, el código C se ejecuta si no es 0.
6. Y así sucesivamente hasta que 'i' no sea 0.

Quick Quiz: Escribe un programa para imprimir 'n' números naturales en orden inverso.

LA INSTRUCCIÓN BREAK EN C

La instrucción 'break' se utiliza para salir del bucle, independientemente de si la condición es verdadera o falsa.

Cada vez que se encuentra una "ruptura" dentro del bucle, el control se envía fuera

del bucle Veamos esto con la ayuda de un ejemplo:

```
para (i=0; I<1000; i++){
   printf("%d\n",i);
   si (i==5){
      quebrar;
   }
}
```

SALIDA

```
0
1
2
3
4
5
```

La salida del programa anterior estará por debajo (y no de 0 a 100)

LA INSTRUCCIÓN CONTINUE EN C

La instrucción 'continue' se utiliza para pasar inmediatamente a la siguiente iteración del bucle. El control se lleva a la siguiente iteración, por lo tanto, se omite todo lo que se encuentra debajo de "continuar"
dentro del bucle para esa iteración.

Ejemplo:

```
#include <stdio.h>

int main() {
    int skip = 5;
    int y = 0;
    while (I < 10) {
        if (i == saltar) {
            i++;
            continuar;  Omite el resto del cuerpo del bucle para i == 5
        }
        printf("%d\n", i);
        I++;
    }
    devuelve 0;
}
```

Notas:

1. A veces, el nombre de la variable puede no indicar el comportamiento del programa.
2. 'break' sale completamente del bucle.

1. 'continue' omite la iteración particular del bucle.

CAPÍTULO 4 – CONJUNTO DE PRÁCTICAS

1. Escribe un programa para imprimir la tabla de multiplicar de un número dado n.
2. Escribe un programa para imprimir la tabla de multiplicar de 10 en orden inverso.
3. Se ejecuta un bucle do while:
    a. Al menos una vez.
    b. Al menos dos veces.
    c. A lo sumo una vez.
4. Lo que se puede hacer usando un tipo de bucle también se puede hacer usando los otros dos tipos de bucles: ¿verdadero o falso?
5. Escribe un programa para sumar los primeros diez números naturales usando el bucle while.
6. Escriba un programa para implementar el programa 5 usando el bucle 'for' y 'do-while'.
7. Escribe un programa para calcular la suma de los números que aparecen en la tabla de multiplicar de 8. (considere de 8 x 1 a 8 x 10).
8. Escribe un programa para calcular el factorial de un número dado usando un bucle for.
9. Repita 8 usando el bucle while.
10. Escribe un programa para comprobar si un número dado es primo o no usando bucles.
11. Implemente 10 usando otros tipos de bucles.

PROYECTO 1: JUEGO DE ADIVINANZAS DE NÚMEROS

Escribiremos un programa que genere un número aleatorio y le pida al jugador que lo adivine. Si la suposición del jugador es mayor que el número real, el programa muestra "Número más bajo, por favor". Del mismo modo, si la estimación del usuario es demasiado baja, el programa imprime "Número más alto, por favor".

Cuando el usuario adivina el número correcto, el programa muestra el número de intentos que el jugador utilizó para llegar al número.

Sugerencia: Use un bucle y use un generador de números aleatorios.

CAPÍTULO 5 – FUNCIONES Y RECURSIVIDAD

A veces, nuestro programa se hace más grande y no es posible que un programador rastree qué parte del código está haciendo qué.

La función es una forma de dividir nuestro código en trozos para que sea posible que un programador los reutilice.

¿QUÉ ES UNA FUNCIÓN?

Una función es un bloque de código que realiza una tarea en particular.

Una función puede ser reutilizada por el programador en un programa dado cualquier número de veces.

Sintaxis:

```
#include <stdio.h>


Prototipo de función
void display();

int main() {
    int a;  Declaración de
    variable display();  Retorno
    de llamada de función  0;
    Declaración de retorno

Definición de la función
void display() {
    printf("Hola, soy pantalla\n");  Impresión del
    mensaje
```


PROTOTIPO DE FUNCIÓN

Un prototipo de función informa al compilador sobre una función que se definirá más adelante en el programa.

La palabra clave void indica que la función no devuelve ningún valor.

LLAMADA A LA FUNCIÓN

Una llamada a una función indica al compilador que ejecute el cuerpo de la función cuando se realiza la llamada.

Tenga en cuenta que la ejecución del programa comienza desde la función principal y sigue la secuencia de instrucciones escritas.

DEFINICIÓN DE LA FUNCIÓN

Esta parte contiene el conjunto exacto de instrucciones ejecutadas durante la llamada a la función.

Cuando se llama a una función desde main(), la función principal se detiene y se suspende temporalmente. Durante este tiempo, el control se transfiere a la función llamada. Una vez que la función termina de ejecutarse, main() se reanuda.

Quick Quiz: Escribe un programa con tres funciones

1. Función de buenos días que imprime "buenos días".
2. Función de buenas tardes en la que se imprime "buenas tardes".
3. Función de buenas noches que imprime "buenas noches".

main() debería llamar a todos estos en orden 1→2→3

PUNTOS IMPORTANTES

- La ejecución de un programa C comienza desde main().
- Un programa C puede tener más de una función.
- Cada función se llama directa o indirectamente desde main().

TIPOS DE FUNCIONES

Hay dos funciones en C. Hablemos de ellos.

1. Funciones de biblioteca → Funciones comúnmente requeridas agrupadas en un archivo de biblioteca en disco.
2. Función definida por el usuario → Estas son las funciones declaradas y definidas por el usuario.

¿POR QUÉ USAR FUNCIONES?

1. Para evitar reescribir la misma lógica una y otra vez.
2. Para realizar un seguimiento de lo que estamos haciendo en un programa
3. Para probar y comprobar la lógica de forma independiente.

PASAR VALORES A LA FUNCIÓN

Podemos pasar valores a una función y podemos obtener un valor a cambio

de una función. Echa un vistazo al fragmento de código a continuación:

```
int suma (int a, int b)
```

Un prototipo de función en programación es una declaración de una función que especifica su nombre, tipo de valor devuelto y parámetros (si los hay), pero no incluye el cuerpo de la función.

El prototipo anterior significa que sum es una función que toma los valores 'a' (de tipo int) y 'b' (de tipo int) y devuelve un valor de tipo int.

La definición de la función de suma puede ser:

```
int sum (int a, int b) {  a y b son parámetros int
   c;
   c = a+b;
   retorno c;
}
Ahora podemos llamar suma (2,3); de main para obtener 5 a cambio. Aquí 2 y
3 son argumentos.
int  d = suma (2,3);  d se convierte en 5
```

NOTA:

1. Los parámetros son los valores o marcadores de posición de variables en la definición de la función. Ejemplo a C b.
2. Los argumentos son los valores reales que se pasan a la función para realizar una llamada. Ejemplo 2 C 3.
3. Una función solo puede devolver un valor a la vez.
4. Si la variable pasada se cambia dentro de la función, la llamada a la función no Cambie el valor de la función de llamada.

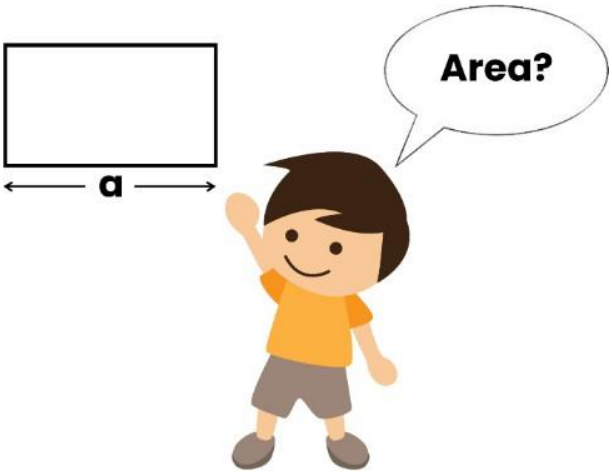
```
int cambio(int a) {
   un = 77;          Nombre
   equivocado devolución 0;
}
```

'cambio' es una función que pretende cambiar 'a' a 77. Ahora, si lo llamamos desde main de esta manera:

```
int b=22;
cambio(b);                El valor de b sigue siendo
printf("b es %d", b);     22
```

Esto sucede porque se pasa una copia de 'b' a la función de cambio

Quick Quiz: Usa la función de biblioteca para calcular el área de un cuadrado con el lado a.



RECURSIÓN

Una función definida en C puede llamarse a sí misma. A esto se le llama recursividad. Una función que se llama a sí misma es
también llamada función 'recursiva'.

Ejemplo:

Un muy buen ejemplo de recursividad es

factorial. Factorial(n) = 1 x 2 x 3 ... x n

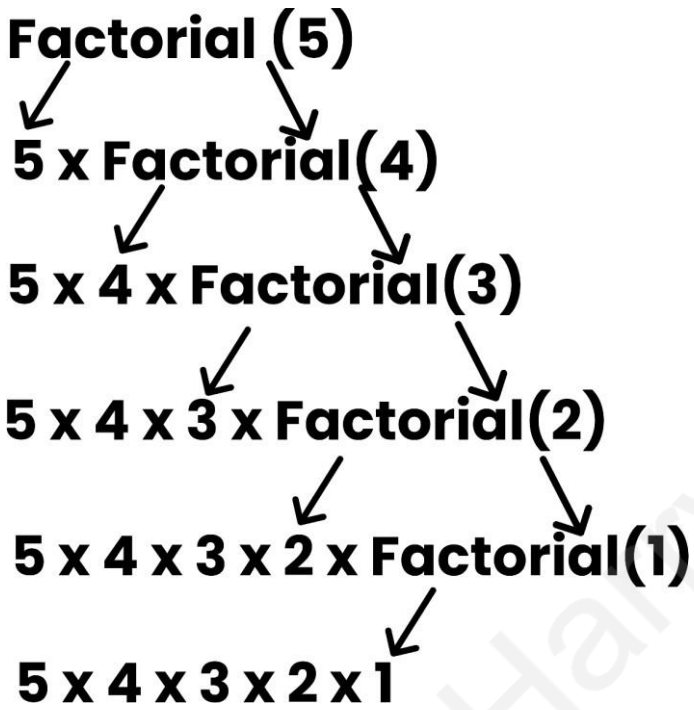
Factorial(n) = 1 x 2 x3 ... (n-1) x n

Factorial(n) = Factorial (n-1) x n

Dado que podemos escribir factorial de un número en términos de sí mismo, podemos programarlo usando la recursividad.

```
int factorial(int x) {
    int f;
    si (x == 0 || x == 1) {
        retorno 1;  Un programa para calcular factorial usando
        recursividad
    } else {
        f = x * factorial(x - 1);
        retorno f;
    }
```

SIMULACRO DE PROGRAMA FACTORIAL RECURSIVO



NOTAS IMPORTANTES:

1. **La recursividad suele ser una forma directa de implementar ciertos algoritmos, pero no siempre es la más directa para todos los algoritmos.** La recursividad es particularmente adecuada para problemas que se pueden dividir en subproblemas más pequeños y similares (como el cálculo factorial o el recorrido de árbol), pero para algunos algoritmos, los enfoques iterativos pueden ser más sencillos o eficientes.
2. **La condición de una función recursiva que detiene la recursividad adicional se denomina caso base.** Esta corrección aclara que el caso base es crucial, ya que evita la recursividad infinita y garantiza que la función finalice correctamente.
3. **A veces, debido a un descuido por parte del programador, una función recursiva puede continuar ejecutándose indefinidamente sin llegar a un caso base, lo que puede causar un desbordamiento de pila o un error de memoria.** Esta afirmación pone de relieve el riesgo de la recursividad infinita y sus consecuencias, enfatizando la importancia de definir correctamente los casos base en las funciones recursivas.

CAPÍTULO 5 – CONJUNTO DE PRÁCTICAS

1. Escribe un programa usando una función para encontrar el promedio de tres números.
2. Escribe una función para convertir la temperatura Celsius en Fahrenheit.
3. Escribe una función para calcular la fuerza de atracción sobre un cuerpo de masa 'm' ejercida por la tierra. Considere g = 9,8 m/s2.
4. Escriba un programa usando recursividad para calcular el enésimo elemento de la serie de Fibonacci.
5. ¿Qué producirá la siguiente línea en un programa C?

```c
int a = 4;
printf("%d %d %d \n", a, ++a, a++);
```

6. Escribe una función recursiva para calcular la suma de los primeros n números naturales.
7. Escriba un programa usando la función para imprimir el siguiente patrón (primeras n líneas)

```
*

***

*****
```

CAPÍTULO 6- PUNTEROS

Un puntero es una variable que almacena la dirección de otra variable.

EL OPERADOR DE LA "DIRECCIÓN DE" (C)

La dirección del operador se utiliza para obtener la dirección de una

variable dada. Si te refieres a los diagramas anteriores,

Ci → 87994

Cj → 87998

El especificador de formato para imprimir la dirección del puntero es

EL OPERADOR 'VALOR EN LA DIRECCIÓN' (*)

El valor en la dirección o el operador * se utiliza para obtener el valor presente en una dirección de memoria determinada. Se denota con *.

*(Ci) =72

*(Cj) = 87994

¿CÓMO DECLARAR UN PUNTERO?

Un puntero se declara mediante la siguiente sintaxis.

- int *j => declarar una variable j de tipo int-pointer
- j=Ci => la dirección de almacenamiento de i en j.

Al igual que el puntero de tipo entero, también tenemos punteros para char, float, etc.

```
int *in_ptr; puntero al entero char
*ch_ptr; puntero al carácter float
*fl_ptr; puntero a float
```

Aunque es una buena práctica usar nombres de variables significativos, debemos ser muy

Tenga cuidado al leer y trabajar en programas de otros programadores.

UN PROGRAMA PARA DEMOSTRAR PUNTEROS

```c
#include <stdio.h>

int main (){
    int i = 8;
    int *j;
    j = &i;
    printf("añadir i= %u\n",&i);
    printf("añadir i= %u\n",j);
    printf("agregar j= %u\n",&j);
    printf("valor i= %d\n",i);
    printf("valor i= %d\n",*(&i));
    printf("valor i= %d\n",*j);
    devuelve 0;
}
```

SALIDA:

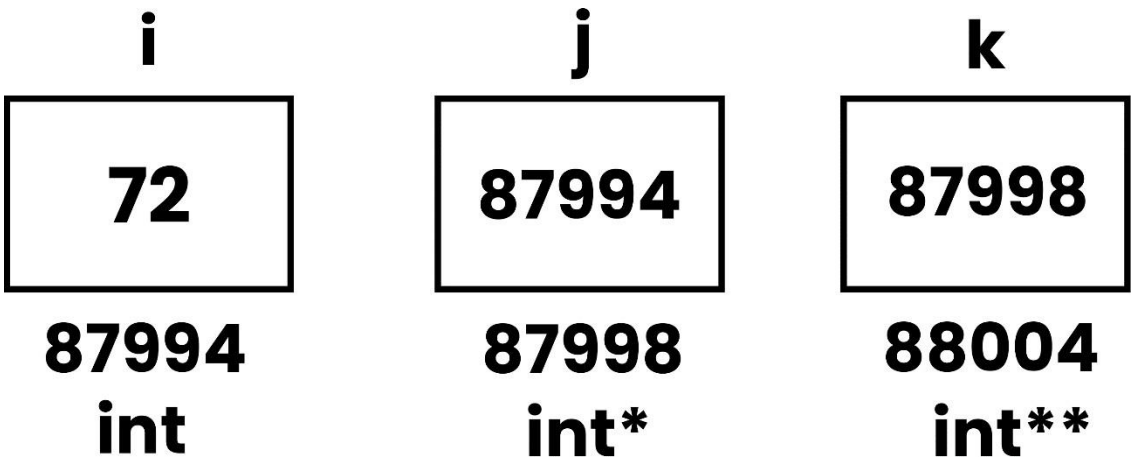
```
Añadir i= 87994
Añadir i= 87994
Añadir J= 87998
Valor i= 8
Valor i= 8
Valor i= 8
```

Este programa lo resume todo. Si lo entiendes, tienes la idea de los indicadores.

PUNTERO A UN PUNTERO

Al igual que 'j' apunta a 'i' o almacena la dirección de 'i', podemos tener otra variable k que puede almacenar aún más la dirección de 'j'. ¿Cuál será el tipo de 'k'?

```
int **k;
k = &j;
```



Incluso podemos ir más allá y crear una variable 'l' de tipo int*** para almacenar la dirección de 'k'. Usamos principalmente int* e int** a veces en programas del mundo real.

TIPOS DE LLAMADA A FUNCIÓN

Según la forma en que pasamos los argumentos a la función, las llamadas a funciones son de dos tipos.

1. Llamada por valor → Envío de los valores de los argumentos.
2. Llamada por referencia → Envío de la dirección de los argumentos.

LLAMADA POR VALOR

Aquí, los valores de los argumentos se pasan a la función. Considere este ejemplo:

```
int c = suma (3,4); Supongamos que x=3 e y=4
```

Si sum se define como sum (int a, int b), los valores 3 y 4 se copian en a y b. Ahora, incluso si cambiamos a y b, no les pasa nada a las variables x e y.

Se trata de una llamada por valor.

En C solemos hacer una llamada por valor.

LLAMADA POR REFERENCIA

Aquí, la dirección de las variables se pasa a la función como argumentos.

Ahora, dado que las direcciones se pasan a la función, la función ahora puede modificar el valor de una variable en la función de llamada usando los operadores * y C.

EJEMPLO

```
Intercambio de vacíos  (int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temperatura;
}
```

Esta función es capaz de intercambiar los valores que se le pasan. Si a = 3 y b = 4 antes de una llamada a swap(a, b), entonces a = 4 y b = 4 después de llamar a swap.

```
int main(){
    int a = 3;
    int b = 4;  a es 3 y b es 4 swap(&a,
    &b);
    devolución 0; Ahora A es 4 y B es 3
}
```

CAPÍTULO 6 – CONJUNTO DE PRÁCTICAS

1. Escriba un programa para imprimir la dirección de una variable. Utilice esta dirección para obtener el valor de la variable.
2. Escriba un programa que tenga una variable 'i'. Escriba la dirección de 'i'. Pase esta variable a una función e imprima su dirección. ¿Son estas direcciones iguales? ¿Por qué?
3. Escriba un programa para cambiar el valor de una variable a diez veces su valor actual.
4. Escriba una función y pase el valor por referencia.
5. Escribe un programa usando una función que calcula la suma y el promedio de dos números. Utilice punteros e imprima los valores de sum y average en main().
6. Escriba un programa para imprimir el valor de una variable i usando el tipo "puntero a puntero"
   de la variable.
7. Pruebe el problema 3 usando la llamada por valor y verifique que no cambie el valor de dicha variable.

CAPÍTULO 7 – ARRAYS

Una matriz es una colección de elementos similares. Array permite que una sola variable almacene varios valores.

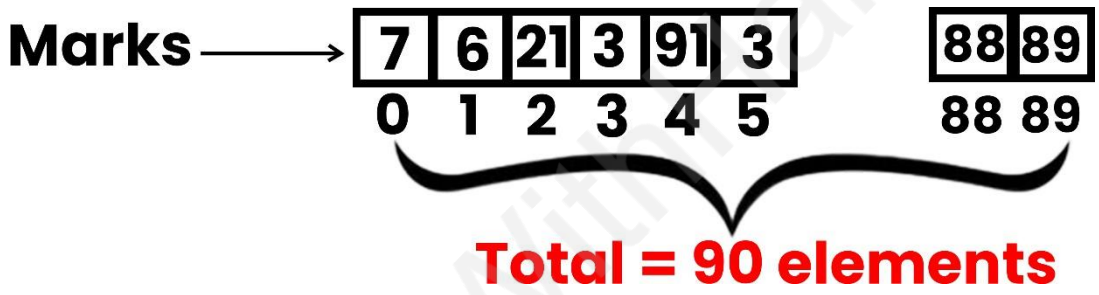
SINTAXIS:

```
marcas                Matriz de enteros
int[90];              matriz de caracteres o
percentil de flotación[90];  Matriz
```

Los valores ahora se pueden asignar para hacer una matriz como esta:

```
Puntos[0] = 33;
notas[1] = 12;
```

Nota: Es muy importante tener en cuenta que el índice de la matriz comienza con 0.



ACCESO A LOS ELEMENTOS

Se puede acceder a los elementos de una matriz mediante:

```
scanf("%d", &marks[0]);  Primer valor de entrada
printf("%d", marcas[0]);  Primer valor de salida de la matriz
```

Quick Quiz: Escribe un programa para aceptar notas de cinco estudiantes en una matriz e imprimirlas en la pantalla.

INICIALIZACIÓN DE UNA MATRIZ

Hay muchas otras formas en las que se puede inicializar una matriz.

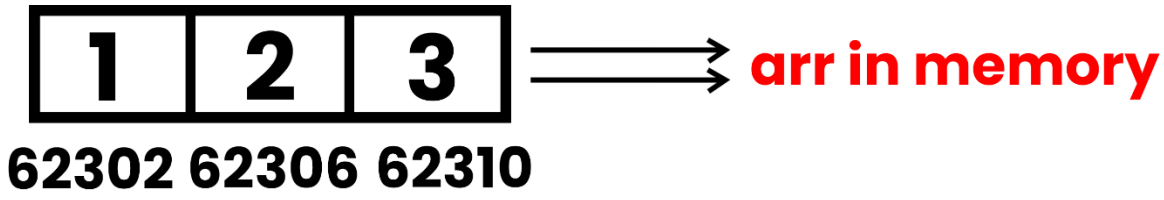
```
int cgpa[3] = {9, 8, 8};  las matrices se pueden inicializar mientras la
declaración float marcas[] = {33, 40};
```

MATRICES EN MEMORIA

Considere esta matriz:

```
Int Arr[3] = {1, 2, 3} 1 entero = 4 bytes
```

Esto reservará 4 x 3 = 12 bytes en memoria (4 bytes para cada entero).



62302 62306 62310

ARITMÉTICA DE PUNTEROS

Un puntero se puede incrementar para que apunte a la siguiente ubicación

de memoria de ese tipo. Considere este ejemplo:

```
int i = 32;
int *a = &i;  a=87994
A++;          dirección de i o valor de a = 87998

char a = 'A';
carbonizar*b = &un; a= 87994
B++;          ahora a = 87995

flotador i = 1,7;
flotar*un = &Yo; ahora a =
87994 A++;      ahora a = 87998
```

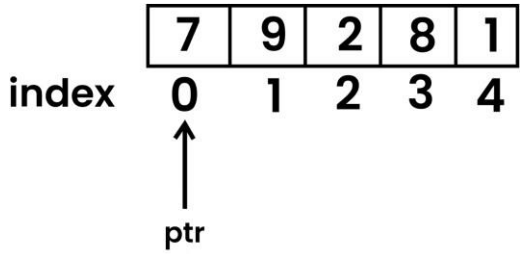
Se pueden realizar las siguientes operaciones en un puntero:

1. Adición de un número a un puntero.
2. Resta de un número de un puntero.
3. Resta de un puntero de otro.
4. Comparación de dos variables punteras.

Quick Quiz: Pruebe estas operaciones en otra variable creando punteros en un programa separado. Demuestre las cuatro operaciones.

ACCESO A LA MATRIZ MEDIANTE PUNTEROS

Considere esta matriz:



Si ptr apunta al índice 0, ptr++ apuntará al índice 1 C y así sucesivamente...

De esta manera podemos tener un puntero entero apuntando al primer elemento de la matriz de la siguiente manera:

```
int *ptr = &arr[0];  o arr simple ptr++;
*El RPP  tendrá 9 como valor
```

PASAR LA MATRIZ A LAS FUNCIONES

La matriz se puede pasar a las funciones de la siguiente manera:

```
printArray(arr, n);              llamada a la
void printArray(int *i, int n);  Prototipo de función
//    o
void printArray(int i[], int n);
```

RAYOS MULTIDIMENSIONALES

Una matriz puede ser de 2 dimensiones/3 dimensiones/n

dimensiones. Una matriz de 2 dimensiones se puede

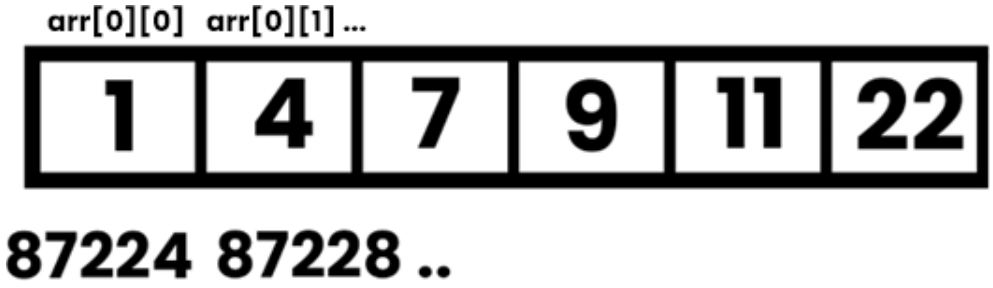
```
int arr[3][2] = {{1, 4}
                 {7, 9}
                 {11, 22}};
```

definir de la siguiente manera:

Podemos acceder a los elementos de esta matriz como arr[0][0] , arr[0][1] C y así sucesivamente ...

MATRICES 2D EN MEMORIA

Una matriz 2d como una matriz 1d se almacena en bloques de memoria contiguos como este:



Quick Quiz: Cree una matriz 2-D tomando la entrada del usuario. Escriba una función de visualización para imprimir el contenido de esta matriz 2D en la pantalla.

CAPÍTULO 7 – CONJUNTO DE PRÁCTICAS

1. Crea una matriz de 10 números. Compruebe mediante la aritmética de punteros que (ptr+2) apunta al tercer elemento donde ptr es un puntero que apunta al primer elemento de la matriz.
2. Si S[3] es una matriz 1-D de números enteros, entonces *(S+3) se refiere al tercer elemento:
    (i) Verdadero.
    (ii) Falso.
    (iii) Depende.
3. Escribe un programa para crear una matriz de 10 números enteros y almacenar una tabla de multiplicar de 5 en ella.
4. Repita el problema 3 para una entrada general proporcionada por el usuario mediante scanf.
5. Escriba un programa que contenga una función que invierta la matriz que se le pasa.
6. Escriba un programa que contenga funciones que cuente el número de enteros positivos en una matriz.
7. Cree una matriz de tamaño 3 x 10 que contenga tablas de multiplicar de los números 2, 7 y 9 respectivamente.
8. Repita el problema 7 para una entrada personalizada dada por el usuario.
9. Cree una matriz tridimensional e imprima la dirección de sus elementos en orden creciente.

CAPÍTULO 8 – CUERDAS

Una cadena es una matriz de caracteres 1-D terminada por un carácter nulo ('\0')

Un carácter nulo se utiliza para denotar la terminación de una cadena. Los caracteres se almacenan en ubicaciones de memoria contiguas.

INICIALIZACIÓN DE CADENAS

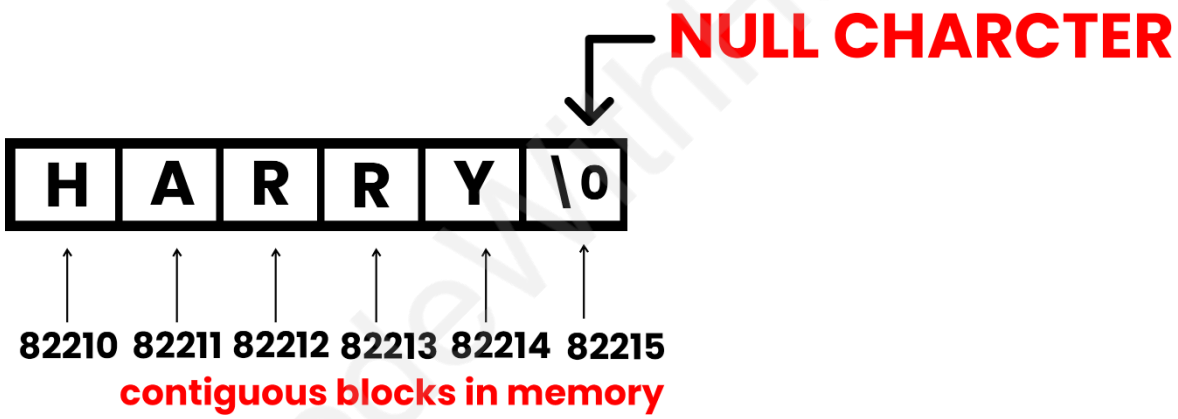
Dado que string es una matriz de caracteres, se puede inicializar de la siguiente manera:

```
char s[] = {'H', 'A', 'R', 'R', 'Y', '\0'};
```

Hay otro atajo para inicializar una cadena en el lenguaje C:

CADENAS EN MEMORIA

Una cadena se almacena como una matriz en la memoria, como



Quick Quiz: Crea una cadena usando comillas dobles e imprime su contenido

CADENAS DE IMPRESIÓN

Una cadena se puede imprimir carácter por carácter utilizando printf

y %c. Pero hay otra forma conveniente de imprimir cadenas en C.

```
char st[] = "HARRY";
printf("%s", st);  Imprime toda la cadena.
```

TOMAR LA ENTRADA DE CADENA DEL USUARIO

Podemos usar %s con scanf para tomar la entrada de cadena del usuario:

```
Char st[50];
scanf ("%s", st);
```

scanf añade automáticamente un carácter nulo cuando se pulsa la tecla Intro.

Nota:

1. La cadena debe ser lo suficientemente corta como para caber en la matriz.
2. scanf no se puede utilizar para introducir cadenas de varias palabras con espacios.

GETS() Y PUTS()

gets() es una función que se puede usar para recibir una cadena de varias palabras.

```
Char St[30];
gets(st);  La cadena introducida se almacena en st!
```

Se necesitarán varias llamadas a gets() para varias cadenas.

Del mismo modo, puts se puede usar para generar una

cadena.

```
puts(st);  Imprime la cadena y coloca el cursor en la siguiente línea
```

DECLARAR UNA CADENA MEDIANTE PUNTEROS

Podemos declarar cadenas usando punteros.

```
cuatro *ptr = "Harry";
```

Esto indica al compilador que almacene la cadena en la memoria y la dirección asignada se almacena en un puntero char.

Nota:

1. Una vez que se define una cadena usando *char st [] = "harry"*, no se puede reinicializar a otra cosa.
2. Se puede reinicializar una cadena definida mediante punteros.

```
ptr = "Rohan";
```

FUNCIONES DE BIBLIOTECA ESTÁNDAR PARA CADENAS

C proporciona un conjunto de funciones de biblioteca estándar para la

manipulación de cadenas. Algunas de las funciones de cadena más

utilizadas son:

STRLEN()

Esta función se utiliza para contar el número de caracteres de la cadena, excluyendo el valor nulo

('\0') caracteres.

```
Int largura = strlen(ST);
```

Estas funciones se declaran en el archivo de encabezado <string.h>.

STRCPY()

Esta función se utiliza para copiar el contenido de la segunda cadena en la primera cadena que se le pasa.

```
fuente de char[] =
"Harry"; objetivo de
char[30];
```

La cadena de destino debe tener suficiente capacidad para almacenar la cadena de origen.

STRCAT()

Esta función se utiliza para concatenar dos cadenas.

```
char s1[12] = "hola";
char s2[] = "harry";
strcat(s1,s2);  S1 ahora contiene "helloHarry" <sin espacio entre ellos>
```

STRCMP()

Esta función se utiliza para comparar dos cadenas. Devuelve 0 si las cadenas son iguales, un valor negativo si el valor ASCII del carácter de discrepancia de la primera cadena es menor que el carácter de discrepancia correspondiente de la segunda cadena y, en caso contrario, un valor positivo.

```
strcmp("lejos",           Valor negativo
"broma");                 Valor positivo
```

CAPÍTULO 8 – CONJUNTO DE PRÁCTICAS

1. ¿Cuál de las siguientes opciones se usa para leer correctamente una cadena de varias palabras?
    1. gets()
    2. puts()
    3. printf()
    4. scanf()
2. Escriba un programa para tomar una cadena como entrada del usuario mediante %c y %s para confirmar que las cadenas son iguales.
3. Escriba su propia versión de la función strlen desde <string.h>
4. Escribe una función slice() para segmentar una cadena. Debe cambiar la cadena original de modo que ahora sea la cadena cortada. Tome 'm' y 'n' como posición inicial y final para la división.
5. Escriba su propia versión de la función strcpy desde <string.h>
6. Escriba un programa para cifrar una cadena añadiendo 1 al valor ascii de sus caracteres.
7. Escriba un programa para descifrar la cadena cifrada usando la función encrypt en el problema 6.
8. Escriba un programa para contar la ocurrencia de un carácter dado en una cadena.
9. Escriba un programa para comprobar si un carácter dado está presente en una cadena o no.

CAPÍTULO 9 – ESTRUCTURAS

Matriz y cadenas → datos similares (int, float,

char). Las estructuras pueden contener → datos

diferentes.

Una estructura C se puede crear de la siguiente manera:

```
Empleado de estructura
{
    código int;  ¡Esto declara un nuevo tipo de datos definido
    por el usuario! salario flotante;
    Nombró cuatro[10];
};  El punto y coma es importante
```

Podemos utilizar este tipo de datos definido por el usuario de la siguiente manera:

```
Empleado de estructura  E1;  creando una variable de
estructura strcpy(e1.name, "harry");
e1.código = 100;
e1.salario = 71,22;
```

Por lo tanto, una estructura en C es una colección de variables de diferentes tipos bajo un solo nombre.

Quick Quiz: Escribe un programa para almacenar los detalles de 3 empleados a partir de datos definidos por el usuario. Utilice la estructura declarada anteriormente.

¿POR QUÉ USAR ESTRUCTURAS?

Podemos crear los tipos de datos en la estructura de empleados por separado, pero cuando aumenta el número de propiedades de una estructura, se nos dificulta crear variables de datos sin estructuras. En pocas palabras:

a. Las estructuras mantienen los datos organizados.
b. Las estructuras facilitan la gestión de datos para el programador.

CONJUNTO DE ESTRUCTURAS

Al igual que una matriz de números enteros, una matriz de flotantes y una matriz de caracteres, podemos crear una matriz de estructuras.

```
Estructura de Facebook para empleados[100];  Un conjunto de estructuras
 Podemos acceder a los datos mediante:
facebook[0].código = 100;
facebook[1].código = 101;
Y así sucesivamente
```

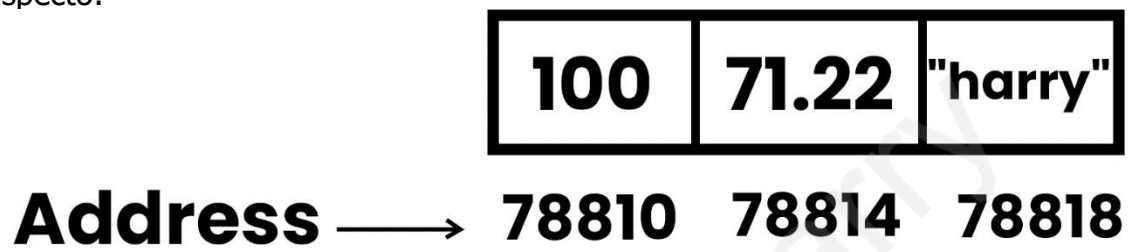
INICIALIZAR ESTRUCTURAS

Las estructuras también se pueden inicializar de la siguiente manera:

```
empleado de la estructura  harry = {100, 71.22, "harry"};
struct empleado shubh = {0}; Todos los elementos establecidos en 0
```

ESTRUCTURAS EN LA MEMORIA

Las estructuras se almacenan en ubicaciones de memoria contiguas. Para la estructura 'e1' del tipo struct employee, el diseño de memoria tiene el siguiente aspecto:



En una matriz de estructuras, estas instancias de empleado se almacenan adyacentes entre sí.

PUNTERO A ESTRUCTURAS

Se puede crear un puntero a las estructuras de la siguiente manera:

```
Empleado de la
estructura  *PTR; ptr =
&e1;
Ahora podemos imprimir elementos de estructura usando:
```

OPERADOR DE FLECHA

En lugar de escribir (*ptr).code, podemos usar el operador de flecha para acceder a las propiedades de la estructura de la siguiente manera:

```
(*ptr).código
o
código > ptr
Aquí -> se conoce como el operador de flecha.
```

PASAR LA ESTRUCTURA A UNA FUNCIÓN

Una estructura se puede pasar a una función como cualquier otro tipo de datos.

```
vacío mostrar(Estructura empleado e); Prototipo de función
```

Quick Quiz: Complete esta función de demostración para mostrar el contenido del empleado.

PALABRA CLAVE TYPEDEF

Podemos usar la palabra clave 'typedef' para crear un nombre de alias para los tipos de datos en C.

'typedef' se usa más comúnmente con estructuras.

```
struct Complejo
{
    flotar real;
    flotar img;  struct complex c1,c2, para definir números complejos
};

typedef struct Complejo
{
    flotar real;
    flotar img;  ComplexNo c1,c2,para definir números complejos
} ComplejoNo;
```

EJEMPLO DE USO

Con el alias typedef, puede declarar variables numéricas complejas de forma más sucinta:

```
ComplejoNo c1, c2;
```

CAPÍTULO 9 – CONJUNTO DE PRÁCTICAS

1. Cree un vector bidimensional usando estructuras en C.
2. Escribe una función 'sumVector' que devuelva la suma de dos vectores que se le hayan pasado. Los vectores deben ser bidimensionales.
3. Se almacenarán veinte números enteros en la memoria. ¿Qué preferirás: matriz o estructura?
4. Escriba un programa para ilustrar el uso del operador de flecha → en C.
5. Escribe un programa con una estructura que represente un número complejo.
6. Cree una matriz de 5 números complejos creados en el problema 5 y muéstrelos con la ayuda de una función de visualización. Los valores deben tomarse como una entrada del usuario.
7. Escribe la estructura del problema 5 usando las palabras clave 'typedef'.
8. Cree una estructura que represente una cuenta bancaria de un cliente. ¿Qué campos utilizaste y por qué?
9. Escribe una estructura capaz de almacenar la fecha. Escribe una función para comparar esas fechas.
10. Resuelva el problema 9 para el tiempo usando la palabra clave 'typedef'.

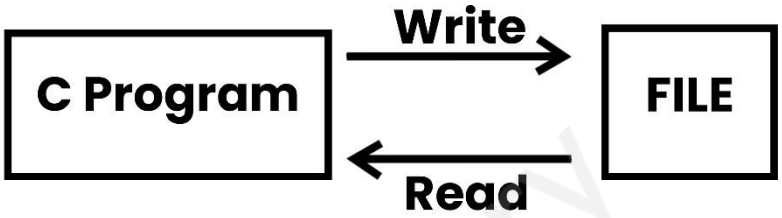
CAPÍTULO 10 – E/S DE ARCHIVOS

La memoria de acceso aleatorio es volátil y su contenido se pierde una vez que finaliza el programa. Con el fin de conservar los datos para siempre, utilizamos archivos.

Un archivo son datos almacenados en un dispositivo de almacenamiento.

Un programa C puede hablar con el archivo leyendo el contenido del mismo y escribiendo contenido en él.

Programmer



PUNTERO DE ARCHIVO

Un "ARCHIVO" es una estructura que debe crearse para abrir el

archivo. Un puntero de archivo es un puntero a esta estructura del

archivo.
(El puntero FILE es necesario para la comunicación entre el archivo y el

programa). Se puede crear un puntero FILE de la siguiente manera:

```
ARCHIVO *ptr;
ptr = fopen("nombredearchivo.ext"; "modo");
```

MODOS DE APERTURA DE ARCHIVOS EN C

C ofrece a los programadores seleccionar un modo para abrir

un archivo. Los siguientes modos se utilizan principalmente

```
"r"   -> abrir para lectura
"rb" -> abierto para lectura en binario
"w"   -> abrir para escritura Si el archivo existe, el contenido se
sobrescribirá
"wb" -> abierto para escribir en binario
"una" -> abrir para añadir Si el archivo no existe, se creará
```

en la E/S de archivos C.

TIPOS DE ARCHIVOS

Principalmente, hay dos tipos de archivos:

1. Archivos de texto (.txt, .c)
2. Archivos binarios (.jpg, .dat)

LECTURA DE UN ARCHIVO

Se puede abrir un archivo para leerlo de la siguiente manera:

```
ARCHIVO *ptr;
ptr = fopen("harry.txt", "r");
int num;
```

Supongamos que "harry.txt" contiene un número entero, podemos leer ese número entero usando:

```
fscanf(ptr, "%d", &num);  fscanf es la contraparte de archivo de scanf
```

Esto leerá un número entero del archivo en las variables Num.

Quick Quiz: Modifique el programa de arriba para verificar si el archivo existe o no antes de abrir el archivo.

CIERRE DEL ARCHIVO

Es muy importante cerrar el archivo después de leer o escribir. Esto se logra usando fclose de la siguiente manera:

```
fclose(ptr);
```

Esto le dirá al compilador que hemos terminado de trabajar con este archivo y que los recursos asociados podrían ser liberados.

ESCRIBIR EN UN ARCHIVO

Podemos escribir en un archivo de una manera muy similar a como leemos el archivo

```
ARCHIVO *fptr;
fptr = fopen("harry.txt", "w");
int num = 432;
fprintf(fptr, "%d", num);
fclose(fptr);
```

FGETC() Y FPUTC()

fgetc y fputc se utilizan para leer y escribir un carácter desde / hacia un archivo.

```
fgetc(ptr);          Se utiliza para leer un carácter de un
fputc('c', ptr);     archivo
```

EOF : FIN DEL FICHERO

fgetc devuelve EOF cuando se han leído todos los caracteres de un archivo. Por lo tanto, podemos escribir un cheque como se muestra a continuación para detectar el final del archivo:

```
mientras(1)
{
    ch = fgetc(ptr);  Cuando se haya leído todo el contenido de un archivo,
¡rompe el bucle!
    if (ch == EOF)
    {
        quebrar;
    }
    código
}
```

CAPÍTULO 10 – CONJUNTO DE PRÁCTICAS

1. Escriba un programa para leer tres números enteros de un archivo.
2. Escribe un programa para generar la tabla de multiplicar de un número dado en formato de texto. Asegúrese de que el archivo sea legible y esté bien formateado.
3. Escriba un programa para leer un archivo de texto carácter por carácter y escriba su contenido dos veces en un archivo separado.
4. Tome el nombre y el salario de dos empleados como entrada del usuario y escríbalos en un archivo de texto en el siguiente formato:
    i. Nombre1, 3300
    ii. Nombre2, 7700
5. Escriba un programa para modificar un archivo que contiene un número entero para duplicar su valor.

PROYECTO 2: SERPIENTE, AGUA, PISTOLA

Serpiente, agua, pistola o piedra, papel, tijeras es un juego que la mayoría de nosotros hemos jugado durante el tiempo escolar. (A veces lo juego incluso ahora).

Escribe un programa en C capaz de jugar a este juego contigo.

Su programa debería ser capaz de imprimir el resultado después de que usted elija serpiente/agua o pistola.

CAPÍTULO 11 – ASIGNACIÓN DINÁMICA DE MEMORIA

C es un lenguaje con algunas reglas fijas de programación. Por ejemplo: No se permite cambiar el tamaño de una matriz.

ASIGNACIÓN DINÁMICA DE MEMORIA

La asignación dinámica de memoria es una forma de asignar memoria a una estructura de datos durante el tiempo de ejecución. Podemos usar la función DMA disponible en C para asignar y liberar memoria durante el tiempo de ejecución.

FUNCIÓN PARA DMA EN C

Las siguientes funciones están disponibles en C para realizar la asignación dinámica de memoria:

1. malloc()
2. calloc()
3. gratis()
4. realloc()

FUNCIÓN MALLOC()

Malloc significa asignación de memoria. Toma el número de bytes que se van a asignar como entrada y devuelve un puntero de tipo void.

Sintaxis:

```
ptr = (int*)malloc(30* sizeof (int))
```

La expresión devuelve un puntero nulo si no se puede asignar la memoria.

Quick Quiz: Escribe un programa para crear una matriz dinámica de 5 flotantes usando malloc().

FUNCIÓN CALLOC()

Calloc significa asignación continua. Inicializa cada bloque de memoria con un valor predeterminado de 0.

Sintaxis:

```
ptr = (float*)calloc(30, sizeof (float));
Asigna espacio contiguo en la memoria para 30 bloques (flotantes)
```

Si el espacio no es suficiente, se produce un error en la asignación de memoria y se devuelve un puntero NULL.

Quick Quiz: Escribe un programa para crear una matriz de tamaño n usando calloc donde n es un número entero introducido por el usuario.

FUNCIÓN FREE()

Podemos usar la función free() para desasignar la memoria. La memoria asignada mediante calloc/malloc no se desasigna automáticamente.

Sintaxis:

```
Gratis(PTR); Se libera la memoria del RPP.
```

Quick Quiz: Escribe un programa para demostrar el uso de free() con malloc().

FUNCIÓN REALLOC()

A veces, la memoria asignada dinámicamente es insuficiente o mayor que la necesaria. Realloc se utiliza para asignar memoria de nuevo tamaño utilizando el puntero y el tamaño anteriores.

Sintaxis:

```
ptr = realloc (ptr, newsize);
ptr = realloc (ptr, 3*sizeof(int));
```

CAPÍTULO 11 – CONJUNTO DE PRÁCTICAS

1. Escriba un programa para crear dinámicamente una matriz de tamaño 6 capaz de almacenar 6 enteros.
2. Utilice la matriz del problema 1 para almacenar 6 números enteros introducidos por el usuario.
3. Resuelva el problema 1 usando calloc().
4. Cree una matriz dinámicamente capaz de almacenar 5 números enteros. Ahora use realloc para que ahora pueda almacenar 10 números enteros.
5. Crea una matriz de tabla de multiplicar de 7 a 10 (7 x 10 = 70). Utilice realloc para
   Que almacene 15 números (de 7 x 1 a 7 x 15).
6. Intenta el problema 4 usando calloc().