PRÁCTICA 0B

REPASO DE PROGRAMACIÓN EN C

ARQUITECTURA DE COMPUTADORES. 2º CURSO

- **OBJETIVOS**

El objetivo de esta práctica consiste en repasar y reforzar los conocimientos del alumno en el ámbito de la programación en lenguaje C.

Estos conocimientos son INDISPENSABLES de cara a afrontar las prácticas que se impartirán en semanas posteriores; relacionadas con la programación de dispositivos de entrada/salida.

El repaso de dicho lenguaje abarcará desde una visión general de un programa básico hasta los conceptos más avanzados del lenguaje. Es por tanto ALTAMENTE RECOMENDABLE que el alumno acuda a la práctica con un repaso previo, en casa, del lenguaje de programación C.

Los objetivos concretos de la práctica se centran en repasar los siguientes conceptos:

- Elementos básicos y estructura principal de un programa en lenguaje C.
- Manejo del entorno Visual Studio.
- Depuración de programas.

- **INTRODUCCIÓN TEÓRICA.**

.1. ESTRUCTURA BÁSICA DE UN PROGRAMA EN C

En este apartado se expone un breve resumen de la programación en el lenguaje C. Se considera el siguiente contenido como un repaso de lo aprendido en otras asignaturas; es por ello que no se profundizará en su explicación.

El lenguaje C es puramente imperativo, de forma que no considera la creación de clases ni objetos; hay que recordar que este lenguaje es uno de los predecesores de los lenguajes orientados a objetos. La creación de un programa en este lenguaje consta de una serie de ficheros ".c" (ficheros de código) y ".h" (ficheros de cabecera). Por la simplicidad de los programas que se realizarán en esta práctica no será necesaria la creación de ningún fichero de cabecera y todo el código se implementará bajo el mismo fichero ".c".

La estructura general, ordenada, de un fichero ".c" puede considerarse:

a) Inclusión de librerías:
Estructura: ***#include <nombre_de_libreria.h>***
Ejemplo: ***#include <stdio.h>***

b) Definición de constantes:
Estructura: ***#define NOMBRE VALOR***
Ejemplo: ***#define MAX 128***

c) Definiciones teóricas de estructuras
Estructura: ***struct nombre_estructura{tipo campo1; tipo campo2; …};***
Ejemplo: ***struct miEstructura{int c1; float c2;};***

d) Prototipos de funciones:
Estructura: ***tipo nombre_funcion(parámetros);***
Ejemplo: ***int miFuncion(int, float);***

e) Declaración de variables globales:
Definición e inicialización idénticas a variables locales; únicamente varía su ámbito.

f) Función principal (main)
Según el compilador: ***main(){…}, void main(){…}, void main(void){…}, int main(){…}, …***

g) Implementación del resto de funciones

Los tipos básicos (atómicos) del lenguaje C son los comúnmente utilizados en la gran mayoría de los lenguajes: ***int y char (enteros); float y double (reales en punto flotante).***

A estos tipos se les puede aplicar una serie de modificadores de tipo: ***signed, unsigned, long y short***; y una serie de modificadores de comportamiento: ***static, auto, register, const y volatile.***

Los tipos compuestos de este lenguaje son, principalmente, dos:

a) Vectores:
- **i.** Unidimensionales: ***tipo nombre[tamaño]; / tipo nombre[tamaño]={valor0, valor1, …};***
- **ii.** Multidimensionales: ***tipo nombre[dimension0][dimension1][…];***

b) Estructuras de datos:
- **i.** Definición teórica de la estructura: ***struct nombreEst{tipo0 var0; tipo1 var1; …};***
- **ii.** Creación de variable: ***struc nombreEst nombreVar;***
- **iii.** Acceso: ***nombreVar.var0*** → operador "**.**".

Un ejemplo global es el siguiente:

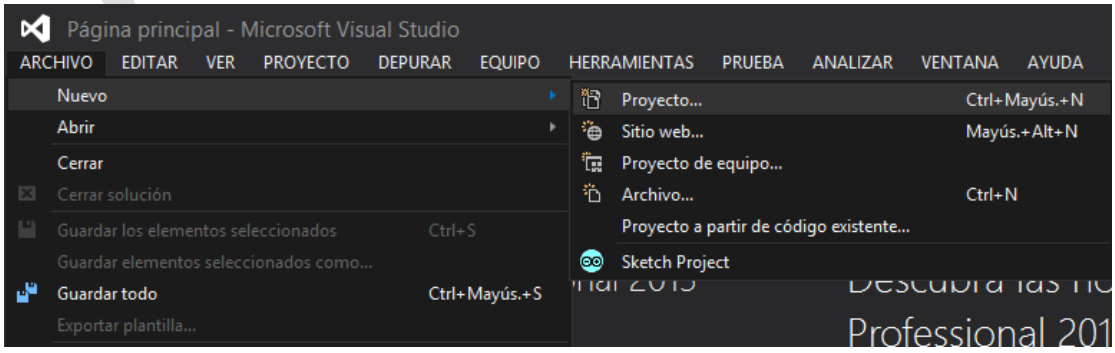
```
#include <stdio.h>
#define PI 3.141592

struct DatosCircunferencia{
	float area;
	float perimetro;
};

struct DatosCircunferencia calculaDatos (float);

main (){
	float radio;
	struct DatosCircunferencia res;
	printf("Calculo del area y perimetro de una circunferencia dado el radio\n");
	printf("Introduzca el radio de la circunferencia:   ");
	scanf("%f", &radio);
	res = calculaDatos (radio);
	printf("El resultado ha sido\n Area: %f\n Perimetro: %f", res.area, res.perimetro);
}
struct DatosCircunferencia calculaDatos (float r){
	struct DatosCincunferencia dat;
	dat.area = PI*r*r;
	dat.perimetro = 2*PI*r;
	return dat;
}
```

.2. ENTORNO DE PROGRAMACIÓN: VISUAL STUDIO

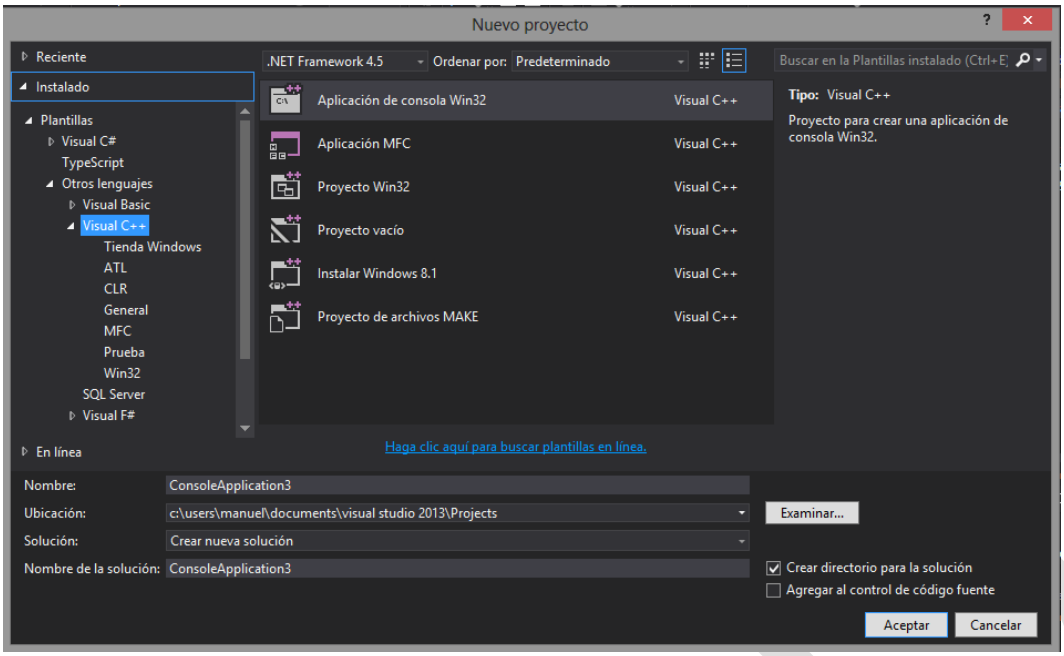
Es importante denotar que el entorno utilizado ha sido Visual Studio por su potencia y facilidad para el usuario (además de ser gratuito para alumnos de la escuela); pero, si lo desea, puede hacer uso de otro entorno sin ningún problema.

A la hora de crear un proyecto nuevo en Visual Studio se creará un proyecto para C++ y, posteriormente, se trabajará con la sintaxis de C (excepto casos puntuales, C++ admite la nomenclatura de ANSI C).

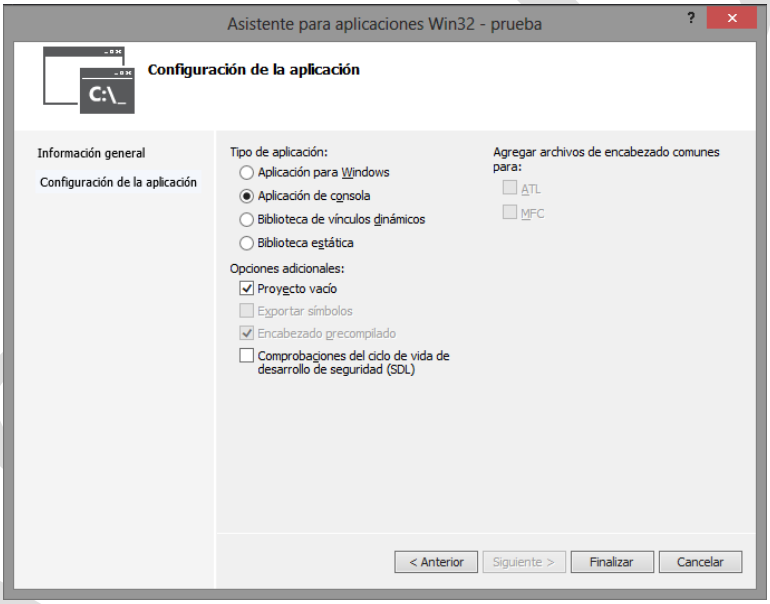
Para crear un proyecto nuevo, accedemos a la opción "Archivo" → "Nuevo" → "Proyecto":



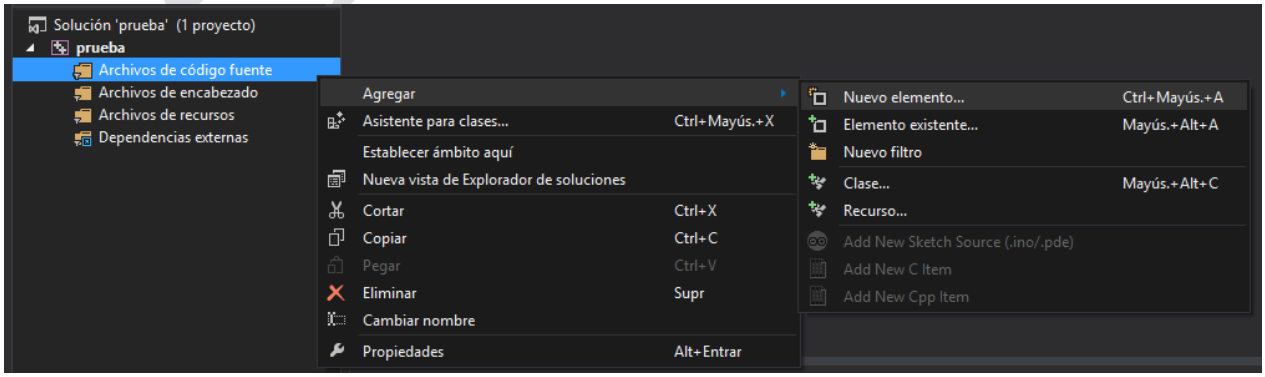
A continuación, aparecerá un cuadro de diálogo que nos permitirá seleccionar el tipo de proyecto a crear. Se hará uso de un proyecto de consola de comandos para C++:



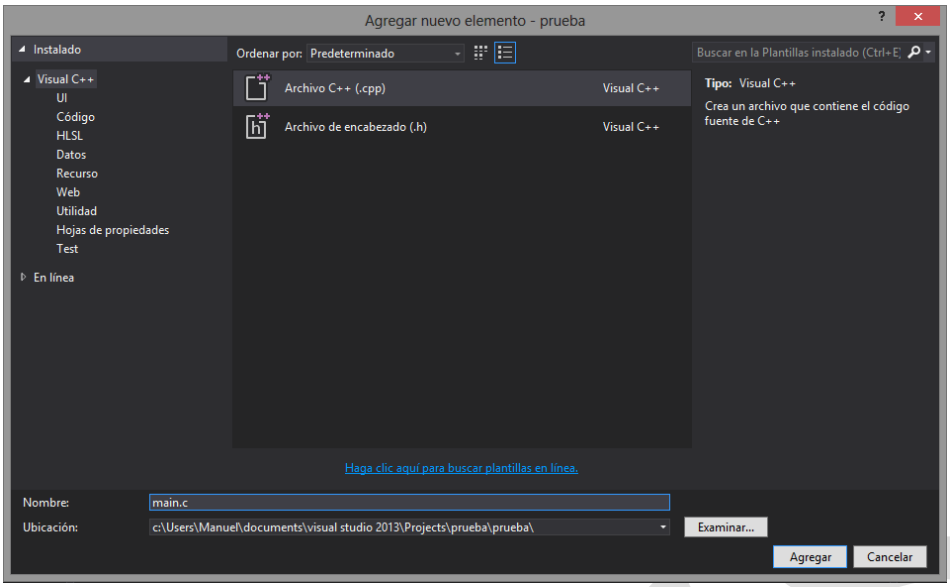
Una vez que se elige el tipo de proyecto y se le aporta un nombre, aparecerá un wizard que nos guiará para introducir una serie de opciones en el proyecto. Principalmente, en nuestro caso se seleccionará un proyecto vacío de consola y se desactivará cualquier otra opción:



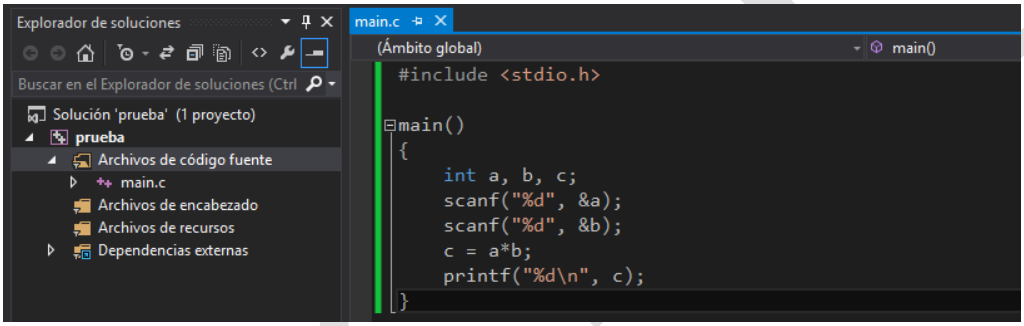
Una vez creado el proyecto, se crearán automáticamente tres subcarpetas. Sobre la carpeta de “Archivos de Código Fuente”, seleccionaremos la opción de “Nuevo Elemento” con el botón secundario:



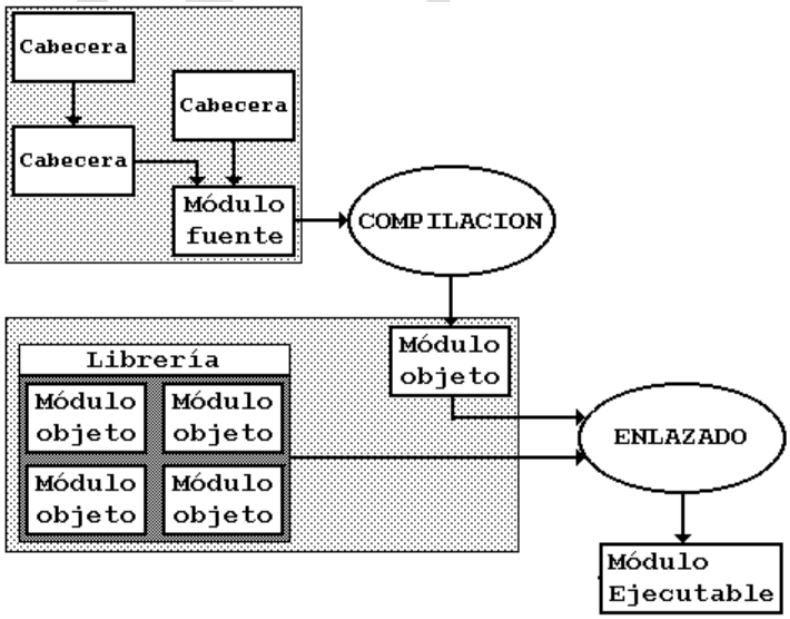
Se creará un archivo de C++ y se renombrará con la extensión “.c”:



Para probar el funcionamiento del entorno, escriba el siguiente código de ejemplo:



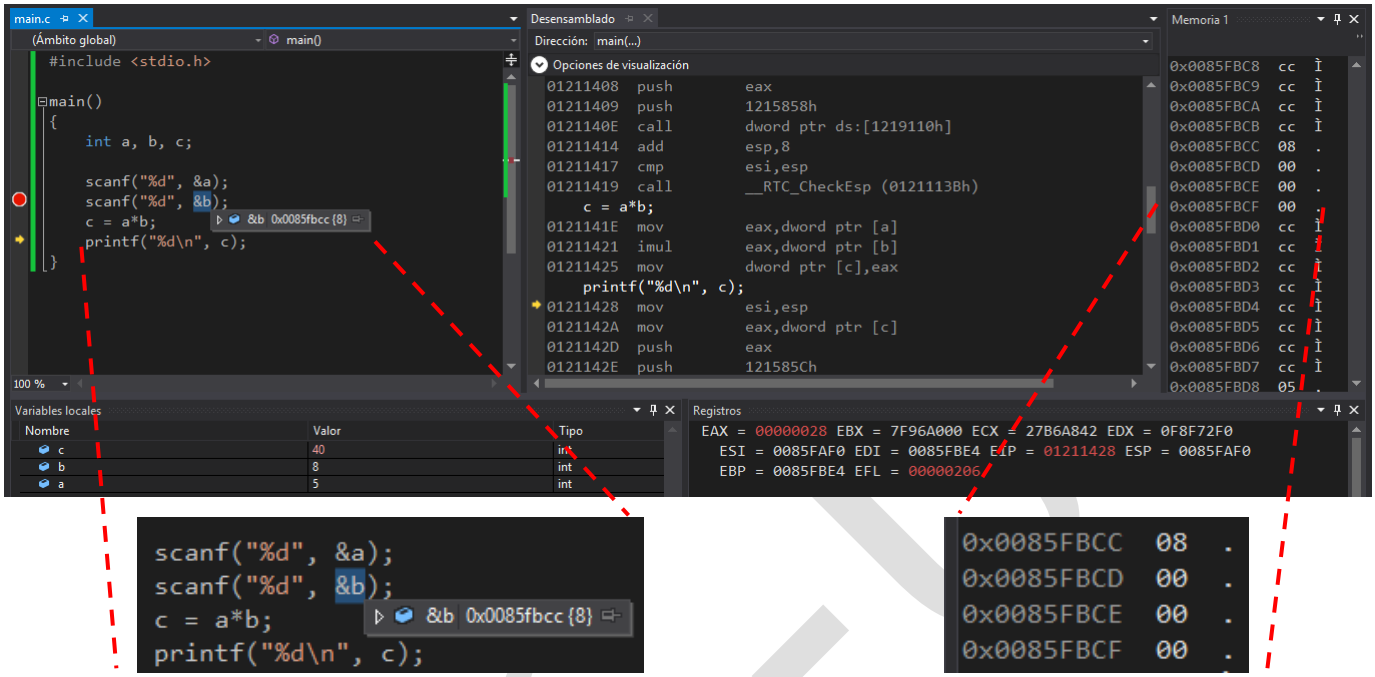
Proceso de compilación de un programa:



Todo entorno de programación, a la hora de realizar la compilación del programa, puede llevarla a cabo en dos modos principales: modo depuración o modo ejecución ("*Release*" en Visual Studio).

A grandes rasgos, la diferencia de compilar en modo "*Debug*" o "*Release*" radica en que, el primer modo, se utiliza en el proceso de testeo de la aplicación, almacenando información importante para permitir ejecutar el código paso a paso. Además, por defecto en este modo, se crea un archivo de extensión ".pdb" (programmer database) que mapea desde el código intermedio generado (MSIL) al código fuente, permitiendo establecer la correspondencia entre estos dos (ventana de depuración de lenguaje ensamblador). Por otra parte, el modo "*Release*" es utilizado para compilar una aplicación final, eliminando toda la información intermedia necesaria para la depuración, además de aplicarle todas las optimizaciones de ejecución posibles y mejorar el aspecto de seguridad.

Por defecto, en Visual Studio, un proyecto está configurado para ejecutarse en modo *"Debug"*, así pues si presionamos F5 se ejecutará la compilación, *linkado* y ejecución en modo depuración. Si, por el contrario, queremos ejecutarlo en modo "*Release"*, no tiene más que presionar *Ctrl* + F5. El entorno Visual Studio permite, en el modo depuración, introducir puntos de ruptura, visualizar el valor de las variables, visualizar el contenido de memoria y de los registros e, incluso, acceder al código ensamblador generado. En la siguiente pantalla se puede apreciar todo lo descrito anteriormente:



RECURSOS:

En internet pueden encontrarse números cursos y tutoriales donde encontrar explicaciones detalladas del lenguaje C. A continuación, se muestran algunos de ellos

- Material didáctico de la asignatura "Fundamentos de programación" de primero disponible en rodas.us.es:

  http://rodas.us.es/items/5c12ed9d-e398-4570-a30c-991953e92eed/1/

  http://rodas.us.es/items/c3145fb0-ed17-4070-9bef-7ec707caa971/1/

- Blog "La programación no es un arte" del Prof. Riquelme:

  http://laprogramacionnoesunarte.blogspot.com.es/

- Libro electrónico de libre acceso: "Aprenda lenguaje ANSI C como si estuviera en primero"

  http://www.josedomingo.org/web/mod/resource/view.php?id=806

CUESTIONARIO DE CONOCIMIENTOS PREVIOS

PRACTICA 0b - Repaso de programación en C

1. Indique si el computador en el que se han ejecutado los ejemplos mostrados más arriba es Big Endian o Little Endian. Si no es posible saberlo, indique qué necesitaría para ello. En caso de ser posible, indique cómo lo ha sabido.

2. Observe el juego de instrucciones en ensamblador del procesador y los registros de usuario. ¿Es normal que haya tan pocos registros? ¿A qué se puede deber?

3. Indique, al menos, 5 errores cometidos en el siguiente código (para lenguaje ANSI C) y qué haría para solucionarlos.

```
#include <stdio.h>
main(){
    unsigned int res, num;
    scanf("%f", num);
    res = fib(num);
    printf("Resultado: \n", res);
}
void fib(unsigned int n){
    int a=0, b=1, r;
    if(n=0)
       r = a;
    if else(n=1)
       r = b;
    else
       for(int i=2; i<n; i++)
       {
              r = a + b;
              a = res;
              b = r;
       }
}
```

4. Rellene los huecos para que el programa de la derecha tenga el mismo comportamiento que el de la izquierda.

Programa con paso de parámetros por valor

```
float calcArea(float r);
main(){
    float radio, res;
    scanf("%f", &radio);
    res = calcArea(radio);
    printf("Area:  %.2f\n", res);
}
float calcArea(float r){
    return (3.141592*r*r);
}
```

Programa con paso de parámetros por referencia

```
void calcArea(float r, __________);
main(){
    float radio, res;
    scanf("%f", &radio);
    calcArea(radio, __________);
    printf("Area:  %.2f\n", __________);
}
void calcArea(float r, __________){
    ___________ = 3.141592*r*r;
}
```

5. Rellene los siguientes apartados.	
Defina un tipo de estructura llamada ***coche***, que contenga internamente dos campos: una cadena de 50 caracteres que indique el ***tipo*** *de coche* y un número entero con el *número de* ***bastidor***.	
Cree una variable del tipo estructura definido antes, y llámela ***miCarro***.	
Modifique el tipo de coche de la variable ***miCarro***, por ***"Delorean DMC-12"***.	
Cree un vector bidimensional de elementos de tipo entero, de 50 filas y 20 columnas. Llámelo ***matriz***.	
Modifique el valor del elemento situado en la primera fila y última columna por ***32***.	

6. Rellene el siguiente programa en C para que almacene en un vector (v2) los elementos pares de otro vector (v1) y, finalmente, muestre por pantalla el resultado.

```
#include <stdio.h>
main()
{
    int v1[50]={/*Una serie de valores enteros*/}, v2[50];









}
```