

# PROGRAMACIÓN BÁSICA

PROF. ALMA GONZÁLEZ



# HIGHLIGHTS DE LA HISTORIA DE LA PROGRAMACIÓN

- 1843 - La matemática Ada Lovelace trabaja con Charles Babbage, quien desarrolla los planos de la primera computadora mecánica. Ada escribe un algoritmo, o programa, que puede ejecutarse en dicha maquina. Se le considera la primera programadora.
- 1938 - Konrad Zuse primera computadora completamente programable. Código binario "ceros y unos"
- 1946 - John Mauchly y J. Presper Eckert completan la primera computadora electronica, ENIAC (Electronic Numerical Integrator Analyzer and Computer). Usada en calculos de balística, ENIAC pesaba ~30 Toneladas y ocupaba un cuarto de 500m<sup>2</sup>. Seis mujeres fueron las programadoras principales de ENIAC.
- 1972 - Se desarrolla el lenguaje C por Dennis Ritchie - AT&T Bell Laboratories.
- 1973 - La cuarta edición del sistema UNIX fue reescrito en lenguaje C. Lo que permitió que fuera portable a diferentes computadoras.
- 1983 - Se desarrolla el lenguaje C++ por Bjarne Stroustrup.
- 1991 - Se desarrolla el lenguaje de programación Python por Guido Van Rossum
- 1991 - Linus Torvalds desarrolla y hace Publico el kernel Linux, que evolucionó en las diferentes distribuciones de Linux. Sistema Operativo de uso Libre.
- <http://blog.connectionsacademy.com/discover-the-history-of-coding-for-computer-science-week/>

# C

- Las ideas del lenguaje provienen de un lenguaje llamado BCPL desarrollado por Martin Richards. Primero se desarrolla B por Ken Thompson en 1970 en los laboratorios Bell, y posteriormente nace C. Dennis Ritchie - AT&T Bell Laboratories - 1972
- 1978 Publicación de "The C Programming Language" de Kernighan y Ritchie revoluciona el computo. Especificación del lenguaje
- Ampliamente usado actualmente
  - Diferentes sistemas operativos y arquitecturas.
  - Micro-controladores: Autos, Aviones, etc.
  - Procesadores integrados: Telefonos, tablets, electronicos en gral.
  - Procesadores Digitales: Sistemas de Audio, TV, etc.
  - Eficiencia/Rendimiento
- Es un lenguaje compilado, i.e. Las instrucciones se escriben en un archivo de texto que es interpretado para generar un ejecutable.
- Programación nivel bajo. Los comandos o funciones en el lenguaje están íntimamente ligadas a las instrucciones del procesador.

## Proyecto 1:

En equipos de (máximo) 5 personas investigar más sobre la historia de la programación y el lenguaje C. Resumirlo en una infografía, mapa, ú otra forma gráfica. Subir en GitHub solo un archivo .pdf o .jpg. Solo un integrante sube el archivo, incluye los nombres de los miembros del equipo.

Fecha limite: 26 de Agosto.

# ESTRUCTURA DE UN PROGRAMA EN C

```
/*Inicia con comentarios.*/
```

//Esta es otra forma de poner un comentario, generalmente corto

Todo programa tiene una función maestra

```
#include <stdio.h> //Incluye librerías necesarias.
```

```
int main( ){
```

Aquí se especificaran todas las instrucciones

Cuerpo del programa

```
}
```

Puede haber otras funciones, no maestras definidas.

```
int funcion1(){
```

```
}
```

# COMENTARIOS

- `/* Este es un comentario */`
- `/* ESTOS COMENTARIOS  
PUEDEN OCUPAR VARIAS  
LINEAS,  
TANTAS COMO SEA NECESARIO.  
*/`
- `//Este es un comentario corto.`
- Los comentarios son ignorados por el compilador.
- Pueden aparecer en cualquier parte del código.



# INSTRUCCIÓN/ENCABEZADO

## #INCLUDE

- Indica que en el programa se incluyan las definiciones de constantes, funciones y otras declaraciones que se encuentran en los archivos especificados.
- `#include <libreria.h>` lee e incluye el contenido de la librería `stdio.h`, que se encuentra en el archivo `.h`
- `stdio.h` Definiciones interacción con la terminal. Es parte de las librerías estándar de C. Otras librerías que veremos/usaremos `ctype.h`, `math.h`, `stdlib.h`, `string.h`, `time.h`.
- Se añaden solo aquellas librerías que se van a usar.
- Podemos usar librerías propias (más adelante en el curso) :
  - `#include "libreria.h"`
  - El compilador buscará primero en el directorio donde se este intentando compilar el programa. Si está en otro directorio se podrá especificar.

# FUNCIÓN MAIN

- Detalles en otra sesión. Por el momento solo diremos que contienen toda la secuencia de instrucciones a realizar por nuestro programa.

```
INT MAIN( ){  
    CUERPO DEL PROGRAMA;  
    EXIT(0);  
}
```

- Las llaves { } nos indican el inicio y final de la función.
- Exit(0); indica que la secuencia de instrucciones sucedió correctamente, si no es así entonces enviara un error. Requiere la librería stdlib.h. Alternativamente se puede poner la instrucción return(0).
- El cuerpo del programa incluye la declaración de variables y las acciones a realizar con dichas variables. Así como la interacción con el usuario.
- Todas las instrucciones terminan con ; (Es muy importante y una de las fuentes principales de errores!)



# CUERPO DEL PROGRAMA

- Declaración de Variables
- Instrucciones.

# VARIABLES

- En C todas las variables a usar deben ser declaradas ANTES de usarse. La declaración de variables incluye el nombre y tipo de variable.
- Tipos de variable:
  - `char` : character
  - `int` : Variable del tipo entero.
    - `short`
    - `signed`
    - `unsigned` ..... En términos prácticos nosotros casi siempre usaremos `int`.
  - `float` : Variable del tipo punto flotante. Maneja típicamente 6 cif
  - `double` : Variable del tipo punto flotante de doble precisión.

# INICIALIZAR VARIABLES

- Se especifica el nombre de la variable y su tipo. Ej.
  - `int edad;`
  - `float temperatura;`
  - `char a;`
  - `char nombre[20];` //char es un solo carácter, para formar palabras necesitamos un "arreglo" de caracteres. El número indica el de máximo de caracteres.
- Podemos inicializar el valor desde la declaración ( conveniente en algunos casos). Ej:
  - `int n=3;`
  - `float phi=1.6180339;`
- Podemos declarar varias variables del mismo tipo a la vez, Ej.
  - `int a,b,c=0,d=4;`
  - `float T1=12.5, T2;`
- Aquellas variables a las que no se le asigne un valor, tendrán uno predeterminado por el compilador, que a priori no sabemos cual es. Por ello debemos inicializar todas las variables. Ej.
  - `n=3;`
  - `T=20;`

# INSTRUCCIONES (O SENTENCIAS)

- Se refiere a la secuencia de instrucciones que realizara nuestro programa.
- Es una secuencia por lo que el orden importa.
  - Instrucciones de interacción con la terminal. Ej. imprimir a la pantalla ó captar información proporcionada por el usuario :
    - `printf ("El texto a mostrar en pantalla");`
    - `scanf ("%i", &edad);` // %i indica el tipo de variable (entero en este caso). &edad indica la variable a la que se asignara el valor proporcionado (previamente definida como entero en la declaración de variables). El símbolo & lo veremos más adelante.
  - Instrucciones de operaciones entre variables. Ej. suma, resta, división o multiplicación, (u combinaciones de ellas)
  - Uso de funciones definidas en otras librerías.

# OPERACIONES ARITMÉTICAS

- Supongamos que  $x$ ,  $y$ , son dos variables. Podemos hacer las siguientes operaciones con ellas:
  - $x+y$ ,  $x-y$ ,  $x*y$ ,  $x/y$ ,  $x\%y$  (% la función modulo calcula el residuo de la división, ej.  $\text{num}\%2$  será igual a cero si el numero es par, y diferente de cero si es impar)
- Podemos asignar el resultado de una operación entre variables a otra variable (previamente declarada). Incluso podemos sustituir una misma variable repetidas veces, estando conscientes que perdemos la información anterior. Ej.
  - $y = x+3*x/(y-4)$  (Sobre-escribimos el valor de  $y$  a su nuevo valor)
  - $z = x+3*x/(y-4)$  (asignamos el resultado de la operación a una variable diferente.)

Cuando sabemos que no volveremos a usar la información previa de la variable podemos sobre escribirla, esto es util para no definir muchas variables de forma innecesaria.

# ORDEN DE LAS OPERACIONES

Jerarquía  
↓

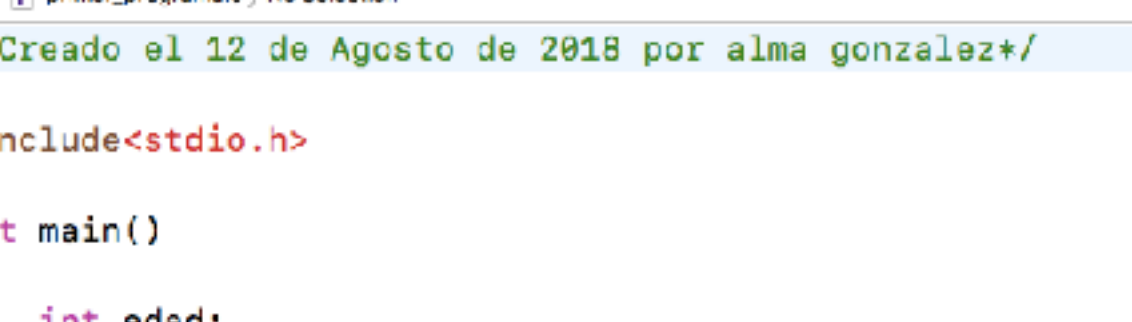
OPERADOR	DIRECCIÓN DE EVALUACIÓN
()	
*, /, %	IZQ-DER
+, -	IZQ-DER
=, +=, -=, *=, /=, %=	DER-IZQ

Usamos los paréntesis para especificar un orden de evaluación preferido



# ESCRIBIR CÓDIGO EN C.

- Archivos extension .c
- Se editan los archivos en editores como "vi", gedit, atom, xcode (mac), etc..



```
1  /*Creado el 12 de Agosto de 2018 por alma gonzalez*/
2
3  #include<stdio.h>
4
5  int main()
6  {
7      int edad;
8
9      printf("Hola Alma \n");
10     printf("Este es el 1er programa del curso");
11     printf("Introduce tu edad: \n");
12     scanf("%i",&edad);
13     printf("\nTu edad es: %i \n",edad);
14     return 0;
15 }
16
```

```

semana1 — vi primer_programa.c — 80x24
/*Creado el 25 de Enero de 2016 por alma gonzalez*/

#include<stdio.h>

int main()
{
    int edad;

    printf("Hola Alma \n");
    printf("Este es el 1er programa del curso");
    printf("Introduce tu edad: \n");
    scanf("%i",&edad);
    printf("\nTu edad es: %i \n",edad);
    return 0;
}
~
~
~
~
~
~
~
~
~
"primer_programa.c" 15L, 200C

```

# COMPILACIÓN (BÁSICA) DE PROGRAMAS C

- gcc (Disponible en casi cualquier distribución de linux y/o mac en versión ~4).
- <https://gcc.gnu.org>. Versión actual 8.

gcc -Wall programa.c -o programa.o

Opción para mostrar la mayoría de advertencias de posibles problemas que pueda haber en nuestro programa

Nombre del archivo del programa a compilar.

Opción para crear el ejecutable

nombre del programa ejecutable

- gcc -Wall -lm programa.c -o programa

Opción que se debe incluir si se usa la librería math

# COMPILACIÓN (BÁSICA) DE PROGRAMAS C

```
Almas-MacBook-Air-2:semana2 alxogm$ gcc primer_programa.c -Wall -o primer_programa.o
Almas-MacBook-Air-2:semana2 alxogm$ ls
operaciones_basicas.c primer_programa.c primer_programa.o print_limites.c
Almas-MacBook-Air-2:semana2 alxogm$
```

- Opciones mas complejas (a ver durante el curso) para:
  - Incluir multiples archivos fuente, separar el programa en multiples archivos.
  - Incluir directorios auxiliares.
  - Optimización y ligado de programas/librerias.

# EJERCICIO

ACTUALIZA TU REPOSITORIO A LOS  
CAMBIOS QUE HAYA REALIZADO YO.

En tu directorio del curso (pb2018-...) ejecuta las instrucciones.

```
git pull
```

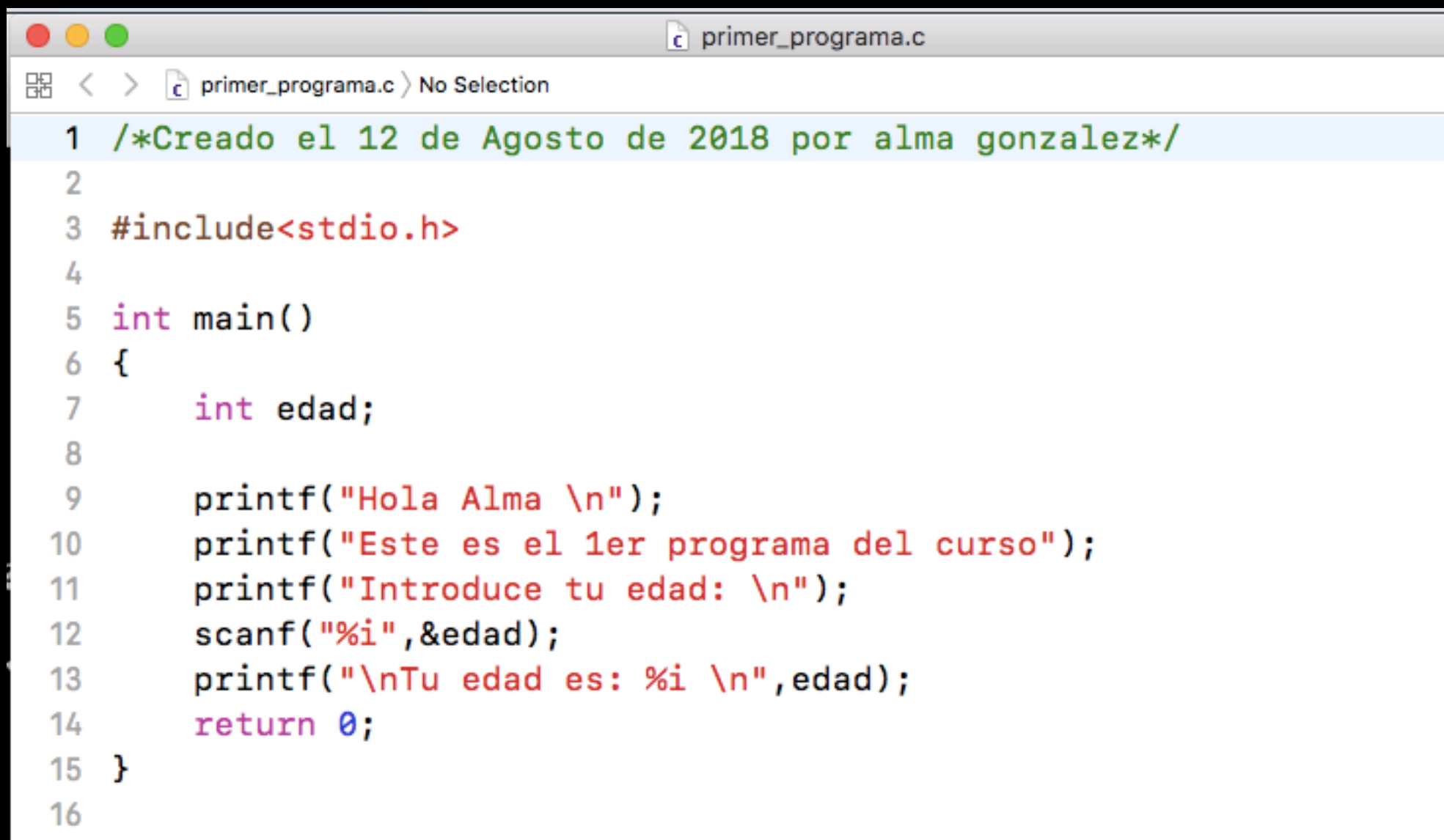
```
git pull https://github.com/alxogm/PB2018 master
```

```
git push
```

Listo ahora puedes empezar a trabajar

# EJERCICIO 1

- Escribe un programa como el de la imagen. Modifica el código para que en lugar de imprimírte tu edad actual, imprima la edad que tendrás en el año 2028. Para ello define una nueva variable llamada `edad_2`.
- Compila, ejecuta y verifica que el programa funciona adecuadamente.
- Extra: modifica el programa para que te solicite dar tu nombre.



```
1 /*Creado el 12 de Agosto de 2018 por alma gonzalez*/
2
3 #include<stdio.h>
4
5 int main()
6 {
7     int edad;
8
9     printf("Hola Alma \n");
10    printf("Este es el 1er programa del curso");
11    printf("Introduce tu edad: \n");
12    scanf("%i",&edad);
13    printf("\nTu edad es: %i \n",edad);
14    return 0;
15 }
16
```

## EJERCICIO 2

- Escribe un programa que te solicite introducir 4 números enteros. El programa debe imprimir a la pantalla las siguientes operaciones.

$$e = (a + b) * c / d$$

$$e = ((a + b) * c) / d$$

$$e = (a + b) * c / d$$

$$e = a + (b * c) / d$$

- Recuerda añadir comentarios a tu código.



## EJERCICIO 3

- Escribe un programa que te solicite introducir 4 números reales. El programa debe imprimir a la pantalla las siguientes operaciones.

$$e = (a + b) * c / d$$

$$e = ((a + b) * c) / d$$

$$e = (a + b) * c / d$$

$$e = a + (b * c) / d$$

- Recuerda añadir comentarios a tu código.

## EJERCICIO 4.

- Analiza el programa `limites.c` que ya se encuentra en la carpeta `semana2` del repositorio. Compíllalo, ejecútalo y analiza también la salida.
- Crea un documento llamado `Readme.md` dentro de `semana2` donde expliques el funcionamiento de los códigos de cada ejercicio y las diferencias que observas en los ejercicios 2 y 3.
- Actualiza el repositorio en GitHub (recuerda que estamos en la semana 2).