UNIVERSIDAD DE OVIEDO
Departamento de Ingeniería Eléctrica,
Electrónica, de Computadores y Sistemas

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

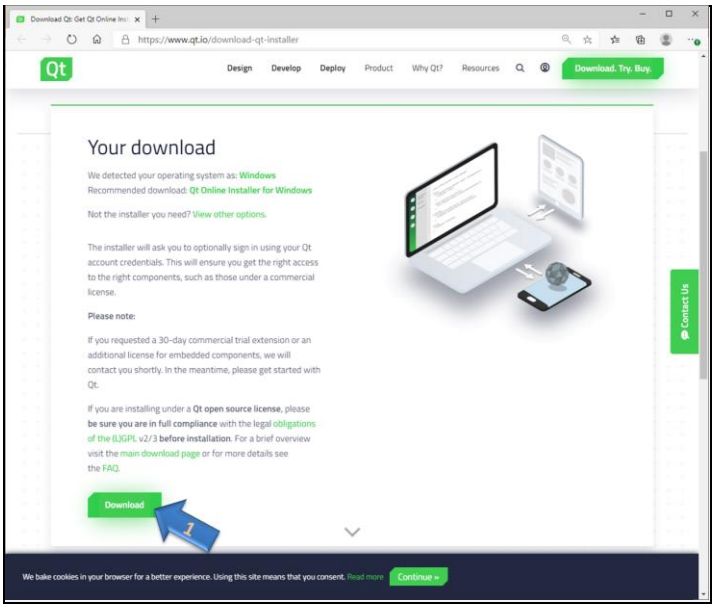
Instalación y uso de QtCreator para programación en lenguaje C (modo consola)

Ignacio Alvarez García – Septiembre 2022

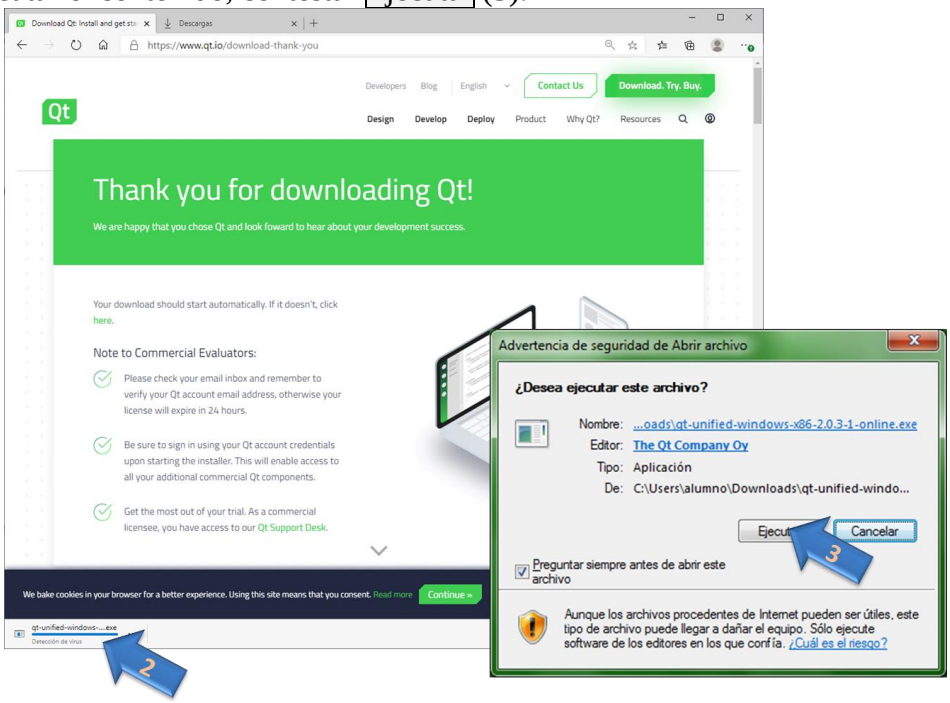
INDICE

1. Instalación

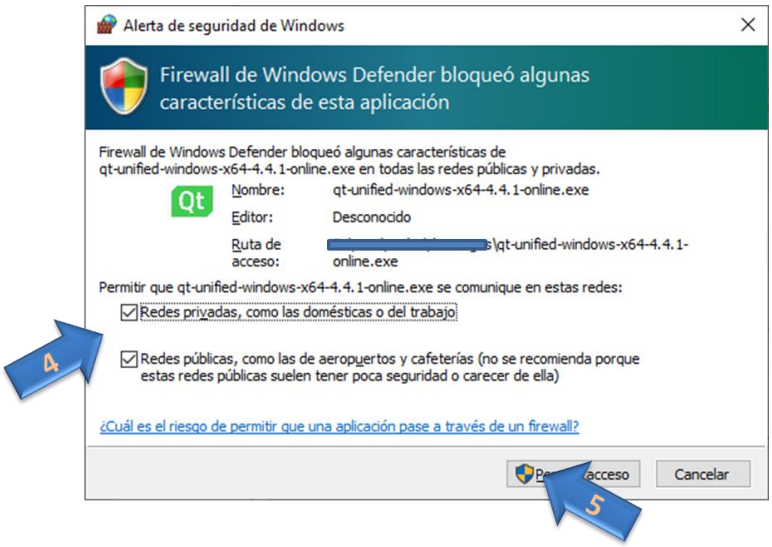
El instalador de descarga de la página https://www.qt.io/download-qt-installer, presionando el botón Download (1).



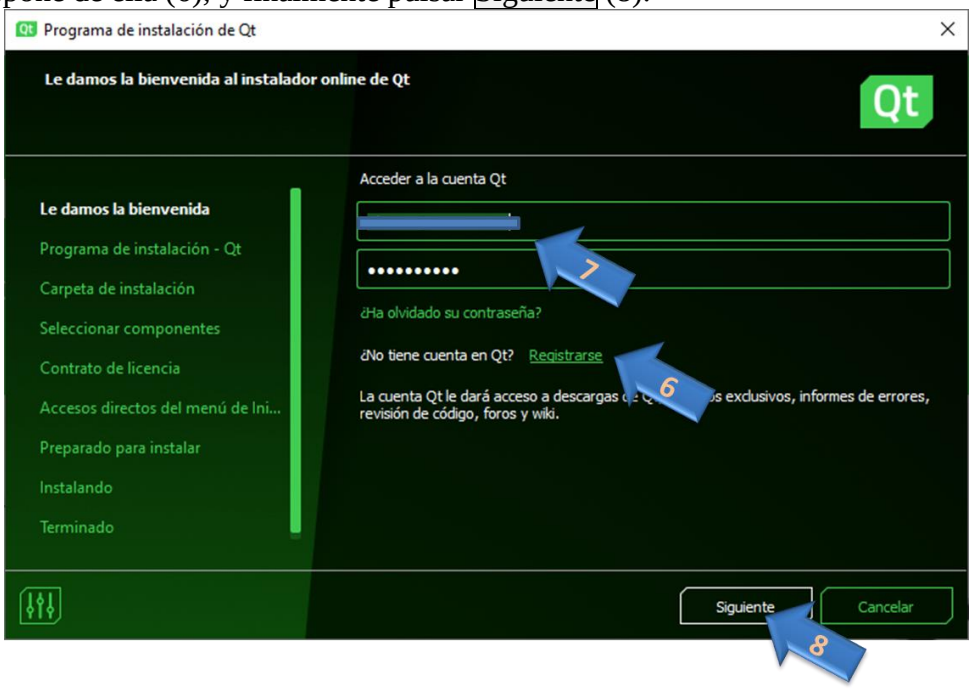
Una vez finalizada la descarga, abrir el archivo obtenido (2). Windows preguntará si se desea ejecutar el contenido, contestar Ejecutar (3).



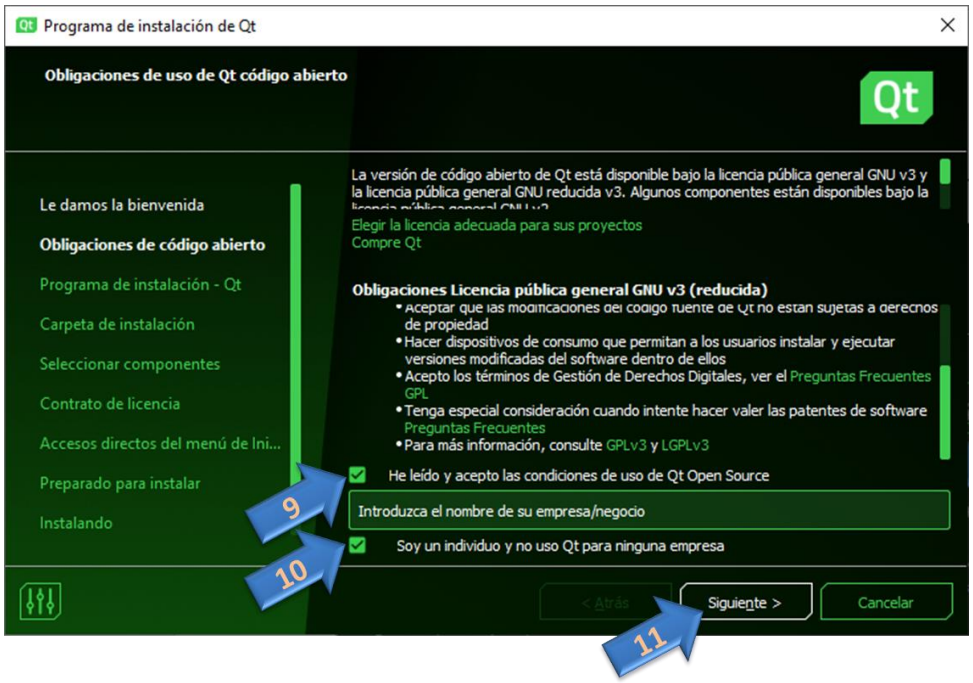
También es posible, en función de la configuración del cortafuegos (Firewall), que el Sistema Operativo solicite autorización para la comunicación por red para el instalador. Se debe seleccionar (4) y aceptar (5).



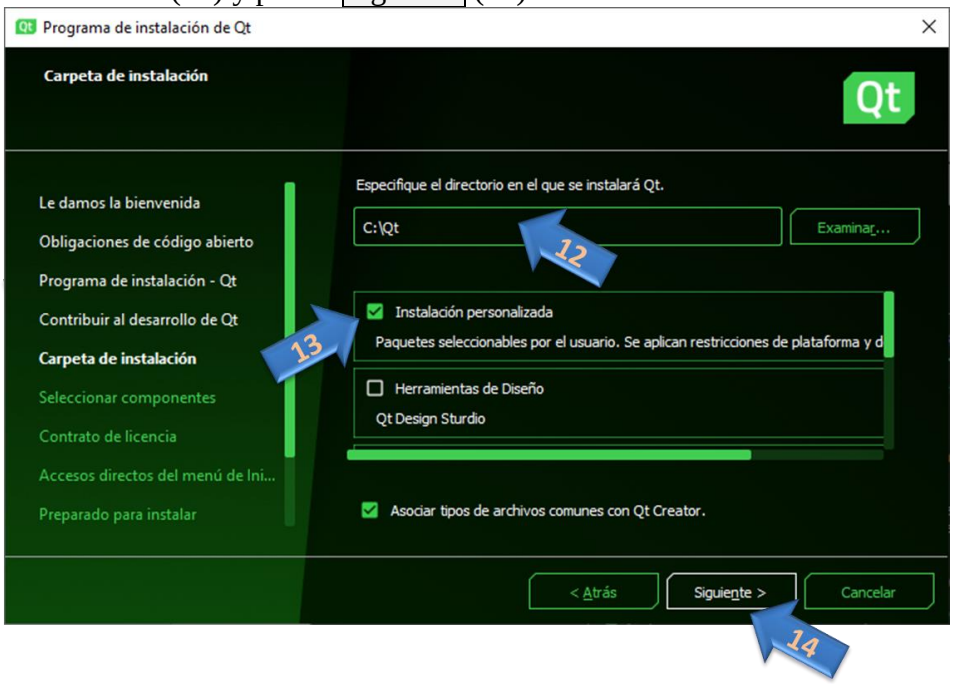
En la pantalla de bienvenida, introducir datos de cuenta de Qt (7) o bien crear una nueva si no se dispone de ella (6), y finalmente pulsar Siguiente (8).



Aceptar los términos de la licencia Open Source Qt (9) e indicar que el uso es individual (10), pasar a siguiente (11).



A continuación, tras pulsar Siguiente comenzará una descarga inicial, tras la cual se solicitará el directorio destino de la instalación (12) (**atención**, **para evitar problemas** en la utilización posterior **no utilice** un camino de directorio que incluya **espacios en blanco** y/o caracteres específicos del alfabeto español como **ñ/Ñ** y letras **con tilde**). Elegir Instalación Personalizada (13) y pulsar Siguiente (14).



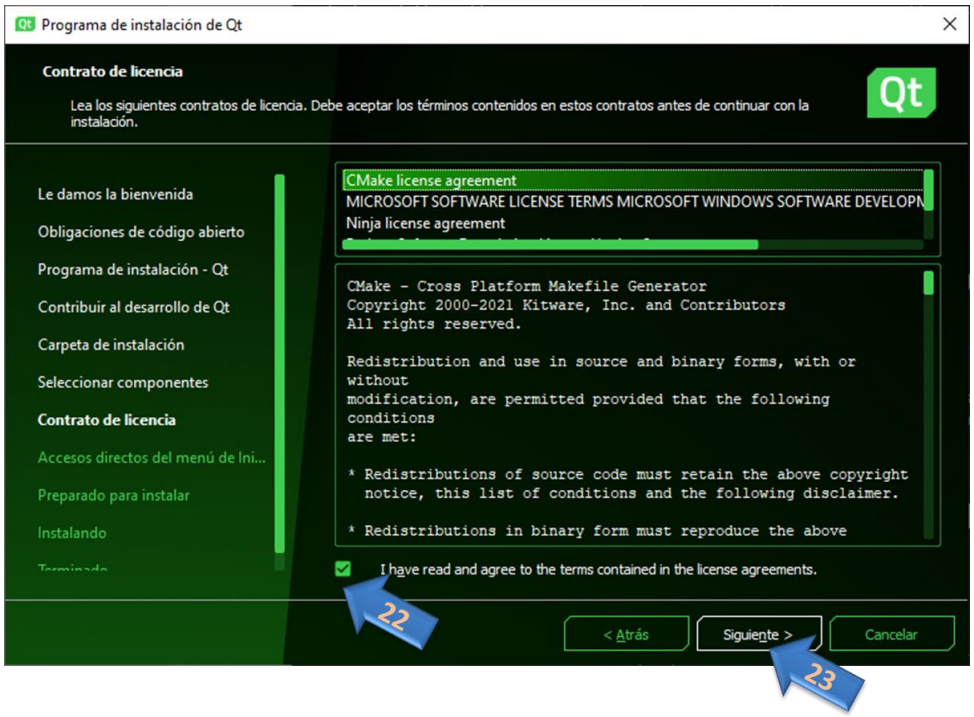
A continuación se presentan las opciones de instalación. **Asegúrese de seleccionar** al menos los siguientes elementos (15 a 20) antes de proceder con Siguiente (21):

- Qt → **Qt 6.3.1** (o versión más reciente que no sea Beta) con **MinGW 64-bit** (15)
- Qt → Developer and designer tools → **Qt Creator** última.versión.disponible (16)
  Debugging Tools (17)

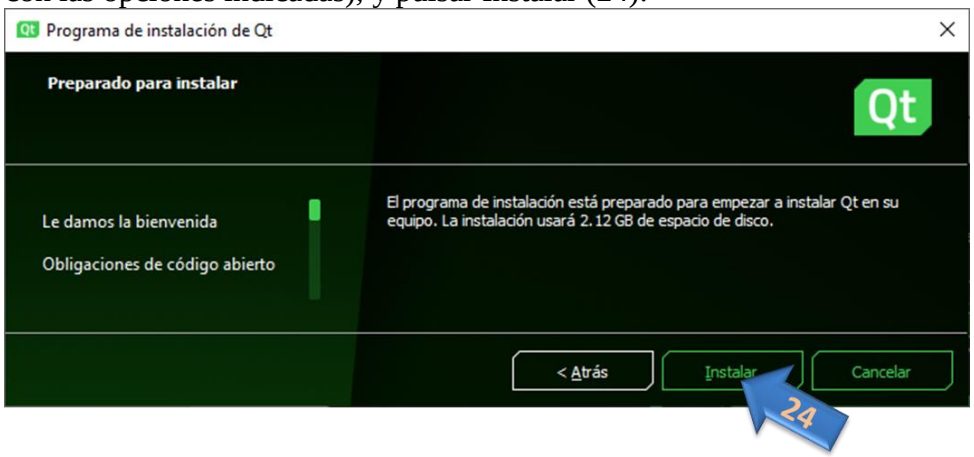
MinGW misma versión que 15 (18)
CMake (19)
Ninja (20)

Otros componentes pueden ser de utilidad para realizar aplicaciones para otras plataformas, se pueden añadir también si se desea, pero no serán necesarias para el desarrollo de la asignatura (Qt 6.3.1 para Android o WebAssembly, librerías adicionales, Design Studio, etc.).

A continuación se presentan acuerdos de licencia para cada uno de los elementos seleccionados que proceden de suministradores diferentes. Es necesario aceptar todos (22) y proceder con Siguiente (23).



Aceptar el resto de pantallas para proceder a la instalación: ubicación de accesos directos de menú inicio, información de espacio necesario según las opciones de instalación (2.12Gb con las opciones indicadas), y pulsar Instalar (24).

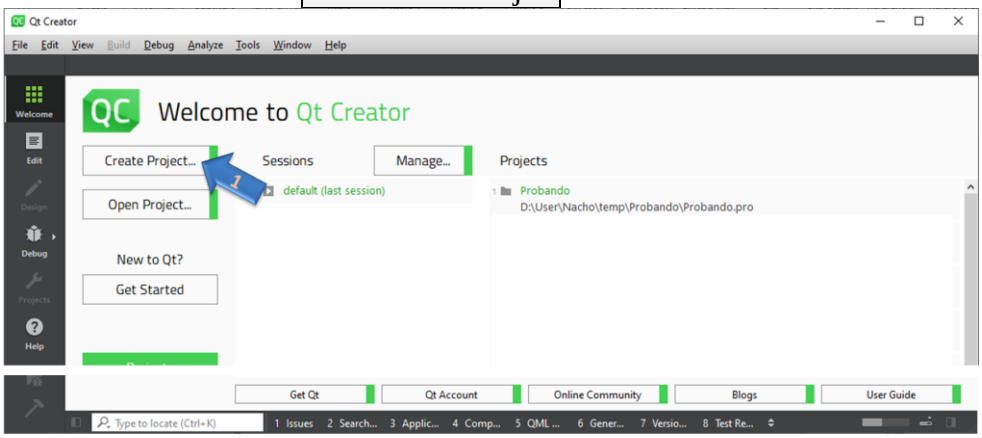


La instalación descargará los componentes y los copiará en el directorio creado, tomando unos 20-30 min para completar la operación. Una vez finalizada, dispondremos del programa Qt Creator en nuestro equipo.

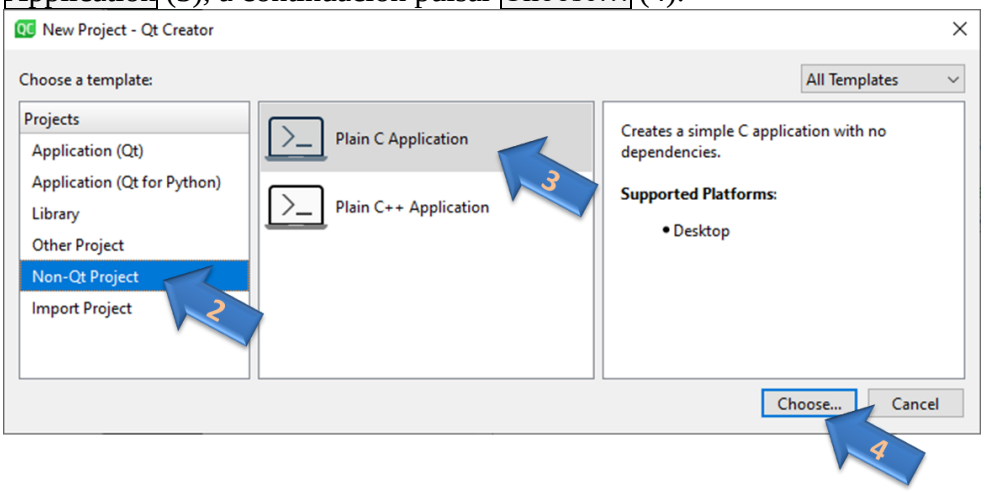
2. Crear un programa en modo consola

Ejecutar el programa Qt Creator y realizar los siguientes pasos para crear un programa en lenguaje C:

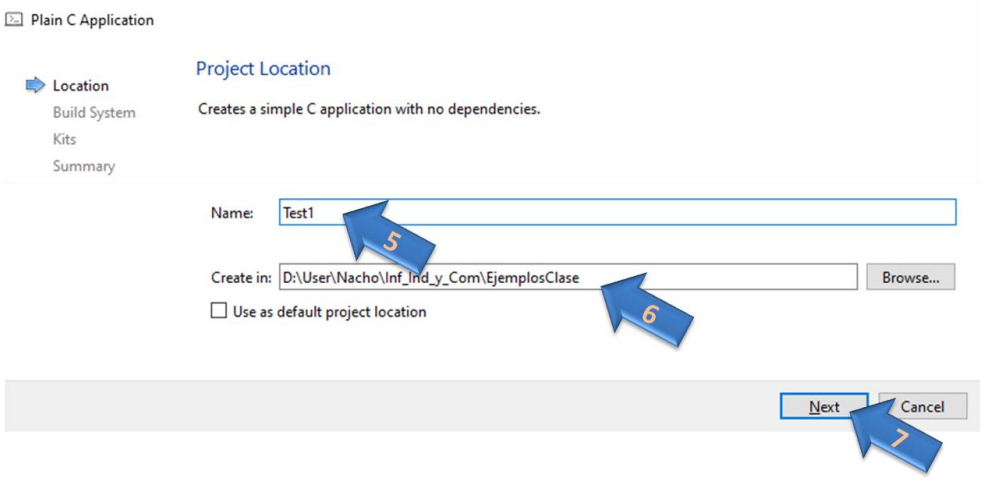
- Crear nuevo proyecto pulsando Create Project (1) en la ventana de bienvenida, o bien mediante el menú File → New Project.



- Seleccionar las opciones de proyecto Non-Qt Project (2), y dentro de éste Plain C Application (3), a continuación pulsar Choose… (4).



- Elegir nombre para el proyecto (5) y ubicación (6), pulsar Next (7). **Importante que ni el nombre de proyecto ni la ubicación contengan espacios y/o caracteres específicos del idioma español (ñ/Ñ, tildes):**



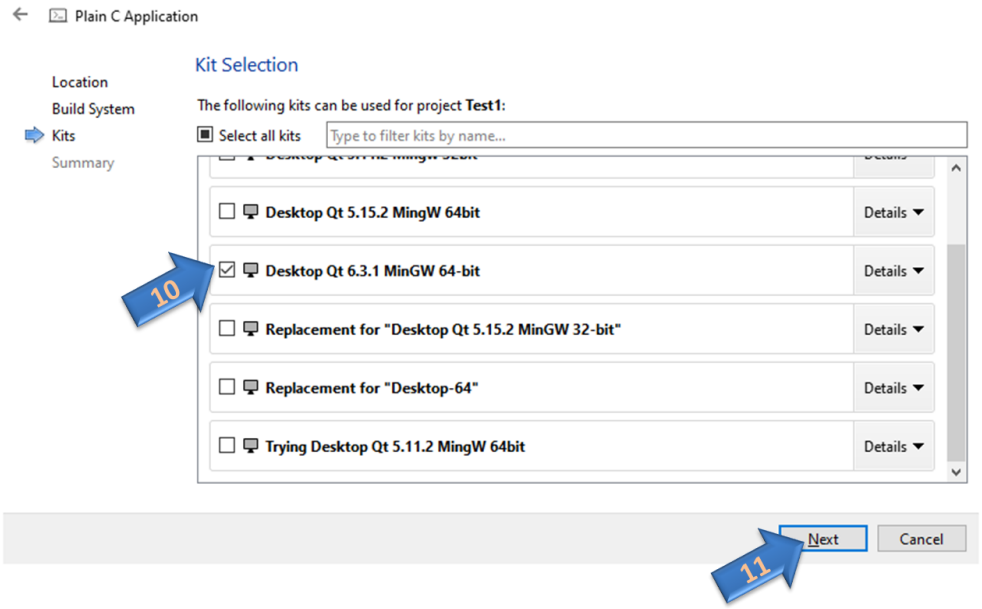
A partir de este punto se debe elegir el sistema de gestión de la compilación, hay 2 opciones principales:

- **CMake**: genérico (no exclusivo de Qt) pero no bien integrado con Qt-Creator. En la documentación se indica que se prefiere su uso para siguientes versiones.
- **qmake**: específico de Qt, bien integrado, pero posiblemente no esté disponible o actualizado en siguientes versiones.

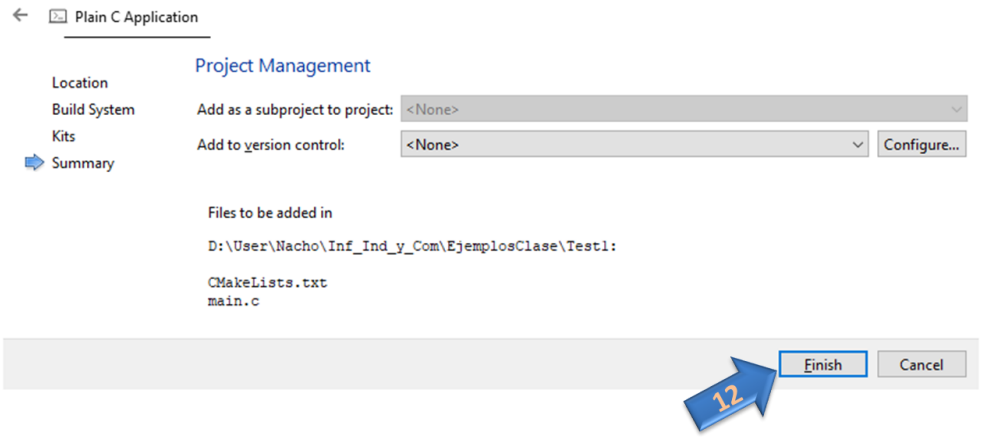
Los siguientes apartados serán muy similares sea cual sea el sistema de gestión de la compilación (build system) elegido.

- Elegir subsistema de compilación. En este caso seleccionamos CMake o qmake (8) y pulsar Next (9).

- Si en las opciones de instalación sólo se eligió Qt 6.3.1 MingGW 64 bit, aparecerá un único Kit de compilación. En caso contrario, hay que seleccionar este Kit (10) y pulsar Next (11).

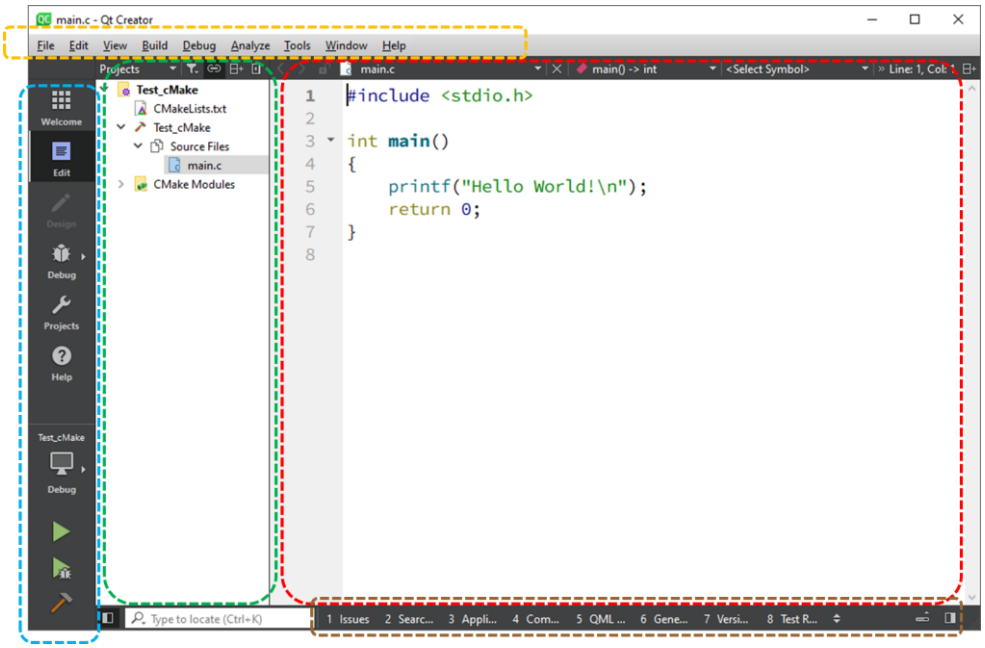


- Se presentará la información final. Aceptar y finalizar pulsando Finish (12).



2.1. Editar y ejecutar un programa de prueba

En cualquiera de las dos opciones (CMake o qmake), tras realizar todos los pasos aparecerá una ventana con varias zonas:

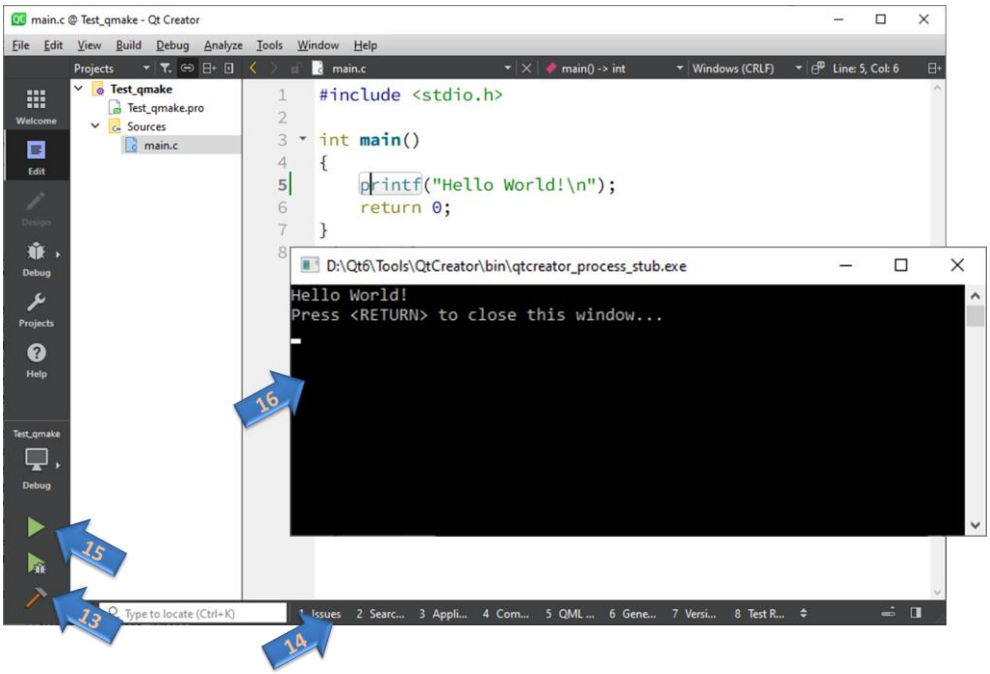


- Menú gráfico a la izquierda (recuadrado en azul). Es una selección rápida para el modo de trabajo y las operaciones más habituales.
- Menú textual superior (recuadrado en amarillo), típico de todas las aplicaciones con interfaz de usuario.
- Ventana de proyectos (recuadrada en verde), muestra y permite seleccionar los proyectos abiertos y los archivos asociados a los mismos.
- Ventana de edición (recuadrada en rojo), permite escribir el texto de los diferentes archivos del proyecto.
- Menú y ventanas de información (recuadrada en marrón), permite visualizar diferentes salidas del entorno y el programa.

Se puede ver que se ha agregado en la ventana de proyecto (en verde en la imagen anterior):

- Un archivo de configuración del proyecto: CMakeLists.txt si se ha seleccionado CMake, y nombre_de_proyecto.pro si se ha seleccionado qmake. En ambos casos se trata de archivos de texto (editables) con el mismo cometido: indicar los pasos para proceder a la generación del código ejecutable. Su diferencia radica en el formato, ya que serán procesados por programas distintos. Para las primeras pruebas, no es necesario editarlos.
- Un archivo con el código fuente (main.c), ya abierto en la ventana de edición. Este archivo se puede modificar para que contenga el código deseado para nuestra aplicación.

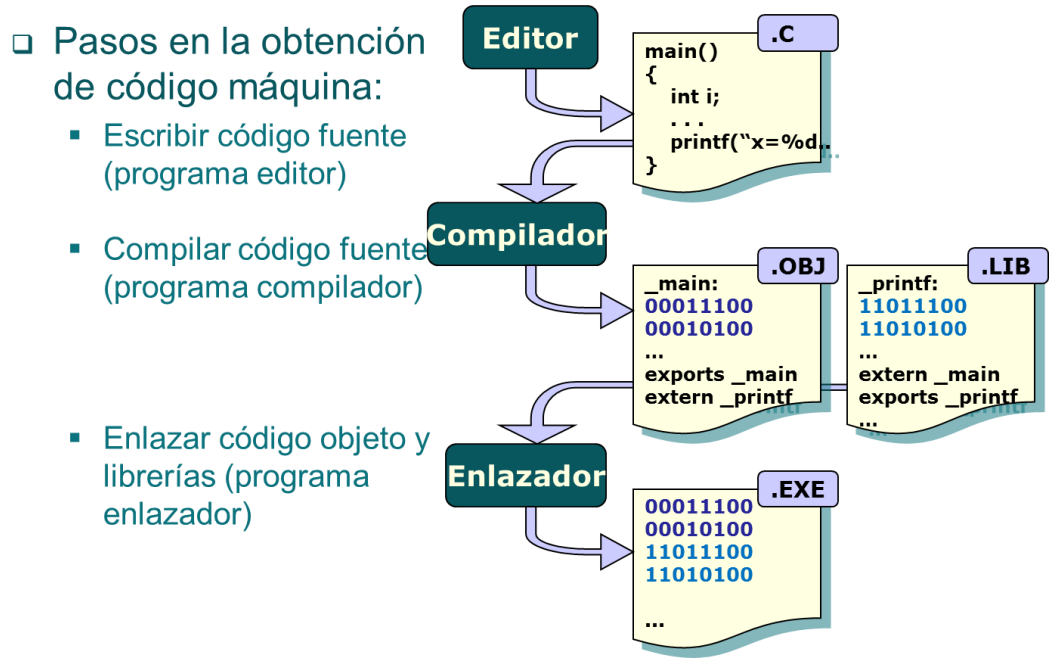
- Una vez escrito y guardado (File → Save All), se procede a la compilación pulsando el icono martillo (Build Project) que aparece abajo a la izquierda (13) (opción de menú alternativa Build → Build All). El resultado de la compilación se comprueba pulsando sobre la ventana Issues (14). Si no hay errores, se puede ejecutar el programa pulsando el triángulo verde Run (15) (opción de menú alternativa Build → Run). Se mostrará una nueva ventana de consola con la salida del programa (16).


main.c @ Test_qmake - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
Projects
Test_qmake
Test_qmake.pro
Sources
main.c
#include <stdio.h>
int main()
{
printf("Hello World!\n");
return 0;
}
D:\Qt6\Tools\QtCreator\bin\qtcreator_process_stub.exe
Hello World!
Press <RETURN> to close this window...
Welcome
Edit
Design
Debug
Projects
Help
Test_qmake
Debug
16
15
13
14

3. Edición, compilación y enlazado

3.1. Pasos en la generación de un programa

El lenguaje C es compilado, esto significa que el código fuente que escribimos debe ser traducido por un compilador (compiler) a una secuencia de instrucciones de máquina, y unido mediante un enlazador (linker) a otros trozos precompilados (librerías) para producir un archivo ejecutable que se puede lanzar para producir los resultados deseados.



Más detalles sobre los programas involucrados en estos pasos, y cómo son invocados por Qt-Creator, en el apartado 6 (Entre bastidores).

3.2. Edición del código

A la hora de escribir el código fuente correspondiente al programa que se desea realizar, se debe prestar especial atención a:

- Sintaxis: el compilador sólo aceptará el código si su sintaxis es correcta.
- Indentación: estructurar los bloques en columnas (usar Tab) de forma que el código sea fácilmente legible.
- Comentarios: facilitan la comprensión de lo que se ha escrito

El entorno facilita la visualización de lo que se está haciendo correctamente:

- Las palabras clave de C aparecerán en color verde grisáceo.
- Los comentarios (texto que no se compila) aparecerán en verde intenso.
- Las cadenas de texto aparecerán en color verde intenso.
- Las directivas y constantes en color azul.
- El texto que no parece estar conforme a la sintaxis aparece subrayado en rojo

Ejemplo **(no copiar/pegar directamente del PDF, pueden aparecer caracteres invisibles que el compilador no entienda):**

```
#include <stdio.h>
#include <stdlib.h>

#define TAM_TABLA      5

int Factorial(int n)
// Función Factorial(). Calcula el factorial de un nº entero
// Parámetros de entrada:
// n (int): entero del que queremos calcular el factorial
// Valor devuelto:
// (int): entero con el valor del factorial
{
    int i,result;
    for (i=1,result=1;i<n;i++)
        result=result*i;
    return result;
}

main()
// Programa principal
{
    int i;
    int x[TAM_TABLA],fact[TAM_TABLA];

    for (i=0;i<TAM_TABLA;i++)
    {
        printf("Introduzca x[%d] = ",i);
        scanf("%d",&x[i]);
        fact[i]=Factorial(x[i]);
        printf("El factorial de %d es %d",x[i],fact[i]);
    }
}
```

Una vez escrito y comprobado el código, salvar el archivo. En caso de archivos grandes, se debe salvar a intervalos regulares para evitar pérdida de información en caso de fallo en el ordenador o la alimentación.

3.3. Compilar y enlazar

Tras la creación del código, usar la opción de menú Build -> Build Project (Ctrl-B) o sobre el icono martillo de la parte inferior izquierda para invocar al compilador.
El compilador intentará producir el código máquina a partir del texto fuente, pero no podrá si hay errores sintácticos en el código.
Se debe comprobar que no se producen errores en las ventanas 1-Issues o 4-Compile Output , ambas accesibles en la zona inferior de la ventana de trabajo.
Para todos los errores, en la ventana indicará el archivo y la línea de código donde se detectó. Haciendo doble-click sobre el error, nos lleva automáticamente a su posición en la ventana de edición.
¡ATENCION! El compilador no sabe lo que pretendíamos escribir, sólo es capaz de detectar que el texto no cumple la sintaxis y las reglas del lenguaje C. No siempre el texto y la posición del error indican exactamente lo que se debe corregir.

3.4. Errores más típicos de compilación

- Errores de sintaxis: falta de ; al final de una sentencia, apertura y cierre descompensado de paréntesis, corchetes, comillas, etc.

```
scanf("%d",&x[i])    // Falta el ;
fact[i]=Factorial(x[i]);
```

```
...\principal.c:29: error: expected ';' before 'fact'
  fact[i]=Factorial(x[i]);
  ^
```

- Nombre de variable o palabra clave incorrecto. Recordar que el compilador distingue mayúsculas de minúsculas. Ejemplo:

```
result=resul*i;    // Falta la t en el 2° result
```

```
...\principal.c:15: error: 'resul' undeclared (first use in this function)
  result=resul*i;
         ^
```

- Utilización incorrecta de un operador (falta de operandos, operandos de tipo inadecuado, etc.). Ejemplo:

```
result=result*;    // Falta 2° operando para el producto
```

```
...\principal.c:15: error: expected expression before ';' token
  result=result*;
               ^
```

3.5. Advertencias (warning) más típicas en compilación

- Declaración de una variable que no se utiliza después en el código:

```
int z; // Ninguna instrucción hace después referencia a z
```

```
...\principal.c:23: warning: unused variable 'z' [-Wunused-variable]
int z;
    ^
```

- Uso de una variable sin inicializar. Ejemplo:

```
int a,b;
a=b+2;       // al no dar antes valor a b, no se sabe lo
             // que valdrá a tras ejecutar la instrucción
```

```
...\principal.c:23: warning: 'b' is used uninitialized in this function [-Wuninitialized]
a=b+2;
  ^
```

3.6. Errores más típicos de enlazado (link)

El enlazado sólo se realiza si la compilación ha sido correcta. Los errores de enlazado siempre terminan con uno final:

```
collect2.exe:-1: error: error: ld returned 1 exit status
```

- Falta de función main()

```
crt0_c.c:-1: error: undefined reference to `WinMain@16'
collect2.exe:-1: error: error: ld returned 1 exit status
```

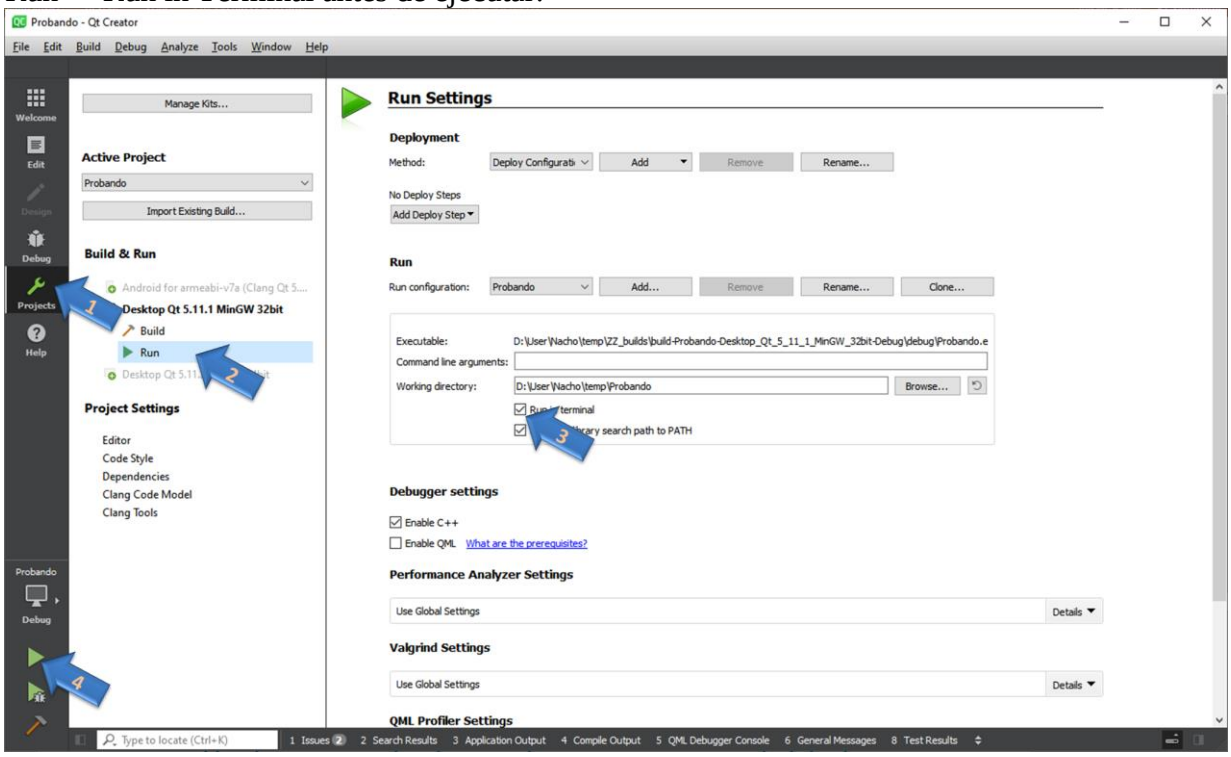
- Nombre de función incorrecto (recordar que el compilador distingue mayúsculas de minúsculas), suele venir acompañado de un warning. Ejemplo:

```
fact[i]=factorial(x[i]); // La función está definida como
                         // Factorial() – F mayúscula
```

⚠ ….\principal.c:32: warning: implicit declaration of function 'factorial' [-Wimplicit-function-declaration]
fact[i]=factorial(x[i]);
^
❗ …\principal.c:32: error: undefined reference to `factorial'
❗ collect2.exe:-1: error: error: ld returned 1 exit status

4. Ejecutar el programa

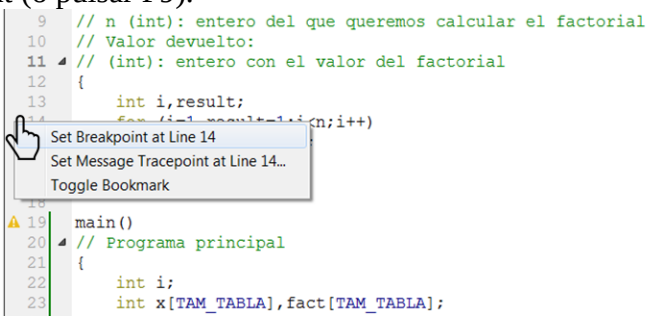
Probar la ejecución del programa mediante la selección de la opción de menú Build ->Run (Ctrl-R) o el icono triángulo (abajo a la izquierda, el que no lleva un 'bicho' encima).
En ese momento se desplegará una nueva ventana, en modo consola, en la que aparecerá el resultado de la ejecución programa.
En caso de que no aparezca una ventana, seleccionar la opción: Projects -> Build & Run -> Run -> Run in Terminal antes de ejecutar:



4.1. Depuración de errores en ejecución

El hecho de que un programa compile correctamente **no implica que haga lo que se espera**. En caso de que la ejecución no se corresponda con lo esperado, es necesario Depurar (Debug), esto es, buscar los errores de ejecución. Las herramientas disponibles para ello son:

- Ejecución hasta punto de interrupción (breakpoint). Seleccionar la línea de código donde queremos que se detenga la ejecución, colocándose a la izquierda del número de línea (el icono del ratón se pasa a ver como una mano), y pulsar botón-derecho + Set BreakPoint (o pulsar F9).

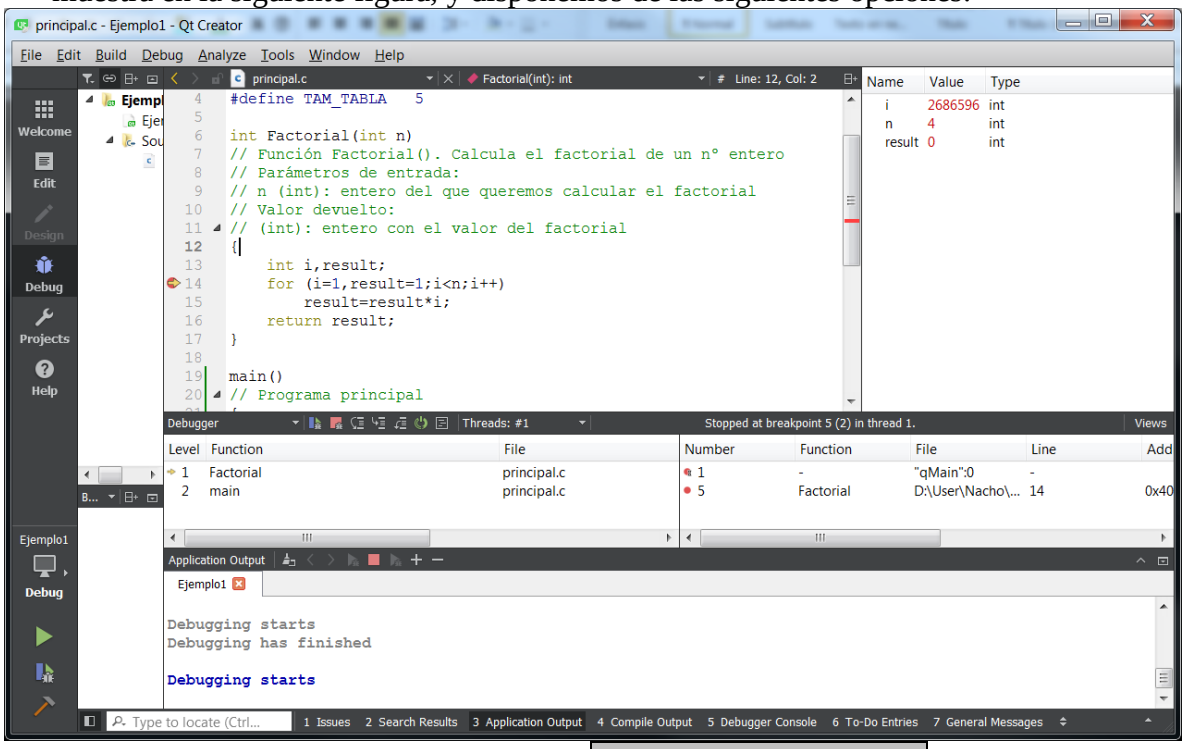


Aparecerá un círculo rojo a la izquierda de la línea:

```
9   // n (int): entero del que queremos calcular el factorial
10  // Valor devuelto:
11  // (int): entero con el valor del factorial
12  {
13      int i,result;
14      for (i=1,result=1;i<n;i++)
15          result=result*i;
16      return result;
17  }
```

que indica que la ejecución se detendrá cuando se llegue a este punto (si se ha iniciado con ejecución en modo depuración), y se podrá continuar paso a paso o hasta el siguiente punto de interrupción. El punto de interrupción se quita con una operación similar.

Una vez establecido el punto de interrupción, iniciar el programa con Debug -> Start Debugging [F5], la ejecución se detendrá al alcanzar el punto de interrupción, en cuyo momento la organización de las ventanas pasa a modo Debug, tal y como se muestra en la siguiente figura, y disponemos de las siguientes opciones:



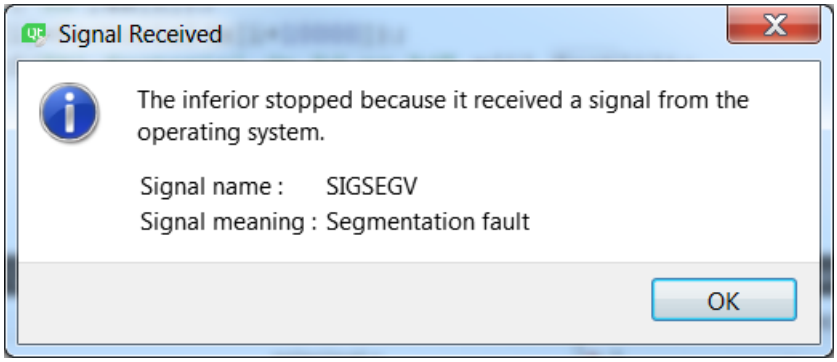
- Ejecución paso a paso (step over). Menú Debug -> Step Over [F10]. La ejecución se detiene tras la ejecución de la siguiente línea de código.
- Ejecución paso a paso (step into). Seleccionar menú Debug -> Step Into [F11]. La ejecución se detiene tras la ejecución de cada línea de código, pero saltará a la ejecución interna de una función que sea llamada en esa línea.
- Continuar hasta siguiente punto de interrupción (nuevamente F5 o con Debug -> Continue [F5]).
- Terminar la ejecución (Debug -> Stop Debugger).
- Visualización de variables. En la ventana de la derecha se muestran las variables relevantes en el momento de ejecución actual, y sus valores actuales. Según se avanza (paso a paso, hasta siguiente punto de interrupción) se puede comprobar en esta ventana los nuevos valores de las variables.
- Visualización de pila de llamadas. En la ventana que queda justo bajo el código se puede comprobar el estado de la pila (stack) hasta el punto de ejecución actual, y

también seleccionar el contexto de pila (variables locales, parámetros) de la función que se quiere visualizar.

- Puntos de interrupción condicionales. En modo depuración, la ventana Puntos de interrupción en la zona inferior derecha de la pantalla permite habilitar, inhabilitar y borrar puntos de interrupción. Adicionalmente, se puede seleccionar un punto de interrupción con el botón derecho del ratón y hacerlo condicional, esto es, que sólo se detenga la ejecución cuando se cumpla una determinada condición (valor de variable, nº de pasadas/visitas por ese punto).

4.2. Parada abrupta del programa ante error de ejecución

En algunas ocasiones, alguna de las instrucciones del programa no puede ser ejecutada porque violaría la seguridad o excede las posibilidades del computador, siendo la más típica el acceso a una posición de memoria inválida.
En estos casos, aparecerá un aviso como el que sigue:

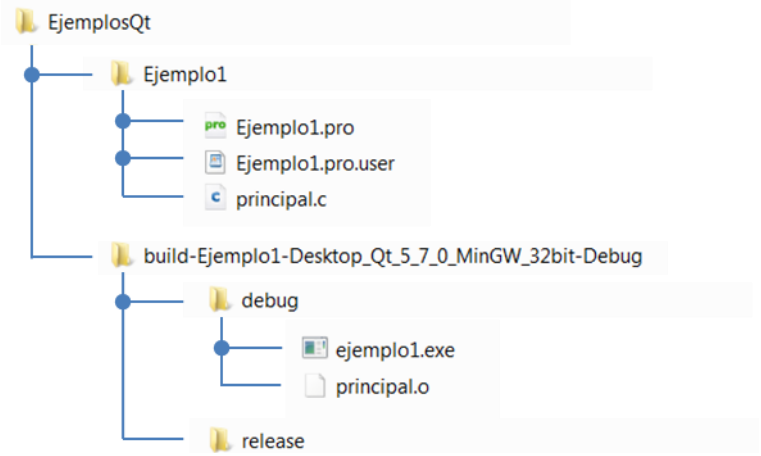


Tras pulsar Ok, se puede acudir a la pila de llamadas para comprobar el punto donde el programa no ha podido ejecutarse, y la pila de llamadas que ha llevado a ese punto:

¡ ATENCION ! En ciertas ocasiones, el error puede provocar daños en la pila de llamadas que hagan imposible al depurador localizar la función que se estaba ejecutando. En estos casos, la única solución es la depuración paso a paso hasta encontrar la instrucción que produce la excepción.

5. Mover un desarrollo a otros equipos

Cuando se ha realizado un programa, la estructura de directorios donde lo hemos desarrollado queda de manera similar a la siguiente:



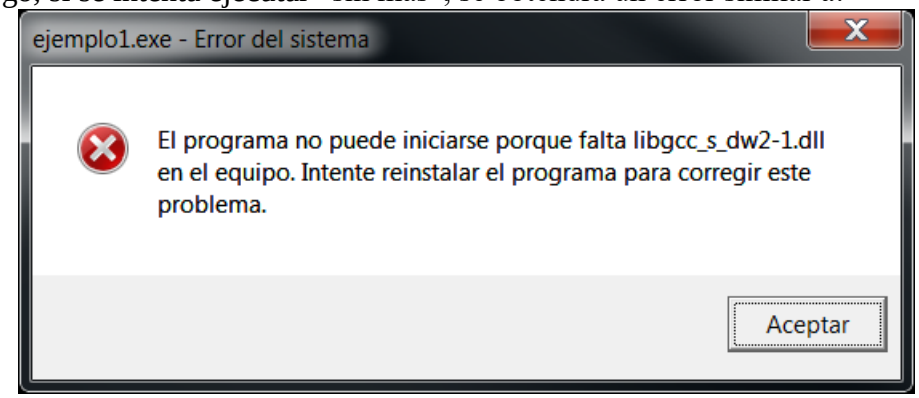
En el directorio de trabajo, especificado en el paso (6) de "2. Crear un programa en modo consola", aparece un directorio con el nombre del proyecto (indicado en el paso 5), dentro del cual se encuentra nuestro archivo de proyecto (.pro), nuestro(s) archivo(s) de código fuente (.c), y un archivo adicional .pro.user (contiene características específicas del proyecto).

Adicionalmente, aparece un 2º directorio build- ….. , que contiene los resultados de compilación: código objeto (.o) y ejecutable (.exe).

5.1. Utilizar el ejecutable fuera del entorno

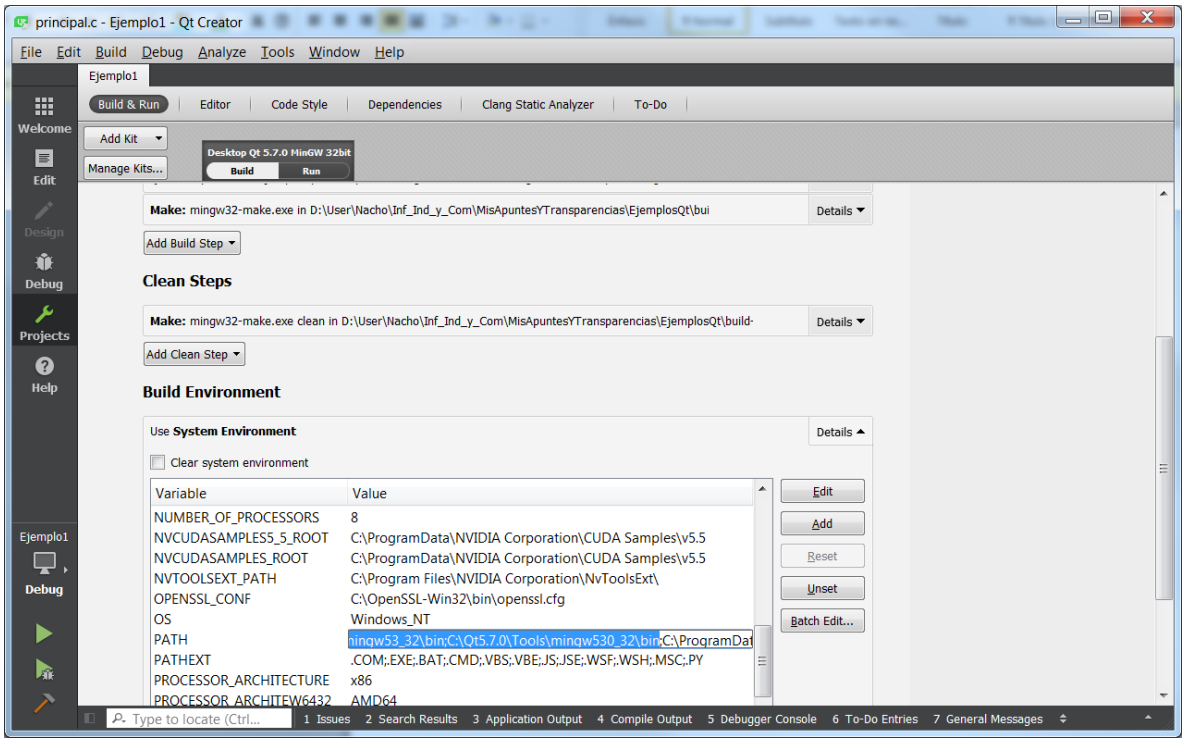
El archivo .exe es el único necesario para la ejecución del programa fuera del entorno. Se puede copiar o mover a otro equipo o directorio, y haciendo doble-click sobre él se produce su ejecución.

Sin embargo, si se intenta ejecutar "sin más", se obtendrá un error similar a:

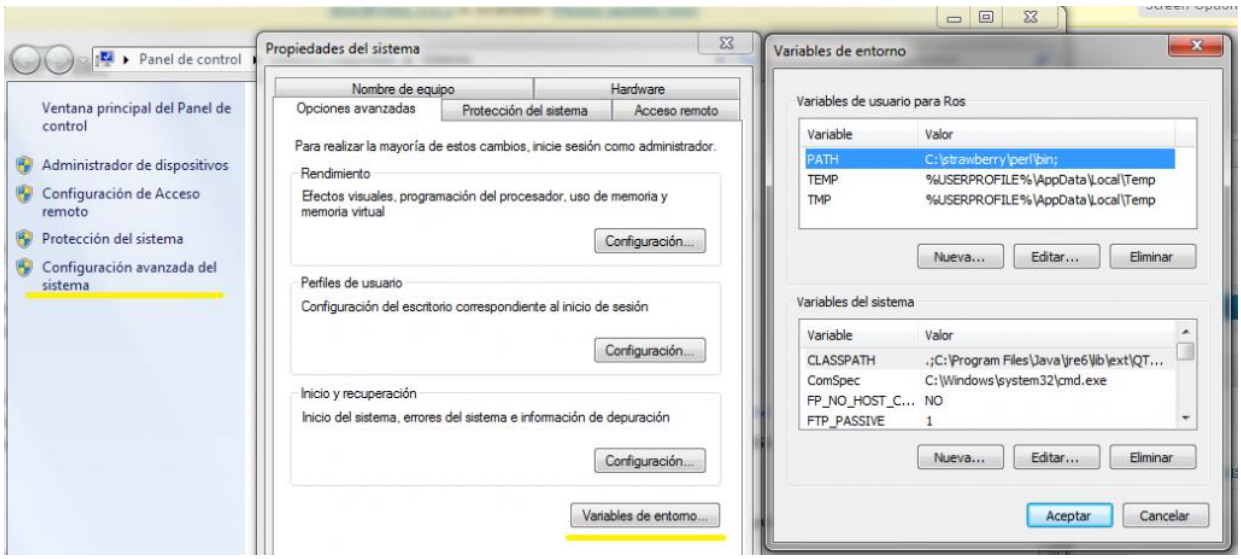


Esto ocurre porque el programa necesita una serie de librerías de enlace dinámico (DLL), cuya ubicación es conocida cuando se ejecuta dentro del entorno Qt Creator, pero no fuera de él.

Si lo queremos ejecutar en el mismo PC de desarrollo habrá que indicar la ubicación de estas librerías, que se puede saber mirando el contenido de PATH en Projects -> Build Environment:



Una vez comprobado el PATH específico de las DLL de Qt (típicamente será algo como C:\Qt\5.7\mingw53_32\bin;C:\Qt\Tools\mingw530_32\bin), se puede añadir al entorno del usuario para que siempre esté presente. En Windows:
Equipo → botón-derecho → Propiedades → Configuración avanzada del sistema → Variables de Entorno → Variables de Usuario → PATH



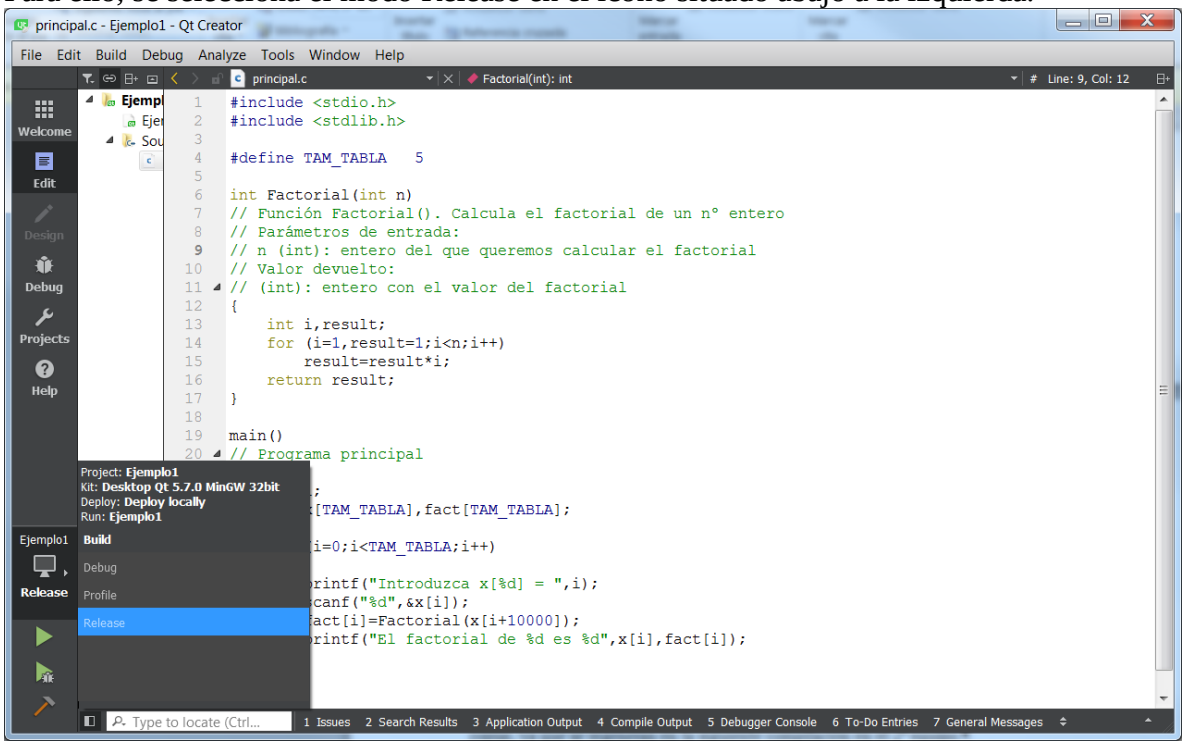
PATH = C:\Qt\5.7\mingw53_32\bin;C:\Qt\Tools\mingw530_32\bin

Para que esta nueva configuración tenga efecto, es posible que sea necesario cerrar y reiniciar la sesión de usuario en Windows.

5.2. Distribuir el ejecutable (a otros equipos)

Si queremos ejecutar el programa en otro PC que no tenga QtCreator, habrá que copiar los archivos DLL de los directorios anteriores (los añadidos al PATH) al nuevo PC. Además, para este caso es conveniente compilar en modo Release (sin información de depuración, ya que no se va a utilizar).

Para ello, se selecciona el modo Release en el icono situado abajo a la izquierda.



Una vez recompilado, el ejecutable a copiar será el que se encuentre en el directorio build-…..-Release\Release.

5.3. Copiar el desarrollo a otro equipo

Si se desea continuar el desarrollo en otro equipo, habrá que copiar únicamente el directorio de proyecto (Ejemplo1 en este caso), excepto el archivo .pro.user . El resto no se deben copiar, ya que se regeneran en la siguiente compilación en el 2º equipo.

Haciendo doble-click sobre el archivo .pro del proyecto se abre con Qt Creator, pero al no disponer del archivo .pro.user (que está ligado a cada equipo), habrá que volver a seleccionar el kit de compilación (paso (8) de “2. Crear un programa en modo consola”) antes de continuar.

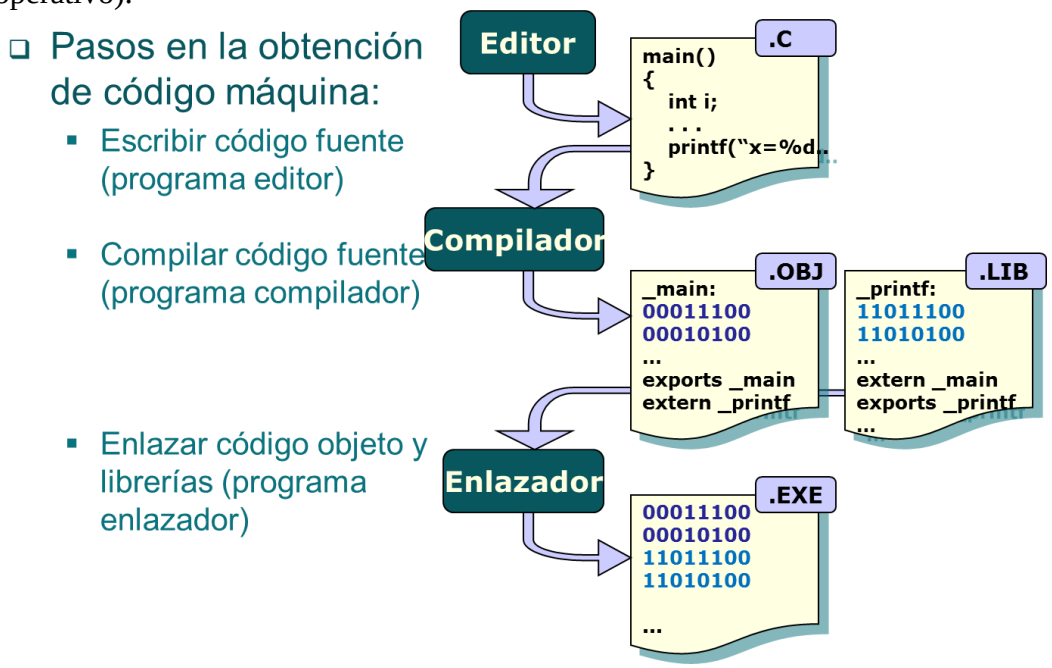
6. Entre bastidores

¿Qué ocurre cuando pulsamos Build ó Run? ¿Qué hacen CMake y qmake? ¿Qué contienen los archivos de configuración .pro o CMakeLists.txt? ¿Qué se puede/suele modificar en estos archivos?

6.1. Compilación directa

En el apartado 3.1 se mostraron los pasos para generación de código:

- **Compilación**: El archivo de código fuente es legible por un programa compilador, que produce el código máquina (sólo de lo que está escrito en el .c) y referencias a las partes de código no incluidas en él (funciones de librería, por ejemplo); el resultado es un archivo de código objeto (extensión típica .obj ó .o según el sistema operativo).



Existen diversos fabricantes que proporcionan el programa compilador. En nuestro caso, hemos elegido en la instalación de Qt-Creator utilizar gcc bajo mingw64. **gcc** es el compilador de GNU (https://es.wikipedia.org/wiki/GNU_Compiler_Collection), software libre y gratuito, multi-plataforma (válido para múltiples equipos y sistemas operativos, aunque desarrollado y orientado a Linux) y el más utilizado en general.

El compilador **gcc** será invocado por Qt-Creator para compilar cada uno de los archivos que se especifiquen como fuentes en el archivo de configuración de proyecto (nombre_de_proyecto.pro si hemos elegido qmake, CMakeLists.txt si hemos utilizado CMake).

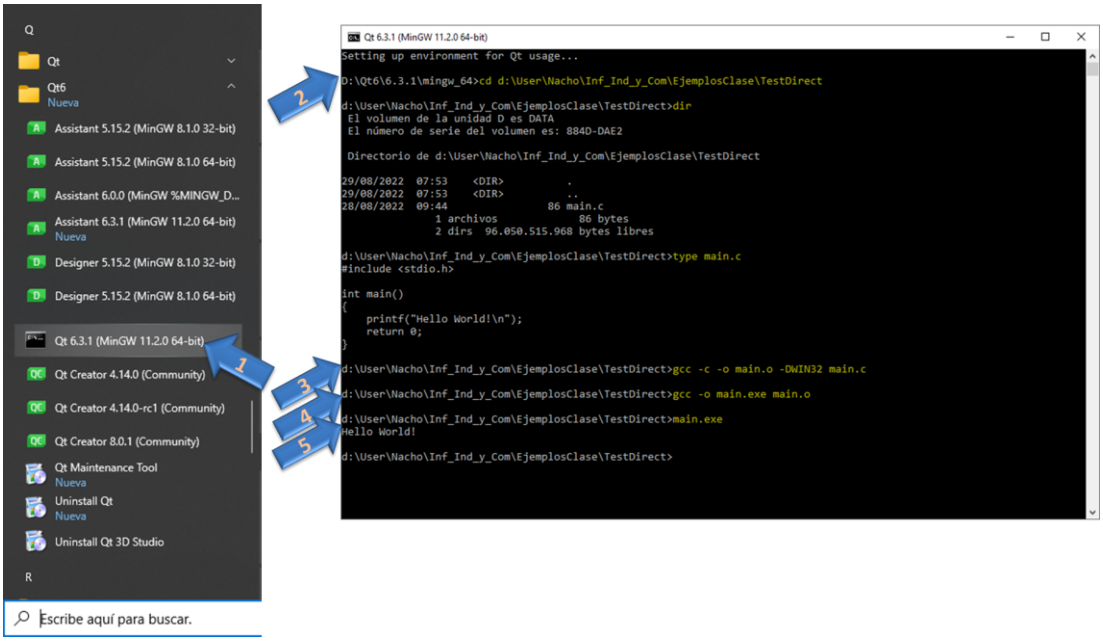
Por defecto sólo se incluye main.c, pero se pueden añadir otros fácilmente, simplemente editando el archivo de configuración.

CMakeLists.txt (CMake)	Nombre_de_proyecto.pro (qmake)
`add_executable(Test1 main.c)`	`SOURCES += \` `main.c`

- **Enlazado (link)**: el código máquina producido por el compilador sólo corresponde a lo escrito en el código fuente, pero falta más código para que el programa pueda ejecutarse, que ya se encuentra compilado en archivos de librería (.lib, .dll, .so). Estos archivos de librería contienen el código de las funciones a utilizar (printf para escribir, por ejemplo) que son comunes a todos los programas. Existen diversos fabricantes que proporcionan el programa enlazador. En nuestro caso, hemos elegido en la instalación de Qt-Creator utilizar gcc bajo mingw64. **gcc** ó **g++** son el compilador/enlazador de GNU.
  El enlazador **gcc** (ó **g++**) será invocado por Qt-Creator para enlazar todos los archivos de código objeto generados junto a las librerías especificadas para producir un solo archivo ejecutable.

Ambos programas, compilador y enlazador, pueden ser invocados directamente por el usuario en una ventana de comandos, que se puede abrir utilizando (1) el menú de Windows, submenú Qt, ventana de consola Qt 6.3.1 (Mingw…), y a continuación:

- Cambiar al directorio donde se encuentra el archivo de código fuente (2).
- Compilar (3) el archivo main.c con las opciones adecuadas para producir el archivo main.o
- Enlazar (4) el archivo main.o con las librerías por defecto para producir main.exe
- Ejecutar (5) main.exe



6.2. "Facilitando" el desarrollo de programas complejos

Para programas sencillos, con uno o pocos archivos de código fuente y librerías por defecto, los pasos anteriores son suficientes y no serían necesarias herramientas adicionales. Sin embargo, los programas a producir suelen ser mucho más complejos, involucrando múltiples archivos de código fuente, librerías, y opciones de compilación.

Cuando el programador produce un cambio, tiene que invocar al compilador y enlazador de nuevo, con las opciones correctas; pero, además, puede tener en cuenta que no sería necesario recompilar todos los archivos de código fuente, sino simplemente aquellos que han sufrido cambios.

El papel de los sistemas de desarrollo (**build system**) como CMake o qmake es facilitar estas tareas al programador; éste simplemente define lo que se necesita para el programa (archivos de código fuente, librerías, etc.) mediante un archivo de configuración, y el "build system" se encarga de generar todas las órdenes de compilación y enlazado necesarias, con las opciones adecuadas.

Estas órdenes de compilación y enlazado están basadas en dependencias, esto es, se genera un nuevo archivo de texto en el que se especifica qué y cómo debe ser recompilado cuando algún archivo cambia.

Además, el build system comprueba (realmente lo hace un paso inferior, el sistema de compilación nativo, en nuestro caso Ninja para CMake, o make para qmake) qué archivos necesitan ser recompilados porque ha cambiado algo desde la última vez que se compilaron, reduciendo drásticamente el tiempo total en grandes programas.

El papel del programa Qt Creator, conocido como Entorno Integrado de Desarrollo (**IDE** por sus siglas en inglés) es facilitar el acceso a todos los programas involucrados desde un único entorno gráfico, los anteriores y algunos más: editor, depurador (debugger), sistema de compilación nativo, ayuda en línea, etc.

6.3. Un ejemplo sencillo

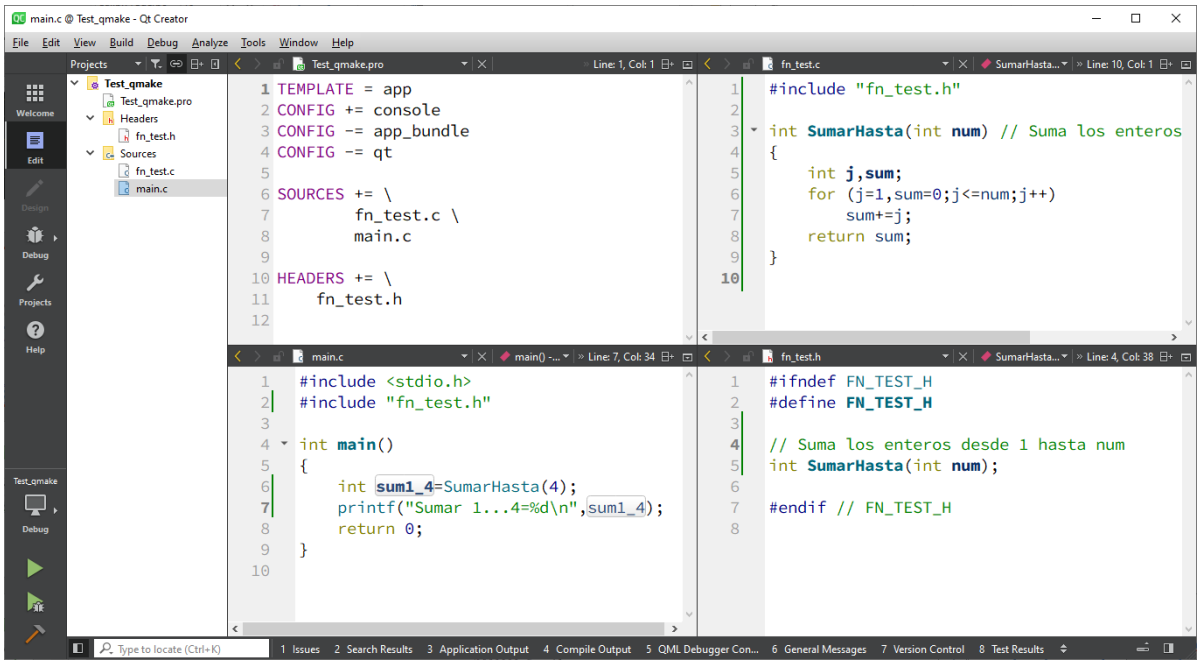
Un ejemplo sencillo con Qt Creator y qmake para crear un programa basado en dos módulos: main.c y fn_test.c , con un archivo de cabecera fn_test.h

Usuario → Creación del proyecto → Qt-Creator genera un test_qmake.pro por defecto

Usuario → Adición de archivos de código fuente y de cabecera para el programa deseado → Qt-Creator modifica automáticamente test_qmake.pro

Usuario → Edición de archivos de código fuente

Tras las operaciones anteriores, tenemos 4 archivos en nuestro proyecto, con contenidos:



Usuario → Orden de compilación →Qt-Creator → invoca al "build system" qmake →

qmake →

Lee Test_qmake.pro y genera Makefile con las dependencias y órdenes de compilación. Para el ejemplo anterior (se ha reducido el contenido a lo más importante):

```
#############################################################################
# Makefile for building: Test_qmake
# Generated by qmake (3.1) (Qt 6.3.1)
# Project:  ..\..\Test_qmake\Test_qmake.pro
# Template: app
#############################################################################

...

####### Compiler, tools and options
CC            = gcc
CXX           = g++
DEFINES       = -DUNICODE -D_UNICODE -DWIN32 -DMINGW_HAS_SECURE_API=1 -DQT_QML_DEBUG
CFLAGS        = -fno-keep-inline-dllexport -g -Wall -Wextra -Wextra $(DEFINES)
CXXFLAGS      = -fno-keep-inline-dllexport -g -Wall -Wextra -Wextra -fexceptions -mthreads $(DEFINES)
INCPATH       = -I../../Test_qmake -I. -ID:/Qt6/6.3.1/mingw_64/mkspecs/win32-g++
LINKER      =        g++
LFLAGS        =        -Wl,-subsystem,console -mthreads
LIBS        =
...

####### Files
SOURCES       = ..\..\Test_qmake\fn_test.c \
                ..\..\Test_qmake\main.c
OBJECTS       = debug/fn_test.o \
                debug/main.o
TARGET         = Test_qmake.exe
DESTDIR_TARGET = debug\Test_qmake.exe

####### Build rules
debug/Test_qmake.exe:  $(OBJECTS)
   $(LINKER) $(LFLAGS) -o $(DESTDIR_TARGET) $(OBJECTS) $(LIBS)

...

####### Compile
debug/fn_test.o: ../../Test_qmake/fn_test.c ../../Test_qmake/fn_test.h
   $(CC) -c $(CFLAGS) $(INCPATH) -o debug\fn_test.o ..\..\Test_qmake\fn_test.c

debug/main.o: ../../Test_qmake/main.c ../../Test_qmake/fn_test.h
   $(CC) -c $(CFLAGS) $(INCPATH) -o debug\main.o ..\..\Test_qmake\main.c

...
```

Declaración de "variables", cada una será sustituida por su texto cuando se use $(nombre_de_variable)

Declaración de "reglas", con el formato siguiente:
archivo_a_generar: archivos_a_chequear
orden_de_compilación_o_enlazado_a_generar
Significado:
Si alguno de los archivos a chequear ha cambiado desde la última compilación, crear el archivo a generar mediante la orden de compilación indicada

qmake →

Invoca al programa make para interpretar el archivo Makefile y ejecutar las órdenes de compilación y enlazado requeridas.
Por ejemplo, si se han cambiado todos los archivos, se ejecutarán (en el orden adecuado) las 3 reglas (2 de compilación y posterior enlazado):

```
gcc -c -fno-keep-inline-dllexport -g -Wall -Wextra -Wextra -DUNICODE -D_UNICODE -DWIN32 -
        DMINGW_HAS_SECURE_API=1  -DQT_QML_DEBUG -I../../Test_qmake -I. -
        ID:/Qt6/6.3.1/mingw_64/mkspecs/win32-g++  -o debug\fn_test.o
        ..\..\Test_qmake\fn_test.c

gcc -c -fno-keep-inline-dllexport -g -Wall -Wextra -Wextra -DUNICODE -D_UNICODE -DWIN32 -
        DMINGW_HAS_SECURE_API=1  -DQT_QML_DEBUG -I../../Test_qmake -I. -
        ID:/Qt6/6.3.1/mingw_64/mkspecs/win32-g++  -o debug\main.o
        ..\..\Test_qmake\main.c

g++ -Wl,-subsystem,console -mthreads -o debug\Test_qmake.exe  debug/fn_test.o
        debug/main.o
```

Sin embargo, si sólo se ha cambiado fn_test.c, sólo se aplicarán las reglas 1 y 3, ya que main.o sigue siendo válido.

6.4. Opciones más habituales en entorno de desarrollo qmake

qmake es específico de Qt-Creator, y además parece tener sus días contados, aunque su integración con el IDE es mejor.
Las opciones de compilación se especifican en un archivo de texto .pro, siendo las siguientes las más habituales:

Comando en .pro	Uso
# cualquier contenido	Comentario del programador, no se tiene en cuenta por qmake

TEMPLATE = app	Generar las órdenes de compilación y enlazado para producir un programa (ejecutable) completo
TEMPLATE = lib	Generar las órdenes de compilación y enlazado para producir una librería, enlazable desde otro programa
TARGET = nombre	Nombre del programa a generar
CONFIG += console	Añadir a la configuración las librerías para crear y utilizar una ventana tipo consola
CONFIG -= qt	Quitar de la configuración el uso de las librerías específicas de Qt
CONFIG += otros_a_especificar	Añadir otras opciones (ver ayuda de Qt-Creator) para estas opciones
CONFIG -= otros_ a_especificar	Quitar otras opciones (ver ayuda)
SOURCES += archivos	Añade archivos para ser compilados (si son varios, separar con espacios; si se desea en varias líneas, terminar cada una con el carácter \ )
HEADERS += archivos	Añade archivos para ser incluidos (si son varios, separar con espacios; si se desea en varias líneas, terminar cada una con el carácter \ )
INCLUDEPATH += directorio	Indica al compilador que debe buscar directorios de archivos #include también en el directorio indicado
DEFINES += texto	Añade #define para ser utilizados en los archivos .c
LIBS += -Ldirectorio	Indica al enlazador que debe buscar directorios de archivos de librería a enlazar también en el directorio indicado
LIBS += -lnombre_de_libreria	Indica al enlazador que se desea enlazar también una librería determinada

6.5. Operaciones más habituales en entorno de desarrollo CMake

Comando en CMakeLists.txt	**Uso**
# cualquier contenido	Comentario del programador, no se tiene en cuenta por CMake
project(nombre_proyecto …)	Declara proyecto a crear
add_executable(nombre_proyecto archivos_de_codigo_fuente_de_ los_que_depende)	Generar las órdenes de compilación y enlazado para producir un programa (ejecutable) completo en el proyecto indicado
add_library(nombre_proyecto SHARED archivos_de_codigo_ fuente_de_los_que_depende)	Generar las órdenes de compilación y enlazado para producir una librería en el proyecto indicado, enlazable desde otro programa
include_directories(directorios)	Indica al compilador que debe buscar directorios de archivos #include también en los directorios indicados
target_link_libraries(nombre_proyecto "-Ldirectorio_libreria -lnombre_libreria")	Enlaza la librería indicada

7. Ampliar los usos de Qt Creator

Aunque en esta asignatura sólo se utilizará la programación en lenguaje C, modo consola, el entorno permite otros usos, que cada usuario puede explorar:

- Creación de programas para entorno gráfico en Windows (con sus ventanas, como cualquier aplicación de Windows). Requiere conocimientos de C++ y de entorno gráfico. Se puede encontrar una introducción a ambos en los enlaces:
  - http://isa.uniovi.es/~ialvarez/Curso/descargas/000-ProgramacionCpp.pdf
  - http://isa.uniovi.es/~ialvarez/Curso/descargas/002-ProgramacionGraficaQt.pdf
- Creación de programas para otros dispositivos y/o Sistemas Operativos, sin modificar el código fuente, utilizando otros "kits" de compilación. Hay kits disponibles para :
  - PC/Linux
  - ARM/Linux
  - ARM/Android
  - Apple/iOS
  - Apple/macOS

  Ver http://doc.qt.io/qt-5/gettingstarted.html para instrucciones de instalación y creación de programas con estos kits.