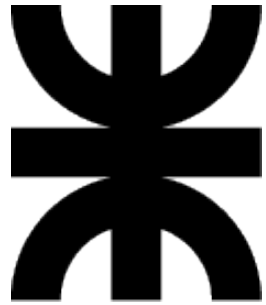
Informática II

El compilador de C del proyecto GNU (`gcc, g++`)

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.
- Puede compilar C++

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.
- Puede compilar C++
- Realiza la optimización del código (flags `-O0`, `-O1`, `-O2`, `-O3`, `-Os`)

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.
- Puede compilar C++
- Realiza la optimización del código (flags `-O0`, `-O1`, `-O2`, `-O3`, `-Os`)
- Genera información de depuración/debugging (flag `-g`)

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.
- Puede compilar C++
- Realiza la optimización del código (flags `-O0`, `-O1`, `-O2`, `-O3`, `-Os`)
- Genera información de depuración/debugging (flag `-g`)
- Es un compilador cruzado (cross-compiler)

El compilador de C del proyecto GNU

Herramientas de compilación del proyecto GNU (GNU Compiler Collection):

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), extensiones GNU, etc.
- Puede compilar C++
- Realiza la optimización del código (flags `-O0`, `-O1`, `-O2`, `-O3`, `-Os`)
- Genera información de depuración/debugging (flag `-g`)
- Es un compilador cruzado (cross-compiler)

  Ejemplos:

```
$> gcc --version
$> gcc -v
```

```
$> avr-gcc --version
$> avr-gcc -v
```

  (ver `--build`, `--host`, `--target`)

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

hola.h (incompleto!)

```c
void hola(const char * nombre);
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

hola.h (incompleto!)

```c
void hola(const char * nombre);
```

Compilación

```
$> gcc -Wall main.c hola.c -o hola
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

hola.h (incompleto!)

```c
void hola(const char * nombre);
```

Compilación

```
$> gcc -Wall main.c hola.c -o hola
```

Se puede compilar separadamente cada archivo fuente

```
$> gcc -Wall -c main.c
$> gcc -Wall -c hola.c
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

hola.h (incompleto!)

```c
void hola(const char * nombre);
```

Compilación

```
$> gcc -Wall main.c hola.c -o hola
```

Se puede compilar separadamente cada archivo fuente

```
$> gcc -Wall -c main.c
$> gcc -Wall -c hola.c
```

y unirlo con el linker (enlazador)

```
$> gcc main.o hola.o -o hola
```

Compilando – Ejemplos con varios archivos fuentes

main.c

```c
#include "hola.h"

int main(void)
{
  hola("mundo");
  return 0;
}
```

hola.c

```c
#include <stdio.h>
#include "hola.h"

void hola(const char * nombre)
{
  printf("Hola, %s!\n", nombre);
}
```

hola.h (incompleto!)

```c
void hola(const char * nombre);
```

Compilación

```
$> gcc -Wall main.c hola.c -o hola
```

Se puede compilar separadamente cada archivo fuente

```
$> gcc -Wall -c main.c
$> gcc -Wall -c hola.c
```

y unirlo con el linker (enlazador)

```
$> gcc main.o hola.o -o hola
```

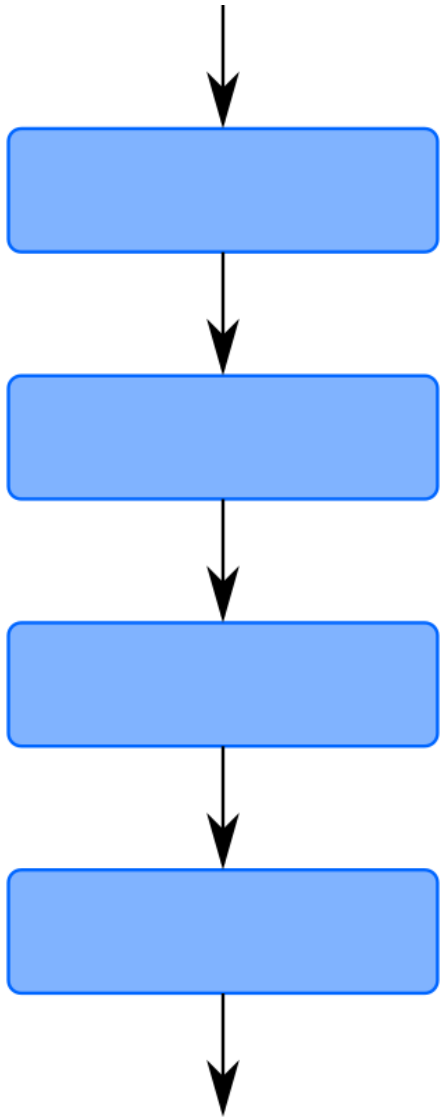
Esto permite modifica un archivo fuente y recompilar solo ese archivo.

Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).

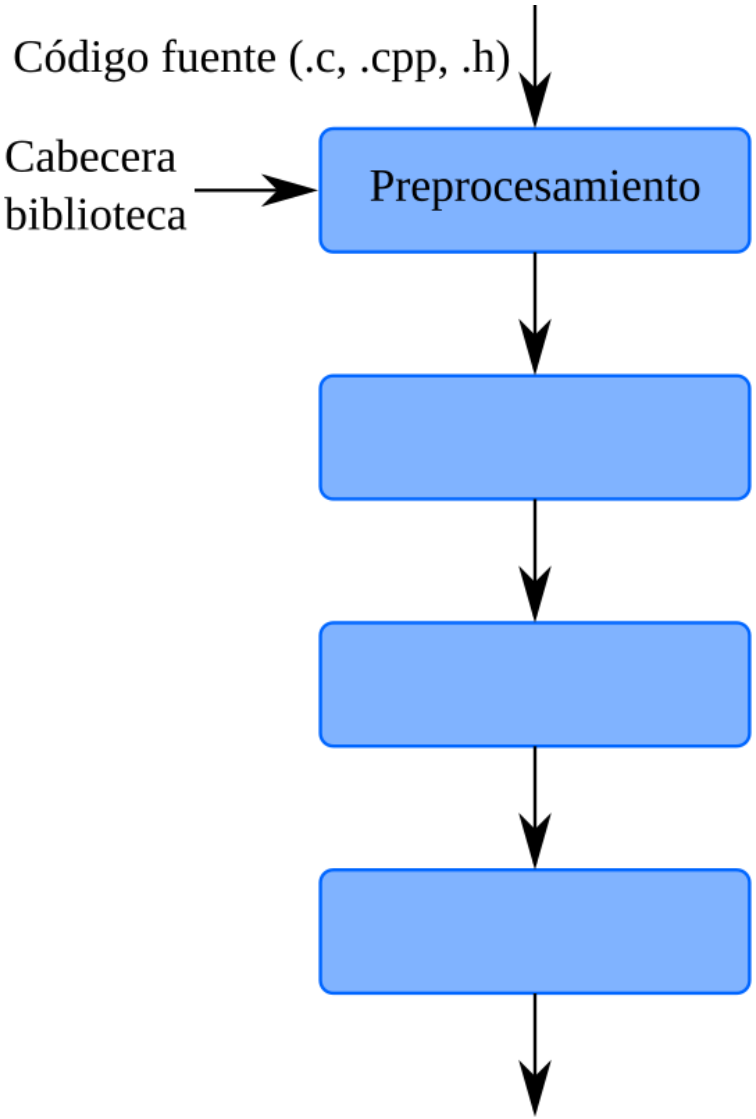
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



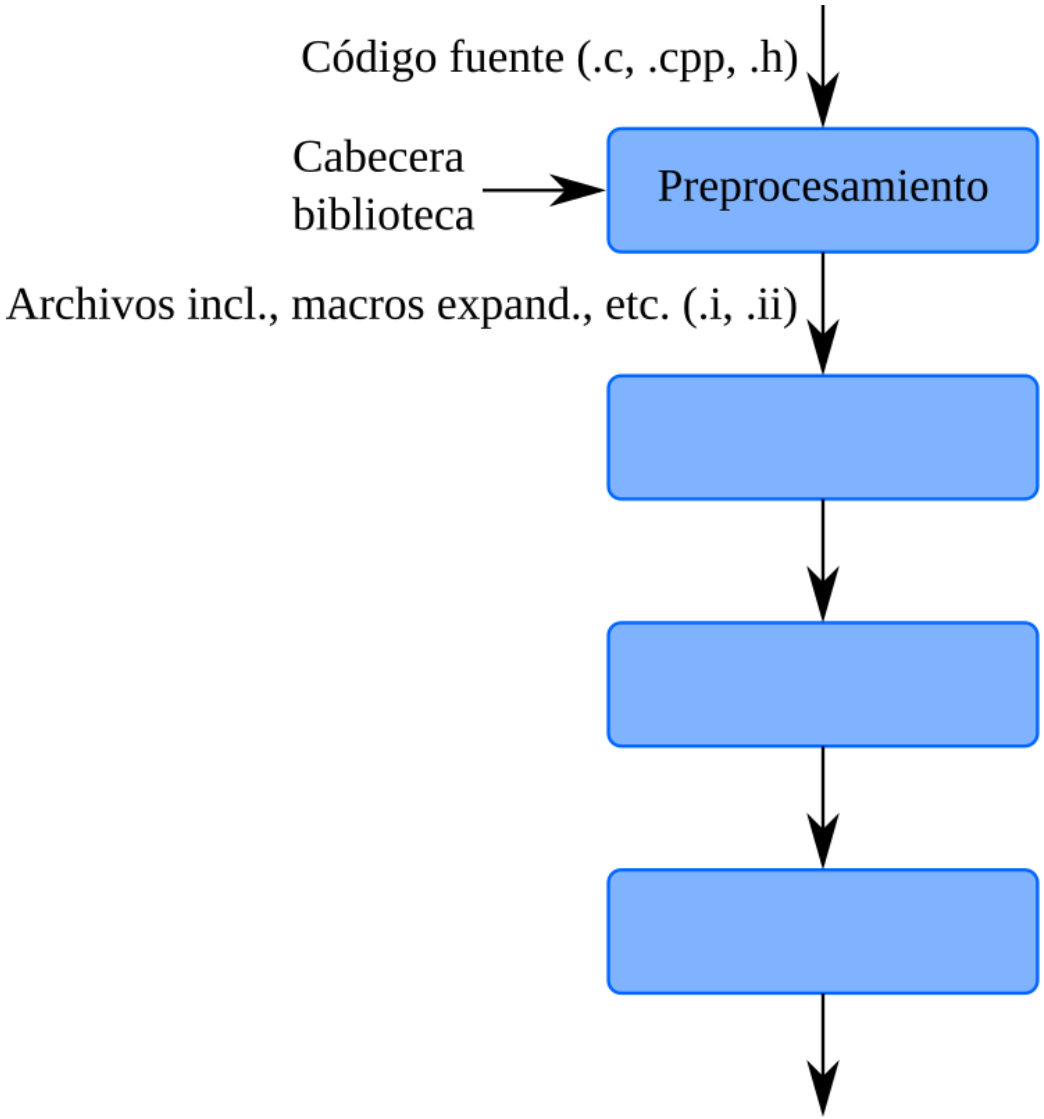
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



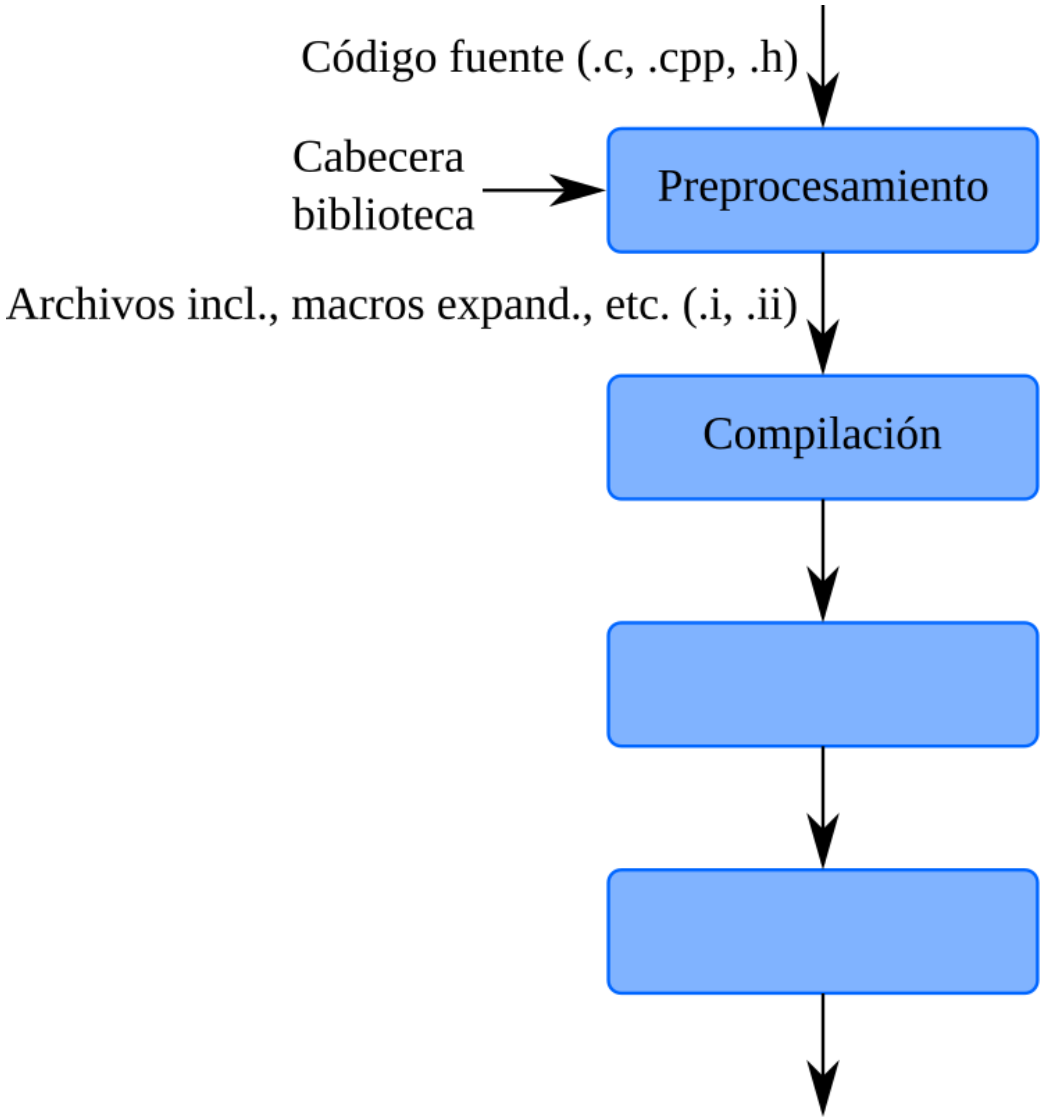
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



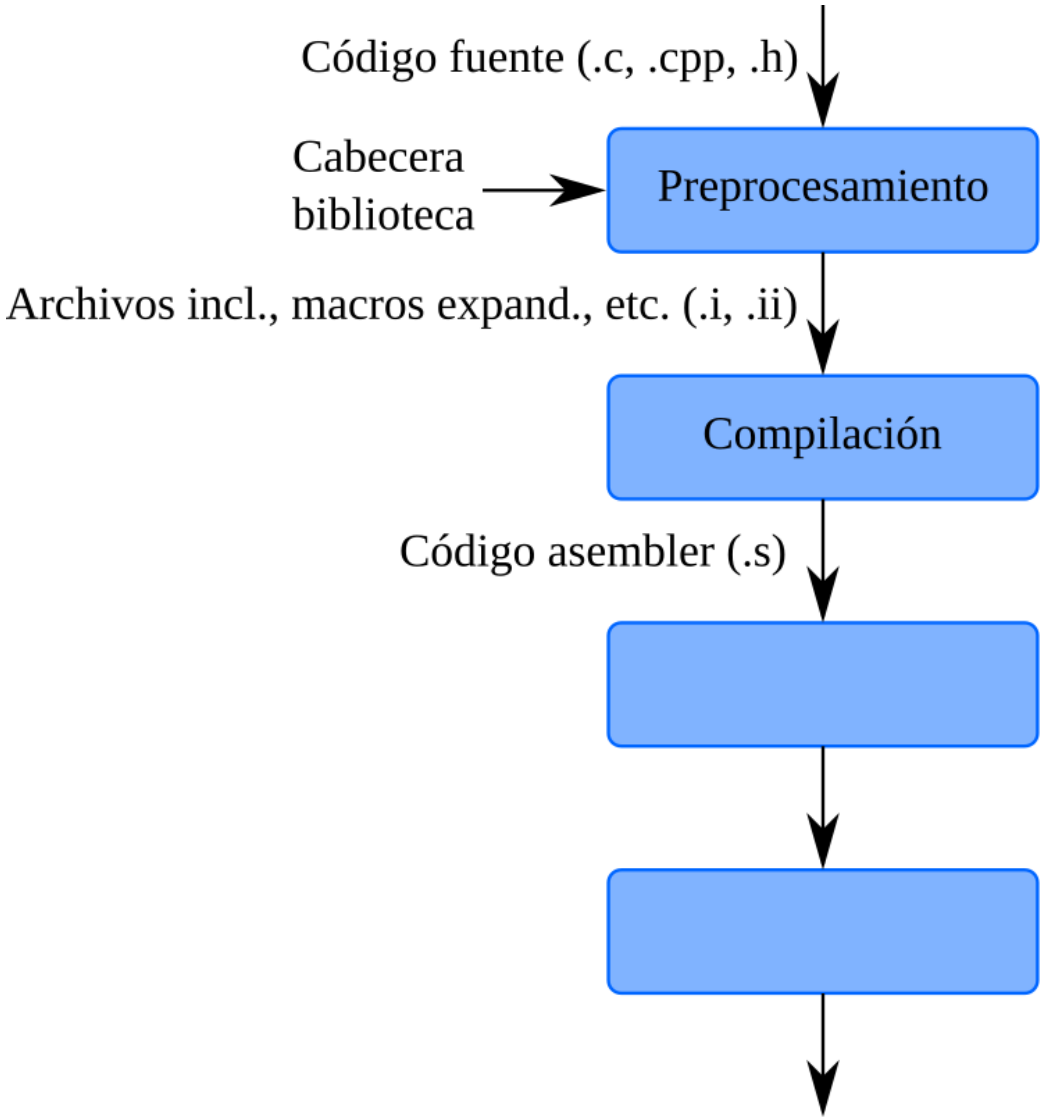
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



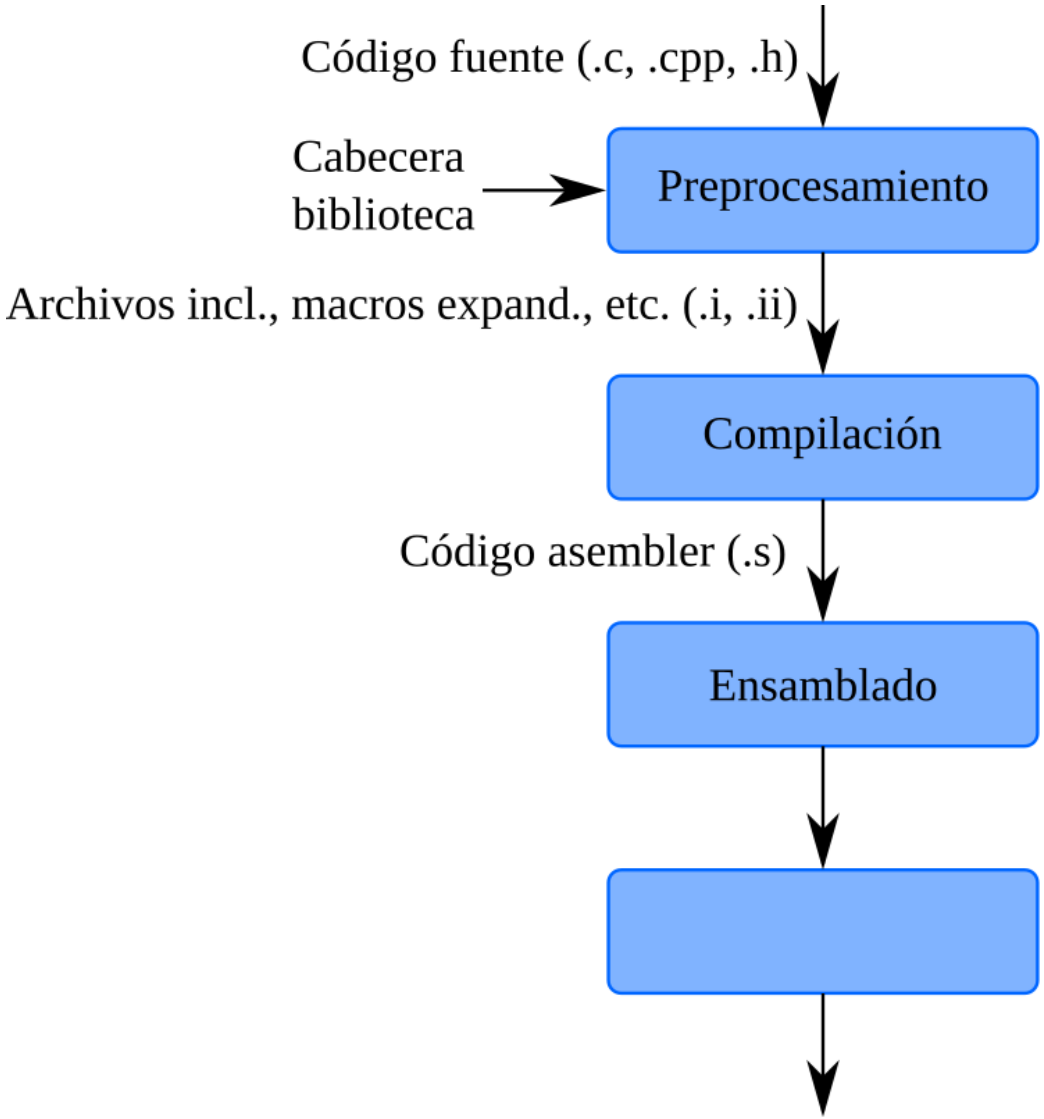
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



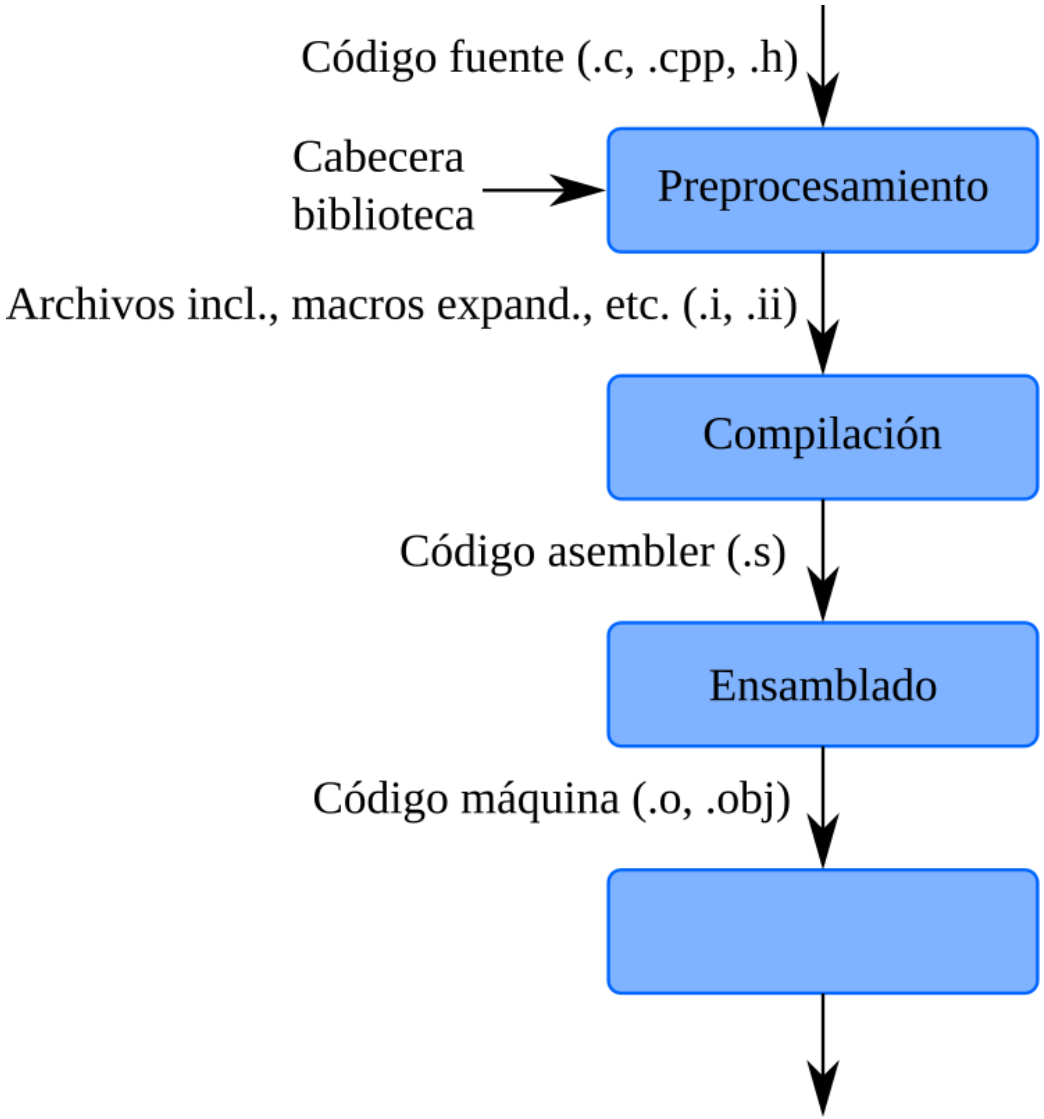
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



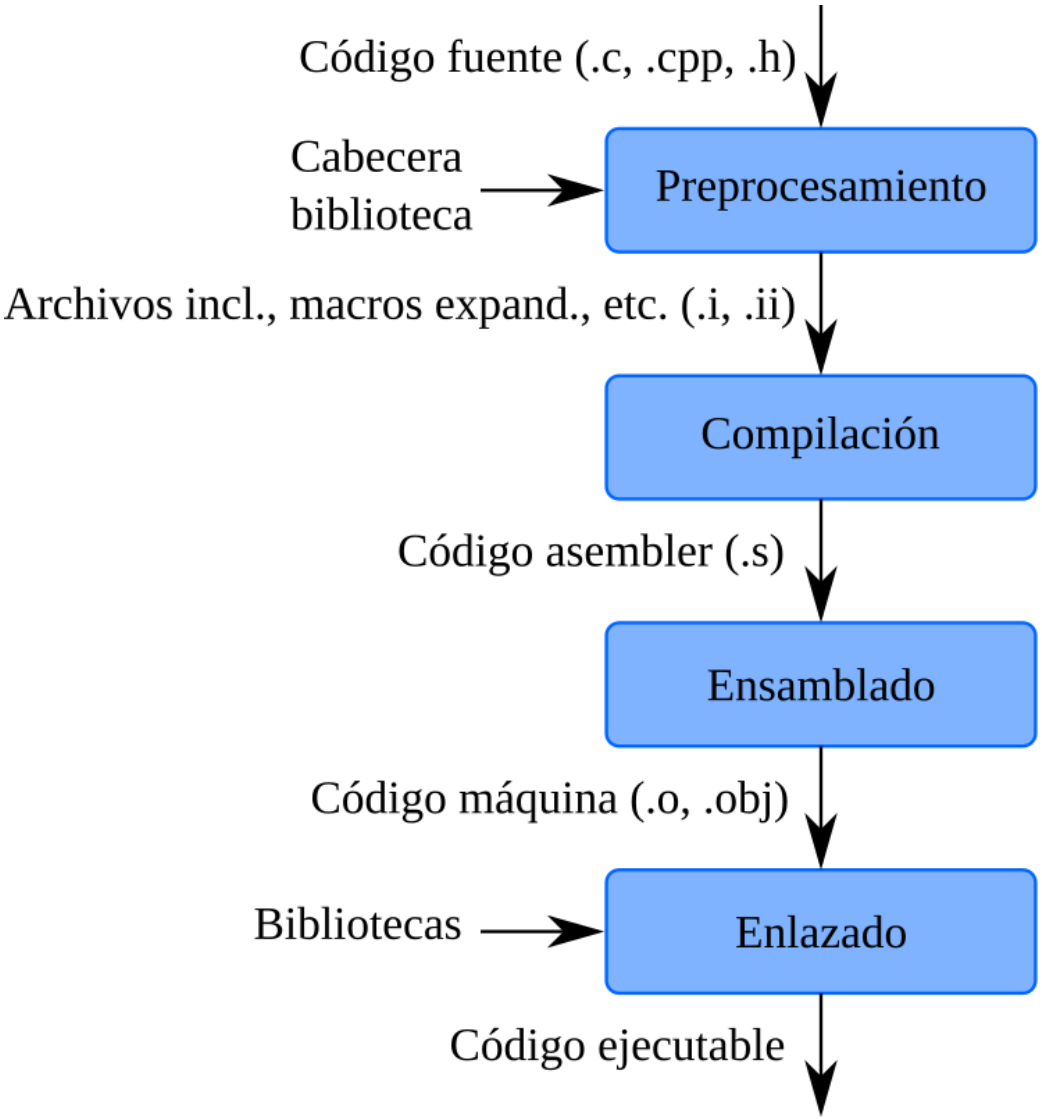
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



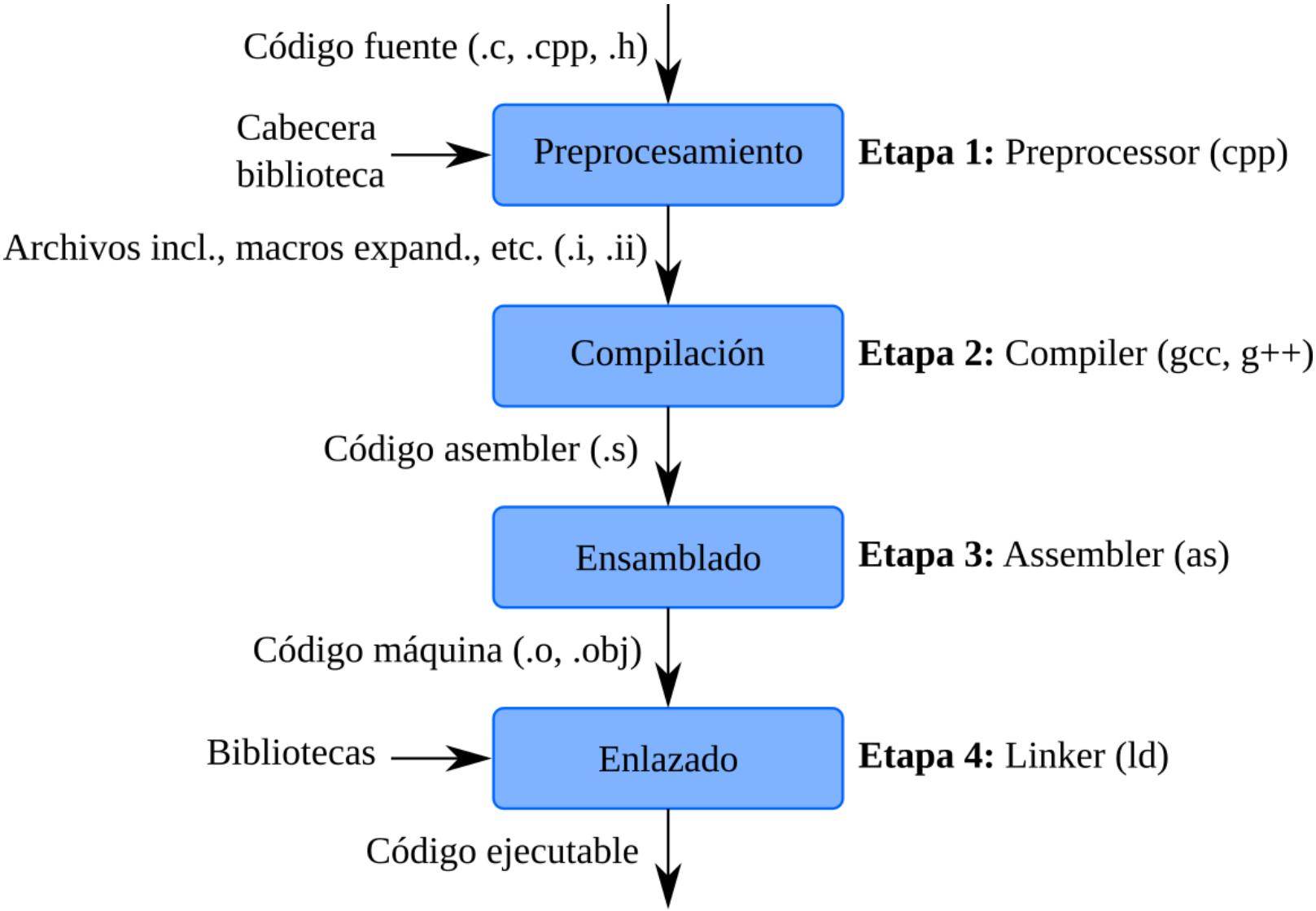
Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado).



Etapas de compilación/construcción

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y enlazado). Conjunto de herramientas: *toolchain*.



Preprocesado

test.c

```
1 #define TEST "Hola mundo!"
2 const char str[] = TEST;
```

Preprocesado

test.c

```
1 #define TEST "Hola mundo!"
2 const char str[] = TEST;
```

```
$> gcc -E test.c
```

(O bien: `cpp test.c`)

Preprocesado

test.c

```
1 #define TEST "Hola mundo!"
2 const char str[] = TEST;
```

```
$> gcc -E test.c
```

(O bien: `cpp test.c`)

Salida:

```
# 1 "test.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "test.c"

const char str[] = "Hola, Mundo!";
```

Preprocesado

test.c

```
1 #define TEST "Hola mundo!"
2 const char str[] = TEST;
```

```
$> gcc -E test.c
```

(O bien: `cpp test.c`)

Salida:

```
# 1 "test.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "test.c"

const char str[] = "Hola, Mundo!";
```

(el preprocesador inserta líneas de registro de archivo fuente y el nro. de línea en la forma `#num-de-linea "archivo fuente"`)

Preprocesado

hola1.c

```c
#include <stdio.h>

/* Función main */
int main(void)
{
  /* Uso de cadena literal */
  printf("Hola mundo!\n");
  return 0;
}
```

hola2.c

```c
#include <stdio.h>
#define MENSAJE "Hola mundo!\n"

/* Función main */
int main(void)
{
  /* Uso de constante simbólica */
  printf(MENSAJE);
  return 0;
}
```

Preprocesado

hola1.c

```c
#include <stdio.h>

/* Función main */
int main(void)
{
  /* Uso de cadena literal */
  printf("Hola mundo!\n");
  return 0;
}
```

Preprocesado:

```
$> gcc -E hola1.c > hola1.i
```

hola2.c

```c
#include <stdio.h>
#define MENSAJE "Hola mundo!\n"

/* Función main */
int main(void)
{
  /* Uso de constante simbólica */
  printf(MENSAJE);
  return 0;
}
```

Preprocesado:

```
$> gcc -E hola2.c > hola2.i
```

Preprocesado

hola1.c

```c
#include <stdio.h>

/* Función main */
int main(void)
{
  /* Uso de cadena literal */
  printf("Hola mundo!\n");
  return 0;
}
```

Preprocesado:

```
$> gcc -E hola1.c > hola1.i
```

hola2.c

```c
#include <stdio.h>
#define MENSAJE "Hola mundo!\n"

/* Función main */
int main(void)
{
  /* Uso de constante simbólica */
  printf(MENSAJE);
  return 0;
}
```

Preprocesado:

```
$> gcc -E hola2.c > hola2.i
```

Comparar los archivos de salida.

Preprocesado

hola1.c

```c
#include <stdio.h>

/* Función main */
int main(void)
{
  /* Uso de cadena literal */
  printf("Hola mundo!\n");
  return 0;
}
```

hola2.c

```c
#include <stdio.h>
#define MENSAJE "Hola mundo!\n"

/* Función main */
int main(void)
{
  /* Uso de constante simbólica */
  printf(MENSAJE);
  return 0;
}
```

Preprocesado:

```
$> gcc -E hola1.c > hola1.i
```

Preprocesado:

```
$> gcc -E hola2.c > hola2.i
```

Comparar los archivos de salida.

Observar: constantes simbólicas, comentarios, archivos cabecera (includes).

Compilado, ensamblado y enlazado

Compilación del archivo de salida del preprocesador:

```
$> gcc -Wall -S hola.i
```

Entrada: `hola.i` – Salida: `hola.s`

Compilado, ensamblado y enlazado

Compilación del archivo de salida del preprocesador:

```
$> gcc -Wall -S hola.i
```

Entrada: `hola.i` – Salida: `hola.s`

Ensamblado:

```
$> gcc -c hola.s -o hola.o
```

Entrada: `hola.s` – Salida: `hola.o` (O bien: `as hola.s -o hola.o`)

Compilado, ensamblado y enlazado

Compilación del archivo de salida del preprocesador:

```
$> gcc -Wall -S hola.i
```

Entrada: `hola.i` – Salida: `hola.s`

Ensamblado:

```
$> gcc -c hola.s -o hola.o
```

Entrada: `hola.s` – Salida: `hola.o` (O bien: `as hola.s -o hola.o`)

Enlazado:

```
$> gcc hola.o -o hola
```

(Se puede utilizar también `ld`)

Construcción paso a paso – Resumen

Ejecutar diferentes etapas de construcción:

- `gcc -E`: Preprocesamiento sin compilación
- `gcc -S`: Compilación sin ensamblado
- `gcc -c`: Preprocesamiento, compilación y ensablado sin enlazado

Construcción paso a paso – Resumen

Ejecutar diferentes etapas de construcción:

- `gcc -E`: Preprocesamiento sin compilación
- `gcc -S`: Compilación sin ensamblado
- `gcc -c`: Preprocesamiento, compilación y ensablado sin enlazado

Ver página de manual de `gcc` (`man gcc`)

Construcción paso a paso – Resumen

Ejecutar diferentes etapas de construcción:

- `gcc -E`: Preprocesamiento sin compilación
- `gcc -S`: Compilación sin ensamblado
- `gcc -c`: Preprocesamiento, compilación y ensablado sin enlazado

Ver página de manual de `gcc` (`man gcc`)

Flag `--save-temps` genera archivos intermedios

```
$> gcc --save-temps hola.c -o hola
```

Construcción paso a paso – Resumen

Ejecutar diferentes etapas de construcción:

- `gcc -E`: Preprocesamiento sin compilación
- `gcc -S`: Compilación sin ensamblado
- `gcc -c`: Preprocesamiento, compilación y ensablado sin enlazado

Ver página de manual de `gcc` (`man gcc`)

Flag `--save-temps` genera archivos intermedios

```
$> gcc --save-temps hola.c -o hola
```

Otros flags: `-std=c90`, `-Wall`, `-Werror`

Construcción paso a paso – Archivos de salida

```
$> file hola.s
hola.s: assembler source, ASCII text
```

[Probar `file hola.c` y `file hola.i`]

Construcción paso a paso – Archivos de salida

```
$> file hola.s
hola.s: assembler source, ASCII text
```

[Probar `file hola.c` y `file hola.i`]

```
$> file hola.o
hola.o: ELF 64-bit LSB relocatable, x86-64,
version 1 (SYSV), not stripped
```

Construcción paso a paso – Archivos de salida

```
$> file hola.s
hola.s: assembler source, ASCII text
```

[Probar `file hola.c` y `file hola.i`]

```
$> file hola.o
hola.o: ELF 64-bit LSB relocatable, x86-64,
version 1 (SYSV), not stripped
```

```
$> file hola
hola ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-
linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=620d3c9fadabd53755c0a647c0a43a172481a6f8, not stripped
```

Construcción paso a paso – Archivos de salida

```
$> file hola.s
hola.s: assembler source, ASCII text
```

[Probar `file hola.c` y `file hola.i`]

```
$> file hola.o
hola.o: ELF 64-bit LSB relocatable, x86-64,
version 1 (SYSV), not stripped
```

```
$> file hola
hola ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-
linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=620d3c9fadabd53755c0a647c0a43a172481a6f8, not stripped
```

```
$> ldd hola
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
```

Construcción paso a paso – Archivos de salida

```
$> file hola.s
hola.s: assembler source, ASCII text
```

[Probar `file hola.c` y `file hola.i`]

```
$> file hola.o
hola.o: ELF 64-bit LSB relocatable, x86-64,
version 1 (SYSV), not stripped
```

```
$> file hola
hola ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-
linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]
=620d3c9fadabd53755c0a647c0a43a172481a6f8, not stripped
```

```
$> ldd hola
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
```

ELF: Executable and Linkable Format

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

Opciones más comunes:

- `-ansi`: en C es equivalente a `-std=c90`

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

Opciones más comunes:

- `-ansi`: en C es equivalente a `-std=c90`
- `-Wall`: habilita todos las advertencias (warnings)

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

Opciones más comunes:

- `-ansi`: en C es equivalente a `-std=c90`
- `-Wall`: habilita todos las advertencias (warnings)
- `-Werror`: convierte las advertencias en errores

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

Opciones más comunes:

- `-ansi`: en C es equivalente a `-std=c90`
- `-Wall`: habilita todos las advertencias (warnings)
- `-Werror`: convierte las advertencias en errores
- `-pedantic`: genera advertencias si se utiliza alguna extensión de GNU

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

Estándares del lenguaje C

`gcc` compila por defecto el dialecto de GNU del lenguaje C, llamado GNU C.

Este dialecto incorpora el estándar oficial ANSI/ISO con varias extensiones[1] útiles para sistemas GNU, por ejemplo:

- funciones definidas dentro de funciones
- vectores de tamaño variable
- estructuras vacías, etc.

Opciones más comunes:

- `-ansi`: en C es equivalente a `-std=c90`
- `-Wall`: habilita todos las advertencias (warnings)
- `-Werror`: convierte las advertencias en errores
- `-pedantic`: genera advertencias si se utiliza alguna extensión de GNU
- `-std`: puede ser `-std=c90`, `-std=c99`, etc.

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html