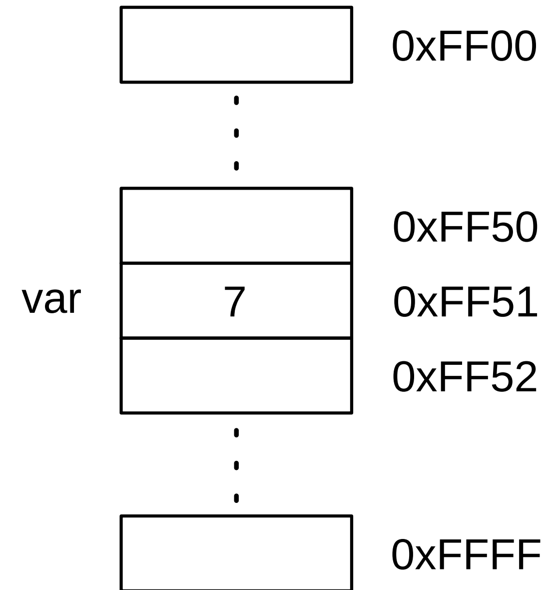
Variables

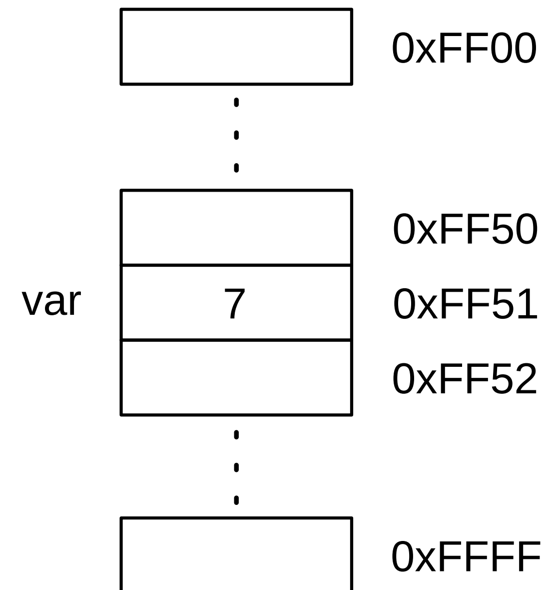
Variables



Variables

Todas las variables tienen:

- Nombre o identificador
- Valor almacenado
- Dirección de memoria
- Tipo



Tipos de datos, tamaño

Tipos de datos, tamaño

Tipo	Menor	Mayor	Bytes
char	-128	127	1
unsigned char	0	255	1
short	-32768	32767	2
unsigned short	0	65535	2
int	-2147483648	2147483647	4
unsigned int	0	4294967295	4
long	-9223372036854775808	9223372036854775807	8
unsigned long	0	18446744073709551615	8
float	1.18e-38	3.4e+38	4
double	2.23e-308	1.8e+308	8
long double	3.36e-4932	1.19e+4932	10

Definición de variables

Definición de variables

Cuando se **define** una variable se reserva una posición de memoria para poder almacenar su contenido.

Definición de variables

Cuando se **define** una variable se reserva una posición de memoria para poder almacenar su contenido.

En la definición se debe explicitar el **tipo** que corresponderá a esa variable.

Definición de variables

Cuando se **define** una variable se reserva una posición de memoria para poder almacenar su contenido.

En la definición se debe explicitar el **tipo** que corresponderá a esa variable.

De esta manera el compilador sabe que tan grande es el espacio de memoria que debe reservar.

Definición de variables

Cuando se **define** una variable se reserva una posición de memoria para poder almacenar su contenido.

En la definición se debe explicitar el **tipo** que corresponderá a esa variable.

De esta manera el compilador sabe que tan grande es el espacio de memoria que debe reservar.

También hay que asignar un **nombre**, con el cual se puede acceder al contenido de la memoria.

Definición de variables

Definición de variables

Como?

Definición de variables

Como?

```
tipo identificador;
```

Definición de variables

Como?

```
tipo identificador;
```

Donde `tipo` puede ser `int, char, float,` etc.

Definición de variables

Como?

`tipo identificador;`

Donde `tipo` puede ser `int, char, float,` etc.

y donde `identificador` puede ser cualquier identificador válido

Definición de variables

Como?

`tipo identificador;`

Donde `tipo` puede ser `int, char, float,` etc.

y donde `identificador` puede ser cualquier identificador válido

```
int entero;
char caracter;
int n1, n2;
float max_temp;
```

Definición de variables

Definición de variables

Donde?

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Por ahora entre las llaves del cuerpo de `main`

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Donde?

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

int main (void)
{
  int la_respuesta;

  la_respuesta = 42;
  printf("%d\n", la_respuesta);

  return 0;
}
```

Cuando se **asigna** un valor a una variable se dice que se **inicializa**

Definición de variables

Definición de variables

Se puede **inicializar** una variable en el mismo momento que se **define**

Definición de variables

Se puede **inicializar** una variable en el mismo momento que se **define**

```
#include <stdio.h>
// u3-definicion-tipos-ini.c

int main (void)
{
  int la_respuesta = 42;

  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Se puede **inicializar** una variable en el mismo momento que se **define**

```c
#include <stdio.h>
// u3-definicion-tipos-ini.c

int main (void)
{
  int la_respuesta = 42;

  printf("%d\n", la_respuesta);

  return 0;
}
```

Definición de variables

Definición de variables

Se pueden definir varias variables del mismo tipo en la misma sentencia.

Definición de variables

Se pueden definir varias variables del mismo tipo en la misma sentencia.

```
char var1, var2, var3;
```

Definición de variables

Se pueden definir varias variables del mismo tipo en la misma sentencia.

```
char var1, var2, var3;
```

Se pueden inicializar varias variables en la misma sentencia.

Definición de variables

Se pueden definir varias variables del mismo tipo en la misma sentencia.

```
char var1, var2, var3;
```

Se pueden inicializar varias variables en la misma sentencia.

```
int a=3, b, c=0;
```

Definición de variables

Se pueden definir varias variables del mismo tipo en la misma sentencia.

```
char var1, var2, var3;
```

Se pueden inicializar varias variables en la misma sentencia.

```
int a=3, b, c=0;
```

Las variables no inicializadas pueden tener cualquier valor.

Definición de variables

Definición de variables

```c
#include <stdio.h>
// u3-def-sin-ini.c

int main (void)
{
  int a=3, b, c=0;

  printf("%d %d %d\n", a, b, c);

  return 0;
}
```

Definición de variables

```c
#include <stdio.h>
// u3-def-sin-ini.c

int main (void)
{
  int a=3, b, c=0;

  printf("%d %d %d\n", a, b, c);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-def-sin-ini.c
```

Definición de variables

```c
#include <stdio.h>
// u3-def-sin-ini.c

int main (void)
{
  int a=3, b, c=0;

  printf("%d %d %d\n", a, b, c);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-def-sin-ini.c
u3-def-sin-ini.c: In function ‘main’:
u3-def-sin-ini.c:8:3: warning: ‘b’ is used uninitialized
                               in this function [-Wuninitialized]
   printf("%d %d %d\n", a, b, c);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$
```

Definición de variables

```c
#include <stdio.h>
// u3-def-sin-ini.c

int main (void)
{
  int a=3, b, c=0;

  printf("%d %d %d\n", a, b, c);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-def-sin-ini.c
u3-def-sin-ini.c: In function ‘main’:
u3-def-sin-ini.c:8:3: warning: ‘b’ is used uninitialized
                               in this function [-Wuninitialized]
   printf("%d %d %d\n", a, b, c);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$./a.out
```

Definición de variables

```
#include <stdio.h>
// u3-def-sin-ini.c

int main (void)
{
  int a=3, b, c=0;

  printf("%d %d %d\n", a, b, c);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-def-sin-ini.c
u3-def-sin-ini.c: In function ‘main’:
u3-def-sin-ini.c:8:3: warning: ‘b’ is used uninitialized
                               in this function [-Wuninitialized]
   printf("%d %d %d\n", a, b, c);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$./a.out
3 32767 0
$
```

Definición de variables

Definición de variables

Si se ejecuta nuevamente `a.out` la variable b puede tener cualquier valor.

Definición de variables

Si se ejecuta nuevamente `a.out` la variable b puede tener cualquier valor.

Posiblemente diferente cada vez que se ejecute.

Definición de variables

Si se ejecuta nuevamente `a.out` la variable b puede tener cualquier valor.

Posiblemente diferente cada vez que se ejecute.

Dependerá de donde sea alojado `a.out` y la *basura* que haya quedado en esa posición.

Definición de variables

Si se ejecuta nuevamente `a.out` la variable b puede tener cualquier valor.

Posiblemente diferente cada vez que se ejecute.

Dependerá de donde sea alojado `a.out` y la *basura* que haya quedado en esa posición.

Para evitar errores inesperados **debe** inicializarse cada variable.

Definición de variables

Si se ejecuta nuevamente `a.out` la variable b puede tener cualquier valor.

Posiblemente diferente cada vez que se ejecute.

Dependerá de donde sea alojado `a.out` y la *basura* que haya quedado en esa posición.

Para evitar errores inesperados **debe** inicializarse cada variable.

Prestar atención a los mensajes del compilador.

Operadores

Operadores

En general, los **operadores** son símbolos que indican que debe realizarse una operación sobre algún conjunto de objetos.

Operadores

En general, los **operadores** son símbolos que indican que debe realizarse una operación sobre algún conjunto de objetos.

Los objetos sobre los que opera un operador se llaman **operandos**.

Operadores

En general, los **operadores** son símbolos que indican que debe realizarse una operación sobre algún conjunto de objetos.

Los objetos sobre los que opera un operador se llaman **operandos**.

Según a cuantos operandos afecten, los operadores pueden ser **unarios**, **binarios** o **ternarios**.

Operadores

En general, los **operadores** son símbolos que indican que debe realizarse una operación sobre algún conjunto de objetos.

Los objetos sobre los que opera un operador se llaman **operandos**.

Según a cuantos operandos afecten, los operadores pueden ser **unarios**, **binarios** o **ternarios**.

Los operadores siempre **devuelven** el resultado de la operación.

Operadores aritméticos

Operadores aritméticos

Unarios

$+$	Signo positivo
$-$	Signo negativo

Operadores aritméticos

Unarios

+	Signo positivo
−	Signo negativo

Binarios

+	Suma
−	Resta
*	Producto
/	División
%	Módulo
=	Asignación

Operadores aritméticos

Operadores aritméticos

El resultado de la operación se puede imprimir

```
#include <stdio.h>
// u3-intro-operadores-1.c

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

  return 0;
}
```

Operadores aritméticos

El resultado de la operación se puede imprimir

```
#include <stdio.h>
// u3-intro-operadores-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-1.c
$
```

Operadores aritméticos

El resultado de la operación se puede imprimir

```c
#include <stdio.h>
// u3-intro-operadores-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-1.c
$ ./a.out
```

Operadores aritméticos

El resultado de la operación se puede imprimir

```c
#include <stdio.h>
// u3-intro-operadores-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-1.c
$ ./a.out
3
$
```

Operadores aritméticos

Operadores aritméticos

El resultado de la operación se puede **asignar**

```
#include <stdio.h>
// u3-intro-operadores-2.c

int main (void)
{
  int resultado;

  resultado = 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

Operadores aritméticos

El resultado de la operación se puede **asignar**

```c
#include <stdio.h>
// u3-intro-operadores-2.c

int main (void)
{
  int resultado;

  resultado = 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-2.c
$
```

Operadores aritméticos

El resultado de la operación se puede **asignar**

```c
#include <stdio.h>
// u3-intro-operadores-2.c

int main (void)
{
  int resultado;

  resultado = 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-2.c
$ ./a.out
```

Operadores aritméticos

El resultado de la operación se puede **asignar**

```c
#include <stdio.h>
// u3-intro-operadores-2.c

int main (void)
{
  int resultado;

  resultado = 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-2.c
$ ./a.out
3
$
```

Operadores aritméticos

Operadores aritméticos

```
#include <stdio.h>
// u3-intro-operadores-3.c

int main (void)
{
  int resultado;

  resultado = 3 * 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

Operadores aritméticos

```c
#include <stdio.h>
// u3-intro-operadores-3.c

int main (void)
{
  int resultado;

  resultado = 3 * 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-3.c
$
```

Operadores aritméticos

```c
#include <stdio.h>
// u3-intro-operadores-3.c

int main (void)
{
  int resultado;

  resultado = 3 * 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-3.c
$ ./a.out
```

Operadores aritméticos

```c
#include <stdio.h>
// u3-intro-operadores-3.c

int main (void)
{
  int resultado;

  resultado = 3 * 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-3.c
$ ./a.out
7
$
```

Operadores aritméticos

```c
#include <stdio.h>
// u3-intro-operadores-3.c

int main (void)
{
  int resultado;

  resultado = 3 * 2 + 1;

  printf("%d\n", resultado);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-intro-operadores-3.c
$ ./a.out
7
$
```

El orden en el que se resuelven las operaciones depende de las reglas de precedencia

Precedencia

Precedencia

Se llama precedencia al orden en el que se evalúan las operaciones en una expresión

Precedencia

Se llama precedencia al orden en el que se evalúan las operaciones en una expresión

Mientras más arriba en la tabla, se dice que tiene más precedencia, y se evalúa primero

Precedencia

Precedencia

Hasta ahora...

Operador	Asociatividad
()	Izq. a Der.
+ - (los de signo)	Der. a Izq.
* / %	Izq. a Der.
+ -	Izq. a Der.
=	Der. a Izq.

Precedencia

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
```

①

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
                   ①       ②
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
                ③   ①       ②
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
                 ③    ①   ④   ②
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
          ⑤   ③      ①   ④   ②
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
          ⑤   ③    ①   ④   ②

resultado = 4 +    7    - 5 * 2;
          ⑤   ③         ④   ②
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
            ⑤   ③    ①   ④   ②

resultado = 4 +    7    - 5 * 2;
            ⑤   ③         ④   ②

resultado = 4 +    7    -   10 ;
            ⑤   ③         ④
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
            ⑤   ③    ①   ④   ②

resultado = 4 +    7    - 5 * 2;
            ⑤   ③         ④   ②

resultado = 4 +    7    -   10 ;
            ⑤   ③         ④

resultado =      11     -   10 ;
            ⑤             ④
```

Precedencia

```
resultado = 4 + 21 / 3 - 5 * 2;
            ⑤   ③    ①   ④   ②

resultado = 4 +    7   - 5 * 2;
            ⑤   ③        ④   ②

resultado = 4 +    7   -   10 ;
            ⑤   ③        ④

resultado =     11     -   10 ;
            ⑤            ④

resultado =        1          ;
            ⑤
```

Operadores relacionales

Operadores relacionales

Los operadores relacionales sirven para comparar constantes o variables.

Operadores relacionales

Los operadores relacionales sirven para comparar constantes o variables.

En Mat.	En C	Descripción
$>$	$>$	Mayor
$<$	$<$	Menor
$\geq$	$>=$	Mayor o igual
$\leq$	$<=$	Menor o igual
$=$	$==$	Igual
$\neq$	$!=$	Distinto

Operadores relacionales

Los operadores relacionales sirven para comparar constantes o variables.

En Mat.	En C	Descripción
$>$	$>$	Mayor
$<$	$<$	Menor
$\geq$	$>=$	Mayor o igual
$\leq$	$<=$	Menor o igual
$=$	$==$	Igual
$\neq$	$!=$	Distinto

Como todos los operadores, devuelve el resultado de la operación

Operadores relacionales

Operadores relacionales

Si la relación se cumple, devuelve un 1

Operadores relacionales

Si la relación se cumple, devuelve un 1

```c
#include <stdio.h>
// u3-relacion-1.c

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

  return 0;
}
```

Operadores relacionales

Si la relación se cumple, devuelve un 1

```c
#include <stdio.h>
// u3-relacion-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-1.c
$
```

Operadores relacionales

Si la relación se cumple, devuelve un 1

```c
#include <stdio.h>
// u3-relacion-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-1.c
$ ./a.out
```

Operadores relacionales

Si la relación se cumple, devuelve un 1

```c
#include <stdio.h>
// u3-relacion-1.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-1.c
$ ./a.out
1
$
```

Operadores relacionales

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

```
#include <stdio.h>
// u3-relacion-2.c

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

  return 0;
}
```

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

```c
#include <stdio.h>
// u3-relacion-2.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$
```

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

```
#include <stdio.h>
// u3-relacion-2.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$ ./a.out
```

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

```
#include <stdio.h>
// u3-relacion-2.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$ ./a.out
0
$
```

Operadores relacionales

Si la relación **no** se cumple, devuelve un `0`

```
#include <stdio.h>
// u3-relacion-2.c

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

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$ ./a.out
0
$
```

Estos operadores son utilizados en los condicionales.

Funciones de entrada/salida (continúa)

Funciones de entrada/salida (continúa)

Cuando se necesita que el usuario ingrese algún valor se puede usar la función `scanf`

Funciones de entrada/salida (continúa)

Cuando se necesita que el usuario ingrese algún valor se puede usar la función `scanf`

`scanf` es otra de las funciones de la **biblioteca estándar**

Función scanf

Función scanf

`scanf` espera *al menos* dos argumentos.

Función scanf

`scanf` espera *al menos* dos argumentos.

```
scanf("%d", &variable);
```

Función scanf

`scanf` espera *al menos* dos argumentos.

```
scanf("%d", &variable);
```

El primero es una cadena de texto con especificadores de formato semejantes a `printf`.

Función scanf

`scanf` espera *al menos* dos argumentos.

```
scanf("%d", &variable);
```

El primero es una cadena de texto con especificadores de formato semejantes a `printf`.

Luego espera tantos argumentos como especificadores de formato tenga la cadena.

Función scanf

`scanf` espera *al menos* dos argumentos.

```
scanf("%d", &variable);
```

El primero es una cadena de texto con especificadores de formato semejantes a `printf`.

Luego espera tantos argumentos como especificadores de formato tenga la cadena.

Por ahora, estos argumentos son las variables (con un & delante) donde se guardarán los valores ingresados por el usuario.

Función scanf

Función scanf

```
#include <stdio.h>
// u3-entrada-1.c

int main (void)
{
  int sum1, sum2;
  int res;

  printf("Ingrese un número: ");
  scanf("%d", &sum1);
  printf("Ingrese otro número: ");
  scanf("%d", &sum2);

  res = sum1 + sum2;

  printf("%d+%d=%d\n", sum1, sum2, res);

  return 0;
}
```

Función scanf

Función scanf

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$ ./a.out
Ingrese un número: 12
Ingrese otro número: 13
12+13=25
$
```

Función scanf

```
$ gcc -Wall -std=c99 -pedantic-errors u3-relacion-2.c
$ ./a.out
Ingrese un número: 12
Ingrese otro número: 13
12+13=25
$
```

Si una variable va a ser utilizada por primera vez en un `scanf` no hace falta inicializarla

Función scanf

Función scanf

Al igual que `printf` tiene distintos especificadores de formato

Función scanf

Al igual que `printf` tiene distintos especificadores de formato

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