



PROGRAMACIÓN ORIENTADA A OBJETOS USANDO C++

1ª EDICIÓN

1ª SESIÓN

Curso 2015-2016

ÍNDICE

1. Cuestiones Iniciales
2. Historia de los lenguajes de programación
3. Historia de C/C++
4. IDEs
5. Qt Creator
6. Introducción/Repaso del lenguaje ANSI C
 1. Conceptos básicos
 2. Arrays
 3. Estructuras de datos

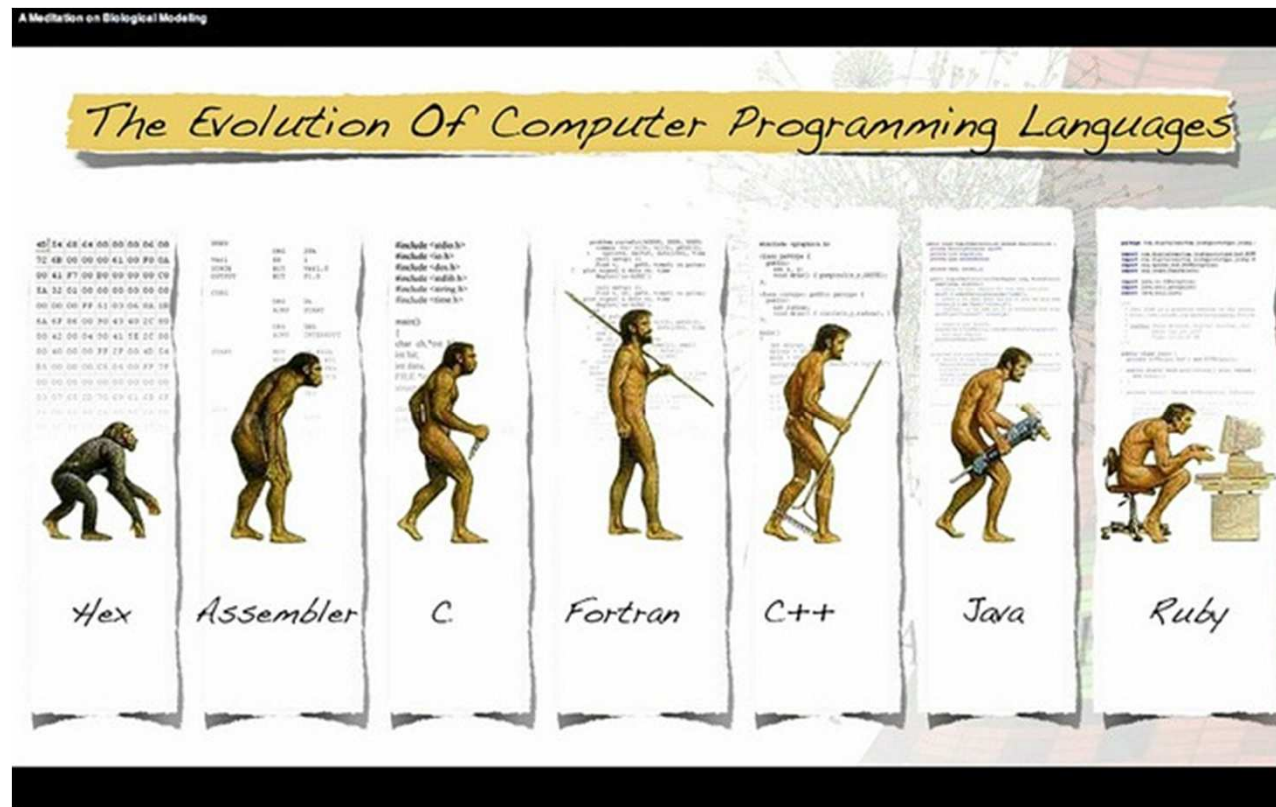
CUESTIONES INICIALES

- Aprender sobre la historia de los lenguajes de programación
- Aprender sobre los diferentes paradigmas de lenguajes de programación
- Aprender sobre diferentes IDEs

CUESTIONES INICIALES

- Recordar/ Introducción al lenguaje ANSI C
- Programación orientada a objetos con C++
- Novedades del nuevo C++11
- Novedades del nuevo C++14

2- HISTORIA DE LOS LENGUAJES DE PROGRAMACIÓN



HISTORIA

- Antes del año 1940 surgieron los primeros indicios de un lenguaje de programación antes de la computación moderna.
- En el año 1801 Joseph Marie Jacquard creó un telar que podía ser programado con tarjetas perforadas



HISTORIA

- Entre el año 1842-1843 Ada Lovelace tradujo las memorias del matemático Luigi Manabre acerca de una máquina para calcular los números de Bernulli.
- Esta máquina es considerada por muchos historiadores como el primer programa de computadora del mundo.
- Como muchos “primeros” en la historia, el primer lenguaje de programación moderno es difícil de identificar. Desde un inicio, las restricciones de hardware definían el lenguaje.

HISTORIA

- Las tarjetas perforadas fueron los primeros códigos de programación. Permitían configurar una máquina mecánica para un trabajo concreto.



HISTORIA

- En los años 40 fueron creadas las primeras computadoras modernas con alimentación eléctrica. Aunque tenían serias restricciones de velocidad y memoria.
- En el año 1943, Konrad Zuse publicó un artículo acerca de su lenguaje de programación: Plankalkül, aunque no fue implementado en su vida.
- En el año 1943 surge el sistema de codificación ENIAC.

HISTORIA

- En los años 1943-1954 ENIAC evoluciona a UNIACAC que es un conjunto de instrucciones destinadas a un fabricante específico.
- Las décadas de 1950 y 1960:
 - FORTRAN (1955), creado por John Backus et al.
 - LISP (1958), creado por John McCarthy et al.
 - COBOL (1959), creado por el Short Range Committee, altamente influenciado por Grace Hopper.
 - Otros lenguajes

HISTORIA

- Entre los años 1968 y 1979 surgen los primeros paradigmas de programación.
- La mayoría de paradigmas más importantes y actualmente en uso se inventaron en este periodo:

¿¿Qué es un paradigma de programación??

HISTORIA

- Un **paradigma de programación** es una propuesta tecnológica adoptada por una comunidad de programadores cuyo núcleo central es *incuestionable* en cuanto que únicamente trata de resolver uno o varios problemas claramente delimitados.
- Tipos de paradigmas de programación más comunes:

HISTORIA

- Procedimental: es considerado el más común y está representado, por ejemplo, por C, BASIC o Pascal.
- Funcional: está representado por Scheme o Haskell. Este es un caso del paradigma declarativo.
- Lógico: está representado por Prolog. Este es otro caso del paradigma declarativo.
- Declarativo: por ejemplo la programación funcional, la programación lógica, o la combinación lógico-funcional.
- Orientado a objetos: está representado por C++, un lenguaje completamente orientado a objetos.
- Programación dinámica: está definida como el proceso de romper problemas en partes pequeñas para analizarlos.
- Programación multiparadigma: es el uso de dos o más paradigmas dentro de un programa.

HISTORIA

- Actualmente el paradigma de programación más usado es el de la programación orientada a objetos.
- Algunos lenguajes importantes que se desarrollaron en este período fueron:
 - 1968 - Logo
 - 1969 - B (precursor C)
 - 1970 - Pascal
 - 1970 - Forth
 - 1972 - C

HISTORIA

- En la década de 1980 fueron consolidados los paradigmas ya inventados. Se trabajó a partir de las ideas de la década anterior.
- Surgió C++ como una combinación de programación orientada a objetos y la programación de sistemas.

HISTORIA

- Algunos de los sistemas más importantes que se desarrollaron en este periodo son:
 - 1980 - C++
 - 1983 - Ada
 - 1984 - Common Lisp
 - 1984 - MATLAB
 - 1985 - Eiffel
 - ...

HISTORIA

- La década de 1990 es la década de Internet.
- Con la apertura de esta plataforma totalmente, se creó una oportunidad para nuevos lenguajes de programación.
- En particular, el lenguaje de programación Java se hizo muy popular debido a su propia integración con los navegadores.

HISTORIA

- La década de 1990 es la década de Internet.
- Con la apertura de esta plataforma totalmente, se creó una oportunidad para nuevos lenguajes de programación.
- En particular, el lenguaje de programación Java se hizo muy popular debido a su propia integración con los navegadores.

HISTORIA

- Algunos de los lenguajes más importantes que se desarrollaron en este periodo son:
 - 1990 - Haskell
 - 1991 - Python
 - 1991 - Visual Basic
 - 1991 - HTML
 - 1993 - Ruby
 - 1995 - Java
 - 1995 - JavaScript
 - 1995 - PHP
 - 1996 - WebDNA
 - 1997 - Rebol
 - 1999 - D
 - ...

¡No memoricéis un lenguaje! Son solo herramientas. Todas las marcas de herramientas tienen las mismas funciones básicas



3- HISTORIA DE C/C++



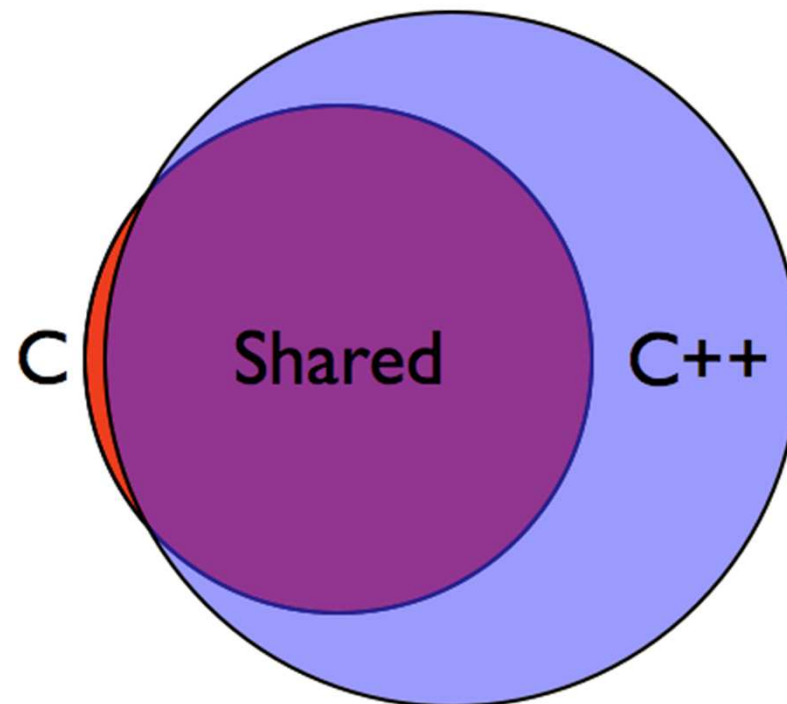
HISTORIA

- **C++** es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup.



HISTORIA

- La intención de su creador fue extender el lenguaje de programación C para que permitiera la manipulación de objetos.



HISTORIA

- Posteriormente se sumaron los paradigmas de programación estructurada y programación orientada a objetos.
- Por lo tanto, se suele decir que C++ es un lenguaje de programación multiparadigma.
- Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de fabricantes de compiladores modernos.

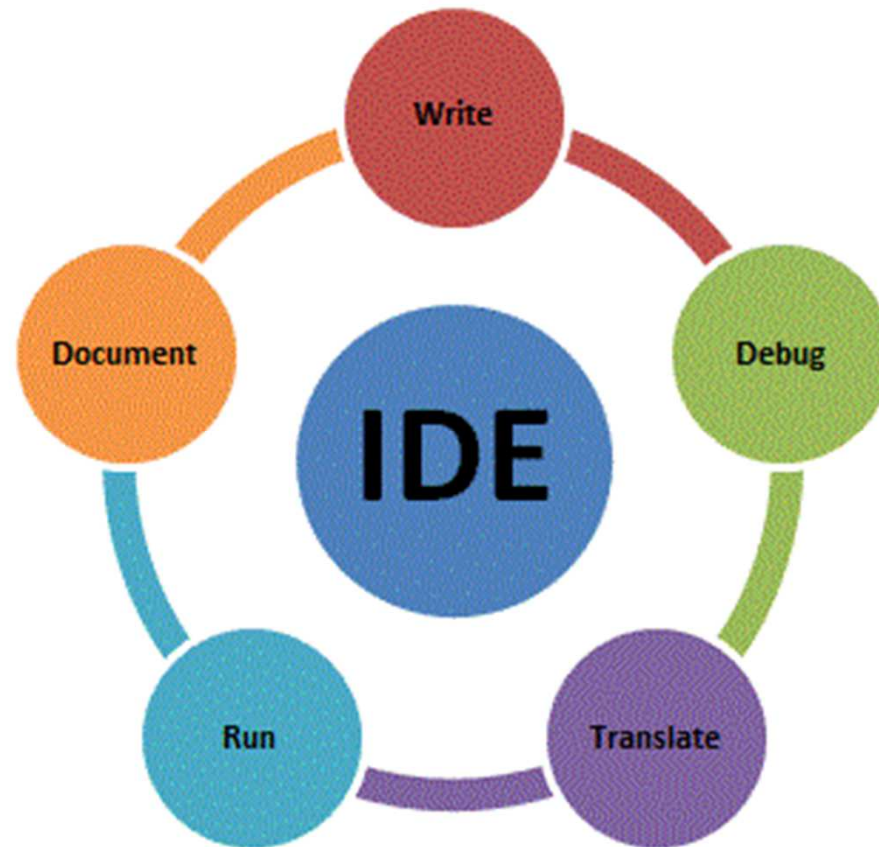
HISTORIA

- En agosto de 2011 surgió C++11
- Se puede descartar de este nuevo estándar de C++11 :
 - La función lambda
 - Referencias rvalue
 - La palabra reservada auto
 - Inicializaciones uniforme
 - Platillas con número variable de argumentos
 - Además de ha actualizado la biblioteca estándar del lenguaje.

HISTORIA

- En mayo de 2013 surgió C++14.
- Principalmente ofrece correcciones de errores y algunas mejoras respecto a C++11.
- Se puede descartar de este nuevo estándar de C++14 :
 - Deducción del tipo de retorno
 - Deducción alternativa de variables
 - Platillas de variables
 - Literales binarios
 - Agrupación de dígitos
 - Lambda genéricos
 - Acceso a tuplas a partir del tipo de dato

4- IDEs



ENTORNOS DE DESARROLLO INTEGRADO

- Un entorno de desarrollo integrado (en inglés Integrated Development Environment, IDE) es una aplicación software que proporciona servicios integrados para facilitar al programador el desarrollo de software.
- Normalmente, un IDE consiste en un editor de texto, herramientas de construcción automática y un depurador.

ENTORNOS DE DESARROLLO INTEGRADO

- La mayoría de los IDEs tiene auto completado inteligente del código.
- Estos entornos de desarrollo están diseñados para maximizar la productividad del programados.
- Algunos de los IDEs más extendidos son: Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Codenvy y Microsoft Visual Studio, Xojo, Delphi y Qt Creator.

ENTORNOS DE DESARROLLO INTEGRADO



¿QUÉ IDE UTILIZAR?

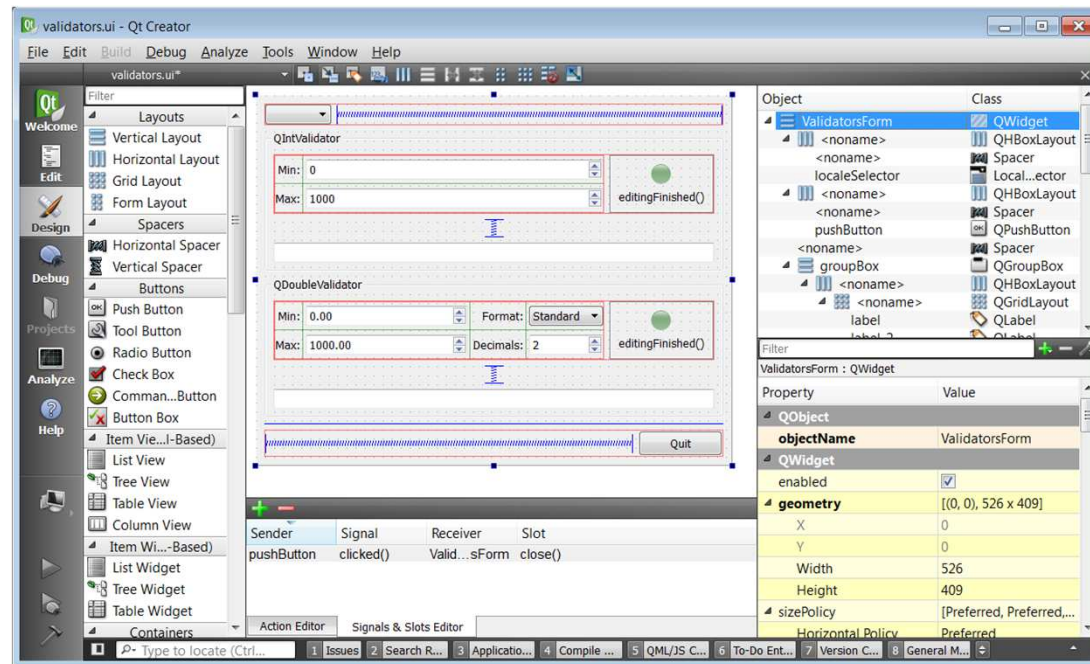
IDE	License	Windows	GNU/Linux	Mac OS X	Other platforms	Debugger	GUI builder	Integrated toolchain	Profiler	Code coverage	Autocomplete	Static code analysis	GUI-based design	Class browser	
Anjuta	GPL	No	Yes	No	FreeBSD	Yes	Yes	Yes	Yes	Unknown	Yes	Unknown	Yes	Yes	
C++Builder	Proprietary	Yes	No (Kylix deprecated)	Yes (Cross compiler)		Yes	Yes	Yes	Yes bundled with ADTime	Yes	Yes	Yes	Yes	Yes	
Code::Blocks	GPL	Yes	Yes	Yes	FreeBSD	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes [4]	Yes	
Codelite	GPL	Yes	Yes	Yes	FreeBSD, Mac OS	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	
Dev-C++	GPL	Yes	No [7]	No	FreeBSD	Yes	No	Yes	Yes	Unknown	Yes	Unknown	Yes	Yes	2012
Eclipse CDT	GPL	Yes	Yes	Yes	JVM	Yes	Yes [2]	No	Unknown	Unknown	Yes	Yes	No	Yes	
Geany	GPL	Yes	Yes	Yes	FreeBSD, OpenBSD	Yes	No	No	No	No	Yes	No	No	Yes	
GNAT Programming Studio	GPL	Yes	Yes	Yes	Solaris	Yes	Unknown	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	
KDevelop	GPL	No	Yes	Yes	FreeBSD, Solaris	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
LabWindows/CVI	Proprietary	Yes	No	No	Target: Unix, Pharos RTOS	Yes	Yes	Yes	Yes	No	Yes	No	Yes	N/A	
LedWin32	Freeware / Proprietary	Yes	Yes (obsolete)	No		Yes	Yes (unstable)	Yes	Yes	Unknown	Yes	Yes	Yes	Unknown	
MonoDevelop	UGPL	No [12]	Yes	Yes	FreeBSD	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	
NetBeans C/C++ pack	ODOL	Yes	Yes	Yes	Solaris	Yes [11]	Yes [11]	Yes [12]	Yes [11]	Yes	Yes	Yes	Yes	Yes	
OpenWatcom	Proprietary freeware	Yes (32-bit only)	partial	No	MS-DOS, OS/2, FreeBSD	Yes GUI remote	Yes	Yes	Yes	No	No	No	Yes	Yes	
PhlaxC	Proprietary freeware	Yes	No	No	Windows CE	Yes	Yes	Yes	Unknown	Unknown	Yes	Unknown	Unknown	Unknown	
Philamcode Embedded Studio	Freeware / Proprietary	Yes	Yes	No		Yes	Yes	Yes	Unknown	Unknown	Yes	Unknown	Yes, w/Widgets	Yes	
Qt Creator	GPL / LGPL / Proprietary	Yes	Yes	Yes	FreeBSD, Symbian, Maemo	Yes	Yes	Unknown	Yes	No	Yes	No	Yes	Yes	
Sun Studio	Proprietary freeware	No	Yes	No	Solaris	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	
National Software Architect (Eclipse IDE)	Proprietary	Yes	Yes	No	JVM	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	
Turbo C++ Explorer	Proprietary freeware	Yes	No	No		Yes	Yes	No	No	No	Yes	No	Yes	Yes	
Turbo C++ Professional	Proprietary	Yes	No	No		Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	
Ultimate++ 3.14.0.0	SSO	Yes	Yes	No		Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	
Microsoft Visual Studio	Proprietary	Yes	No	No	cross-compile to Windows Mobile, Mac OS 7 (i386/x64 only)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Microsoft Visual Studio Express	Proprietary freeware	Yes	No	No		Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	
wxDev-C++	GPL	Yes	No	No		Yes	Yes	Unknown	Yes	Unknown	Yes	Unknown	Yes	Unknown	
Xcode Apple	Proprietary	No	No	Yes	cross-compile to iOS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
AppCode	Proprietary	No	No	Yes	No	Yes	No	No	Xcode profiler	No	Yes	Yes	Yes	Yes	
IDE	License	Windows	Linux	Mac OS X	Other platforms	Debugger	GUI builder	Integrated toolchain	Profiler	Code coverage	Autocomplete	Static code analysis	GUI-based design	Class browser	

http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments

NOSOTROS UTILIZAREMOS QT CRETOR

- Porque es multiplataforma
- Fácil de usar

5- QT CREATOR



HISTORIA

- Qt es una herramienta multiplataforma que soporta C++, JavaScript y QLM.
- Está escrito en C++
- Qt Creator fue creado en el año 2007
- Incluye un depurador y un diseñador (designer)
- Incluye resaltado de palabras resaltadas y autocompletado.

¿DÓNDE DESCARGAR?

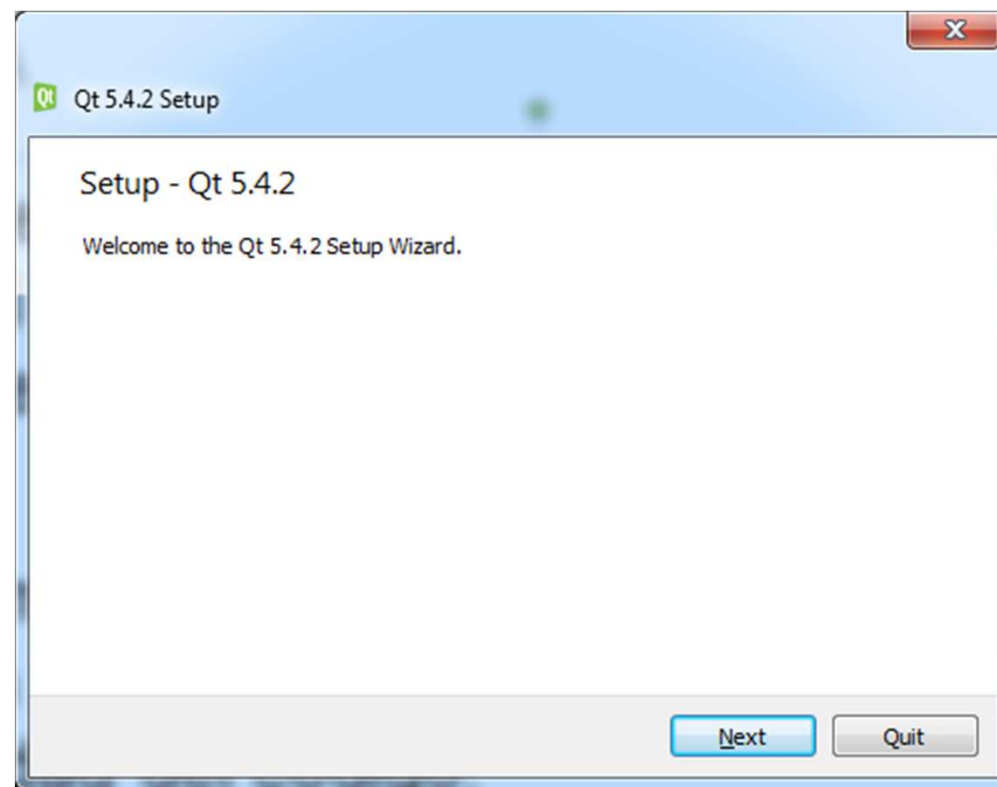
- Qt creator puede ser descargado desde la página oficial de Qt. <http://www.qt.io/download/>
 - La versión básica es gratuita pero tenemos que registrarnos.
- Más sencillo descargarlo desde aquí (también es la página oficial):
 - <http://download.qt.io/archive/qt/>
 - Aquí tenemos todas las versiones completas para Mac, Linux y Windows.

INSTALACIÓN

- Vamos a descargar la versión más reciente:
 - 5.4 -> 5.4.2 -> qt-opensource-windows-x86-mingw491_opengl-5.4.2.exe (Windows)
 - 5.4 -> 5.4.2 -> qt-opensource-linux-x64-5.4.2.run (Linux de 64 bits)
 - 5.4 -> 5.4.2 -> qt-opensource-linux-x86-5.4.2.run (Linux de 32 bits)
 - 5.4 -> 5.4.2 -> qt-opensource-mac-x64-ios-5.4.2.dmg (Mac de 64 bits)

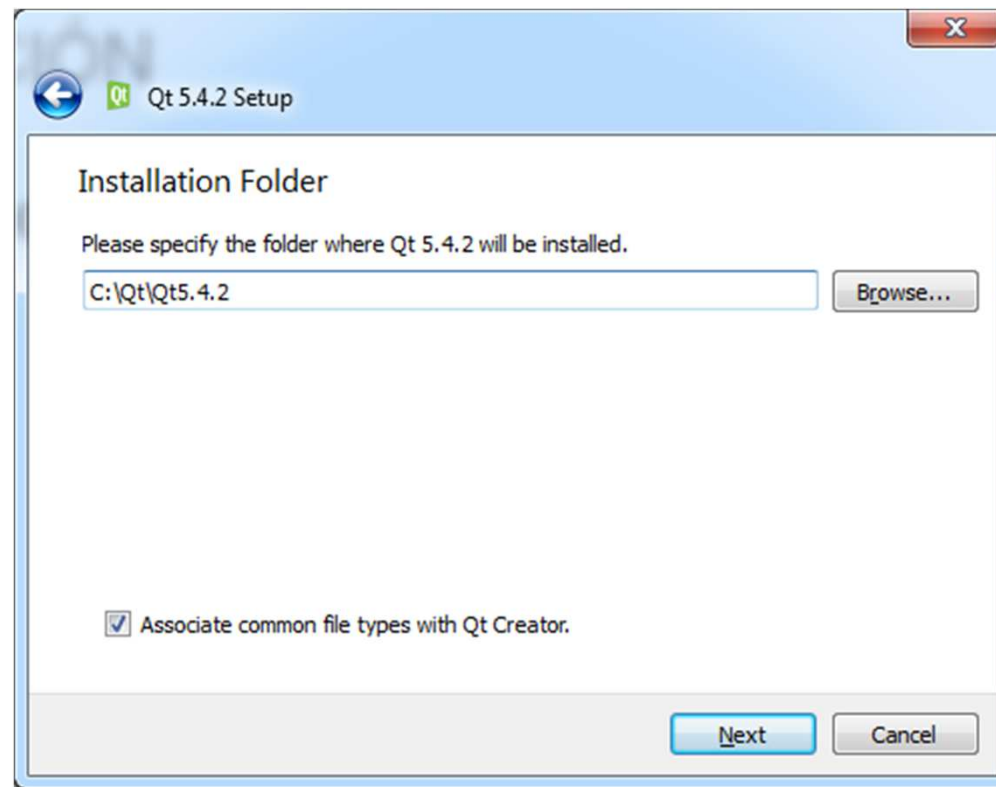
INSTALACIÓN

- Instalamos el archivo descargado



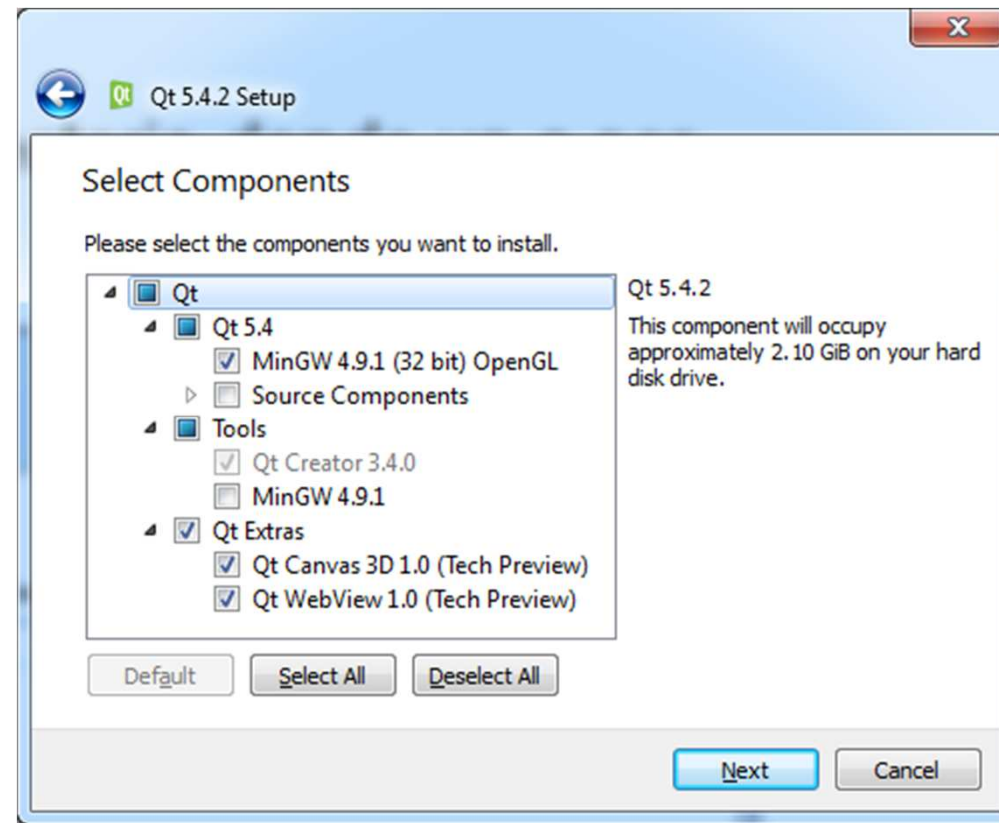
INSTALACIÓN

- Seleccionamos el directorio donde va a ser instalado:



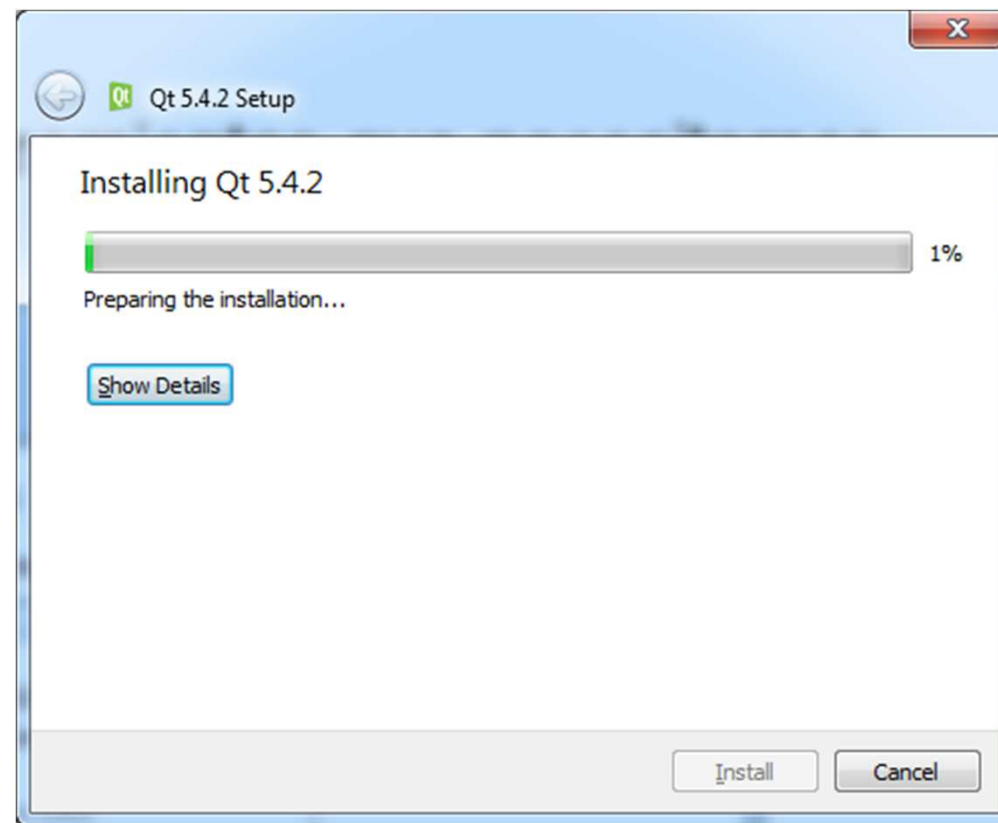
INSTALACIÓN

- Seleccionamos la herramientas que necesitamos de Qt:



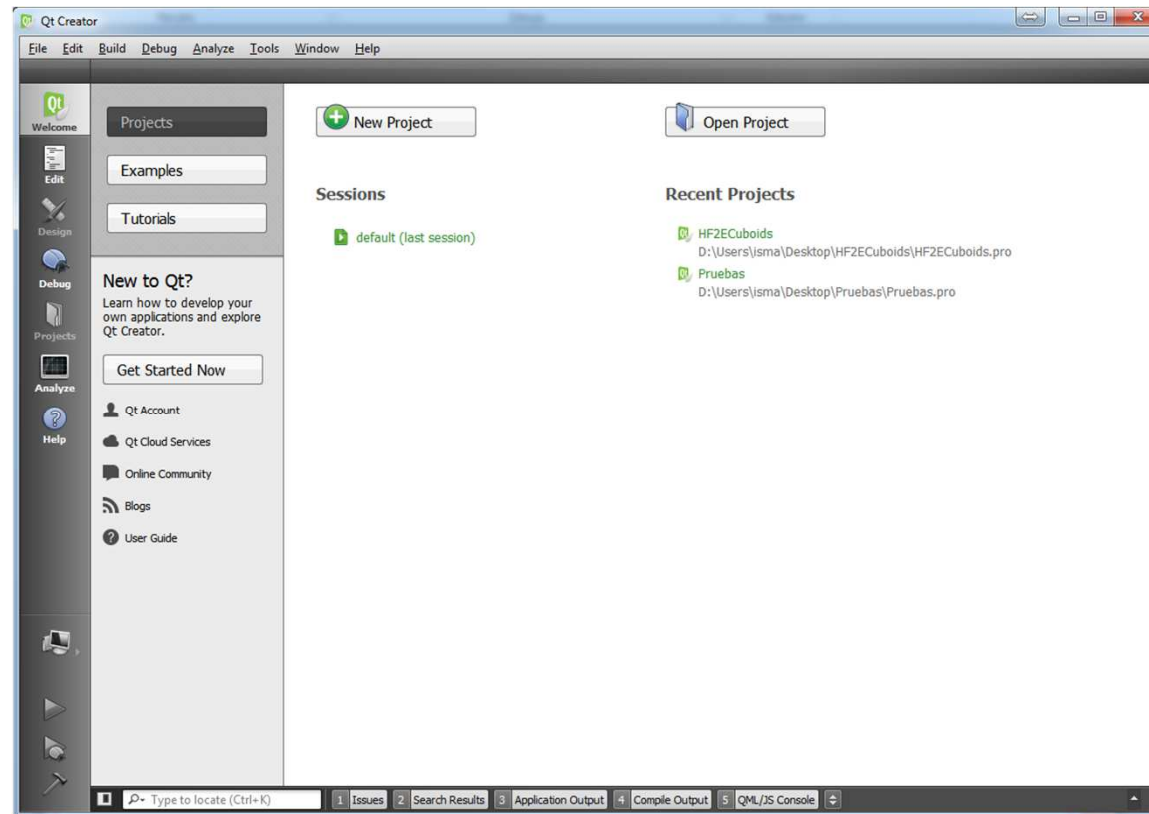
INSTALACIÓN

- Continuar y esperamos a que se instale:



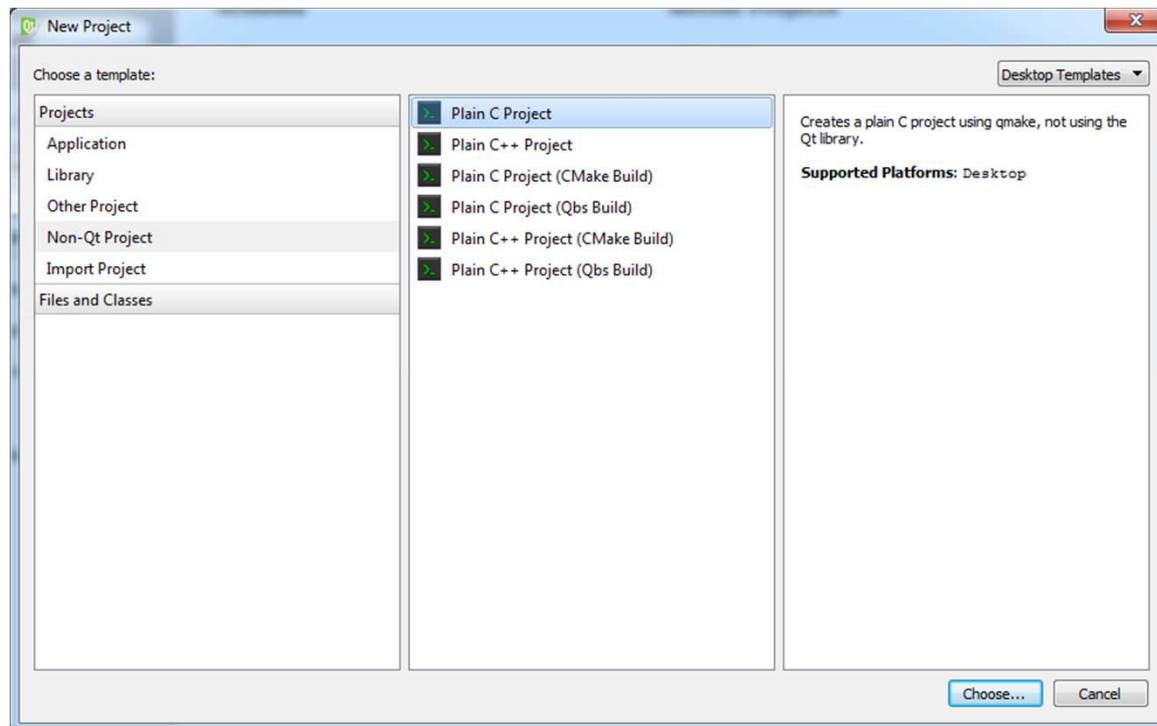
QT CREATOR

- Una vez que ha sido instalado y lo abrimos, veremos algo como esto:



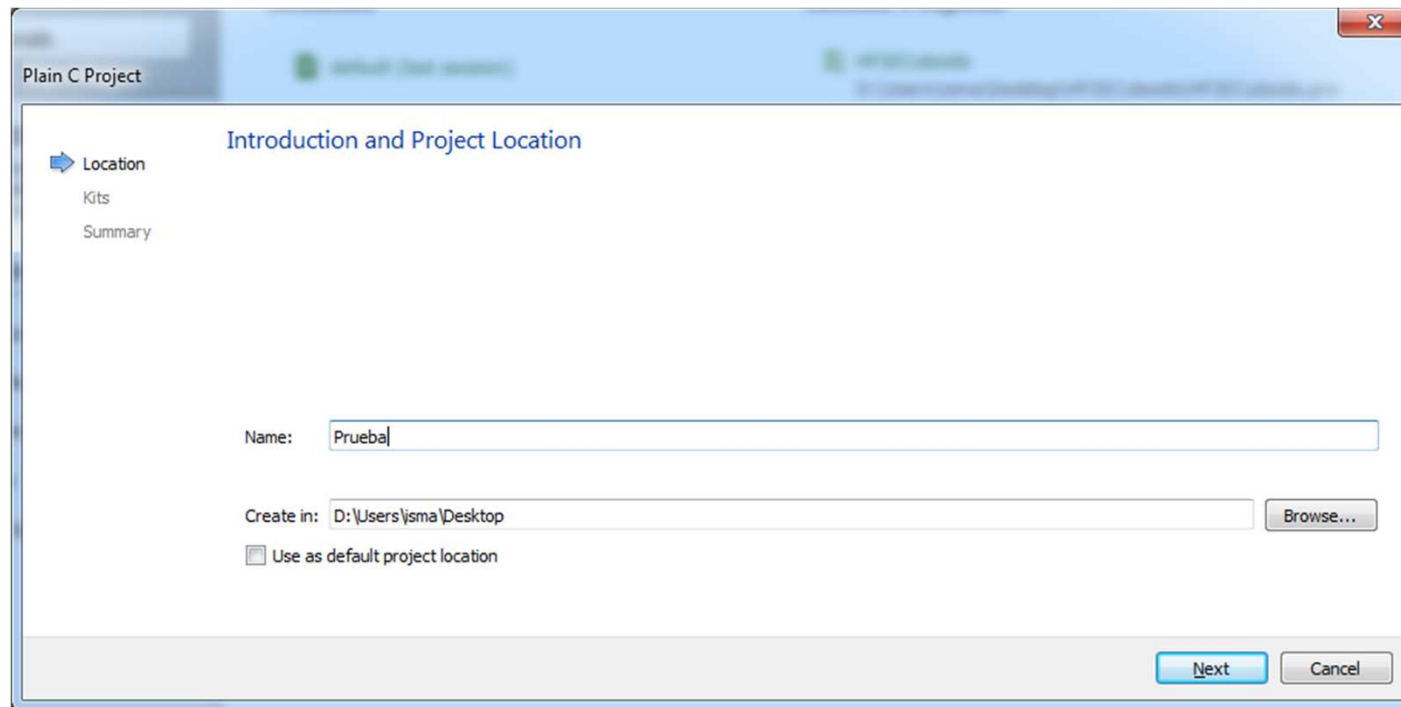
QT CREATOR

- Vamos a crear nuestro primer proyecto (Un proyecto en C). «New Project», seleccionamos:
 - Non-Qt Project → Plain C Project



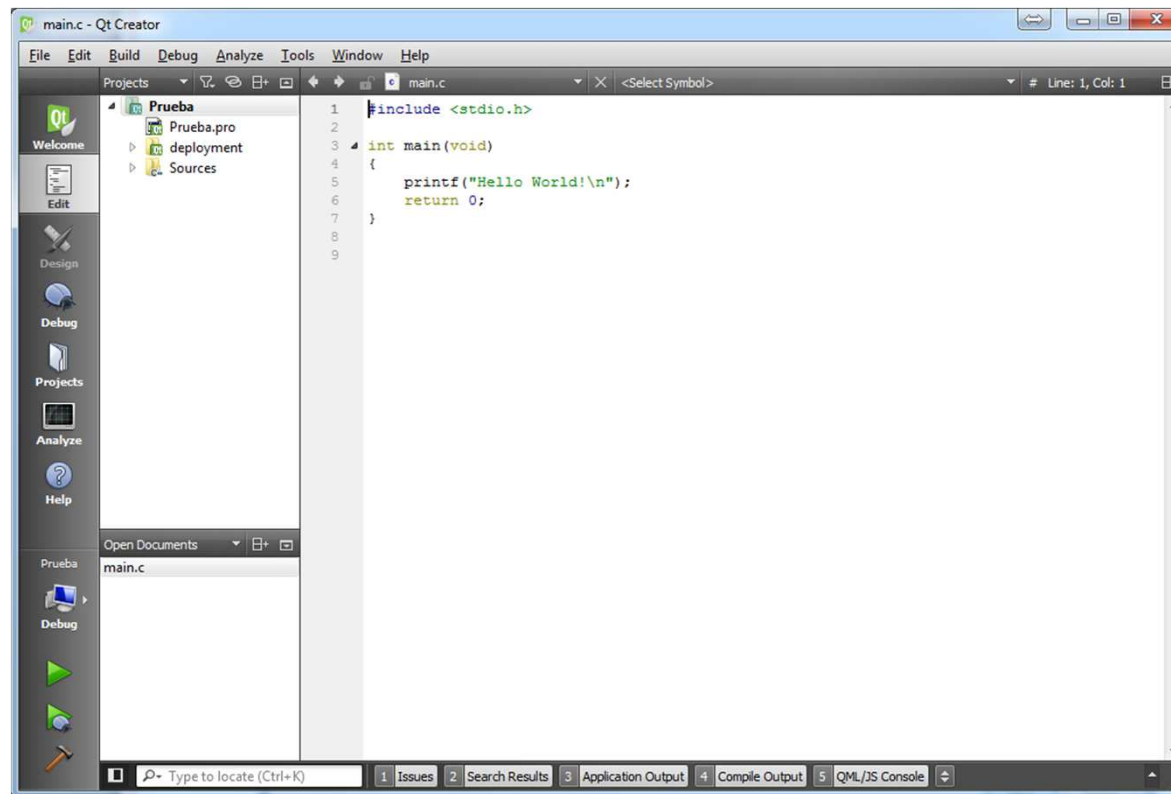
QT CREATOR

- Seleccionamos donde queremos que se cree nuestro proyecto:

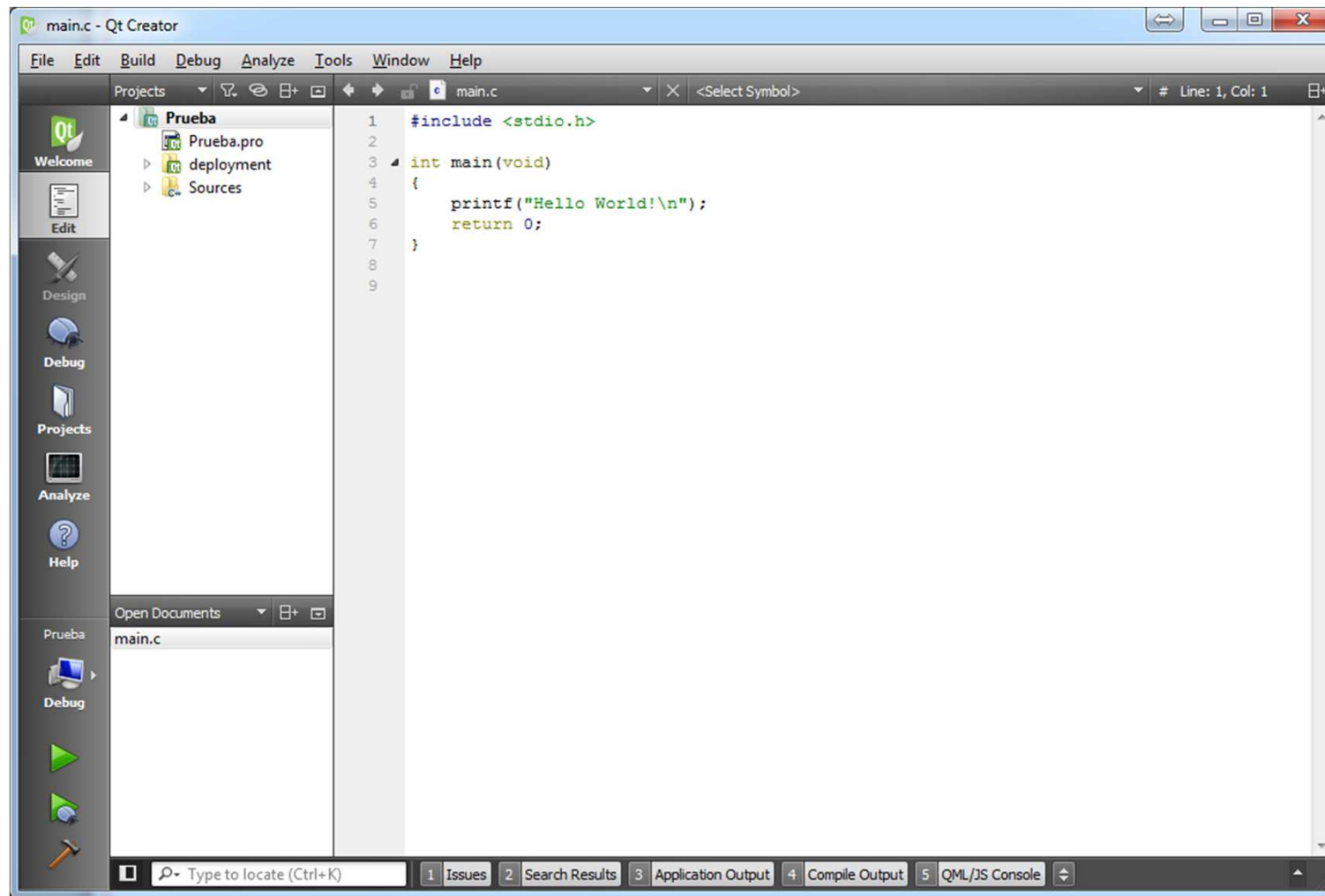


QT CREATOR

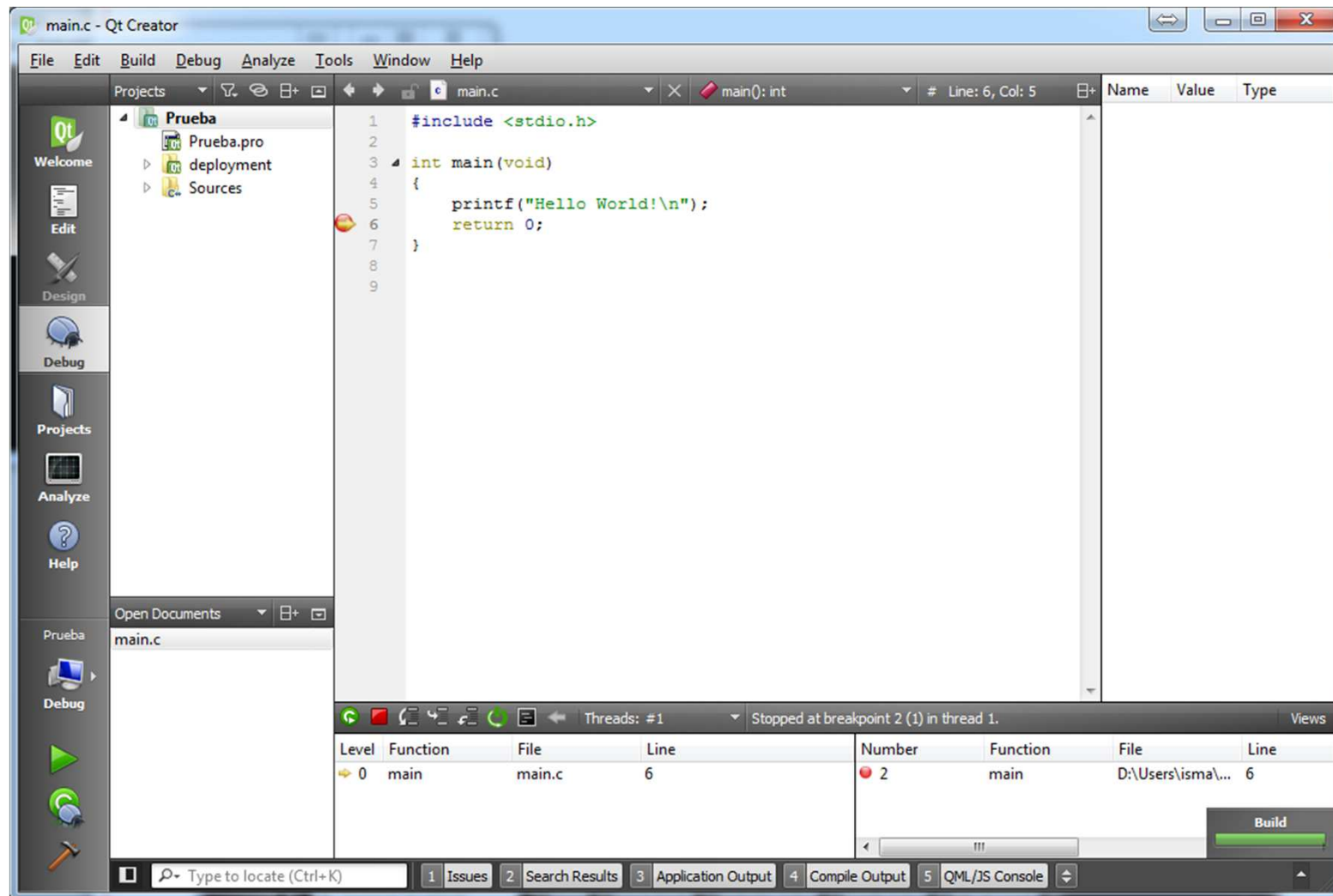
- Automáticamente Qt nos crea nuestro primer programa (un Hola Mundo).



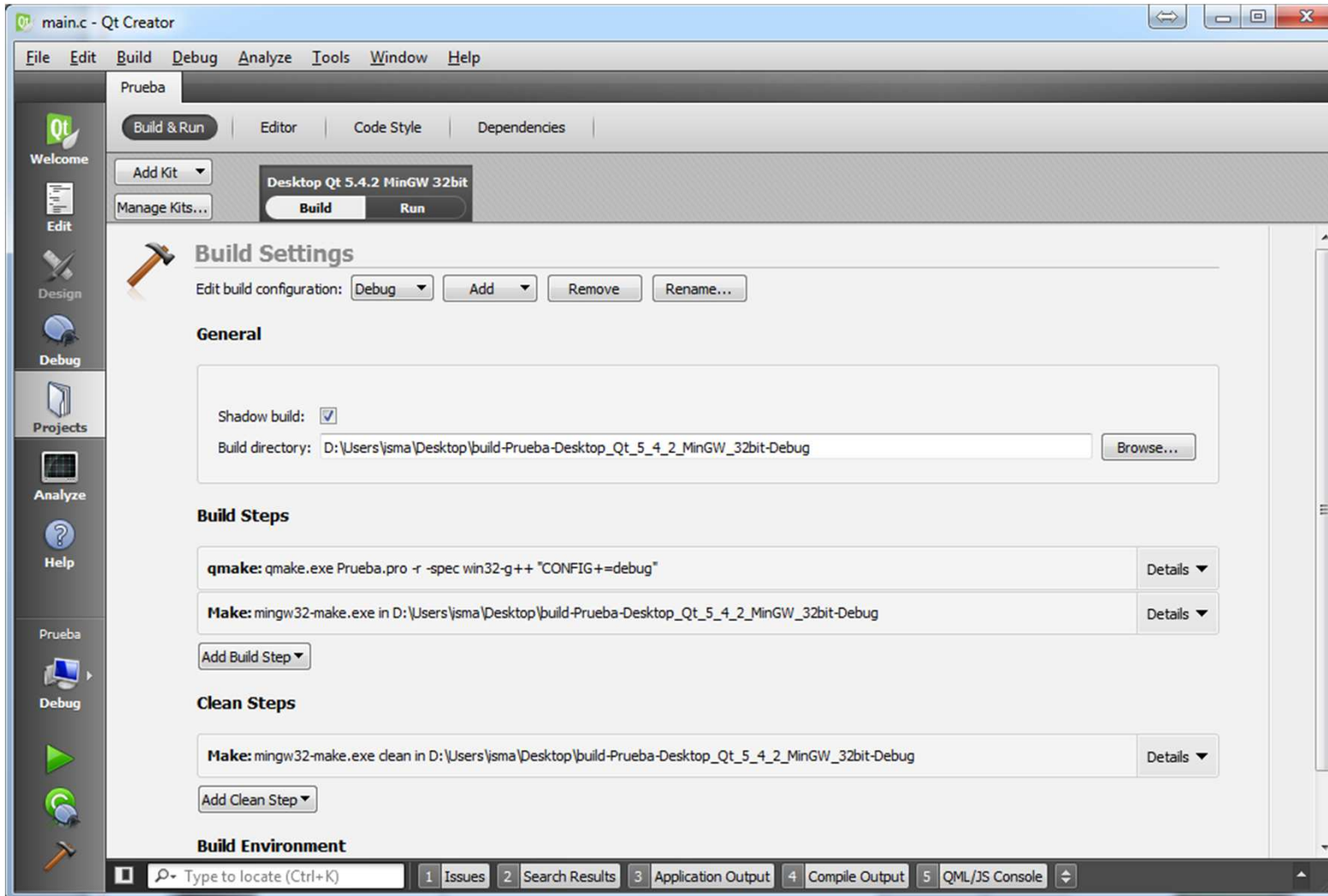
QT-EL EDITOR



QT-EL DEPURADOR

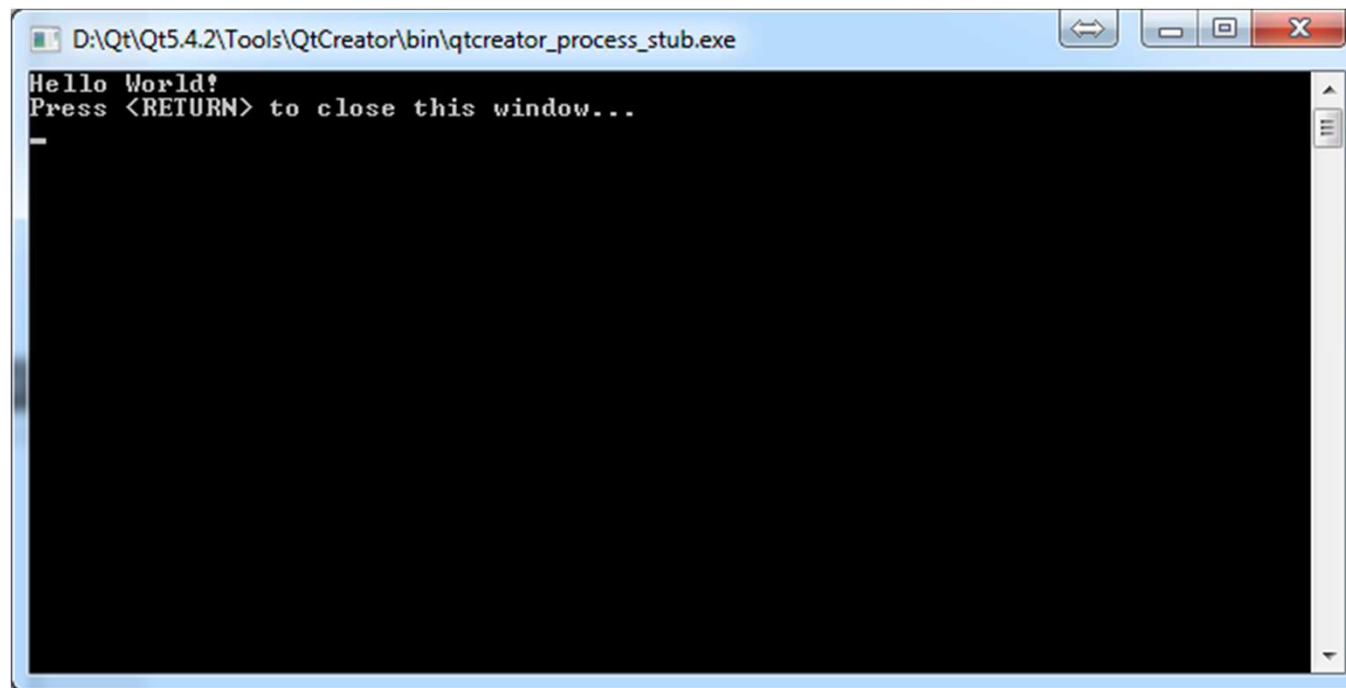


QT-OPCIONES DE NUESTRO PROYECTO



QT CREATOR

- Si todo va correctamente y pulsamos en ejecutar (botón de play) deberá salir esto:



6- INTRODUCCIÓN/REPASO ANSI C



HISTORIA

- ANSI C es un estándar publicado por el Instituto Nacional Estadounidense de Estándares (ANSI) para el lenguaje de programación C.
- Se recomienda que los desarrolladores de C cumplan los requisitos de ANSI para facilitar la portabilidad del código.

HISTORIA

- C es un lenguaje de programación originalmente desarrollado por Dennis M. Ritchie entre 1969 y 1972



HISTORIA

- Es una evolución de B y está basado en BCPL.
- Es un lenguaje orientado a implementar Sistemas Operativos, concretamente Unix.
- C es apreciado por su eficiencia
- Es el lenguaje de programación más popular para crear software de sistemas, aunque también permite crear aplicaciones completas.

BIBLIOGRAFÍA RECOMENDADA:

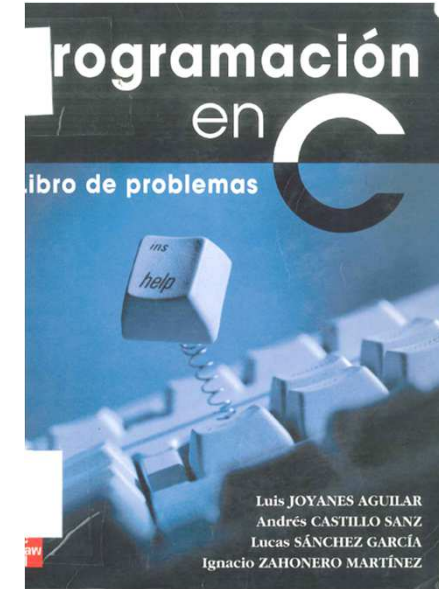
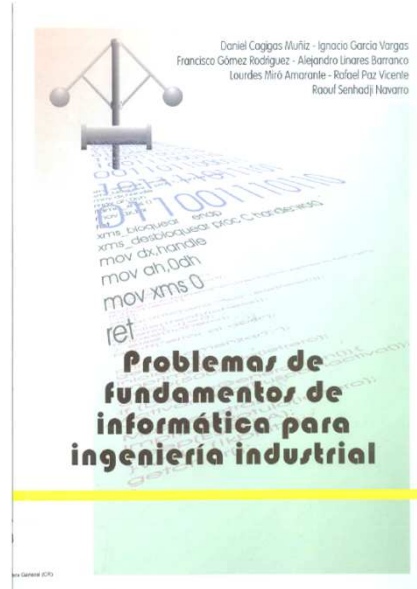


J.D. Muñoz, R. Palacios,
*Fundamentos de Programación
Utilizando el Lenguaje C*, ICAI-
Universidad Pontificia de Comillas,
2006.

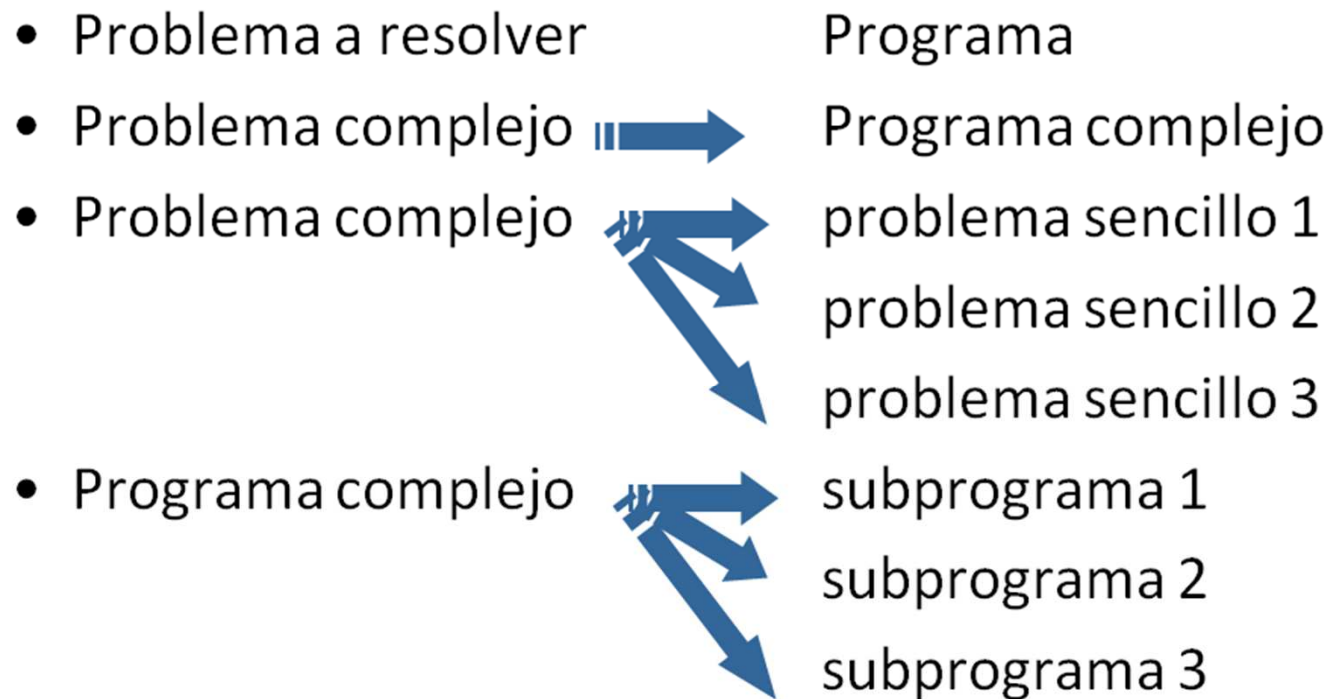


B. Gottfried, *Programación en C*,
Serie Schaum, McGraw-Hill, 2005.

BIBLIOGRAFÍA RECOMENDADA:



COMO CREAR UN PROGRAMA CON C → PROGRAMACIÓN ESTRUCTURADA



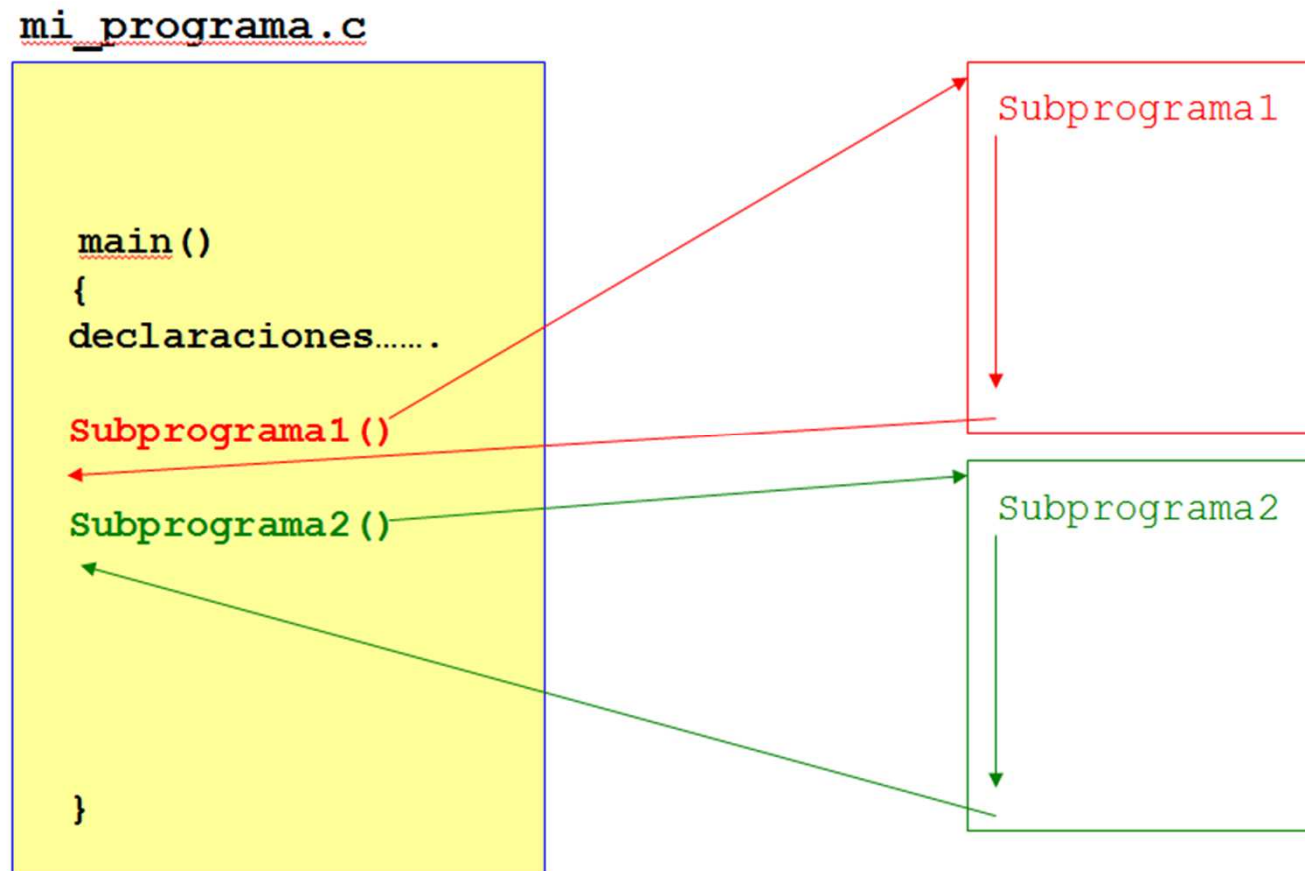
¿CUÁNDO USAR SUBPROGRAMAS?

- Programas **extensos** o **complejos**
- **Repetición** de acciones.
- Definir subprogramas con **suficiente entidad**

SUBPROGRAMAS

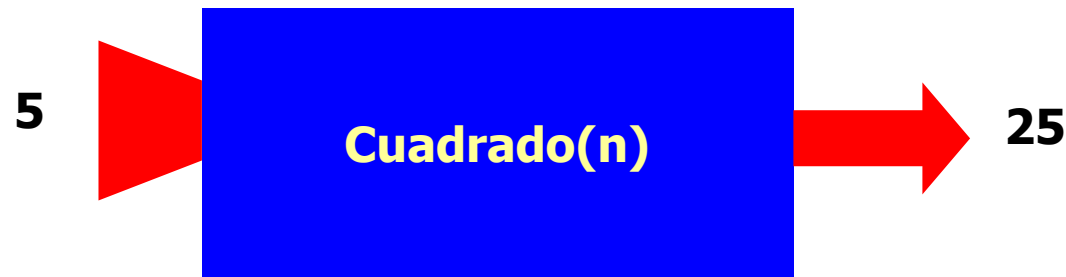
- Recurso de programación presente en la mayoría de los lenguajes, donde se llaman PROCEDURES, FUNCTIONS, SUBROUTINAS... Etc.
- En Lenguaje C son las FUNCIONES

SUBPROGRAMAS



SUBPROGRAMAS

- Los subprogramas en C son iguales a funciones
- El objetivo de una función es realizar alguna operación a partir de los parámetro de entrada. Por ejemplo:



FUNCIONES ESTÁNDAR DE C

- Matemáticas

 - `sin(parámetros), cos(), tan(), sqrt() ...`

- Entrada/Salida

 - `printf(argumentos)`
 - `scanf(argumentos)`
 - `fprintf(argumentos)`
 - ...

- Para cadenas de caracteres, tiempo, etc.

- Hay que especificar cabeceras con *#include* (ver más adelante)

CÓMO SE DEFINE UNA FUNCIÓN EN C

- Estructura análoga a la función `main()`
- `main()` es la función principal del programa
- Al resto se las identifica con nombres

```
Prototipo de la función;  
...  
Cabecera de la función  
{  
    declaraciones  
    secuencia de instrucciones  
}
```

CABECERA O PROTOTIPO

tipo_resultado **Identificador**(**Argumentos**);

- **Identificador...**
 - Nombre simbólico para invocar la función (el nombre de la función)
 - Explicativo y conciso
 - Identificador válido en C
- **tipo_resultado...**
 - Es el tipo del dato que se devuelve: **int**, **char**, **float**, ...
 - Si no se devuelve ningún valor se debe poner **void**
- **Argumentos...**
 - Declarar datos que se quieren pasar: **, tipo identificador,**

CABECERA O PROTOTIPO: EJEMPLOS

```
int    Cuadrado(int numero);  
float  Perimetro(float ax, float ay,  
                float bx, float by,  
                float cx, float cy);  
float  Area_triangulo(float base,  
                    float altura);  
  
void    Imprimir_resultado(int resultado);  
void    Leer_coordenadas(void);
```

EJEMPLO DE FUNCIÓN

```
float Area_triangulo(float base, float altura)
{
    /* declaración de variables */
    float area;
    /* Expresiones con variables y parámetros
*/
    area = base*altura*0.5;
    /* devolución de resultados */
    return area;
}
```


¿DÓNDE DEFINIR EL CÓDIGO DE UNA FUNCIÓN?

```
float Area_triangulo(float base,float altura)
{
    float area;
    area = base*altura*0.5;
    return area;
}

main()
{
    .....
}
```

¿DÓNDE DEFINIR EL CÓDIGO DE UNA FUNCIÓN?

```
float Area_triangulo(float base,float altura);

main()
{
.....
}

float Area_triangulo(float base,float altura)
{
    float area;
    area = base*altura*0.5;
    return area;
}
```

DÓNDE INVOCAR UNA FUNCIÓN

- Dentro de las llaves del main o dentro de cualquier otra función

```
main( )  
{  
    float area;  
    area = Area_triangulo(3.0,4.0);  
    printf("área del triangulo es %f \n",area);  
}
```

SENTENCIA RETURN

- Devuelve el valor que calcula la función y finaliza la ejecución de la misma. Ejemplos:
 - valor constante `return -1; return FALSE;`
 - contenido de una variable `return area;`
 - resultado de una expresión: `return area*altura;`
- En una función puede haber varios `return`, pero sólo se ejecuta uno

SENTENCIA RETURN

```
float Maximo(float valor1, valor2)
{
    if (valor1 > valor2)
        return valor1;
    else
        return valor2;
}
```

TIPOS DE DATOS DE C

Tipo	Tam. Bits	Dígitos de precisión	Rango	
			Min	Max
Bool	8	0	0	1
Char	8	2	-128	127
Signed char	8	2	-128	127
unsigned char	8	2	0	255
short int	16	4	-32,768	32,767
unsigned short int	16	4	0	65,535
Int	32	9	-2,147,483,648	2,147,483,647
unsigned int	32	9	0	4,294,967,295
long int	32	9	-2,147,483,648	2,147,483,647
unsigned long int	32	9	0	4,294,967,295
long long int	64	18	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long long int	64	18	0	18,446,744,073,709,551,615
Float	32	6	1.17549e-38	3.40282e+38
Double	64	15	2.22507e-308	1.79769e+308

TIPOS DE DATOS DE C

- El lenguaje C depende directamente de la arquitectura donde se ejecute el programa, por lo tanto, el tamaño y rango pueden variar.
- ¿Cómo podemos conocer el tamaño de nuestras variables?

```
main(){  
    int tamanno;  
    tamanno = sizeof(int);  
    printf("El tamaño de un int es de %d  
bits \n", tamanno*8);  
}
```

TIPOS DE DATOS DE C

- Las variable pueden ser:
 - Locales (a la función)
 - Globales (accesibles desde todo el programa)

TIPOS DE DATOS DE C

- Ejemplo de variables globales vs locales:

```
// Global variable declaration:  
int g;  
  
int main () {  
    // Local variable declaration:  
    int a, b;  
  
    // actual initialization  
    a = 10;  
    b = 20;  
    g = a + b;  
  
    printf( "%d \n", g );  
    return 0;  
}
```

TIPOS DE DATOS DE C

- Además podemos definir nuevos tipos de datos usando:
 - typedef type nombre. Ejemplo:

```
typedef int milla;  
  
main(){  
    milla = 20;  
    printf(``El número de millas es %d  
\n``, millas);  
}
```

TIPOS DE DATOS DE C

- También podemos lista de nombre como una estructura de datos usando:
 - Enum enum-name {lista de nombres} variable_list

```
main(){  
    enum color{rojo, azul, verde } c;  
    c = blue;  
}
```

TIPOS DE DATOS DE C

- La declaración de variable acepta los siguiente modificadores:
 - **static** → el valor de la variable se conserva entre llamadas. Como si fuera una variable global. Ej.:

```
void cuenta()  
{  
    static int cnt=0;  
    printf("%d\n",cnt++)  
}  
int main(){  
    cuenta();cuenta();cuenta();cuenta();  
    return 0;  
}
```



TIPOS DE DATOS DE C

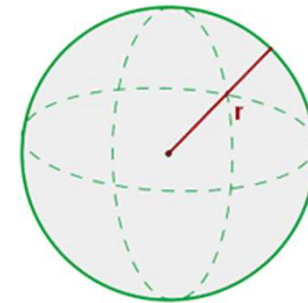
- **const** → la variable no puede ser modificada. Ej.:

```
const int mi_variable = 150;
```

EJERCICIO

- Crear una función C que calcule el área de una esfera y la devuelva a partir de su radio.

- Recuerda que la fórmula es:
- La función ***pow(base, exponente)*** calcula la potencia de un número
- Debemos incluir ***#include <math.h>*** que es la librería matemática de c
- Pi se puede calcular como: ***double pi = acos(-1)***



$$V = \frac{4}{3} \pi \cdot r^3$$

EJERCICIO (II)

- Pedir el radio de la esfera al usuario
- Mostrar el resultado por pantalla

PREPROCESADOR EN C

- Conjunto de instrucciones o directivas que se ejecutan antes del procesamiento del código fuente en C.
- Sirve para modificar los ficheros fuente antes de su compilación.
 - La modificación puede ser en cualquier punto
 - Inclusión de un fichero fuente dentro de otro
 - Sustitución de identificadores por expresiones

PREPROCESADOR EN C

- o `#directiva` argumentos

- o **directivas principales:**

- `#include`
- `#define, ifdef, endif, undef`

PREPROCESADOR EN C

- **#define** → Sustituye todas las apariciones de un identificador por una expresión:
- Ejemplos:
 - **#define** Pi 3.141592
 - **#define** tamaño 23

PREPROCESADOR EN C

```
#define DEBUG
int main()
{
    int i,acc;
    for(i=0;i<10;i++)
        acc=i*i-1;
#ifdef DEBUG
    printf("Fin bucle acumulador: %d",acc);
#endif
    return 0;
}
```

PREPROCESADOR EN C

- **#include** → Incluye el contenido de un fichero dentro de otro en el punto en el que aparece
 - `#include <archivo.h>`
 - `#include <stdio.h>`
 - `#include <math.h>`

PREPROCESADOR EN C

Inclusión condicional → Si en dos ficheros .c tengo `#include "Fichero.h"` se intentaría declarar cosas dos veces
=> Error!

Solución:

```
#ifndef FICHERO_H
#define FICHERO_H
    /* contenido que queremos
    incluir */
#endif
```

PREPROCESADOR EN C

aux.h

```
#ifndef _AUX_H_
#define _AUX_H_
<definiciones>
#endif
```

Evita la redefinición de funciones y variables

```
#include "aux.h"
#include "aux.h"
int main()
{
...
}
```

OPERADORES

- Operadores Aritméticos (A=10, B=20):
 - + → suma de dos operadores. Ej.: $A + B \rightarrow 30$
 - - → resta los dos operadores. Ej.: $A - B \rightarrow -10$
 - * → multiplica los dos operadores. Ej.: $A * B \rightarrow 200$
 - / → divide los dos operadores. Ej.: $A / B \rightarrow 0.5$
 - % → resto de la división, módulo. Ej.: $A \% B \rightarrow 0$
 - ++ → incremento operador. Ej.: $A++$ ó $++A \rightarrow 11$
 - -- → decremento del operador. Ej.: $B--$ ó $--B \rightarrow 19$

OPERADORES

○ Operadores Relacionales (A=10, B=20):

- `==` → comprueba si son iguales. Ej.: `A == B` → false
- `!=` → comprueba si son diferentes. Ej.: `A != B` → true
- `>` → comprueba si es mayor. Ej.: `A > B` → false
- `<` → comprueba si es menor. Ej.: `A < B` → true
- `>=` → comprueba si es mayor o igual. Ej.: `A >= B` → false
- `<=` → comprueba si es menor o igual. Ej.: `A <= B` → true

OPERADORES

- Operadores Lógicos (A=1, B=0)
 - && → realiza la multiplicación lógica (AND). Ej.: A && B → false
 - || → realiza la suma lógica (OR). Ej.: A || B → true
 - ! → realiza la negación lógica (NOT). Ej.: !A → false

OPERADORES

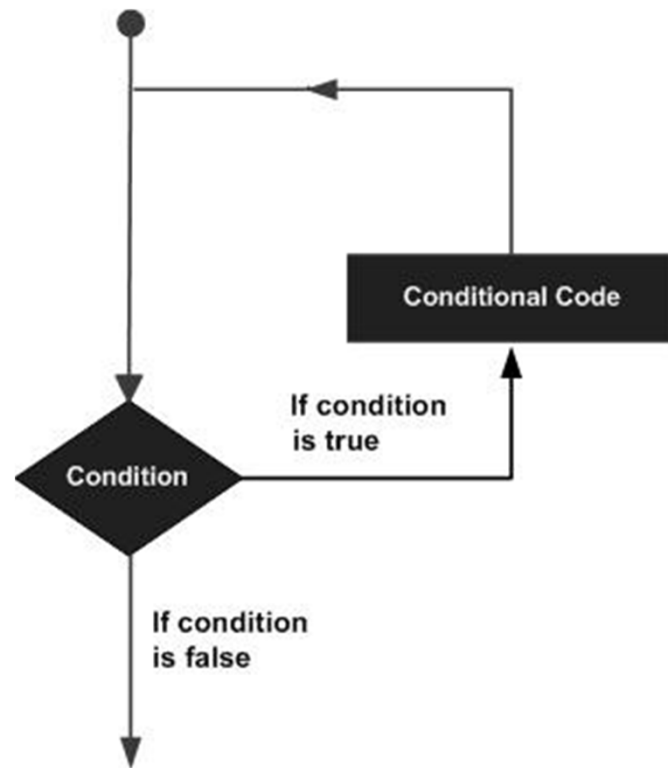
- Operadores a nivel de bits (A=60 (0011 1100), B=13 (0000 1101))
 - $\&$ \rightarrow realiza la operación lógica AND bit a bit. Ej.: $A \& B \rightarrow 12$ (0000 1100)
 - $|$ \rightarrow realiza la operación lógica OR bit a bit. Ej.: $A | B \rightarrow 61$ (0011 1101)
 - \wedge \rightarrow realiza la operación lógica XOR bit a bit. Ej.: $A \wedge B \rightarrow 49$ (0011 0001)

OPERADORES

- Operadores a nivel de bits (A=60 (0011 1100), B=13 (0000 1101))
 - \sim → realiza la operación lógica de negación bit a bit. Ej.: $\sim A \rightarrow 387$ (1100 0011)
 - \ll → desplazamiento hacia la izquierda a nivel de bits. Ej.: $A \ll 2 \rightarrow 240$ (1111 0000)
 - \gg → desplazamiento hacia la derecha a nivel de bits. Ej.: $A \gg 2 \rightarrow 15$ (0000 1111)

BUCLES

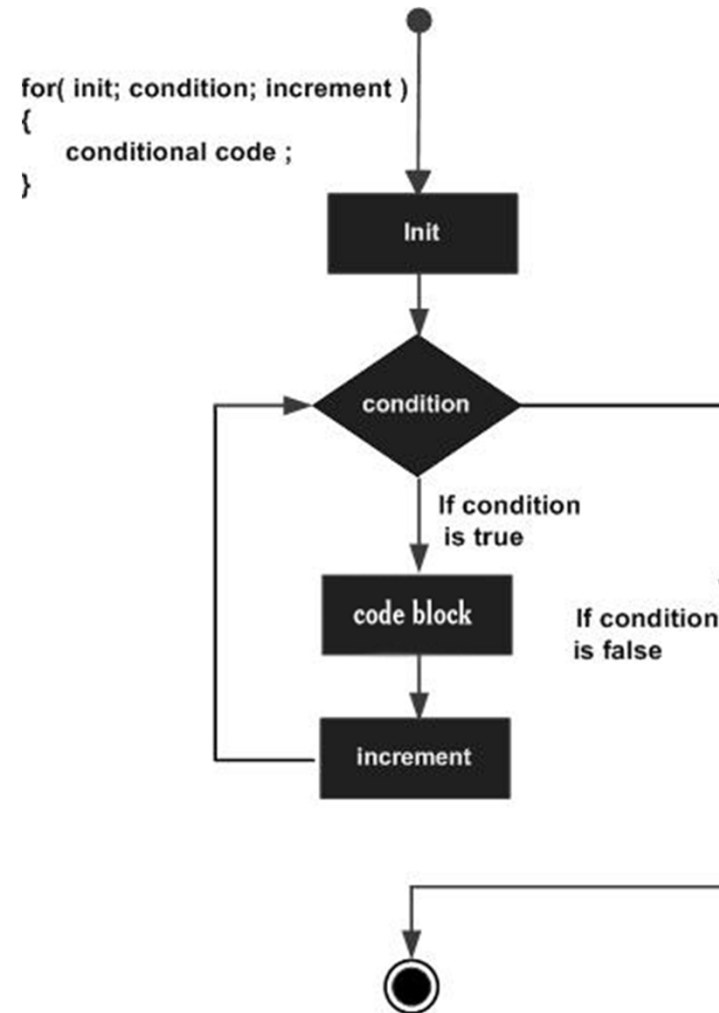
- En C existen 3 tipos de bucles



BUCLES

○ Bucle **for**. Ej.:

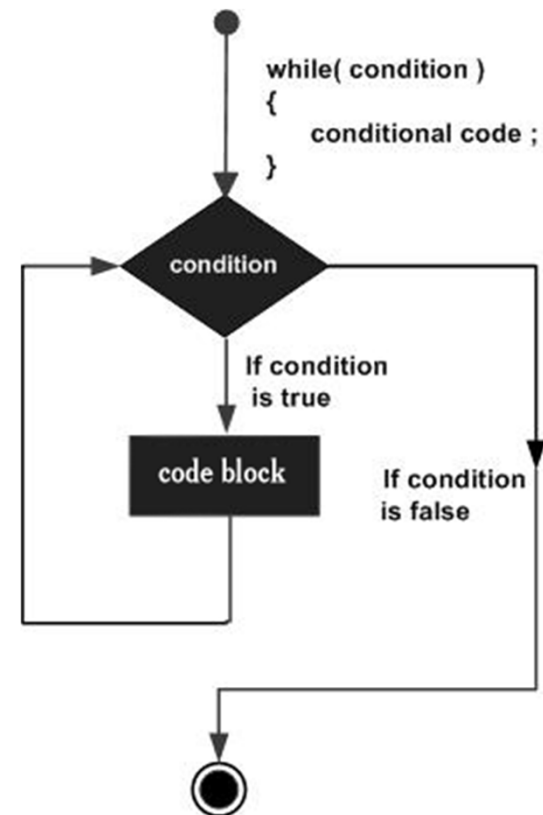
- **for** (int i = 0; i < 1000; i++){
 sentencias
 ...
}



BUCLES

○ Bucle **while**. Ej.:

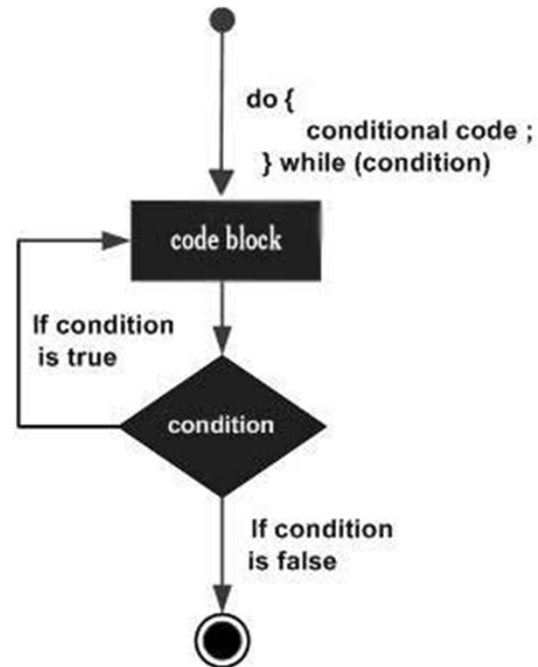
- **while**(a > 100){
 sentencias
 ...
}



BUCLES

- Bucle **do-while**. Ej.:

- **do** {
 sentencias
 ...
} **while**(a > 100);



BUCLES

- Sentencias de control de los bucles. Con estas sentencias cambiamos el ejecución normal del bucle cuando hemos alcanzado nuestro objetivo.
 - **break** → termina la ejecución del bucle y continua justo después de las llaves del bucle. Ejemplo:


```
for (int i = 0; i < 100; i++){  
    sentencias  
    ...  
    if (i == 88) break;  
}
```



BUCLES

- **continue** → termina la iteración actual del bucle y continua con la siguiente arriba. Ej.:

```
for (int i = 0; i < 100; i++){  
    sentencias  
    ...  
    if (i == 88) continue;  
}
```



BUCLES

- **goto** → realiza un salto incondicional a una sentencia etiquetada. Se recomienda no usar esta sentencia porque hacen que el programa sea difícil de entender. Ej.:

```
LOOP:do{  
  {  
    sentencias  
    ...  
    if (i == 88) goto LOOP;  
    ...  
  }while(a < 20);
```

EJERCICIOS

- Implementar la función `pow(base, exponente)`.
Crear «mi_pow» que calcule $\text{base}^{\text{exponente}}$.
Usando algunos de los bucle que hemos visto. Ej.:
 $10^3 = 1000$
- Crear una función que calcule el factorial de un número. Ej.: $5! = 5*4*3*2*1 = 120$
- (Un poco más avanzado) Crear una función que cree un triángulo de '*' conocida la base. Ej.:

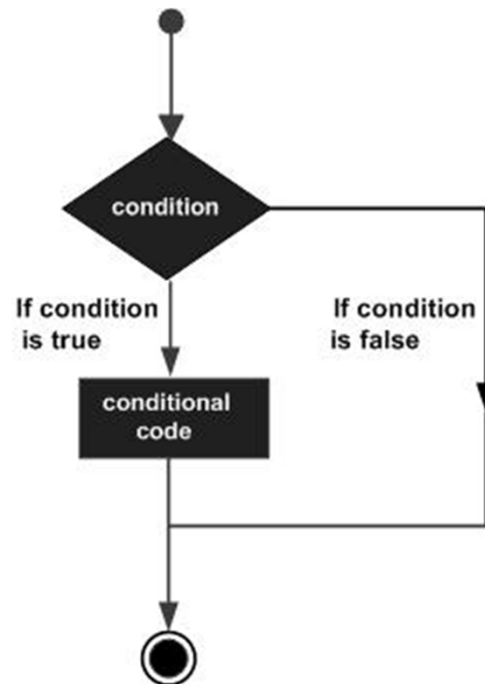
*

* *

* * *

DECISIONES

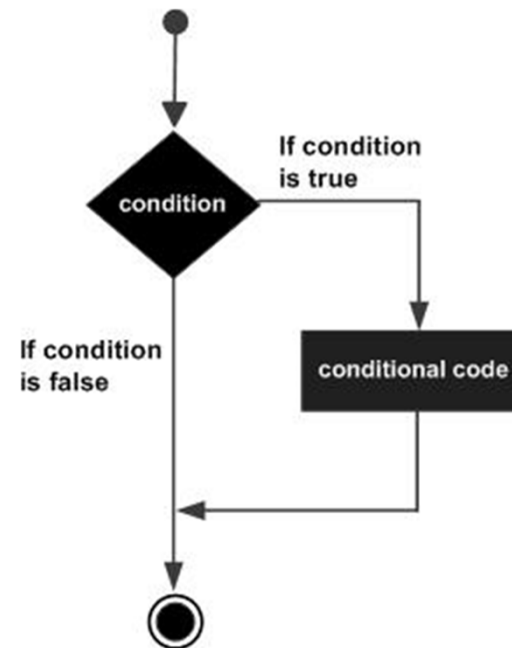
- En C existen 3 tipos de decisiones



DECISIONES

- **if** → si cumple la condición ejecuta el código que hay dentro, si no continua abajo. Ej.:

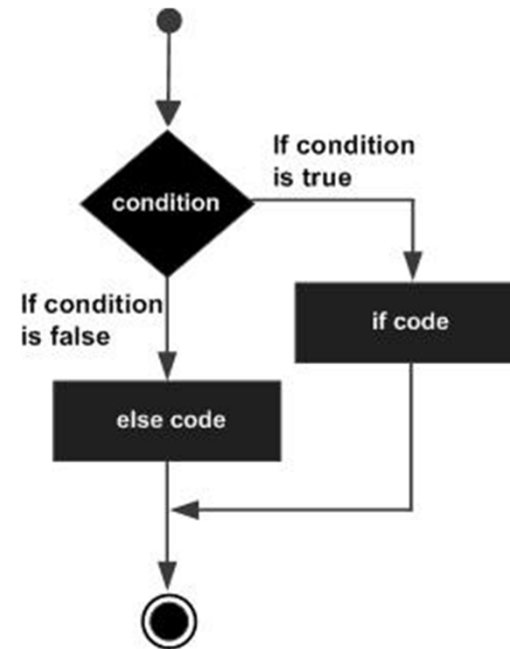
```
if (a == 5){  
    sentencias  
    ...  
}
```



DECISIONES

- **if-else** → si cumple la condición ejecuta el código de primer bloque, si no ejecuta el contenido del segundo bloque. Ej.:

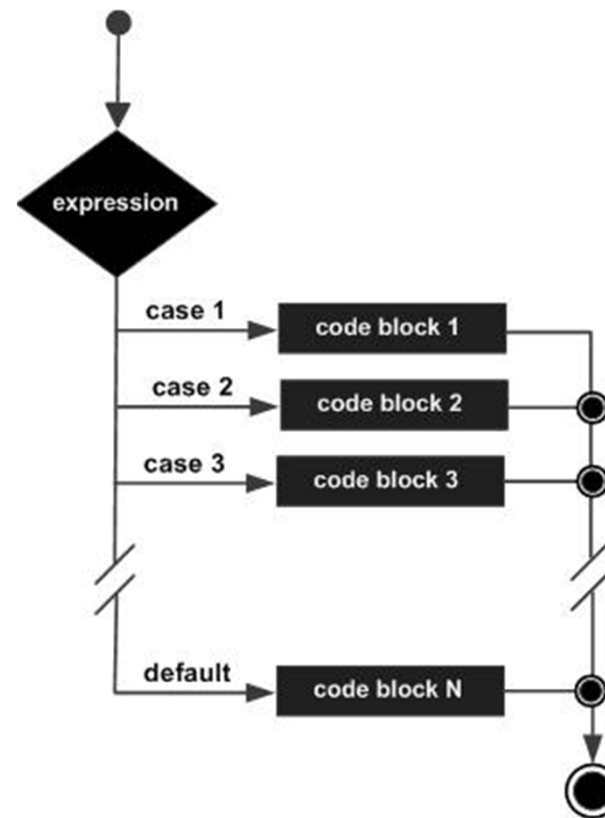
```
if (a == 5){  
    sentencias  
    ...  
}else{  
    sentencias  
    ...  
}
```



DECISIONES

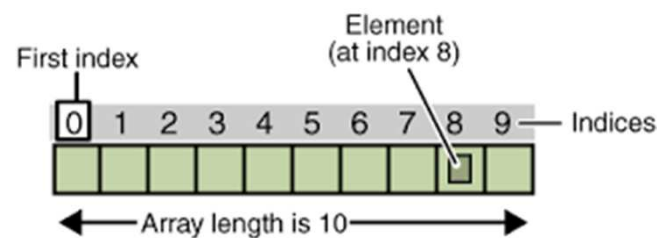
- **switch** → evalúa una expresión y ejecuta el bloque correspondiente. Ej.:

```
switch (a){  
    case 1: sentencias  
        ...  
        break;  
    case 2: sentencias  
        ...  
        break;  
    ...  
    default: sentencias  
        ...  
}
```



ARRAYS

- Un array es una estructura de datos que almacena elementos del mismo tipo.
- Nos pueden servir para generar vectores (array) o matrices (multi-array) del tamaño que necesitamos.



ARRAYS

- En la declaración de arrays, el programador (normalmente) debe especificar el número de elementos
 - *tipo_de_datos nombre [tamaño_del_array]*
- Ejemplos:
 - *double pesos[5] = {55.5, 45, 75.3, 88.1, 58}*
 - *int alturas[] = {180, 155, 175, 148, 195}*

ARRAYS

- Modificar valores el array:
 - *pesos[3] = 72.3;*
- Consultar valores del array:
 - *int altura_pedro = alturas[4];*

ARRAYS

- Además C permite definir multi-arrays de las dimensiones que necesitemos. La definición es así:
 - *Tipo_de_datos nombre [dim1][dim2]...[dimN]*
- Ejemplo:
 - *int a [3][4] = { {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };*

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

ARRAYS

- Existen 3 formas de pasar los arrays a una función:
 - Usando punteros (se verá más adelante)
 - *void miFuncion(int *nombre_array)*
 - Usando arrays parametrizados:
 - *void miFuncion(int nombre_array[tamaño])*
 - Usando arrays sin especificar el tamaño, aunque es recomendable pasar otro parámetro con la longitud:
 - *void miFuncion(int nombre_array[], int longitud)*

EJERCICIO

- Crear una función que sume dos vectores y muestre el resultado de la suma.
- (Modificación) Comprobar que tienen la misma longitud y devolver el resultado en otro vector.

ARRAYS DE CARACTERES

- Los Arrays de caracteres son una estructura de datos muy utilizada en C. Esta estructura permite almacenar cadenas de caracteres.

- Definición, inicialización y uso:

- char cadena[5] = "Hola";*

'H'	'o'	'l'	'a'	'\0'
-----	-----	-----	-----	------

cadena[0]	'H'
cadena[1]	'o'
cadena[2]	'l'
cadena[3]	'a'
cadena[4]	'\0'

ARRAYS DE CARACTERES

- Definición, inicialización y uso sin definir el tamaño del array:
 - *char cadena[] = "Hola";*

'H'	'o'	'l'	'a'	'\0'
-----	-----	-----	-----	------

cadena[0]	'H'
cadena[1]	'o'
cadena[2]	'l'
cadena[3]	'a'
cadena[4]	'\0'

ARRAYS DE CARACTERES

- Para la lectura de cadenas de caracteres tenemos dos opciones:
 - *scanf("%s", cadena)*
 - No &
 - No lee espacios es blanco. Posible necesidad de usar fflush()
 - *gets(cadena)*
 - No &
 - Posibilidad de espacios en blanco

ARRAYS DE CARACTERES

- Ejemplo:

```
char cadena [];  
{  
    printf("Introduzca una cadena ... ");  
    scanf("%s ", cadena);  
    ...  
    gets(cadena);  
}
```

ARRAYS DE CARACTERES

- Algunas funciones útiles para el manejo de arrays de caracteres son: (necesitamos incluir <strings.h>)
 - `strlen(cadena)` → devuelve la longitud de la cadena
 - `strcat(cadena1,cadena2)` → concatena las dos cadenas
 - `strncat(cadena1,cadena2,n)` → concatena n caracteres
 - `strcmp(cadena1,cadena2)` → compara las dos cadenas
 - `strchr(cadena,'c')` → busca el carácter 'c' en la cadena
 - ...

EJERCICIO

- Contar el número de vocales de una cadena introducida por el usuario
- Comprobar si la cadena introducida es un palíndromo.
- (Un poco más avanzado) Crear un codificador y decodificador de texto (código ASCII + 1)

ESTRUCTURAS DE DATOS

- C permite definir variables que contienen datos de diferentes tipos. Estas variables se les llama estructuras (struct).
- La definición es:

```
struct [nombre_estructura]  
{  
    tipo_dato nombre1;  
    tipo_dato nombre2;  
    ...  
}[uno o varios nombre de la estructura]
```

ESTRUCTURAS DE DATOS

- Como ejemplo vamos a crear una estructura para definir un coche. El coche tiene como atributos la matrícula, marca, modelo, número de pasajeros y el número de puertas.

```
struct Coche  
{  
    char matricula[MAX];  
    char marca[MAX];  
    char modelo[MAX];  
    int n_pasajeros;  
    int n_puertas;  
} coche1, coche2;
```

ESTRUCTURAS DE DATOS

- Ejemplo de uso de la estructura anterior:

```
main(){  
    struct Coche coche3;  
  
    strcpy(coche3.matricula, "06555GGG");  
    coche3.n_puertas = 3;  
  
    strcpy(coche1.matricula, "96244GHH");  
    coche1.n_puertas = 5;  
  
    ...  
}
```

EJERCICIO

- Crear un programa que pida al usuario los datos de diferentes coches (matrícula, marca, modelo, ...) y los vaya creando (usando la estructura que hemos visto antes).
- (Nota) Se puede crear un array de coche de esta forma:
`struct Coche coches[20];`