Uso del compilador.

Uso del compilador.

En adelante, en los *slides* emularemos la terminal con cuadros de texto como el siguiente

```
$
```

Uso del compilador.

En adelante, en los *slides* emularemos la terminal con cuadros de texto como el siguiente

```
$
```

El signo $ se llama *prompt* y es el inicio de la línea de comandos en SO GNU/Linux.

Uso del compilador.

En adelante, en los *slides* emularemos la terminal con cuadros de texto como el siguiente

```
$
```

El signo $ se llama *prompt* y es el inicio de la línea de comandos en SO GNU/Linux.

El signo $ NO debe escribirse.

Uso del compilador.

Uso del compilador.

Si contamos con un archivo llamado `hello.c` con el siguiente contenido:

Uso del compilador.

Si contamos con un archivo llamado `hello.c` con el siguiente contenido:

```c
#include <stdio.h>

int main (void)
{
  printf("Hola, Mundo!\n");

  return 0;
}
```

Uso del compilador.

Uso del compilador.

...podemos compilarlo usando el gcc:

```
$ gcc hello.c
$
```

Uso del compilador.

...podemos compilarlo usando el `gcc`:

```
$ gcc hello.c
$
```

Si no cometimos errores en el programa, el `gcc` no da ningún mensaje y crea un archivo llamado `a.out` el cual podemos ejecutar.

Uso del compilador.

Uso del compilador.

Para ejecutar cualquier archivo en linux, debemos anteponer un punto y una barra

```
$ ./a.out
Hola, Mundo!
$
```

Uso del compilador.

Uso del compilador.

Se puede utilizar el `gcc` con *parámetros* más apropiados a nuestras necesidades

Uso del compilador.

Se puede utilizar el `gcc` con *parámetros* más apropiados a nuestras necesidades

Por ejemplo, para cambiar el nombre del archivo ejecutable se agrega `-o` seguido del nuevo nombre

```
$ gcc hello.c -o saludo
$
```

Uso del compilador.

Uso del compilador.

...luego, se ejecuta como antes, agregando el punto y la barra, ahora invocando el programa con su nuevo nombre

```
$ ./saludo
Hola, Mundo!
$
```

Uso del compilador.

Uso del compilador.

Otros parámetros útiles son:

```
$ gcc -Wall -std=c99 -pedantic-errors hello.c -o saludo
```

Uso del compilador.

Otros parámetros útiles son:

```
$ gcc -Wall -std=c99 -pedantic-errors hello.c -o saludo
```

`-Wall` **W**arning **all**

Uso del compilador.

Otros parámetros útiles son:

```
$ gcc -Wall -std=c99 -pedantic-errors hello.c -o saludo
```

`-Wall` **W**arning **all**

`-std=c99` para el estándar C99

Uso del compilador.

Otros parámetros útiles son:

```
$ gcc -Wall -std=c99 -pedantic-errors hello.c -o saludo
```

`-Wall` **W**arning **all**

`-std=c99` para el estándar C99

`-pedantic-errors` si algo no cumple el estándar da error

Funciones de entrada/salida

Funciones de entrada/salida

Funciones

Las funciones son fragmentos de código que se *ejecutan* cada vez que son *llamadas*.

Funciones de entrada/salida

Funciones

Las funciones son fragmentos de código que se *ejecutan* cada vez que son *llamadas*.

Generalmente las funciones reciben información por parte de *quien* las llame. Esta información se pasa a la función por medio de los paréntesis.

Funciones de entrada/salida

Funciones

Las funciones son fragmentos de código que se *ejecutan* cada vez que son *llamadas*.

Generalmente las funciones reciben información por parte de *quien* las llame. Esta información se pasa a la función por medio de los paréntesis.

Por ejemplo:

Funciones de entrada/salida

Funciones

Las funciones son fragmentos de código que se *ejecutan* cada vez que son *llamadas*.

Generalmente las funciones reciben información por parte de *quien* las llame. Esta información se pasa a la función por medio de los paréntesis.

Por ejemplo:

```
printf("Hola, Mundo!\n");
```

Funciones de entrada/salida

Funciones de entrada/salida

`stdio.h` es el archivo de cabecera que contiene las declaraciones de las funciones de la **biblioteca estándar**

Funciones de entrada/salida

`stdio.h` es el archivo de cabecera que contiene las declaraciones de las funciones de la **biblioteca estándar**

Algunas funciones son:
`printf`, `scanf`, `getchar`, `putchar`

Funciones de entrada/salida

Funciones de entrada/salida

Para usar estas funciones debe incluirse este archivo con la directiva de preprocesador `#include`

Funciones de entrada/salida

Para usar estas funciones debe incluirse este archivo con la directiva de preprocesador `#include`

```c
#include <stdio.h>

int main (void)
{
  printf("Hola, Mundo!\n");

  return 0;
}
```

Función printf

Función printf

Como se vió, `printf` es una función para imprimir en pantalla.

Función printf

Como se vió, `printf` es una función para imprimir en pantalla.

Esta función debe recibir la cadena de caracteres para imprimir. Por ejemplo, en los ejemplos se usó la cadena `"Hola, Mundo!\n"`

Función printf

Como se vió, `printf` es una función para imprimir en pantalla.

Esta función debe recibir la cadena de caracteres para imprimir. Por ejemplo, en los ejemplos se usó la cadena `"Hola, Mundo!\n"`

Como se puede ver en los ejemplos, el `\n` no se imprime.

Función printf

Como se vió, `printf` es una función para imprimir en pantalla.

Esta función debe recibir la cadena de caracteres para imprimir. Por ejemplo, en los ejemplos se usó la cadena `"Hola, Mundo!\n"`

Como se puede ver en los ejemplos, el `\n` no se imprime.

La barra invertida (\) se llama **caracter de escape**, y cambia el *significado* del caracter que sigue.

Función printf

Función printf

Secuencias de escape de la función `printf`

Escape	Descripción
\n	Nueva línea
\t	Tabulador horizontal
\v	Tabulador vertical
\b	Retroceso
\r	Retorno de carro
\\	Diagonal invertida
\ ”	Comillas

Función printf

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tres.c

int main (void)
{
  printf("Uno\nDos\nTres\n");

  return 0;
}
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tres.c

int main (void)
{
  printf("Uno\nDos\nTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tres.c
$
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tres.c

int main (void)
{
  printf("Uno\nDos\nTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tres.c
$ ./a.out
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tres.c

int main (void)
{
  printf("Uno\nDos\nTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tres.c
$ ./a.out
Uno
Dos
Tres
$
```

Función printf

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tabs.c

int main (void)
{
  printf("Uno\tDos\tTres\n");

  return 0;
}
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tabs.c

int main (void)
{
  printf("Uno\tDos\tTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tabs.c
$
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tabs.c

int main (void)
{
  printf("Uno\tDos\tTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tabs.c
$ ./a.out
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-tabs.c

int main (void)
{
  printf("Uno\tDos\tTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-tabs.c
$ ./a.out
Uno     Dos     Tres
$
```

Función printf

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-retroceso.c

int main (void)
{
  printf("Uno\tDos\rTres\n");

  return 0;
}
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-retroceso.c

int main (void)
{
  printf("Uno\tDos\rTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-retroceso.c
$
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-retroceso.c

int main (void)
{
  printf("Uno\tDos\rTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-retroceso.c
$ ./a.out
```

Función printf

Ejemplos

```c
#include <stdio.h>
// u3-retroceso.c

int main (void)
{
  printf("Uno\tDos\rTres\n");

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-retroceso.c
$ ./a.out
Tres   Dos
$
```

Función printf

Función printf

Especificadores de conversión o de formato

Especificadores	Descripción
%c	Caracter
%d o %i	Entero decimal con signo
%u	Entero decimal sin signo
%f	Decimal de punto flotante

Función printf

Especificadores de conversión o de formato

Función printf

Especificadores de conversión o de formato

```c
#include <stdio.h>
// u3-conversion.c

int main (void)
{
  printf("%f\n", 3.14);
  printf("%d %d %d\n", 1, 2, 4);

  return 0;
}
```

Función printf

Especificadores de conversión o de formato

```c
#include <stdio.h>
// u3-conversion.c

int main (void)
{
  printf("%f\n", 3.14);
  printf("%d %d %d\n", 1, 2, 4);

  return 0;
}
```

Por cada especificador que tenga la cadena de texto se espera un valor que tomará su lugar, todos separados por comas (llamados **argumentos**)

Función printf

Función printf

La cadena puede tener cualquier caracter válido junto con los especificadores de formato

Función printf

La cadena puede tener cualquier caracter válido junto con los especificadores de formato

```c
#include <stdio.h>
// u3-conversion-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 3);

  return 0;
}
```

Función printf

La cadena puede tener cualquier caracter válido junto con los especificadores de formato

```c
#include <stdio.h>
// u3-conversion-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 3);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-suma.c
$
```

Función printf

La cadena puede tener cualquier caracter válido junto con los especificadores de formato

```c
#include <stdio.h>
// u3-conversion-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 3);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-suma.c
$ ./a.out
```

Función printf

La cadena puede tener cualquier caracter válido junto con los especificadores de formato

```c
#include <stdio.h>
// u3-conversion-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 3);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-suma.c
$ ./a.out
1+2=3
$
```

Función printf

Función printf

Los valores que usan los especificadores pueden ser el resultado de operaciones

Función printf

Los valores que usan los especificadores pueden ser el resultado de operaciones

```c
#include <stdio.h>
// u3-conversion-op-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 1+2);

  return 0;
}
```

Función printf

Los valores que usan los especificadores pueden ser el resultado de operaciones

```c
#include <stdio.h>
// u3-conversion-op-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 1+2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-op-suma.c
$
```

Función printf

Los valores que usan los especificadores pueden ser el resultado de operaciones

```c
#include <stdio.h>
// u3-conversion-op-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 1+2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-op-suma.c
$ ./a.out
```

Función printf

Los valores que usan los especificadores pueden ser el resultado de operaciones

```c
#include <stdio.h>
// u3-conversion-op-suma.c

int main (void)
{
  printf("%d+%d=%d\n", 1, 2, 1+2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-op-suma.c
$ ./a.out
1+2=3
$
```

Función printf

Función printf

```c
#include <stdio.h>
// u3-conversion-div-mod.c

int main (void)
{
  printf("%d\n", 5%2);
  printf("%d\n", 5/2);

  return 0;
}
```

Función printf

```c
#include <stdio.h>
// u3-conversion-div-mod.c

int main (void)
{
  printf("%d\n", 5%2);
  printf("%d\n", 5/2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-div-mod.c
$
```

Función printf

```c
#include <stdio.h>
// u3-conversion-div-mod.c

int main (void)
{
  printf("%d\n", 5%2);
  printf("%d\n", 5/2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-div-mod.c
$ ./a.out
```

Función printf

```c
#include <stdio.h>
// u3-conversion-div-mod.c

int main (void)
{
  printf("%d\n", 5%2);
  printf("%d\n", 5/2);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-conversion-div-mod.c
$ ./a.out
1
2
$
```