

Universidad Nacional de la Patagonia Austral
Unidad Académica Río Gallegos



Lenguaje C



Fundamentos de Lenguajes de Programación

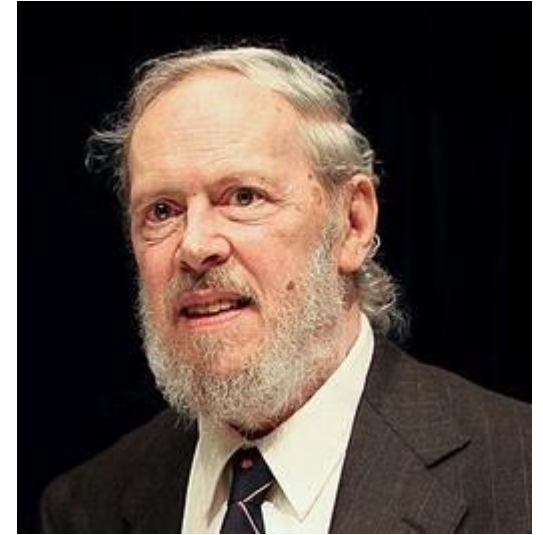
Bibliografía

- **C How to Program.With an Introduction to C++**
Autores: Paul Deitel, Harvey Deitel
Editorial: Pearson International, Año: 2016
ISBN: 129211097X
- **Como Programar En C ,C++ Y Java 4ta ed.**
Autores: Harvey Deitel, Paul Deitel
Editorial: Pearson Educación, Año: 2004
ISBN: 9702605318



Lenguaje C

- C es un lenguaje de programación creado en 1972 por Dennis M. **Ritchie** en los Laboratorios Bell como evolución del anterior lenguaje B.
- Se trata de un lenguaje débilmente tipificado de **nivel medio** ya que dispone de las estructuras típicas de los lenguajes de alto nivel así como de construcciones del lenguaje que permiten un control a muy bajo nivel.
- El lenguaje se estandarizó en 1990 y surgió **ANSI C** (también llamado C90)
- A fines de la década del '90 se logró la publicación del estándar ISO 9899:1999 conocido como C99 pero no tiene la misma aceptación que C90.



- Mainframes
- Se cataloga como un lenguaje de nivel medio
 - combina elementos de lenguajes de alto nivel (Fortran, Pascal, Basic, etc.) con la funcionalidad del lenguaje ensamblador.
- Pensado para escribir sistema operativo
- Permite el manejo de bits, bytes y direcciones de memoria.
- Posee sólo 32 palabras clave, definidas por el comité ANSI.

- Compilado
- Procedural
- Fuertemente tipado

32 Palabras Reservadas

char	void	default	return
int	if	break	auto
float	else	continue	extern
double	do	goto	register
long	while	struct	const
short	for	union	static
signed	switch	enum	volatile
unsigned	case	typedef	sizeof

ANSI C

- ANSI C está soportado hoy en día por casi la totalidad de los compiladores.
- La mayoría del código C que se escribe actualmente está basado en ANSI C.
- Cualquier programa escrito *sólo* en C estándar sin código que dependa de un hardware determinado **funciona correctamente en cualquier plataforma** que disponga de una implementación de C compatible.



Características de C

- Un núcleo del lenguaje simple que opera con bibliotecas (ej: las operaciones de E/S).
- Es un lenguaje muy flexible que soporta la programación estructurada (permitiendo ciertas licencias de ruptura).
- Un sistema de tipos que impide operaciones sin sentido.
- Usa un lenguaje de preprocesado con posibilidades para definir macros e incluir múltiples archivos de código fuente.



Características de C

- Acceso a memoria de bajo nivel mediante el uso de punteros.
- Interrupciones al procesador.
- Un conjunto reducido de palabras clave.
- Pasaje de parámetros por valor.
- Tipos de datos agregados (struct) equivalentes a los registros de Pascal.



Code::Blocks

- Para realizar las prácticas utilizaremos **Code::Blocks**.
- **Code::Blocks** es un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C++.
- Puede usarse libremente en diversos sistemas operativos.
- Está licenciado bajo la Licencia pública general de GNU.
- Dirección de descarga:

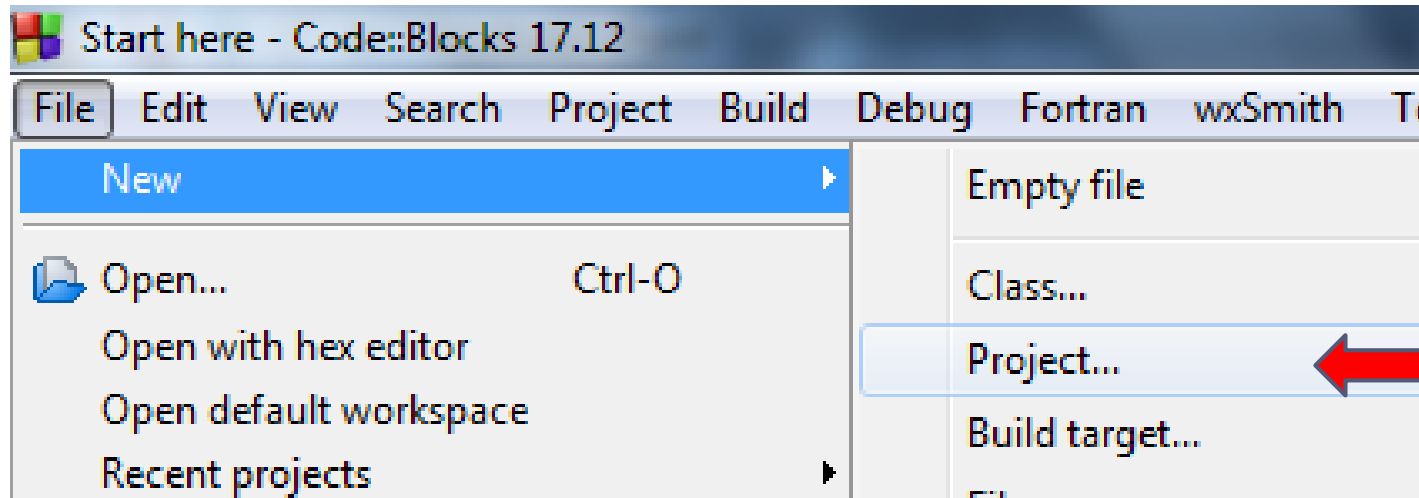
<http://www.codeblocks.org/downloads/binaries>

Elegir alguno que tenga el compilador GCC y el debugger GDB.
Por ejemplo para Windows descargar **codeblocks-20.03mingw-setup.exe**



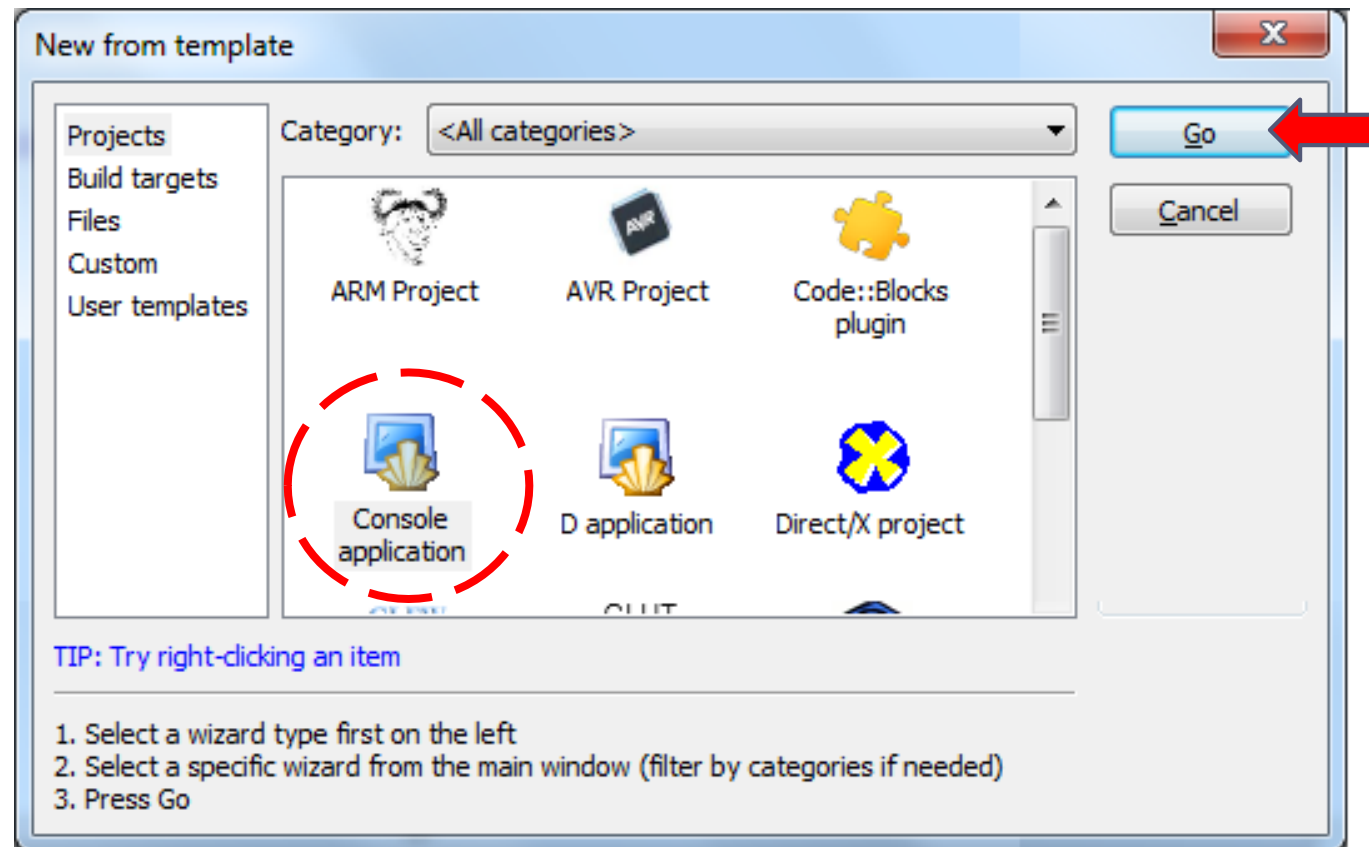
Cómo empezamos a programar?

- **Paso 1** :Comenzaremos creando un proyecto



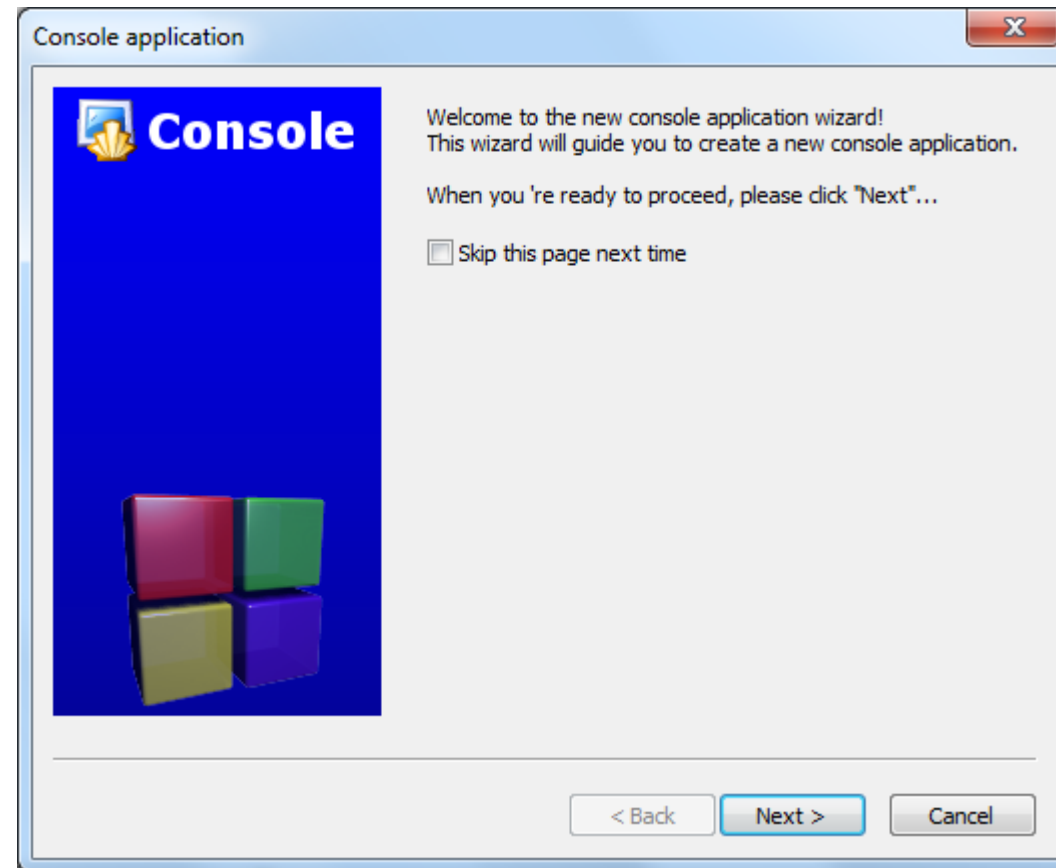
Cómo empezamos a programar?

- **Paso 2 :** Dentro del proyecto pondremos una aplicación de consola



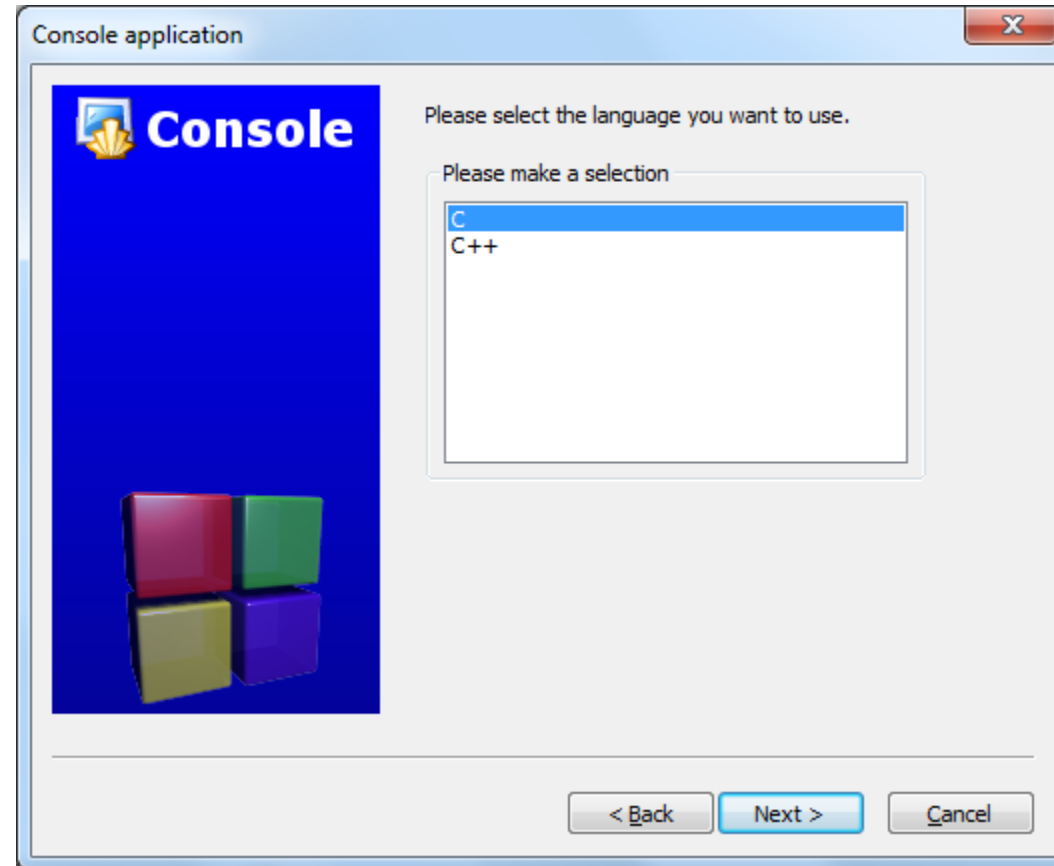
Creando una aplicación de consola

- **Paso 3 :**Seguir las indicaciones del Wizard ...



Creando una aplicación de consola

- **Paso 4 : Elegir el lenguaje C**



Creando una aplicación de consola

- **Paso 5 :** Indicar el título y el directorio del proyecto

Console application

Please select the folder where you want the new project to be created as well as its title.

Project title:
Ejemplo 1

Folder to create project in:
C:\TL1

Project filename:
Ejemplo 1.cbp

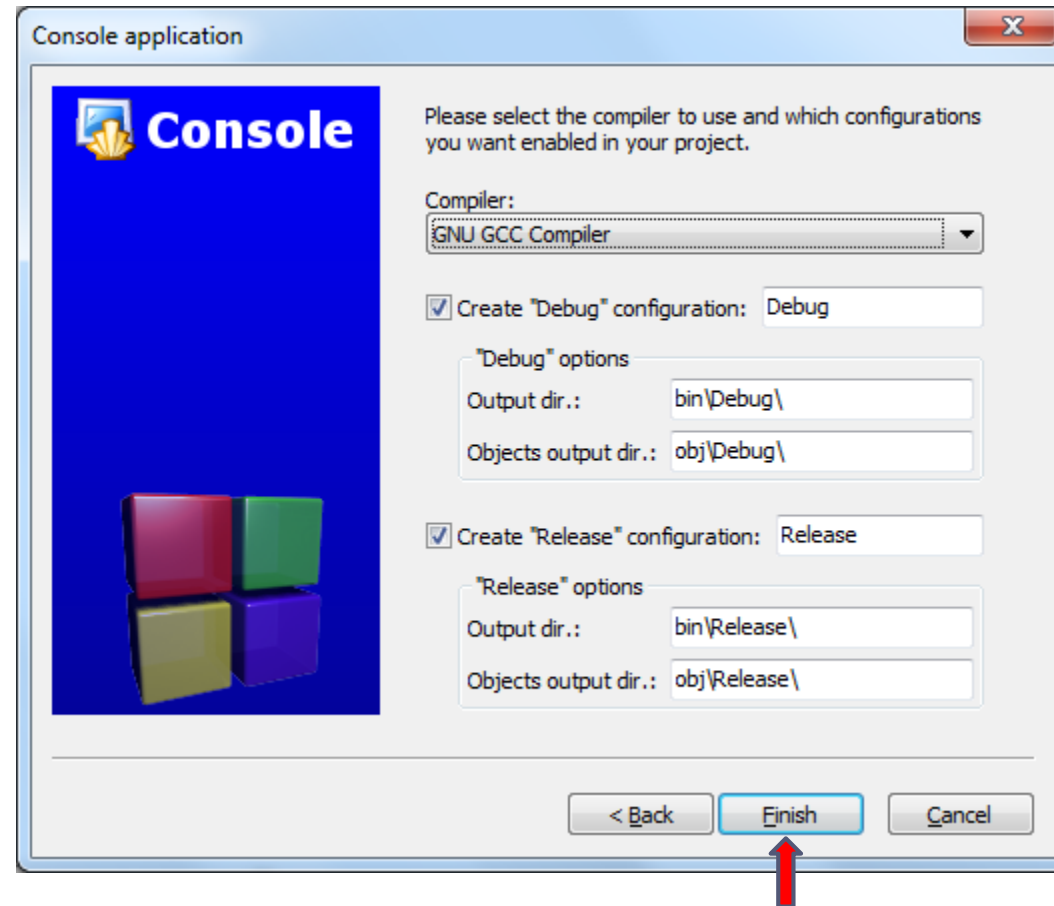
Resulting filename:
C:\TL1\Ejemplo 1\Ejemplo 1.cbp

< Back Next > Cancel

Estas se
completan
solas

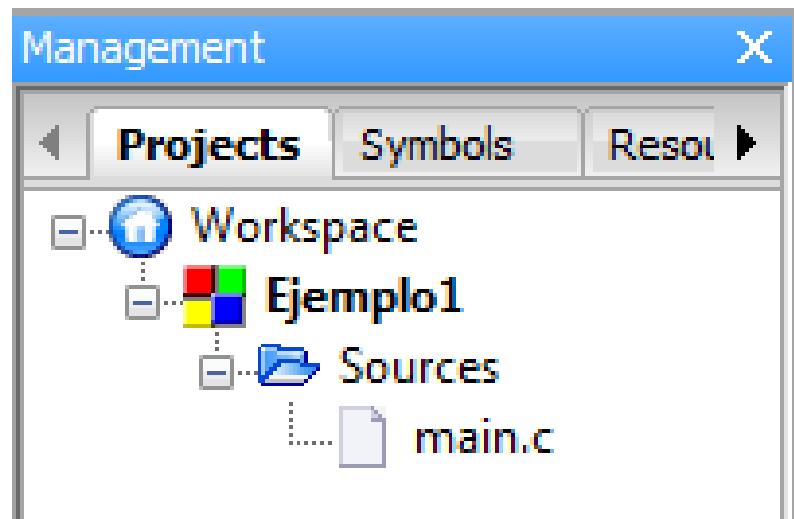
Creando una aplicación de consola

- Paso 6 : Indicar el compilador a utilizar



Creando una aplicación de consola

- Luego de haber creado la aplicación de consola el administrador de proyectos mostrará lo siguiente:



Ya estamos en condiciones de comenzar a trabajar con el lenguaje



Qué es un identificador?

En C, un **identificador** es una combinación de caracteres siendo el primero una letra del alfabeto o un símbolo de subrayado y el resto cualquier letra del alfabeto, cualquier dígito numérico ó símbolo de subrayado.

- **IMPORTANTE**

- Se distinguen mayúsculas de minúsculas.

Ej: los identificadores **TALLER**, **Taller** y **taller** son todos distintos.

- De acuerdo al estándar ANSI-C, sólo serán significativos los primeros 31 caracteres de un identificador. Todo carácter mas allá de este límite será ignorado por cualquier compilador que cumpla la norma ANSI-C.



Identificadores en C

- El compilador utiliza identificadores iniciados con **doble subrayado** o con un subrayado seguido de una letra mayúscula.
- Evite el **uso del subrayado** para iniciar un identificador. Esto reducirá los errores de compilación.
- La **legibilidad** de un programa se incrementa notablemente al utilizar nombres descriptivos para las variables.

Los programadores de Pascal tienden a utilizar nombres descriptivos largos, pero la mayoría de los programadores C por lo general utilizan nombres cortos y crípticos. Se remarca la importancia de utilizar nombres descriptivos que a su vez eviten comentarios redundantes.



Primer programa en C

```
/* Mi primer programa en C */
```



```
#include <stdio.h>
```

```
int main() {  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de  
Programación\n");  
    return 0;  
}
```

Los comentarios se escriben entre `/* */`
y pueden tener varios renglones



Primer programa en C

```
/* Mi primer programa en C */  
  
#include <stdio.h>  
  
int main() {  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de  
Programación\n");  
    return 0;  
}
```

El programa principal es una función y siempre se llama **main**. Puede tener argumentos. Lo encerrado entre { } es el cuerpo de la función

Primer programa en C

```
/* Mi primer programa en C */  
  
#include <stdio.h>  
  
int main() {  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de Programación\n");  
    return 0;  
}
```

La función **printf** permite mostrar resultado en pantalla.



Primer programa en C

```
/* Mi primer programa en C */
```

```
#include <stdio.h> 
```

```
int main() {  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de Programación\n");  
    return 0;  
}
```

Contiene la definición de la función **printf**




Primer programa en C

```
/* Mi primer programa en C */  
  
#include <stdio.h>  
  
int main() {  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de Programación\n");  
    return 0;  
}
```

\n es una secuencia de escape que indica salto de línea.
Más adelante veremos otras secuencias de escape.



Primer programa en C

```
/* Mi primer programa en C */  
  
#include <stdio.h>  
  
int main(){  
    printf("Bienvenidos a la materia ");  
    printf("Fundamentos de Lenguajes de Programación\n");  
    return 0;   
}
```

No es necesaria en este ejemplo pero siempre se espera que una función devuelva un valor a quien la llamó.
El valor 0 se interpreta como que no hubo error.



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9      printf("Ingrese el 2do. nro :");
10     scanf("%d", &nro2);
11
12     suma = nro1 + nro2;
13     printf("La suma es %d \n", suma);
14
15     return 0;
16 }
17
```



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {
5
6
7
8
9
10
11
12
13
14
15
16
17 }
```

- Todos los programas comienza con **main**
- { marca el inicio de la función
- } indica el final



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6
7
8
9
10
11
12
13
14
15
16 }
17
```

- Declara tres variables de tipo `int` es decir, enteras.
- Un nombre de variable en C es cualquier identificador válido.
- Recuerde que C es sensible a mayúsculas y minúsculas.
- Deben declararse antes de usarse. Usualmente después de la `{` de la función `main`.



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {  int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7
8
9
10
11
12
13
14
15
16  }
17
```

- Imprime en pantalla el texto “Ingrese el 1er.nro:”
- El cursor se queda en la misma línea.



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9
10
11
12
13
14
15
16
17 }
```

- **scanf** ingresa un valor por teclado.
- El primer parámetro es la *cadena de control de formato* e indica el tipo de dato a ingresar por el usuario. El **%d** indica que debe ser entero decimal.
- El segundo parámetro empieza con **&** seguido del nombre de la variable. Más adelante veremos mejor el significado del **&**

Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9      printf("Ingrese el 2do. nro :");
10     scanf("%d", &nro2);
11
12     

- Ingresa un entero por teclado en nro2


13
14
15
16 }
17
```



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9      printf("Ingrese el 2do. nro :");
10     scanf("%d", &nro2);
11
12     suma = nro1 + nro2;
13
14     • Calcula la suma de nro1 y nro2
15
16 }
17
```



Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9      printf("Ingrese el 2do. nro :");
10     scanf("%d", &nro2);
11
12     suma = nro1 + nro2;
13     printf("La suma es %d \n", suma);
14
15
16 }
17
```

- Muestra el resultado. Se reemplazará %d por el valor de suma.

Un segundo ejemplo sencillo

```
main.c x
1  /* lee dos enteros y los suma */
2  #include <stdio.h>
3  int main()
4  {   int nro1, nro2, suma;
5
6      printf("Ingrese el 1er. nro :");
7      scanf("%d", &nro1);
8
9      printf("Ingrese el 2do. nro :");
10     scanf("%d", &nro2);
11
12     suma = nro1 + nro2;
13     printf("La suma es %d \n", suma);
14
15     return 0;
16 }
17
```

- Devuelve 0 indicando que terminó bien.

Ejercicio 1

- Analice el siguiente código e indique cuáles son las instrucciones correctas y cuáles las incorrectas.

```
1  /* Este código tiene errores */
2  #include <stdio.h>
3  int main()
4  {
5      printf("Ingrese un número entero : \n");
6      scanf("d", nro);
7
8      printf("El valor ingresado es %d", &nro);
9
10     return 0;
11 }
12
```



Imprimiendo números decimales con **printf**

%d	Número entero
%6d	Número entero con al menos 6 caracteres de ancho
%f	Número con decimales
%6f	Número con decimales que ocupará al menos 6 caracteres de ancho
%.2f	Número con dos decimales
%6.2f	Número con 6 caracteres como mínimo de ancho y dos decimales (incluidos dentro de los 6)



Imprimiendo números decimales con **printf**

- **Ejemplos**

- `printf("%d", 234)` `/* imprime 234 */`
 - `printf("%6d", 234)` `/* imprime 234 */`
 - `printf("%4f", 234.15)` `/* imprime 234.15 */`
 - `printf("%4.1f", 1234.15)` `/* imprime 1234.2 */`
-
- Note que la longitud máxima sólo se utiliza para completar con blancos adelante cuando el número tiene menos dígitos de los indicados.
 - La cantidad de decimales modifica el resultado porque si son menos completa con cero pero si son más redondea.



Aritmética en C

Operación	Operador en C	Detalle
Suma	+	Suma dos números
Resta	-	Resta dos números
Multiplicación	*	Multiplica dos números
División	/	El resultado de la división entre enteros es entero. Ej : 22 / 5 da como resultado 4 22.0 / 5 da como resultado 4.4
Módulo	%	r % s retorna el resto de dividir r por s. Ej : 7 % 4 da como resultado 3



Orden de operadores

Operador	Operación	Orden de cálculo (precedencia)
()	Paréntesis	Se calculan primero. Si están anidados, la expresión del par más interno se evalúa primero. Si están al mismo nivel se evalúan de izquierda a derecha.
* / %	Multiplicación, División y Módulo	Se evalúan en 2do. lugar. Si existen varias se calcularán de izquierda a derecha.
+ -	Suma o Resta	Se calculan al final. Si existen varios serán evaluados de izquierda a derecha.

Operadores Relacionales

Operador	Ejemplo	<u>Significado</u>
==	$x == y$	x es igual a y
!=	$x != y$	x no es igual a y
>	$x > y$	x es mayor que y
<	$x < y$	x es menor que y
>=	$x >= y$	x es mayor o igual que y
<=	$x <= y$	x es menor o igual que y



Operadores lógicos

Operador	Operación lógica
&&	AND
	OR
!	NOT



Tipos de datos simples

Denominación	Tipo de Datos
char	Caracter
int	Número entero
float	Número real de precisión simple
double	Número real de precisión doble



Tipo INT

Tipo de dato	Memoria (bytes)	Rango de valores
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295

- Los tamaños pueden variar con el compilador.



Tipos de datos reales

Tipo de dato	Memoria (bytes)	Rango de valores	Formato
float	4	1.175494351e-38 a 3.402823466e+38	%f %e
double	8	2.22507385850720e-308 a 1.79769313486231e+308	%lf %le
long double	12	3.36210314311209e-4932 a 1.18973149535723e+4932	%lf %le



Códigos de formato para tipos INT

Tipo de dato	Formato
short int	%hd
unsigned short int	%hu
int	%d
unsigned int	%u
long int	%ld
unsigned long int	%lu



Tipos de datos (de mayor a menor)

Tipo	printf	scanf
long double	%Lf	%Lf
double	%lf	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short int	%hu	%hu
short int	%hd	%hd
unsigned char	%u	%u
short	%hd	%hd
char	%c	%c



Tipo de dato lógico

- En C no existe el tipo de dato lógico. En su lugar se utiliza un entero representando con 0 el valor falso y cualquier otro valor (generalmente 1) el valor verdadero.

```
if (valor>100)
    printf("Es mayor\n");

if (40)
    printf("Se imprime siempre\n");

if (0)
    printf("No se imprime nunca");
```



Ejercicio 2

- Suponga que $i=1$, $j=2$, $k=3$, $m=2$. Qué imprime cada uno de los siguientes enunciados?
 - `printf("%d", i == 1);`
 - `printf("%d", j == 3);`
 - `printf("%d", i >= 1 && j > 4);`
 - `printf("%d", m <= 99 && k < m);`
 - `printf("%d", j >= i || k == m);`
 - `printf("%d", k + m < j || 3 - j >= k);`



Conversión explícita de tipos

```
1  #include <stdio.h>
2  int main()
3  {
4      float AVG;
5      int suma, cant;
6
7      suma = 232;
8      cant = 10;
9
10     AVG = (float) suma / cant;
11
12     printf("Promedio = %f", AVG);
13
14     return 0;
15 }
```

Convierte el valor entero de **suma** en un flotante ANTES de dividir por **cant**. El resultado será un número flotante.



Selección

- Estructuras de selección
 - if
 - if - else
- Operador ternario



Estructura de selección **if**

Sintaxis

```
if (condición)  
  /* Acción a realizar si la  
    la condición es verdadera */
```

- Ejemplo

```
if (dato1 > dato2)  
  mayor = dato1;
```

```
if (condición) {  
  /* bloque de acciones a realizar si  
    la condición es verdadera */  
}
```

- Ejemplo

```
if (dato1 > dato2) {  
  mayor = dato1;  
  printf("%d", dato1);  
}
```



Ejemplo 4

```
1  #include <stdio.h>
2  int main()
3  {   int dato1, dato2, mayor;
4
5      printf("ingrese dos enteros : ");
6      scanf("%d %d", &dato1, &dato2);
7
8      mayor = dato2;
9
10     if (dato1>dato2) {
11         printf("el primer valor es mayor");
12         mayor = dato1;
13     }
14     printf("El mayor valor ");
15     printf("ingresado fue %d \n", mayor);
16
17     return 0;
18 }
19
```

- Qué imprime?



Ejercicio 4

- Qué imprime?

```
#include <stdio.h>
int main()
{
    float nro1, nro2, menor;

    printf("Ingrese nro1 : ");
    scanf("%f", &nro1);

    printf("Ingrese nro2 : ");
    scanf("%f", &nro2);

    menor = nro2;
    if (nro1 < nro2)    menor = nro1;

    printf("\nEl valor menor es %f", menor);

    return 0;
}
```

%f indica que se leerá un número con decimales.



Operador ternario

- Ejemplo

```
int a = 10, b = 20, c;  
if (a < b) {  
    c = a;  
}  
else {  
    c = b;  
}  
printf("%d", c);
```

- Operador Ternario:

Sintaxis:

condición ? valor_true : valor_false

Esta sentencia se evalúa a valor_true si se cumple la condición, y a valor_false en caso contrario.

```
int a = 10, b = 20, c;  
c = (a < b) ? a : b;  
printf("%d", c);
```



Operador condicional

- Es el único operador ternario de C
- **Sintaxis**

Expresión lógica ? valor1 : valor2

- Evalúa la expresión y si es verdadera devuelve **valor1** sino devuelve **valor2**.
- Por lo general, **valor1** y **valor2** son del mismo tipo lo que determina el valor de toda la expresión.
- **Ejemplo:**
 - Mayor = dato1>dato2 ? dato1 : dato2



Estructura de selección **if - else**

Sintaxis

```
if (condición) {  
    /* Acción o bloque de acciones a realizar si la  
    condición es verdadera */  
}  
else { /* Acción o bloque de acciones a realizar si la  
    condición es false */  
}
```

- A diferencia de Pascal
 - No tiene **then**
 - El bloque se marca con { } en lugar de usar **begin-end**



Ejercicio 4b

```
/* Selección */
#include <stdio.h>
int main()
{
    float nro1, nro2, menor;

    printf("Ingrese nro1 : ");
    scanf("%f", &nro1);

    printf("Ingrese nro2 : ");
    scanf("%f", &nro2);

    menor = (nro1 < nro2) ? nro1 : nro2;

    printf("\nEl valor menor es %f", menor);

    return 0;
}
```



Ejercicio 4c

```
/* Selección */
#include <stdio.h>
int main()
{   float nro1, nro2;

    printf("Ingrese nro1 : ");
    scanf("%f",&nro1);

    printf("Ingrese nro2 : ");
    scanf("%f",&nro2);

    printf("\nEl valor menor es %f",
           (nro1<nro2) ? nro1 : nro2);

    return 0;
}
```



Estructura iterativa condicional **while**

- **Sintaxis**

while (condición)

/* acción o bloque de acciones a realizar mientras
la condición sea verdadera */

- **Ejemplo**

```
dato = 0;
```

```
while (dato<10) dato = dato + 1;
```

```
printf(“%d \n”, dato);
```



Operadores de asignación

- Asuma : **int c=3, d=5, e=4, f=6, g=12**

Operador	Ejemplo	Explicación	Asigna
+=	c += 7	c = c + 7	10 a c
-=	d -= 4	d = d - 4	1 a d
*=	e *= 5	e = e * 5	20 a e
/=	f /= 3	f = f / 3	2 a f
%=	g %= 9	g = g % 9	3 a g



Operadores incrementales y decrementales

Operador	Ejemplo	Explicación
++	++a	Se incrementa a en 1 y luego se utiliza el nuevo valor de a en la expresión en la cual reside a .
++	a++	Utilizar el valor actual de a en la expresión en la cual reside a y después se incrementa a en 1
--	--b	Se decrementa b en 1 y a continuación se utiliza el nuevo valor de b en la expresión en la cual reside b .
--	b--	Se utiliza el valor actual de b en la expresión en la cual reside b y después se decrementa a b en 1

Ejemplo 5

- Qué imprime?

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

    c = 5;
    printf("%d \n", c);
    printf("%d \n", c++); /* postincremento */
    printf("%d \n", c);

    c = 5;
    printf("%d \n", c);
    printf("%d \n", ++c); /* preincremento */
    printf("%d \n", c);

    return 0;
}
```



Sentencia **for**

- **Sintaxis**

for (**inicialización** ; **condición** ; **acciones_posteriores**)
/* acción o bloque de acciones
pertenecientes al cuerpo del **for** */

donde

- **inicialización** : es una acción o una secuencia de acciones separadas por comas que se ejecuta ANTES de iniciar el **for**.
- **condición** : es una expresión lógica cuyo valor se evalúa ANTES de iniciar el **for** y debe ser verdadera para que el **for** se ejecute.
- **acciones_posteriores** : es una acción o una secuencia de acciones separadas por comas que se ejecutan LUEGO de las instrucciones del **for**.



Ejemplo 6

```
/* Enteros entre 1 y 9 */  
#include <stdio.h>  
int main()  
{   int i;  
  
    for (i=1; i<10; i=i+1)  
        printf("i = %d \n", i);  
  
    return 0;  
}
```



Ejemplos

- La variable de control va de 1 a 100 con paso 1
`for (i=1; i<=100; i++)`
- La variable de control va de 100 a 1 decrementándose en 1 con cada paso
`for (i=100; i>=1; i--)`
- La variable de control va de 7 a 77 en pasos de 7
`for (i=7; i<=77; i+=7)`
- La variable j toma los valores 17, 14, 11, 8, 5 y 2.
`for (j=17; j>0; j -=3)`



Ejemplo 7

```
/* Tabla Fahrenheit-Celsius */
#include <stdio.h>
int main()
{   int fahr;
    float celsius;

    for (fahr=0; fahr<=300; fahr += 20)
    {
        celsius = (5.0/9)*(fahr-32);
        printf("%3d    %6.1f \n", fahr, celsius);
    }
    return 0;
}
```



Break y Continue

- Las instrucciones **break** y **continue** permiten alterar la ejecución de las estructuras iterativas.
- **break** :Al ejecutarla, la iteración termina y la ejecución del programa continua en la próxima línea a la estructura iterativa.
- **continue** :al ejecutarla se saltean las instrucciones que siguen hasta terminar la iteración actual y el loop continua por la siguiente iteración.

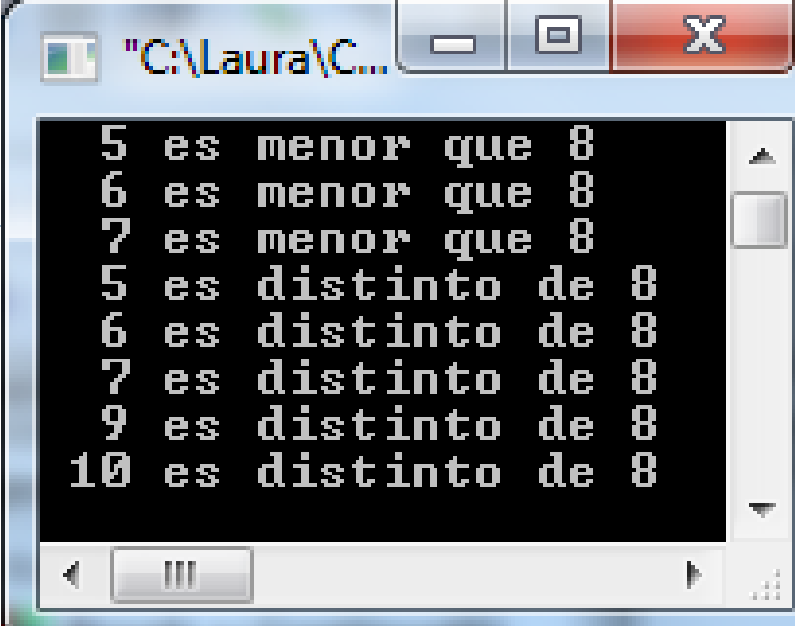


Ejemplo 8

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

    for(n=5; n<=10; n++)
    {   if (n==8)
        break;
        printf("%d es menor que 8\n", n);
    }

    for(n=5; n<=10; n++)
    {   if (n==8)
        continue;
        printf("%d es distinto de 8\n", n);
    }
    return 0;
}
```



```
5 es menor que 8
6 es menor que 8
7 es menor que 8
5 es distinto de 8
6 es distinto de 8
7 es distinto de 8
9 es distinto de 8
10 es distinto de 8
```

Sentencia switch

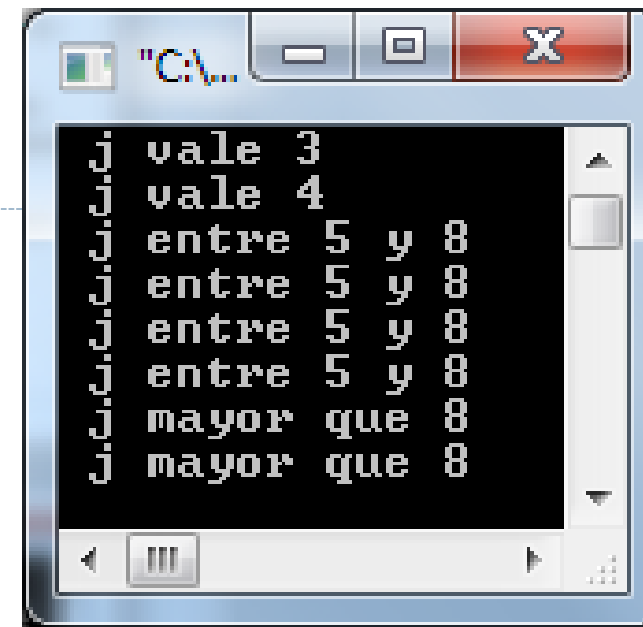
- Permite realizar selección múltiple
- Sintaxis

```
switch (variable)
{ case valor1 :
    /* acción o acciones a realizar */
    break;
  case valor2 :
    /* acción o acciones a realizar */
    break;
  ...
  default :
    /* acción o acciones por defecto */
}
```



Ejemplo 9

```
/* Qué imprime? */
#include <stdio.h>
int main()
{   int j;
    for (j=3; j<=10; ++j)
        switch (j)
        {
            case 3 : printf(" j vale 3\n"); break;
            case 4 : printf(" j vale 4\n"); break;
            case 5 :
            case 6 : case 7 :
            case 8 : printf(" j entre 5 y 8\n"); break;
            default :
                printf(" j mayor que 8\n");
        }
    return 0;
}
```



```
j vale 3
j vale 4
j entre 5 y 8
j entre 5 y 8
j entre 5 y 8
j entre 5 y 8
j mayor que 8
j mayor que 8
```



Sentencia condicional iterativa **do-while**

- Sintaxis

do

/* acción o bloque de acciones */

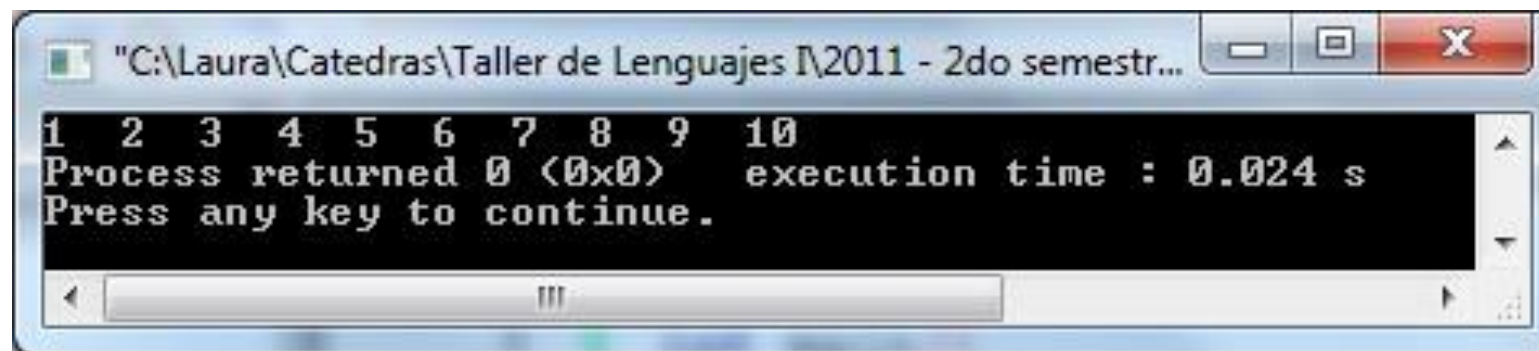
while (condición)

- Note que la condición no se verifica hasta que no se hayan ejecutado las instrucciones indicadas entre las palabras **do** y **while**.
- Al igual que la instrucción **while** itera mientras la condición sea verdadera.



Ejemplo 10

```
/* Usando do-while */  
#include <stdio.h>  
int main()  
{   int cant=1;  
    do  
        printf("%d  ",cant);  
    while (++cant<=10);  
    return 0;  
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Laura\Catedras\Taller de Lenguajes I\2011 - 2do semestr...". The window contains the following text:

```
1 2 3 4 5 6 7 8 9 10  
Process returned 0 (0x0)   execution time : 0.024 s  
Press any key to continue.
```

The output displays the numbers 1 through 10, each followed by two spaces, on a single line. Below the output, it shows the process return code and execution time, and a prompt to press any key to continue.

FUNCIONES EN C

FUNCIONES

- Los módulos en C se llaman **funciones**.
- Por ejemplo funciones de la biblioteca estandar "stdio.h" como por ejemplo **printf** y **scanf**.
- Se verán algunas funciones de la biblioteca matemática y luego la manera de definir funciones en C.
- En C, las funciones **sólo reciben parámetros por valor**



FUNCIONES MATEMÁTICAS (MATH.H)

Función	Descripción	Ejemplo
<code>sqrt(x)</code>	Raíz cuadrada de x	<code>sqrt(900.0)</code> es 30.0
<code>exp(x)</code>	Función exponencial e^x	<code>exp(1.0)</code> es 2.718282 <code>exp(2.0)</code> es 7.389056
<code>log(x)</code>	Logaritmo natural de x (base e)	<code>log(2.718282)</code> es 1.0 <code>log(7.389056)</code> es 2.0
<code>log10(x)</code>	Logaritmo de x base 10	<code>log10(1.0)</code> es 0.0 <code>log10(10.0)</code> es 1.0



FUNCIONES MATEMÁTICAS (MATH.H)

Función	Descripción	Ejemplo
<code>fabs(x)</code>	Valor absoluto de x	Si $x > 0$, <code>fabs(x)</code> es x Si $x = 0$, <code>fabs(x)</code> es 0 Si $x < 0$, <code>fabs(x)</code> es -x
<code>ceil(x)</code>	Redondea x al entero menor que no sea inferior a x	<code>ceil(9.2)</code> es 10.0 <code>ceil(-9.8)</code> es -9.0
<code>floor(x)</code>	Redondea x al entero más grande no mayor que x	<code>floor(9.2)</code> es 9.0 <code>floor(-9.8)</code> es -10.0
<code>pow(x,y)</code>	x elevado a la potencia y (x^y)	<code>pow(2,7)</code> es 128.0 <code>pow(9, 0.5)</code> es 3.0

FUNCIONES EN C

- **Sintaxis**

```
TipoValorRetornado NombreDeLaFuncion(parámetros)
{ declaraciones locales
  Instrucciones de la función
}
```

- **Ejemplo**

```
int Cuadrado ( int nro)
{ int resultado;
  resultado = nro * nro;
  return (resultado) ;
}
```

```
int Cuadrado ( int nro)
{
  return (nro * nro) ;
}
```



```
/* Función definida por el programador */  
#include <stdio.h>
```

```
double cuadrado(double);
```

← Prototipo de la función

```
int main()
```

```
{ int x;
```

```
  for (x=1; x<=10; x++) {
```

```
    printf("%5.0lf \n", cuadrado(x));
```

```
  }
```

```
  return 0;
```

```
}
```

Definición de la función

```
double cuadrado(double a)
```

```
{ return (a*a);
```

```
}
```

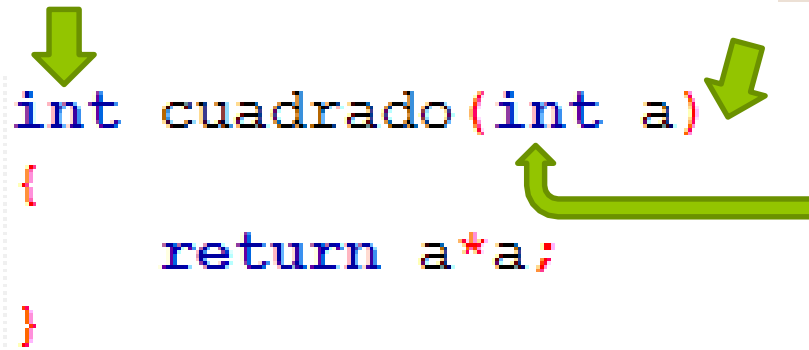
Invocación



DEFINICIÓN DE UNA FUNCIÓN

Si se omite el tipo del valor a devolver se asumirá **int**

Note que no lleva ; al cerrar el paréntesis



```
int cuadrado(int a)
{
    return a*a;
}
```

Cada parámetro debe ir precedido por el nombre del tipo. Si se omite se asume **int**

- También pudo haberse codificado así pero **NO cumple con ANSI C**

```
int cuadrado(a)
int a
{    return a*a
}
```

```
cuadrado(a)
int a
{    return a*a
}
```

```
/* Función definida por el programador */
```

```
#include <stdio.h>
```

```
double cuadrado(double);
```

 **Prototipo de la función**

- Permite al compilador hacer validaciones referidas a los tipos, cantidad y orden de los parámetros y al tipo de valor retornado.
- También podríamos poner

int cuadrado(int a)

pero el compilador ignora los nombres de los parámetros.

- El prototipo de función se caracteriza por la **coerción de argumentos** ya que fuerza su conversión al tipo apropiado.
- Si el prototipo de la función se omite se tomará la primera invocación como prototipo. Esto puede llevar a errores.

RETORNO DE LA FUNCIÓN

- Hay tres formas de regresar al punto desde el cual se hizo la invocación de la función:
 - Si la función no retorna nada, el control sólo se devuelve cuando se llega a la llave derecha que termina la función.
 - Cuando se ejecuta la instrucción
return;
 - Si la función devuelve un resultado, la instrucción
return expresion;
devuelve el valor de **expresion** al punto de llamada.



RETORNO DE LA FUNCIÓN

**Qué
devuelve?**

```
#include <stdio.h>
int main()
{
    int a,b;

    printf("Ingrese dos números enteros : ");
    scanf("%d %d", &a, &b);

    printf("El mayor es %d\n", Mayor(a,b));
    return 0;
}
int Mayor(int nro1, int nro2)
{
    return (nro1);
    if (nro2>nro1)
        return(nro2);
}
```



TIPO VOID

Void es un tipo de dato que representa que no hay valor

```
#include <stdio.h>
void saludo(void);
void main()
{
    saludo();
    saludo();
}
void saludo(void)
{
    printf("Bienvenido al ");
    printf("curso de Taller de ");
    printf("Lenguajes 1!\n\n\n");
}
```

No cumple con el estándar. Se espera que la función **main** siempre retorne un entero



PASAJE DE PARÁMETROS

- El lenguaje **C sólo posee pasaje de parámetros por valor** o por copia. Por lo tanto no es posible cambiar el valor de un parámetro desde una función.
- ¿Cómo hacer entonces para permitir que un función devuelva algo a través de su lista de parámetros?

utilizando punteros



PUNTEROS EN C

PUNTEROS

- Permiten simular el pasaje de parámetros por referencia.
- Permiten crear y manipular estructuras de datos dinámicas.
- Su manejo es de fundamental importancia para programar en C.



PUNTEROS

- Un puntero es una variable que contiene una dirección de memoria.
- Por lo general, una variable contiene un valor y un puntero a ella contiene la dirección de dicha variable.
- Es decir que la variable se refiere **directamente** a un valor mientras que el puntero lo hace **indirectamente**.

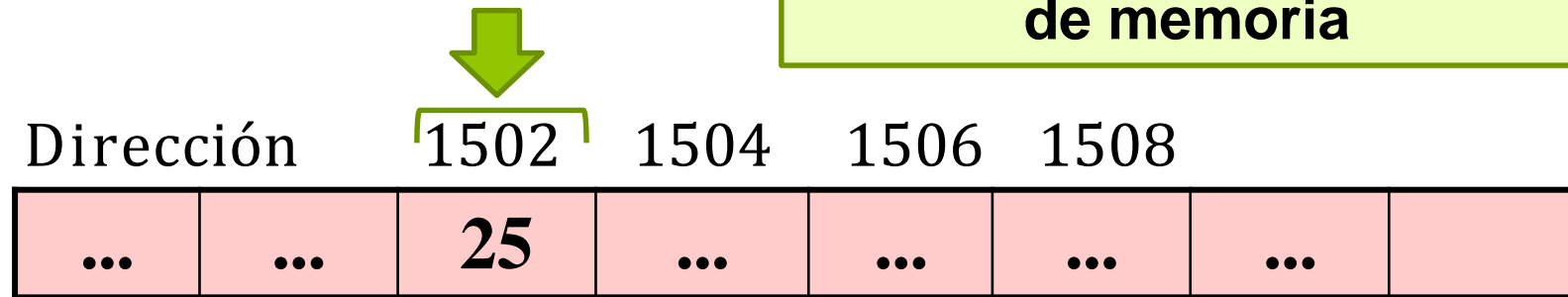


DIRECCIÓN Y CONTENIDO DE MEMORIA

- Una dirección de memoria y su contenido no es lo mismo.

```
int x = 25;
```

Un **puntero** es una variable que contiene una **dirección de memoria**



La **dirección** de la variable **x** es 1502

El **contenido** de la variable **x** es 25

DECLARACIÓN DE PUNTEROS

○ Ejemplo

`int *countPtr, count;`



Puntero a un
entero



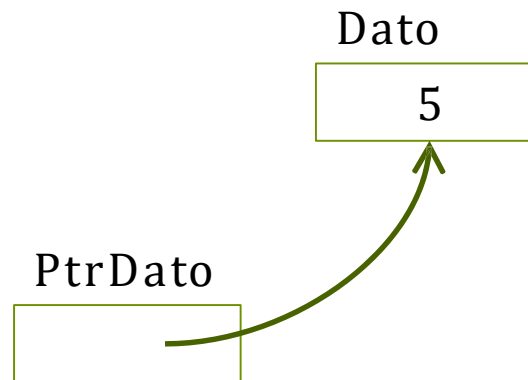
Es un entero
(NO un puntero)

- El * no se aplica a todos los nombres de variables de una declaración. Cada puntero debe llevar su nombre precedido por *.

OPERADORES DE PUNTEROS

- El operador & u *operador de dirección*, es un operador unario que retorna la dirección de su operando.
- **Ejemplo**

```
int Dato = 5;  
int *PtrDato;  
  
PtrDato = &Dato;
```



OPERADORES DE PUNTEROS

- El operador *, también llamado *operador de indirección*, retorna el valor del objeto hacia el cual apunta su operando.

- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%d\n", *PtrDato);
```



Imprime 5

PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

```
#include <stdio.h>
```

```
int main()
```

```
{  int *ptr;  
    int dato=30;
```



Declara un puntero a un entero

```
ptr = &dato;
```

```
*ptr = 50;
```

```
printf("Dato = %d\n", dato);
```

```
return 0;
```

```
}
```



PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

```
#include <stdio.h>
int main()
{   int *ptr;
    int dato=30;

    ptr = &dato;
    *ptr = 50;

    printf("Dato = %d\n", dato);

    return 0;
}
```

& es el **operador de dirección**: permite obtener la dirección de memoria de la variable que le sigue

VISUALIZANDO EL VALOR DE UN PUNTERO

- Puede utilizarse printf con la especificación de conversión **%p** para visualizar el valor de una variable puntero en forma de entero hexadecimal.
- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%p\n", PtrDato);
```

Qué imprime?



VISUALIZANDO EL VALOR DE UN PUNTERO

- Puede utilizarse printf con la especificación de conversión **%p** para visualizar el valor de una variable puntero en forma de entero hexadecimal.
- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%p\n", PtrDato);
```

0028FF1C



```
/* Operadores & y * */
#include <stdio.h>
int main()
{   int Dato = 5, *PtrDato;
    PtrDato = &Dato;
    printf("\n La direccion de Dato es %p\n"
           " El valor de PtrDato es %p\n\n",
           &Dato, PtrDato);
    printf(" La valor de Dato es %d\n"
           " El valor de *PtrDato es %d\n\n",
           Dato, *PtrDato);
    printf(" Note la relacion entre * y & \n"
           " &*PtrDato = %p\n *&PtrDato = %p\n",
           &*PtrDato, *&PtrDato);
    return 0;
}
```

Que imprime?



SALIDA DEL PROGRAMA ANTERIOR

```
La direccion de Dato es 0028FF1C  
El valor de PtrDato es 0028FF1C
```

```
La valor de Dato es 5  
El valor de *PtrDato es 5
```

```
Note la relacion entre * y &  
&*PtrDato = 0028FF1C  
*&PtrDato = 0028FF1C
```



PUNTEROS

- Las **direcciones de memoria** dependen de la arquitectura de la computadora y de la gestión que el sistema operativo haga de ella.
- Desde C no es posible indicar numéricamente una dirección de memoria para guardar información (esto se hace a través de funciones específicas).
- Utilizamos punteros para acceder a la información a través de su dirección de memoria.



INICIALIZACIÓN DE PUNTEROS

- Los punteros deben ser inicializados.
- Utilice el identificador **NULL** (definido en `<stdio.h>`) para indicar que el puntero no apunta a nada.
- El **0** es el único valor entero que puede asignarse directamente a un puntero y es equivalente a **NULL**.
- Cuando se asigna **0** a un puntero se realiza un casting previo automático al tipo apropiado.



EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325; ←
    int *ptr2 = 0;
    int *ptr3 = NULL;

    return 0;
}
```

No es posible asignarle un valor fijo a un puntero. No es posible saber si es una posición válida.

EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325;

    int *ptr2 = 0;      ←
    int *ptr3 = NULL;

    return 0;
}
```

El 0 es el único valor que puede asignarse a un puntero.
La conversión a (int *) es automática.

EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325;

    int *ptr2 = 0;

    int *ptr3 = NULL;

    return 0;
}
```

NULL equivale a 0 y está
definido en <stdio.h>



PASAJE DE PARÁMETROS POR REFERENCIA

PASAJE DE PARÁMETROS POR REFERENCIA

- Vimos que en C los parámetros de las funciones siempre se pasan por valor.
- Para simular el pasaje de parámetro por referencia se utiliza la dirección de la variable, es decir, que lo que se envía es un puntero a su valor.
- El puntero es un parámetro sólo de entrada que permite modificar el valor de la variable a la que apunta.




```
/* parámetro por referencia */
```

```
#include <stdio.h>
```

```
void cuadrado(int *);
```

```
int main()
```

```
{ int a = 5;
```

```
printf("Valor original = %d\n", a);
```

```
cuadrado(&a);
```

Envía la dirección de la variable (un puntero)

```
printf("Valor al cuadrado = %d\n", a);
```

```
return 0;
```

Recibe un puntero a un entero

```
}
```

```
void cuadrado(int * nro)
```

```
{
```

```
*nro = *nro * *nro;
```

Valor de la variable apuntada por **nro**

```
}
```

RECURSIÓN

- Al igual que Pascal, C soporta la definición de funciones recursivas.
- Recuerde que el objetivo de este tipo de funciones es reducir la complejidad del problema.
- Se utiliza un caso base no recursivo y en cada llamada se busca reducir el tamaño de la entrada de manera de cercarse a dicho caso base.
- Las soluciones recursivas si bien facilitan la escritura de la solución ocupan más memoria que las soluciones iterativas.



```
#include <stdio.h>
```

```
unsigned int Factorial(unsigned int);
```

```
int main()
```

```
{
```

```
    printf("El factorial de 8 es");
```

```
    printf(" %u\n", Factorial(8));
```

```
    return 0;
```

```
}
```

```
unsigned int Factorial(unsigned int nro)
```

```
{
```

```
    if ((nro==0) || (nro==1))
```

```
        return 1;
```

```
    else return(nro * Factorial(nro -1));
```

```
}
```

Lenguaje C

- Fin de clase

Próximas clases

- Clase práctica:
 - Viernes 18 y 22/08
- Clase teórica:
 - Jueves 28/08: Datos