cssm wide

Liste des différents outils disponibles grâce à la « librairie » cssm_wide :

```
void  AddToMessageBox(char*,int,int,int,int,unsigned short*);          // Ajoute une entrée dans une boite aux coordonnées (x,y)
void  AddToMessageBoxEx(char*,struct FakeWindowPos,unsigned short*);   // ...idem mais avec moins de paramètres ^^
void  AfficherXY(char*,int,int);                                       // Affiche un texte aux coordonnées (x,y)
void  BackToShell(short,short);                                        // Fonction qui permet d'afficher le prompt au bon endroit
void  EffacerEcran();                                                  // Efface l'écran
void  SetCursor(int,int);                                              // Permet de placer le curseur à une coordonnée (x,y)
void  GetConsoleDimensions(unsigned int*,unsigned int*);               // Permet de récupérer les dimensions d'une console (devrait fonctionner sur Windows)
// Ajouté le 19 octobre 2014
void Log(FILE*,char*);                                                 // Permet d'ajouter des informations dans le fichier de log...
void NoCarriage(char*);                                                // Permet d'enlever les caractères '\n' et '\r' d'une chaîne de caractères...
// Ajouté le 31 octobre 2015
void InitCSSMWIDE(char*);                                              // Permet d'initialiser la librairie CSSM_WIDE
// Ajouté le 05 novembre 2015
void setForegroundColor(struct s_RGB);                                 // Permet de spécifier la couleur RGB de l'avant plan
void setBackgroundColor(struct s_RGB);                                 // Permet de spécifier la couleur RGB de l'arrière plan
// Ajouté le 4 juin 2016
void PaintBox(struct FakeWindowPos,struct s_RGB);                      // Permet de "peindre" le background d'un cadre...
void  Push(void*,Stack*,short);                                        // Permet de "pusher" (mettre sur la pile) une variable de n'importe quel type (haute voltige)
void* Pop(Stack*,short);                                               // Permet de "popper" (enlever de la pile) une variable de n'importe quel type (haute voltige)
void  EmptyStack(Stack*);                                              // Permet de vider la pile
int   DisplayAligned(char*,int,int,int,int);                           // Affiche un texte avec une justification aux coordonnées (x,y)
int   getch(void);                                                     // Effectue une saisie d'un caractère sans besoin d'appuyer sur return
int   nbgetch(void);                                                   // Effectue une saisie d'un caractère sans besoin d'appuyer sur return (non bloquant)
int   getchAt(int,int);                                                // Effectue un getch aux coordonnées (x,y)
int   wstrlen(char*);                                                  // Calcule le nombre de caractères lisibles sans se préoccuper de l'encodage
char* getnchar(int);                                                   // Effectue la saisie d'une chaîne de caractères
char* getncharAt(int,int,int);                                         // Effectue la saisie d'une chaîne de caractères aux coordonnées (x,y)
char* getncharAtEx(struct FakeWindowPos);                              // ...même chose mais nous ne pourrons pas dépasser les marges du cadre...
char* getRGBString(struct s_RGB);                                      // Permet de convertir une structure s_RGB en chaîne ANSI (foreground)
char* getReverseRGBString(struct s_RGB);                               // Même chose mais pour le background (affichage inversé)
char* getStringFromInputBox(int,int,int,int,char*);                    // Fonction émulant un contrôle texte en mode console
char* splitDateToTime(char*);                                          // Permet de récupérer l'heure
chrono    getTimeElapsed(char*,char*);                                 // Permet de connaître le temps écoulé
struct FakeWindowPos DrawBox(int,int,int,int,int);                     // Trace une boite
struct FakeWindowPos DrawBoxWithRGB(int,int,int,int,struct s_RGB);     // Même chose mais en RGB
struct FakeWindowPos DrawTitledBox(int,int,int,int,int,char*);         // Trace une boite avec un titre
struct FakeWindowPos DrawTitledBoxWithRGB(int,int,int,int,struct s_RGB,char*,struct s_RGB,short);   // Amélioration de DrawTitledBox pour la prise en charge de l'RGB

struct s_RGB getRGB(short,short,short,bool);                           // Permet de convertir des valeurs RGB en structure s_RGB
```

AfficherXY

Cadre d'utilisation : permet d'afficher un texte à un emplacement spécifique à l'écran.

Entrées :

pMsg : Une chaîne de caractères

posX : La position en terme de colonne (x)

posY : La position en terme de ligne (y)

Sortie :

Néant

Exemple d'utilisation :

```
AfficherXY("Coucou !!",2,4);
```

AddToMessageBox

Cadre d'utilisation : permet d'afficher un message dans une « boîte » spécifique. Le concept de « boîte » concerne n'importe quel élément visuel à l'écran créé à l'aide des fonctions de la « librairie ». Cette fonction est utile dans le cas où vous n'avez qu'une seule « boîte » servant à afficher les messages, si vous avez plusieurs « boîtes » susceptibles d'afficher des messages il convient alors d'utiliser **AddToMessageBoxEx** qui est plus adaptée. Les messages sont affichés jusqu'à ce que le nombre maximum de messages soit atteint.

A ce moment la « boite » est vidée de son contenu, tous les messages précédents sont oubliés.

Entrées :

pMessage :	la chaîne de caractères à afficher
posx :	la position en X à partir de laquelle afficher le message (faites en sorte de ne pas sortir de la « boîte » ce serait mieux!!)
maxX :	indique le nombre de colonnes maximum, la taille maximum du message
maxY :	indique le nombre de lignes maximum, le nombre maximum de messages que pourra contenir la « boîte »
basevert :	la position Y à partir de laquelle afficher le message
pCurrentNumber :	le nombre actuel de lignes contenues dans la « boite » → **valeur modifiée**

Sortie :

Néant

Exemple d'utilisation :

```
AddToMessageBox("\x1b[34;1mDéploiement de la flotte...\x1b[0m",4,80,12,4,&nbreMessages);
```

Il faut considérer que, avant d'appeler cette fonction, vous ayez créé une « boîte », par exemple via **DrawTitledBox(3,3,16,84, COULEUR,"[Titre]")**.

Ainsi une « boîte » sera créée, elle fera 80 colonnes et 12 lignes, les messages devraient s'afficher à partir de la ligne 4 et de préférence à la colonne 4.

AddToMessageBoxEx

Cadre d'utilisation : fonctionne de la même manière que **AddToMessageBox** mais permet une meilleure prise en charge pour un contexte où il existerait plusieurs « boîtes » censées afficher des messages.

Entrées :

pMessage : le message à ajouter dans la « boîte »

datas : est une structure normalement récupérée lors de la création de la « boîte » ceci facilite la gestion de plusieurs « boîtes »

pCurrentNumber : permet de mettre à jour le nombre de messages contenus dans cette «boîte » → **valeur modifiée**

Sortie :

Néant

Exemple d'utilisation :

```
...
positionMessages=DrawTitledBoxWithRGB(2,43,23,MAX_COLONNES-2,
                                              getRGB(25,153,195,false),"[Messages]",getRGB(90,153,155,true),COLORED_SEPARATE);
...
switch(targetbox)
{
     case BX_CLI:
                    AddToMessageBoxEx(pMessage,positionListeClients,&nbMessages_clients);
                    break;
     case BX_MSG:
                    AddToMessageBoxEx(pMessage,positionMessages,&nbMessages_comm);
                    break;
     ...
                    break;
     ...
}
...
```

BackToShell

Cadre d'utilisation : permet de sortir proprement d'une application console dans laquelle des cadres ont été tracés. Dans ce genre d'application il convient de laisser la dernière ligne disponible pour pouvoir afficher la ligne d'invite du shell. Lorsque le programme rend la main au shell, peu importe la raison, la ligne d'invite du shell est affichée de manière propre.

Entrées :

posx : colonne à laquelle positionner le curseur avant de quitter le programme

posy : ligne à laquelle positionner le curseur avant de quitter le programme

Sortie :

Néant

Exemple d'utilisation :

```
BackToShell(1,58);
```

Lorsque pour une raison ou une autre, sauf à cause d'erreurs de segmentation, qu'il n'est pas possible, en C, d'intercepter, il convient de « sortir proprement » du programme. La fonction **BackToShell** va positionner le curseur à la première colonne et la dernière ligne de votre console avant de quitter le programme. Une fois que le shell va récupérer la main qui sera rendue par le programme à la fin de son exécution, il va afficher une nouvelle invite de commande là où nous avons décidé de l'afficher.

Illustration 1: Exemple de sortie propre d'un programme en mode console

DisplayAligned

Cadre d'utilisation : permet d'afficher un texte de manière alignée. Vous avez le choix entre trois types d'alignements : à gauche, au centre et à droite.

Entrées :

- **pMsg** : le message à afficher dans le cadre
- **posX** : position de la colonne à partir de laquelle nous allons afficher le message
- **posY** : position de la ligne sur laquelle nous allons afficher le message
- **widthmax** : nombre maximum de colonnes (logiquement la largeur de votre console ou de la « boîte » dans laquelle afficher du texte)
- **align** : code indiquant si il faut aligner à gauche, au centre ou à droite.

Les « valeurs » possibles de **align** sont : **ALIGN_LEFT**, **ALIGN_CENTER**, **ALIGN_RIGHT** ou respectivement 0,1,2.

Sortie :

Néant

Exemple d'utilisation :

```
DrawBox(1,29,1,80,31);
DisplayAligned("Aligné au centre",1,30,80,ALIGN_CENTER);
DrawBox(1,32,1,80,31);
DisplayAligned("Aligné à gauche",1,33,80,ALIGN_LEFT);
DrawBox(1,35,1,80,31);
DisplayAligned("Aligné à droite",1,36,80,ALIGN_RIGHT);
```

Ces instructions permettent de tracer trois cadres et dans ceux-ci d'afficher, de manière alignée, différents messages.

DrawBox

Cadre d'utilisation : cette fonction trace une « boîte » de forme rectangulaire ou carrée en fonction des valeurs passées en paramètre.

Entrées :

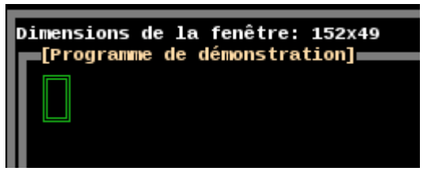
posX : indique la position en x du coin supérieur gauche.
posY : indique la position en y du coin supérieur gauche.
height : indique la hauteur (en nombre de lignes) de la « boîte ».
length : indique la largeur (en nombre de colonnes) de la « boîte ».
color : indique le code couleur à utiliser pour dessiner le cadre de la « boîte ».
Ce code est un entier positif, vous pouvez trouver tous les codes standards utilisés dans la norme ANSI en suivant le lien du cours de logique de programmation appliquée en C (page 110).

Sortie :

Une structure **FakeWindowPos** pour pouvoir plus aisément retrouver les caractéristiques de la boîte ainsi tracée.

Exemple d'utilisation :

```
PositionBoite=DrawBox(4,4,1,2,32);
```



En fonction de la police d'écran utilisée, il sera possible ou non de faire des cadres « carrés ». Tout repose sur le bon dosage des paramètres **height** et **length**.

Ici il aurait mieux valu utiliser le terme anglais « width », m'enfin bon.

DrawBoxWithRGB

Cadre d'utilisation : cette fonction est une extension de la première, elle permet de tracer le cadre en utilisant les codes couleurs ANSI certes, mais cette fois avec beaucoup plus de nuances : 255 niveaux de Rouge, Vert et Bleu.

Entrées :

- **posX** : indique la position en x du coin supérieur gauche.
- **posY** : indique la position en y du coin supérieur gauche.
- **height** : indique la hauteur (en nombre de lignes) de la « boîte ».
- **length** : indique la largeur (en nombre de colonnes) de la « boîte ».
- **color** : pour obtenir le code couleur vous devez passer par la fonction getRGB().

Sortie :

Une structure **FakeWindowPos** pour pouvoir plus aisément retrouver les caractéristiques de la boîte ainsi tracée.

Exemple d'utilisation :

```
int wWidth,wHeight;
GetConsoleDimensions(&wWidth,&wHeight);
PositionFenetrePrincipale=DrawBoxWithRGB(1,1,wHeight,wWidth-1,getRGB(125,125,125,true));
```

Ces instructions permettent de tracer un cadre faisant le périmètre de la console, peu importe sa taille. Cela pourrait s'avérer pratique.

DrawTitledBox

Cadre d'utilisation : permet de tracer une boîte à laquelle nous allons afficher un « titre ». Le « titre » permet d'apporter une signification particulière à une boîte que vous auriez tracé. Par exemple, il permet à l'utilisateur de savoir, de manière visuelle, ce que contient cette boîte.

Entrées :

posX : valeur en x de la coordonnée du coin supérieur gauche (colonne).

posY : valeur en y de la coordonnée du coin supérieur gauche (ligne).

height : hauteur en lignes.

length : largeur en colonnes.

color : valeur au format ANSI de la couleur du cadre (et du titre).

pTitle : chaîne de caractères représentant le titre.

Sortie :

struct FakeWindowPos

Exemple d'utilisation :

```
positionMessages=DrawTitledBox(2,2,14,12,46,"[Titre de la fenêtre]");
```

Permet de tracer une boite à partir de la coordonnée (2,2) de 14 lignes et 12 colonnes, avec le code couleur 46 dont le titre est "[Titre de la fenêtre]".

positionMessages reçoit en retour de la fonction les coordonnées utiles pour pouvoir adresser des messages à cette boîte (**voir page** 5).

DrawTitledBoxWithRGB

Cadre d'utilisation : fonctionne de la même manière que la fonction précédente mais permet d'apporter quelques comportements différents. Par exemple vous pouvez spécifier un titre de couleur différente à la couleur du cadre. Ces couleurs sont exprimées au format RGB, via l'utilisation de **getRGB()**.

Entrées :

posX : valeur en x de la coordonnée du coin supérieur gauche (colonne).

posY : valeur en y de la coordonnée du coin supérieur gauche (ligne).

height: spécifie la hauteur de la boîte.

length : spécifie la largeur (en colonnes) de la boîte.

color : indique le code couleur **du cadre** au format RGB.

pTitle : chaîne de caractères représentant le titre à afficher.

titlecolor : indique le code couleur **du titre** au format RGB.

design : indique par un code comment les couleurs du titre et du cadre seront traitées.

Concernant le « design » voici la signification des valeurs possibles :

COLORED_BOX : prend en compte le paramètre **color** uniquement pour tracer le cadre de la boîte.

COLORED_TITLE : prend en compte le paramètre **titlecolor** uniquement pour la couleur du texte composant le titre.

COLORED_BOTH : prend en compte le paramètre **color** pour le titre ET le cadre.

COLORED_SEPARATE : prend en compte les paramètres **color** ET **titlecolor**. **color** pour le cadre et **titlecolor** pour le titre.

Sortie :

struct FakeWindowPos

Exemple d'utilisation :

```
positionListeClients=DrawTitledBoxWithRGB(2,5,10,80,
                                          getRGB(70,193,195,false),
                                          "[Liste des clients connectés]",
                                          getRGB(90,213,215,true),COLORED_SEPARATE);
```

EffacerEcran

Cadre d'utilisation : aurait pu s'appeler **cls()** ou **clearscreen()**. Voici donc une incursion francophone dans un monde anglo-saxon.

Cette fonction sert à effacer l'écran et repositionner le curseur à la première colonne de la première ligne.

Entrées : Néant.

Sortie : Néant.

Exemple d'utilisation :

```
EffacerEcran();
```

getch

Cadre d'utilisation : cette fonction a été écrite pour pouvoir permettre facilement de récupérer une frappe au clavier en mode console (terminal) sous Linux.

En effet, sous Linux, la gestion du terminal est beaucoup plus fine que sous Windows dans le mode « invite de commande ».

Entrées : Néant.

Sortie : un caractère de type **char**.

Exemple d'utilisation :

```
char EntreeClavier=getch();                    // trop simple je sais...
```

Permet de ranger le caractère, acquis au clavier, dans la variable **EntreeClavier**.

La variable recevant le caractère entré peut-être de type **int** ou **char**.

getchAt

Cadre d'utilisation : cette fonction permet de faire la même chose que le **getch()** mais, en plus, permet de spécifier à quel endroit il faut positionner le curseur pour attendre la saisie du caractère.

Entrées :

posX : valeur en x de la coordonnée du coin supérieur gauche (colonne).

posY : valeur en y de la coordonnée du coin supérieur gauche (ligne).

Sortie :

un caractère de type **int** ou **char**.

Exemple d'utilisation :

```
char EntreeApresTexteAffiche=getchAt(15,2);
```

GetConsoleDimensions

Cadre d'utilisation : cette fonction permet de récupérer les dimensions d'une fenêtre de type terminal (en mode console).

Entrées :

colonnes : reçoit le nombre de colonnes composant la fenêtre de terminal.

lignes : reçoit le nombre de lignes composant la fenêtre de terminal.

Sortie :

Néant.

Exemple d'utilisation :

```
GetConsoleDimensions(&wWidth,&wHeight);
```

Il faut passer les variables (paramètres) représentant les lignes et les colonnes **par adresse**. Voir le cours concernant l'utilisation des pointeurs.

getnchar

Cadre d'utilisation : extension de la fonction **getch()**. Cette fonction est appropriée si vous voulez spécifier une longueur maximum aux données que vous voulez acquérir au clavier. La fonction retourne à la fonction appelante une fois que la taille maximum est atteinte ou que vous ayez appuyé sur entrée (***return***).

Entrées :

maximum : nombre de caractères maximum pour composer la chaîne de caractères.

Sortie :

un pointeur sur une variable de type **char**.

Exemple d'utilisation :

```
char *pasbesoindespecifierlataille=getnchar(12);
```

Comme le nom de la variable l'indique, il n'est pas nécessaire de spécifier une taille à votre chaîne de caractères, la fonction s'occupe de réserver l'espace nécessaire pour contenir les données acquises au clavier. C'est là toute la puissance et l'utilité de cette fonction (et de l'***allocation dynamique*** en langage C).

getncharAt

Cadre d'utilisation : effectue les mêmes opérations que **getnchar()** mais permet de spécifier à quelles coordonnées (x,y) il faut positionner le curseur.

Entrées :

posx : valeur en x de la coordonnée du coin supérieur gauche (colonne).

posy : valeur en y de la coordonnée du coin supérieur gauche (ligne).

maximum : nombre de caractères maximum pour composer la chaîne de caractères.

Sortie :

un pointeur sur une variable de type **char**.

Exemple d'utilisation :

```
char *pasbesoindespecifierlataille=getncharAt(30,7,12);
```

Cette fonction peut être très pratique pour, par exemple, récupérer des coordonnées de tir de maximum 4 caractères dans des zones spécifiques de l'écran.

getncharAtEx

Cadre d'utilisation : il s'agit d'une fonction qui permet de s'assurer qu'un utilisateur ne pourra pas dépasser la limite droite d'un cadre tracé à l'écran à l'aide des fonctions permettant de le faire.

Entrées :

DimensionsCadreCible : permet de définir le cadre dans lequel il faut opérer la saisie et ce en faisant en sorte que le nombre de caractères saisis soit inférieur à la largeur du cadre.

Sortie :

un pointeur sur une donnée de type char (une chaîne de caractères)

Exemple d'utilisation :

```
...
EffacerEcran();
GetConsoleDimensions(&wWidth,&wHeight);
PositionBoite=DrawBox((wWidth-30)/2,(wHeight-5)/2,5,30,32);
char *Chaine;
Chaine=getncharAtEx(PositionBoite);
...
```

nbgetch()

Cadre d'utilisation : Si vous voulez acquérir au clavier un seul caractère mais sans bloquer le programme et sans appuyer sur return.

Entrées :

Néant

Sortie :

Un caractère au format

Exemple d'utilisation :

```
...
// ne pas afficher les caractères saisis...

struct termios      oldt;
struct termios      newt;

tcgetattr(STDIN_FILENO, &oldt);
newt = oldt;

newt.c_lflag &= ~(ICANON | ECHO );
tcsetattr(STDIN_FILENO, TCSANOW, &newt);

temporisation.tv_sec=0L;
temporisation.tv_nsec=50000000L;                                    // 50.000.000 de nanosecondes (milliardièmes de secondes) 50 millisecondes

// Vider le buffer...


while(cSaisie!=EOF) cSaisie=nbgetch();

do
{
     clock_gettime(CLOCK_REALTIME,&tmpSeed);                        // la granularité de notre thread sera de l'ordre du milliardième de secondes (REALTIME)
     srand(tmpSeed.tv_nsec);                                        // nous "mélangeons" les nombres

     iScore=rand()%faces+1;                                         // choisi un nombre entre 1 et faces (6,8,4,12,20,100, peu importe)
     //wprintf(L"[fct_JetDe] %d\n",iScore);

     DisplayDice(idxDe,-1,target,getRGB(255,255,255,true));
     nanosleep(&temporisation,NULL);

     cSaisie=nbgetch();                                             // permet d'effectuer la saisie sans nécessairement devoir recourir aux threads...

     DisplayDice(idxDe,iScore,target,getRGB(255,255,255,true));
     nanosleep(&temporisation,NULL);

}while(cSaisie!=32);

...
```

getRGB

Cadre d'utilisation : permet de spécifier les valeurs pour les différentes intensités dans les trois « canaux » : rouge, vert et bleu. De plus il est possible de spécifier si le mode ***gras*** est activé ou pas.

Entrées :

Red : intensité de rouge entre 0 et 255.

Green : intensité de vert entre 0 et 255.

Blue : intensité de bleu entre 0 et 255.

bold : spécifie par **false** ou **true** si le mode **gras** doit être activé.

Sortie :

struct s_RGB

Exemple d'utilisation :

```
PositionFenetrePrincipale=DrawBoxWithRGB(1,1,wHeight,wWidth-1,getRGB(125,125,125,true));
```

Cette fonction permet de générer une structure **s_RGB** nécessaire à l'utilisation de la fonction **DrawBoxWithRGB()**.

getRGBString

Cadre d'utilisation : permet de récupérer la chaîne ANSI correspondant à la couleur RGB désirée (avant-plan).

Entrées :

target : spécifie le code couleur au format RGB (norme ANSI).

Sortie :

Une chaîne de caractères représentant la chaîne ANSI permettant d'appliquer les couleurs ainsi choisies ainsi que les codes ANSI pour afficher des informations à l'écran en ***gras***.

Exemple d'utilisation :

Cette fonction n'a pas de raison d'être utilisée autre part que dans les fonctions prévues dans **cssm_wide**.

getRerverseRGBString

Cadre d'utilisation : permet de récupérer la chaîne ANSI correspondant à la couleur RGB désirée (arrière plan).

Entrées :

target : spécifie le code couleur au format RGB (norme ANSI).

Sortie :

Une chaîne de caractères représentant la chaîne ANSI permettant d'appliquer les couleurs ainsi choisies ainsi que les codes ANSI pour afficher des informations à l'écran en ***gras***.

Exemple d'utilisation :

Cette fonction n'a pas de raison d'être utilisée autre part que dans les fonctions prévues dans **cssm_wide**.

getStringFromInputBox

Cadre d'utilisation : il s'agit d'une fonction qui simule les contrôle de type texte dans les pages Web, ou les objets en Java, par exemple, qui dessinent une zone de texte, on dit alors *un contrôle*, dans laquelle nous pourrions y introduire un texte (**JTextField**).

Entrées :

startposX : position de départ en x (colonne) du coin supérieur gauche du « contrôle ».

startposY : position de départ en y (ligne) du coin supérieur gauche du « contrôle ».

nbreCols : indique le nombre de colonnes (la longueur maximum de la chaîne retournée)

color : indique une couleur au format ANSI (un entier de type **short**)

***pTitle :** si **pTitle** est NULL le contrôle n'aura pas de titre, si il est différent de NULL le contrôle aura un titre.

Sortie :

un pointeur sur une donnée de type char (une chaîne de caractères)

Exemple d'utilisation :

```
char *pNom=getStringFromInputBox(3,3,40,43,"Nom");
char *pPrenom=getStringFromInputBox(3,6,40,43,"Prénom");
```

getTimeElapsed

Cadre d'utilisation : cette fonction permet d'obtenir le temps qui s'est écoulé entre deux moments précis dans votre programme.

Entrées :

pBegin : chaîne de caractères représentant l'heure (obtenue via la fonction **splitDateToTime()**) de départ.
pEnd : chaîne de caractères représentant l'heure (obtenue via la fonction **splitDateToTime()**) de référence signifiant la fin d'un processus dont il faut calculer l'intervalle de temps.

Sortie :

une structure de type **chrono**. Comprenant les heures, minutes et secondes écoulées entre les deux références de temps.

Exemple d'utilisation :

```
char heuredebut[SIZEDATE]="\0";  // SIZEDATE est défini dans cssm_wide.h comme constante...
char heurefin[SIZEDATE]="\0";
...
DateStart=time(NULL);
strcpy(heuredebut,asctime(localtime(&DateStart)));
...
DateEnd=time(NULL);
strcpy(heurefin,asctime(localtime(&DateEnd)));
...
chrono TempsJoue=getTimeElapsed(heuredebut,heurefin);
...
```

SetCursor

Cadre d'utilisation : permet de spécifier dans la console la position du curseur.

Entrées :

posx : valeur en x de la coordonnée du coin supérieur gauche (colonne).

posy : valeur en y de la coordonnée du coin supérieur gauche (colonne).

Sortie :

Néant.

Exemple d'utilisation :

```
SetCursor(4,4);
```

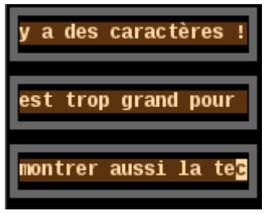
setBackgroundColor

Cadre d'utilisation : permet de définir la couleur de fond de la console.

Entrées : une structure de type **s_RGB** (obtenue grâce à la fonction **getRGB()**)

Sortie : Néant.

Exemple d'utilisation : **/exemples/AnimatedText**



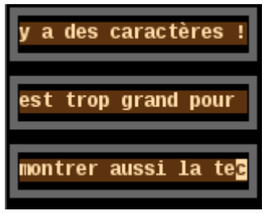
setForegroundColor

Cadre d'utilisation : permet de définir la couleur d'avant plan pour un texte de couleur sur un fond de couleur par exemple.

Entrées : une structure de type **s_RGB** (obtenue grâce à la fonction **getRGB()**)

Sortie : Néant.

Exemple d'utilisation : **/exemples/AnimatedText**



splitDateToTime

Cadre d'utilisation : cette fonction est utilisée par la fonction **getTimeElapsed()** elle sert à récupérer l'heure d'une date dans sa représentation système.

Entrées :

pCompleteDateString : chaîne de caractères qui contient la date système.

Sortie :

un pointeur sur une variable de type char (un tableau de caractères).

Exemple d'utilisation :

```
...
char *SplittedBegin=malloc(strlen(pBegin));  // on alloue dynamiquement la taille pour les deux chaînes de caractères...
char *SplittedEnd=malloc(strlen(pEnd));      // ...mais dans la réalité cette taille sera toujours la même...
...
memset(SplittedBegin,0,strlen(pBegin));      // On met les deux chaînes à 0
memset(SplittedEnd,0,strlen(pEnd));

strcpy(SplittedBegin,splitDateToTime(pBegin));    // On copie ce que la fonction splitDateToTime a renvoyé comme valeur de retour
strcpy(SplittedEnd,splitDateToTime(pEnd));
...
```

wstrlen

Cadre d'utilisation : dû aux soucis liés aux différentes langues utilisées, ici le français, il s'avère que la fonction strlen() retourne la longueur d'une chaîne de caractères mais comptabilise deux caractères (voire plus) pour des caractères accentués. Cela pose des soucis quand nous devons calculer l'espace occupé, en nombre de colonnes, par une chaîne de caractères sans se soucier de la taille occupée en mémoire. Cette fonction effectue le calcul exact du nombre de caractères sans compter le caractère de fin de chaîne '\0'.

Entrées :

pChaine : la chaîne dont nous voudrions obtenir la longueur **REELLE** qu'elle soit composée de caractères ASCII ou UTF8.

En d'autres mots nous voudrions compter le nombre de caractères la composant et non la place occupée en mémoire.

Sortie :

un entier représentant l'espace, en nombre de colonnes, occupé par la chaîne en question.

Exemple d'utilisation :

```
...
taillereelle=wstrlen(pMsg);
tmpMsg=malloc(sizeof(wchar_t)*taillereelle+1);
...
```

PaintBox

Cadre d'utilisation : permet de définir la « couleur de fond » d'un cadre… un peu à la manière du Midnight Commander (vous n'avez pas connu cela mais il existe toujours...)

Entrées :

target : une « fausse fenêtre » (**struct FakeWindowPos**)

color : une couleur au format RGBa (**struct s_RGB**)

Sortie : Néant

Exemple d'utilisation :

Log

Cadre d'utilisation : permet de « journaliser » des événements pour repérer des dysfonctionnements dans une application ou pour garder une trace de ceux-ci dans un fichier texte.

Entrées :

openedFile : descripteur de fichier de haut niveau (**FILE**).

pMessage : une chaîne de caractères représentant le message à « journaliser ».

Sortie : Néant (mais dans le fichier dont on a obtenu le descripteur le message sera ajouté avec la date et l'heure ainsi que les milliardièmes de seconde).

Exemple d'utilisation : **/exemples/rpg/core/rpg2014.c**

NoCarriage

Cadre d'utilisation : permet de « sucrer » (enlever) le caractère de « retour de ligne » ('**\n**') d'une chaîne de caractères… surtout utile dans les communications série (RS232), ...

Entrées :

pChaine : une chaîne de caractères.

Sortie :

Néant.

Exemple d'utilisation : **/exemples/arduino/controleur.c**

InitCSSMWIDE

Cadre d'utilisation : permet d'initialiser la librairie **cssm_wide**.

Entrées :

pLocale : chaîne de caractères reprenant un identifiant de la « locale » à utiliser, usuellement nous devrions indiquer **fr_BE.UTF-8**.

Sortie :

Néant.

Exemple d'utilisation : **/exemples/jauge/jauge.c**

Push

Cadre d'utilisation : nous sommes dans la « haute voltige », même si ce concept a depuis l'aube de l'informatique été utilisé, il permet de poser un « objet » (un type de données de n'importe quel type) sur la « pile ».

Entrées :

value : pointeur sur une donnée à placer sur la pile

pToStack : pointeur sur une pile (**Stack**)

DatasSize : nombre entier positif qui exprime la taille des données (en bytes) à placer sur la pile (généralement c'est sizeof(<type>) où <type> représente le type de données).

Sortie :

Néant.

Exemple d'utilisation : **/exemples/fonctions/récursion/automate.c**

Pop

Cadre d'utilisation : permet de récupérer un élément de n'importe quel type d'une pile quelconque.

Entrées :

pToStack : la pile dont il faut extraire « l'objet ».

DatasSize : taille en bytes de la données à extraire.

Sortie :

Un élément de n'importe quel type (**void***)

Exemple d'utilisation : **/exemples/fonctions/récursion/automate.c et aussi /exemples/puissance4/puissance4.c**

EmptyStack

Cadre d'utilisation : permet de vider complètement une pile.

Entrées :

target : un pointeur sur la pile à vider

Sortie :

Néant.

Exemple d'utilisation : **/exemples/fonctions/récursion/automate.c et aussi /exemples/puissance4/puissance4.c**

Table des matières