

Une clinique est établissement ou section d'établissement hospitalier public ou privée. Un univers où le rapport avec la clientèle est très important, où l'organisation est vitale. Ces centres hospitaliers sont de nos jours très prisés obligeant à la recherche d'une nouvelle approche au parcours d'un patient qui se présente à la clinique. La mise en place d'un bon milieu d'échange avec la clientèle repose, d'abord, sur une bonne organisation des heures de visite. Autrement dit, une gestion de rendez-vous moderne, innovatrice...un gestionnaire de rendez-vous totalement informatisé.

C'est dans cette mouvance que nous nous sommes lancé dans la réalisation d'une application permettant de gérer les rendez-vous dans une clinique. Nous l'avons baptisé **AGENDA ELECTRONIQUE**. Une application à usage facile, fiable, conviviale et facile à intégrer dans le milieu du travail.

Ainsi, l'objectif de ce projet -présenté dans ce mémoire- est la conception et la réalisation d'un système de prise de rendez-vous. Ce dernier permettra entre autre de créer, modifier ou supprimer des rendez-vous.

PARTIE 1 : ETUDE METHODOLOGIQUE

- ❖ Définition d'un Gestionnaire de rendez-vous
- ❖ Le langage C : historique et pourquoi C ?
- ❖ Système de Sauvegarde : les fichiers
- ❖ Le SDL

DÉFINITION D'UN GESTIONNAIRE DE RENDEZ-VOUS

Un gestionnaire de rendez-vous est un outil permettant de créer, modifier, supprimer et déplacer des rendez-vous. Il permet de mieux organiser notre quotidien de tous les jours. Les gestionnaires de rendez-vous ont pris différentes formes, ils ont évolué au fil du temps. Devenant de plus en plus performant. Dans cette ère du numérique, on parle d'agenda électronique.

Les agendas électroniques sont des applications (programme informatique) disponible partout jouant le même rôle qu'un agenda (comme vous le connaissez), voir même plus sophistiqué. Elles sont soit en ligne (application web) ou téléchargeable dans votre ordinateur ou portable (applications desktop).

En ce qui nous concerne nous réaliserons une application programmée en C.

LE LANGAGE C : HISTORIQUE ET POURQUOI C ?

Le langage C est un langage de programmation bas niveau (*dans l'ordre des un peu difficile*). La programmation est le faite de communiquer avec la machine et le langage C est le moyen de communication. C'est comme le wolof, le français ou l'anglais entre deux personnes. Le C est un des langages de programmation les plus utilisés, les autres langages plus modernes ont une syntaxe similaire à celle de C et reprennent en parti sa logique.

Un peu d'histoire

*Le langage C a été inventé au cours des années 1972 dans les laboratoires Bell. Il était développé en même temps que UNIX par Dennis Ritchie et Ken Thompson. En 1978, Kernighan fut le principale hauteur du livre *the C programming decribing* le langage. Le langage C fut normalisé en 1983. Entre 1994 et 1996 le groupe de travail ISO apporta quelques changements en ajoutant trois fichiers d'entête, un concernant certains nombres macros en rapport avec la norme des caractères et les autres concernant les caractères. En 1999 le C99, une nouvelle évolution à été normalisée par ISO. Avec comme nouveautés les tableaux dynamiques, les nombres complexes... Et puis en 2011 le C11, une nouvelle évolution introduisant le support de la programmation multi-thread, les expressions à types génériques et un meilleur support d'Unicode.*

Pourquoi avoir choisi le langage C ?

Nous avons eu à beaucoup faire sur le langage C, nous sommes allé jusqu'à l'étude des fichiers. Et il nous fallait un moyen d'assimiler tout ça, d'où le choix de ce sujet. Travailler sur ce projet est d'un bon exercice pour bien maîtriser les notions apprises sur le langage C.

Toutes les notions de base en C étaient à notre portée. Nous connaissions les différents types de variables, les conditions en C, les boucles, les structures, les énumérations, la gestion des fichiers... L'avantage pour nous d'avoir utiliser le C était qu'on avait déjà connaissance de ces notions. Il nous rester plus qu'à bien les assimiler et combiner cela à une autre bibliothèque pour la réalisation de l'application en fenêtre.

Si nous parvenons à la création des rendez-vous, une question subsiste une fois des rendez-vous créés et qu'il y ait moyen de les modifier, supprimer et afficher. Imaginons l'existence d'un tel programme et quittons le, alors toutes les données disparaissent. Ce qui veut dire que lorsqu'on relance le programme aucun rendez-vous ne sera disponible et à chaque fois qu'on quitte les rendez-vous créés disparaissent. Ce qu'il faut c'est un moyen de sauvegarde nous permettant ainsi de récupérer les informations enregistrées à chaque fois que l'application est lancée et d'enregistrer les modifications subit.

Les fichiers pourraient bien faire notre affaire.

Pourquoi les fichiers ?

Les fichiers furent une notion assez difficile pour nous. Notre choix des fichiers se justifie par une ambition de maîtrise de la « gestion des fichier en C ». Au lieu d'apprendre les bases de données, nous avons jugé plus intéressant de créer notre base de données avec que des fichiers. Avec une bonne gestion des ces derniers, nous obtenons un résultat similaire à celui d'une vraie base de données.

Gestion des fichiers en C

Pour se convaincre que les fichiers peuvent satisfaire notre affaire, il faut d'abord bien les connaître. En C, les variables n'existent que dans la mémoire vive. Et donc une fois le programme arrêté, ces variables perdent leurs valeurs. Heureusement qu'on peut écrire dans des fichiers et lire ces fichiers qui seront eux stockés dans le disque dur, on ne les perd pas même si le programme est arrêté.

Pour travailler sur des fichiers nous utilisons la bibliothèque stdio. Donc il faut s'assurer d'avoir inclus `<stdio.h>`. Dans cette bibliothèque se trouve des fonctions permettant de gérer les fichiers. Avant de travailler avec les fichiers, il faut leur créer un pointeur. Celle-ci est d'un type particulier, c'est le type **FILE**. La déclaration est de la forme `FILE* fichier = NULL ;`

Les fonctions les plus utilisées sont :

- La fonction `fopen` pour ouvrir un fichier. Le prototype est donné par :

```
FILE* fopen(const char* NomDuFichier, const char* ModeOuverture) ;
```

Le Nom du fichier pour indiquer le fichier que vous souhaitez ouvrir. Et le mode indique ce que vous souhaitez faire du fichier (lecture seulement, lecture et écriture, écriture seulement, ajout).

- Les fonctions d'écriture `fprintf` et `fwrite` selon le type de fichier.
 - `fwrite` pour les fichier binaires prototype

```
size_t fwrite(const void *pointeur, size_t size, size_t NombreMembre, FILE* stream) ;
```
 - `fprintf` sinon. Une fonction au mode de fonctionnement similaire à celui de `printf`. Assez facile à utiliser.
- Les fonctions de lecture `fscanf` et `fread` selon le type de fichier :
 - `Fread` pour le binaire, prototype :

```
size_t fread(const void *pointeur, size_t size, size_t NombreMembre, FILE* stream) ;
```

- fscanf sinon. Fonctionnement similaire à scanf.
- La fonction remove permet de supprimer un fichier. Syntaxe `remove(Nom_Du_Fichier)` ;
- Et la fonction de fermeture fclose :

```
int fclose(FILE* NomFichier) ;
```

Donc vous aurez à voir dans le code les fonctions énoncées ci-dessus. Ayant connaissance des fonctions permettant de lire et d'écrire dans des fichiers, on peut construire notre « base de donnée » avec que des fichiers. Nous ne développons pas sur ces fonctions, on les énonce seulement pour vous sachiez quelles fonctions nous avons utilisé et vous permettant ainsi de nous suivre dans la réalisation de l'application.

Les fichiers binaires

Un fichier binaire, en informatique, est un fichier qui n'est pas un fichier texte. Ils permettent aussi de stocker des informations comme les fichiers textes. Ci-dessus, nous avons énuméré les fonctions permettant de lire et d'écrire dans des fichiers binaires.

Nous comptons, en faites, utiliser les fichiers binaire pour enregistrer nos rendez-vous. Parce qu'il est plus facile de stocker et de récupérer des structures dans des fichiers binaires.

LA SDL

La SDL est une bibliothèque écrite en C, donc utilisée par les programmeurs en C. Elle est libre et gratuite, on peut réaliser des programmes commerciaux et propriétaires avec. Elle fonction aussi bien sous windows, Max ou linux. C'est une bibliothèque multi-plate-formes et elle est de bas niveau. Nous avons choisi la SDL, parce qu'on avait une main dessus et il fallait encore entrer au fin fond de la bibliothèque en découvrir le contenu. Cela nous permettrait de maîtriser une bibliothèque dédiée au programmeur en C. Le choix n'est pas vraiment intéressant, mais plutôt l'intérêt de la SDL.

La SDL fera de notre application comme toutes celles connu de nos jours. Elle nous fera franchir l'étape de la console. Avant de l'utiliser, nous vous présentons les éléments de la SDL auxquels nous avons fait appel.

Création de fenêtre

Nous sommes rentré dans la programmation en fenêtre, fini la console. Donc Avant tout, il faut générer une fenêtre dans laquelle nous dessinerons. Nous pourrons y mettre des images, créer des points, mettre des animations...

Avant toutes utilisation de la SDL, il faut l'initialiser avec `SDL_Init(flag)`. Un flag est comme une variable définie dans la bibliothèque indiquant des caractéristiques particulières (Pour les flags voir en annexe). Nous avons utilisé un flag (le `SDL_INIT_VIDEO`) suffisant pour nous satisfaire. Et parallèlement fermer avec `SDL_Quit()` ;

Une fenêtre dispose d'un boutons de sorti pour quitter, d'agrandissement et de réduction. On crée une fenêtre en utilisant la fonction *SDL_SetVideoMode*. On indique à cette fonction les paramètres suivant et dans cet ordre : La longueur, la largeur de la fenêtre, une valeur entière indiquant le nombre de couleurs et des flags (indique si la fenêtre est redimensionnable, par exemple). Pour créer une fenêtre, il nous faut un pointeur de type *SDL_Surface*. Exemple de création :

```
SDL_Surface* ecran = NULL ; ecran = SDL_SetVideoMode(100, 30, 32, SDL_HWSURFACE)
```

On peut donner à cette fenêtre une icône avec la fonction *SDL_WM_SetIcon* , comme ça :

```
SDL_WM_SetIcon("source/image", NULL) ;
```

Ajouter aussi une légende juste en haut de la fenêtre, avec *SDL_WM_SetCaption*, comme ça :

```
SDL_WM_SetCaption("description de la fenêtre") ;
```

Ces fonctions sont plus détailler dans les cours de SDL.

Pour maintenir la fenêtre, on utilise une boucle infinie. Sinon la fenêtre apparaît et disparaît en un clic. Et c'est dans cette boucle que tourne l'application et voici cette boucle

```
int continuer = 1 ;
while(continuer){
    //là tourne l'application
}
```

Création de Surface – Insertion d'images

Dessinez revient à tracer des rectangles et uniquement des rectangles. Tracer un rectangle, c'est créer une surface en SDL, donc un pointeur de type *SDL_Surface*. Et on utilise, pour cela, la fonction *SDL_CreateRGBSurface*. Elle prend un flag, suivi de la longueur, la largeur du rectangle, le nombre de couleurs, et quatre autres paramètres que vous pouvez mettre à 0. Exemple d'utilisation,

```
SDL_Surface *rectangle ; rectangle = SDL_CreateRGBSurface(SDL_HWSURFACE, longueur largeur, 16, 0, 0, 0, 0) ;
```

Ces rectangles peuvent être coloriés par la fonction *SDL_FillRect*. Exemple d'utilisation

```
SDL_Fillrect(rectangle, NULL, SDL_MapRGB(valeur_rouge, valeur_verte, valeur_bleue) ;
```

Cette fonction est aussi utilisée pour colorier la fenêtre, car initialement la fenêtre est toute noire. Pour avoir une fenêtre rouge, il vous suffit de faire ça :

```
SDL_FillRect(ecran ,NULL, SDL_MapRGB(255, 0, 0)) ;
```

Nous n'avons pas encore dessiné le rectangle, nous l'avons créé avec ces dimensions et sa couleur. Pour le dessiner sur la fenêtre, on fait un « Blittage » avec *SDL_BlittSurface*. Il faut indiquer à cette dernière le rectangle à blitter, dans quelle fenêtre le faire et la position.

Pour la position, on utilise une structure de type `SDL_Rect`, Cette structure stockera l'abscisse et l'ordonnée du blittage.

Exemple de blittage, `SDL_Rect position ; position.x = 0 ; position.y = 0 ; SDL_BlitSurface(Rectangle, NULL, ecran, &position)`

Maintenant, voyons les fonctions permettant de mettre des images. Nous ne vous présenterons que les fonctions que nous avons utilisées pour la création de notre application.

Et pour les images une seule à suffit, car elle permet de charger toutes les format d'images possibles (png, jpg, bmp...). Cette fonction nécessite l'installation de la bibliothèque `SDL_Image`. Si l'installation est faite on include la `SDL_image.h` par `include <SDL\SDL_image.h>`. Dans ce *header* se trouve la fonction `IMG_Load ("Source de l'image à mettre")` ; une image c'est aussi une surface, donc un pointeur de type `SDL_Surface`. Exemple `SDL_Surface *image = NULL ; image = IMG_Load("maphoto.jpg") ;`

Voici ces autres fonction permettant de gérer la transparence dans des exemple :

Exemple 1 : `SDL_SetAlpha(image, SDL_SRCALPHA, 125)` plus la valeur est petit, plus l'image devient transparente.

Exemple 2 : `SDL_SetColorKey(image, SDL_SRCSETCOLORKEY, SDL_MapRGB(0, 255, 0))` ; veut dire rend la couleur verte de l'image transparente.

Il est conseillé de mettre toutes les images qu'on aura à utiliser dans un même dossier (images) se trouvant à côté de l'exécutable. Ainsi la source d'une image serait de la forme **images\nomdelaphoto.format**.

Après utilisation, on libère les surfaces par `SDL_FreeSurface(NomDelaSurface)` ;

Gestion des événements

Un événement ici est toute action de l'utilisateur en rapport avec l'application. Ça peut être un déplacement de la souris, un clic, agrandissement de la fenêtre ou réduction... Tous ces événement peuvent être gérer par la SDL grâce à une structure de type `SDL_Event`. Le contrôle de ces événements permettra une interaction entre l'application et l'utilisateur. Par exemple, on peut créer des boutons tels que si l'utilisateur clic dessus ça ouvre un nouvelle surface ou dessiner un outil sur la fenêtre et que lorsque la souris bouge on fait bougé cet outil...

La création d'une application repose sur une bonne gestion des événements. Comme on la dit, il faut d'abord une structure de type `SDL_Event`. Déclaration `SDL_Event event` ;. Cette structure contient tous les types d'événement possibles. On récupère ces événements de deux façon :

- On bloque le programme jusqu'à l'arrivée d'un événement, on le récupère puis on effectue une action. Ceci se fait grâce à la fonction `SDL_WaitEvent()` ; syntaxe `SDL_WaitEvent(&event)` ; elle récupère l'événement et le stock dans "event".
- Ou bien, `SDL_PollEvent()` qui laisse le programme tourné même si il n'y pas d'événement et récupère un événement dès qu'il en arrive un. Syntaxe `SDL_PollEvent(&event)` ;

SDL_PollEvent est plus adéquate pour une meilleure gestion du temps, des animations...donc nous utiliserons celle là.

Selon l'événement obtenu, on effectue une action. Et voici comment ça marche en générale :

```
while(contiuer){
    while(SDL_PollEvent(&event)){
        switch(event.type){
            //selon le type on fait ce qu'on a à faire
        }
    }
}
```

Nous avons mis le *SDL_PollEvent* dans une boucle pour éviter une répétition non souhaitée. Par exemple, l'utilisateur clique sur "a" une fois donc l'événement c'est un clic sur "a". Le retour sur la boucle infinie même si l'utilisateur ne fait rien « event » vaudra toujours un clic sur "a" et ainsi de suite. Pour éviter cela on fait une boucle sur *SDL_PollEvent* qui transmettra **NULL** à « event » si l'utilisateur ne fait rien.

Les types d'événements que nous avons eu à gérer dans notre agenda sont :

- ✚ SDL_QUIT : pour quitter si l'utilisateur clic sur le boutons fermer ou le bouton escape du clavier.
- ✚ SDL_MOUSEMOTION : pour les déplacements de la souri
- ✚ SDL_MOUSEBUTTONDOWN : Les clics de la souri
- ✚ SDL_KEYDOWN : Gère l'appui d'un bouton du clavier
- ✚ SDL_VIDEORESIZE : L'agrandissement de la fenêtre

Ecrire dans une fenêtre

Souvent il faut un guide pour l'utilisateur, une légende pour certains boutons... tous ceci pour une meilleure compréhension de l'application. Elle doit être claire et facile d'usage. D'où l'importance des textes. Comme pour les images il nous faut une nouvelle bibliothèque la *SDL_ttf*.

Il faudra la téléchargé et l'installer puis sans oublier de l'inclure (include <SDL\SDL_ttf.h>). Cette bibliothèque regorge des fonctions permettant de charger des polices, de créer du texte...

Pour la police, c'est au programmeur d'en trouver et de les mettre tous dans un dossier (polices). Une police est obtenu grâce à un pointeur de type *TTF_Font*. Une fois déclarée, la police est chargée par la fonction *TTF_OpenFont*. Même fonctionnement que *SDL_LoadBMP*. Exemple de chargement de police, `TTF_Font *mapolice = TTF_OpenFont("polices\arial.ttf") ;`

Un texte est une surface, donc on déclare un pointeur de type *SDL_Surface*. On construit cette surface grâce à la fonction *TTF_RenderText_Blended* (ou *TTF_RenderText_Solid* ou *TTF_RenderText_Shaded*). Nous n'avons travailler qu'avec *TTF_RenderText_Blended*. Exemple `TTF_RenderText_Blended(mapolice, "mettez ici le texte à écrire", couleur) ;`

La couleur s'obtient ainsi `SDL_Color couleur = {valeur R, valeur V, valeur B}.`

Gestion du temps

Nous avons utilisé deux façons de gérer le temps. Une est en rapport avec la SDL et l'autre n'a rien à avoir avec.

- La première fait appel à la fonction *SDL_GetTicks*, une fonction qui renvoie le nombre de milliseconde écoulé depuis le lancement de l'application. Nous l'avons utilisé pour une animation de photos. Une animation du genre on blitte une photo et après chaque 30s on change la photo.
- La seconde manière c'est avec la bibliothèque **time**, on y accède en incluant le header *time.h*. Elle permet de récupérer la date de la machine ou l'application est exécutée. Elle contient la structure *tm*, permettant d'obtenir séparément le jour, le mois, l'année, l'heure, la minute... On se rend, tout de suite, compte qu'on pourrait dès lors connaître les rendez-vous périmés, ceux du jour et ceux qui sont pas encore arrivés.
On commence donc par déclarer une variable de type *time_t* pour stocker le temps. On le récupère la date avec la fonction *time*, et la fonction *localtime* pour obtenir la date plus en détail. Le code ressemble à ça :

```
time_t currenttime ;time(&currenttime) ;struct tm *mytime=localtime(&currenttime) ;
```


Le jour est donnée par *tm_mday*, le mois *tm_mon* + 1, l'année *tm_year* + 1900, l'heure *tm_hour* et les minutes *tm_min*.

Pourquoi toutes ces informations ?

Nous allons, juste après, vous présenter les fonctions que nous avons créées. Nous les avons créées à base des éléments ci-dessus. Tous ces éléments vont apparaître dans les lignes de code que nous vous montrons ci-après. Nous ne pouvons pas trop développer ces notions, notre but n'est pas de vous les enseigner. Par contre, vous les présentez rendra plus facile la lecture du code.

PARTIE II : MISE EN ŒUVRE

- ❖ Mode Console
 - Fonction créées
 - Démonstration
- ❖ Mode graphique
 - Fonction créées
 - Démonstration

Présentation des fonctions

Pour la réalisation de l'application proprement dite, nous ne nous sommes pas trop vite lancés dans le graphisme. Nous avons préféré y aller par étape, en mettant en place une application console servant de repère. Elle est constituée d'un ensemble de fonctions chacune exerçant une tâche particulière. Voici ces différentes fonctions :

Accueille

Cette fonction présente juste un menu demandant à l'utilisateur une action à réaliser.

```
int Accueille(int *nombre1, int *nombre2, int *nombre3){  
    int choix;  
  
    printf("-----GESTIONNAIRE DE RENDEZ-VOUS-----\n");  
  
    printf("\t1.Enregistrer un rv\t2.Modifier un rv\t3.Supprimer un rv\n\t4. Voir les rv\n");  
  
    scanf("%d", &choix);  
  
    system("cls");  
  
    initialement(nombre1, nombre2, nombre3);  
  
    return choix;  
}
```

Elle-même fait appel à une fonction nommée initialement. Celle-ci, récupère le nombre de rendez-vous pour chacun des médecins dès le lancement de l'application. La voici

```
void initialement(int *nbr_p1, int *nbr_p2, int *nbr_p3){  
    char* tab3[10] = {"Medecin3/p1.bin", "Medecin3/p2.bin", "Medecin3/p3.bin",  
"Medecin3/p4.bin", "Medecin3/p5.bin", "Medecin3/p6.bin", "Medecin3/p7.bin", "Medecin3/p8.bin",  
"Medecin3/p9.bin", "Medecin3/p10.bin"};  
    char* tab2[10] = {"Medecin2/p1.bin", "Medecin2/p2.bin", "Medecin2/p3.bin",  
"Medecin2/p4.bin", "Medecin2/p5.bin", "Medecin2/p6.bin", "Medecin2/p7.bin", "Medecin2/p8.bin",  
"Medecin2/p9.bin", "Medecin2/p10.bin"};  
    char* tab1[10] = {"Medecin1/p1.bin", "Medecin1/p2.bin", "Medecin1/p3.bin",  
"Medecin1/p4.bin", "Medecin1/p5.bin", "Medecin1/p6.bin", "Medecin1/p7.bin", "Medecin1/p8.bin",  
"Medecin1/p9.bin", "Medecin1/p10.bin"};  
    int i, j = 0;  
    FILE* fichier = NULL;  
    for(i=0; i<10; i++){  
        fichier = fopen(tab1[i], "rb");  
        if(fichier != NULL)  
            j++;  
        fclose(fichier);  
    }  
    if(j!=0)
```

```

        *nbr_p1 = j;
    else
        *nbr_p1 = 0;
    j = 0;
    for(i=0; i<10; i++){
        fichier = fopen(tab2[i], "rb");
        if(fichier != NULL)
            j++;
        fclose(fichier);
    }
    if(j!=0)
        *nbr_p2 = j;
    else
        *nbr_p2 = 0;
    j = 0;
    for(i=0; i<10; i++){
        fichier = fopen(tab3[i], "rb");
        if(fichier != NULL)
            j++;
        fclose(fichier);
    }
    if(j!=0)
        *nbr_p3 = j;
    else
        *nbr_p3 = 0;
    j = 0;
}

```

Les tableaux contiennent les noms des fichiers et les dossiers dans lesquels ils sont.

« Medecin1/p2.bin » indique le deuxième rendez-vous du medecin1. Pour récupérer le nombre de rendez-vous, par exemple, pour le medecin1 il suffit d'ouvrir les éléments du tableau un à un. Si l'ouverture ne marche pas *fopen* renvoie **NULL**, donc on ne compte pas sinon on compte. C'est ainsi qu'on stock le nombre de rendez-vous pour chaque médecin dans la variable correspondante (Nbr_p1, 2 ou 3).

La fonction principale

Elle fait appel à toutes les autres selon le choix effectué au niveau de la fonction accueil

```

int main(){
    int nbr_patient1 = 0, nbr_patient2 = 0, nbr_patient3 = 0, quit;
    do{
        switch(Accueille(&nbr_patient1, &nbr_patient2, &nbr_patient3)){
            case 1:
                Enregistrer(&nbr_patient1, &nbr_patient2, &nbr_patient3);
                break;
            case 2:
                Modifier();
                break;
            case 3:
                Supprimer();
                break;
        }
    } while (quit != 1);
}

```

```

        case 4:
            Affiche(&nbr_patient1, &nbr_patient2, &nbr_patient3);
            break;
        default :
            Acceuille(&nbr_patient1, &nbr_patient2, &nbr_patient3);
            break;
    }
    printf("5 => pour quitter Quitter, autre chiffre sinon \n");
    scanf("%d", &quit);
}while(quit != 5);
}

```

La fonction d'enregistrement

```

patient Enregistrer(int *nbr_rv1, int *nbr_rv2, int *nbr_rv3){
    int nbr_p;
    char* tab[10];
    char* tab3[10] = {"Medecin3/p1.bin", "Medecin3/p2.bin", "Medecin3/p3.bin",
"Medecin3/p4.bin", "Medecin3/p5.bin", "Medecin3/p6.bin", "Medecin3/p7.bin", "Medecin3/p8.bin",
"Medecin3/p9.bin", "Medecin3/p10.bin"};
    char* tab2[10] = {"Medecin2/p1.bin", "Medecin2/p2.bin", "Medecin2/p3.bin",
"Medecin2/p4.bin", "Medecin2/p5.bin", "Medecin2/p6.bin", "Medecin2/p7.bin", "Medecin2/p8.bin",
"Medecin2/p9.bin", "Medecin2/p10.bin"};
    char* tab1[10] = {"Medecin1/p1.bin", "Medecin1/p2.bin", "Medecin1/p3.bin",
"Medecin1/p4.bin", "Medecin1/p5.bin", "Medecin1/p6.bin", "Medecin1/p7.bin", "Medecin1/p8.bin",
"Medecin1/p9.bin", "Medecin1/p10.bin"};
    int i;
    printf("1. Medecin 1    2. Medecin 2    3. Medecin 3\n");
    int choix;
    scanf("%d", &choix);
    system("cls");
    switch(choix){
        case 1:
            for(i=0; i<10;i++)
                tab[i] = tab1[i];
            nbr_p = *nbr_rv1;
            break;
        case 2:
            for(i=0; i<10;i++)
                tab[i] = tab2[i];
            nbr_p = *nbr_rv2;
            break;
        case 3:
            for(i=0; i<10;i++)
                tab[i] = tab3[i];
            nbr_p = *nbr_rv3;
            break;
        default :
            printf("Vous devez indiquer avec quel medecin\n");
            main();
            break;
    }
}

```

```

    patient client = Creer_Patient();
    char *chaine;
    FILE *fichier = NULL;
    for(i=0; i<10; i++){
        fichier = fopen(tab[i], "rb");
        if(fichier == NULL && nbr_p !=0){
            chaine = tab[i];
            break;
        }
        fclose(fichier);
    }
    if(nbr_p == 0)
        chaine = tab[0];
    fichier = fopen(chaine, "wb");
    fwrite(&client, sizeof(client), 1, fichier);
    fclose(fichier);
    return client;
}

```

L'appel de cette fonction ouvre un nouveau menu demandant le médecin pour lequel on souhaite créer le rendez-vous. Selon ce choix, on utilise un des tableaux au-dessus et on récupère le nombre de rendez-vous (Nbr_rv1, 2 ou 3) déjà pris par ce médecin. Ce sont ces tableaux qui orientent les fichiers contenant les rendez-vous dans le dossier du médecin correspondant. Et ensuite on crée un patient, qui une structure, grâce à la fonction « *créer_patient* ». Un patient est une structure qui elle-même contient une structure de type rendez-vous. « *créer_patient* » fait appelle à « *créer_rv* ». Ces deux fonctions ont un fonctionnement similaire. Elles ne font que récupérer les saisies de l'utilisateur à la console. L'utilisation des fichiers binaires se justifie par l'implication des structures car il est plus rapide et facile d'écrire des structures dans des fichiers binaires. Une fois le patient créé, il est accompagné d'un numéro (sa position dans le tableau plus 1 pour avoir un repère), on le stock dans un fichier binaire pour ne pas le perdre.

```

rv Creer_Rv(){
    rv prv;
    printf("Jours : ");
    scanf("%s", prv.jours);
    printf("Heure : ");
    scanf("%d", &prv.heure);
    printf("Minutes : ");
    scanf("%d", &prv.minute);
    return prv;
}

patient Creer_Patient(){
    patient p;
    printf("-----Renseigner les champs-----\n");
    printf("Nom : ");
    scanf("%s", p.nom);
    printf("Prenom : ");

```

```

scanf("%s", p.prenom);
printf("age : ");
scanf("%d", &p.age);
printf("sexe(M/F) : ");
scanf("%s", p.sexe);
printf("-----Son rendez-vous-----\n");
p.SonRv = Creer_Rv();
return p;
}

```

Exemple, « Medecin1/p1.bin » est le nom du fichier qu'on indique à *fwrite*, pour lui dire de créer dans le dossier Medecin1 son premier rendez-vous appelé p1.bin. Même après arrêt du programme, on trouvera toujours le fichier « p1.bin » dans le dossier du Medecin1.

La fonction de suppression

```

void Supprimer(){
    char* tab[10];
    char* tab3[10] = {"Medecin3/p1.bin", "Medecin3/p2.bin", "Medecin3/p3.bin",
"Medecin3/p4.bin", "Medecin3/p5.bin", "Medecin3/p6.bin", "Medecin3/p7.bin", "Medecin3/p8.bin",
"Medecin3/p9.bin", "Medecin3/p10.bin"};
    char* tab2[10] = {"Medecin2/p1.bin", "Medecin2/p2.bin", "Medecin2/p3.bin",
"Medecin2/p4.bin", "Medecin2/p5.bin", "Medecin2/p6.bin", "Medecin2/p7.bin", "Medecin2/p8.bin",
"Medecin2/p9.bin", "Medecin2/p10.bin"};
    char* tab1[10] = {"Medecin1/p1.bin", "Medecin1/p2.bin", "Medecin1/p3.bin",
"Medecin1/p4.bin", "Medecin1/p5.bin", "Medecin1/p6.bin", "Medecin1/p7.bin", "Medecin1/p8.bin",
"Medecin1/p9.bin", "Medecin1/p10.bin"};
    int i, num, choix;
    FILE *fichier = NULL;
    printf("1. Medecin 1      2. Medecin 2      3. Medecin 3\n");
    scanf("%d", &choix);
    system("cls");
    switch(choix){
        case 1:
            for (i = 0; i < 10; i++)
            {
                tab[i] = tab1[i];
            }
            break;
        case 2:
            for (i = 0; i < 10; i++)
            {
                tab[i] = tab2[i];
            }
            break;
        case 3:
            for (i = 0; i < 10; i++)
            {
                tab[i] = tab3[i];
            }
            break;
    }
}

```

```

        }
        break;
    default :
        printf("Vous devez choisir le dossier devant subir les suppressions\n");
        main();
        break;
    }
    printf("Vous voulez supprimer un rv : (1) oui ou (2) non : ");
    scanf("%d", &choix);
    switch(choix){
        case 1:
            printf("Donner le numero du patient : ");
            scanf("%d", &num);
            remove(tab[num-1]);
            break;
        case 2:
            break;
        default :
            break;
    }
}

```

Comme pour l'enregistrement, un menu demandant le médecin. C'est la même action qui se répète. Chaque rendez-vous créé est accompagné d'un numéro vu qu'on travaille avec des tableaux. Pour supprimer un élément, il faut donc indiqué sa position. Et on effectue juste *un remove*. Exemple `remove("Medecin1/p1.bin") ; ⇔ remove(tab1[0]) ;`

La fonction de modification

De la même manière que la fonction de suppression, on doit lui indiquer le numéro du rendez-vous et elle s'occupe d'ouvrir le dossier correspondant en mode écriture. Selon ce que vous voulez modifier, vous renseignez le champ demandez et puis le fichier est enregistré à nouveau.

```

void Modifier(){
    char* tab[10];
    char* tab3[10] = {"Medecin3/p1.bin", "Medecin3/p2.bin", "Medecin3/p3.bin",
"Medecin3/p4.bin", "Medecin3/p5.bin", "Medecin3/p6.bin", "Medecin3/p7.bin", "Medecin3/p8.bin",
"Medecin3/p9.bin", "Medecin3/p10.bin"};
    char* tab2[10] = {"Medecin2/p1.bin", "Medecin2/p2.bin", "Medecin2/p3.bin",
"Medecin2/p4.bin", "Medecin2/p5.bin", "Medecin2/p6.bin", "Medecin2/p7.bin", "Medecin2/p8.bin",
"Medecin2/p9.bin", "Medecin2/p10.bin"};
    char* tab1[10] = {"Medecin1/p1.bin", "Medecin1/p2.bin", "Medecin1/p3.bin",
"Medecin1/p4.bin", "Medecin1/p5.bin", "Medecin1/p6.bin", "Medecin1/p7.bin", "Medecin1/p8.bin",
"Medecin1/p9.bin", "Medecin1/p10.bin"};
    int i, num, choix;
    printf("1. Medecin 1    2. Medecin 2    3. Medecin 3\n");
    scanf("%d", &choix);
    system("cls");
    switch(choix){
        case 1:
            for (i = 0; i < 10; i++)

```



```

        {
            tab[i] = tab1[i];
        }
        break;
case 2:
    for (i = 0; i < 10; i++)
    {
        tab[i] = tab2[i];
    }
    break;
case 3:
    for (i = 0; i < 10; i++)
    {
        tab[i] = tab3[i];
    }
    break;
default :
    printf("Pour modifier, indiquer le dossier qui doit subir les modifications\n");
    main();
    break;
}
printf("Indiquer le numero du patient : ");
scanf("%d", &num);
printf("(1) Modifier le rv\n(2) Remplacer le client \n");
scanf("%d", &choix);
patient p;
FILE *fichier = fopen(tab[num - 1], "rb");
fread(&p, sizeof(p), 1, fichier);
fclose(fichier);
switch(choix){
    case 1:
        printf("Jours : ");
        scanf("%s", p.SonRv.jours);
        printf("Donner l'heure : ");
        scanf("%d", &p.SonRv.heure);
        printf("Minutes: ");
        scanf("%d", &p.SonRv.minute);
        break;
    case 2:
        printf("Nom : ");
        scanf("%s", p.nom);
        printf("Prenom : ");
        scanf("%s", p.prenom);
        printf("age : ");
        scanf("%d", &p.age);
        printf("sexe(M/F) : ");
        scanf("%s", p.sexe);
        printf("-----Rendez-vous-----\n");
        printf("Jours : ");
        scanf("%s", p.SonRv.jours);
        printf("Heure : ");

```

```

        scanf("%d", &p.SonRv.heure);
        printf("Minutes: ");
        scanf("%d", &p.SonRv.minute);
        break;
    default :
        printf("Vous avez deux parametre à modifier il vous faut choisir un \n");
        Modifier();
        break;
    }
    fichier = fopen(tab[num-1], "wb");
    fwrite(&p, sizeof(p), 1, fichier);
    fclose(fichier);
}

```

La fonction d'affichage

Elle aussi il faut lui indiquer les rendez-vous de quel médecin vous souhaitez voir. Elle entre dans ces dossier et ouvre tous les fichiers créés en mode lecture seulement et les affichent grâce à la fonction *show*.

```

void show(patient p, int nbr){
    printf("patient n_%d\n", nbr+1);
    printf("Patient : Nom = %s\t Prenom = %s\t Sexe = %s\t age : %d dans\n",
p.nom, p.prenom, p.sexe, p.age);
    printf("Rendez-vous : %s a %dh%dm\n", p.SonRv.jours, p.SonRv.heure,
p.SonRv.minute);
    printf("-----\n");
}

//ici on appelle la fonction show()
void Affiche(int *nbr_p1, int *nbr_p2, int *nbr_p3){
    int nbr_p;
    char* tab[10];
    char* tab3[10] = {"Medecin3/p1.bin", "Medecin3/p2.bin",
"Medecin3/p3.bin", "Medecin3/p4.bin", "Medecin3/p5.bin", "Medecin3/p6.bin", "Medecin3/p7.bin",
"Medecin3/p8.bin", "Medecin3/p9.bin", "Medecin3/p10.bin"};
    char* tab2[10] = {"Medecin2/p1.bin", "Medecin2/p2.bin",
"Medecin2/p3.bin", "Medecin2/p4.bin", "Medecin2/p5.bin", "Medecin2/p6.bin", "Medecin2/p7.bin",
"Medecin2/p8.bin", "Medecin2/p9.bin", "Medecin2/p10.bin"};
    char* tab1[10] = {"Medecin1/p1.bin", "Medecin1/p2.bin",
"Medecin1/p3.bin", "Medecin1/p4.bin", "Medecin1/p5.bin", "Medecin1/p6.bin", "Medecin1/p7.bin",
"Medecin1/p8.bin", "Medecin1/p9.bin", "Medecin1/p10.bin"};
    int i, num, choix;
    printf("1. Medecin 1      2. Medecin 2   3. Medecin 3\n");
    scanf("%d", &choix);
    system("cls");
    switch(choix){

```

```

        case 1:
            for(i=0; i<10;i++)
                tab[i] = tab1[i];
            nbr_p = *nbr_p1;
            break;
        case 2:
            for(i=0; i<10;i++)
                tab[i] = tab2[i];
            nbr_p = *nbr_p2;
            break;
        case 3:
            for(i=0; i<10;i++)
                tab[i] = tab3[i];
            nbr_p = *nbr_p3;
            break;
        default :
            printf("Vous devez indiquer avec quel medecin \n");
            main();
            break;
    }
    patient p;
    FILE *fichier = NULL;
    for(i=0; i<10; i++){
        fichier = fopen(tab[i], "rb");
        if(fichier != NULL){
            fread(&p, sizeof(p), 1, fichier);
            show(p, i);
        }
        fclose(fichier);
    }
}

```

Mode de fonctionnement

Le fonctionnement est le suivant :

- ✓ Les patients et les rendez-vous seront des structures.
- ✓ Pour enregistrer les rendez-vous créés, on les stocke dans des fichiers binaires.
- ✓ Chaque médecin de la clinique disposera d'un dossier avant toute utilisation de l'application.
- ✓ Pour chaque médecin, les fichiers binaires contenant ces rendez-vous seront enregistrés dans son dossier.
- ✓ Ces fichiers pourront être récupérer pour des éventuelles modifications ou suppression.

Démonstration

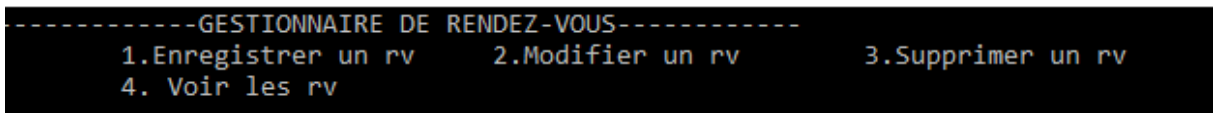
Pour le test de l'application, Nous considérons trois médecins.

Création de dossier :



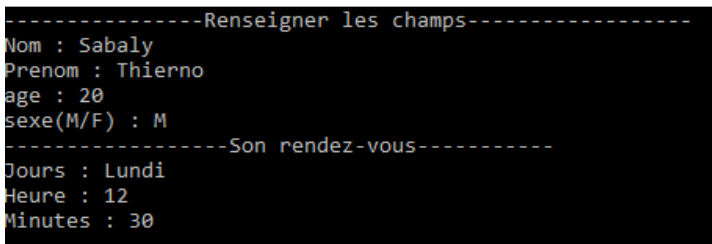
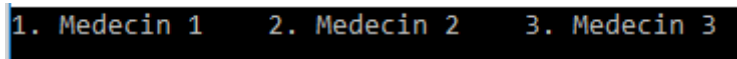
Ces dossiers sont placés au même endroit que l'exécutable. Et il faut les créer manuellement.

Menu

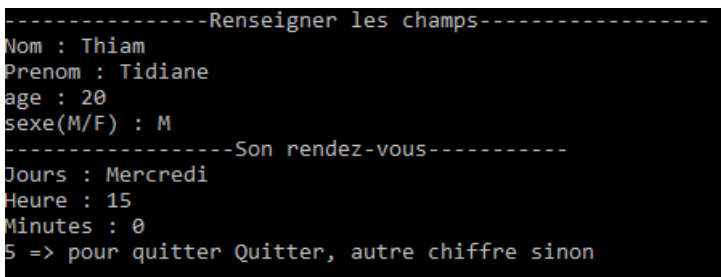


Création de rendez-vous

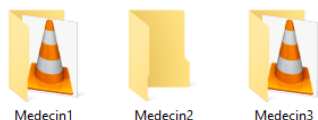
Nous le ferons pour deux médecins (1 et 3), afin de voir le partage.



Le rendez-vous pour le médecin 1.

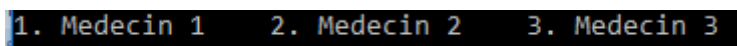


Le rendez-vous du médecin 3.



Les dossiers après cette action :

Affichage



Toujours on indique le médecin

```

patient n_1
Patient : Nom = Sabaly   Prenom = Thierno       Sexe = M       age : 20ans
Rendez-vous : Lundi a 12h30mn
-----
5 => pour quitter Quitter, autre chiffre sinon

```

médecin1

```

patient n_1
Patient : Nom = Thiam    Prenom = Tidiane       Sexe = M       age : 20ans
Rendez-vous : Mercredi a 15h0mn
-----
5 => pour quitter Quitter, autre chiffre sinon

```

médecin3

Modification

```

1. Medecin 1      2. Medecin 2      3. Medecin 3

```

```

Indiquer le numero du patient : 1
(1) Modifier le rv
(2) Remplacer le client
1
Jours : Jeudi
Donner l'heure : 16
Minutes: 0
5 => pour quitter Quitter, autre chiffre sinon

```

On modifie ici, le rendez-vous du patient pour le médecin 1

Affichage :

```

patient n_1
Patient : Nom = Sabaly   Prenom = Thierno       Sexe = M       age : 20ans
Rendez-vous : Jeudi a 16h0mn
-----
5 => pour quitter Quitter, autre chiffre sinon

```

Suppression :

```

1. Medecin 1      2. Medecin 2      3. Medecin 3

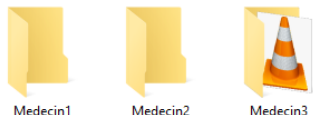
```

Nous allons supprimer le rendez-vous du médecin 1.

```

Vous voulez supprimer un rv : (1) oui ou (2) non : 1
Donner le numero du patient : 1
5 => pour quitter Quitter, autre chiffre sinon

```



Résultat : Medecin1

Cette application en console nous prouve l'efficacité des fichiers dans la sauvegarde et la restitution des rendez-vous. Les fichiers sont accessibles tout le temps. Ce mode de fonctionnement est le modèle de base de la réalisation en fenêtre de l'application. Les codes sources sont très différents, mais en arrière plan c'est le même système.

MODE GRAPHIQUE

La présentation de l'application sera plus jolie à voir, l'application sera attirante et plus communicante. Le code source est plus volumineux avec plus de fonctions.

Avant les fonctions je vous présente le fichier d'entête. Il contient l'ensemble des énumérations faites et des structures créées.

```
#ifndef _MACONS_
#define _MACONS_
#define BOUT_LARG 40
#define BOUT_LONG 150
#define BOUT_CHIFF_LARG 20
#define BOUT_CHIFF_LONG 70

//un rendez-vous est une structure contenant le jours, l'heure.
typedef struct Rendez_Vous Rendez_Vous;
struct Rendez_Vous{
    int jours;
    int mois;
    int annee;
    int heure;
    int minutes;
};
//un patient est une atstructure contenant un rendez-vous
typedef struct Patient Patient;
struct Patient{
    char prenom[50];
    char nom[50];
    char age[50];
    char traitement[50];
    char sexe;
    Rendez_Vous rv;
};

/*
    Dans le cadre de la gestion du temps, nous avons rassemblé tous les mois en
une
    enumeration.
*/
typedef enum mois mois;
enum mois{
    JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET, AOUT, SEPTEMBRE,
    OCTOBRE, NOVEMBRE, DECEMBRE
```

```

};
//Les actions possible de l'utilisateur sont notées ici dans une enumeration.
typedef enum Lieu Lieu;
enum Lieu
{
    MEDECIN, SECRETAIRE, ACCUEILLE, AIDE
};

typedef enum bouton bouton;
enum bouton{
    ACC, MED, SEC, AID,
    //Les boutons d'accueil
    ACC_BOUT_0, ACC_BOUT_1, ACC_BOUT_2, NULL_PART
};

//Les énumérations liées au secteur Medecin
typedef enum Bout_Acc Bout_Acc;
enum Bout_Acc{
    ESP_MEDACC_BOUT_1, ESP_MEDACC_BOUT_2, ESP_MEDACC_BOUT_3,
    DECONNECTE, ESP_MEDACC_BOUT_5, NONE
};

typedef enum Bout_Esp_Med Bout_Esp_Med;
enum Bout_Esp_Med{
    ESP_MED_BOUT_1, ESP_MED_BOUT_2, ESP_MED_BOUT_3,
    ESP_MED_BOUT_4, ESP_MED_BOUT_5, AUCUN
};

typedef enum Bout_Creer_Compte Bout_Creer_Compte;
enum Bout_Creer_Compte{
    ESP_MED_BOUT_CC1, ESP_MED_BOUT_CC2, ESP_MED_BOUT_CC3,
    ESP_MED_BOUT_CC4, ESP_MED_BOUT_CC5, ANN, NUL
};

typedef enum Bout_ID Bout_ID;
enum Bout_ID{
    ESP_MED_BOUT_ID1, ESP_MED_BOUT_ID2, ESP_MED_BOUT_ID3, NO_ID,
    RIEN
};

typedef enum Bout_MesDispo Bout_MesDispo;
enum Bout_MesDispo{
    NEW_DISPO, DELETE_DISPO, EDIT_DISPO, ANNUL_EDITING, NOTHING
};

```

```

typedef enum Dispo_Creation Dispo_Creation;
enum Dispo_Creation{
    DC_JOURS, DC_MOIS, DC_ANNEE, DC_HEURE, DC_MINUTE, DC_VALIDE,
    DC_APRES_1, DC_APRES_2, DC_APRES_3, DC_APRES_4, DC_APRES_5,
    DC_AVANT_1, DC_AVANT_2, DC_AVANT_3, DC_AVANT_4, DC_AVANT_5,
    DC_QUIT, DC_RIEN
};
//les enums lies au secteur secretaire
typedef enum Bout_Esp_Sec Bout_Esp_Sec;
enum Bout_Esp_Sec{
    ESP_SEC_BOUT_CONNECTE, ESP_SEC_BOUT_DECONNECTE,
    ESP_SEC_BOUT_CREERCOMPTE, ESP_SEC_BOUT_CREERRV, ESP_SEC_AUCUN
};

typedef enum Bout_Connect_Sec Bout_Connect_Sec;
enum Bout_Connect_Sec{
    BOUT_CONNECT_NOM, BOUT_CONNECT_MDP, BOUT_CONNECT_OK,
    BOUT_CONNECT_QUIT, BOUT_CONNECT_RIEN
};

typedef enum Bout_Compte_Sec Bout_Compte_Sec;
enum Bout_Compte_Sec{
    BOUT_COMPTE_NOM, BOUT_COMPTE_MDP, BOUT_COMPTE_CMDP,
    BOUT_COMPTE_VALIDE, BOUT_COMPTE_QUIT, BOUT_COMPTE_RIEN
};

typedef enum Bout_CreerPatient_Sec Bout_CreerPatient_Sec;
enum Bout_CreerPatient_Sec
{
    NOM_MED, CP_SUITE, NOM_PATIENT, PRENOM_PATIENT, SEXE_PATIENT,
    AGE_PATIENT, TRAITEMENT, FEMME, HOMME, RV_SUIVANT, RV_PRECEDENT,
    CP_BOUT_VALIDE, BOUT_CP_QUITTER, AUCUN_BOUT_CP
};

typedef enum souri souri;
enum souri{
    SURVOL, CLIC, AUTRE
};

typedef struct identifiant identifiant;
struct identifiant{
    char Nom_Utilisateur[50];
    char Mot_De_Pass[50];
};

```


#endif

Mode de fonctionnement

Le système en arrière plan est le même que la console que la console. Ici il est plus complexe. Pour cela il faut :

- créer un dossier, au même endroit que l'exécutable, appelé "**Medecins**"
- Dans ce dernier, on crée un dossier pour chaque médecin de la clinique.
- Dans le dossier de chaque médecin, on crée deux sous-dossiers. Un appelé "**disponibilités**" et l'autre "**RendezVous**".
- De même, on crée un dossier "Secrétaires" et chaque secrétaire on lui crée un sous-dossier dedans en sont nom.

Ceci doit être fait avant toute utilisation de l'application.

Ainsi le mode de fonctionnement est le suivant :

- ✚ Les disponibilités seront des fichiers binaires
- ✚ Des fichiers qui se trouveront dans le dossier du médecin qui les aura indiqués. Ils seront plus précisément dans le sous-dossier "**disponibilités**".
- ✚ Le nombre de disponibilité sera compté et stocké dans un fichier binaire stocké dans le dossier du médecin correspondant.
- ✚ Les rendez-vous seront des fichiers binaires.
- ✚ Ils seront eux stockés dans le sous-dossier "**Rendezvous**" du médecin correspondant.
- ✚ Le nombre de rendez-vous sera aussi compté et stocké dans un fichier binaire. Ce dernier sera conservé dans le dossier du médecin correspondant.

Présentation des fonctions

Ce sont des fonctions très complexes. Alors on s'est dit que mieux vaut expliquer les fonctions permettant de contrôler les déplacements de la souris, les clics et la saisie. En plus nous avons déjà eu un aperçu des fonctions de la SDL que nous avons fait appel, on pourra ensuite exposer les codes source des autres fonctions dont on expliquera le fonctionnement.

Les déplacements de la souris : La souris est facile à repérer avec la SDL. Chaque déplacement de la souris est un événement de type `SDL_MOUSEMOTION` qui nous renvoie l'abscisse de la souris (`event.motion.x`) et son ordonnée (`event.motion.y`).

```
while(continuer){
    while(SDL_PollEvent(&event)){
        switch(event.type){
            case SDL_MOUSEMOTION :
                break ;
        }
    }
}
```

Une fois les composantes de la souris obtenues, il devient facile de savoir si elle est ou pas sur un bouton. Sauf que nos boutons sont dans différentes zones de l'application. Certains au niveau de l'accueil, d'autres dans l'espace médecin...

Ce qui donne importance aux énumérations se trouvant dans le fichier d'entête présenté avant. Pour chaque partie de l'application dispose d'un type d'énumération, on lui crée donc deux variables de ce type. Une pour les cas de survole et l'autre pour les clics. Ces variables sont actives que si on est dans la zone qu'elles représentent.

Ces énumérations sont définies comme suit : on numérote les boutons à partir de 1 ou on nomme chaque bouton en majuscule. Chaque valeur représentera un bouton.

On récupère ces valeurs grâce à des fonctions particulières nommées selon la zone pour laquelle on est. Leurs prototypes

```
type fonction (int abscisse_souri, int ordonnee_souri, type Bouton) ;
```

Illustrons cela pour l'espace médecin. L'énumération est :

```
typedef enum Bout_Esp_Med Bout_Esp_Med;
enum Bout_Esp_Med{
ESP_MED_BOUT_1, ESP_MED_BOUT_2, ESP_MED_BOUT_3, ESP_MED_BOUT_4, ESP_MED_BOUT_5,
AUCUN
};
```

La fonction :

```
Bout_Esp_Med Bout_Appuye_Med(int abscisse, int ordonnee, Bout_Esp_Med Bout_Appuye){
if(abscisse > 300 && abscisse < 300 + BOUT_LONG && ordonnee > 40 && ordonnee < 40 +
BOUT_LARG){
    return ESP_MED_BOUT_1;
}else if(abscisse > 450 && abscisse < 450 +BOUT_LONG && ordonnee > 40 && ordonnee < 40 +
BOUT_LARG){
    return ESP_MED_BOUT_2;
}else if(abscisse > 0 && abscisse < 100 && ordonnee > 460 && ordonnee < 560){
    return ESP_MED_BOUT_5;
}else
    return Bout_Appuye;
}
```

Déclaration de la variables de survole :

```
Bout_Esp_Med Bout_Med_Survole = AUCUN;
```

Récupération des valeurs :

```
while(continuer){
    while(SDL_PollEvent(&event)){
        case SDL_MOUSEMOTION :
            Bout_Med_Survole = Bout_Appuye_Med(event.motion,
event.motion.y, Bout_Med_Survole)
        }
    }
}
```

Les clics de la souris : De la même manière que les survole, on récupère l'endroit où un clic est produit. Et pouvoir stocker cela dans une variable. Tous ce qui change c'est le type d'événement, cette l'événement est SDL_MOUSEBUTTONDOWN.

Un exemple : Déclaration

```
Bout_Esp_Med Bout_Med_Appuye = AUCUN;
```

```
while(continuer){
    while(SDL_PollEvent(&event)){
        case SDL_MOUSEBUTTONDOWN:
            Bout_Med_App = Bout_Appuye_Med(event.button.x, event.button.y,
Bout_Med_App)
        }
    }
}
```

Leurs utilisations : dans la fonction correspondante, on effectue un switch, selon les cas on effectue l'action qu'on souhaite. Exemple :

```
switch(Bout_Med_App){
case ESP_MED_BOUT_1:
//ouvre disponibilités si le médecin connecté
break;
case ESP_MED_BOUT_2:
//ouvre les rv si connecté
break;
case ESP_MED_BOUT_5:
//On se rend à l'accueil
break;
default:
//rien
break;
}
```

Les événements du clavier : Il s'agit de l'événement SDL_KEYDOWN. Lorsqu'on appui sur une touche. Pour un tel événement, nous avons mise en place deux fonctions qui permettent de saisir. La première permet de saisir des lettres. Là voici

```
char* Saisi_Chaine(SDL_KeyboardEvent bouton){
    char* aretourner;
    switch(bouton.keysym.sym){
        case SDLK_a:
            strcpy(aretourner, "a");
            break;
        case SDLK_b:
            strcpy(aretourner, "b");
```

```
        break;
case SDLK_c:
    strcpy(aretourner, "c");
    break;
case SDLK_d:
    strcpy(aretourner, "d");
    break;
case SDLK_e:
    strcpy(aretourner, "e");
    break;
case SDLK_f:
    strcpy(aretourner, "f");
    break;
case SDLK_g:
    strcpy(aretourner, "g");
    break;
case SDLK_h:
    strcpy(aretourner, "h");
    break;
case SDLK_i:
    strcpy(aretourner, "i");
    break;
case SDLK_j:
    strcpy(aretourner, "j");
    break;
case SDLK_k:
    strcpy(aretourner, "k");
    break;
case SDLK_l:
    strcpy(aretourner, "l");
    break;
case SDLK_SEMICOLON:
    strcpy(aretourner, "m");
    break;
case SDLK_n:
    strcpy(aretourner, "n");
    break;
case SDLK_o:
    strcpy(aretourner, "o");
    break;
case SDLK_p:
    strcpy(aretourner, "p");
    break;
case SDLK_q:
    strcpy(aretourner, "a");
```

```

        break;
    case SDLK_r:
        strcpy(aretourner, "r");
        break;
    case SDLK_s:
        strcpy(aretourner, "s");
        break;
    case SDLK_t:
        strcpy(aretourner, "t");
        break;
    case SDLK_u:
        strcpy(aretourner, "u");
        break;
    case SDLK_v:
        strcpy(aretourner, "v");
        break;
    case SDLK_w:
        strcpy(aretourner, "z");
        break;
    case SDLK_x:
        strcpy(aretourner, "x");
        break;
    case SDLK_y:
        strcpy(aretourner, "y");
        break;
    case SDLK_z:
        strcpy(aretourner, "w");
        break;
    case SDLK_SPACE:
        strcpy(aretourner, " ");
        break;
    default:
        strcpy(aretourner, "");
        break;
}

return aretourner;
}

```

Et l'autre des chiffres

```

char* Saisi_Chiffre(SDL_KeyboardEvent bouton){
    char* aretourner;
    switch(bouton.keysym.sym){
        case SDLK_KP0:
            strcpy(aretourner, "0");

```

```

        break;
    case SDLK_KP1:
        strcpy(aretourner, "1");
        break;
    case SDLK_KP2:
        strcpy(aretourner, "2");
        break;
    case SDLK_KP3:
        strcpy(aretourner, "3");
        break;
    case SDLK_KP4:
        strcpy(aretourner, "4");
        break;
    case SDLK_KP5:
        strcpy(aretourner, "5");
        break;
    case SDLK_KP6:
        strcpy(aretourner, "6");
        break;
    case SDLK_KP7:
        strcpy(aretourner, "7");
        break;
    case SDLK_KP8:
        strcpy(aretourner, "8");
        break;
    case SDLK_KP9:
        strcpy(aretourner, "9");
        break;
    default:
        strcpy(aretourner, "");
        break;
}

return aretourner;
}

```

On appelle ces fonctions que quand on veut saisir. C'est-à-dire, lorsqu'une touche du clavier est enfoncée on vérifie si un champ de saisi est activé. Si oui, on écrit dessus. Sinon on ne fait rien. Ces fonctions ci-dessus ne font que renvoyer les touches enfoncées.

Les fonctions de création de compte : Il y en a deux une pour les secrétaires et une autre pour les médecins. Ce sont les deux autorisés à utiliser l'application, ils doivent s'assurer d'avoir un dossier en leur nom. Les fonctions de création de compte présentent trois champs qui sont : le nom, le mot de passe et une confirmation du mot de passe.

Elles regardent d'abord si votre dossier existe et que les mots de passe entrés sont identiques. Si oui, elles créent un fichier binaire stockant votre nom et mot de passe.

Prototypes :

```
void Creation_Compte_Med(SDL_Surface* fenetre, identifiant id, Bout_Creer_Compte CC_App, char masqueur1[50], char masq_check[50], char masqueur2[50], char confirmation[50]) ;
```

```
void Creation_Compte_Sec(SDL_Surface* fenetre, Bout_Compte_Sec Bout_Appuye, identifiant id, char masqueur1[50], char masqueur2[50], char verifie[50]) ;
```

Les fonctions de connexion : là aussi nous en avons deux. Pour se connecter, on renseigne deux champs le nom et le mot de passe. Une fois validé, on ouvre dans le dossier portant ce nom, le fichier binaire stockant le nom et le mot de passe. Si ce fichier existe, on récupère les informations s'y trouvant pour comparer avec celles saisies. Sinon, c'est parce que le dossier n'existe pas.

Prototypes des deux fonctions :

Médecins :

```
void connecte(SDL_Surface* fenetre, identifiant* id, Bout_ID ID_App, char masqueur1[50], int *connected, char confirmation[50], char Med_Conn_Nom[50]) ;
```

Secrétaire :

```
void Connexion_Sec(SDL_Surface* fenetre, Bout_Connect_Sec Bout_Appuye, identifiant id, char masqueur1[50], int *connected) ;
```

La fonction pour créer des disponibilités : Elle n'existe que pour les médecins. A travers elle le médecin peut indiquer la date (jours-mois-année) et l'heure à laquelle il est libre pour un rendez-vous. Une fois les champs renseignés, elle ouvre dans le dossier du médecin correspondant le sous-dossier **disponibilités**. Dans lequel, il crée un fichier binaire stockant la date et l'heure.

En plus, si une date est déjà prise le fichier n'est pas créé car déjà existant.

Prototype :

```
void Entrer_Dispo(SDL_Surface* fenetre, char dossier[50], Rendez_Vous Cr_Rv, Dispo_Creation DC_App, int *nombre_Dispo, char Dossier_En_Cours[50]) ;
```

La fonction pour créer un rendez-vous : Cette partie est dédiée à la secrétaire. On renseigne d'abord le nom du médecin (comme dans la console d'ailleurs). Une fois le nom renseigné, la fonction ouvre le dossier médecin à la recherche d'un sous-dossier portant ce nom. Dans le cas où il en trouve, il donne la main à la secrétaire pour renseigner le nom, prénom, sexe et l'âge. En même temps, elle ouvre le dossier contenant les disponibilités de ce médecin pour permettre à la secrétaire de faire un choix. Lorsqu'elle valide, on crée un fichier binaire dans le sous-dossier **RendezVous** du médecin correspondant.

Prototype :

```
void Creer_Patient(SDL_Surface *fenetre, Bout_CreerPatient_Sec Bout_Appuye, char
Medecin[50], Patient patient, int *compteur,int *Nbr_Rv, int *Nbr_Dispo) ;
```

Les fonctions d'affichages : Elle permette aux médecins de voir leurs disponibilités et leurs rendez-vous. Elles sont faciles à concevoir car elles ne font que lire les fichiers se trouvant sous-dossiers **disponibilités** et **RendezVous** du médecin connecté.

Prototypes :

```
void BlitRv(SDL_Surface* fenetre, int Nbr_Rv, char Dossier[50]) ;
```

```
void BlitDispo(SDL_Surface* fenetre, char Dossier[50], int nombre_Dispo) ;
```

Les fonctions accueil médecin : L'espace médecin une page d'accueil en plus de la page de bienvenue. C'est elle qui contient les boutons permettant de se connecter, de créer un compte et de se déconnecter. Elle fait donc appel à toutes les autres fonctions selon le bouton appuyé.

Prototype :

```
void Accueille_Medecin(SDL_Surface* fenetre, identifiant ID, Bout_Acc AccMed_Surv, Bout_Acc
AccMed_App, Bout_ID ID_App, Bout_Creer_Compte CC_App, char masqueur1[50], char
masq_check[50], char masqueur2[50], int *connected, char confirmation[50],char
Med_Conn_Nom[50]) ;
```

Fonctions pour l'espace médecin : La fonction principale gérant l'espace médecin est la fonction **Espace_Medecin** dont le prototype est :

```
void Espace_Medecin(SDL_Surface* fenetre, Bout_Esp_Med SURV, Bout_Esp_Med Bout_Appuye,
identifiant mes_iden, Bout_Acc AccMed_Surv,Bout_Acc AccMed_App, Bout_ID ID_App,
Bout_Creer_Compte CC_App, Bout_MesDispo Surv_MesDispo, Bout_MesDispo
App_MesDispo,Dispo_Creation DC_App, char masqueur1[50], char masq_check[50], char
masqueur2[50], int *connected, char confirmation[50],char Med_Conn_Nom[50], char
Dossier_En_Cours[50], Rendez_Vous Cr_Rv, int *Nbr_Dispo,int Nbr_Rv) ;
```

C'est une zone très importante, active. Elle récupère le nombre de disponibilité et le nombre de rendez-vous

```
FILE *fichier_nbrDispo = NULL, *fichier = NULL, *file = NULL;
char le_dossier1[50] = "", le_dossier2[50] = "", le_dossier3[50] = "";
int Pas_de_dispo = 0;
strcpy(le_dossier1, Dossier_En_Cours);
strcat(le_dossier1, "/disponibilites/dispo1.bin");

strcpy(le_dossier2, Dossier_En_Cours);
strcat(le_dossier2, "/nbrDispo.bin");
```



```

strcpy(le_dossier3, Dossier_En_Cours);
strcat(le_dossier3, "/nbr_rv.bin");

if(*connected){
    fichier = fopen(le_dossier1, "rb");
    if(fichier == NULL){
        remove(le_dossier2);
        *Nbr_Dispo = 0;
        fclose(fichier);
    }else{
        fclose(fichier);
    }
    fichier_nbrDispo = fopen(le_dossier2, "rb");
    if(fichier_nbrDispo == NULL){
        fclose(fichier_nbrDispo);
        fichier_nbrDispo = fopen(le_dossier2, "wb");
        fwrite(Nbr_Dispo, sizeof(int), 1, fichier_nbrDispo);
        fclose(fichier_nbrDispo);
    }else{
        fread(Nbr_Dispo, sizeof(int), 1, fichier_nbrDispo);
        fclose(fichier_nbrDispo);
    }
    file = fopen(le_dossier3, "rb");
    if(file != NULL){
        fread(&Nbr_Rv, sizeof(int), 1, file);
        fclose(file);
    }else{
        Nbr_Rv = 0;
        fclose(file);
    }
}
}

```

Elle est la porte d'entrée dans la zone médecin. Elle oriente vers l'accueil médecin, les disponibilités et les rendez-vous (ces dernières font appel à toutes les autres fonctions concernant le médecin). Comme on peut le voir ici.

```

switch(Bout_Appuye){
case ESP_MED_BOUT_1:
    if(*connected){
        Mes_Dispo(fenetre, Surv_MesDispo, App_MesDispo, DC_App, Dossier_En_Cours, Cr_Rv,
        Nbr_Dispo);
    }else{
        Pas_Connecter(fenetre);
    }
    break;
case ESP_MED_BOUT_2:
    if(*connected){
        Mes_Rv(fenetre, Nbr_Rv, Dossier_En_Cours);
    }else
        Pas_Connecter(fenetre);
    break;
}

```

```

case ESP_MED_BOUT_5:
    Accueille_Medecin(fenetre, mes_iden, AccMed_Surv, AccMed_App, ID_App, CC_App,
    masqueur1, masq_check, masqueur2, connected, confirmation, Med_Conn_Nom);
    break;
default:
    Message_Welcome(fenetre);
    break;
}

```

Si un médecin pas connecté appui sur ces bouton, On fait appel à la fonction pas connecté pour alerté ce dernier qu'il doit d'abord se connecter.

L'espace secrétaire : la réalisation de cette partie à trouver que nous avons beaucoup plus d'expérience. En le code source ici est très explicite.

- Nous avons créé une zone d'alerte
- Puis un texte descriptif
- Ensuite nous avons récupéré le nombre de rendez-vous et de disponibilité
- Et enfin, nous avons gérer les survole et clique des boutons dans une fonction appart appelée **Les_Boutons**.

Le code source

```

void Espace_Secretaire(SDL_Surface* fenetre, Bout_Esp_Sec BoutSurv_Sec, Bout_Esp_Sec
BoutApp_Sec, Bout_Connect_Sec BoutApp_Connexion_Sec, Bout_Compte_Sec
BoutApp_Compte_Sec, identifiant mes_id, char masqueur1[50], char masqueur2[50], char
verifie[50], int *connected, Bout_CreerPatient_Sec BoutApp_CP, char Med_Pour_Rv[50], Patient
Le_Patient, int *Nbr_Rv, int *compteur, int *Nbr_Dispo){
    Zone_Alerte(fenetre); Texte_Accueille(fenetre);
    //récupération nombre de rv et de dispos
    char le_dossier1[50] = "Medecins/", le_dossier2[50] = "Medecins/", dossier1[50] = "Medecins/";
    strcat(le_dossier1, Med_Pour_Rv);
    strcat(le_dossier1, "/RendezVous/rv1.bin");
    strcat(le_dossier2, Med_Pour_Rv);
    strcat(le_dossier2, "/nbr_rv.bin");
    strcat(dossier1, Med_Pour_Rv);
    strcat(dossier1, "/nbrDispo.bin");
    FILE *fichier = NULL, *Nbr_Rv_File = NULL, *file = NULL;

    if(*connected && strcmp(Med_Pour_Rv, "") != 0){
        fichier = fopen(le_dossier1, "rb");
        if(fichier == NULL){
            remove(le_dossier2);
            *Nbr_Rv = 0;
            fclose(fichier);
        }else
            fclose(fichier);
    }
}

```

```

Nbr_Rv_File = fopen(le_dossier2, "rb");
if(Nbr_Rv_File == NULL){
    fclose(Nbr_Rv_File);
    Nbr_Rv_File = fopen(le_dossier2, "wb");
    fwrite(Nbr_Rv, sizeof(int), 1, Nbr_Rv_File);
    fclose(Nbr_Rv_File);
}else{
    fread(Nbr_Rv, sizeof(int), 1, Nbr_Rv_File);
    fclose(Nbr_Rv_File);
}

file = fopen(dossier1, "rb");
if(file != NULL){
    fread(Nbr_Dispo, sizeof(int), 1, file);
    fclose(file);
}else
    fclose(file);
}

Les_Bouttons(fenetre, BoutSurv_Sec, BoutApp_Sec, BoutApp_Connexion_Sec,
BoutApp_Compte_Sec, mes_id, masqueur1, masqueur2,verifie, connected, BoutApp_CP,
Med_Pour_Rv, Le_Patient, Nbr_Rv, compteur, Nbr_Dispo);
}

```

La fonction main : C'est ici qu'on gère les événements.

- Nous avons déclaré les différentes énumérations pour la gestion des clics et survole

```

souri La_Souri;
boutton Appui_Sur_Bout = NULL_PART;
boutton Zone_d_avant = NULL_PART;
/*Bouttons appuye dans la zone medecin*/

Bout_Esp_Med Bout_Med_Appuye = AUCUN;
Bout_Esp_Med Bout_Med_AppAvant = AUCUN;
Bout_Esp_Med Bout_Med_Survele = AUCUN;

//Ce sont les variables permettant de stocker les boutons appuyés dans l'accueil des
medecins
Bout_Acc Bout_Surv_AccMed = NONE;
Bout_Acc Bout_App_AccMed = NONE;
Bout_Acc Bout_AppPrec_AccMed = NONE;

//Ici les boutons appuyés lors de l'identification (connexion)
Bout_ID Bout_For_Iden = RIEN;
Bout_ID Bout_For_IdenBefore = RIEN;

//ici les boutons appuyé lors de la creation de compte

```

```

    Bout_Creer_Compte Bout_For_CC = NUL;
    Bout_Creer_Compte Bout_For_CCBefore = NUL;
//zone disponibilités
    Bout_MesDispo Bout_NouvelDispo_Surv = NOTHING;
    Bout_MesDispo Bout_NouvelDispo_App = NOTHING;
    Bout_MesDispo Bout_NouvelDispo_AppBefore = NOTHING;

    Dispo_Creation Bout_For_DC = DC_RIEN;

    /*Boutton appuye dans le zone secretaire.*/
    Bout_Esp_Sec Bout_Sec_Appuye = ESP_SEC_AUCUN;
    Bout_Esp_Sec Bout_Sec_AppAvant = ESP_SEC_AUCUN;
    Bout_Esp_Sec Bout_Sec_Survole = ESP_SEC_AUCUN;

    Bout_Connect_Sec Bout_ForConnexion_Sec = BOUT_CONNECT_RIEN;

    Bout_Compte_Sec Bout_ForCC_Sec = BOUT_COMPTE_RIEN;

    Bout_CreerPatient_Sec Bout_ForCP_Sec = AUCUN_BOUT_CP;

    bouton Pass_Sur_Bout = NULL_PART;
    Lieu Zone = ACCUEILLE;

```

- On récupère leurs valeur lors d'un déplacement de la souris

```

case SDL_MOUSEMOTION:
    Pass_Sur_Bout = Mouse_Zone(event.motion.x, event.motion.y, Fenetre_Long,
Fenetre_Larg, Pass_Sur_Bout);
//Ici commence la gestion des boutons survoles dans la zone medecin
    Bout_Med_Survole = Bout_survole_Med(event.motion.x, event.motion.y,
Fenetre_Long, Fenetre_Larg, Bout_Med_Appuye);
    Bout_Surv_AccMed = Bouton_AccSurv(event.motion.x, event.motion.y,
Fenetre_Long, Fenetre_Larg, Bout_Surv_AccMed);
    Bout_NouvelDispo_Surv = Bouton_MesDispos_Surv(event.motion.x,
event.motion.y, Fenetre_Long, Fenetre_Larg, Bout_NouvelDispo_Surv);
//Fin de la gestion des survole dans la zone medecin
//Ici commence la gestion des boutons survoles dans la zone secretaire
    Bout_Sec_Survole = Bout_Survole_Sec(event.motion.x, event.motion.y, Fenetre_Long,
Fenetre_Larg);
break;

```

Les fonctions permettant de récupérer ces positions ont déjà été présenté au haut.

- On fait la même chose pour les clics.
- Pour éviter les collisions. C'est-à-dire par exemple, je suis dans la zone secrétaire et que je clic sur un bouton se trouvant dans la zone médecin. On doit normalement empêcher l'exécution d'un tel clic. Pour se faire, la solution c'est de remettre les variables des autres zones à leur valeur nulle. Voici un exemple de code pour éviter la collision entre espace secrétaire, espace médecin, accueil et à propos

```

//gestion de collision des zones
if(Zone == ACCUEILLE){
    Bout_Med_Appuye = AUCUN;
}else if(Zone == SECRETAIRE){

```

```

Bout_Med_Appuye = AUCUN;
}else if(Zone == AIDE){
Bout_Med_Appuye = AUCUN;
Bout_Sec_Appuye = ESP_SEC_AUCUN;
}else if(Zone == MEDECIN){
Bout_Sec_Appuye = ESP_SEC_AUCUN;
}

```

C'est la même stratégie pour les autres cas. Par exemple, dans l'espace accueil médecin éviter les collisions entre connexion et création de compte. Cette gestion se fait lorsqu'un clic intervient.

- La saisie : C'est dans la fonction principale que l'on construit nos mots et envoie aux autres fonctions. Ces mots se construisent selon la zone de saisie activée. De ce fait, nous avons créé une variable de type identifiant (le cas où on se trouve dans une des fonctions de connexion), une variable de type Rendez_Vous (pour lorsqu'on crée des rendez-vous)...

Par exemple :

```

Case SDL_KEYDOWN :
if(Bout_Sec_Appuye == ESP_SEC_BOUT_CONNECTE && Bout_ForConnexion_Sec ==
BOUT_CONNECT_NOM){
    strcat(Sec_Iden.Nom_Utilisateur, Saisi_Chaine(event.key));
    strcat(Sec_Iden.Nom_Utilisateur, Saisi_Chiffre(event.key));
}
break ;

```

ESP_SEC_BOUT_CONNECTER vaut dire que nous sommes déjà dans l'espace secrétaire et que le bouton connecter est appuyé. En plus, BOUT_CONNECT_NOM vaut que le champ "nom est actif". Donc on construit le nom de la secrétaire. Cette construction se fait par concaténation lettre par lettre des boutons tapés. De la même manière pour les autres

- Cette fonction **main** quitte le programme dès qu'on appuie sur le bouton quitter ou escape du clavier

```

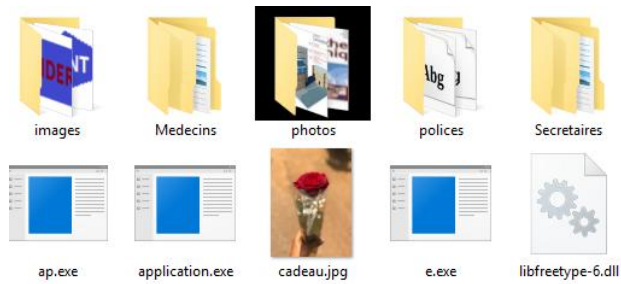
case SDL_QUIT:
    continuer = 0; // Quite le programme
    break;
case SDL_KEYDOWN :
    switch(event.key.keysym.sym){
        case SDLK_ESCAPE:
            continuer = 0;
            break;
    }

```

- Il y'a tellement d'autres fonctions : comme celle qui anime les photos, celle qui affiche la date, et récemment nous avons créé celles qui ordonnent les disponibilités et les rendez-vous. Toutes ces fonctions sont appelées dans la fonction main.

Démonstration

Fin la théorie, voyons tous cela en pratique. Commençons par avoir un dossier "Medecins".



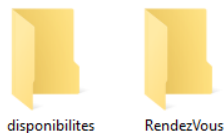
le voici avec l'exécutable.

Créons dedans un dossier du nom de "sall".



voilà

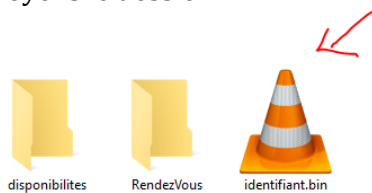
Dans le dossier "sall", créer les deux sous dossiers "disponibilités" et "RendezVous"



Sall est maintenant prêt à utiliser l'application.

- Création de compte :
Un compte ici correspond à une activation de dossier. Actuellement, M. Sall a un dossier mais il ne pourra l'utiliser sans créer de compte.
Donc on va lui créer un compte

Voyons le dossier



voici ces identifiant

- Connexion

Connexion : Medecin

Nom

Mot de passe

[Ok](#) [Quitter](#)

Agenda Electronique

Accueil Espace Secrétaire **Espace_Medecin** A propos

Mes Disponibilités Mes Rendez-Vous

Docteur sall est connecte

Après identification, chaque médecin pourra renseigner ses disponibilités en indiquant seulement la date (le jour et l'heure). Le secrétariat se chargera de créer les rendez-vous pour chaque date indiquée. Avant tout cela, il faut d'abord un compte d'utilisateur.

[Connectez-vous](#)
[Créer un compte](#)

[Se deconnecter](#)


Bienvenue Dans l'espace des medecins
Voici quelques elements a savoir :
La creation d'un compte est equivalent a la creation d'un dossier.
Pour que chaque medecin n'accede qu'a son dossier, il doit lui creer un compte d'utilisateur

Accueille Medecin

ICI

DATE
Mon Oct 15 23:34:52 2018

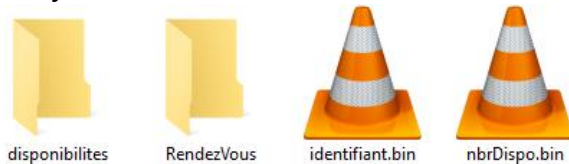
Assurez-vous que votre PC soit toujours a jours, pour la bonne gestion du temps.



une clinique est un centre hospitalier
Souvent specialisee en un domaine bien preci. Il y'a qui sont privees et d'autres sont publiques.

Image de photos de cliniques ou de produits medecins....

Revoyons le dossier



Un fichier pour stocker les disponibilités est créé.

- Désormais, M. Sall a accès à son dossier. Il peut créer ses disponibilités (initialement vide).



Créons en deux.

Indiquez la date et l'heure

jours 15 heure 23

mois 10 minutes 37

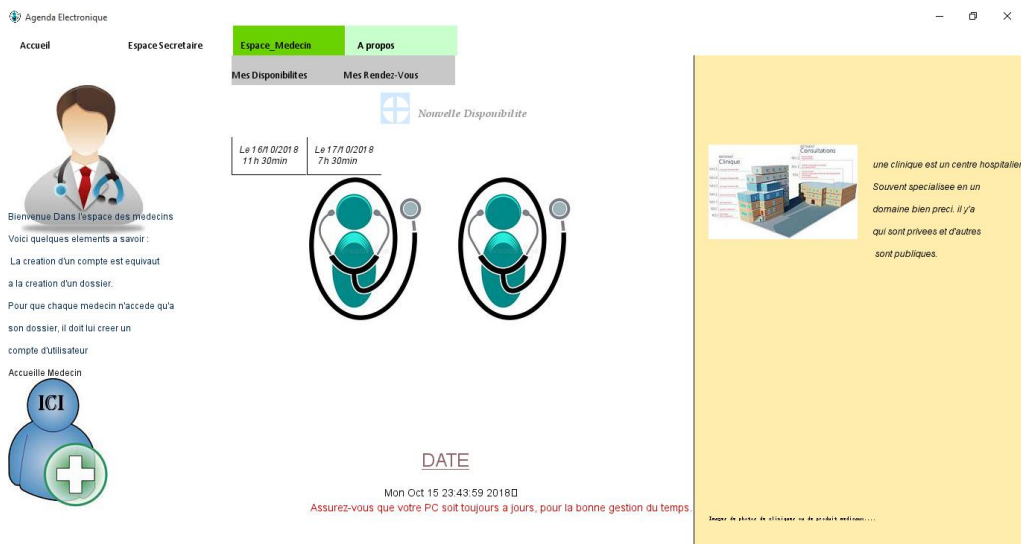
annee 2018

VALIDER

fonction permettant de créer des disponibilités



voilà des disponibilités ont été créées



- Les rendez-vous

C'est au secrétariat que se créent les rendez-vous. On va utiliser un compte que j'ai déjà créé pour remplir le dossier de rendez-vous de M. Sall



Secrétariat.

Connexion

fonction connexion

fonction pour la création de rendez-vous.

Voyons le dossier de sall.



vous.

Un fichier gérant le nombre de Rendez-vous est généré plus le rendez-

