Lenguaje C

Sintaxis

El lenguaje C facilita un método estructurado y disciplinado para el diseño de programas de computación.

En el siguiente capítulo presentaremos las funciones elementales para trabajar en el lenguaje C.

Al comenzar a trabajar en cualquier IDE (Entorno de desarrollo integrado para desarrollar un programa) aparecerá:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Hello world!\n");
   return 0;
}
```

Analicemos cada una de las líneas de este código

Las líneas `#include <stdio.h>` e `#include <stdlib.h>`

Son directrices del procesador de C.

C tiene ciertos procesos ya desarrollados y guardados ordenadamente, según sus aplicaciones, en distintas carpetas: entrada/salida, pantalla, string, matemática, etc.. Estas carpetas se llaman en general **bibliotecas**.

Las líneas que se inician con # son procesadas o ejecutadas por el procesador, antes de la compilación del programa. Esta línea le indica al procesador que incluya dentro del programa, el contenido del "archivo de cabecera de entrada/salida estándar (stdio.h)".

Este archivo de cabecera contiene información y declaraciones utilizadas por el compilador al compilar funciones estándar de la biblioteca de entrada/salida como son: scanf (leer) y printf (mostrar).

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

La línea int main ( )

Forma parte de todo programa en C.

Los paréntesis después de *main* indican que es un bloque constructivo del programa conocido como una función.

Todos los programas de C empiezan a ejecutarse en la función *main*.

La llave izquierda **{** inicia el **cuerpo** de la función *main*. La llave derecha **}** da por terminada la función *main*.

Este par de llaves y el conjunto de instrucciones que abarca recibe el nombre de **bloque**.

Antes de la llave de cierre, colocamos “return 0;”. Esta instrucción brinda información de cómo el programa finaliza su ejecución. Todo programa que ha terminado exitosamente retorna el valor 0 (cero) al sistema operativo

La línea printf ("Hello World!\n");

printf ( ) es una función que permite mostrar o imprimir en la pantalla una leyenda.

Los paréntesis se escriben después del nombre de toda función para escribir entre ellos el argumento.

En este programa se quiere mostrar la cadena de caracteres enunciada entre comillas dobles; al ejecutarlo, aparecerá en la pantalla ***Hello World***!.

La cadena de caracteres escrita entre comillas también se llama **mensaje, leyenda o literal**, y al mostrarla se respetan exactamente todos los caracteres que aparecen allí, salvo la barra invertida ´\´y el carácter escrito inmediatamente después.

La \ se llama **carácter de escape** y siempre se combina con el siguiente carácter:

\n significa línea nueva (el cursor se coloca al comienzo de la siguiente línea).

\t tabulador horizontal

**** imprime una diagonal invertida \

\% imprime el símbolo %

\" imprime la doble comilla

Para indicar que luego de una instrucción se debe seguir en secuencia con la siguiente línea, todo enunciado debe terminar con ";" (punto y coma).

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

En el programa se puede indicar o agregar una línea como comentario para documentar los programas y mejorar la legibilidad de los mismos.

Si el comentario ocupa solo **una línea**, se coloca al comenzar el mismo //, si ocupa **más de una línea**, comienza con **/*** y termina con ***/**. Al ejecutar el programa los comentarios no realizan ninguna acción. Son instrucciones no ejecutables.

Variables, Tipos y Declaración:

Una **variable** es una posición en memoria donde se puede almacenar un valor. Antes de usar una variable en un programa se la debe declarar, decir de qué tipo es (para determinar cuántos bytes se reservan en memoria para almacenar su valor) y cuál es su nombre (para poder nombrarla en el programa). Es conveniente declararlas todas inmediatamente después de la llave de inicio del main.

Los nombres de las variables son cadenas de letras y dígitos de 1 a 32. Se acepta el guión bajo en los nombres nom_1, pero no puede comenzar con dígitos.

En C se consideran distintas las minúsculas de las mayúsculas. Es distinta la variable contc a la variable contC.

Todo se escribe en minúscula salvo las constantes y otros casos que ya se detallarán.

Declaración:

La forma general para la **declaración** de variables:

```
Tipo      nombre variable;
```

Tabla de datos:

TIPO	DESCRIPCIÓN	PALABRA CLAVE	FORMATO y TAMAÑO
Entero	Número sin decimales entre -32768 y 32767	Int	%d 2 byte
Entero largo	Número sin decimales entre ±2.147.483.647	Long	%ld 4 byte

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Punto flotante	Número con decimales entre 3.4E-38 decimales(6)	Float	%f 6 byte
Doble punto flotante	Número con decimales entre 1.7E-308 decimales(10)	Double	%lf 8 byte
Caracter	Un carácter	Char	%c 1 byte
String	Cadena de caracteres	char variable[longitud]	%s tantos byte como caracteres

Por ejemplo, para declarar la variable a como flotante, b como entera, nom como una cadena de caracteres de a lo sumo 10 caracteres y z como caracter:

```
float a;
int  b;
char nom[10];
char z;
```

Operadores Aritméticos:

OPERADOR	ACCION
_	Resta
+	Suma
*	Multiplica
/	Divide
%	Resto de la división

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

- -	Decrementa
+ +	Incrementa

Operadores Relacionales y Lógicos:

Los **operadores relacionales** son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Operadores relacionales:

OPERADOR	SIGNIFICADO
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
= =	Igual
!=	No igual

Los operadores lógicos retornan un valor lógico basados en las tablas de verdad.

Operadores lógicos:

OPERADOR	SIGNIFICADO
&&	And (y)
| |	Or (o)

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

!	Not (negación)

La precedencia es:

Más alta

- !
- >= > < <=
- = = !=
- & &
- | |

Más baja

Operadores de asignación:

En C, el operador de asignación es un único signo = (igual).

Una **asignación** significa que la variable que se encuentra a la izquierda del = toma el valor que contiene la variable de la derecha.

Si a la derecha hay un operando o más, primero se resuelve esa expresión y luego se asigna.

Variable 1 = Variable 2;	x = y;

Variable 1 = Variable 2 operador Variable 3;	x = y + z;

C me permite un formato de "atajo" en el siguiente caso:

Variable1 = Variable 1 operador expresión;

Por ejemplo:

x = x +10;

y = y /z;

Si la variable está a la derecha y a la izquierda del =, se pueden abreviar como:

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Variable 1 operador = expresión;

x = x + 10; por **x +=10;**

y = y/z; por **y / = z;**

x = x - 23; por **x -= 23;**

x = x * y; por **x *= y;**

La variable 1 se debe inicializar, según corresponda, en la declaración.

```
int x = 0;
```

Para tener en cuenta!!

- Si un operador aritmético tiene operandos enteros se ejecuta una operación entera (sin decimales):

  5/9 da 0 (cero)

- Si un operador aritmético tiene un operando en punto flotante y otro entero, este último será convertido a punto flotante antes de hacer la operación

  5.0/9 da 0.55

- En una asignación del tipo f = ini;

  Donde fueron declaradas como

  ```
  float f;
  int ini;
  ```

  Convierte **ini a float** antes de la asignación.

- En una decisión del tipo

  ```
  if ( f <= ini)
  ```

  Convierte **ini a float** para la comparación.

Printf y Scanf (instrucciones de e/s):

Para mostrar por pantalla o imprimir utilizamos la función **printf()**.

Existen dos formas de escribir esta función, con un argumento o con más de un argumento.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

```
printf ("argumento1");
```

Si tiene solo un argumento es porque es una leyenda o secuencia de caracteres donde se respeta la escritura entre comillas dobles.

printf ("leyenda");

Dentro de esta leyenda no hay variables.

Por ejemplo: printf(“Este es un ejemplo”);

```
printf( "argumento 1" , lista de  argumentos);
```

```
printf ("leyenda que contiene uno o varios formatos de variables”, nombre de la o las variables);
```

Si hay más de una variable se deben nombrar en el orden en que sus formatos fueron escritos dentro de la leyenda.

Por ejemplo: printf(“Juan tiene %d años y Pedro %d ”, añosjuan , añospedro);

Para ingresar datos por teclado utilizamos la instrucción **scanf().**

```
scanf( "argumento 1" , &argumento 2);
```

El primer argumento es la cadena de control de formato, especifica el tipo de dato que debe entrar el usuario y va entre comillas dobles.

El segundo argumento empieza con **un ampersand & u operador de dirección,** seguido por el nombre de la variable cuyo formato se declaró como argumento 1.

Ejemplo

¿Cómo sumar 2 números enteros?

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

```
# include < stdio.h>
int main ( )
{
int a, b, sum;                          /* declaración de variables*/
printf ("Ingrese el primer número entero\ n");
scanf (" %d" , & a);        /* lectura del primer número entero a */
printf ("Ingrese el segundo número entero \n");
scanf ( "% d" , &b);      /* lectura del segundo número entero b*/
sum= a+b;                                /* asignación en sum */
printf ( " La suma es % d", sum);          /* muestra la suma */
return 0;
```

La línea: **int a,b,sum;**

Corresponde a una **declaración**: a,b y sum son los nombres de las variables.

Una variable es una posición en memoria donde se puede almacenar un valor para uso de un programa.

Esta declaración especifica que a, b y sum son enteros, y reserva en memoria el lugar correspondiente, según el tipo, para cada variable.

Las variables se declaran enseguida de la llave izquierda del main, antes de ser utilizadas en el programa.

La línea:

printf ("Ingrese el primer número entero \ n") ;

Muestra la leyenda en la pantalla y coloca el cursor al principio del renglón siguiente, por terminar con \ n.

La línea:

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

scanf ("% d", &a);

Usa la función **scanf** para que el usuario entre por teclado un valor entero.

La función scanf toma la entrada del teclado.

Scanf tiene en este caso 2 argumentos "%d" y & a.

Recordemos:

La instrucción **scanf** se escribe de la siguiente forma:

scanf("argumento 1" , & argumento 2);

El primer argumento es la cadena de control de formato, especifica el tipo de dato que debe entrar el usuario y va entre comillas dobles.

El segundo argumento empieza con un ampersand & u operador de dirección, seguido por el nombre de la variable cuyo formato se declaró como argumento 1.

En caso de las variables carácter o las string el & es opcional.

&a le indica a scanf la posición de memoria en la cual está almacenada la variable a. La computadora almacena el valor de entrada por teclado en esa posición de memoria.

Importante

En los scanf delante de las variables **se coloca &,** salvo excepciones. En caso de no colocarlo, no guarda la dirección de memoria donde la almacenó y luego encontrarla es casi imposible, utilizando cualquier valor en su reemplazo. Este valor se lo conoce como **basura.**

Esta secuencia printf y scanf facilita la interacción entre el usuario y la computadora. Ya que con el printf me muestra una leyenda que me indica que hacer, que dato ingresar y el scanf me permite ingresar por teclado los datos.

Si sólo colocamos el scanf se muestra en la pantalla un cursor titilando y al no tener una indicación sobre que está esperando como entrada: mi edad, mi nombre, mi sueldo… se hace poco amigable para el usuario.

Por eso es recomendable colocar un printf() y luego el scanf().

La línea:

printf ("La suma es %d", sum);

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Utiliza la función printf para mostrar por pantalla la leyenda: ”La suma es” seguida del valor numérico de la variable sum.

Importante

Este printf tiene 2 argumentos:

- El primer argumento es la leyenda que incluye dentro de las comillas dobles %d que indica que se mostrará un número entero en ese lugar.
- El segundo argumento especifica el nombre de la variable que contiene el valor para ser impreso

Es decir al mostrar por pantalla, en la leyenda se reemplaza el %d por el valor que tomó sum.

Dentro de un printf también se pueden hacer cálculos.

Se puede reemplazar:

```
sum= a+b;

printf ("La suma es %d", sum);

Por:  printf("La suma es % d", a+b);
```

y en este caso no es necesario el uso de la variable sum.

Para ser más clara la salida podría ponerse:

printf ("La suma entre los números %d y %d es %d", a,b,sum);

Si en el primer argumento hay varios controles de formato significa que serán reemplazados por los valores de las variables del segundo argumento en el orden en que están enumeradas.

Primer %d corresponde al valor de a

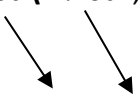
Segundo %d corresponde al valor de b

Tercer %d corresponde al valor de sum

Importante

- **Para justificar dentro de un campo a la derecha**

printf ("%6d**\ n**%6d", a,b);



© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

campo de 6 lugares

Salida:

Si quiero mostrar 2 números en distinto renglón y alineados a derecha, debo saber cual es la cantidad máxima de dígitos y colocar ese número antepuesto a la d.

Por ejemplo si tengo 2 números y el que ocupa más es de 6 dígitos

123 de los 6 lugares ocupa los 3 últimos

154378 considera el campo de 6 lugares alineando de derecha a izquierda

- **Para mejorar la salida de puntos flotantes**

Para mejorar el formato %f, que mostrará el valor con 6 decimales aunque no sea real, por ejemplo en el caso de sueldo no queda bien mostrar

$1245,500000 podemos recurrir a estas mejoras.

% 6.1f son 6 caracteres en total de los cuales 1 es decimal

%.2f si delante del ´.´ no hay dígito significa que la cantidad de caracteres totales no está restringida, pero el dígito que está detrás del ´.´ es la cantidad de decimales que se mostrarán.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Estructuras de decisión o selección:

Sentencia IF:

Permite tomar una decisión para ejecutar una acción u otra, basándose en el resultado V o F de una condición.

Sintaxis:

```
if (condición)
sentencia 1;
[ else
        sentencia 2;]  opcional
```

La salida por el VERDADERO siempre debe existir.

Es por eso que el **else** es opcional, es posible que ante una decisión por la opción V se tengan que ejecutar sentencias, pero por el F no se ejecute ninguna sentencia.

En caso de no ser así, se debe colocar la condición inversa.

Si tanto por la salida de V o F se deben ejecutar más de 1 sentencia, las salidas se deben colocar entre { } para limitar el bloque.

Los operadores de relación que van en la condición son <, >, <=, >=,

== (igualdad) ! = (distinto).

Cualquier regla convencional de sangrías que escoja deberá aplicarse con cuidado en todo el programa.

Un programa que no siga reglas de esparcimiento uniformes es difícil de leer.

OPERADOR ?:

C tiene este operador que utiliza 3 operandos.

Los operandos junto con el operador condicional conforman una expresión condicional.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

El primer operando es una condición, el segundo operando es el valor de toda la expresión condicional si la condición es verdadera, y el tercer operando es el valor de toda la expresión condicional si la condición es falsa.

Sintaxis:

```
Condición? expresión 1: expresión 2;
```

Si la condición es verdadera se ejecuta la expresión 1, si es falsa se ejecuta la expresión 2.

Ejemplo:

Leer dos números y mostrar el mayor

```
# include   <stdio.h>
int main ( )
{
int x,a,b;
printf ("Ingrese un número: ");
scanf ("%d", &a);
printf ("Ingrese otro ; ");
scanf ("%d", &b);
x= a>b? a:b;
printf ("El mayor es %d", x);
return 0;
```

Sin utilizar la variable x sería:

printf ("%d", a>b? a:b);

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

ANIDAMIENTO DE IF:

Sintaxis:

```
if  (condición 1)
{
        if  (condición 2)
                sentencia 1;
}
```

Al colocar las { } estamos indicando que **el else pertenece al primer if**. Esas llaves son obligatorias ya que estamos alterando el orden de pertenencia de los else.

Cond. 1	Cond. 2	Sent 1	Sent 2
F	F	NO	SI
F	V	NO	SI
V	F	NO	NO
V	V	SI	NO

y si quitamos las claves:

```
if  (condición 1)
        if  (condición 2)
                sentencia 1;
else
                sentencia 2;
```

El else correspondería al segundo if.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Cada ELSE se corresponde con el IF más próximo que no haya sido emparejado.

Cond. 1	Cond. 2	Sent 1	Sent 2
F	F	NO	NO
F	V	NO	NO
V	F	NO	SI
V	V	SI	NO

Operadores Lógicos:

AND Y &&

OR O ||

Para recordar

> En un **&& (y),** la salida es verdadera sólo si ambas condiciones son VERDADERAS.
>
> En un **|| (o),** la salida es falsa sólo si ambas condiciones son FALSAS.

Ejemplo:

Realizar un programa para sumar dos números reales tales que, si la suma es mayor que 10, se divide por 2 y si la suma es menor que 10 se multiplica por 3, si es 10 no se modifica. Listar en todos los casos el resultado logrado.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

```
#include <stdio.h>
int main( )
{

float sum, num1, num2, res=0;
printf ("Ingresa el primer número ");
scanf ("%f",&num1);
printf ("Ingresa el segundo número ");
scanf ("%f",&num2);
sum=num1+num2;
if (sum>10)
        res=sum/2;
else
        if(sum<10)
                res=sum*3;
        else
                res=sum;
printf ("Resultado %.2f",res);
return 0;
}
```

Estructuras de repetición

Una estructura de repetición le permite al programador que se repita una acción, en tanto cierta condición se mantenga verdadera.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Sentencia WHILE

Sintaxis:

```
while (condición)
        sentencia 1;

Si se repiten más de una sentencia

 while (condición)
 {
        sentencia 1;
       sentencia 2;
 }
   /* próxima sentencia ejecutable */
```

Si la condición es verdadera entra al while o sea se ejecutan las sentencias del while.

Si la condición es falsa no entra y ejecuta la próxima instrucción ejecutable fuera del while.

IMPORTANTE!!

Cuando la variable es un contador que se incrementa en +1 se puede escribir como:

variable++;

Cuando la variable es un contador que se incrementa en -1 se puede escribir como:

variable--;

Sentencia DO/WHILE

Es similar al while. En el while la condición se encuentra al principio, antes de ejecutarse el cuerpo del mismo.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

La estructura DO/WHILE prueba la condición de continuación del ciclo después de ejecutarse el cuerpo del ciclo, y por lo tanto el cuerpo se ejecutará por lo menos una vez.

Cuando la condición es falsa se ejecuta la acción siguiente a la del while.

Sintaxis:

```
do
{
        sentencia 1;
        sentencia 2;
} while (condición);
```

¡IMPORTANTE!

- **En el caso de repeticiones controladas por contadores se requiere:**

  El nombre de una variable de control (controlador del ciclo).

  El valor inicial de dicha variable.

  El incremento (o decremento) con el cual, cada vez que se termine un ciclo, la variable de control se modifica.

  La condición que compruebe la existencia del valor final de la variable de control.

- **En caso de no ser variables de tipo controlador** siempre debe haber una primer lectura antes de la condición en el while (sino la condición se compararía contra "basura") y las siguientes lecturas antes de la llave de fin del while.

```
primer lectura;
while (condición)
{
   sentencias;
   siguiente lectura;  }
```

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

Sentencia FOR

La utilizo sólo cuando sé exactamente la cantidad de repeticiones que se debe hacer:

Sintaxis:

```
for (variable1 = expresión1; condición ; progresión de la condición)
            sentencia1;

for (variable1 = expresión1; condición ; progresión de la condición)
{
        sentencia1;
       sentencia2;
 }
```

En variable1= expresión1 **inicializa** la variable1 con el valor de expresion1.

La **condición** es la prueba de control de fin (nunca va un igual), mientras sea VERDADERA se repite el ciclo.

La **progresión de la condición** es el incremento de avance de la variable1.

El ciclo termina si la condición es falsa.

En caso de ser más de una sentencia lo que se repite se encierra entre { }.

Ejemplos:

Imprimir los múltiplos de 7 que hay entre 7 y 112.

```
# include <stdio.h>
int main ( )
{
 int x;
 for (x = 7; x < = 112; x + = 7)
          printf ("%d  ", x);
 }
```

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.

El valor inicial es 7, la condición es que sea <=112 y el incremento o progresión es de 7, por pedir múltiplos de 7.

Imprimir los números del 9 al 1.

```
# include <stdio.h>
int main ( )
{
 int k;
 for (k = 9; k > = 1 ; k - - )
          printf ("%d  ", k);
 }
```

En este caso el valor inicial es 9, pero el final es 1, valor menor al inicial.

Es por eso que la condición debe ser con >=1 y el incremento es de -1 ya que va disminuyendo la variable.

© Universidad de Palermo. Prohibida la reproducción total o parcial de imágenes y textos.