SISTEMAS PARALELOS Y DISTRIBUIDOS.
3º Grado en Ingeniería Informática - Ingeniería de Computadores
PRÁCTICA: REPASO DE AC DE 2º Y ENTORNO VISUAL STUDIO

OBJETIVOS:

- Repaso de AC (2º curso):
  - instrucciones RISC usuales,
  - fallos de caché y cálculo básico del Miss Rate (razón de fallos)
  - dependencias RAW, WAW y WAR
- Manejar VS (Visual Studio)
  - Creación de un proyecto
  - Configuraciones de compilación
- Medición simple de tiempos de ejecución
- Desensamblador de VS
- Comparación de tiempos de ejecución de un programa
  - al compilarlo con opciones de optimización
  - al reescribirlo de forma eficiente

REPASO DE AC DE 2º

El alumno **debe realizar los ejercicios de repaso de AC** (desde el 1 hasta el 6 inclusive). Podría realizarse un test de evaluación en la EV sobre esto.

1. Indique cuál de las siguientes instrucciones suele existir en un RISC genérico, y sí es así, cuáles son sus registros fuente y destino.

	¿suele existir?	Reg Fuentes (solo para las que existan en RISC)	Reg Destinos
`ADDI R1, R1, 100`			
`BNEZ R3, etiq`			
`BGT R1, R2, etiq`			
`SF (F7)4, R1`			
`SF (R7)-4, F1`			
`SF (R7+R4)4, F1`			
`LW (R7)4, R1, R2`			

2. ¿Cuántas instrucciones de salto (condicional o incondicional) tiene una sentencia `if ( ) { }`?

   ¿Y una sentencia `if ( ) { } else { }` ?

   ¿Y un bucle `do { } while ( )` ?

3. Dibujar con flechas las dependencias RAW, WAW y WAR de esta traza de ejecución de dos iteraciones de un bucle (que empieza en la instrucción `LF`) ? ¿Habrá dependencias entre las dos iteraciones?

```
LF F0,0(R1),
ADDF F4,F0,F2
SF 0(R1),F4,
SUBI R1,R1,4
BNEZ R1,BUCLE
LF F0,0(R1),
ADDF F4,F0,F2
SF 0(R1),F4,
SUBI R1,R1,4
BNEZ R1,BUCLE
```

4. ¿Por qué son incorrectas las siguientes traducciones a ensamblador de un RISC genérico de este bucle? NOTA: no importa que la traducción sea más o menos óptima

```
float s=0.0, a[N];
for (i=N-1; i>=0; i--)  s = s + a[i];
     // inicialmente Ra apunta a:  a[N-1]. Ri contiene N-1. F1 contiene 0.0
```

etq: CARGA_float F0, (Ra)0 SUMA_float F1, F1, F0 RESTA_entero Ri, Ri, 1 COMPARA_>= R3, Ri, 0 SALTA_si_verdad R3, etiq	RAZONES:
etq: SUMA_float F1, F1, F0 CARGA_float F0, (Ra)0 RESTA_entero Ri, Ri, 1 SUMA_entero Ra, Ra, 4 COMPARA_>= R3, Ri, 0 SALTA_si_verdad R3, etiq	
etq: CARGA_float F0, (Ra)0 SUMA_float F1, F1, F0 RESTA_entero Ri, Ri, 1 SUMA_entero Ra, Ra, 1 SALTA_si_verdad Ri, etiq	

5. ¿Cuánto vale el Miss Rate (razón de fallos) del bucle si el tamaño de línea de caché es de 64 Bytes? NOTAS: suponed que la dirección de `a[0]` es múltiplo de 64 (comienzo de línea). Un float son 4 bytes.

VALOR del Miss Rate:	Bucle: `float s=0.0, a[4091];` `for (i=0; i<1024; i++)  a[i] = s + a[i];` Razón:

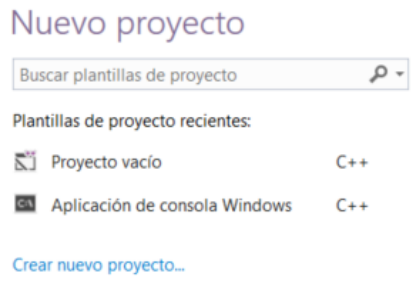
6. ¿Qué tamaño completo en bytes tendrá este vector de estructuras?

```
struct {
   float f;
   char c[4];
   int i[2];
   double d;
  } s[3];
```

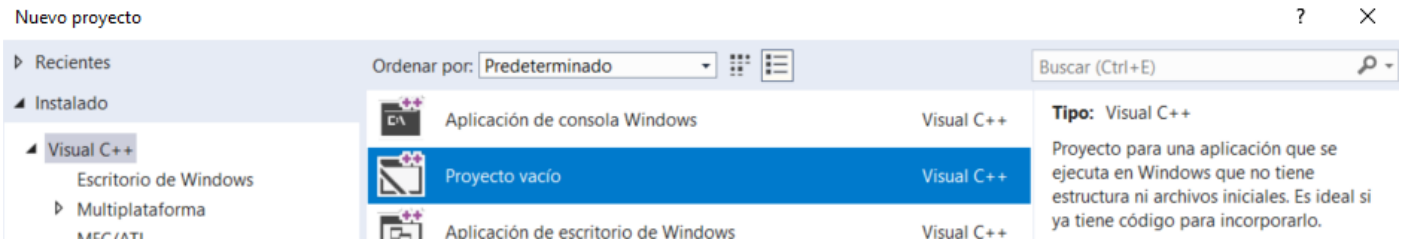
ENTORNO VISUAL STUDIO

Ejecutar Visual Studio.
Crear nuevo proyecto de consola vacío. Seguir estos pasos:

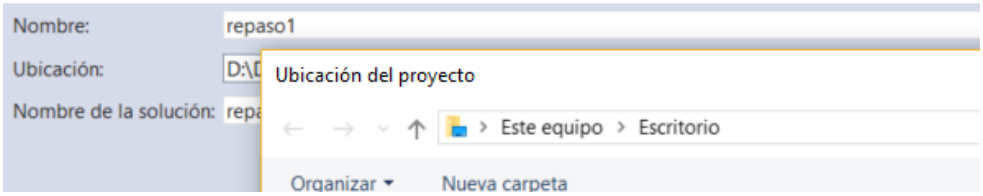
1. Hacer clic en: "`Crear nuevo proyecto`"



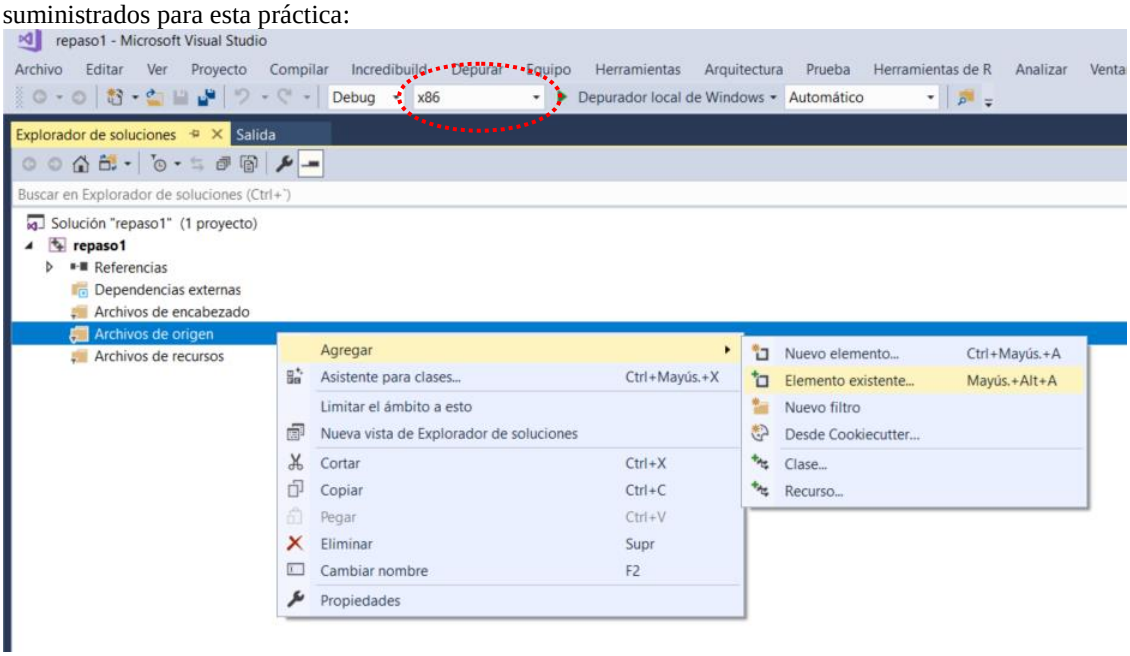
2. Hacer clic en: "`C++ → Proyecto vacío`"



3. Y ponerle un nombre; p ej. "`repaso1`", eligiendo la ubicación (carpeta) p ej. en el Escritorio:

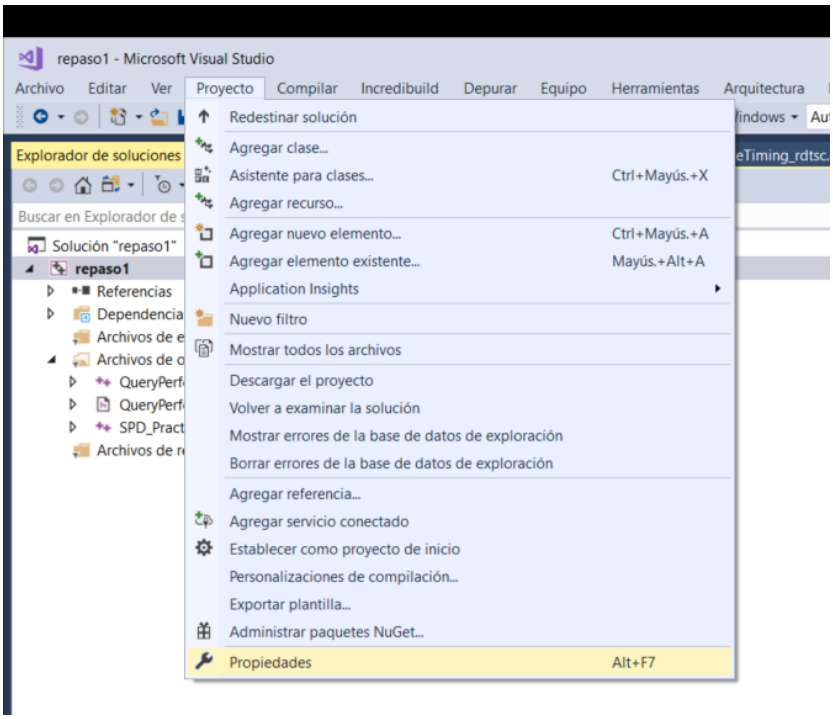


4. Copiar los ficheros dados en esta práctica de código fuente (*.cpp) y cabeceras (*.h) a la carpeta del proyecto (debe ser algo como "`Escritorio\repaso1\repaso1`") ya que se crea una carpeta nueva y con el mismo nombre para los códigos del proyecto.
5. Chequear que la arquitectura de este proyecto va a ser x86 (ver óvalo en captura siguiente)
   Después, con el botón derecho pulsando sobre "`Archivos de origen`", agregar los elementos (ficheros en nuestro caso) suministrados para esta práctica:

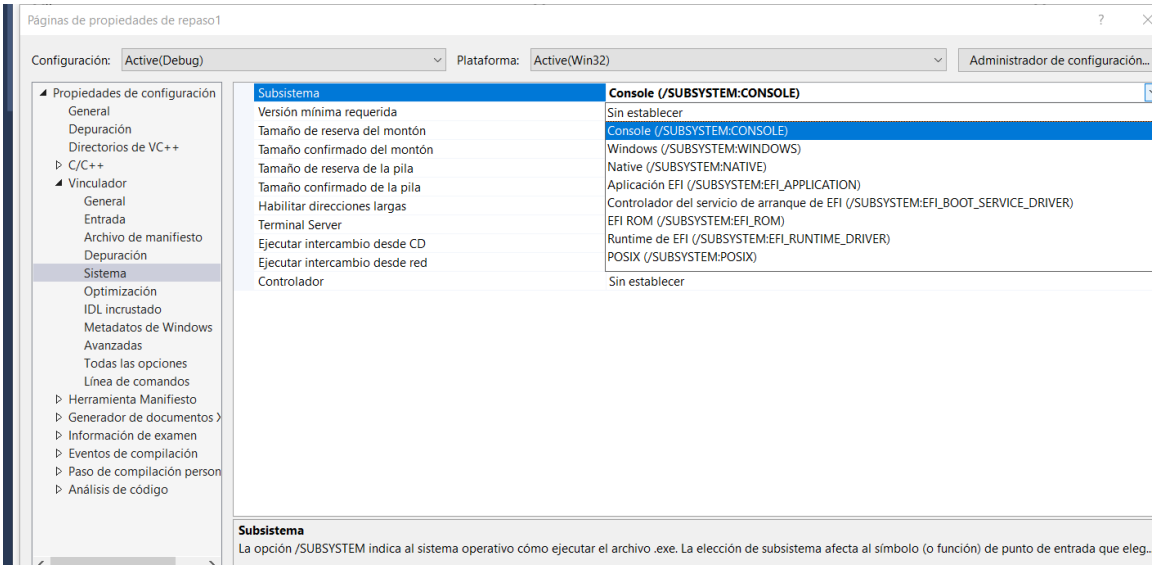


6. Para que al ejecutar sea visible la ventana de "`consola`" de Windows, hay que activar cierta propiedad del proyecto (en algunas versiones de VS viene activada por defecto). Para ello:

Elegir del menú: *`Proyecto → Propiedades:`*



Y a continuación activar la siguiente opción del vinculador (*linker*) . "*`Vinculador → Sistema → Subsistema → Console`*":



7. Si todo ha ido bien, debería poder compilarse y enlazarse ("*link*" o "vincular") el programa. Usar la opción del menú:
   `Compilar → Compilar solución`
   O la tecla:
   F7
   Y en la ventana de "*`Salida`*" (*Output*) no debería haber ningún error

RESULTADOS

Para medir los tiempos de ejecución de forma precisa se están usando unas rutinas (métodos) de la clase *QPTimer* (creadas por el dpto. de ATC, ficheros *QueryPerformanceTiming_rdtsc.h* , *QueryPerformanceTiming_rdtsc.cpp*) . No hace falta entenderlos; solamente los vamos a usar.

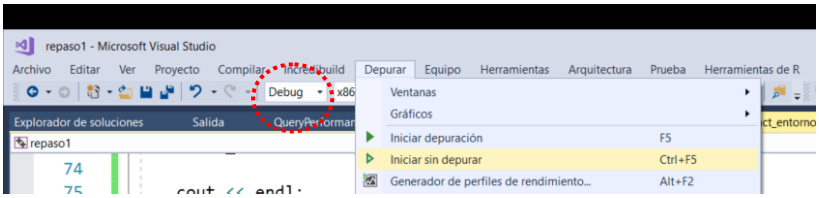
1. Primero vamos a analizar qué rutina queremos ejecutar y medir en esta práctica
   a. ¿Qué operación hacen *`pp_a()`* y *`pp_b()`* con las matrices?
   b. ¿Serán los mismos o diferentes los valores que almacenen en *`m3`*[][] las funciones *`pp_a()`* y *`pp_b()`*?

c. ¿Cuál es la diferencia entre las funciones *pp_a() y pp_b*()?

VS define un conjunto de configuraciones de compilación. El usuario puede crear otras con un "*Administrador de configuración*", o incluso cambiar las opciones de las configuraciones por defecto.
Normalmente existen dos configuraciones por defecto:

a) Configuración *Debug*. Como indica su nombre es para depuración. Produce un código lento, pero útil para ejecutar paso a paso, ya que compila sin ninguna optimización.
b) Configuración *Release*. Como indica su nombre es para entregar el código ya depurado. Produce un código rápido, pero, a veces, difícil para para ejecutar paso a paso y depurar; ya que compila con muchas optimizaciones.

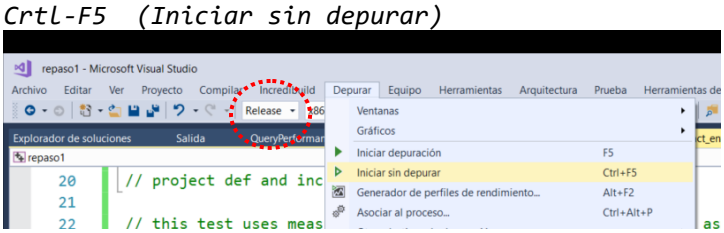
2. Ejecución y medición de tiempos en configuración **DEBUG**
   a. Comprobar que estamos usando la configuración *Debug* en barra del menú estándar. Ejecutar con la tecla: *Crtl-F5 (Iniciar sin depurar)*



   b. Anotar tiempos de ejecución en ciclos
   c. ¿Por qué los tiempos medios son bastante mayores que los tiempos mínimos (Si no lo fuera, preguntar al profesor)? Algo como:

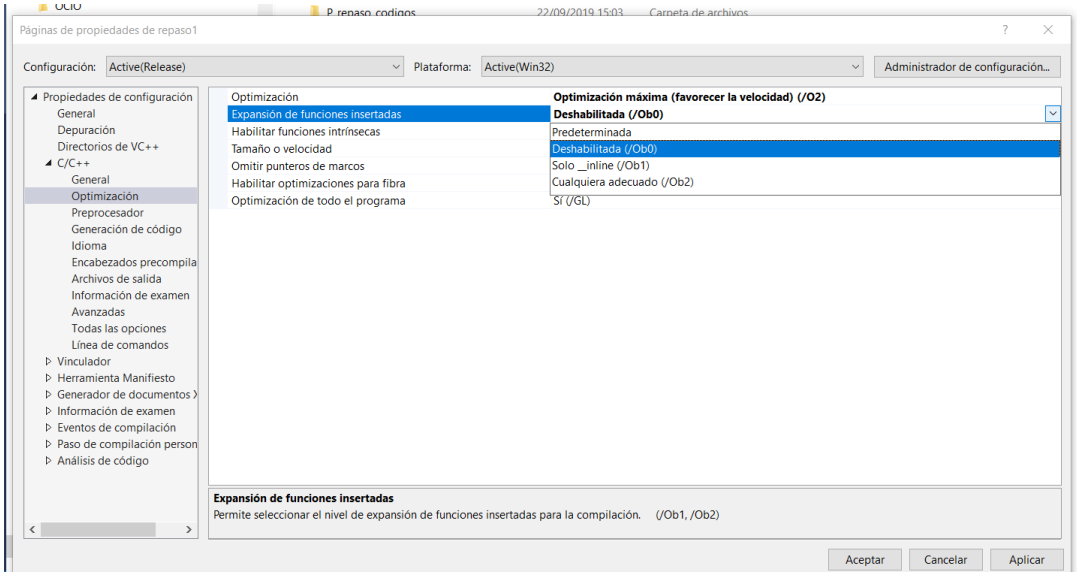
```
Tiempo minimo en ciclos para las repeticiones de pp_a()  es:   2.64845e+07
Tiempo medio  en ciclos de pp_a() es: 3.5107e+07
```

3. Ejecución y medición de tiempos en configuración **RELEASE**
   a. Cambiar a la configuración *Release* (ver captura).
   b. Ejecutar con la tecla:
   Crtl-F5 (Iniciar sin depurar)

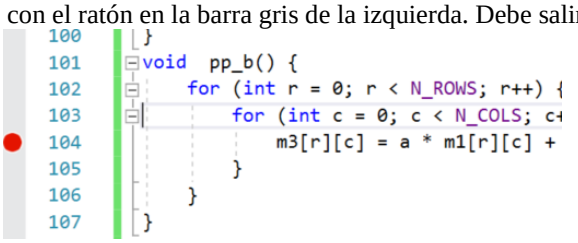


   c. Anotar tiempos de ejecución en ciclos

4. Comparar los tiempos de ejecución de ambas configuraciones (p ej hallar los cocientes entre los tiempos mínimos). Debería ser siempre más rápido la configuración *Release* que el *Debug*. Si no lo fuera, preguntar al profesor.
5. Comparar los tiempos de ejecución de ambas funciones *pp_a() , pp_b()*. ¿Por qué una de las funciones es bastante más rápida que la otra, si hacen lo mismo?
6. Para ver el ensamblador que genera VS, vamos a continuar en la configuración **Release**.
   a. Vamos a cambiar la siguiente opción de compilación (ver captura) para que el código de las funciones llamadas no se empotren o expandan dentro de la función llamante (es decir, *pp_b()* dentro del *main*()):
   c++ → optimización → expansión de funciones inline → deshabilitada

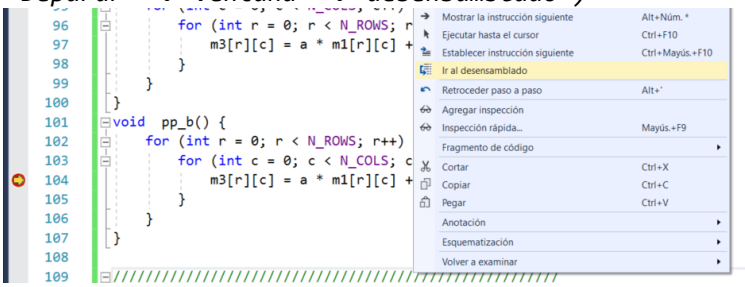


b. Poner un punto de parada de ejecución (*Breakpoint*) dentro de los bucles de la función *pp_b(),* pinchando con el ratón en la barra gris de la izquierda. Debe salir un círculo rojo; en la línea 104 en esta captura:



c. Ejecutar ahora con la tecla:
F5 (Iniciar depuración)
Debe parar la ejecución en tal sentencia del bucle.

d. Pinchando con el botón derecho sobre el código , buscar la opción "ver desensamblador" (también en el menú: *"Depurar → Ventana → desensamblado")*



e. Avanzar paso a paso la ejecución con la tecla F11. Escribir qué operaciones están en el bucle , y demostrar que son las correspondientes a : `m3[r][c] = a * m1[r][c] + m2[r][N_COLS - c - 1];`

NOTA: esta instrucción

```
movss       xmm0, dword ptr m1 (0985398h)[eax]
```

Significa:

Cargar (del inglés *move*) de la dirección `(0985398h + eax)` un valor float y almacenarlo en el registro `xmm0`

f. Calcular los Ciclos por una iteración de bucle interno de *pp_b()* , dividiendo el tiempo de *pp_b()* (en Release) por el número total de iteraciones *N_ROWS*N_COLS.* Como el bucle tiene al menos cuatro instrucciones por iteración, cada instrucción dura menos de 1 ciclo en ejecutarse (si no fuera así, algo se ha hecho mal). ¿Es posible esto?