

# Programación en C moderno

Álvaro Neira Ayuso <alvaro@soleta.eu>

# Introducción al lenguaje C

- Un poco de historia: Desde 1972 hasta hoy día.
- Proyectos de referencia
- Por qué y para qué el lenguaje C.
- Características de C.
- Bibliografía recomendada.

# Un poco de historia: Desde 1972 hasta hoy día

- El desarrollo inicial de C se llevó a cabo en los entre 1969 y 1973
- Laboratorios Bell de AT&T
- "C" porque muchas de sus características fueron tomadas de un lenguaje anterior llamado "B".
- Lo programadores querían jugar con Space Travel

# Un poco de historia: Desde 1972 hasta hoy día

- El objetivo de desarrollar Unix fue crear un sistema que automatizase archivos de patentes.
- La versión original de Unix se desarrolló en lenguaje ensamblador.
- Más tarde, el lenguaje C se desarrolló para poder reescribir el sistema operativo.

# Un poco de historia: Desde 1972 hasta hoy día

- En 1978, Ritchie y Brian Kernighan publicaron la primera edición de El lenguaje de programación C.



# Un poco de historia: Desde 1972 hasta hoy día

- Mas tarde, se añadieron al lenguaje muchas características no oficiales, que estaban soportadas por los compiladores de AT&T, como el uso de void, struct o bibliotecas estándares

# Un poco de historia: Desde 1972 hasta hoy día

```
int power()

power()
int a, b;
{
    int n;
    for (n = 1; b > 0; --b)
        n *= a;
    return n;
}
```

# Un poco de historia: Desde 1972 hasta hoy día

- A finales de la década de 1970, C empezó a sustituir a BASIC como lenguaje de programación.
- Bell Labs creó un nuevo lenguaje de programación C++. Basado en el lenguaje C añadiendo funcionalidades de programación orientada a objetos a C.
- También nace Objective C. Otro lenguaje basado en C añadiendo también características de programación orientada a objetos a C.



# Un poco de historia: Desde 1972 hasta hoy día

- Con el objetivo de estandarizar C se crea un comité por el Instituto Nacional Estadounidense de Estándares en 1983.
- En 1989 se ratificó el primer estándar de C: ANSI C o Lenguaje de Programación C ANSI.
- Organización Internacional para la Estandarización (ISO) en 1990, acepta el estándar ANSI C. Y recibe el nombre de C90.

# Un poco de historia: Desde 1972 hasta hoy día

- En 1999 aparece el estándar C99
- Algunas nuevas características:
  - Las variables pueden declararse en cualquier sitio
  - Nuevos datos como long long int (para reducir el engorro de la transición de 32 bits a 64 bits) o un tipo de datos booleano.
  - Arrays de longitud variable.
  - Soporte para comentarios de una línea que empiecen con //
  - muchas funciones nuevas, como snprintf()
  - algunos headers nuevos, como stdint.h.
- En 2011, la ISO publica el C11.

- Núcleo de Linux con 37.792 ficheros y mas de 19.688.408 líneas de código.

**Key**

- [Light Blue Box] No Coverage
- [Dark Blue Box] Coverage
- [Medium Blue Box] Medium Coverage
- [Light Cyan Box] Low Coverage
- [Red Box] Other Isolates
- [Green Box] Reported Isolates
- [Yellow Box] Extensively cultured Isolates
- [Orange Box] Multi-Isolate
- [Purple Box] Isolation source
- [Grey Box] Source Reference

# Proyectos en C: Núcleo de Linux

Company	Changes	Total
None	12,550	13.6%
Red Hat	9,483	10.2%
Intel	8,108	8.8%
Texas Instruments	3,814	4.1%
Linaro	3,791	4.1%
SUSE	3,212	3.5%
Unknown	3,032	3.3%
IBM	2,858	3.1%
Samsung	2,415	2.6%
Google	2,255	2.4%
Vision Engraving Systems	2,107	2.3%
Consultants	1,529	1.7%
Wolfson Microelectronics	1,516	1.6%
Oracle	1,248	1.3%
Broadcom	1,205	1.3%

Company	Changes	Total
NVidia	1,192	1.3%
Freescall	1,127	1.2%
Ingics Technology	1,075	1.2%
Renesas Electronics	1,010	1.1%
Qualcomm	965	1.0%
Cisco	871	0.9%
The Linux Foundation	840	0.9%
Academics	831	0.9%
AMD	820	0.9%
Inktank Storage	709	0.8%
NetApp	707	0.8%
LINBIT	705	0.8%
Fujitsu	694	0.7%
Parallels	684	0.7%
ARM	664	0.7%

 Linux Foundation

# Proyectos en C

- navegador web Firefox
- el servidor web Apache
- la interfaz web cgit
- el toolkit gráfico GTK
- el NDK de Android
- el juego Doom



# Por qué y para qué el lenguaje C

- Lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas. Si quieres sacarle el máximo rendimiento a una máquina, programa en C.
- Lenguaje que puede ser compilado para casi todos los sistemas conocidos.
- Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes.

# Por qué y para qué el lenguaje C

Feb 2015	Feb 2014	Change	Programming Language	Ratings	Change
1	1		C	16.488%	-1.85%
2	2		Java	15.345%	-1.97%
3	4	⬆	C++	6.612%	-0.28%
4	3	⬇	Objective-C	6.024%	-5.32%
5	5		C#	5.738%	-0.71%
6	9	⬆	JavaScript	3.514%	+1.58%
7	6	⬇	PHP	3.170%	-1.05%
8	8		Python	2.882%	+0.72%
9	10	⬆	Visual Basic .NET	2.026%	+0.23%
10	-	⬆⬆	Visual Basic	1.718%	+1.72%

# Por qué y para qué el lenguaje C

<input type="text"/>	C,
52°North Initiative for Geospatial Open Source Software GmbH	geo, geospatial, java, gis, spatiotemporal, geoprocessing, traf
AerospaceResearch.Net	space, aviation, citizen science, distributed computing, high p
Apache Software Foundation	c, java, python, c++, perl, opensource, apache, erlang
Apertium	mt, nlp, linguistics, grammar, c, c++, python, java, xml, morpho
Blender Foundation	3D, computer graphics, modeling, animation, rendering, compo
BRL-CAD	computer graphics, scientific computing, engineering analysis,
Catrobat (formerly Catroid Project)	Android, TDD, Scratch, robotium, agile, children, educational, v
CCEXtractor development	c, c++, video, mpeg, h264, accesibility, subtitles, closed captio
Computational Science and Engineering at TU Wien	CSE, Java, C++, R, XML, Qt, Network Services, Data Encoding,
Crystal Space	3d engine, game engine, c++, opengl, bullet, game logic, cross p
Debian Project	os, linux, distributions, web, qa, quality_assurance, shell_scrip
Dr. Memory	instrumentation, Windows, C, C++, assembly, debugger, progra
FreeBSD	virtualization, hypervisor, os, operating system, bsd, web, com
Ganglia	c, python, java, javascript, jquery, php, monitoring, hpc, grid,
GCC - GNU Compiler Collection	gcc, compiler, c, c++, toolchain, glibc, binutils, gdb
Gentoo Foundation	virtualization, linux, distributions, xml, web, java, gtk, shell_sc
Git	vcs, c, git
GNOME	application, banshee, boxes, c, clutter, desktop, easytag, game
GNU Octave	c++, mathematics, math, numerical, matlab, mercurial, scientif
GNU Project	C, python, metalink, tests, networking, P2P, distributed update
Haiku	c++, operating system, c, virtio, usb, driver, kernel, arm, uefi,
HelenOS group at Department of Distributed and Dependable Sy	microkernel, multiserver, os, operating system, operating, syst
illumos	illumos, operating system, unix, kernel, dtrace, zfs, openzfs, v
Jitsi	jitsi, audio, video, java, sip, xmpp, rtp, webrtc, javascript, ht
KDE	accessibility, audio, browser, c++, desktop, education, games,
Linux Trace Toolkit next generation project (LTTng)	tracing, linux, android, arm, c, c++, python, kernel internals
Liquid Galaxy Project	google earth, immersive, 3d, virtual reality, html5, google maps
MetaBrainz Foundation Inc.	music, metadata, linux, sql, perl, python, id3
Mixxx DJ Software	dj, audio, music, qt, c++, realtime
Monkey Project	C, Linux, Raspberry, HTTP, SPDY, Websocket, Caching, WebServ
Mono Project	C#, C, C++, Mono, VM, GC, Runtime, MonoDevelop, GTK, OSX, L



# Características de C

- Es el lenguaje de programación de propósito general asociado al sistema operativo UNIX
- Es un lenguaje muy versátil. Trata con tipos básicos como caracteres, números y también a nivel de bits y direcciones de memoria
- Posee una gran portabilidad
- Se utiliza para la programación de sistemas, construcción de interpretes, compiladores, editores de texto, etc

# Bibliografía recomendada

- Lenguaje de programación C de Ritchie, Kernighan
- Lenguaje C (wikipedia, artículo en Inglés)

# Hola mundo del C

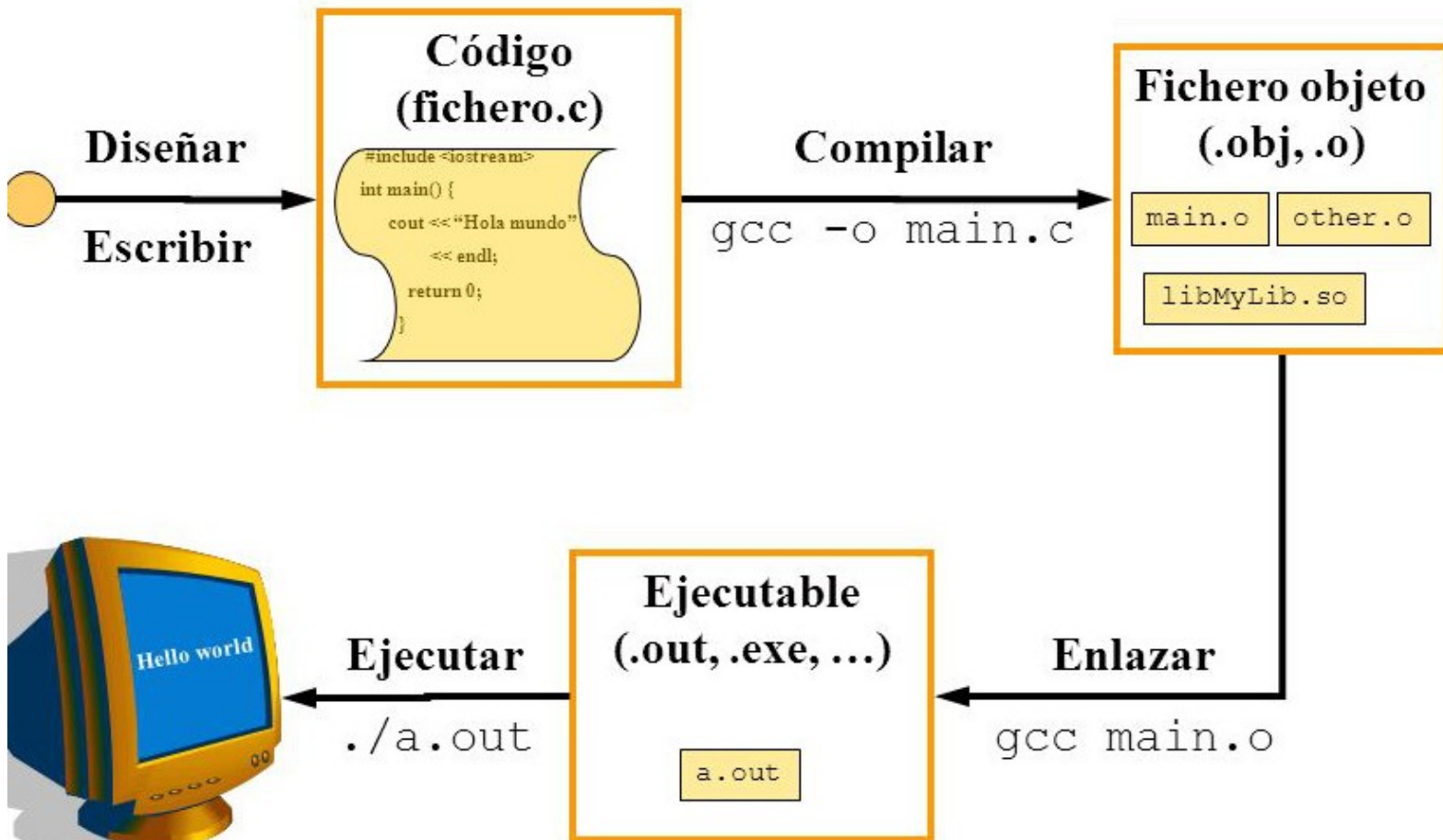
- El compilador GNU cc.
- Los #includes.
- La biblioteca estándar de C (libc)
- Introducción a las funciones en C.
- Definición y uso de funciones.
- Compilando y configurando un proyecto a través de scripts. (autoconf/automake).
- Introducción al gestor de versiones GIT y el editor de texto vim.

# El Compilador GNU cc

- El GNU Compiler Collection (colección de compiladores GNU) es un conjunto de compiladores creados por el proyecto GNU.
- GCC es software libre y lo distribuye la Free Software Foundation (FSF)
- La licencia GPL.

# El compilador GNU cc

## Escribir, compilar, enlazar, ejecutar



# El Compilador GNU cc

- gcc --help

```
-S          Compile only; do not assemble or link
-c          Compile and assemble, but do not link
-o <file>   Place the output into <file>
-pie        Create a position independent executable
-shared     Create a shared library
-x <language> Specify the language of the following input files
             Permissible languages include: c c++ assembler none
             'none' means revert to the default behavior of
             guessing the language based on the file's extension
```

# El Compilador GNU cc

- Compila el programa en C hola.c, genera un archivo ejecutable a.out.

```
gcc hola.c
```

- Compila el programa en C hola.c, genera un archivo ejecutable hola.

```
gcc -o hola hola.c
```

- Genera código objeto, en el archivo hola.o.

```
gcc -c hola.c hola.c
```

# El Compilador GNU cc

- Genera el código objeto indicando el nombre de archivo.

```
gcc -c -o holamundo.o holamundo.c
```

- Indica dos directorios donde han de buscarse bibliotecas. La opción -L debe repetirse para cada directorio de búsqueda de bibliotecas.

```
gcc -L/lib -L/usr/lib holamundo.c
```

- Indica un directorio para buscar archivos de encabezado (de extensión .h).

```
gcc -I/usr/include holamundo.c
```



# Los #includes

- La palabra include es una palabra reservada para el lenguaje.
- De forma genérica se usa para añadir un archivo al código
- Se pueden incluir desde bibliotecas externas a cabeceras generadas por nosotros mismos.

# Los #includes

- #include “fichero.h”
  - En el mismo directorio que el archivo que contiene la instrucción #include.
- #include <fichero.h>
  - A lo largo de la ruta de acceso especificada por cada opción del compilador

# La biblioteca estándar de C (libc)

- La biblioteca estándar de C (también conocida como libc)
- Es una recopilación de ficheros cabecera y bibliotecas, estandarizadas por un comité de la Organización Internacional para la Estandarización (ISO)
- Implementan operaciones comunes, tales como las de entrada y salida de información o el manejo de cadenas

# La biblioteca estándar de C (libc)

- `<stdio.h>` Conjunto de funciones para realizar entrada/salida de información.
- `<stdint.h>` Definición varios tipos enteros (nuevo en C99).
- `<stdlib.h>` Prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.
- `<errno.h>` Define las macros para mostrar errores
- `<assert.h>` Macro `assert` (aserción), utilizada para detectar errores lógicos y otros tipos de fallos en la depuración de un programa.

# Tipos de datos

- Entero

```
int numero = 0;
```

- Enteros sin signo (Biblioteca stdint.h)

```
uint32_t, uint16_t o uint8_t
```

```
uint32_t numero = 0;
```

- Real (Biblioteca float.h)

```
float numero=12.2;
```

- Carácter

```
char letra = 'a';
```

- Cadena de carácter

```
char palabra[5] = "HOLA";
```

# Definición de variables

- Variables locales

Variables que solo tienen validez en el interior (en el contexto) de una función donde se haya definido.

```
void hola()  
{  
    int i = 0;  
    ...  
}
```

# Definición de variables

- Variables globales

Variables que tiene validez (en el contexto completo) en un fichero.

```
int i;  
void hola()  
{  
    printf("valor de i = %d\n", i);  
}  
void aumentar()  
{  
    i++;  
}
```

# Tipos de datos: Coding Style

- Tipo Nombre;  
`int numero;`
- Tipo Nombre = valor;  
`int numero = 0;`



# Operadores

- Aritméticos
  - + suma
  - - resta
  - \* multiplicación
  - / división
  - % Modulo

# Operadores

- De relación
  - $<$  menor que
  - $<=$  menor o igual que
  - $>$  mayor que
  - $>=$  mayor o igual que
  - $==$  igual a
  - $!=$  distinto

# Operadores

- Lógicos
  - && AND
  - || OR
  - ! NOT
- Operadores de bit
  - & AND bit a bit
  - | OR bit a bit
  - << Desplazamiento de bits a la izquierda
  - >> Desplazamiento de bits a la derecha

# If statement

- Se trata de una estructura de control. Evalúa la condición si es verdadero o falso, y en función del resultado ejecuta unas sentencias u otras

```
if (condición) {  
    BLOQUE 1  
} else {  
    BLOQUE 2  
}
```

# If statement

- Uso del “else if”

```
if (condicion1) {  
    BLOQUE 1  
} else if (condicion2) {  
    BLOQUE 2  
} else {  
    BLOQUE 3  
}
```

# If statement: Coding style

```
int function(int x)
{
    body of function
}
```

# Switch

- Permite seleccionar las acciones a realizar de acuerdo al valor que toma una expresión

```
switch (expresión) {  
    case constante1:  
        BLOQUE1  
        break;  
    case constante2:  
        BLOQUE2  
        break;  
    default:  
        BLOQUE3  
        break;  
}
```

# Switch

```
int i = 0;
switch (i) {
case 0: // Si i == 0, entonces ejecutamos el código siguiente
    printf("es igual a 0\n");
    break;
case 1:
case 2:
    printf("no es igual a 0\n");
    break;
default:
    printf("Valor no esperado\n");
}
```



# Switch: Coding style

```
switch (suffix) {  
case 'G':  
case 'g':  
    mem <=& 30;  
    break;  
case 'M':  
case 'm':  
    mem <=& 20;  
    break;  
case 'K':  
case 'k':  
    mem <=& 10;  
    /* fall through */  
default:  
    break;  
}
```

# While

- BLOQUE de código que se repite mientras una condición previa se cumpla.

```
while (condición) {  
    BLOQUE  
}
```

```
int i = 0;  
while (i < 4) {  
    printf("Valor de i %d\n");  
    i++;  
}
```

# While: Coding style

```
int i = 0;  
while (i < 1000) {  
    i = i + 100;  
}
```

# For

```
for (expr1; expr2; expr3)  
    sentencia
```

- Bucle que se ejecuta tantas veces hasta que no se cumpla una condición (expr2)

```
int i;  
for (i = 0; i < 4; i++)  
    printf("hola\n");
```

# For

- Traducción del bucle for:

```
int i = 0;
while (i < 4) {
    printf("hola\n");
    i++;
}
```

# For: Coding style

```
int i;  
for (i = 0; i < 10; i++) {  
    ...  
}
```

# Salidas de los bucles

- C proporciona dos modos de salida de los bucles:
  - break: Provoca la salida del bucle. Si hay varios bucles anidados provoca la salida de aquel donde se encuentra.
  - continue: Provoca la salida de la presente iteración del bucle. Se vuelve a la condición.

# Salidas de los bucles

```
int i;
```

```
char cadena[30] = "hola amigos";
```

```
for (i = 1; i < 30; i++) {
```

```
    if (cadena[i] != 's')
```

```
        continue;
```

```
    printf("la posicion de s es %d\n", i);
```

```
    break;
```

```
}
```



# goto y etiquetas

- Una etiqueta tiene el nombre seguida de dos puntos
- Debe estar en la misma función en donde se encuentra el goto

```
int i = 1000)  
if (i == 1000)  
    goto mil:
```

```
printf("no es igual a mil\n");  
return 0;
```

```
mil:
```

```
printf("es igual a mil\n");  
return 1;
```

# Introducción a funciones

- Un programa en C es una colección de funciones.
- La función principal se llama main y es la primera en ejecutarse.
- Las funciones pueden residir en uno o varios ficheros fuente.

# Funciones: Definición

- Cada función tiene la forma:

```
tipo_de_dato nombre_función (argumentos)
```

```
{
```

```
    declaraciones y sentencias
```

```
}
```

```
void nombre_función(void)
```

```
{
```

```
    ...
```

```
}
```

# Funciones: Definición

- Tipos de datos que podemos usar:
  - int : Entero
  - uint32\_t, uint16\_t, uint8\_t: Enteros sin signo
  - char : Carácter
  - char \* : Puntero a una cadena de caracteres
  - void : La función no devuelve nada
  - void \* : Puntero a un objeto de tipo no definido
- Ejemplo:

```
int nombre_función (int a, char b, ...)  
{  
    declaraciones y sentencias  
}
```

# Funciones: Coding Style

```
int fun(int a)
{
    int result = 0;
    char *buffer;

    buffer = kmalloc(SIZE, GFP_KERNEL);
    if (!buffer)
        return -ENOMEM;

    if (condition1) {
        while (loop1) {
            ...
        }
        result = 1;
        goto out_buffer;
    }
    return 0;
out_buffer:
    kfree(buffer);
    return result;
}
```

# Cabeceras

- Son ficheros con la extensión .h
- Contienen declaraciones de funciones, que pueden ser usadas desde cualquier otro fichero.

# Cabeceras

- fichero util.h

```
int es_una_A(char letra);
```

- fichero util.c

```
int es_una_A(char letra) {  
    ...  
}
```

- fichero prog.c

```
#include "util.h"
```

```
void main()
```

```
{
```

```
    ...
```

```
if (es_una_A(cadena[i]))
```

```
    ...
```

```
}
```

# Funciones estáticas

- `static tipo_de_dato nombre_función(argumentos) {  
 declaraciones y sentencias  
}`



# Compilando y configurando con autoconf y automake

- En nuestros ejercicios hemos compilado los ficheros con gcc. Debido a que la cantidad de ficheros no es grande.
- Compilar proyectos con varios ficheros, se vuelve una tarea tediosa.
- Las herramientas autoconf y automake facilitan esta labor.

# Compilando y configurando con autoconf y automake: Pasos

1) Creamos un fichero configure.ac en la carpeta principal del proyecto.

2) En ese fichero escribimos la linea:

```
AC_INIT(NOMBREDELPROYECTO, VERSION,  
CORREO)
```

Esta sentencia nos inicializa el proyecto. Por ello, cerramos y guardamos el fichero.

# Compilando y configurando con autoconf y automake

- 3) Ejecutamos autoconf. Este comando nos creará una carpeta llamada autom4te.cache y el ejecutable configure.
- 4) Creamos el fichero Makefile.am en la carpeta principal. Con los siguientes comandos.

```
AUTOMAKE_OPTIONS = foreign  
SUBDIRS = src
```

La primera linea lo que establece es que nuestro programa no es de GNU por tanto "foreign". La segunda linea debemos establecer cada uno de los subdirectorios que contiene nuestro proyecto. En este caso solo tenemos src.

# Compilando y configurando con autoconf y automake

5) Crear el fichero Makefile.am dentro de src. Con la informacion correspondiente:

```
bin_PROGRAMS = holaclose
```

```
holaclose_SOURCES = holaclose.c
```

La segunda linea decimos que queremos que nos compile el fichero holaclose.c.

Con la primera linea decimos que nos genere el binario/ejecutable a partir del holaclose\_SOURCES.

# Compilando y configurando con autoconf y automake

6) Continuamos con la configuración de nuestro fichero configure.ac. Añadir las siguientes sentencias en él:

```
/* Nos inicia el proyecto usando automake */  
AM_INIT_AUTOMAKE(holaclase, 1.0)  
/* Designamos en que carpeta tenemos el código del programa principal */  
AC_CONFIG_SRCDIR([src/holaclase.c])  
/* Designamos donde deben generarse nuestros ficheros Makefile.  
AC_OUTPUT(Makefile src/Makefile)
```

# Compilando y configurando con autoconf y automake

7) Ejecutamos:

```
aclocal
```

Este comando nos genera un fichero de nombre `aclocal.m4`. Este fichero contiene todos los comandos de Makefile por ejemplo `AM_INIT_AUTOMAKE`.

8) Ejecutamos:

```
automake --add-missing
```

`automake` a partir de los ficheros `configure.ac` y `Makefile.am`, genera unos ficheros llamados `Makefile.in`. Si existe algún error o algo mal escrito, se nos informará de ello.

# Compilando y configurando con autoconf y automake

9) Ejecutamos de nuevo autoconf. Esto genera un fichero ejecutable configure, el cual tendrá en cuenta las nuevas comprobaciones y nuestros ficheros

```
Makefile.am
```

10) Ejecutamos:

```
./configure
```

11) Compilamos el programa usando

```
make
```

12) Entramos en la carpeta src y veremos nuestro programa "holaclase".