

Lenguaje C

Tipos de Datos Simples y Estructuras de Control

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.



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:

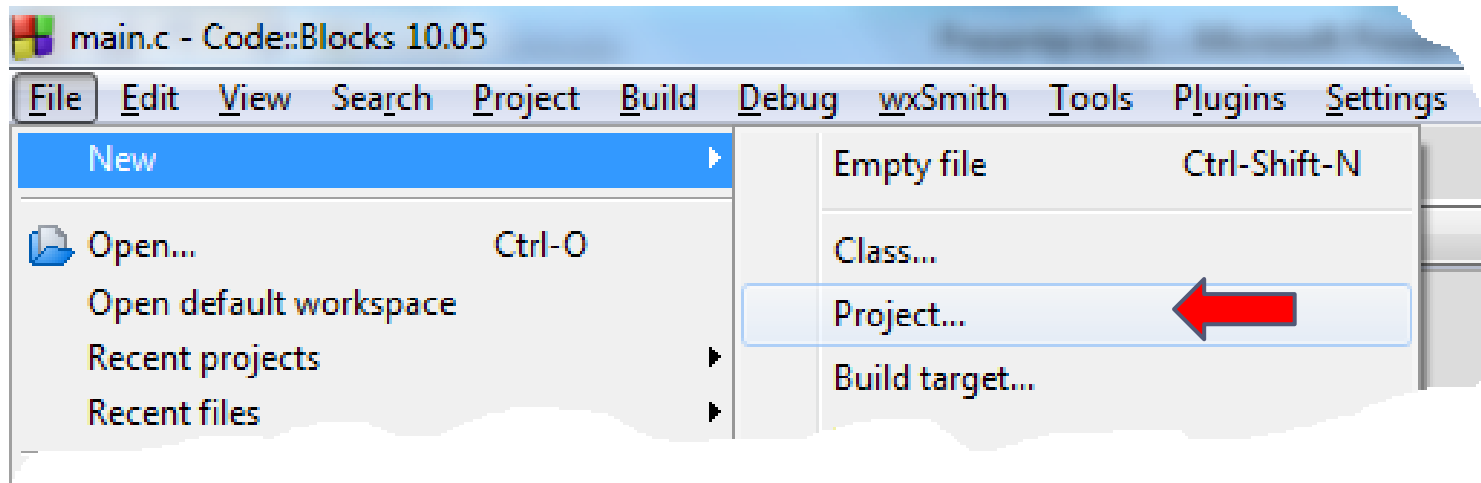
www.codeblocks.org/downloads/26

Elegir alguno que tenga el compilador **GCC** y el debugger **GDB**.
Por ejemplo para Windows 7 descargar codeblocks-10.05mingw-setup.exe



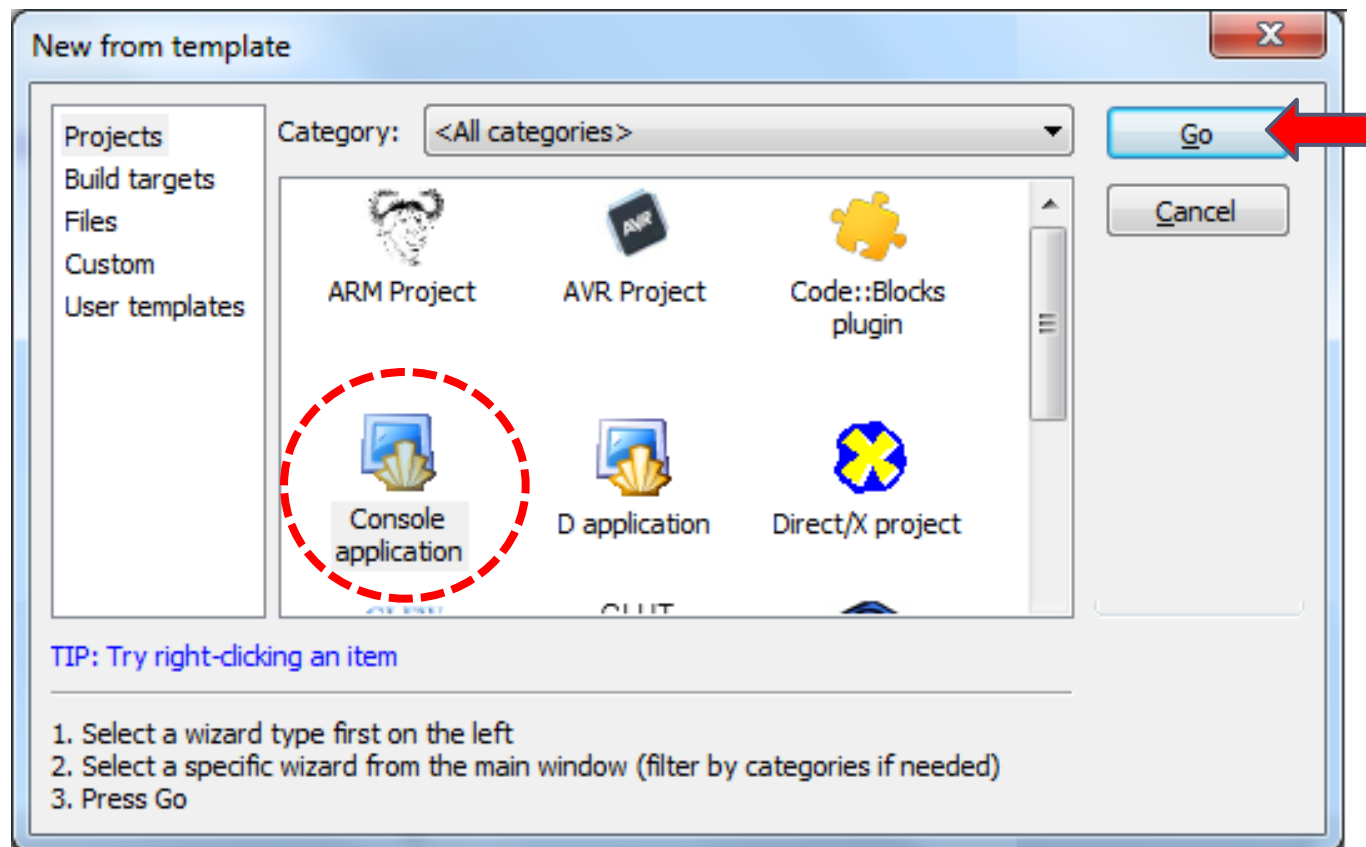
Cómo empezamos a programar?

► **Paso I** : Comenzaremos creando un proyecto



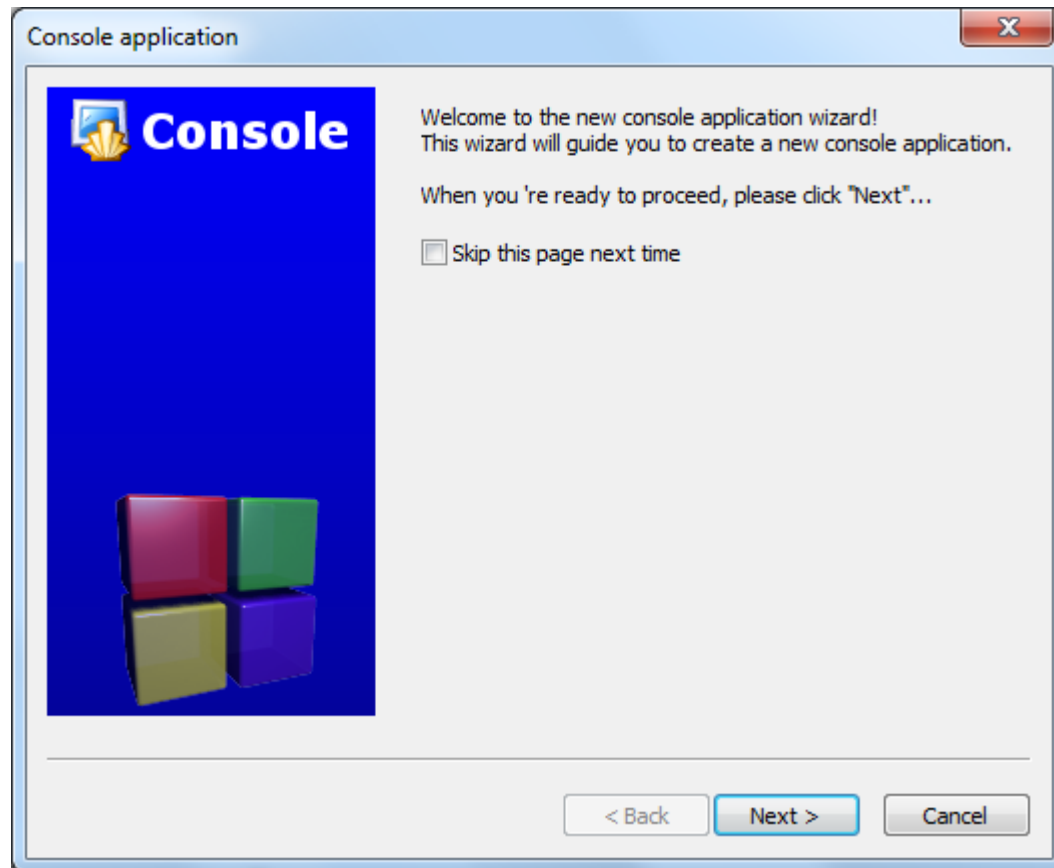
Cómo empezamos a programar?

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



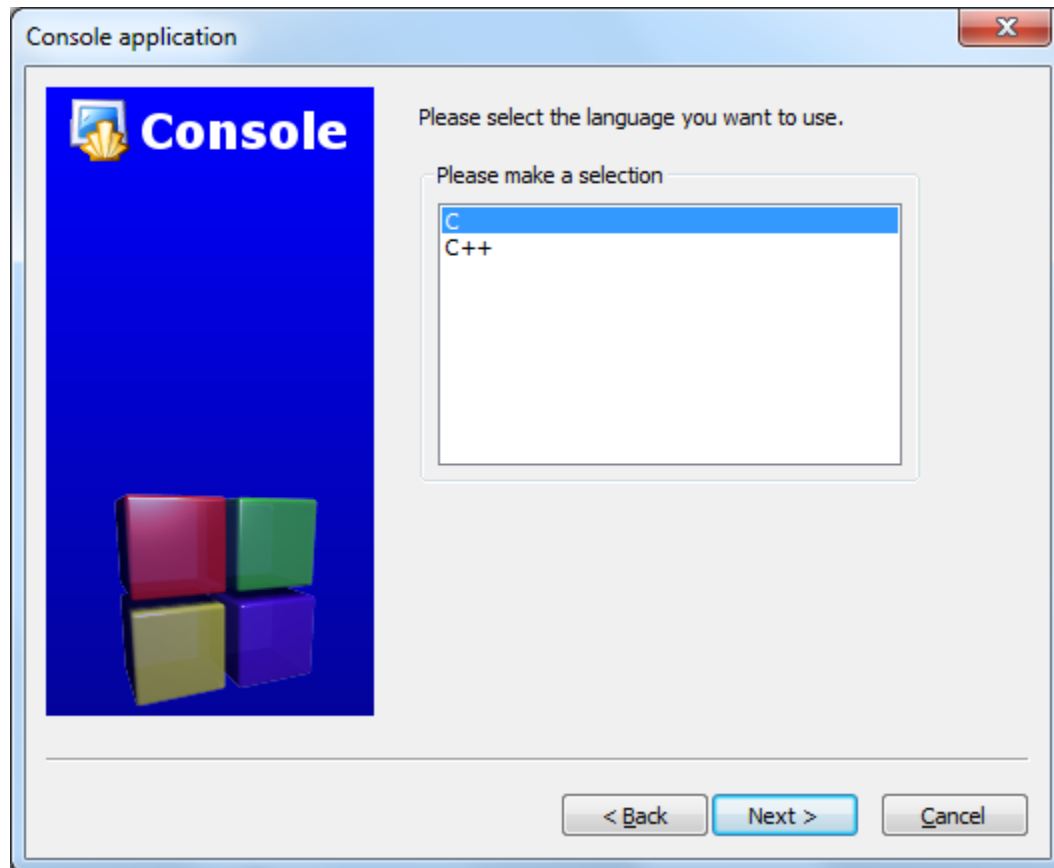
Creando una aplicación de consola

► Paso 3 : Seguir las indicaciones del Wizard ...



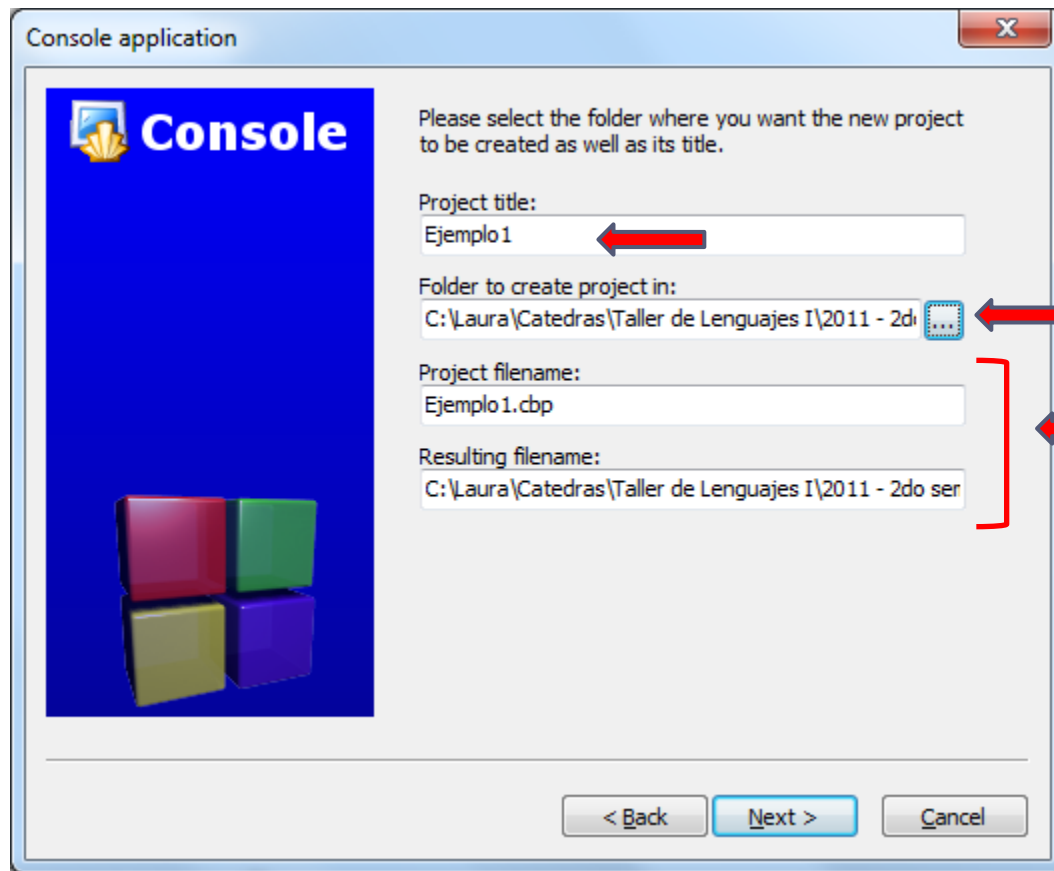
Creando una aplicación de consola

► Paso 4 : Elegir el lenguaje C



Creando una aplicación de consola

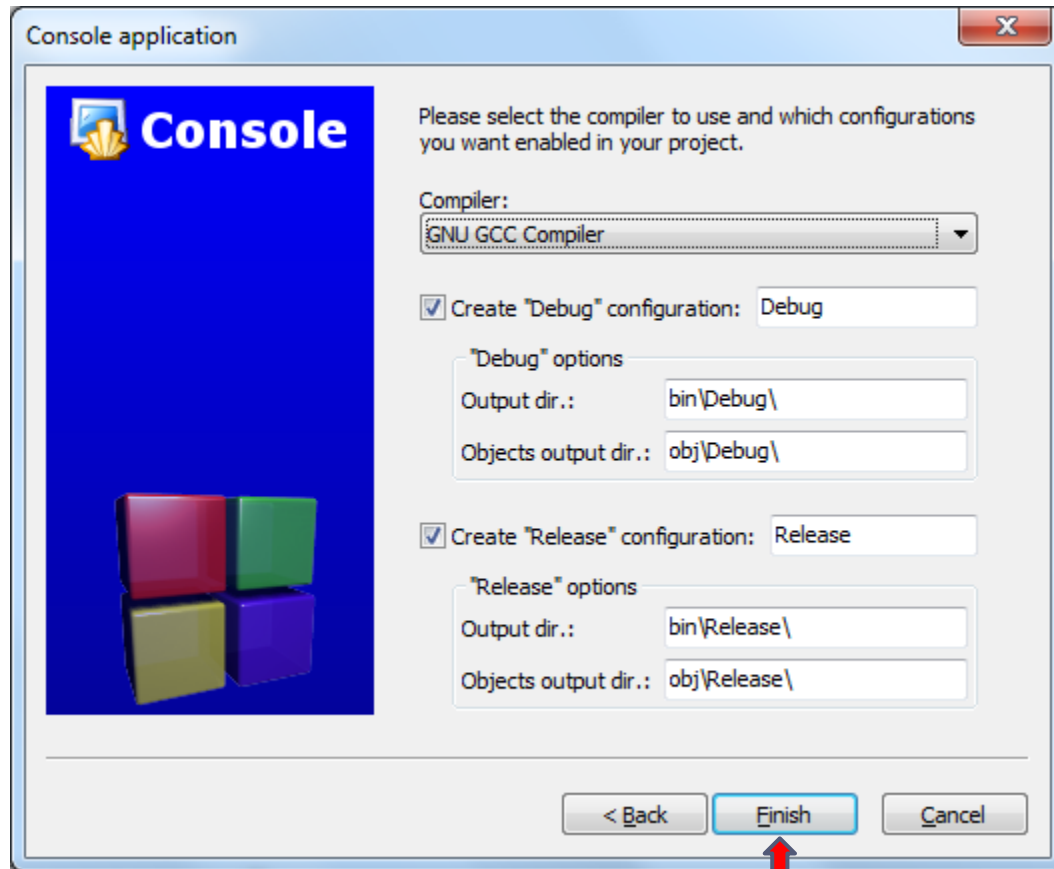
► Paso 5 : Indicar el título y el directorio del proyecto



Estas se completan solas

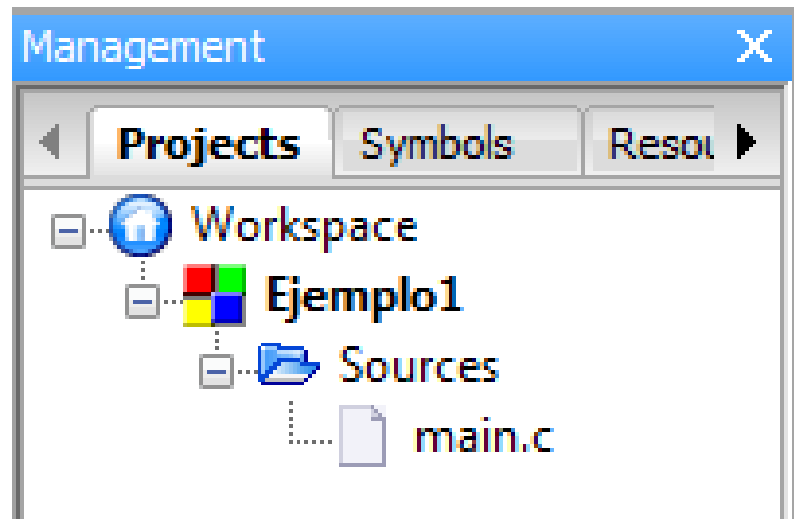
Creando una aplicación de consola

► 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.



Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h>
3  int main()
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0;
9  }
10
```

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

Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h>
3  int main() ←
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0;
9  }
10
```

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

Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h>
3  int main()
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0;
9  }
10
```

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

Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h> ←
3  int main()
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0;
9  }
10
```

Contiene la definición de la función **printf**

Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h>
3  int main()
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0;
9  }
10
```

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

Mi primer programa en C

```
main.c X
1  /* Mi primer programa C */
2  #include <stdio.h>
3  int main()
4  {
5      printf("Bienvenido a ");
6      printf("Taller de Lenguajes 1!\n");
7
8      return 0; ←
9  }
10
```

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
13
14
15
16 }
17
```

► **Ingresar un entero por teclado en nro2**

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



Ejercicio 2

- ▶ Suponga que $i=1$, $j=2$, $k=3$, $m=2$. Qué imprime cada uno de los siguientes enunciados?

- ▶ `printf("%d", i == 1);`

Es verdadero, imprime 1

- ▶ `printf("%d", j == 3);`

Es falso, imprime 0

- ▶ `printf("%d", i >= 1 && j > 4);`

Es falso, imprime 0

- ▶ `printf("%d", m <= 99 && k < m);`

Es falso, imprime 0

- ▶ `printf("%d", j >= i || k == m);`

Es verdadero, imprime 1

- ▶ `printf("%d", k + m < j || 3 - j >= k);`

Es falso, imprime 0



Ejercicio 2

- ▶ Suponga que $i=1$, $j=2$, $k=3$, $m=2$. Qué imprime cada uno de los siguientes enunciados?

- ▶ `printf("%d", ! m);`

Es falso, imprime 0

- ▶ `printf("%d", ! (j-m));`

Es verdadero, imprime 1

- ▶ `printf("%d", ! (k<m));`

Es verdadero, imprime 1

- ▶ `printf("%d", ! (j > k));`

Es verdadero, imprime 1

- ▶ `printf("%d", ! (j - k));`

Es falso, imprime 0



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

- ▶ Los tamaños en bits pueden variar con el compilador.
- ▶ 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.



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.



Ejercicio 3

- ▶ Escriba un programa C que lea de teclado un número entero correspondiente a una temperatura en grados fahrenheit e imprima otro valor entero correspondiente a su conversión a grados celsius según la siguiente ecuación

$$^{\circ}\text{C} = (5/9) * (^{\circ}\text{F} - 32)$$



Selección

- ▶ Estructuras de selección
 - ▶ **if**
 - ▶ **if – else**
- ▶ Operador ternario



Estructura de selección **if**

Sintaxis

```
if (condición)  
  /* Acción a realizar si  
  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

► Qué imprime?

```
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
```

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 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 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



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);
```



Ejercicio 5

- ▶ Escriba un programa en C que lea de teclado una secuencia de números enteros terminada en -1.

Al finalizar deberá imprimir en pantalla el promedio de los valores leídos y el máximo valor ingresado.



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 b en 1

Ejemplo 5

► Qué imprime?

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

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

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

    return 0;
}
```

Imprime

5

5

6

5

6

6

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;
}
```



Ejercicio 6

► Qué imprime?

```
/* Qué imprime? */  
#include <stdio.h>  
int main()  
{   int i;  
  
    for (i=1; i<10; i=i+1)  
        if (i % 3 == 0)  
            printf("i = %d \n", i);  
  
    return 0;  
}
```

Cambia en algo si en lugar de **i=i+1** ponemos **i++** ?

Y si ponemos **++i** ?

Cambia algo si sacamos el **==0** de la condición?

Ejercicio 7

- ▶ Escriba un programa C para calcular la suma de todos los números enteros pares entre 2 y 230



```
/* Suma de pares entre 2 y 230 */
#include <stdio.h>
int main()
{   int nro, suma=0;

    for (nro=2; nro<=230; nro+=2)
        suma = suma + nro;
    printf("Suma = %d \n", suma);

    for(suma=0,nro=2;nro<=230; suma=suma+nro, nro+=2);
    printf("Suma = %d \n", suma);

    for(suma=0,nro=2;nro<=230; suma+=nro, nro+=2);
    printf("Suma = %d \n", suma);

    for(suma=0,nro=2;nro<=230; nro+=2, suma=suma+nro);
    printf("Suma = %d \n", suma);

    return 0;
}
```

► Imprime 4 veces el mismo valor?



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.

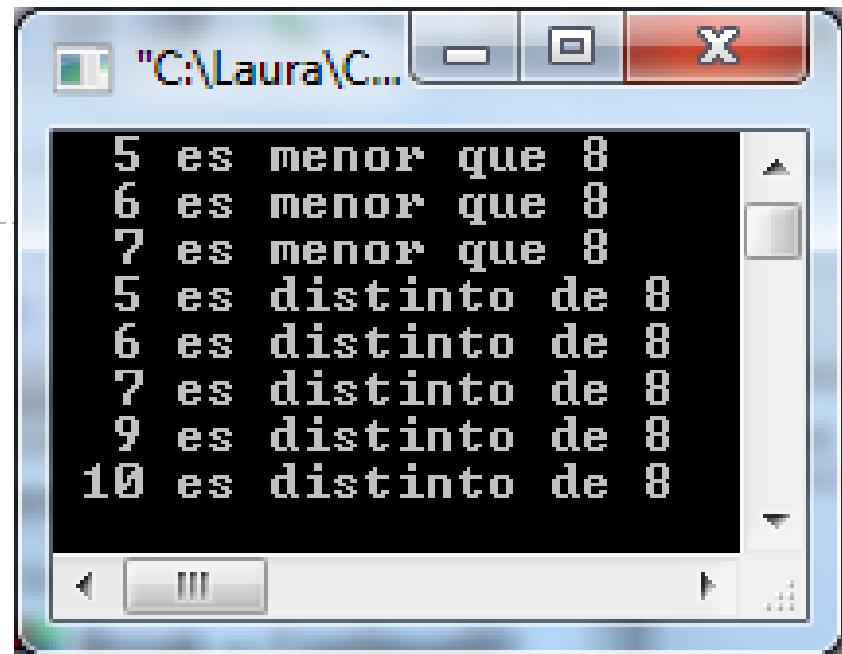


Ejemplo 8

```
#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;
```

```
...
```

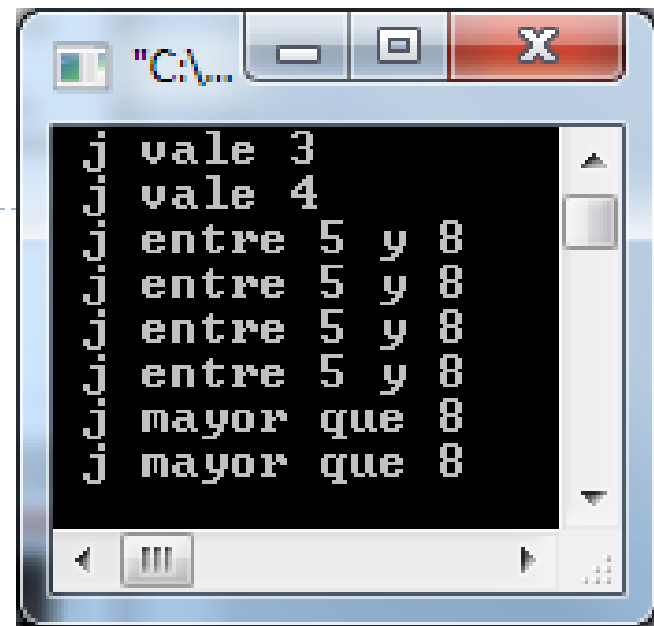
```
default :
```

```
    /* 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 */

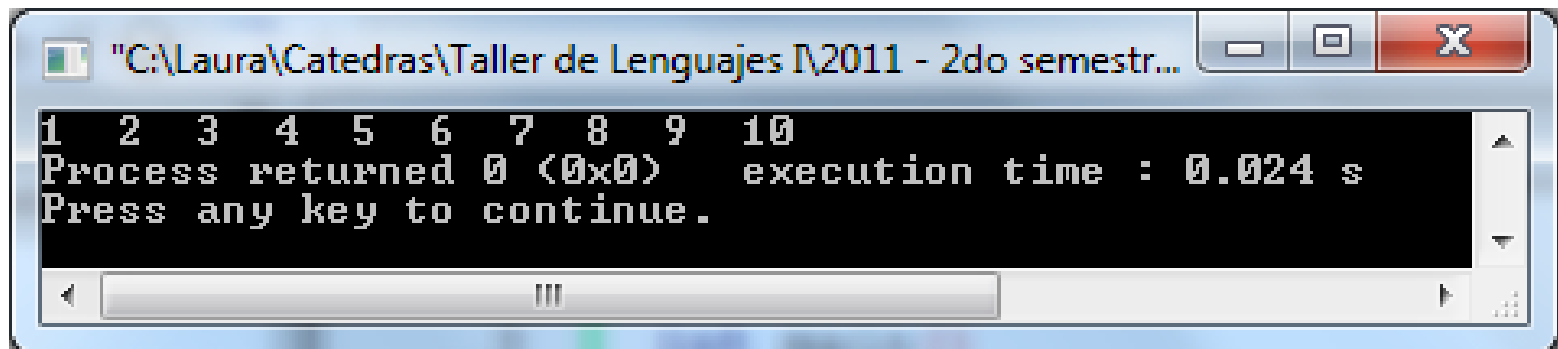
while (condición)

- ▶ 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;  
}
```



```
"C:\Laura\Catedras\Taller de Lenguajes I\2011 - 2do semestr..."  
1 2 3 4 5 6 7 8 9 10  
Process returned 0 (0x0) execution time : 0.024 s  
Press any key to continue.
```

Conversión entre tipos de datos

```
/* Tipos de datos nuevos */
int main()
{
    int a, b, c; /* Entero, de -32768 a 32767 sin punto decimal */
    char x, y, z; /* De -128 a 127 sin punto decimal */
    float numero, gato, casa; /* De 3.4E-38 a 3.4E+38 con punto decimal */

    a = b = c = -27;
    x = y = z = 'A';
    numero = gato = casa = 3.6792;
    a = y; /* a es ahora 65 (caracter A) */
    x = b; /* x es ahora -27 */
    numero = b; /* num será -27.00 */
    a = gato; /* a tomará el valor de 3 */

    return 0;
}
/* Este programa no muestra nada, :- ) */
```

► El tipo de dato **char** es casi igual al entero excepto que solo se le pueden asignar valores entre -128 y 127 (estos valores dependen del tamaño en bytes).

► Al asignar un **float** a un **int** el valor se trunca al entero menor.



Ejercicios Adicionales



Ejercicio 8

- ▶ Analice el siguiente segmento de código e indique los errores que encuentre

```
For ( k = 1; k < 10; k++) ;  
    printf("%3d" , k);
```



Ejercicio 9

- El siguiente código busca informar si el número leído es par o impar. Indique los errores que encuentre

```
scanf("%d", nro);  
switch nro % 2  
{  
    case 0 : printf("Es par\n");  
    case 1 : printf("Es impar\n");  
}
```



Ejercicio 10

- Escriba un programa que haga el siguiente dibujo

```
      *
    ***
  *****
 *****
*****
 *****
  *****
    ***
      *
```

