

Informática II

Construcción de proyectos con make

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2018 –

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, \u0021%s!\n", nombre);
7 }
```

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, \u0026s!\n", nombre);
7 }
```

hola.h

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

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, %s!\n", nombre);
7 }
```

hola.h

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

Compilación

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

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, %s!\n", nombre);
7 }
```

hola.h

```
1 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
```

Contrucción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, %s!\n", nombre);
7 }
```

hola.h

```
1 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 unirlos con el linker

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

Construcción de proyectos con make

main.c

```
1 #include "hola.h"
2
3 int main(void)
4 {
5     hola("mundo");
6     return 0;
7 }
```

hola.c

```
1 #include <stdio.h>
2 #include "hola.h"
3
4 void hola(const char * nombre)
5 {
6     printf("Hola, %s!\n", nombre);
7 }
```

hola.h

```
1 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 unirlos con el linker

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

Esto permite modificar un archivo fuente y recompilar solo el archivo modificado.

Construcción de proyectos con make

Construcción con make

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

Construcción de proyectos con make

Construcción con make

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

Ejecutar

```
Hola, mundo!
```

Construcción de proyectos con make

Construcción con make

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

Ejecutar

```
Hola, mundo!
```

Modificar el archivo fuente main.c y reconstruir

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

Construcción de proyectos con make

Construcción con make

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

Ejecutar

```
Hola, mundo!
```

Modificar el archivo fuente main.c y reconstruir

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

Ejecutar

```
Hola, InfoII!
```


Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- ▶ `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.

Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- ▶ `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- ▶ Automatiza: qué partes construir, cómo construirlas, y cuando.

Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- ▶ `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- ▶ Automatiza: qué partes construir, cómo construirlas, y cuando.
- ▶ Le permite al programador poder concentrarse en el código.

Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- ▶ `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- ▶ Automatiza: qué partes construir, cómo construirlas, y cuando.
- ▶ Le permite al programador poder concentrarse en el código.
- ▶ `make` minimiza el tiempo de construcción (determina qué archivos cambiaron), además trabaja con [dependencias](#).

Construcción de proyectos con `make` – Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- ▶ `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- ▶ Automatiza: qué partes construir, cómo construirlas, y cuando.
- ▶ Le permite al programador poder concentrarse en el código.
- ▶ `make` minimiza el tiempo de construcción (determina qué archivos cambiaron), además trabaja con `dependencias`.

(Optimiza el tiempo del ciclo `editar-compilar-verificar`)

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`

Construcción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`
- ▶ Qué pasa si se modifica `b.h` y no se recompila `2.c`

Construcción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`
- ▶ Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.o
```

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`
- ▶ Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.o
main.o: main.c a.h
```

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`
- ▶ Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.o
main.o: main.c a.h
2.o: 2.c a.h b.h
```

Contrucción de proyectos con `make` – Múltiples archivos fuentes

```
1 /* main.c */
2 #include "a.h"
3
4
5 . . .
```

```
1 /* 2.c */
2 #include "a.h"
3 #include "b.h"
4
5 . . .
```

```
1 /* 3.c */
2 #include "b.h"
3 #include "c.h"
4
5 . . .
```

- ▶ Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- ▶ El archivo `3.c` depende del archivo `c.h`
- ▶ Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.o
main.o: main.c a.h
2.o: 2.c a.h b.h
3.o: 3.c b.h c.h
```

Construcción de proyectos con `make` – Archivo `Makefile`

Un archivo `Makefile` es un archivo de texto que contiene *reglas* que le indican a `make` qué construir y cómo. Una *regla* consiste en:

Construcción de proyectos con `make` – Archivo `Makefile`

Un archivo `Makefile` es un archivo de texto que contiene *reglas* que le indican a `make` qué construir y cómo. Una *regla* consiste en:

- ▶ Un *target* (objetivo): lo que se debe construir
- ▶ Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- ▶ Una lista de *comandos* a ejecutar para construir el objetivo

Construcción de proyectos con `make` – Archivo `Makefile`

Un archivo `Makefile` es un archivo de texto que contiene *reglas* que le indican a `make` qué construir y cómo. Una *regla* consiste en:

- ▶ Un *target* (objetivo): lo que se debe construir
- ▶ Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- ▶ Una lista de *comandos* a ejecutar para construir el objetivo

```
target: dependency dependency [...]  
    command  
    command  
    [...]
```

Construcción de proyectos con `make` – Archivo `Makefile`

Un archivo `Makefile` es un archivo de texto que contiene *reglas* que le indican a `make` qué construir y cómo. Una *regla* consiste en:

- ▶ Un *target* (objetivo): lo que se debe construir
- ▶ Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- ▶ Una lista de *comandos* a ejecutar para construir el objetivo

```
target: dependency dependency [...]  
    command  
    command  
    [...]
```

Cuando se ejecuta, `make` busca los archivos `GNUmakefile`, `makefile`, y `Makefile`, en ese orden.

Contrucción de proyectos con make – Archivo Makefile

Ejemplo

```
1 editor : editor.o screen.o keyboard.o
2   gcc -o editor editor.o screen.o keyboard.o
3
4 editor.o : editor.c editor.h keyboard.h screen.h
5   gcc -c editor.c
6
7 screen.o : screen.c screen.h
8   gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

Contrucción de proyectos con make – Archivo Makefile

Ejemplo

```
1 editor : editor.o screen.o keyboard.o
2   gcc -o editor editor.o screen.o keyboard.o
3
4 editor.o : editor.c editor.h keyboard.h screen.h
5   gcc -c editor.c
6
7 screen.o : screen.c screen.h
8   gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Construcción de proyectos con make – Archivo Makefile

Ejemplo

```
1 editor : editor.o screen.o keyboard.o
2   gcc -o editor editor.o screen.o keyboard.o
3
4 editor.o : editor.c editor.h keyboard.h screen.h
5   gcc -c editor.c
6
7 screen.o : screen.c screen.h
8   gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Construir/compilar el proyecto `editor`

```
$ make
```

Construcción de proyectos con make – Archivo Makefile

Ejemplo

```
1 editor : editor.o screen.o keyboard.o
2   gcc -o editor editor.o screen.o keyboard.o
3
4 editor.o : editor.c editor.h keyboard.h screen.h
5   gcc -c editor.c
6
7 screen.o : screen.c screen.h
8   gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Construir/compilar el proyecto `editor`

```
$ make
```

(\$ make clean)

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`
- ▶ Editar `main.c`, `2.c` y `3.c`

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`
- ▶ Editar `main.c`, `2.c` y `3.c`
- ▶ `$ make -f Makefile1`

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`
- ▶ Editar `main.c`, `2.c` y `3.c`
- ▶ `$ make -f Makefile1`
- ▶ Renombrar archivo makefile

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`
- ▶ Editar `main.c`, `2.c` y `3.c`
- ▶ `$ make -f Makefile1`
- ▶ Renombrar archivo makefile
- ▶ `$ touch b.h`

Contrucción de proyectos con make – Probando make

```
1 myapp: main.o 2.o 3.o
2   gcc -o myapp main.o 2.o 3.o
3
4 main.o: main.c a.h
5   gcc -c main.c
6
7 2.o: 2.c a.h b.h
8   gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11  gcc -c 3.c
```

- ▶ `$ make -f Makefile1`, y analizar error
- ▶ Crear archivos header. `$ touch a.h b.h c.h`
- ▶ Editar `main.c`, `2.c` y `3.c`
- ▶ `$ make -f Makefile1`
- ▶ Renombrar archivo makefile
- ▶ `$ touch b.h`
- ▶ Eliminar el archivo `2.o`, y probar nuevamente

Contrucción de proyectos con make – Comentarios y macros

```
1 all: myapp
2
3 # Which compiler
4 CC = gcc
5
6 # Where are include file kept
7 INCLUDE = .
8
9 # Options for development
10 CFLAGS = -g -Wall -ansi
11 # Options for release
12 # CFLAGS = -O -Wall -ansi
13
14 myapp: main.o 2.o 3.o
15     $(CC) -o myapp main.o 2.o 3.o
16
17 main.o: main.c a.h
18     $(CC) -I$(INCLUDE) $(CFLAGS) -c main.c
19
20 2.o: 2.c a.h b.h
21     $(CC) -I$(INCLUDE) $(CFLAGS) -c 2.c
22
23 3.o: 3.c b.h c.h
24     $(CC) -I$(INCLUDE) $(CFLAGS) -c 3.c
```

(\$ rm myapp *.o, y \$ make -f Makefile2)

Contrucción de proyectos con make – Comentarios y macros

```
1 all: myapp
2
3 # Which compiler
4 CC = gcc
5
6 # Where are include file kept
7 INCLUDE = .
8
9 # Options for development
10 CFLAGS = -g -Wall -ansi
11 # Options for release
12 # CFLAGS = -O -Wall -ansi
13
14 myapp: main.o 2.o 3.o
15     $(CC) -o myapp main.o 2.o 3.o
16
17 main.o: main.c a.h
18     $(CC) -I$(INCLUDE) $(CFLAGS) -c main.c
19
20 2.o: 2.c a.h b.h
21     $(CC) -I$(INCLUDE) $(CFLAGS) -c 2.c
22
23 3.o: 3.c b.h c.h
24     $(CC) -I$(INCLUDE) $(CFLAGS) -c 3.c
```

(\$ rm myapp *.o, y \$ make -f Makefile2)

VARIABLE=VALUE

Construcción de proyectos con make

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

Construcción de proyectos con make

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

- ▶ `make` tiene muchas reglas por defecto llamadas *reglas implícitas*

Construcción de proyectos con make

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

- ▶ `make` tiene muchas reglas por defecto llamadas *reglas implícitas*
- ▶ Por ejemplo: indicar que los archivos `.o` sean construidos desde archivos `.c`, y que el binario sea creado enlazando los archivos `.o` juntos

Construcción de proyectos con `make`

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

- ▶ `make` tiene muchas reglas por defecto llamadas *reglas implícitas*
- ▶ Por ejemplo: indicar que los archivos `.o` sean construidos desde archivos `.c`, y que el binario sea creado enlazando los archivos `.o` juntos
- ▶ Se definen mediante las variables de `make` (`CC` y `CFLAGS`)

Construcción de proyectos con `make`

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

- ▶ `make` tiene muchas reglas por defecto llamadas *reglas implícitas*
- ▶ Por ejemplo: indicar que los archivos `.o` sean construidos desde archivos `.c`, y que el binario sea creado enlazando los archivos `.o` juntos
- ▶ Se definen mediante las variables de `make` (`CC` y `CFLAGS`)
- ▶ Para el lenguaje C
 - ▶ `CC` es el compilador
 - ▶ `CFLAGS` son opciones del compilador

Construcción de proyectos con make

Makefile para el ejemplo del comienzo

```
1 C=gcc
2 CFLAGS=-Wall
3
4 hola: main.o hola.o
5
6 clean:
7   rm -f hola main.o hola.o
```

- ▶ **make** tiene muchas reglas por defecto llamadas *reglas implícitas*
- ▶ Por ejemplo: indicar que los archivos `.o` sean construidos desde archivos `.c`, y que el binario sea creado enlazando los archivos `.o` juntos
- ▶ Se definen mediante las variables de **make** (**CC** y **CFLAGS**)
- ▶ Para el lenguaje C
 - ▶ **CC** es el compilador
 - ▶ **CFLAGS** son opciones del compilador
- ▶ Para el lenguaje C++
 - ▶ **CXX** es el compilador
 - ▶ **CXXFLAGS** son opciones del compilador

