Informática I

Claudio Paz

claudiojpaz@gmail.com

Abril 2019

Unidad 3

Introducción al lenguaje C

Reseña histórica

DEC PDP-7. Oslo, Noruega

Toresbe [CC SA 1.0], via Wikimedia Commons

Reseña histórica

Fines de los 60's y principios de los 70's

- 1966 - BCPL - Martin Richards.
- 1969 - B - Ken Thompson con Dennis Ritchie.
- 1969-1973 - C - Dennis Ritchie.

Reseña histórica

Fines de los 70's hasta fines de los 90's.. y más

- 1978 - Libro: Lenguaje de Programación C
  Kernighan & Ritchie
- 1989 - American National Standards Institute - ANSI C - C89
- 1990 - International Organization for Standardization - ISO C - C90
- 1999 - ANSI adopta el estándar ISO para C - C99
- 2011 - ISO C - C11
- 2018 - ISO C - C18

Elementos del lenguaje C

Elementos del lenguaje C

Token: conjunto de símbolos que tiene un significado coherente en un lenguaje de programación

Elementos del lenguaje C

Token: conjunto de símbolos que tiene un significado coherente en un lenguaje de programación

Existen seis clases de tokens en el vocabulario del lenguaje C

Elementos del lenguaje C

Token: conjunto de símbolos que tiene un significado coherente en un lenguaje de programación

Existen seis clases de tokens en el vocabulario del lenguaje C

Palabras clave	Identificadores
Constantes	Comentarios
Operadores	Separadores

Elementos del lenguaje C

Elementos del lenguaje C

Palabras clave

Elementos del lenguaje C

Palabras clave

32 palabras reservadas para el lenguaje

Elementos del lenguaje C

Palabras clave

Elementos del lenguaje C

Palabras clave

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

Elementos del lenguaje C

Elementos del lenguaje C

Identificadores

Elementos del lenguaje C

Identificadores

Conjunto de caracteres alfanuméricos que asocian a entidades del programa (variables, funciones, etc.).

Elementos del lenguaje C

Identificadores

Conjunto de caracteres alfanuméricos que asocian a entidades del programa (variables, funciones, etc.).

No pueden ser iguales a ninguna palabra reservada.

Elementos del lenguaje C

Identificadores

Conjunto de caracteres alfanuméricos que asocian a entidades del programa (variables, funciones, etc.).

No pueden ser iguales a ninguna palabra reservada.

Pueden estar formados por letras del alfabeto inglés, números o guiones bajos (_).

Elementos del lenguaje C

Identificadores

Conjunto de caracteres alfanuméricos que asocian a entidades del programa (variables, funciones, etc.).

No pueden ser iguales a ninguna palabra reservada.

Pueden estar formados por letras del alfabeto inglés, números o guiones bajos (_).

No pueden comenzar con números.

Elementos del lenguaje C

Identificadores

Conjunto de caracteres alfanuméricos que asocian a entidades del programa (variables, funciones, etc.).

No pueden ser iguales a ninguna palabra reservada.

Pueden estar formados por letras del alfabeto inglés, números o guiones bajos (_).

No pueden comenzar con números.

No pueden tener espacios ni operadores aritméticos.

Elementos del lenguaje C

Identificadores

Elementos del lenguaje C

Identificadores

Correcto

```
var
n1
_node05
contador
max_temp
i
```

Elementos del lenguaje C

Identificadores

Correcto

```
var
n1
_node05
contador
max_temp
i
```

Incorrecto

```
3var
if
5
node-05
max temp
```

Elementos del lenguaje C

Elementos del lenguaje C

Constantes

Elementos del lenguaje C

Constantes

Son valores que no pueden cambiar una vez que el programa fue compilado.

Elementos del lenguaje C

Constantes

Son valores que no pueden cambiar una vez que el programa fue compilado.

También se llaman literales.

Elementos del lenguaje C

Constantes

Son valores que no pueden cambiar una vez que el programa fue compilado.

También se llaman literales.

Pueden ser números, caracteres individuales o cadenas de texto.

Elementos del lenguaje C

Constantes

Elementos del lenguaje C

Constantes

Números

```
3   3.1416   31.416e-1   0.31416e1   .31416e1   .31416e+1
```

Elementos del lenguaje C

Constantes

Números

```
3  3.1416  31.416e-1  0.31416e1  .31416e1  .31416e+1

3.1416f  3.1416F  31.416E-1  0xfe01  0xFE01  0XFE01
```

Elementos del lenguaje C

Constantes

Números

```
3  3.1416  31.416e-1  0.31416e1  .31416e1  .31416e+1

3.1416f  3.1416F  31.416E-1  0xfe01  0xFE01  0XFE01
```

Caracteres

Elementos del lenguaje C

Constantes

Números

```
3   3.1416   31.416e-1   0.31416e1   .31416e1   .31416e+1

3.1416f   3.1416F   31.416E-1   0xfe01   0xFE01   0XFE01
```

Caracteres

```
'c'    'F'    '\n'    '7'
```

Elementos del lenguaje C

Constantes

Números

```
3   3.1416   31.416e-1   0.31416e1   .31416e1   .31416e+1

3.1416f   3.1416F   31.416E-1   0xfe01   0xFE01   0XFE01
```

Caracteres

```
'c'    'F'    '\n'    '7'
```

Cadenas de caracteres

Elementos del lenguaje C

Constantes

Números

```
3   3.1416   31.416e-1   0.31416e1   .31416e1   .31416e+1

3.1416f   3.1416F   31.416E-1   0xfe01   0xFE01   0XFE01
```

Caracteres

```
'c'    'F'    '\n'    '7'
```

Cadenas de caracteres

```
"Hola, mundo!"
```

Elementos del lenguaje C

Elementos del lenguaje C

Operadores

Elementos del lenguaje C

Operadores

Los operadores en C son conjuntos de caracteres (uno o dos) que indican al programa que debe hacer.

Elementos del lenguaje C

Operadores

Los operadores en C son conjuntos de caracteres (uno o dos) que indican al programa que debe hacer.

Aritméticos: +, -, *, / y %

Asignación: =, +=, -=, *= y /=

Incrementales: ++ y --

Relacionales: <, >, >=, <=, == y !=

Lógicos: && y ||

Elementos del lenguaje C

Elementos del lenguaje C

Separadores o delimitadores

Elementos del lenguaje C

Separadores o delimitadores

Se utilizan en distintas construcciones del lenguaje

Elementos del lenguaje C

Separadores o delimitadores

Se utilizan en distintas construcciones del lenguaje

[ ] ( ) {} , ; : ... * = #

Elementos del lenguaje C

Elementos del lenguaje C

Comentarios

Elementos del lenguaje C

Comentarios

Cadenas de texto que no se compilan.

Elementos del lenguaje C

Comentarios

Cadenas de texto que no se compilan.

Sirven para que el programador deje aclaraciones.

Elementos del lenguaje C

Comentarios

Cadenas de texto que no se compilan.

Sirven para que el programador deje aclaraciones.

O para anular partes del código.

Elementos del lenguaje C

Comentarios

Cadenas de texto que no se compilan.

Sirven para que el programador deje aclaraciones.

O para anular partes del código.

La doble barra (//) anula todo hasta el final de la línea.

Elementos del lenguaje C

Comentarios

Cadenas de texto que no se compilan.

Sirven para que el programador deje aclaraciones.

O para anular partes del código.

La doble barra (//) anula todo hasta el final de la línea.

/* comenta todo hasta encontrar un */ aunque sea muchas líneas más abajo

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Todos los programas deben tener una función `main`

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Las llaves definen *bloques*.

En este caso el bloque es el *cuerpo* de la función `main`

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

A partir del estándar C90 en adelante, la función `main` debe terminar con un `return`.

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Indica al *proceso* que *lanzó* el programa, que el mismo ya terminó.

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Las líneas que comienzan con # se llaman *directivas de preprocesador*.

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Son necesarias para que el compilador trabaje de forma correcta.

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Otras son:

`#define` `#ifdef`

`#pragma`

etc.

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

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

  return 0;
}
```

En este caso `printf` y `return 0` son sentencias simples...

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

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

  return 0;
}
```

Sentencias simples deben terminar con punto y coma (;)

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

`printf` es una función definida en `stdio.h`...

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

Primer ejemplo: Hola, Mundo!

```c
#include <stdio.h>

// programa de prueba

int main (void)
{

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

  return 0;
}
```

...utilizada para imprimir mensajes en pantalla.

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` Warning **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
```

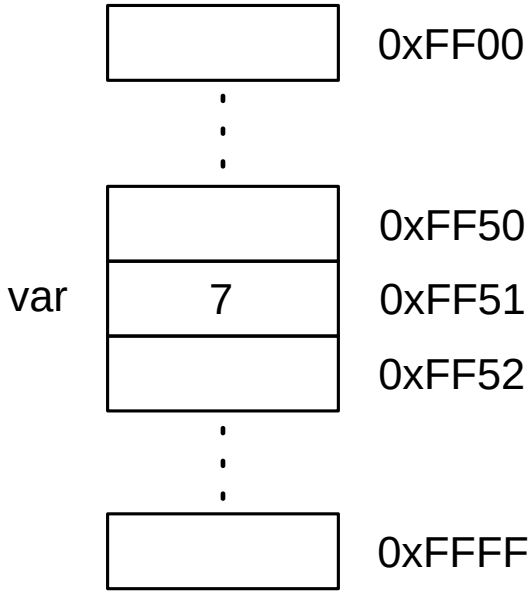
Función printf

```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
$
```

Variables

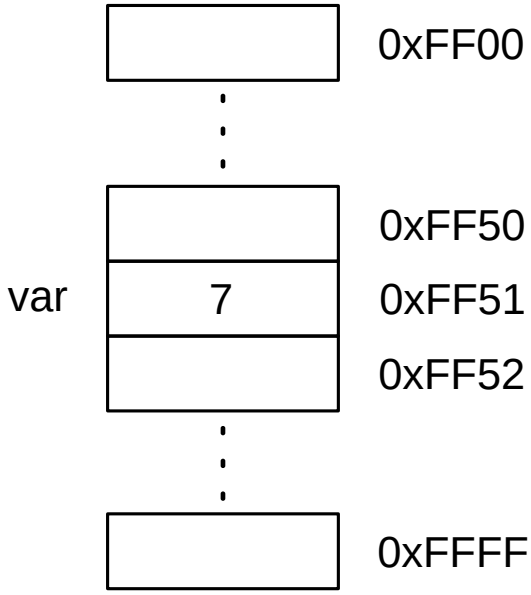
Variables



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	$1.18e\text{-}38$	$3.4e{+}38$	4
double	$2.23e\text{-}308$	$1.8e{+}308$	8
long double	$3.36e\text{-}4932$	$1.19e{+}4932$	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**

```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

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

```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
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

```c
#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

```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
$
```

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**

```c
#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

```c
#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`

```c
#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`

```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
$ ./a.out
```

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
$ ./a.out
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
$ ./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

```c
#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

Función scanf

Función scanf

También se pueden ingresar más valores por sentencia

Función scanf

También se pueden ingresar más valores por sentencia

```
scanf("%d %d", &var1, &var2);
```

Función scanf

También se pueden ingresar más valores por sentencia

```
scanf("%d %d", &var1, &var2);
```

donde para diferenciar los valores desde el teclado se ingresan con un espacio, un tab o un enter entre ellos.

Función putchar

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

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

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

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

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-putchar.c
$ ./a.out
Ingrese un número (1-127):
```

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-putchar.c
$ ./a.out
Ingrese un número (1-127): 65
```

Función putchar

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

int main (void)
{
  int numero;

  printf("Ingrese un número (1-127): ");
  scanf("%d", &numero);

  printf("En la tabla ASCII: ");
  putchar(numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-putchar.c
$ ./a.out
Ingrese un número (1-127): 65
En la tabla ASCII: A
$
```

Función putchar

Función putchar

La función `putchar` tiene el mismo efecto que `printf` si solo se imprime un caracter con `"%c"`

Función putchar

La función `putchar` tiene el mismo efecto que `printf` si solo se imprime un caracter con `"%c"`

La sentencia

Función putchar

La función `putchar` tiene el mismo efecto que `printf` si solo se imprime un caracter con "%c"

La sentencia

```
putchar(65);
```

Función putchar

La función `putchar` tiene el mismo efecto que `printf` si solo se imprime un caracter con "%c"

La sentencia

```
putchar(65);
```

Tiene el mismo efecto que la sentencia

Función putchar

La función `putchar` tiene el mismo efecto que `printf` si solo se imprime un caracter con "%c"

La sentencia

```
putchar(65);
```

Tiene el mismo efecto que la sentencia

```
printf("%c", 65);
```

Función putchar

Función putchar

Además de la diferente complejidad de las sentencias, la diferencia fundamental reside en el valor devuelto.

Función putchar

Además de la diferente complejidad de las sentencias, la diferencia fundamental reside en el valor devuelto.

La función `putchar` devuelve el valor entero del carater impreso.

Función putchar

Además de la diferente complejidad de las sentencias, la diferencia fundamental reside en el valor devuelto.

La función `putchar` devuelve el valor entero del carater impreso.

La función `printf` devuelve la cantidad de caracteres impresos.

Función getchar

Función getchar

Se puede ingresar cualquier caracter desde el teclado utilizando la función `getchar`

Función getchar

Se puede ingresar cualquier caracter desde el teclado utilizando la función `getchar`

`getchar` devuelve un entero correspondiente al caracter ingresado

Función getchar

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

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

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

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

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-getchar.c
$ ./a.out
Ingrese un caracter de la tabla ASCII:
```

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-getchar.c
$ ./a.out
Ingrese un caracter de la tabla ASCII: A
```

Función getchar

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

int main (void)
{
  int numero;

  printf("Ingrese un caracter de la tabla ASCII: ");
  numero = getchar();

  printf("En la tabla ASCII es el %d\n", numero);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors u3-getchar.c
$ ./a.out
Ingrese un caracter de la tabla ASCII: A
En la tabla ASCII es el 65
$
```

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

En ocaciones se necesita obtener resultados de un tipo de datos a partir de variables de tipos diferentes.

Operador de Conversión de tipo (cast)

En ocaciones se necesita obtener resultados de un tipo de datos a partir de variables de tipos diferentes.

Por ejemplo un promedio (punto flotante) a partir calificaciones (enteras)

Operador de Conversión de tipo (cast)

En ocaciones se necesita obtener resultados de un tipo de datos a partir de variables de tipos diferentes.

Por ejemplo un promedio (punto flotante) a partir calificaciones (enteras)

```
promedio = suma_notas / cuantas_notas;
```

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

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

int main (void)
{
  int suma_notas, cuantas_notas;
  float promedio;

  printf("Ingrese la suma de todas las notas: ");
  scanf("%d", &suma_notas);
  printf("Ingrese cuantas notas son: ");
  scanf("%d", &cuantas_notas);

  promedio = suma_notas / cuantas_notas;

  printf("Promedio: %.2f\n", promedio);

  return 0;
}
```

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

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

Operador de Conversión de tipo (cast)

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

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas:
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son:
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
Promedio: 6.00
$
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
Promedio: 6.00
$
```

El problema es que cuando la división es entre dos enteros, se realiza de la manera básica, devolviendo un entero y dejando resto...

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
Promedio: 6.00
$
```

El problema es que cuando la división es entre dos enteros, se realiza de la manera básica, devolviendo un entero y dejando resto...

...asignando un valor entero a la variable `promedio` (aunque quede almacenado como de punto flotante)

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-sin-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
Promedio: 6.00
$
```

El problema es que cuando la división es entre dos enteros, se realiza de la manera básica, devolviendo un entero y dejando resto...

...asignando un valor entero a la variable `promedio` (aunque quede almacenado como de punto flotante)

Para esto se usa el operador de conversión de tipo

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

El operador de conversión de tipo o simplemente *cast* es un operador **unario** que cambia temporalmente el tipo de datos de su operando.

Operador de Conversión de tipo (cast)

El operador de conversión de tipo o simplemente *cast* es un operador **unario** que cambia temporalmente el tipo de datos de su operando.

Consiste en colocar entre parentesis el tipo de datos al que se quiere *convertir* el operando, delante del mismo.

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

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

int main (void)
{
  int suma_notas, cuantas_notas;
  float promedio;

  printf("Ingrese la suma de todas las notas: ");
  scanf("%d", &suma_notas);
  printf("Ingrese cuantas notas son: ");
  scanf("%d", &cuantas_notas);

  promedio = (float) suma_notas / cuantas_notas;

  printf("Promedio: %.2f\n", promedio);

  return 0;
}
```

Operador de Conversión de tipo (cast)

Operador de Conversión de tipo (cast)

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

Operador de Conversión de tipo (cast)

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

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-con-cast.c
$ ./a.out
Ingrese la suma de todas las notas:
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-con-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-con-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son:
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-con-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
```

Operador de Conversión de tipo (cast)

```
$ gcc -Wall -std=c99 -pedantic-errors u3-con-cast.c
$ ./a.out
Ingrese la suma de todas las notas: 13
Ingrese cuantas notas son: 2
Promedio: 6.50
$
```

Precedencia (actualizada)

Precedencia (actualizada)

Operador	Asociatividad
$()$	Izq. a Der.
$+ \quad - \quad (\text{tipo})$	Der. a Izq.
$* \quad / \quad \%$	Izq. a Der.
$+ \quad -$	Izq. a Der.
$< \quad <= \quad > \quad >=$	Izq. a Der.
$== \quad !=$	Izq. a Der.
$=$	Der. a Izq.