Sesión 1: Entorno de trabajo y herramientas (2h)

Esta primera sesión describe los conceptos y las herramientas necesarias para trabajar en el laboratorio. El objetivo principal es que el estudiante se familiarice con el sistema operativo Linux y su entorno gráfico; que comprenda el funcionamiento del sistema de ficheros y aprenda a utilizar los comandos básicos para gestionarlos. Además, al finalizar la sesión el estudiante deberá conocer las etapas que intervienen en la elaboración de un programa y ser capaz de utilizar las herramientas necesarias para realizar el proceso de codificación y prueba: crear, editar, compilar, ejecutar y depurar un programa en C.

1. Introducción a Linux

Un sistema operativo es un programa que actúa como intermediario entre un usuario de ordenador y el hardware del mismo. El objetivo de un sistema operativo es controlar y coordinar de forma eficiente la utilización de los recursos (periféricos, memoria, procesador) entre varios programas de diversos usuarios.

Algunos ejemplos de familias de sistemas operativos actuales son *Windows 8*, *GNU/Linux*, *MAC OS X*, *iOS*, etc. En este curso trabajaremos sobre Ubuntu (versión 20.04 LTS), que es una distribución gratuita de GNU/Linux.

No es objetivo de la asignatura que se estudien detalles de la instalación, administración e implementación del sistema. Lo que se pretende es facilidad de uso, similar a la que ya podéis tener en otros sistemas como Windows.

2. Entorno gráfico

El entorno de trabajo en el laboratorio incluye una interfaz gráfica con utilidades y servicios similares a los de *Windows*.

El entorno gráfico presenta una organización del sistema a partir de un *Escritorio o Desktop* donde se pueden encontrar los diferentes íconos que hacen accesibles los diferentes elementos del sistema (ver fig. 1).

Descripción de la barra de menú

La barra de menú (ver fig. 1) se encuentra situada en la parte superior del entorno gráfico. Esta barra contiene dos áreas:

- El área de indicadores, situada a la derecha de la barra de menú. Esta área contiene diferentes íconos, entre los cuáles los más comunes son: el indicador de teclado, el indicador de red, el indicador de mensajes, el indicador de sonido, el reloj, menú de usuario y el indicador de sesión. La aparición de estos íconos en el área de indicadores puede variar dependiendo de la instalación de *Ubuntu* que se realice.
- El menú de la aplicación, situado a la izquierda de la barra de menú. Este menú corresponde con el menú propio de la aplicación que esté activa. Para ver el menú de la aplicación activa basta con mover el ratón a la barra de menú del escritorio. Mientras el ratón esté posicionado ahí, los menús de la aplicación activa se dispondrán sobre la barra de menú del escritorio de Ubuntu, permitiendo utilizar los menús de la aplicación.

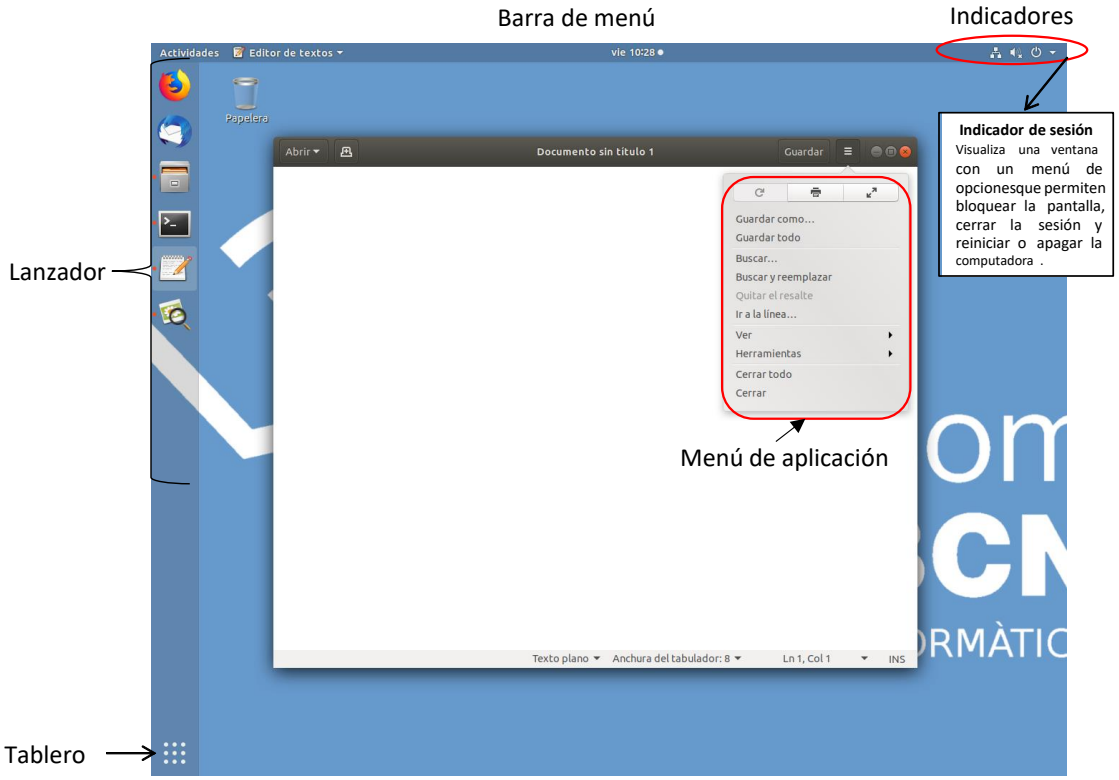


Figura 1. Aspecto del entorno gráfico de trabajo

Además de la barra de menú, el entorno gráfico presenta una barra vertical o *lanzador*, situada verticalmente en el lado izquierdo del *Escritorio* (ver fig. 1).
El lanzador proporciona un fácil acceso a las aplicaciones, dispositivos montados y a la papelera. Para ejecutar una aplicación desde el lanzador (o para hacer que se muestre una aplicación que ya se encuentra en ejecución), basta con pulsar sobre el ícono de la aplicación.

El primer ícono en la parte superior del lanzador es el tablero. El tablero es una herramienta que nos ayuda a acceder y buscar aplicaciones y ficheros en el equipo de forma rápida.

En el lanzador es posible añadir y eliminar aplicaciones.

Para **añadir una aplicación** en el lanzador podemos hacerlo de dos maneras:

1. Abrir el tablero, buscar la aplicación a añadir, arrastrarla (con el ratón) y soltarla en el lanzador.
2. Ejecutar la aplicación que se desea añadir al lanzador, hacer clic derecho sobre el icono de la misma en el lanzador y seleccionar "*Mantener en el lanzador*".

Para **eliminar una aplicación del lanzador**, hacemos clic derecho sobre el icono de la aplicación y seleccionamos "No mantener en el lanzador".

La figura 2 muestra el lanzador y describe algunos íconos de interés para trabajar en las sesiones de laboratorio.

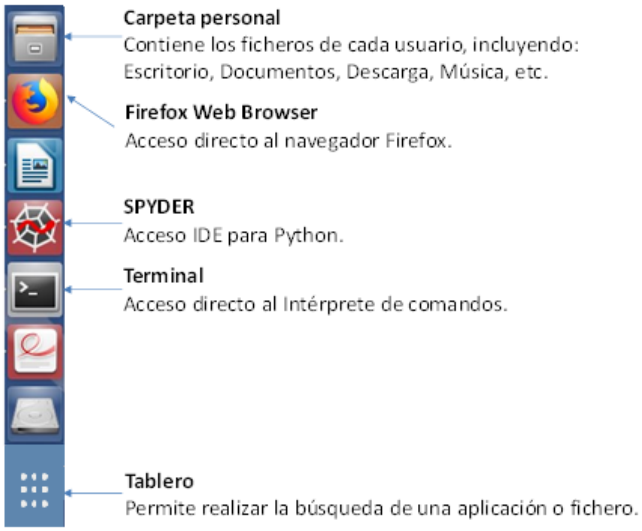


Figura 2. Descripción del lanzador

Nota: Dado que el entorno de laboratorio tiene definido como idioma el inglés, en adelante nos referiremos a la carpeta *Escritorio*, como *Desktop*.

Ejercicio 1

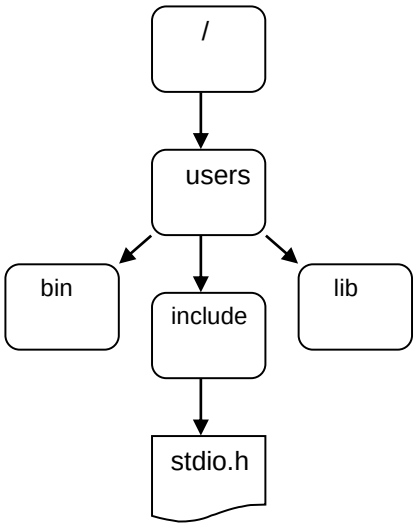
a) Abrir una ventana que muestre el contenido de la carpeta *Desktop*.
b) Abrir un *Terminal*.
c) Abrir un navegador. Ir a la Web: http://atenea.upc.edu

Sistema de ficheros

Linux posee un sistema de ficheros jerárquico, similar a un árbol invertido. Define un punto inicial que denominamos *root* (raíz) y que se representa con el símbolo "/". Dentro del sistema de ficheros, cualquier fichero o carpeta queda identificado de forma única por su camino de acceso desde *root* (camino absoluto). También es posible indicar caminos de accesos relativos a cualquier punto del árbol (camino relativo).

El nombre de un camino relativo de acceso está compuesto por la lista de carpetas que se atraviesan para ir desde la carpeta actual hasta el fichero o carpeta destino.

Ejemplo: Caminos de acceso al fichero *stdio.h*



Camino *absoluto* de acceso:
/users/include/stdio.h

Camino *relativo* de acceso desde *users*:

./include/stdio.h
o
include/stdio.h

desde *bin*:
../include/stdio.h

En el ejemplo anterior, la carpeta *users* se dice que es subcarpeta de "/" y las carpetas *bin*, *include* y *lib* se dice que son subcarpetas de *users*.

Dentro del sistema de ficheros cada estudiante tiene reservada una zona de trabajo, a la que puede acceder directamente al abrir un *Terminal*, y que nos conduce a una carpeta que denominaremos carpeta actual. Dentro de esta carpeta se encuentra la subcarpeta carpeta *Desktop*.

La zona de trabajo también es posible accederla desde el exterior (ejemplo: desde casa) a través del "*Escriptori Virtual*". Para conocer las instrucciones de acceso debes ir a: https://rvd6.upc.edu

Comandos básicos del sistema de ficheros

A través de la ventana de un *Terminal* se dispone de un *shell* o interfaz de comandos que permite ejecutar comandos del sistema.

A continuación, se resumen los comandos básicos necesarios para poder trabajar en las sesiones de laboratorio:

cd *carpeta*	Cambia de carpeta actual
ls *carpeta*	Lista el contenido de una carpeta
mv *fichero_origen fichero_destino*	Renombra o mueve un fichero. También se aplica a carpetas.
mkdir *carpeta*	Crea una carpeta en el camino especificado
rmdir *carpeta*	Borra una carpeta. La carpeta debe estar vacía
rm *fichero*	Borra un fichero
cp *fichero_original fichero_copia*	Copia un fichero
man *comando*	Muestra las páginas de ayuda referentes al comando
tar xvf *fichero.tar*	Extrae el contenido de un fichero comprimido con extensión *.tar*
tar cvf *fichero.tar carpeta*	Comprime una carpeta y genera un fichero *.tar*

El comando *man* se utiliza para consultar los manuales de ayuda de cualquier comando del sistema.

Ejemplo: Muestra todas las opciones del comando *ls*.

man ls

Símbolos especiales en un camino de acceso

Los símbolos "." y ".." también se utilizan con finalidades específicas. El símbolo "." referencia a la carpeta actual. El símbolo ".." referencia a la carpeta que está inmediatamente superior a la actual.

Finalmente, el símbolo "*" también tiene un significado dentro de un camino de acceso: el sistema lo sustituye por "cualquier cadena de caracteres".

Ejemplo: Lista los ficheros con extensión .c que se encuentran almacenados en la carpeta actual.

*ls *.c*

Para acabar, es necesario comentar que la mayoría de los comandos descritos anteriormente se pueden realizar dentro del entorno gráfico que ofrece el sistema. Por ejemplo, para realizar la copia de un fichero, se puede hacer mediante la secuencia típica de: "*copiar - pegar*" o en inglés "*copy – paste"*.

Ejercicio 2

a) Desde la Web de Atenea (http://atenea.upc.edu) dentro de *FONAMENTS D'ORDINADORS* (Entra en Metacurs), en Laboratorio/Sesión 1 descargar el fichero *sesion1.tar* (haciendo clic sobre el nombre). Almacenar dicho fichero en la carpeta *Desktop*. Comprobar la existencia de la descarga desde el entorno gráfico y desde un *Terminal*.

b) Descomprimir el fichero *sesion1.tar*.

c) En un *Terminal*, desde la carpeta *Desktop*, listar el contenido. Comprobar que existe la carpeta *sesion1*.

d) Listar todos los ficheros con extensión .c de la carpeta *sesion1* desde *Desktop*.

e) Cambiar de carpeta actual a *sesion1*.

f) Copiar el fichero *sesion1_ej1.c* a *sesion1_ej1.c.bak.*

g) Debajo de la carpeta *sesion1*, crear una subcarpeta con el nombre: *entrega_sesion1*.

3. Proceso de codificación y prueba de un programa

Independientemente del entorno de programación que se elija, el proceso de codificación y de prueba de un programa está compuesto de 5 etapas: edición, compilación, enlace, ejecución y depuración. La figura 3 esquematiza este proceso.
La etapa de edición consiste en escribir el programa, utilizando cualquier editor de texto, siguiendo las reglas sintácticas del lenguaje de programación elegido. El resultado de esta etapa es un fichero de texto que contiene el código del programa (fichero fuente). La extensión de este fichero indica el lenguaje de programación utilizado (*.c* para códigos en C, *.f* para Fortran, *.cpp* para C++, etc.).
La etapa de compilación determina si hay errores sintácticos en el código. Todos los errores detectados deben ser corregidos. Para ello se utiliza el compilador como herramienta de detección. Una vez compilado el programa (ya no hay errores sintácticos), el compilador genera un fichero que contiene el código objeto (fichero con extensión *.o* o *.obj*).
La etapa de enlace combina todos los ficheros *.o* o *.obj* asociados a un programa (ya que el programa puede tener varios ficheros fuente o utilizar funciones propias del lenguaje)

para generar un único fichero ejecutable que normalmente tiene extensión *.out*, *.exe* o simplemente no tiene extensión.
La siguiente etapa consiste en ejecutar el programa para determinar su correcto funcionamiento. Para esto, es necesario probar exhaustivamente el programa utilizando diferentes datos de entrada. En el caso de detectar algún error semántico será necesario pasar a la etapa de depuración.
El depurador es una herramienta que permite observar el comportamiento de un programa paso a paso y facilitar así la detección de errores de ejecución. De esta forma, cuando se detecta un error de ejecución hay que volver a la etapa de edición para corregir el programa y nuevamente volver a compilar, enlazar y ejecutar. Una vez que el programa funciona correctamente se puede dar por finalizado el proceso de codificación y de prueba.
A continuación, explicaremos cómo se realiza cada una de estas etapas para el lenguaje C utilizando las herramientas que proporciona la distribución libre *GNU/Linux*.

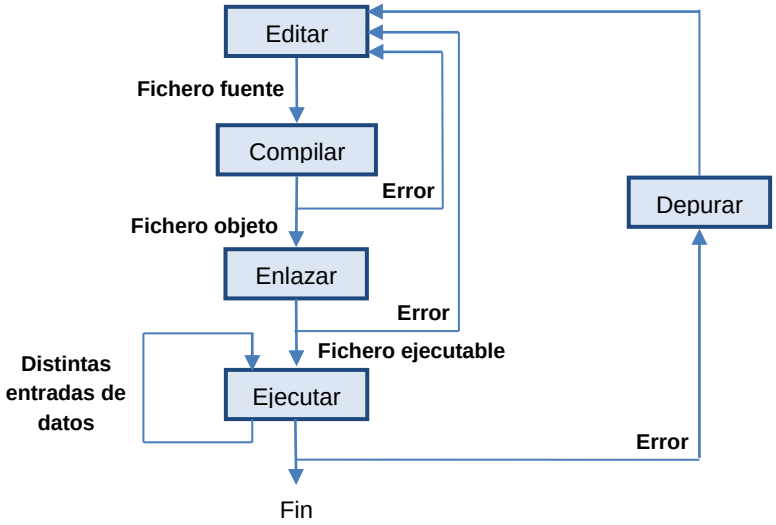


Figura 3. Proceso de codificación y prueba de un programa

4. Editar un programa en C

El entorno de trabajo del laboratorio ofrece varias opciones para editar ficheros de texto. Se recomienda la utilización del editor *gedit* dada su facilidad de uso.

Creación

Pasos para la creación de un fichero fuente:

a) Para crear un fichero fuente nuevo, usaremos el comando *gedit* seguido del nombre del fichero (con extensión *.c*) desde un *Terminal*. Por ejemplo, si queremos crear un fichero fuente llamado *nuevo_programa.c*, debemos escribir un *Terminal* lo siguiente:

```
gedit nuevo_programa.c &
```

b) Luego, escribimos el programa dentro del editor de texto y lo salvamos.

Modificación

Si queremos modificar un fichero ya existente que se llama *mi_prograna.c* será necesario abrirlo utilizando alguna de las siguientes alternativas:

- Desde la ventana de la carpeta donde se encuentra el fichero, dar doble clic sobre el fichero a modificar.
- En un *Terminal*, desde la carpeta donde se encuentra el fichero, mediante el comando:

```
gedit mi_programa.c &
```

Ejercicio 3

a) Editar el fichero *sesion1_ej1.c* ya existente en la carpeta *sesion1*. Modificar la sentencia:

a = 10;

y cambiarla por:

a = b;

Finalmente, salvar el fichero.

b) Crear un fichero nuevo (llamado *sesion1_ej2.c*) con el editor de texto y copiar el siguiente código:

```
/* Autores: Nombre1 Apellido1  */
/*          Nombre2 Apellido2  */

#include <stdio.h>

int main()
{
    printf("\n Hola Mundo ! \n");
}
```

5. Compilar un programa en C

Para poder ejecutar un programa es necesario compilarlo previamente. Un compilador es una herramienta capaz de detectar los errores sintácticos de un programa y, en caso de no haber errores, generar un fichero ejecutable. El compilador NO corrige errores y no asegura que el ejecutable generado funcione correctamente.

Para compilar los programas escritos en C se utilizará el compilador *gcc* ya instalado en el entorno de trabajo del laboratorio y accesible desde un *Terminal*.

Las opciones que acepta este compilador son diversas. Para conocerlas se ejecuta el compilador con la opción *–h* (comando: *gcc –h*), o bien se ejecuta el comando *man gcc*.

A continuación, se describen las opciones más utilizadas:

-h	muestra el conjunto de opciones aceptadas por el compilador.
-c	compila los ficheros .c y genera ficheros objeto .o (no genera ejecutable)
-g	compila y genera información necesaria para la depuración
-o	genera el fichero ejecutable con el nombre especificado

De esta forma, para compilar el programa fuente *mi_programa.c* y generar el fichero ejecutable con nombre *mi_programa*, ejecutaremos el siguiente comando:

```
gcc mi_programa.c –o mi_programa
```

Nota: Observe que el fichero fuente debe tener extensión *.c*, mientras que el fichero ejecutable no lleva extensión. Si por error, coloca como nombre del fichero ejecutable *mi_programa.c*, perderá el fichero fuente.

Si la compilación ha sido correcta (el compilador no ha detectado errores de sintaxis), la línea de comandos aparecerá otra vez sin indicar nada. De lo contrario, si el compilador ha detectado algún error, aparecerá en pantalla la lista de errores. Para cada error se indica la línea donde se encuentra el error y una breve descripción (que intenta indicar el origen del fallo).

6. Enlazar un programa en C

Cuando el programa utiliza funciones propias del lenguaje o varios ficheros fuente, el enlazador es el encargado de combinar todos los ficheros correspondientes y generar un único fichero ejecutable. El comando *gcc* permite compilar y enlazar en un solo paso o en pasos separados.

Para compilar y enlazar en un solo paso el fichero fuente *mi_programa.c* es necesario, desde un *Terminal* (y en la carpeta donde se encuentre el fichero), ejecutar el siguiente comando:

```
gcc mi_programa.c –o mi_programa
```

Si el programa está compuesto por más de un fichero fuente, por ejemplo *colours.c* y *mi_programa.c*, para compilar y enlazar en un solo paso tendríamos que hacer lo siguiente:

```
gcc colours.c mi_programa.c –o mi_programa
```

Observe que simplemente se colocan todos los ficheros fuentes en la línea de comando para generar el fichero ejecutable.

Para compilar y enlazar en dos pasos separados ejecutaríamos los siguientes comandos:

```
gcc –c mi_programa.c        (genera fichero objeto: mi_programa.o)
gcc –o mi_programa mi_programa.o
                            (genera fichero ejecutable: mi_programa)
```

Si el programa está compuesto por más de un fichero fuente, por ejemplo *colours.c* y *mi_programa.c*, para compilar y enlazar en dos pasos tendríamos que hacer lo siguiente:

```
gcc –c colours.c mi_programa.c
(genera ficheros objeto: mi_programa.o y colours.o)

gcc –o mi_programa colours.o mi_programa.o
(genera fichero ejecutable: mi_programa)
```

7. Ejecutar un programa en C

Si el compilador y el enlazador no han detectado errores, se generará el fichero ejecutable con el nombre que se haya puesto a continuación de la opción "-*o*" (en el ejemplo de la sección anterior: *mi_programa*). Para ejecutarlo desde el *Terminal* será necesario escribir el nombre del fichero ejecutable precedido de "./", como se indica a continuación:

```
./mi_programa
```

8. Depurar un programa C

Si el programa ejecutado no funciona correctamente será necesario depurarlo. Un depurador es una herramienta que permite ejecutar un programa paso a paso (instrucción a instrucción) para facilitar la detección de errores de ejecución.

Para depurar un programa están disponibles los programas: *gdb* y *ddd*. Se trata del mismo depurador, sólo que el segundo incorpora una interfaz gráfica. Debido a que el uso de *gdb* se realiza a través de una línea de comandos, en este curso utilizamos el *ddd* ya que provee una interfaz gráfica mucho más amigable y fácil de usar.

De esta forma, depurar el programa *mi_programa* es necesario compilar primero con la opción *–g* de la siguiente manera:

```
gcc –g mi_programa.c –o mi_programa
```

y a continuación abrir el depurador *ddd* indicando el nombre del fichero ejecutable:

```
ddd mi_programa &
```

Nota: En esta sesión de laboratorio no explicaremos como utilizar los comandos básicos del *ddd* para depurar nuestros códigos, lo haremos en las próximas sesiones de laboratorio. Sin embargo, para más detalle puede consultarse el documento publicado en la Web de Atenea, dentro de *FONAMENTS D'ORDINADORS*, sección de Laboratorio, documento "*Depurar programas usando el ddd*".

Ejercicio 4

a) Compilar el programa *sesion1_ej2.c* con opción para que pueda ser depurado. El ejecutable tiene que tener el nombre *sesion1_ej2*.

b) Abrir el depurador *ddd* con el fichero ejecutable *sesion1_ej2*.

c) Cerrar el depurador.

d) Ejecutar el fichero *sesion1_ej2*.

e) Abrir el depurador *ddd* con el fichero ejecutable *sesion1_ej2*.

f) Cerrar el depurador.

g) Ejecutar el fichero *sesion1_ej2*.

h) Copiar los ficheros *sesion1_ej2.c* y *sesion1_ej1.c* en la carpeta *entrega_sesion1*.

i) En el *Terminal*, desde la carpeta *sesion1*, comprimir la carpeta *entrega_sesion1* utilizando el siguiente comando: *tar cvf entrega_sesion1.tar entrega_sesion1*

j) En la Web de Atenea, dentro de *FONAMENTS D'ORDINADORS* (Entra en tu curso), ir a *Laboratorio/Sesión 1/Primera Entrega*. Allí realiza la entrega del fichero *entrega_sesion1.tar*.

9. Etapas que intervienen en la elaboración de un programa

Anteriormente, hemos explicado el proceso de codificación y de prueba de un programa utilizando las herramientas correspondientes. Sin embargo, previo a este proceso es necesario realizar el análisis del problema y el diseño del programa, entre otros. La figura 4 muestra todas las etapas que intervienen en la elaboración de un programa. Observe que las etapas de codificación y prueba incluyen un resumen de los comandos de compilación, ejecución y depuración vistas en la sección anterior.

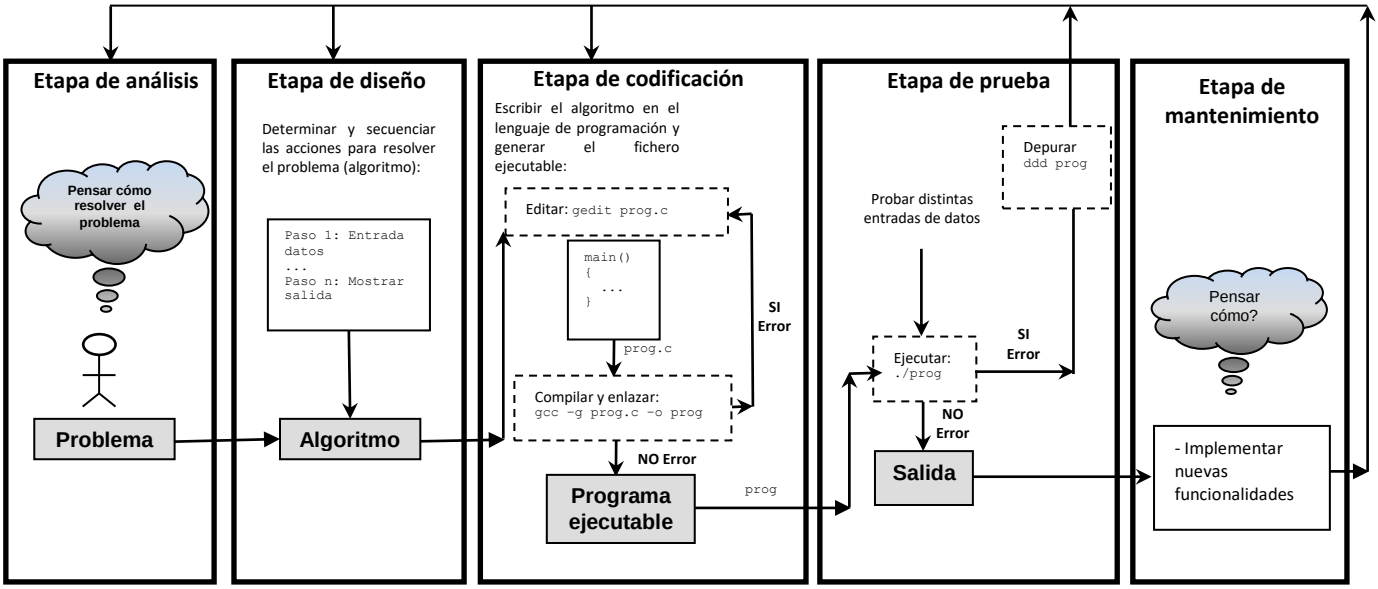


Figura 4. Etapas que intervienen en la elaboración de un programa

En la etapa de análisis se define el problema, determinando las entradas requeridas y la salida que debe generar el programa; además, en esta etapa, se debe comprender correctamente el problema. Seguidamente, en la etapa de diseño, se especifican los pasos necesarios para resolver el problema. El conjunto de estos pasos, colocados en un orden determinado y escritos utilizando un lenguaje natural, se denomina algoritmo. Finalizado el diseño del algoritmo se pasa a la etapa de codificación, donde se escribe el algoritmo en un lenguaje de programación determinado, obteniendo el programa. En esta etapa se verifica si el programa ha sido escrito utilizando las reglas de sintaxis del lenguaje de programación. Esta verificación se realiza utilizando el compilador del lenguaje. Si el programa tiene errores de sintaxis se editará el programa para solucionarlos. Una vez compilado el programa (ya no hay errores sintácticos), el compilador genera el fichero ejecutable. Obtenido este fichero, pasamos a la etapa de prueba donde se realizarán pruebas exhaustivas del programa utilizando distintos datos

de entrada, para verificar su correcto funcionamiento. En el caso que no sea correcta la salida del programa para algún dato de entrada, entonces se utilizará el depurador para localizar el error semántico, y se volverá a las etapas de análisis, diseño y/o codificación para incorporar la solución encontrada y resolver el problema semántico hallado en el programa. Finalizada la etapa de prueba, pasamos a la etapa de mantenimiento para diseñar e incorporar nuevas funcionalidades al programa.

A continuación, se muestra un ejemplo que ilustra cómo poner en práctica las etapas que intervienen en la elaboración de un programa. Para hacer esto, suponemos que necesitamos realizar un programa que dibuje en la pantalla del ordenador un rectángulo de longitud de lado 100 puntos y de altura 50 puntos, utilizando una librería gráfica (*librería_grafica.h*). De esta forma, en la etapa de análisis pensamos cómo resolver el problema considerando detalles cómo: la posición inicial en la que pondremos el lápiz en la pantalla, y la dirección en la que dibujaremos la base del rectángulo (vertical u horizontal). En la etapa de diseño escribimos el algoritmo (utilizando un lenguaje natural), tal y como lo muestra la figura 5. Observe que la figura 5.a muestra una versión extendida del algoritmo mientras que la figura 5.b muestra la versión compacta del mismo.

```
1) Inicializar la posición del lápiz
2) Inicializar la dirección del lápiz
3) Trazar 100 puntos
4) Girar 90°
5) Trazar 50 puntos
6) Girar 90°
7) Trazar 100 puntos
8) Girar 90°
9) Trazar 50 puntos
10)Girar 90°
```

(a)

```
1) Inicializar la posición del lápiz
2) Inicializar la dirección del lápiz
   Repetir
      3) Trazar 100 puntos
      4) Girar 90°
      5) Trazar 50 puntos
      6) Girar 90°
   hasta 2 veces
```

(b)

Figura 5. Algoritmo para dibujar un rectángulo: (a) Versión extendida; (b) Versión compacta

En la etapa de codificación escribimos el algoritmo en el lenguaje C, tal y como lo muestra la figura 6.

```
#include <librería_grafica.h>
main()
{
   inic_lapiz();
   inic_direc();
   dibujar_linea(100);
   girar(90);
   dibujar_linea(50);
   girar(90);
   dibujar_linea(100);
   girar(90);
   dibujar_linea(50);
   girar(90);
}
```

(a)

```
#include <librería_grafica.h>
main()
{
   int i;

   inic_lapiz();
   inic_direc();
   for(i=0; i<2; i++)
   {
     dibujar_linea(100);
     girar(90);
     dibujar_linea(50);
     girar(90);
   }
}
```

(b)

Figura 6. Programa en C para dibujar un rectángulo: (a) Versión extendida; (b) Versión compacta

A partir de aquí, generamos el fichero ejecutable y comprobamos que dibujamos un rectángulo en la pantalla del ordenador (etapa de prueba).