Generalidades para programar en C

Vocabulario inicial

- Sintaxis: Se ocupa de la forma en que se escribe y habla, ejemplo, en español decimos "caballo blanco"; en inglés al revés, "white horse".
- Un lenguaje de programación es un conjunto de **instrucciones** que una computadora interpreta y ejecuta. En modo parecido a los lenguajes humanos español, inglés, etc, hay varios lenguajes de programación: C, java, PHP, etc.
- Lenguaje de alto nivel: utiliza **instrucciones** para ordenar qué hacer sin enfatizar en el cómo hacer.

Lenguaje C: Es un lenguaje de programación de propósito general de sintaxis reducida, control de flujo y estructuras sencillas y un buen conjunto de operadores (aritméticos: +, -, ...; relación: <, <=, ... ; lógicos: **no**, **y**, **o**; etc.). No es un lenguaje de muy alto nivel y más bien un lenguaje pequeño (pocos conceptos y tamaño físico reducido), sencillo y no está especializado en ningún tipo de aplicación. Esto lo hace un lenguaje potente, con un campo de aplicación ilimitado y sobre todo, se aprende rápidamente; en poco tiempo, un programador puede utilizar la totalidad del lenguaje. C es a la computación, lo que las matemáticas son a las ciencias: C es el lenguaje de los lenguajes: sistemas operativos, compiladores, desarrollo de otros lenguajes y software de bajo y mediano nivel.

Aprender un lenguaje para humanos va mucho más allá de entenderlo; necesario es pensar, escribir, hablar, comunicar, etc. en él. Un lenguaje humano-máquina, como C, requiere entenderlo, pensar, escribir programas y comentar lo que hacen los programas, por ejemplo:

En matemáticas, un número **n** es par si es múltiplo de 2.
En C un un número **n** es par si **n%2 == 0** // el resto de dividir **n/2** es 0.

Dado un problema complejo, no siempre se piensa en C directamente; primero se desarrolla un algoritmo, puede utilizar matemáticas u otra ciencia, y luego se traduce a C.

Algunas ventajas de programar en un lenguaje como C

- Programar algoritmos complejos
- Como todas las herramientas del mundo virtual, los procesos son muy económicos y rápidos; lo cual es muy bueno para:
  - Utilizar la estrategia **Ensayo y error**, se aprende de los errores. Sea proactivo, no tema equivocarse.
  - Desarrollar el pensamiento abstracto de las personas.

Una desventaja de trabajar en el mundo virtual: Se operan modelos del mundo real.

Código de colores

Para facilitar el aprendizaje, utilizaremos fondos de colores para sugerir la acción a desarrollar:
celeste: Entienda y memorice, generalmente para la sintaxis de sentencias
amarillo: Entienda y revise con atención, generalmente para programas

Conceptos de base

Desarrollador (developer): es un profesional que desarrolla programas en lenguajes de programación.

Programa: Conjunto de instrucciones para ejecutar una tarea, por ejemplo, el siguiente programa da un saludo al mundo:

```
// saludo.c, autor C. Bazán, enero 20XX
#include<stdio.h>
void main(void){
    printf("Hola Mundo");
}
```

Definiciones y Notas:

1) Las letras mayúsculas y minúsculas son distintas; por ejemplo: **include** es una palabra totalmente distinta a **Include**.
2) Se usan verbos y sujetos de una sola palabra. Preferiblemente utilice caracteres del alfabeto inglés y la notación camello, ejemplo:
   numeroPar
   Mayormente use letras minúsculas; una nueva palabra inicia con mayúscula. Las mayúsculas suelen ser usadas para llamar la atención, ejemplo: las constantes se escriben en mayúsculas: PI = 3.141516.
3) **Comentarios**: Son textos, dentro del programa, que sirven solamente para hacer aclaraciones al programador, C no los usa para nada más. Hay dos tipos de comentarios:
   De fin de línea (// ....)**:**
   `#include<stdio.h>` // este comentario es solo para el fin de esta línea
   De varias líneas(/* .... */)**:**

```
/* Este comentario inicia acá,
   puede incluir varias líneas
*/
```

4) Un editor especializado, puede numerar las líneas del programa y dar colores distintos a las palabras para indicar la categoría a la que pertenecen; por ejemplo si es reservada, comentario, o carácter o del programador:
El editor mostrará algo parecido a:

```
01    // saludo.c, autor C. Bazán, enero 20XX
02    #include<stdio.h>              //  Incluye librería estándar que permite activar el teclado y el monitor
03    void main(void){               //  Inicia la función main <principal en inglés>  que inicia la ejecución del programa
04        char  gente[ ]= "Mundo!";  //  Define la cadena de caracteres gente
05        printf("Hola %s\n", gente);//  Imprime el saludo en el monitor.
06    }                              //  Finaliza la función main.
```

Salida:

```
Hola Mundo
```

5) Directivas e instrucciones

Se llama **directiva** a una orden a ser ejecutada en tiempo de compilación, ejemplos:

```
#include<stdio.h>          // Incluye a la librería stdio.h, la cual permite manejar el teclado y el monitor.
#define PI 3.1415          // define la constante PI, atento no lleva = ni ;
```

Se llama **instrucción o sentencia** (viene del inglés sentence) a una orden a ser ejecutada en tiempo de ejecución, termina en "**;**" (automáticamente inicia la siguiente), ejemplos:

```
printf("Hola mundo");  // imprime Hola mundo.
;                      // instrucción vacía: no hace nada; parece ociosa, pero le encontraremos utilidad más adelante.
```

Las instrucciones se pueden agrupar en **bloques** lógicos que comienza con "**{**" y terminan con "**}**", ejemplo:

```
if (a>b) {                          // Si a > b haga lo siguiente:
    printf("Ingrese el radio: ");
    scanf("%f", &radio);
}
```

6) Un programa debe ser **ORDENADO**, para facilitar la "visualización de la lógica". Revise el programa anterior saludo.c, es ¡muy explícito!! es lo ideal, por que ayuda a disminuir las equivocaciones de escritura que son frecuentísimas.

- Preferentemente escriba una instrucción en cada línea, de ser adecuado, escriba más de una.
- Indente las líneas usando el tabulador, no la barra espaciadora; separe bloques lógicos; puede escribir varios espacios en blanco, ejemplo:

```
if (a>b) {
    printf("Ingrese el radio          : ");
    scanf("%f", &radio);
} else{
    printf("Ingrese la circunferencia: ");
    scanf("%f", &circunferencia);
}
```

Pero sería más legible y corto así:

```
if (a>b) {printf("Ingrese el radio          : ");   scanf("%f", &radio);}
else     {printf("Ingrese la circunferencia: ");   scanf("%f", &circunferencia);}
                // para el "else" se copia la primera línea, se modifica y se compara.
```

- Atento al alineamiento de las instrucciones dentro de los bloques:

```
void main(void){
    int i, m = 1, n = 2;
    if(m>1) {
        ……
    } else{
        for(i=0; i<10; i++) {
            …..
        }
    }
}
```

7) Tipos de programa:

Programa **fuente** : Conjunto de instrucciones, en lenguaje C, legibles por el programador, parecido al programa saludo.c
Programa **Objeto** : Traducción del programa fuente a lenguaje de máquina (escrito en binario: 0 y 1).
Programa **Ejecutable**: Programa objeto + librerías auxiliares, se arma en tiempo de ejecución.

8) **Sintaxis:** Regla que cumple una instrucción, ejemplo, para definir y asignar valor a un número entero m, siga la sintaxis:

```
int m;          // Define el nombre y tipo de una variable
m = 4;          // Asigna valor 4 a m.
```

También puede definirse y asignarse en una sola instrucción:

```
int m = 4;
```

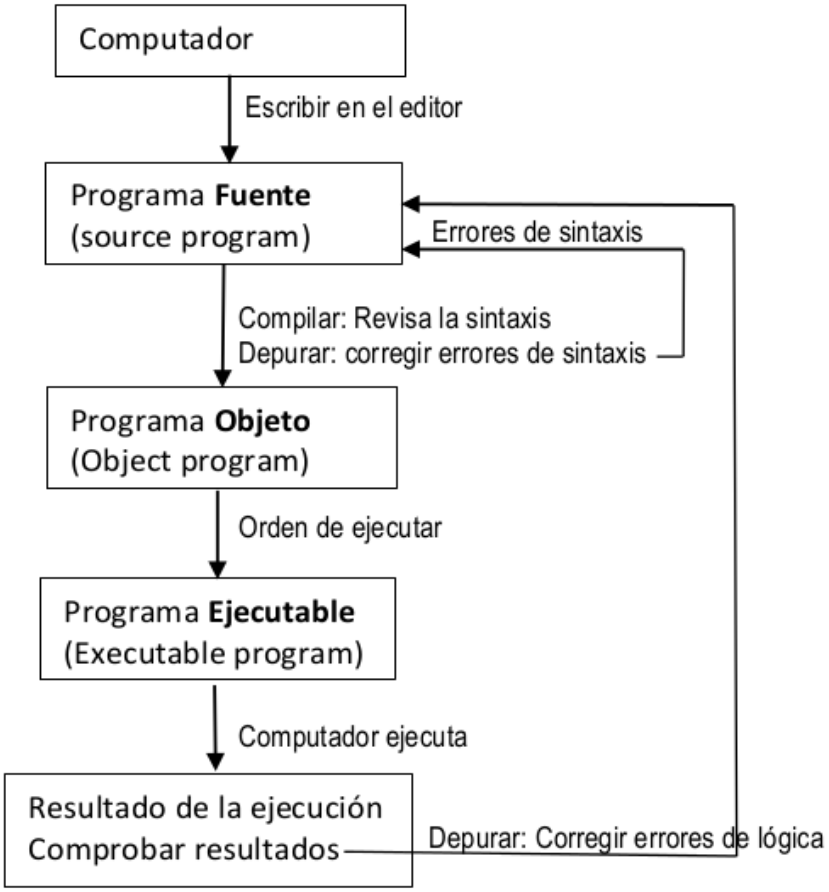
Algunas veces hay partes opcionales. Lo cual se indica con [ ], por ejemplo:

```
if (condicion) bloqueTrue
[else          bloqueFalse]          // Esta parte es opcional.
```

El **compilador** controla que se cumpla la sintaxis de todas las instrucciones de un programa.

9) Recomendación para aprender un lenguaje: puede aplicar 3 procesos:
   a) Entienda el tema: leyendo y/o en clase.
   b) Entienda y memorice: generalmente para la sintaxis de instrucciones
   c) Practicar: aplicar la sintaxis en ejercicios que requieren aplicar la sintaxis en modo repetido

Ciclo de desarrollo de un programa:



Una vez editado (escrito o corregido) un programa fuente, por ejemplo, **saludo.c**; se debe:

- **Salvar** saludo.c: En el editor, teclee **ctrl+s** o **clic** en el botón salvar.

- **Compilar** saludo.c: En la interfaz de comandos, ubíquese en su carpeta de programas y escriba:

  gcc saludo.c

  Si no hay errores de sintaxis, obtendremos el programa **objeto**: **a.out** (o saludo, si así lo ordenó)
  Si hay errores de sintaxis, será necesario depurar el programa: corregir el código, salvar y recompilar, cuantas veces sea necesario.
  Puede indicar un nombre para el programa objeto, por ejemplo saludo, entonces debe escribir:

  gcc saludo.c -o saludo

- **Ejecutar** saludo.c: En la interfaz de comandos, ubíquese en su carpeta de programas y escriba:

  ./a.out (**./saludo** Si indicó compilar con el nombre **saludo**)

  El programa se ejecuta; pero no se garantiza que la salida sea la esperada. El programa es solo una herramienta para ejecutar lo que se programó, no corrige los errores de lógica. Inclusive pueden producirse errores en tiempo de ejecución, no detectables en la compilación.

- **Depurar:** Corregir el código, salvarlo y recompilarlo, cuantas veces sea necesario; siguiendo una técnica que facilite el proceso. Lo estudiaremos un poquito más adelante.

Ejercicios:

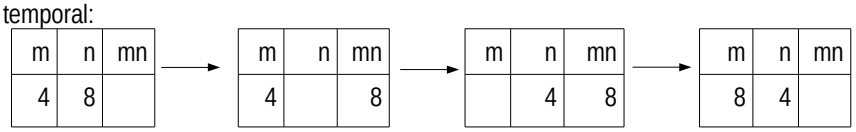
1) Cambie la salida del programa **saludo.c** a: Hola amigos
2) Edite, compile, depure y ejecute un programa que lee un radio y calcula el perímetro y área de la circunferencia.

```
// 01_01.c, autor C. Bazán, enero 20XX
//  Leer un radio y calcular el perímetro y área de la circunferencia
#include<stdio.h>
#define PI 3.1416
void main(void){
        float r;
        printf("Perímetro y área de una circunferencia\n");          // escribe en el monitor
        printf("Ingrese un número mayor que 0: ");
        scanf("%f", &r);                                             // lee del teclado.
        printf("El diámetro de la circunferencia es: %f", 2 * PI * r);
        printf("\nEl área de la circunferencia es: %f\n", PI * r * r);
}
```

Salida:

Perímetro y área de una circunferencia
Ingrese un número mayor que 0: 3 // tecleó **3**
El diámetro de la circunferencia es: 18.849600
El área de la circunferencia es: 28.274400

Ejemplo: Defina dos datos enteros m = 4, y n = 8; luego intercambie los valores. Procedamos como en la vida real; definamos un espacio mn temporal:




```
// 01_02.c : Intercambiar valores de dos variables
#include<stdio.h>
void main(void){
   int m = 4, n = 8, mn;
   mn = n;
   n = m;
   m = mn;
   printf("%d, %d\n", m, n);
}
```

Salida:

8, 4

Programación estructurada

Es un paradigma (forma) para programar. Se demuestra que se es posible programar con solo 3 tipos de instrucciones:

1) instrucciones que se ejecutan en secuencia como en saludo.c
2) Estructuras de decisión:

```
if (4>5) printf("4>5");
else     printf("4<=5");
```

3) Estructuras repetitivas (iteración):

```
for (i = 1; i < 5; i= i+1) {                  // repetir para i = 1, 2, 3 y 4  la siguiente instrucción
    printf("\ni = %d, i+2 = %d", i, i+2);     // imprimir i , i+2
}
```

Si agregamos un cuarto tipo, para dar eficiencia a los programas:

4) Bloques de programa que ejecutan tareas: funciones, como, void main(void) {…}

Estructura del programa fuente

Estructura	Ejemplo	
Documentación del programa	// mundo, autor C. Bazán, enero 20XX	
Incluir programas auxiliares	#include<stdio.h> // incluye rutinas de entrada/salida estándar #include<math.h> // incluye rutinas para cálculos matemáticos	
Definir constantes y variables de uso global (opcional)	#define PI 3.1416 int **m, n**; // antes de main()	
Prototipo de funciones (opcional)	int miFuncion(float x);	
función main (principal)	void main(void){	Inicio
	char gente[ ]= "Mundo!"; printf("Hola %s\n", gente);	Cuerpo
	}	Fin
Otras funciones (opcional)	int miFuncion(float x) { … }	

Ejercicio: Identifique todos la estructura del programa 01_01.c

Elementos de programación

Elementos	Definición	Ejemplos
Variable	Dato que varía	int n; // n toma diferentes valores.
Función	Un bloque de instrucciones que ejecutan una tarea	Función main: void **main**(void) {…..}
Identificadores	Secuencia de letras mayúsculas, minúsculas (no incluye ñ, Ñ, acentos, ni espcios), subguión "_" y números. No empieza en número. Son sensibles al tipo de letra: ma es distinto de Ma Un identificador identifica variables y funciones.	gente, n, N, n1, n1A, n1_a, suma, masGrande, _a **NO**: 12abc, $uno
Constantes	Dato que no varía; por convención, el nombre se escribe con mayúsculas	Se puede definir de dos modos: #define PI 3.1416 const float PI = 3.1416;
Palabras reservadas (por C)	Palabras que no pueden ser usadas como identificadores; al escribirlas toman un color	int, char, float
operadores	Signos para ejecutar operaciones	Aritméticos: +, -, *, /, % Lógicos : &&, ||, ! Relación: == (igual), <, <=, >, >= Asignación: =
Expresiones	Combinación de variables y operadores aritméticos, de relación y/o lógicos	4 // expresión sencilla a + b // da un número a > b // da un valor 0,1
Instrucción	Orden para que el computador ejecute una acción; las instrucciones terminan en **;**	printf("El número es: %d", n);
Directiva (para el compilador)	Se ejecuta en tiempo de ejecución, no termina en **";"**	Incluir la librería: stdio.h #include<stdio.h>

Bloque de instrucciones	Agrupa cero ó más instrucciones dentro de llaves.	{ printf("Ingrese número: ");
		scanf("%d", &n);
		}

Programa	Conjunto de instrucciones (y bloques) para ejecutar una tarea compleja	

Alcance de variables

Concepto base de ejecución: En tiempo de ejecución: El programa se ejecuta de arriba hacia abajo, Se respetan los bloques de instrucciones. Sea:

```
int var;    // una variable entera cualquiera
```

El **alcance** de var, es el espacio en el que var es referenciable (visible) para leer/asignar valores, tiene:

Inicio : al definir var
Fin : al cerrar el bloque que la contiene, también es visible dentro de los bloques subsumidos.

Ejemplo:

```
…..
{ bloque1 }
{                           // bloque2 que contiene a var
    ….
    int var;                // inicio de alcance de var
    …
    { bloque3  }

    { bloque4  }

}                           // Fin de alcance de var: finaliza el bloque que la contiene.
{ bloque5}
….
```

var es visible en parte del bloque 2, y en los bloques 3 y 4.

Un bloque interno, por ejemplo el bloque3, que ve a var, puede definir sus propias variables:

```
{                   // bloque 3
…                   // ve a var, definida antes.
```

```
int m, n, var = 0;    // Inicia una nueva var, con el mismo nombre, pero es otra; se superpone a var definida previamente.
var = 6;              // se refiere a var definida en este bloque 3

}                     // finaliza var definida en este bloquqe
...                   // se ve a var definida en el bloque 2 nuevamente, su valor no fue alterado por la var del bloque 3.
```

Variable global: Es una variable definida al inicio del programa (fuera de todos los bloques) es visible en todos ellos, por lo tanto la pueden leer y asignar valores, ejemplo:

```
#include<stdio.h>
int var = 4;                        // Variable global
void main(void){
    ...
    var = 8;                        // var es visible en main, se cambio el valor a 8.
    x = var+2;
    var = 6;                        // Se cambia el valor de var
}

float miFun(int j){                 // Bloque: { ... }

        return (float)j/var         // var es visible en miFun()
}
```

Precedencia en los operadores: Se ejecutan en el orden:

Matemáticos	relacionales	lógicos	asignación
1	2	3	4

Las siguientes sentencias son equivales:

```
if ((a <  b * 3     &&  c > 1 0 * d   ) . . .
if ((a < ( b * 3 ) ) && (c > (10 * d))). . .
```

Ejercicio: Buscar en internet las reglas de precedencia de los operadores

Tipos primarios de datos

Los principales tipos de datos **primarios** reconocidos por C son:

Tipo	**Palabra reservada ***	**Bytes de memoria requeridos**	**Rango**
Caracter	char	1	-128 a 127
Entero corto	short	2	-32 767 a 32 767
Entero	Int	4	-2 147 483 647 a 2 147 483 647
Precisión simple en punto flotante	float	4	±1.2 × 10 −38 a ±3.4×10 38 (7 dígi-tos de
No existe dato de tipo boolean (lógico): Un número con valor cero, también representa falso : int n=0; Un número con valor distinto de cero, también representa verdad: int n=1;			

* Son palabras básicas definidas por C, el programador las puede usar; pero no redefinirlas.

Al definir una variable se indica su tipo y puede asignarle valor:

```
int i, j, k;          // i, j y k son variables de tipo int.
char c1, c2;          // c1 y c2 son caracteres
float x, y = 1.3;     // x, y son flotantes; el valor 1.3  es asgnado a y.
x = 2.34;             // asigna valor a x.
```

Aritmética con variables tipo int: Es la misma aritmética que utilizamos, excepto en la división (/) y resto (%), ejemplo:

```
int i = 7,  j = 4, m, n;
m = i/j;              // m vale 1. se pierde el resto.
n =  i%j              // n vale 3 = resto de 7/4.
```

La aritmética con otros tipos de datos es similar a la que ejecutamos en matemáticas. Si se mezclan datos de diferentes tipos, por ejemplo al sumar un entero con un flotante, se convierte el entero a flotante y se suman los flotantes. La conversión de tipos se ejecuta en el orden:

char → int → unsigned int → long → unsigned long → long long → unsigned long long → float → double → long double

Conversión de tipo (casting) de dato

Podemos forzar la ejecución del casting; por ejemplo para dividir números enteros:

```
int i = 5, j = 2;
float k;
k = i/j;      // resultado 2 (se hizo una división de enteros), diferente al esperado 2.5
```

Para resolver este problema, utilizamos el operador de casting (float)

```
k = (float)i/j;          // casting a i
```

También pudimos hacer casting a j:

```
k = i/(float)j
```

El operador typecast (sobre una variable **var**) tiene la sintaxis: (type)**var**.

Resultados inesperados

Debido a que los datos tienen asignado un espacio fijo de memoria, al ejecutar operaciones, se pueden ocasionar:

- Desbordamientos (overflow): los datos no alcanzan en la memoria fija
- Pérdida de precisión (los datos, enteros o decimales, menos significativos, se pierden).

Ejemplo:

```
int dato1 = 4, dato3;
long dato2 = 55555555555;
dato3 = dato1 + dato2;          // dato1 convierte su valor 4 a long, se suma y se asigna a un dato de tipo int.
printf("\n%d", dato3);
```

Salida:

```
-279019289
```

La salida es un valor no esperado, debido a un desbordamiento y a menudo no lo notamos. El programador es responsable de que no se produzca errores no reportados por la computadora.

- La precisión se mejora: asignando correctamente los tipos de datos; secuencia adecuada de cálculos y/o utilizando algoritmos de alta precisión para números muy grandes o muy pequeños.
- El desbordamiento se suele controlar: asignando un tipo del dato de mayor nivel.

Entrada de datos, salida de información

Para la entrada de datos al computador desde el teclado se utiliza la función scanf( ) // scan con formato
Para la salida de información del computador al monitor se utiliza la función printf( ) // print con formato
Veamos un ejemplo de entrada de datos del teclado y salida de información al monitor:

```
01  // 01_03.c : Entrada de datos, salida de información
02  #include<stdio.h>                                          // librerías estándar para entrada salida
03  void main() {
04        int    dato1;
05        float  dato2;
06        char   dato3;
07        printf("Ingrese un dato entero, uno flotante y un caracter: ");          // Mensaje para el usuario
08        scanf("%d %f %c", &dato1, &dato2, &dato3);                               // lee datos
09        printf("Dato1 = %d,  Dato2 = %f, Dato3 =  '%c' \n", dato1, dato2, dato3); // imprime información
10  }
```

Salida:

```
Ingrese un dato entero, uno flotante y un caracter: _          // ingrese: 4  14.6  a <enter>
Dato1 = 4,  Dato2 = 14.600000, Dato3 =  a                      // respuesta
```

Veamos en detalle las funciones que nos interesan, **printf()** y **scanf()**

Imprimir información

Suponga los datos:

```
int dato1 =  4;
char dato2 = 'a';
float dato3 = 14.6;
```

Para imprimir información en el monitor se utiliza:

```
printf("Dato1 = %d,  Dato2 = %c, Dato3 =  %f\n", dato1, dato2, dato3);
```

Descripción

Sirve para imprimir los datos dato1, dato2, ... según el **formato** "Dato1 = %d, Dato2 = %c, Dato3 = %f".
El formato puede contener 3 tipos de componentes:

%xx	(uno para cada dato, xx son caracteres), %xx especifica la forma de impresión, ejemplo
%d	para formatear dato1 en tipo entero
%c	para formatear dato2 en tipo carácter
%f	para formatear dato3 en tipo flotante con 6 decimales
%4.2f	para formatear dato3 en tipo flotante con 4 enteros y 2 decimales
%s	para formato de cadena de caracteres.
\x	(x es un carácter), a **\x** se le llama secuencia de **escape** y especifica operaciones de impresión, ejemplo:
\n	Salto de línea
\t	Salto de tabulador

Cualquier otro carácter a imprimir en el monitor

La salida en el monitor será:

```
Dato1 = 4,  Dato2 = a, Dato3 = 14.600000.                    // Al final salta de línea; lo cual se indicó con: \n
```

Hay muchos más detalles sobre los formatos, puede revisarlos en internet.

Leer datos

Suponga los datos:

```
int dato1;
char dato2;
float dato3;
```

Para leer valores desde el teclado se utiliza:

```
scanf("%d %c %f", &dato1, &dato2, &dato3);
```

Descripción

Sirve para leer datos para dato1, dato2, ... según el **formato** "%d %c %f".
El formato del scanf() es similar al de printf(); solo que no se usan secuencias de escape **\x,** y los caracteres se suelen limitar a espacios en blanco.
Atento las variables van precesidas por &
Comparación de las funciones de entrada/salida:

```
scanf("%d %c %f"                          , &dato1, &dato2, &dato3);
printf("Dato1 = %d,  Dato2 = %c, Dato3 =  %f\n",   dato1,   dato2,   dato3);
```

El usuario puede tipear tres valores, ejemplo: **4 a 14.6** separados por un espacios en blanco y asignará:

```
dato1 =  4
dato2 = 'a'
dato3 = 14.6
```

Al tipear los datos, sepárelos por uno o más blancos o por la tecla enter.

Muy importante:

- Cumpla las indicaciones que se le hacen al momento de teclear datos, si le piden:

  Ingrese un entero > 0: __ // escriba, por ejemplo: **99<Enter>**

  No escriba letras, espacios en blanco, símbolos etc., esto corromperá los datos de ingreso y no funcionará el programa.

- Si va a leer más de un dato es conveniente que use un formato simple, ejemplo:

```
scanf("%d %d, %c", &n, &m, &c);        // lee dos enteros y un carácter
tipee por ejemplo:  4 223 z            // separe con un espacio
```

No escriba formatos como:

```
scanf("  %d %d,   %c ", &n, &m, &c);    // esto le dará problemas al momento de ingresar datos.
```

Más adelante aprenderá a controlar estas incidencias, pero por el momento, sea disciplinado y evítese problemas.

Veamos un programa que ilustra el manejo de formatos:

```
// 01_04.c : Equivalencia de caracteres y números
#include<stdio.h>
void main(void){
      int n = 65;
      char ch;
```

```
    printf("%c\n", n);          // n se comporta como carácter:   si 0<= n <= 255
    printf("%d\n", n);          // n se comporta como entero
    if(n)   printf("true\n");   // n se comporta como booleano:  n !=0 → true;   n== 0 → false
    else   printf("false\n");
    printf("----------\n");
    ch = n;                     // Asigna entero a caracter
    printf("%c\n", ch);         // se comporta como carácter:     si 0<= n <= 255
    printf("%d\n", ch);         // se comporta como entero
    if(n)   printf("true\n");   // n se comporta como booleano:  n !=0 → true;   n== 0 → false
    else   printf("false\n");
}
```

Salida:

```
A
65
true
----------
A
65
true
```

Ejercicio: Profundice más en el tema de formatos en internet: busque: Lenguaje C **printf**

Equivalencias de tipos de dato numérico, caracter y booleanos

En **C** existen tipos de datos:

numérico como: int, float, etc.
caracter: char

Una variable de tipo **entero entre 0 y 255** se puede interpretar como un **carácter** (con operaciones limitadas), en un contexto de **carácter**.
Una variable de tipo carácter se puede interpretar como un **entero entre 0 y 255** (con operaciones limitadas a sumas y restas), en un contexto de **número.**
Ejemplo:

```
#include<stdio.h>
void main(void){
    int n = 65;
    char c = 'A';               // En la tabla ascii, el código de A es 65
    printf("%d %c\n", n, n);    // n se imprime con formato numérico y carácter  → 65 A
    printf("%d %c\n", c, c);    // c se imprime con formato numérico y carácter  → 65 A
    n++;                        // suma 1
    c++;                        // suma 1
    printf("%d %c\n", n, n);    // n se imprime con formato numérico y caracter  → 66 B
    printf("%d %c\n", c, c);    // c se imprime con formato numérico y caracter  → 66 B
}
```

Salida:

```
65  A
65  A
66  B
66  B
```

En **C No** existe el tipo booleano:

Una variable **numérica** se interpreta como **booleana**, en un contexto booleano:
Un valor **distinto de cero** se interpreta como **verdad.**
Un valor **cero** se interpreta como **falso.**

Ejemplo:

```
#include<stdio.h>
void main(void){
    int n = 65;
    char c = 'B';                   // Código ascii 66
    if (n) printf("n = %d\n", n);   // imprime 65
    n = !n;                         // n(verdad) → no n = falso = 0 → n = 0
    if (n) printf("n = %d\n", n);   // no imprime porque n == 0
    n = c + 1 + !n;                 // !n = 1 → n = 66 + 1 + 1 = 68
                                    //  Se sumó: char + int + booleano
    if (n) printf("n = %d\n", n);   // imprime 68
```

```
}
```

Salida:

```
n = 65
n = 68
```

Operadores de asignación complejos

Hay instrucciones que se usan con mucha frecuencia; para ellas hay formas cortas (atajos) de escribirlas

Operación	Ejemplo de escritura	
	Larga	**corta**
Sumar 1 a una variable, en dos modos	j = j +1;	++j: // preincremento J++; // postincremento
Restar 1 de una varible, en dos modos	j = j -1;	--j // predecremento j--; // postdecremento

+ - * / una variable a otra	j = j + k;
	j = j – k;
	j = j * k;
	j = j / k;
		j += k;
		j -= k;
		j *= k;
		j /= k;

Ejemplo:

```
int m, n = 1;
m = ++n;          // resultado:  m = 2    n = 2   Primero se incrementa n y luego se asigna n a m.
m = n++;          // resultado:  m = 2    n = 3   Primero se asigna n a m y luego se incrementa n.
m += n * 2;       // resultado:  m = 2 +  3 * 2 = 8, n = 3.
```

Uso de internet

Internet, por medio de sus motores de búsqueda, es una gran fuente de información, sobre temas puntuales de computación, por ejemplo para buscar información sobre **tipo de datos**, visite google y teclee:

Lenguaje C tipo de dato

Aparecerá una lista grande de ayudas.

También se ofrecen tutoriales, libros gratis y pagados; y foros. El mayor inconveniente con internet es que las ayudas suministradas no suelen integrar conceptos, y obviamente, las interacciones entre varias personas son dificultosas.

Fases de un programa

Un programa tiene 4 fases:

1) **Análisis y diseño** (Los ponemos juntos, porque es de gran ayuda hacerlo así): El análisis especifica el qué hacer, y el diseño un algoritmo de cómo hacerlo. Para hacer un programa intervienen, cuando menos: El usuario y el desarrollador (programador).
2) **Programa**: instrucciones en lenguaje C.
3) **Ejecutar**: Se ejecuta el programa, se ingresa los datos de entrada en el teclado y se obtienen las salidas.
4) **Verificar resultados y corregir**: Es muy frecuente que los resultados no sean los esperados, debido a errores de lógica, lo cual es difícil de resolver, ya que no hay mensaje de error y los resultados se muestran; pero son incorrectos y muchas veces usted no se da cuenta.

Depurar un programa

Depurar es, corregir los errores; en inglés se escribe **debug** y se pronuncia **dibáág;** la traducción literal en el argot popular sería eliminar insectos o microbios (algo molestoso). Un programa complejo puede tener errores en cada una de sus fases de desarrollo, peor aún varios errores entremezclados, lo mismo que sucede en la vida real; lo bueno del mundo virtual es que los tiempos son cortos, y las urgencias no son dramáticas. Presentamos las fases con sus errores asociados y las estrategias recomendadas para corregirlos:

1) **Análisis diseño y programación**: las fuentes de error pueden ser:
   - Error del usuario al especificar su requerimiento
   - Error del programador al mal interpretar el requerimiento.
   - Algoritmo de solución incorrecto
   - Programa con errores
   - combinaciones de los errores anteriores.

   Para corregir estos errores, se recomienda:
   - Trabajar en equipo para completar las especificaciones y algoritmos de solución
   - Definir la **matriz de casos de prueba** de los programas, la cual es parte del documento de aceptación del programa, se llena antes de escribir el programa y tiene la forma:

Matriz de casos de prueba

Caso	Descripción (opcional)	Entradas (variables independientes)			Salidas (variables dependientes)			Chequeo (se chequea al final con el usuario)
		var1	var22	...	sal1	Sal2	...	
1								
2								
...								

Todas las columnas, excepto la de chequeo, se llenan de antemano; cada caso es una condición crítica de las variables de entrada a controlar.

La columna de chequeo se llena al momento de entrega. Cada fila se chequea como buena, si al correr el programa con la entrada especificada se obtiene la salida especificada.

La matriz de prueba ayuda a chequear los casos críticos; pero no garantiza que la lógica sea correcta u óptima

2) **Mis dos inolvidables &**:

- Para editar un programa, ejemplo, miPrigrama.c, utilice el comando:

  gedit miPrograma.c**&**

  Si no escribe **&**, el sistema operativo se dedica únicamente a editar miPrograma.c y no procesa comandos (no aparecerá el **prompt**); ni siquiera podrá compilar a miPrograma.c. Si esto le ocurriera: salve el programa, ubíquese en la ventana **de comandos** y ordene **ctrl+c** para "matar" al comando gedit; aparecerá el **prompt**; vuelva a editar: gedit miPrograma.c**&**.
- Cuando use la instrucción **scanf** para leer datos, por ejemplo:

  scanf("%d, %f", **&**datoEntero, **&**datoFlotante);

  Si no escribe **&** delante de cada variable, al compilar le saldrá un **warning** (advertencia) difícil de entender; un warning no es error, compilará; pero puede ejecutar mal; el motivo de este caso raro será explicado en el capítulo de funciones. Ejemplo:

```
01   #include<stdio.h>
02   void main() {
03         int  mm;
04         float ff
05         printf("Ingrese un número entero y uno flotante: ");
06         scanf("%d %ff", &mm, &ff);          // se usa &
07         printf(""%d %f",    mm,   ff);      // no se usa &
08   }
```

Salida:

```
Ingrese un número entero y uno flotante: 2 4.1
2   4.100000
```

3) **Programación**: Se presenta dos tipos de error:

- Compilación: El compilador revisa que las instrucciones del programa cumplan con las reglas de sintaxis, y reporta los errores: línea de ocurrencia y descripción del error. Para la programación se recomienda:
  - Cerrar, DE INMEDIATO, todo lo que abra: { }, [ ], ( ), " "; para evitar olvidarse de cerrar; ejemplo:

    void main(void) { ..};

    El compilador da un mensaje de error muy confuso cuando falta un cierre y perderá tiempo para detectarlo.
  - No se olvide del punto y coma "**;**" al final de cada instrucción.
  - No es necesario corregir todos los errores; ya que es muy frecuente que un error de código genere muchos errores de sintaxis; así es que corrija los primeros 1, 2, ó 3 errores de sintaxis y vuelva a compilar.
- Lógica: En este caso el programa ejecuta pero los resultados (de la matriz de prueba) no son correctos. Hay dos fuentes de error:
  - Algoritmo incorrecto
  - Programación incorrecta

  Recomendaciones:
  - indentar las instrucciones del programa con tabulaciones (**tecla tab**) para hacer visibles los rangos de los bloques del programa, ejemplo:

```
void main(void) {
    int i, j;
    for (...) {
        ……;
        …;
    }
};
```

    La indentación "visibiliza la lógica".
  - Compare el programa contra el algoritmo

- Se pueden presentar eventos inesperados:
  - El computador se bloquea (cuelga - no termina nunca – entra en loop): presione Cterl+c
  - El programa termina sin dar ningún resultado o dar resultados absurdos: escriba **trazas**. Una **traza** es un mensaje (que se imprime en pantalla y lo ve), se escriba en partes estratégicas, hay dos tipos:
    - **De recorrido del programa**: printf("1111111"); printf("2222222"); … // permite identificar el paso por la traza. Se repite el número para distinguirlo en pantalla, muchas veces aparecen un montón de símbolos y no se sabe donde aparece la traza.
    - **De valores**: printf("111111: x = %d, y = %d", x, y); // se usa para conocer los valores que toman las variables

    Puede combinar recorrido con valores.
- Verifique la matriz de casos de prueba.

4) **Mi estrategia personal para desarrollar**: Un programa es un documento escrito para que una máquina -que no piensa; pero si es obediente y rápida- lo ejecute, una buena estrategia para trabajar con este asistente es balancear dos enfoques:
   - Flujo de trabajo: Problema → algoritmo → programa → prueba
   - **Ensayo y error:** sea proactivo para programar, aviéntese aunque se equivoque varias veces; por los siguientes motivos:
     - La máquina no se molesta, ni opina nada de usted; aunque se "eqiboque" un millón de veces.
     - En programación, Los errores cuestan solo unos pocos segundos, son fáciles de detectar, y se aprende del error.
     - Los problemas son complejos y es muy probable que se equivoque.
     - La compilación, que controla la sintaxis, es muy quisquillosa y suele arrojar una larga lista de errores, no se asuste, no los corrija todos, solo los primeros, limpie su pantalla (ctrl+l) y vuelva a compilar.

   Una estrategia parecida empleo yo para hacer otras cosas, ¿para qué será?

Ejemplo: Escriba un programa que lea dos enteros m y n mayores que 0 y ejecute las 4 operaciones básicas con ellos:

```
// 01_05.c : Leer dos datos y ejecutar las 4 operaciones aritméticas elementales
#include<stdio.h>
void main(void){
    int m, n, o;
    printf("Ingrese un entero m = ");    scanf("%d",&m);
    printf("Ingrese un entero n = ");    scanf("%d",&n);
    printf("suma  = %d\n", m+n);                          // suma
    printf("Resta = %d\n", m-n);                          // resta
    printf("Multiplicación = %d\n", m*n);                 // multiplicación
    printf("División = %.2f\n", (float)m/n);              // división, atento al formato y casting a flotante
}                                                         // puede haber división por 0
```

Salida:

```
m + n = 5
m - n = 1
m * n = 6
m / n = 1.50
```

Matriz de prueba:

Caso	Entradas		Salidas				Chequeo
	m	n	m+n	m-n	m*n	m/n	
1	3	2	5	1	6	1.50	√
2	9	3	12	6	18	3	√
3	5	0	5	5	0	Divisor 0	Esta prueba se completará el próximo capítulo

La columna **Chequeo** se llena línea por línea para cada prueba del programa.

Este en un problema muy sencillo, sin embargo requiere 3 casos de prueba (depende de m/n).

Ejecute el programa para cada caso de prueba: Ingrese los datos de entrada y verifique que la salida concuerda con la esperada. Si no es así; corrija el programa, recompile, ejecute y verifique nuevamente los casos de prueba. Sea cuidadoso al depurar, ya que al corregir un error puede producir otro y usted no se da cuenta, lo cual quiere decir que debieran repetirse todos los casos de prueba; afortunadamente, la experiencia y la buena lógica nos permiten correr el riesgo de no probarlo todo, una y otra vez.

Ejemplo: Lea tres datos flotantes a, b y c que son los coeficientes de la ecuación cuadrática $ax^2 + bx + c = 0$. Calcule y reporte las raíces de acuerdo al valor del discriminante:

La matriz de casos de prueba toma la forma:

Caso	Descripción	Entradas			Salida		Chequeo
		a	b	c	raiz1	raiz2	
1	$b^2 - 4ac > 0$						
2	$b^2 - 4ac = 0$						
3	$b^2 - 4ac < 0$						

Para determinar las entradas, tengamos en cuenta que la ecuación:

$b^2 - 4ac = 0$

Tiene 3 variables, por lo tanto podemos dar valores, que faciliten los cálculos, a dos variables, ejemplo: a = 1, c = 1; la ecuación se convierte en:

$b^2 - 4 = 0$

Resolviendo:

Para $b^2 - 4ac = 0 \rightarrow b = 2$
Para $b^2 - 4ac > 0 \rightarrow$ elegimos b = 3
Para $b^2 - 4ac < 0 \rightarrow$ elegimos b = 2

Calculemos las raíces. Finalmente la matriz de **Casos de Prueba** será:

Caso	Descripción	Entradas			Salida		Chequeo
		a	b	c	raiz1	raiz2	
1	No cuadrática	0			No es ecuación cuadrática		
2	$b^2 - 4ac > 0$	1	3	1	-0.382	-2.618	
3	$b^2 - 4ac = 0$	1	2	1	-1	-1	
4	$b^2 - 4ac < 0$	1	1	1	-.5 + i 0.866	-.5 - i 0.866	

Ejemplo: Un programa para combatir el insomnio: Muestra en su monitor:

3 ovejitas
6 conejitos
9 ovejitas
12 conejitos
…
96 conejitos
99 ovejitas
96 conejitos
….
6 conejitos
3 ovejitas

Observe que se muestra:

Un número n que inicia en 3, aumenta de 3 en 3 hasta llegar a 99, luego desciende del mismo modo.
Un animalito que depende de n:
conejitos si n es par
ovejitas si n es impar

Matriz de casos de prueba

Caso	Descripción	Entrada	Salida	Chequeo
		n		
1	n impar (inicio)	3	3 ovejitas	
2	n par (inicio)	6	6 conejitos	
3	Antes de cambio	96	96 conejitos	
4	Cambio	99	99 ovejitas	
5	Después del cambio	96	96 conejitos	
6	n par (fin)	6	6 conejitos	
7	n impar (fin)	3	3 ovejitas	

Al inicio la depuración parece un proceso tedioso, pero con la práctica se facilita. Este proceso de control y corrección, en el mundo virtual consume tiempos y recursos y, es similar al que ejecutamos en el mundo real, con la diferencia de que en el mundo real, los recursos son muy caros, y los tiempos son muy largos, podemos sacar varias conclusiones, entre otras:

- Programar y depurar es una gran herramienta que facilita el desarrollo del pensamiento organizado en el mundo real.
- La velocidad del mundo virtual, facilita la detección de los problemas invisibilisados por la lentitud del mundo real.
- La batería de metodologías y pruebas desarrolladas dan mayor velocidad y precisión en las soluciones.
- Tomando en cuenta los indicadores "adecuados", se puede recomendar virtualizar muchas actividades del mundo real.
- Ahhh y algo muy importante: En el mundo virtual los costos son mucho menores, la máquina no se cansa, le espera, no le regaña, no chismosea, etc. Atrévase a actuar en el mundo virtual, ¡equi**B**óquese, rompa los platos!, son gratis… Una pregunta filosófica: ¿Usted cree que su cerebro y el contenido de él, es real o virtual? Mmmm???. Usted tiene mucho de virtual y no lo sabía; pues ya lo sabe, no se espante, y compórtese como lo que es: Una integración de materia real y virtual. Le dije materia virtual, sí lo virtual tiene una base física, no es solo fantasía inmaterial.

Ejercicios

Escriba, compile y ejecute programas que hagan lo siguiente:

1) Solicite la temperatura en grados Celcius **tgc**, luego calcule la temperatura en grados Farenheit **tgf** mediante la fórmula: tgf = 1.8 * tgc + 32 y finalmente la muestre.
   Sugerencia:
   Defina tgc y tgf
   Lea tgc, calcule tgf, imprima tgc y tgf

2) Lea un número entero **ne** de tres cifras y muestre las unidades **u**, decenas **d**, y centenas **c**.
   Sugerencia, si: ne = 437;

```
u  = ne%10;                    // 7 enteros
ne = (ne – u)/10;  d =  ne%10  // 3 decenas
c  = (ne – d)/10;              // 4 centenas
```

3) Solicite un número de segundos **ns** y reporte el número de horas **nh**, minutos **nm** y segundos **ns** equivalentes.
   Sugerencia, si: ns = 4562:

```
nh = ns/3600;                          // 1
ns = ns – nh * 3600;     nm = ns/60;   // 962; 16
ns = ns – nm * 60                      // 2
```

4) Solicite un número flotante **x** y calcule e imprima **x**, su cuadrado y su raíz. Agregue el include #include<math.h> al inicio del programa y compile con: **gcc 4.c -lm**

5) Lea dos números m y n e intercambie su valor.