**1) Qu’est-ce que la SDL 2 ? (10 points)**

=> La SDL2 est une librairie de développement multi-plateformes écrite en C.

**2) Que peut faire la SDL 2 ? (10 points)**

=> La bibliothèque met à disposition un ensemble de fonctions permettant à l'utilisateur de gérer de manière très basique l'audio, les input (clavier/souris) et afficher des images.

**3) Sur quelles plateformes fonctionne la SDL 2 ? (10 points)**

=> La SDL2 étant une librairie multi-plateformes, elle est supportée sur Windows, Mac OS X, Linux, iOS et Android.

**4) Comment obtenir et installer la SDL 2 sur un EDI ? Donnez les étapes de la configuration d’un projet SDL2 avec CodeBlocks (gcc). (10 points)**

=> Il suffit de télécharger la librairie sur le site et de lier la bibliothèque à son compilateur.   
Pour CodeBlocks, il faut aller dans "Project" => "Build options" => "Debug" => "Linker settings", cliquer sur "Add" et chercher le dossier i686-w64-mingw23/lib et sélectionner le fichier libSDL2.dll.a  
Il faut ajouter également le fichier libSDL2main.a.  
Il faut également rajouter « -lmingw32 » dans la case « Other linker options ».  
Enfin, il faut également aller dans « Search directories », et dans « Compiler » inclure le dossier i686-w64-mingw23/include, et dans l’onglet « Linker » il faut inclure le dossier i686-w64-mingw23/lib.  
Pour finir, il faut mettre le fichier SDL2.dll à la racine du dossier où est contenu le main.c

**5) Quelles sont les différences entre SDL 1 et SDL 2 ? (5 points)**

Une des plus grosses fonctionnalités de la SDL 2 est le support de la 3D, elle est plus performante que la SDL 1, la licence de distribution est zlib et non plus LGPL, permettant une utilisation gratuite et libre de la bibliothèque.

**6) Pour utiliser la SDL 2 dans votre projet, vous devez initialiser l’instance SDL 2, quelles sont les étapes d’initialisation à réaliser pour utiliser la lib SDL 2, donnez le code correspondant et expliquez précisément chaque instruction. (15 points)**

Pour initialiser la SDL 2, il faut utiliser la fonction SDL\_Init() qui peut prendre divers paramètres, et va initialiser le sous-système correspondant.

|  |  |
| --- | --- |
| **SDL\_INIT\_TIMER** | timer subsystem |
| **SDL\_INIT\_AUDIO** | audio subsystem |
| **SDL\_INIT\_VIDEO** | video subsystem; automatically initializes the events subsystem |
| **SDL\_INIT\_JOYSTICK** | joystick subsystem; automatically initializes the events subsystem |
| **SDL\_INIT\_HAPTIC** | haptic (force feedback) subsystem |
| **SDL\_INIT\_GAMECONTROLLER** | controller subsystem; automatically initializes the joystick subsystem |
| **SDL\_INIT\_EVENTS** | events subsystem |
| **SDL\_INIT\_EVERYTHING** | all of the above subsystems |
| **SDL\_INIT\_NOPARACHUTE** | compatibility; this flag is ignored |

Ci-dessus les flags disponibles selon le wiki officiel de SDL.   
  
  
Dans notre cas on utilise SDL\_INIT\_EVERYTHING, qui va initialiser tous les sous-systèmes directement.  
La fonction retourne 0 en cas de succès, ou bien une valeur négative en cas d’erreur.  
Du coup pour vérifier que l’initialisation s’est bien passée, on va donc faire :  


**7) Comment créer une fenêtre avec SDL 2, donnez et expliquez le code correspondant à la création d’une fenêtre avec SDL 2. (10 points)**

Pour créer une fenêtre en SDL 2, il faut d’abord commencer par créer un pointeur de type SDL\_Window possédant la référence d’une window(Win32) ainsi :



Ensuite, il faut utiliser la fonction SDL\_CreateWindow() qui prend ces paramètres suivants :

const char\* title,

int x,

int y,

int w,

int h,

Uint32 flags)

Title = le titre de la fenêtre qui va être créer  
x = position x de la fenêtre  
y = position y de la fenêtre  
w = largeur de la fenêtre  
h = hauteur de la fenêtre  
  
Les flags peuvent prendre la valeur suivante :

|  |  |
| --- | --- |
| SDL\_WINDOW\_FULLSCREEN | fullscreen window |
| SDL\_WINDOW\_FULLSCREEN\_DESKTOP | fullscreen window at the current desktop resolution |
| SDL\_WINDOW\_OPENGL | window usable with OpenGL context |
| SDL\_WINDOW\_VULKAN | window usable with a Vulkan instance |
| SDL\_WINDOW\_HIDDEN | window is not visible |
| SDL\_WINDOW\_BORDERLESS | no window decoration |
| SDL\_WINDOW\_RESIZABLE | window can be resized |
| SDL\_WINDOW\_MINIMIZED | window is minimized |
| SDL\_WINDOW\_MAXIMIZED | window is maximized |
| SDL\_WINDOW\_INPUT\_GRABBED | window has grabbed input focus |
| SDL\_WINDOW\_ALLOW\_HIGHDPI | window should be created in high-DPI mode if supported (>= SDL 2.0.1) |

On a ainsi :   


**8) Gestion d’une fenêtre SDL 2, donnez les primitives associées à la fenêtre SDL 2. (10 points)**

**Créer une fenêtre (déjà vu)** :

SDL\_CreateWindow(**const** char\* title, int x, int y, int w, int h, Uint32 flags);

**Détruire une fenêtre :**

void SDL\_DestroyWindow(SDL\_Window\* window);

On passe en argument la fenêtre à détruire.



**Donner un titre à la fenêtre :**

void SDL\_SetWindowTitle(SDL\_Window\* window , **const** char\* title);

****

**Récupérer le titre de la fenêtre :**

**const** char\* SDL\_GetWindowTitle(SDL\_Window\* window);

****

**Changer la position de la fenêtre :**

void SDL\_SetWindowPosition(SDL\_Window\* window , int x, int y);

****

**Changer la taille de la fenêtre :**

|  |
| --- |
| void SDL\_SetWindowSize(SDL\_Window\* window , int w , int h); |
|  |

****

Il existe pour ces deux fonctions l’équivalent en Get, on peut également agrandir, réduire, restaurer, passer en mode plein écran ou en mode normal, définir la luminosité de la fenêtre ( ou la récupérer ), ou pareil avec l’ID de la fenêtre.

**9) Gérer un rendu : qu’est-ce qu’un renderer, comment est-il associé à la fenêtre SDL 2 ? Donnez et expliquez le code permettant de gérer un rendu, création, affichage, effacement…. (15 points)**

Un renderer est un contexte d’affichage : c’est ce qui permet de générer l’image dans la fenêtre SDL2. Sans renderer, rien ne peut être affiché. A une fenêtre est associé un contexte d’affichage, il ne peut exister deux renderers pour une fenêtre.

**Création d’un renderer :**

SDL\_Renderer\* SDL\_CreateRenderer(SDL\_Window\* window,

int index,

Uint32 flags)

Cette fonction prend en paramètre le fenêtre à laquelle on va associer le renderer, un index (qui correspond au driver de rendu à initialiser mais prendra la valeur -1 pour initialiser le premier), et les flags suivants :

|  |  |
| --- | --- |
| SDL\_RENDERER\_SOFTWARE | the renderer is a software fallback |
| SDL\_RENDERER\_ACCELERATED | the renderer uses hardware acceleration |
| SDL\_RENDERER\_PRESENTVSYNC | present is synchronized with the refresh rate |
| SDL\_RENDERER\_TARGETTEXTURE | the renderer supports rendering to texture |

De la même manière qu’une fenêtre, il faut d’abord déclarer une variable (pRenderer) qui est un pointeur de type SDL\_Renderer

  
On associe ensuite ce Renderer à la fenêtre précédemment crée, grâce à la fonction ci-dessus  


Il existe ensuite différentes fonctions permettant de dessiner des rectangles, points, etc…

**Changer la couleur d’affichage :**

int SDL\_SetRenderDrawColor(SDL\_Renderer\* renderer,

Uint8 r,

Uint8 g,

Uint8 b,

Uint8 a)

La fonction suivante permet de définir la couleur d’affichage du renderer, ce qui est utilisé par la fonction SDL\_RenderClear ) (qui efface le contenu du renderer en utilisant la couleur d’affichage), elle prend en paramètre les valeurs r,g,b (0 à 255) et a pour l’opacité (0 à 255).

  
SDL\_ALPHA\_OPAQUE étant un flag, et est l’équivalent de 255.

**Affichage :**

Cette fonction permet d’actualiser le rendu contenu dans le renderer passé en paramètre.

void SDL\_RenderPresent(SDL\_Renderer\* renderer)

Cette fonction ne prend en paramètre que le renderer dans lequel elle doit actualiser le rendu.



**Effacer le contenu du renderer :**

int SDL\_RenderClear(SDL\_Renderer\* renderer)

****  
Rappel : la fonction utilise la couleur d’affichage définie : si on l’a changée, cela coloriera l’écran de la couleur définie.  
  
**Supprimer le renderer :**

void SDL\_DestroyRenderer(SDL\_Renderer\* renderer)

****

**10) SDL\_Point et SDL\_Rect : Donnez et expliquez le code de la déclaration d’un SDL\_Point et SDL\_Rect (5 points)**

SDL\_Point est une structure permettant de définir un point, elle comporte un entier x et un entier y pour ses positions.

|  |  |  |
| --- | --- | --- |
| int | **x** | the x coordinate of the point |
| int | **y** | the y coordinate of the point |

Pour déclarer un point, il suffit juste de faire :



Concernant le SDL\_Rect, c’est une structure permettant de définir un rectangle, qui comporte un entier x et un entier y correspondant au coin en haut à gauche du rectangle, un entier w pour la longueur et un entier h pour la largeur.

|  |  |  |
| --- | --- | --- |
| int | **x** | the x location of the rectangle's upper left corner |
| int | **y** | the y location of the rectangle's upper left corner |
| int | **w** | the width of the rectangle |
| int | **h** | the height of the rectangle |

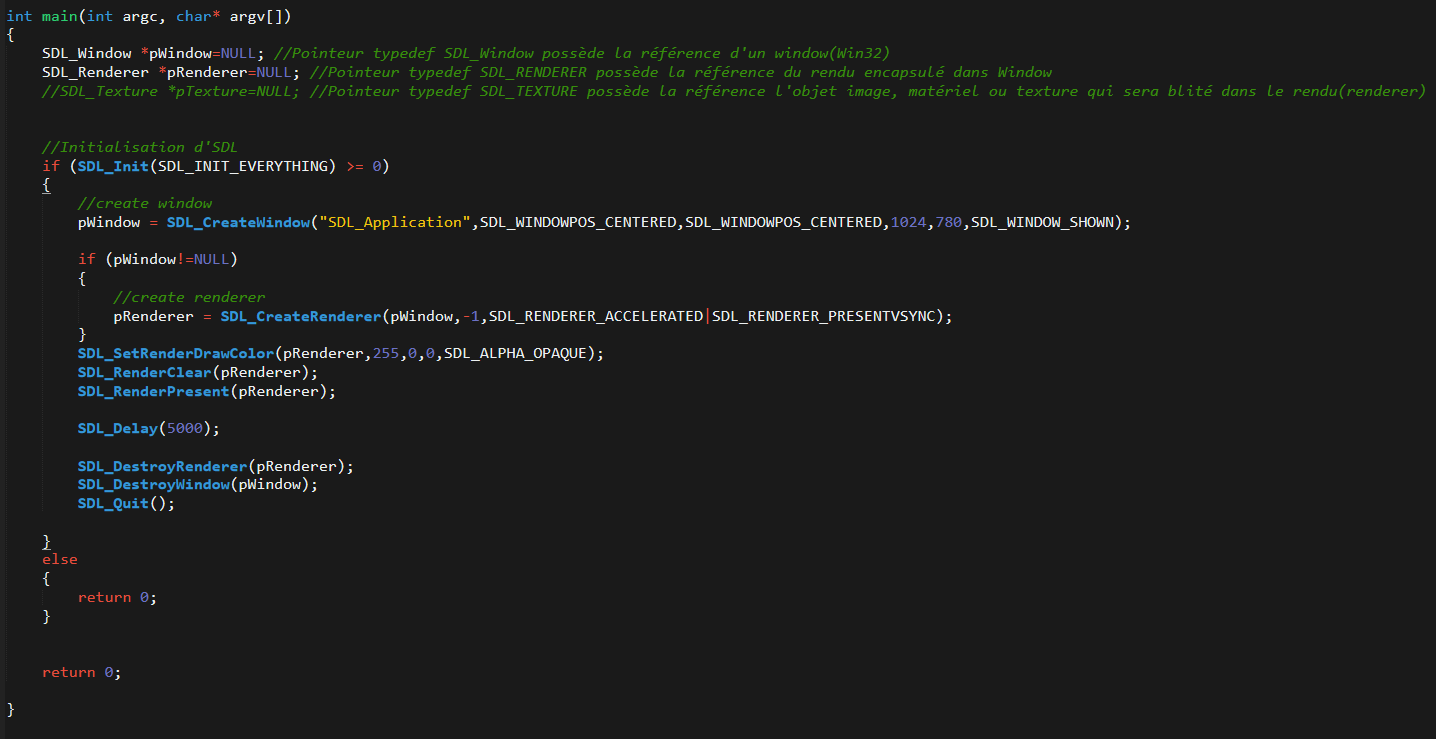
Pour déclarer un rectangle, on écrit :



**11) Les couleurs avec la SDL2 : donnez les instructions permettant de gérer la couleur avec la SDL 2**

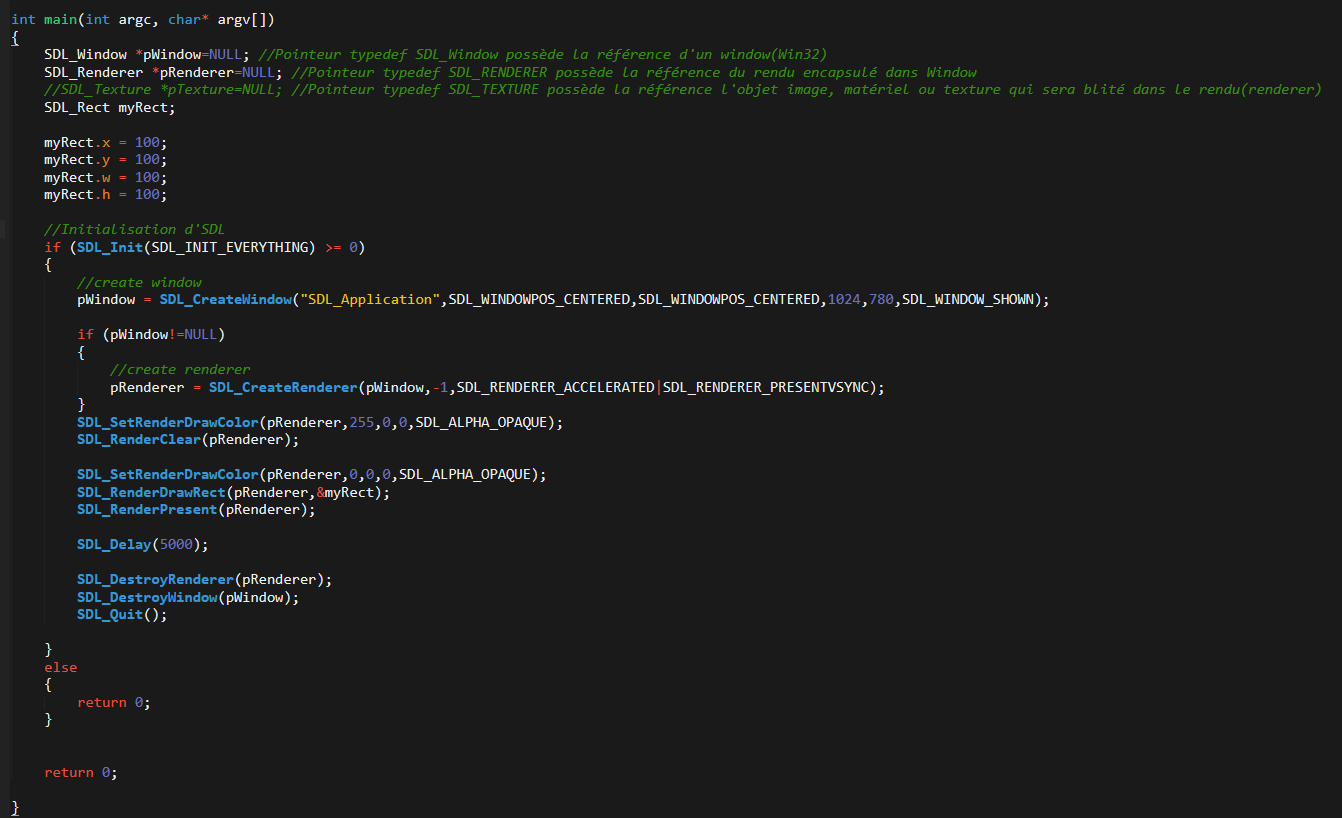
Pour définir la couleur que SDL va utiliser pour dessiner, on utilise la fonction SDL\_SetRenderDrawColor (déjà vue, voir Question 9 « **Changer la couleur d’affichage** »)

**12) Donnez le code permettant d’afficher un fond rouge dans le rendu.**

****

**13)** **Dessiner dans le rendu : donnez le code des fonctions permettant de dessiner dans le rendu les formes suivantes : (voir screenshot) - Carré vide - Carré plein - Cercle vide - Cercle plein.**

**Carré vide :**

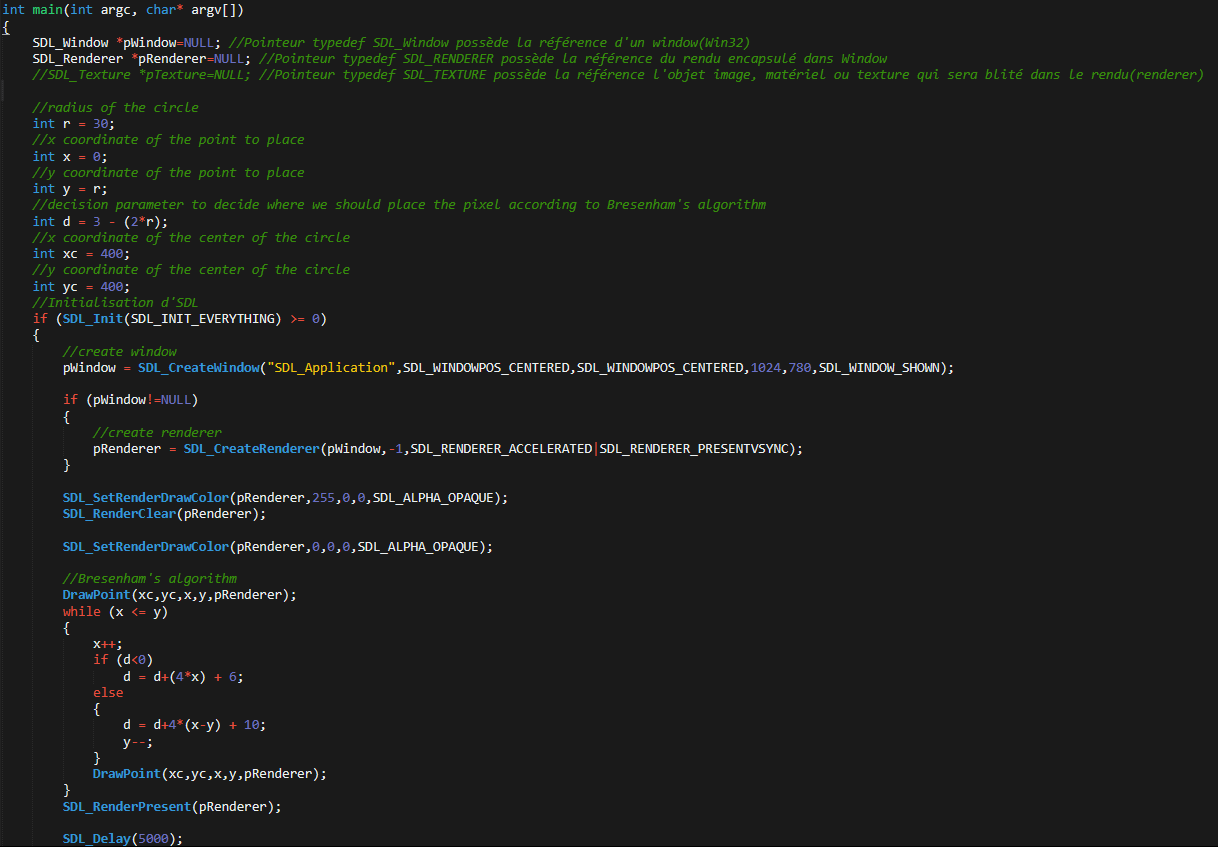
****

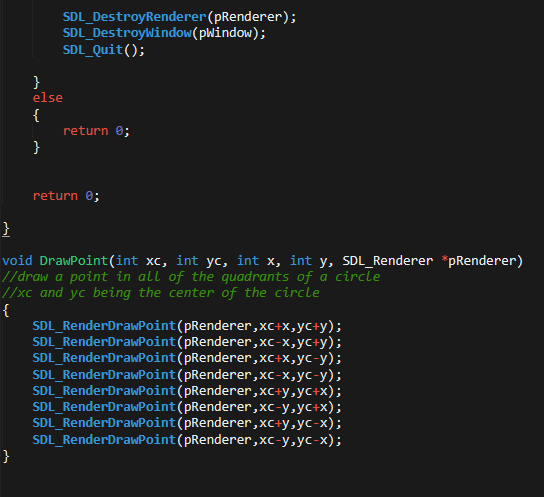
**Carré plein :** même code mais on remplace la ligne

****

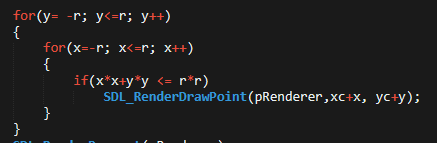
par

****

**Cercle vide : **

****

**Cercle plein :** on remplace juste l’algorithme de Bessenham (juste avant le SDL\_RenderPresent) par ce bout de code :



**14) Quelles sont les fonctions permettant de dessiner des points et des lignes ? (5 points)**

Pour dessiner un point on utilise la fonction suivante qui prend en paramètre le renderer, ainsi que les coordonnées x et y du point que l’on va placer.

int SDL\_RenderDrawPoint(SDL\_Renderer\* renderer,

int x,

int y)

Pour dessiner une ligne, on utilise la fonction ci-dessous, qui prend en paramètre le renderer, les coordonnées x et y du début de la ligne et les coordonnées x et y de la fin de la ligne.

int SDL\_RenderDrawLine(SDL\_Renderer\* renderer,

int x1,

int y1,

int x2,

int y2)

**15) A quoi servent les fonctions SDL\_RenderClear et SDL\_RenderPresent (5 points)**

SDL\_RenderClear sert à effacer le contenu présent dans le Render en utilisant la couleur d’affichage actuelle, et SDL\_RenderPresent sert à actualiser le contenu du Render (étant donné que ça a déjà été vu, pour plus d’infos, se référer à la Question 9)

**16) Expliquez SDL\_Delay (5 points)**

void SDL\_Delay(Uint32 ms)

Cette fonction permet de dire au programme d’attendre le temps passé en paramètre (millisecondes) avant de continuer l’exécution du reste.

**17) Qu’est-ce qu’une surface en SDL 2 (5 points)**

Une surface est l’équivalent de la texture qui est utilisée en SDL2, la surface était surtout utilisée en SDL 1.2. C’est plus ou moins un rectangle de pixels qui vont être affichés.

**18) Donner le code pour créer une surface (10 points)**

SDL\_Surface\* SDL\_CreateRGBSurface(Uint32 flags,

int width,

int height,

int depth,

Uint32 Rmask,

Uint32 Gmask,

Uint32 Bmask,

Uint32 Amask)

Voici la fonction permettant de créer une surface : les flags seront à 0, ensuite il y a la longueur/largeur de la surface, depth correspond au nombre de bits par pixel (généralement 32 bits par pixel) et les quatres masks permettent à SDL de savoir comment extraire la couleur de chaque pixel (on mettra 0). La fonction retourne un pointeur sur SDL\_Surface.

**19) Dessiner dans une surface : Donnez le code de la fonction SDL\_FillRect (10 points)**

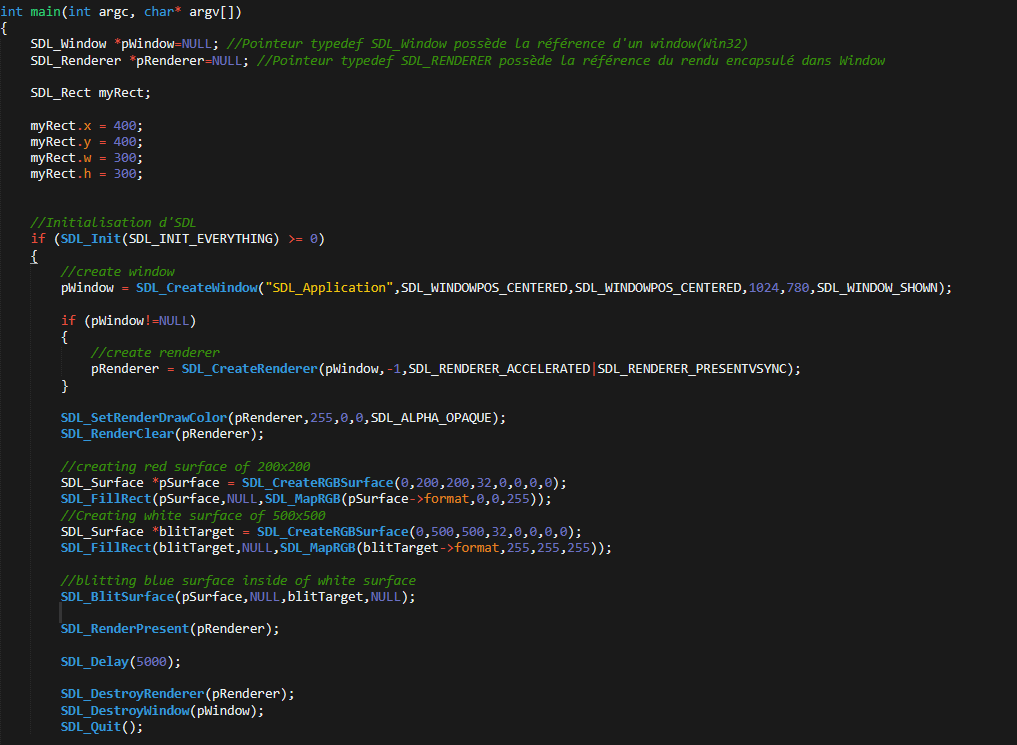
int SDL\_FillRect(SDL\_Surface\* dst,

const SDL\_Rect\* rect,

Uint32 color)

La fonction prend en paramètre la surface dans laquelle on va dessiner, le rectangle à dessiner (qu’il faudra déclarer au préalable et lui attribuer ses valeurs) ou bien mettre NULL pour remplir l’entièreté de la surface, et la couleur

**20) SDL\_BlitSurface(..) : donnez le code pour tester cette méthode. (5 points)**



(Cela n’affichera rien car pour afficher une surface, il faut la convertir en texture… !)

**21) Les textures : Qu’est-ce qu’une texture ? Donnez le code permettant de créer une texture. (10 points)**

Les textures sont l’équivalent des surfaces mais en SDL 2.0, elles utilisent l’accélération matérielle et de ce fait sont beaucoup plus optimisées que les surfaces (c’est pourquoi on les utilise au lieu des surfaces)

SDL\_Texture\* SDL\_CreateTexture(SDL\_Renderer\* renderer,

Uint32 format,

int access,

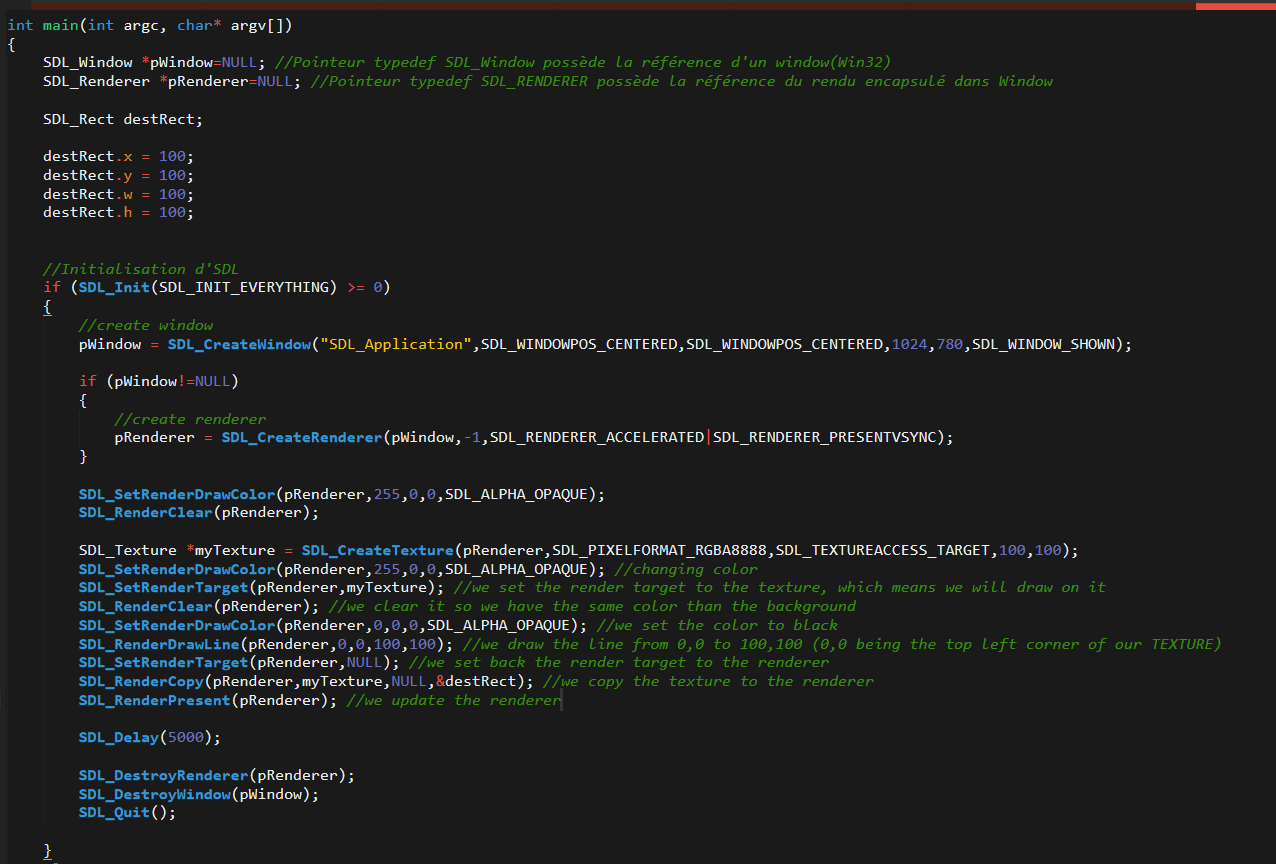
int w,

int h)

La fonction prend en paramètre le renderer dans lequel on va créer la texture, le format de pixel (il y en a plusieurs mais le format SDL\_PIXELFORMAT\_RGBA8888 permet de coder la couleur avec quatre nombres de 8 bits entre 0 et 255, il y a ensuite le format d’accès (on mettra SDL\_TEXTUREACCESS\_TARGET sinon on ne pourra presque rien faire), et enfin la largeur et la hauteur)

**22) Dessinez dans une texture : SDL\_RenderTarget, donnez le code pour tester cette méthode. (10 points) + 23) SDL\_RenderCopy : A quoi sert cette méthode, donnez le code pour tester cette méthode (10 points)**

Ici on dessine une ligne aux coordonnées 100,100 et de longueur 100,100 (par rapport au coin gauche de l’écran qui est l’origine)  
RenderCopy sert à copier une texture au render qui est défini par le SetRenderTarget

****

**24) SDL\_QueryTexture : A quoi sert cette méthode, donnez le code pour tester cette méthode (10 points)**

Cette méthode sert à récupérer les informations d’une texture, notamment ses dimensions.