INSTITUTO TECNOLÓGICO BELTRÁN

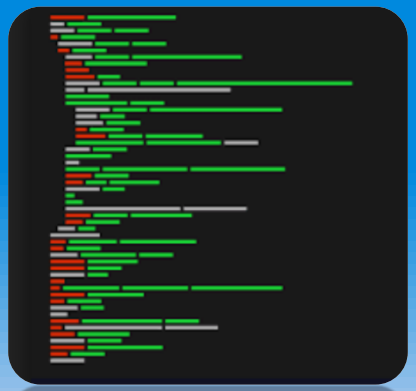
Centro de Tecnología e Innovación

LENGUAJE C

El lenguaje de programación C está caracterizado por ser de uso general, con una sintaxis sumamente compacta y de alta portabilidad.

Es común leer que se lo caracteriza como un lenguaje de bajo nivel. No debe confundirse el término bajo con poco, ya que el significado del mismo es en realidad profundo, en el sentido que C maneja los elementos básicos presentes en todas las computadoras: caracteres, números y direcciones.

La descripción del lenguaje se realiza siguiendo las normas del ANSI C, por lo tanto, todo lo expresado será utilizable con cualquier compilador que se adopte; sin embargo en algunos casos particulares se utilizaron funciones Compilador o Sistema Operativo-dependientes, explicitándose en estos casos la singularidad de las mismas.



ANATOMIA DE UN PROGRAMA C

Siguiendo la metodología de trabajo, la mejor forma de aprender a programar en cualquier lenguaje es editar, compilar, corregir y ejecutar pequeños programas descriptivos.
Analicemos por lo tanto el primer ejemplo

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - MAIN

FUNCION main()

La función main() indica donde empieza el programa, cuyo cuerpo principal es un conjunto de sentencias delimitadas por dos llaves, una inmediatamente después de la declaración main() " { ", y otra que finaliza el listado "} ".

Todos los programas C arrancan del mismo punto: la primer sentencia dentro de dicha función, en este caso printf (".....").

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - PRINTF

En el ejemplo, el programa principal está compuesto por sólo dos sentencias: la primera es un llamado a una función denominada *printf()*, y la segunda, *return*, que finaliza el programa retornando al Sistema Operativo.
El lenguaje C no tiene operadores de entrada-salida por lo que para escribir en pantalla es necesario llamar a una función externa. En este caso se invoca a la función *printf(argumento)* existente en la Librería y a la cual se le envía como argumento aquellos caracteres que se desean escribir en la pantalla. Los mismos deben estar delimitados por comillas. La secuencia \n que aparece al final del mensaje es la notación que emplea C para indicar una nueva linea y que hace avanzar al cursor a la posición extrema izquierda de la línea siguiente.

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - RETURN

La segunda sentencia *return 0* termina el programa y devuelve un valor al Sistema operativo, por lo general cero si la ejecución fue correcta y valores distintos de cero para indicar diversos errores que pudieron ocurrir.

Si bien no es obligatorio terminar el programa con un *return*, es conveniente indicarle a quien lo haya invocado, sea el Sistema Operativo o algún otro programa, si la finalización ha sido exitosa, o no. De cualquier manera en este caso, si sacamos esa sentencia el programa correrá exactamente igual, pero al ser compilado, el compilador nos advertirá de la falta de retorno.

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - TERMINADOR

Cada sentencia de programa queda finalizada por el terminador *"; "*, el que indica al compilador el fin de la misma.

Esto es necesario ya que, sentencias complejas pueden llegar a tener más de un renglón, y habrá que avisarle al compilador donde terminan.

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - ENCABEZAMIENTO

Las líneas anteriores a la función *main()* se denominan encabezamiento (header) y son informaciones que se le suministran al Compilador.
La primera línea del programa está compuesta por una directiva: *"#include"* que implica la orden de leer un archivo de texto especificado en el nombre que sigue a la misma *(<stdio.h>)* y reemplazar esta línea por el contenido de dicho archivo.

En este archivo están incluidas declaraciones de las funciones luego llamadas por el programa (por ejemplo *printf()*) necesarias para que el compilador las procese. Por ahora no nos preocupemos por el contenido del archivo ya que más adelante, en el capítulo de funciones, analizaremos exhaustivamente dichas declaraciones.

Hay dos formas distintas de invocar al archivo: si el archivo invocado está delimitado por comillas (por ejemplo "stdio.h") el compilador lo buscará en el directorio activo en el momento de compilar y si, en cambio, se lo delimita con los signos <.....> lo buscará en algún otro directorio.

Por lo general estos archivos son guardados en un directorio llamado include y el nombre de los mismos está terminado con la extensión .h.

ANATOMIA DE UN PROGRAMA C - ENCABEZAMIENTO

La razón de la existencia de estos archivos es la de evitar la repetición de la escritura de largas definiciones en cada programa.

Nótese que la directiva *"#include"* no es una sentencia de programa sino una orden de que se copie literalmente un archivo de texto en el lugar en que ella está ubicada, por lo que no es necesario terminarla con *";"*.

```
#include <stdio.h>

main()
{
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0;
}
```

ANATOMIA DE UN PROGRAMA C - COMENTARIOS

La inclusión de comentarios en un programa es una saludable práctica, como lo reconocerá cualquiera que haya tratado de leer un listado hecho por otro programador o por sí mismo, varios meses atrás. Para el compilador, los comentarios son inexistentes, por lo que no generan líneas de código, permitiendo abundar en ellos tanto como se desee.

En el lenguaje C se toma como comentario todo carácter interno a los símbolos: /* */. Los comentarios pueden ocupar uno o más renglones. También podemos tener comentarios sobre una misma líneas mediante la inclusión de doble barras //.

```
#include <stdio.h>

main()
{
        /* Comentario
           de más de una línea */
        printf ("Bienvenido a la Programación en lenguaje C \n");
        return 0; //Comentario de una sola línea
}
```

DEFINICIÓN DE VARIABLES

Si deseara imprimir los resultados de multiplicar un número fijo por otro que adopta valores entre 0 y 9, la forma normal de programar esto sería crear una constante para el primer número y un par de variables para el segundo y para el resultado del producto.
Un programa debe definir a todas las variables que utilizará, antes de comenzar a usarlas, a fin de indicarle al compilador de que tipo serán, y por lo tanto cuanta memoria debe destinar para albergar a cada una de ellas.

```
#include <stdio.h>

main()
{
        int multiplicador; //Defino multiplicador como un entero
        int multiplicando; // Defino multiplicando como un entero
        int resultado;  //Defino resultado como un entero
        multiplicador = 1000; //Les asigno valores
        multiplicando = 2;
        resultado = multiplicando * multiplicador;
        printf("Resultado = %d\n", resultado);  //Muestro el resultado
        return 0;
}
```

DEFINICIÓN DE VARIABLES

En las primeras líneas de texto dentro de main() definimos las variables como números enteros, es decir del tipo *"int"* seguido de un identificador (nombre) de la misma. Este identificador puede tener la cantidad de caracteres que se desee, aunque es conveniente darle a los identificadores de las variables, nombres que tengan un significado que luego permita una fácil lectura del programa.

Los identificadores deben comenzar con una letra o con el símbolo de subrayado *"_"*, pudiendo continuar con cualquier otro carácter alfanumérico o el símbolo *"_"*.

El único símbolo no alfanumérico aceptado en un nombre es el *"_"*. El lenguaje C es sensible al tipo de letra usado; así tomará como variables distintas a una llamada *"variable"*, de otra escrita como *"VARIABLE"*.

Es una convención entre los programadores de C escribir los nombres de las variables y las funciones con minúsculas, reservando las mayúsculas para las constantes.

DEFINICIÓN DE VARIABLES

Vemos en las dos líneas subsiguientes a la definición de las variables, que puedo ya asignarles valores (1000 y 2) y luego efectuar el cálculo de la variable *"resultado"*. Si prestamos ahora atención a la función *printf()*, ésta nos mostrará la forma de visualizar el valor de una variable.

```
#include <stdio.h>

main()
{
	int multiplicador; //Defino multiplicador como un entero
	int multiplicando; // Defino multiplicando como un entero
	int resultado;  //Defino resultado como un entero
	multiplicador = 1000; //Les asigno valores
	multiplicando = 2;
	resultado = multiplicando * multiplicador;
	printf("Resultado = %d\n", resultado);  //Muestro el resultado
	return 0;
}
```

DEFINICIÓN DE VARIABLES

Insertada en el texto a mostrar, aparece una secuencia de control de impresión *"%d"* que indica, que en el lugar que ella ocupa, deberá ponerse el contenido de la variable (que aparece luego de cerradas las comillas que marcan la finalización del texto , y separada del mismo por una coma) expresado como un número entero decimal. Así, si compilamos y corremos el programa, obtendremos la salida con el resultado directamente concatenado.

```
#include <stdio.h>

main()
{
	int multiplicador; //Defino multiplicador como un entero
	int multiplicando; // Defino multiplicando como un entero
	int resultado;  //Defino resultado como un entero
	multiplicador = 1000; //Les asigno valores
	multiplicando = 2;
	resultado = multiplicando * multiplicador;
	printf("Resultado = %d\n", resultado);  //Muestro el resultado
	return 0;
}
```

INICIALIZACIÓN DE VARIABLES

Las variables del mismo tipo pueden definirse mediante una definición múltiple separandolas mediante *","*:

int multiplicador, multiplicando, resultado;

Esta sentencia es equivalente a las tres definiciones separadas en el ejemplo anterior. Las variables pueden también ser inicializadas en el momento de definirse.

int multiplicador = 1000, multiplicando = 2, resultado;

```
#include <stdio.h>

main()
{
	int multiplicador=1000, multiplicando=2;
	printf("Resultado = %d\n", multiplicando * multiplicador);
	return 0;
}
```

INICIALIZACIÓN DE VARIABLES

Obsérvese que en la primer sentencia se definen e inicializan simultáneamente ambas variables. La variable *"resultado"* la hemos hecho desaparecer ya que es innecesaria. Si analizamos la función *printf()* vemos que se ha reemplazado *"resultado"* por la operación entre las otras dos variables. Esta es una de las particularidades del lenguaje C: en los parámetros pasados a las funciones pueden ponerse operaciones (incluso llamadas a otras funciones), las que se realizan antes de ejecutarse la función, pasando finalmente a esta el valor resultante de las mismas.

El siguiente ejemplo funciona exactamente igual que antes pero su código ahora es mucho más compacto y claro.

```
#include <stdio.h>

main()
{
	int multiplicador=1000, multiplicando=2;
	printf("Resultado = %d\n", multiplicando * multiplicador);
	return 0;
}
```

TIPOS DE VARIABLES - ENTEROS

En el ejemplo anterior definimos a las variables como enteros (int).

De acuerdo a la cantidad de bytes que reserve el compilador para este tipo de variable, queda determinado el alcance o máximo valor que puede adoptar la misma.

Debido a que el tipo int ocupa dos bytes su alcance queda restringido al rango entre -32.768 y +32.767 (incluyendo 0).

En caso de necesitar un rango más amplio, puede definirse la variable como *long int nombre_de_variable* o en forma más abreviada *long nombre_de_variable*
Declarada de esta manera, *nombre_de_variable* puede alcanzar valores entre -2.347.483.648 y +2.347.483.647.

A la inversa, si se quisiera un alcance menor al de int, podría definirse *short int* o simplemente *short*, aunque por lo general, los compiladores modernos asignan a este tipo el mismo alcance que int.

TIPOS DE VARIABLES - ENTEROS

Debido a que la norma ANSI C no establece taxativamente la cantidad de bytes que ocupa cada tipo de variable, sino tan sólo que un "long" no ocupe menos memoria que un "int" y este no ocupe menos que un "short", los alcances de los mismos pueden variar de compilador en compilador.

Para variables de muy pequeño valor puede usarse el tipo "char" cuyo alcance está restringido a -128, +127 y por lo general ocupa un único byte.

Todos los tipos citados hasta ahora pueden alojar valores positivos o negativos y, aunque es redundante, esto puede explicitarse agregando el calificador "signed" delante; por ejemplo:

signed int
signed long
signed long int
signed short
signed short int
signed char

TIPOS DE VARIABLES - ENTEROS

Si en cambio, tenemos una variable que sólo puede adoptar valores positivos (como por ejemplo la edad de una persona) podemos aumentar el alcance de cualquiera de los tipos, restringiéndolos a que sólo representen valores sin signo por medio del calificador *unsigned*. A continuación, se resume los alcances de distintos tipos de variables enteras:

TIPO	BYTES	VALOR MÍNIMO	VALOR MÁXIMO
Signed char	1	-128	127
Unsigned char	1	0	255
Signed short	2	-32,768	+32767
Unsigned short	2	0	+65,535
Signed int	2	-32,768	+32,767
Unsigned int	2	0	+65,535
Signed long	4	-2,147,483,648	+2,147,483,647
Unsigned long	4	0	+4,294,967,295

Si se omite el calificador delante del tipo de la variable entera, éste se adopta por omisión (default) como *signed*.

TIPOS DE VARIABLES – REAL O PUNTO FLOTANTE

Un número real o de punto flotante es aquel que además de una parte entera, posee fracciones de la unidad. En nuestra convención numérica solemos escribirlos de la siguiente manera: 2,3456. Lamentablemente, los compiladores usan la convención del PUNTO decimal (en vez de la coma). Así el numero Pi se escribirá: 3.14159.
Otro formato de escritura, normalmente aceptado, es la notación científica. Por ejemplo podrá escribirse 2.345E+02, equivalente a 2.345 * 100 o 234.5.

De acuerdo a su alcance hay tres tipos de variables de punto flotante:

TIPO	BYTES	VALOR MÍNIMO	VALOR MÁXIMO
Float	4	3.4E-38	3.4E+38
Double	8	1.7E-308	1.7E+308
Long double	10	3.4E-4932	3.4E+4932

Las variables de punto flotante son SIEMPRE con signo, y en el caso que el exponente sea positivo puede obviarse el signo del mismo.

CONVERSION AUTOMATICA DE TIPOS

Cuando dos o más tipos de variables distintas se encuentran dentro de una misma operación o expresión matemática, ocurre una conversión automática del tipo de las variables.

En todo momento de realizarse una operación, se aplica la siguiente secuencia de reglas de conversión (previamente a la realización de dicha operación):

1. Las variables del tipo *char* o *short* se convierten en *int*

2. Las variables del tipo *float* se convierten en *double*

3. Si alguno de los operandos es de mayor precisión que los demás, estos se convierten al tipo de aquel y el resultado es del mismo tipo

4. Si no se aplica la regla anterior y un operando es del tipo *unsigned* el otro se convierte en *unsigned* y el resultado es de este tipo.

CONVERSION AUTOMATICA DE TIPOS

Las reglas 1 a 3 no presentan problemas, sólo nos dicen que previamente a realizar alguna operación las variables son promovidas a su instancia superior. Esto no implica que se haya cambiado la cantidad de memoria que las aloja en forma permanente.

Otro tipo de regla se aplica para la conversión en las asignaciones.
Si definimos los términos de una asignación como, *lIzquierdo* a la variable a la izquierda del signo igual y *lDerecho* a la expresión a la derecha del mismo, es decir:

lIzquierdo = lDerecho;

Posteriormente al cálculo del resultado de *lDerecho* (de acuerdo con las reglas antes descriptas), el tipo de este se iguala al del *lIzquierdo.*

El resultado no se verá afectado si el tipo de *lIzquierdo* es igual o superior al del *lDerecho*, en caso contrario se efectuará un truncamiento o redondeo, según sea el caso.

Por ejemplo, el pasaje de float a int provoca el truncamiento de la parte fraccionaria, en cambio de double a float se hace por redondeo.

TIPOS DE VARIABLES – CARACTER

El lenguaje C guarda los caracteres como números de 8 bits de acuerdo a la norma ASCII extendida, que asigna a cada caracter un número comprendido entre 0 y 255 (un byte de 8 bits). El tipo de dato a utilizar es char.

Lamentablemente existen una serie de caracteres que no son imprimibles, en otras palabras que cuando editemos nuestro programa fuente (archivo de texto) nos resultará difícil de asignarlas a una variable ya que el editor las toma como un comando y no como un caracter.

Un caso típico sería el de nueva línea (ENTER).

Con el fin de tener acceso a los mismos, es que aparecen ciertas secuencias de escape convencionales. Las mismas están listadas en la siguiente tabla y su uso es idéntico al de los caracteres normales, asi para resolver el caso de una asignación de nueva línea se escribirá:

```
char c = '\n';
```

TIPOS DE VARIABLES – SECUENCIAS DE ESCAPE

TIPO	SIGNIFICADO
\n	Nueva línea
\r	Retorno de carro
\f	Nueva página
\t	Tabulador horizontal
\b	Retroceso (backspace)
\’	Comilla simple
\”	Comilla doble
\\	Barra
\?	Interrogración
\nnn	Cualquier caracter (donde nnn es el código ASCII expresado en octal)
\xnn	Cualquier caracter (donde nn es el código ASCII expresado en hexadecimal)

TIPOS DE VARIABLES – CONSTANTES

Son aquellos valores que, una vez compilado el programa, no pueden ser cambiados, como por ejemplo los valores literales que hemos usado hasta ahora en las inicializaciones de las variables (1000 , 2 , 'a' , '\n' , etc).

Como dichas constantes son guardadas en memoria de la manera que al compilador le resulta más eficiente suelen aparecer ciertos efectos secundarios, a veces desconcertantes, ya que las mismas son afectadas por las reglas de reconversión automática de tipo vista previamente.

TIPOS DE VARIABLES – CONSTANTES SIMBÓLICAS

Por lo general es una mala práctica de programación colocar en un programa constantes en forma literal (sobre todo si se usan varias veces en el mismo) ya que el texto se hace difícil de comprender y aún más de corregir, si se debe cambiar el valor de dichas constantes.

Se puede en cambio asignar un símbolo a cada constante, y reemplazarla a lo largo del programa por el mismo, de forma que este sea más legible y además, en caso de querer modificar el valor, bastará con cambiarlo en la asignación.

El compilador, en el momento de crear el ejecutable, reemplazará el símbolo por el valor asignado.

Para dar un símbolo a una constante bastará, en cualquier lugar del programa (previo a su uso) poner la directiva:

```
#define VALOR_CONSTANTE 342

#define PI 3.1416
```

INSTITUTO TECNOLÓGICO BELTRÁN

Centro de Tecnología e Innovación