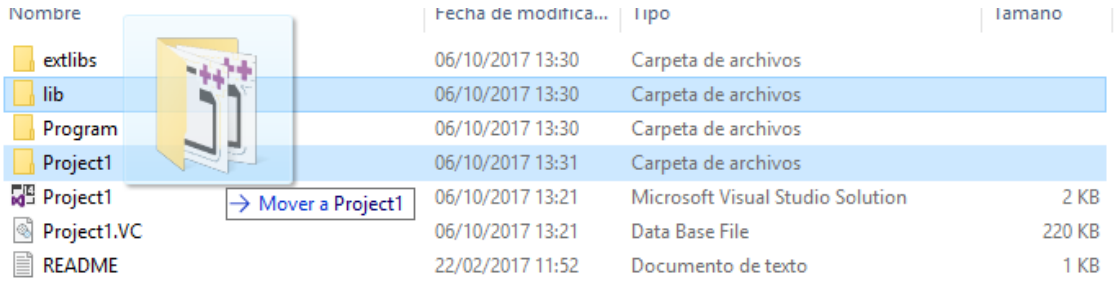
Pràctiques de Laboratori de Programació
Manual d'utilització de la llibreria gràfica

Creació d'un projecte que utilitzi la llibreria gràfica

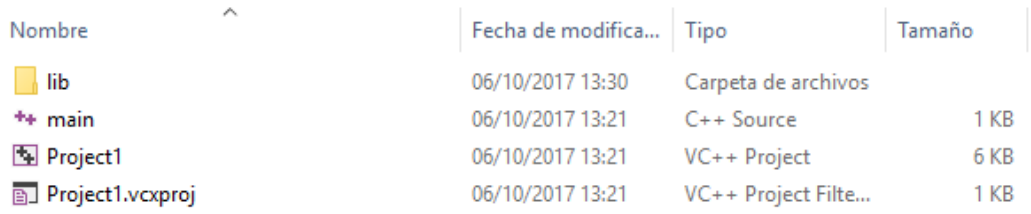
Per crear un projecte nou en Visual Studio que utilitzi la llibreria heu de seguir els passos següents:

1. Descomprimir tots els fitxers de "LlibreriaGrafica_VS.zip" a la carpeta arrel del projecte, on es troba la solució del projecte.

Un cop copiats, moure el directori lib dins del directori del projecte:

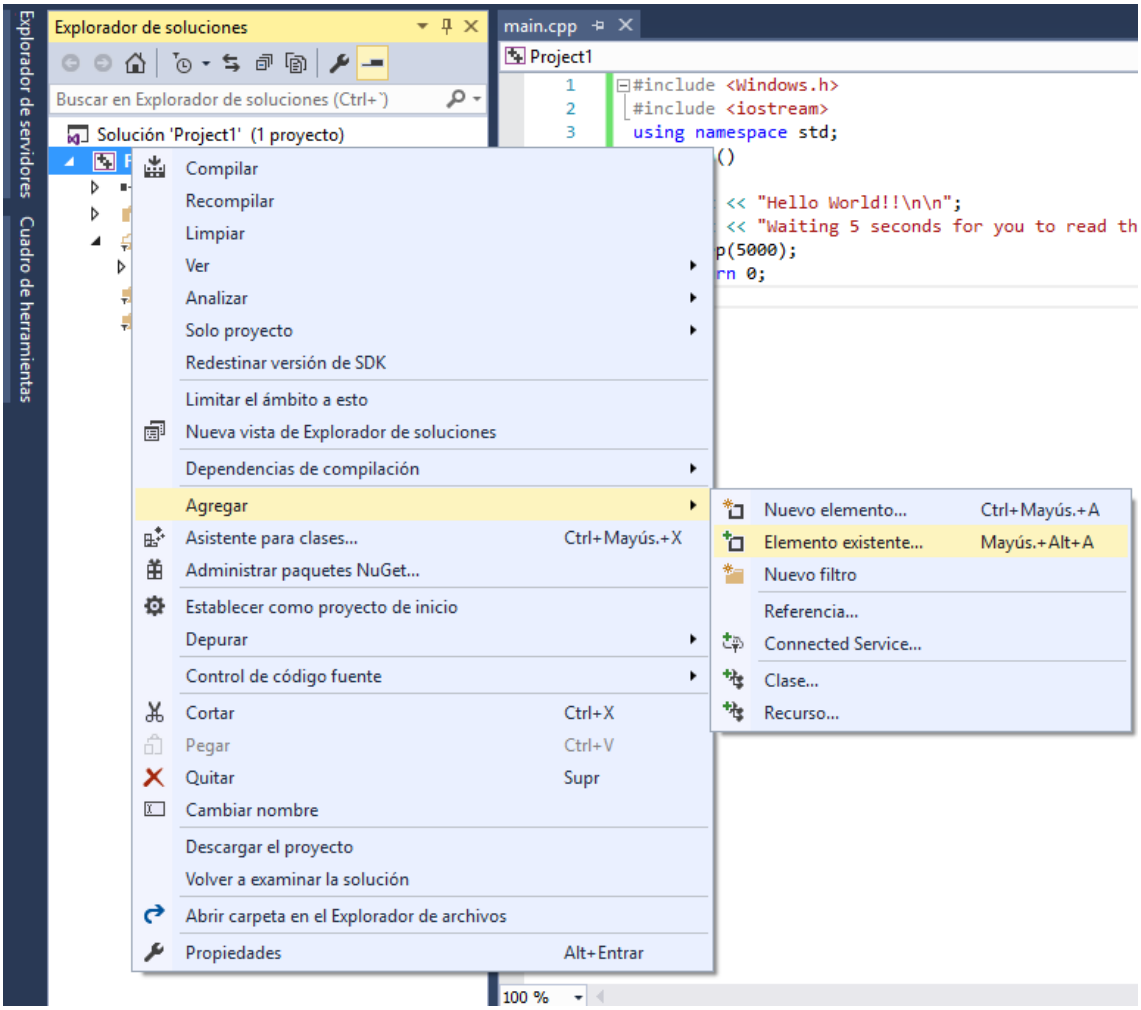


Directori del projecte:

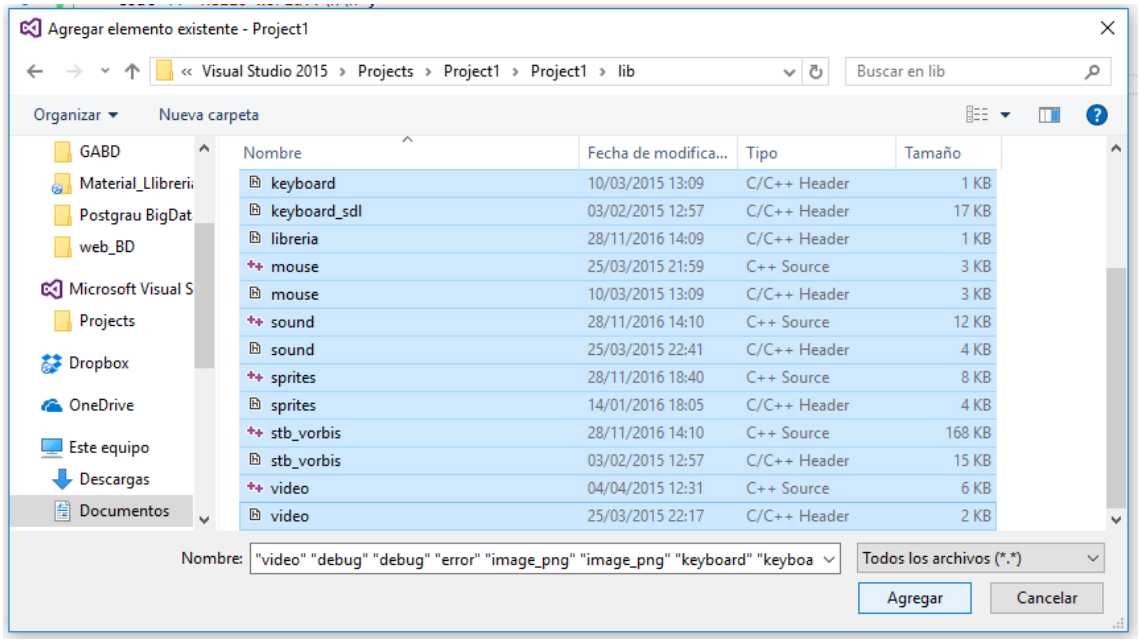


2. Afegir tots els fitxers font (.cpp i .h) del directori `\lib` al projecte de Visual Studio (opció del menú: *Proyecto - Agregar Elemento Existente*) i agregar-los al filtre lib:

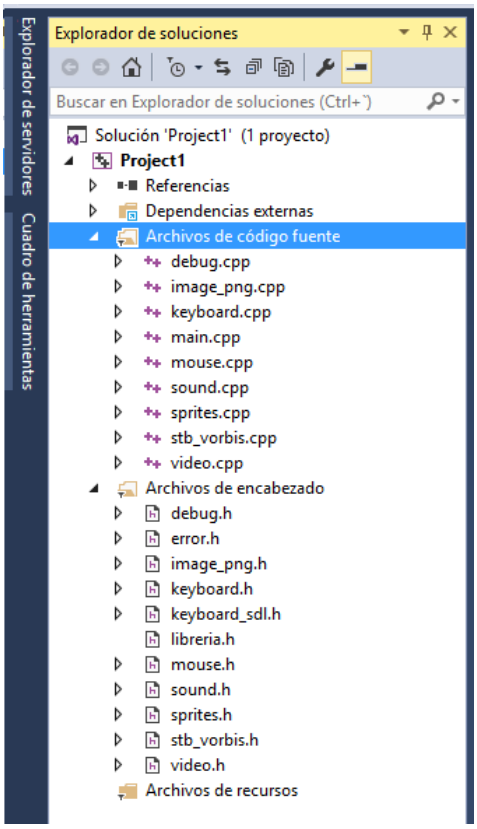
2.a


Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl+ˇ)
Solución 'Project1' (1 proyecto)
main.cpp
Project1
#include <Windows.h>
#include <iostream>
using namespace std;
<< "Hello World!!\n\n";
<< "Waiting 5 seconds for you to read th
p(5000);
rn 0;
Compilar
Recompilar
Limpiar
Ver
Analizar
Solo proyecto
Redestinar versión de SDK
Limitar el ámbito a esto
Nueva vista de Explorador de soluciones
Dependencias de compilación
Agregar
Asistente para clases... Ctrl+Mayús.+X
Administrar paquetes NuGet...
Establecer como proyecto de inicio
Depurar
Control de código fuente
Cortar Ctrl+X
Pegar Ctrl+V
Quitar Supr
Cambiar nombre
Descargar el proyecto
Volver a examinar la solución
Abrir carpeta en el Explorador de archivos
Propiedades Alt+Entrar
Nuevo elemento... Ctrl+Mayús.+A
Elemento existente... Mayús.+Alt+A
Nuevo filtro
Referencia...
Connected Service...
Clase...
Recurso...
100 %

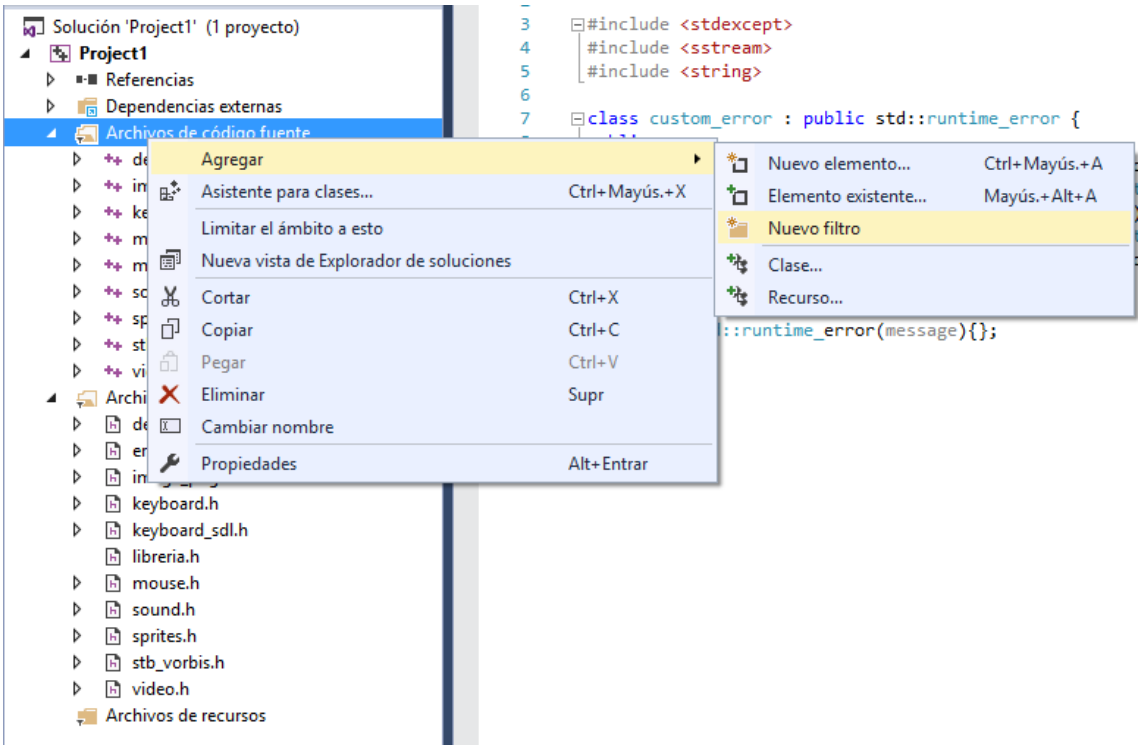
2.b


Agregar elemento existente - Project1
« Visual Studio 2015 › Projects › Project1 › Project1 › lib
Buscar en lib
Organizar
Nueva carpeta
GABD
Material_Llibreri
Postgrau BigDat
web_BD
Microsoft Visual S
Projects
Dropbox
OneDrive
Este equipo
Descargas
Documentos
Nombre
Fecha de modifica...
Tipo
Tamaño
keyboard 10/03/2015 13:09 C/C++ Header 1 KB
keyboard_sdl 03/02/2015 12:57 C/C++ Header 17 KB
libreria 28/11/2016 14:09 C/C++ Header 1 KB
mouse 25/03/2015 21:59 C++ Source 3 KB
mouse 10/03/2015 13:09 C/C++ Header 3 KB
sound 28/11/2016 14:10 C++ Source 12 KB
sound 25/03/2015 22:41 C/C++ Header 4 KB
sprites 28/11/2016 18:40 C++ Source 8 KB
sprites 14/01/2016 18:05 C/C++ Header 4 KB
stb_vorbis 28/11/2016 14:10 C++ Source 168 KB
stb_vorbis 03/02/2015 12:57 C/C++ Header 15 KB
video 04/04/2015 12:31 C++ Source 6 KB
video 25/03/2015 22:17 C/C++ Header 2 KB
Nombre: "video" "debug" "debug" "error" "image_png" "image_png" "keyboard" "keyboa
Todos los archivos (*.*)
Agregar
Cancelar

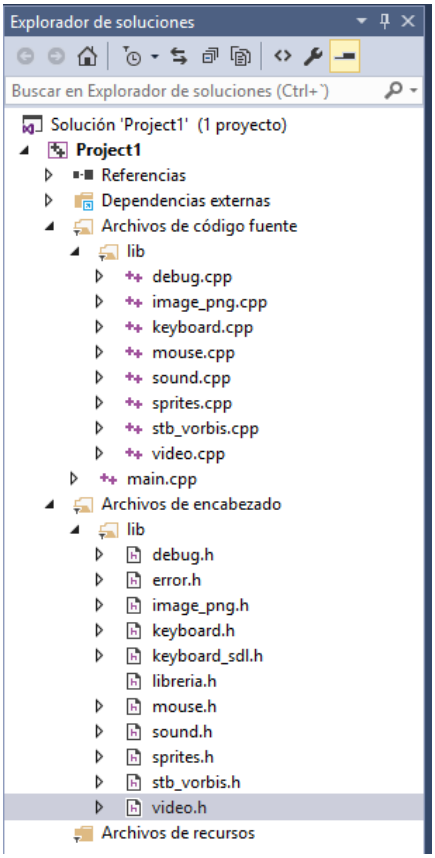
2.c



2.d Agregar filtres \lib tant al codi font com a les capçaleres i afegir tots els arxius de la llibreria gràfica:

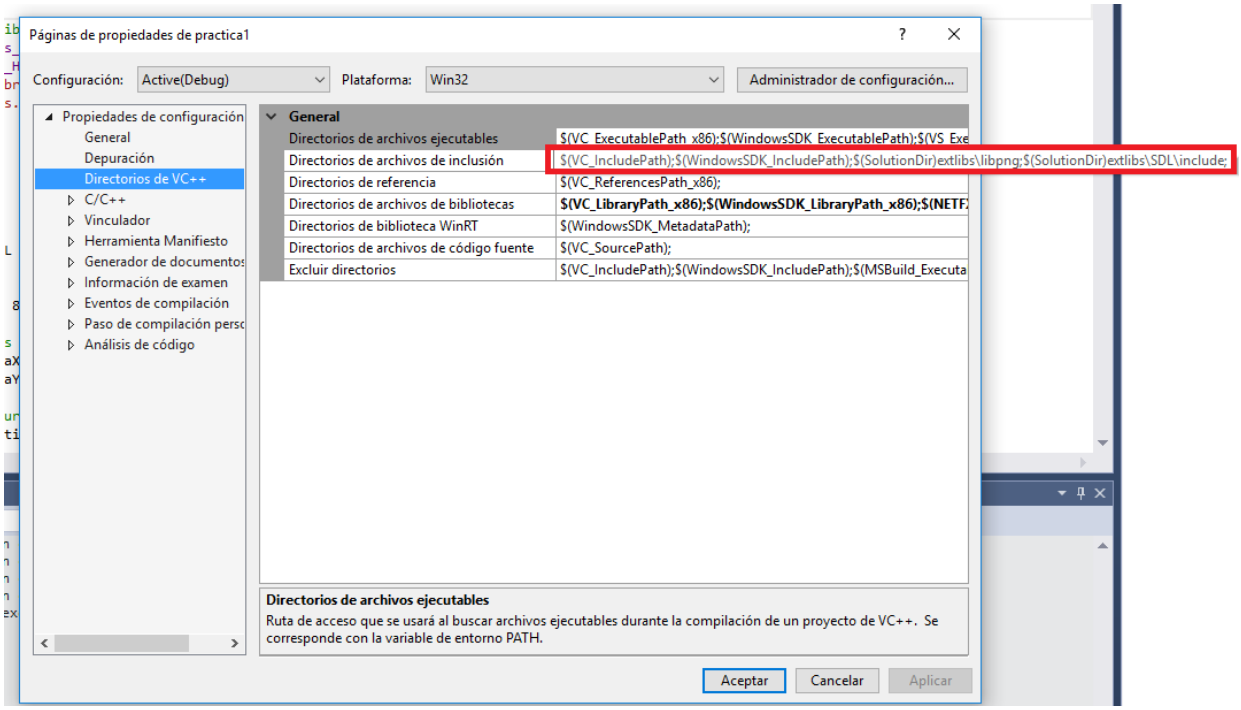


2.e Configuració final:



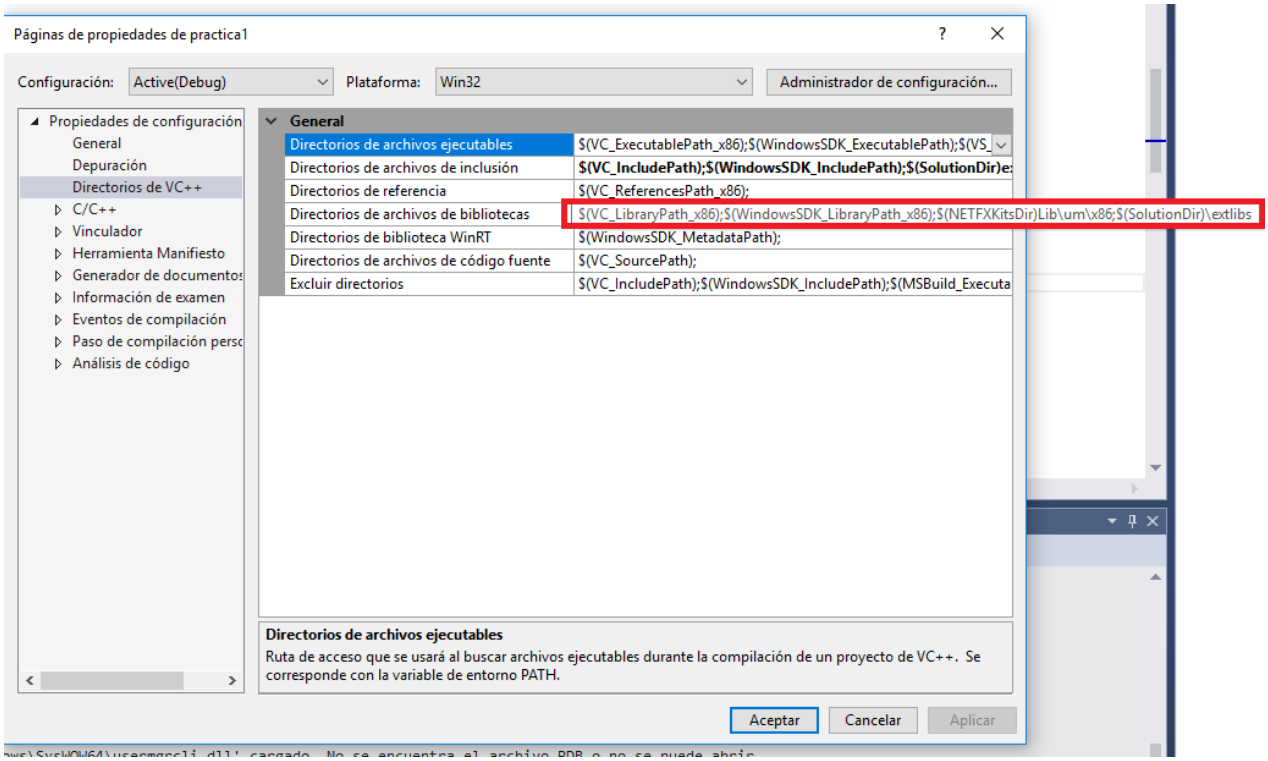
3. Afegir als directoris "`include`" del projecte (*Propiedades - Directorios VC++ - Directorios de archivos de inclusión*) els següents:

```
$(SolutionDir)\extlibs\SDL\include
$(SolutionDir)\extlibs\libpng
```



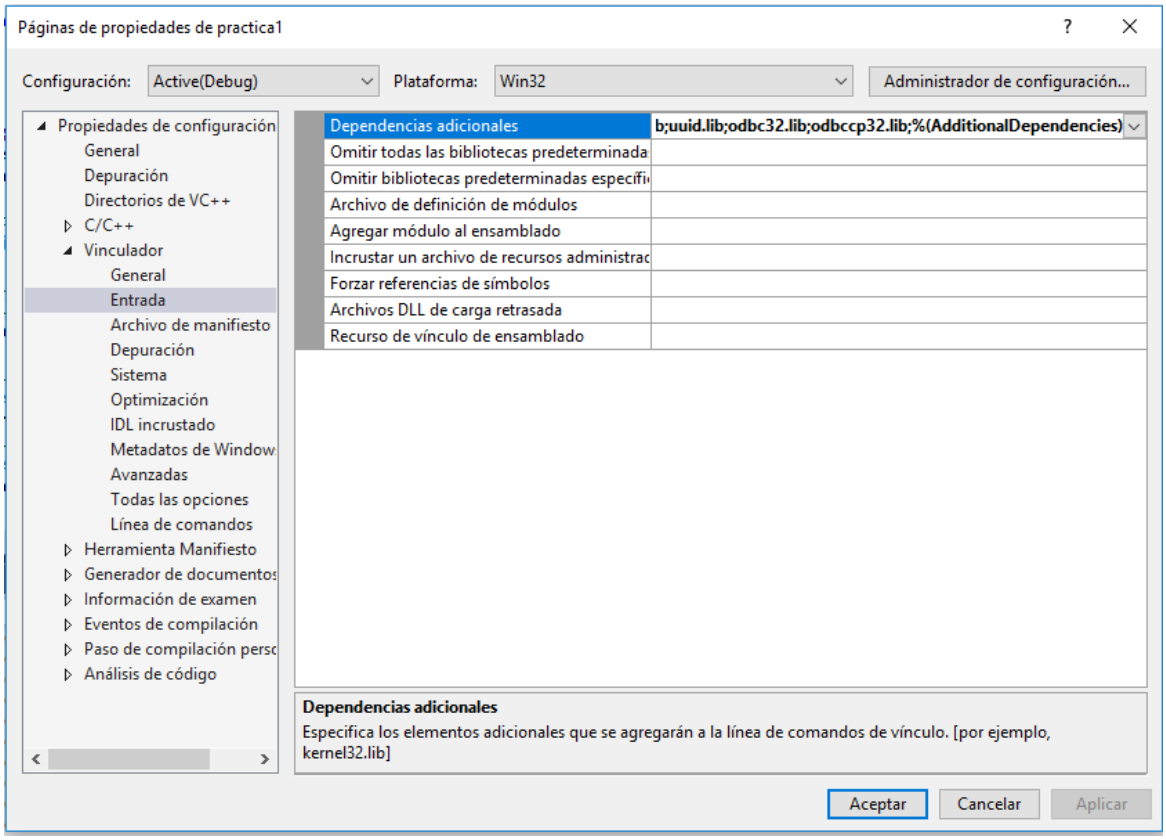
4. Afegir als directoris de llibreries (*Propiedades - Directorios VC++ - Directorios de archivos de bibliotecas*) del projecte el següent:

```
$(SolutionDir)\extlibs
```



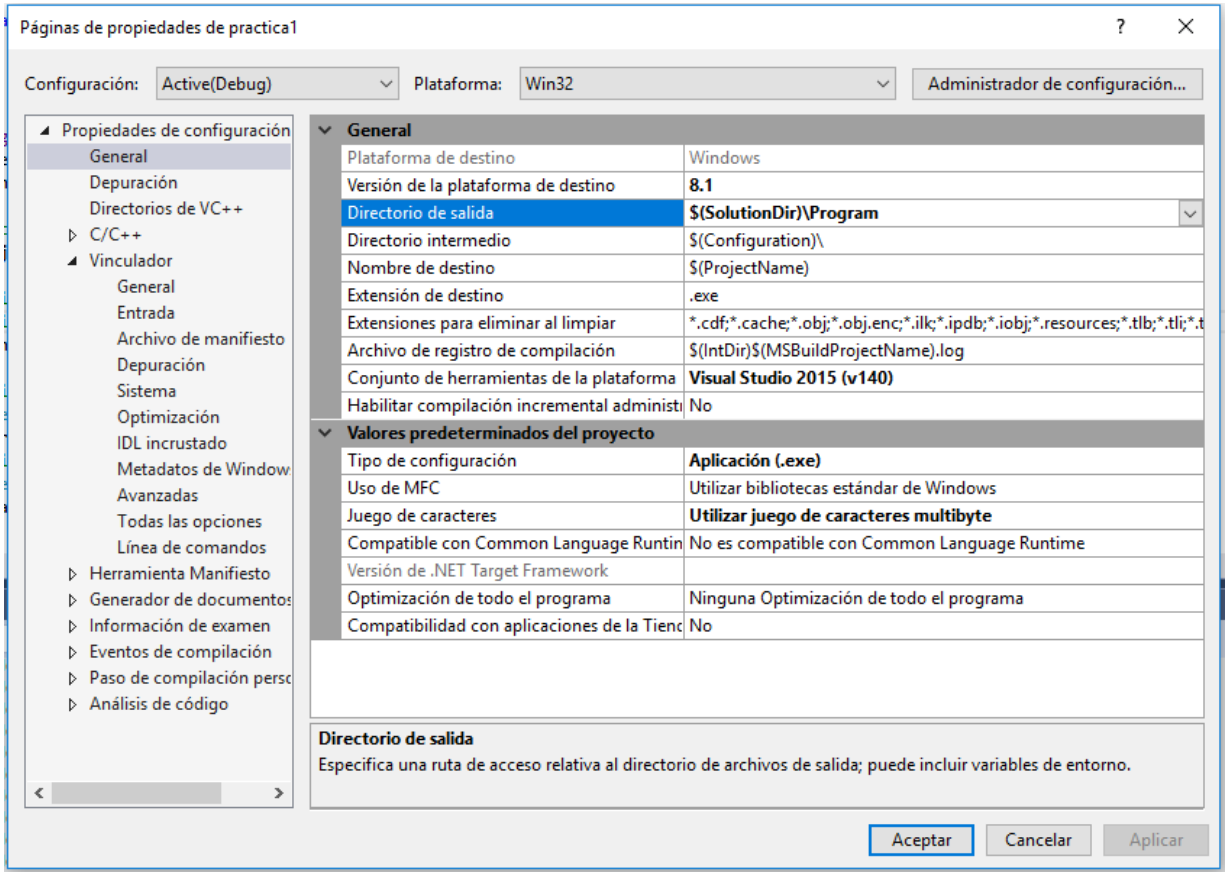
5. Afegir com a dependències addicionals (*Propiedades - vinculador - entrada - dependencias adicionales*) els següents fitxers:

```
libpng16.lib
SDL2.lib
SDL2_image.lib
```



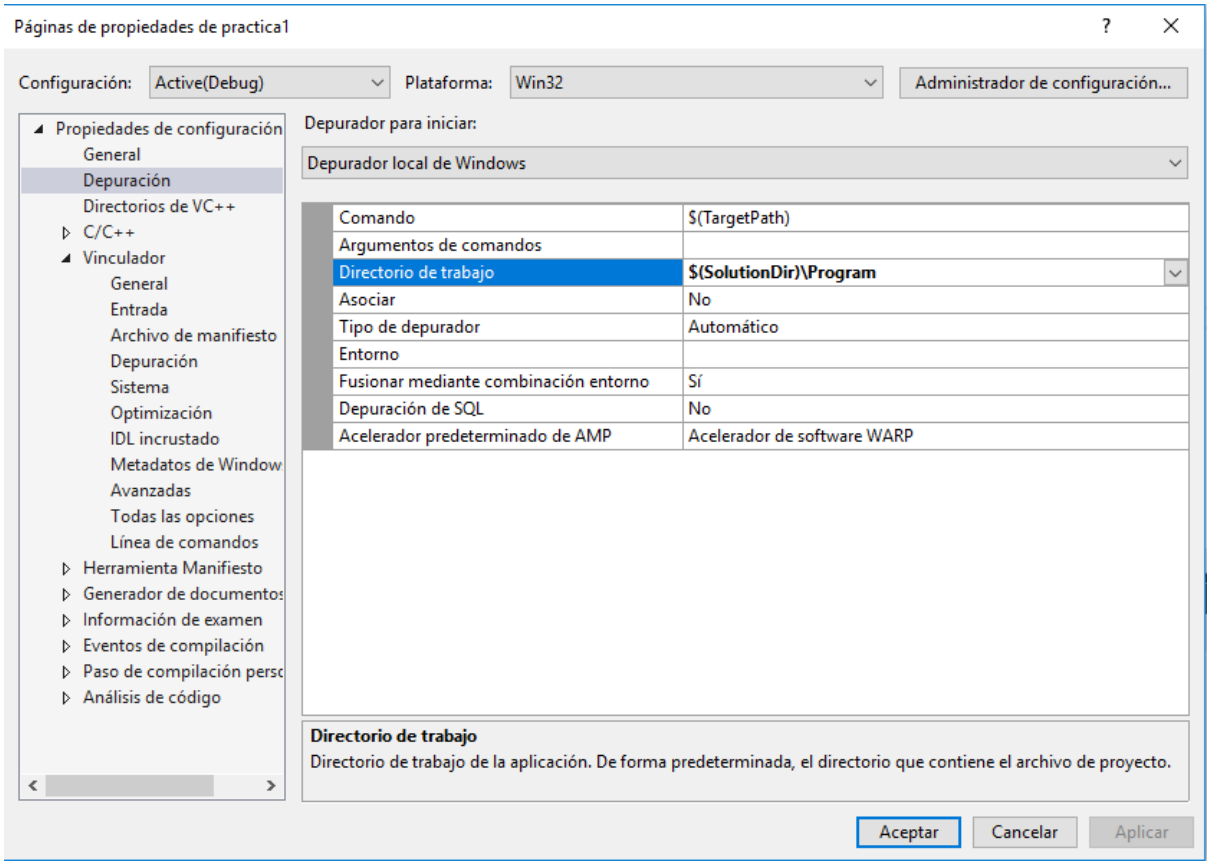
6. A *Propiedades - general*

```
Directorio de salida = $(SolutionDir)\Program
```

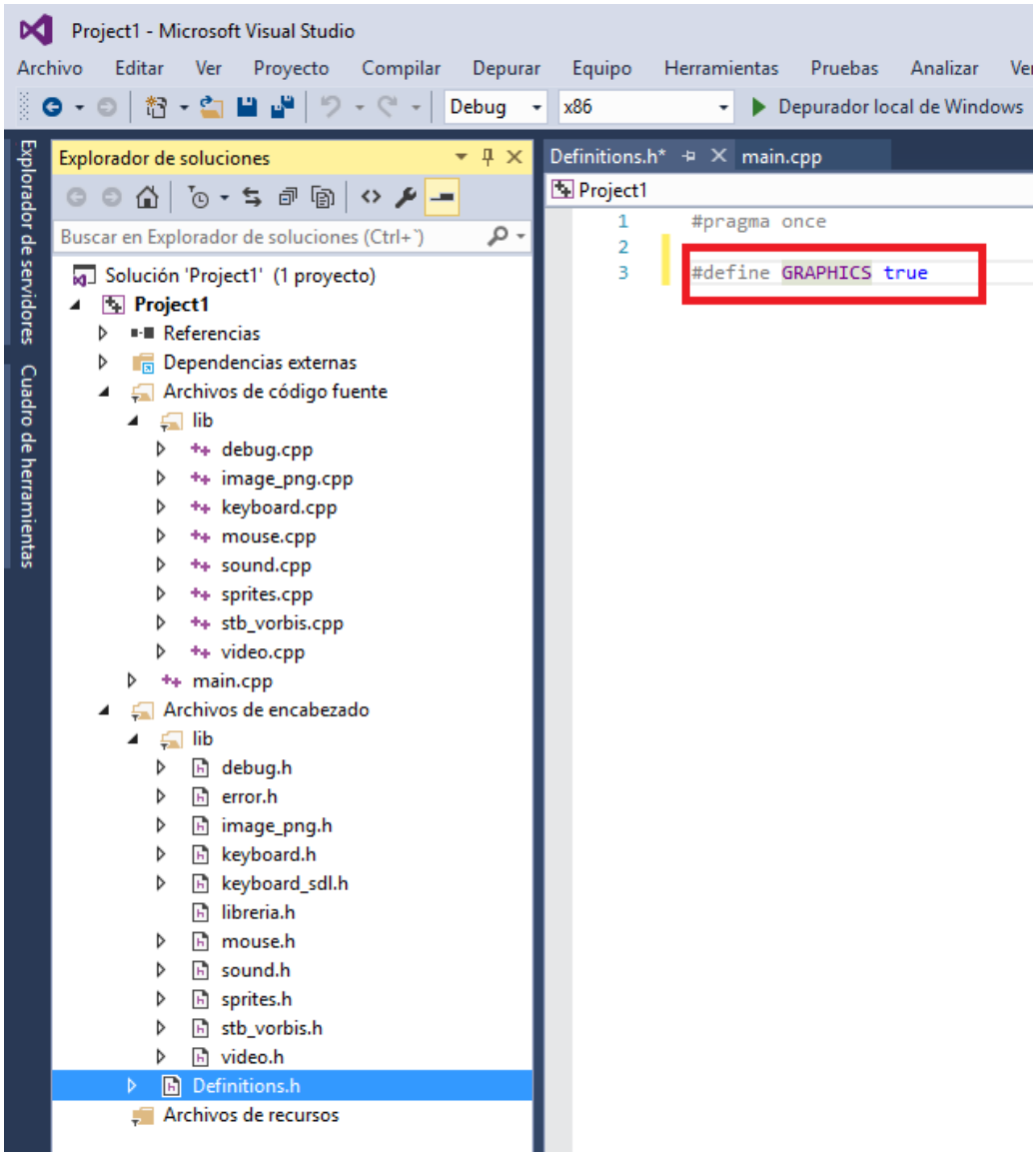


7. A *Propiedades - Depuración*

```
Directorio de trabajo = $(SolutionDir)\Program
```



8. Per activar i desactivar els gràfics, utilitzarem una variable definida en una capçalera que anomenarem Definitions.h:



9. Exemple de codi en el que apareix una imatge (mao.png) dibuixada a la pantalla durant 5 segons. Per tal que no falli en ser executat aquest codi, heu d'incloure el fitxer mao.png (o un altre png i canviar el nom al codi) dins la carpeta Program.

```
//Definicions amb la variable d'activació de la llibreria gràfica, GRAPHICS
#include "Definitions.h"

//Includes de la llibreria gràfica
#ifdef GRAPHICS
	#define SDL_MAIN_HANDLED
	#include "lib\libreria.h"
	#include <windows.h>
#endif

//Mida del taulell
const int MIDA_TOTAL = 480;

int main(int argc, char* argv[])
{
#ifdef GRAPHICS
	// Declaració de l'objecte de la classe Screen amb una mida
determinada
	Screen pantalla = Screen(MIDA_TOTAL, MIDA_TOTAL);

	// Inicialització de la variable mao amb el gràfic mao.png
	//imatge guardada al directori de treball del projecte
	Sprite mao = Sprite("mao.png");

	//Mostra la pantalla gràfica
	pantalla.show();

	//Dibuixa el mao dalt a l'esquerra de la pantalla
	mao.draw(0, 0); //(columna/fila)

	//Actualització de la pantalla
	pantalla.update();

	//Espera 5s
	Sleep(5000);

	//Destrueix la pantalla tancant-la degudament
	pantalla.~Screen();
#endif

	return 0;
}
```

Explicació de les funcions de la llibreria gràfica

Per poder utilitzar qualsevol de les funcions de la llibreria s'ha d'incloure el fitxer de capçalera "`libreria.h`". Tingueu en compte d'especificar correctament el *path* relatiu dins del projecte on es troba aquest fitxer. Si heu seguit bé els passos anteriors, aquest fitxer estarà dins d'un directori `\lib` i per tant, la inclusió l'haureu de fer així:

```
#include "lib/libreria.h"
```

Visualització de gràfics per pantalla

La creació i visualització dels gràfics per pantalla es fa utilitzant la **classe** Sprite de la llibreria (lib\sprites.h). Els mètodes de la classe `Sprite` que haureu de fer servir són els següents:

- `Sprite(const char* ruta)`: aquest constructor s'encarrega d'inicialitzar un gràfic a partir del fitxer especificat en el paràmetre `ruta`. El fitxer ha de ser un fitxer gràfic en format *.png*. Tingueu en compte que la ruta ha d'incloure tot el *path* relatiu a partir del directori `Program` del projecte.

  La inicialització del gràfic implica que es carrega la informació del gràfic des del fitxer PNG i es guarda en una estructura en memòria. És **crear** el gràfic amb aquest constructor abans d'intentar dibuixar-lo per pantalla. La creació del gràfic s'ha de fer **un únic cop**. Un cop creat, un mateix gràfic es pot dibuixar tantes vegades com vulguem en diferents posicions de la pantalla.

- `void draw(int x, int y)`: aquest mètode dibuixa el gràfic a les coordenades `x,y` de pantalla que es passen com a paràmetre. La coordenada `(0,0)` correspon a la cantonada superior esquerra de la pantalla. El punt de referència del gràfic és també la coordenada superior esquerra del gràfic.

- `int getScaleX()`: recupera la mida en x (amplada) del gràfic.

- `int getScaleY()`: recupera la mida en y (alçada) del gràfic.

- `void setScale(float scale)`: modifica la mida del gràfic, multiplicant el tamany original pel factor d'escala que es passa com a paràmetre.

Fixeu-vos que no tenim cap mètode per esborrar un gràfic per pantalla. Tots els gràfics s'esborren automàticament a cada cicle de refresc de la pantalla i s'han de tornar a dibuixar.

Tampoc tenim cap mètode per destruir o alliberar un gràfic. La destrucció del gràfic es fa automàticament quan s'acaba el programa o la funció on s'ha declarat cridant al destructor per defecte de la classe `Sprite`.

Exemple d'utilització de la classe `Sprite`

```
// Declaració de la variable de la classe Sprite per guardar un gràfic
Sprite bloc;

// Inicialització de la variable "bloc" amb el gràfic "mauBlau.png" que es
// troba al subdirectori data de Program
bloc = Sprite("data/maoBlau.png");

...

// Dibuixa el gràfic a pantalla cinc cops, en cinc posicions diferents
// separades en x per l'amplada del gràfic
for (int i = 0; i < 5; i++)
      bloc.draw(i * bloc.getScaleX(), 0);
```

Funcions de control de la llibreria

A la llibreria hi ha la classe `Screen` que permet controlar l'estat de la pantalla i de la finestra gràfica i capturar els events de teclat que es produeixen. La part pública d'aquesta classe és la següent:

```
class Screen {
public:
  Screen(unsigned int resolucioX, unsigned int resolucioY);
  //Default: resolucioX = 600, resolucioY = 550
  Screen ();
  ~Screen ();

  void show();
  void processEvents();
  void update();
  void init();
  bool isExit() const;
...

};
```

Els mètodes que es poden utilitzar són els següents:

- `Screen(unsigned int resolucioX, unsigned int resolucioY)`: Constructor que permet definir el tamany de la pantalla en píxels.

- `void show()`: fa que aparegui la finestra gràfica on es dibuixaran els gràfics. S'ha de **cridar** també **obligatòriament** un sol cop al **principi** del programa abans de dibuixar cap objecte a pantalla.

- `void processEvents()`: captura els events de teclat i ratolí que s'han produït des de l'anterior crida a aquest mètode. S'ha de cridar a cada cicle del bucle principal del programa, al principi del cicle. A la següent secció expliquem com detectar quines tecles s'han pressionat.

- `void update()`: s'encarrega de fer el refresc de pantalla que cal fer a cada cicle: esborra tota la pantalla i dibuixa tots els gràfics que s'hagin dibuixant (cridant al mètode `Draw` de la classe `Sprite`) des de l'última crida al mètode. S'ha de cridar a cada cicle del bucle principal del programa, al final del cicle.

- `bool isExit()`: retorna `true` si s'ha donat ordre de tancar la finestra gràfica clicant amb el ratolí al botó de tancar finestra.

Qualsevol programa que faci servir la llibreria gràfica, haurà d'incloure un objecte de la classe `Screen`, i haurà de cridar als mètodes de la classe de la forma que es mostra a l'exemple que trobareu a continuació.

Exemple d'utilització de la classe `Screen`

```
// Declaració de l'objecte de la clase Scren per controlar la part gràfica.
// Per defecte el tamany de la pantalla és 600 x 550
Screen pantalla;

// Declaració de l'objecte de la classe Screen canviant el tamany per defecte
// de la pantalla. En aquest cas, el tamany de la pantalla es MIDA_X x MIDA_Y
pantalla = Screen(MIDAX, MIDAY);

// Fa que aparegui la pantalla gràfica
pantalla.show();

// Bucle de control
do
{
	// Captura tots els events de teclat que s'han produit en el darrer cicle
	// des de l'última crida al mètode
	pantalla.processEvents();

	...

	// Dibuixa els gràfics (sprites) que estan actius en aquest cicle
	fons.draw();
	mur.draw();

	...

	// Fa el refresc de pantalla. Esborra tota la pantalla i redibuixa
	// tots els gràfics que s'hagin fet dibuixar en un cicle del bucle,
	// en aquest cas fons i mur
	pantalla.update();

} while (<no hem acabat>);
```

Captura de teclat

El mètode `processEvents()` de la classe `Screen` permet detectar totes les tecles que s'han pressionat des de l'últim cop que s'ha executat aquest mètode. Això els "events" del ratolí. Es pot detectar més d'una tecla, especialment si es pressionen a la vegada.

Després d'aquesta crida, per saber si s'ha pressionat una tecla determinada s'ha de cridar a la funció:

```
Keyboard_GetKeyTrg(CODI_TECLA)
```

que retorna cert si s'ha pressionat la tecla especificada. Els codis de les tecles estan definits al fitxer "`keyboard_sdl.h`" que trobareu dins del directori "`\lib`".

Exemple d'utilització de les funcions de captura de teclat

```
Screen pantalla;

pantalla.show();

// Bucle de control
do
{
	// Captura tots els events de teclat que s'han produit en el darrer cicle
	// des de l'última crida al mètode
	pantalla.processEvents();

	...

	// Comprovem si s'ha pressionat la tecla de la fletxa esquerra
	if (Keyboard_GetKeyTrg(KEYBOARD_LEFT))
		<accions a fer si es pressiona la fletxa esquerra>
	// Comprovem si s'ha pressionat la tecla de la fletxa dreta
	if (Keyboard_GetKeyTrg(KEYBOARD_RIGHT))
		<accions a fer si es pressiona la fletxa dreta>
	// Comprovem si s'ha pressionat la tecla de l'espai
	if (Keyboard_GetKeyTrg(KEYBOARD_SPACE))
		<accions a fer si es pressiona la tecla de l'espai>

	...

} while (<no final>);
```