INTRODUCTION GENERALE

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’historique

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 decrivant 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 concernent 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.

Système de Sauvegarde : les fichiers

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 veux 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 celuis 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 créé 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. Example 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\_SetIcone*, comme ça :

SDL\_WM\_SetIcone(‘’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\_BlitSurface. 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 :

Exemple1 : 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’éxecutable. 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 souri, 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 souri 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 touts 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 éffectue une action. Ceci se fait grâce à la fonction *SDL\_WaitEvent()*; sytaxe 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’événment 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\_MOUSEBOUTONUP : 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.tff’’) ;

Un texte est une surface, donc on déclare un pointeur de type SDL\_Surface. On construit cette surface grâce à la fonction *TTF\_RenderTexte\_Blended* (ou TTF\_Rendertexte\_Solid ou TTF\_renderTexte\_Shaded). Nous n’avons travailler qu’avec TTF\_RenderTexte\_Blinded. Exemple TTF\_RendreTexte\_Blinded(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 détaille. Le code ressemble à ça :

time\_t currenttime ;time(&currenttime) ;struct tm \*mytime=localtime(&currentime) ;.

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

Mode console

***Présentation des fonctions***

Pour la réalisation de l’application proprement dite, nous ne nous sommes pas trop vite lancer dans le graphisme. Nous avons préférer 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 Acceuille(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 accueille

int main(){

int nbr\_patient1 = 0, nbr\_patient2 = 0, nbr\_patient3 = 0, quit;

do{

switch(Acceuille(&nbr\_patient1, &nbr\_patient2, &nbr\_patient3)){

case 1:

Enregistrer(&nbr\_patient1, &nbr\_patient2, &nbr\_patient3);

break;

case 2:

Modifier();

break;

case 3:

Supprimer();

break;

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 créé 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 saisis 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, ont 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;

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\tSexe = %s\tage : %dans\n", p.nom, p.prenom, p.sexe, p.age);

printf("Rendez-vous : %s a %dh%dmn\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***

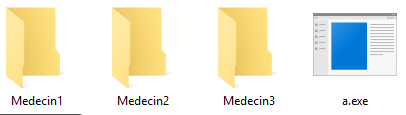
Le fonctionnement est le suivant :

* 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 placer au même endroit que l’exécutable. Et il faut les créer manuellement.

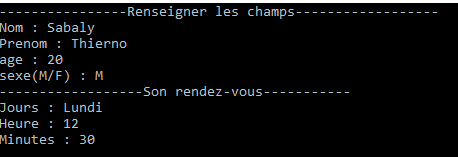
*Menu*

**

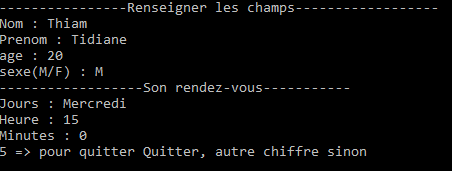
*Création de rendez-vous*

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





Le rendez-vous pour le médecin 1.

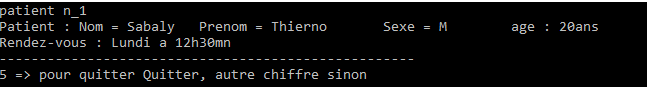


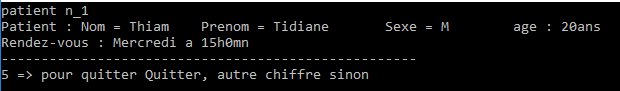
Le rendez-vous du médecin 3.

Les dossiers après cette action : 

*Affichage*

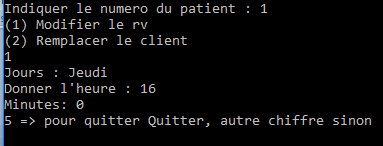
 Toujours on indique le médecin

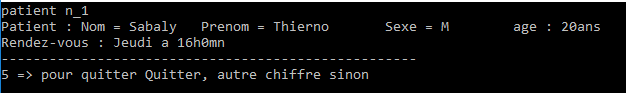
médecin1

médecin3

*Modification*



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

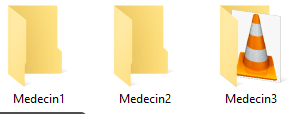
Affichage : 

*Suppression :*



Nous allons supprimer le rendez-vous du médecin 1.



Résultat : 

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 atructure 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 boutton boutton;

enum boutton{

ACC, MED, SEC, AID,

//Les bouttons d'accueille

ACC\_BOUT\_0, ACC\_BOUT\_1, ACC\_BOUT\_2, NULL\_PART

};

//Les enumérations 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 créé un dossier pour chaque médecin de la clinique.
* Dans le dossier de chaque médecin, on créé deux sous-dossiers. Un appelé ‘**’disponibilités’’** et l’autre ‘**’RendezVous’’** .
* De même, on créé un dossier ‘’ Secrétaires’’ et chaque secrétaire on lui créé 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 souri, les clics et la saisi. En plus nous avons déjà eu un aperçu des fonctions de la SDL que nous avons fait appel, on pourra ensuite exposé les codes source des autres fonctions dont on expliquera le fonctionnement.

*Les déplacements de la souri :* La souri est facile à repérer avec la SDL. Chaque déplacement de la souri est un événement de type SDL\_MOUSMOTION qui nous renvoie l’abscisse de la souri (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 souri obtenues, il devient facile de savoir si elle est ou pas sur un bouton. Sauf que nos boutons sont dans différentes zone de l’application. Certains au niveau de l’accueil, d’autre 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 créé 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 souri :* 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\_MOUSEBUTONUP.

Un exemple : Déclaration

Bout\_Esp\_Med Bout\_Med\_Appuye = AUCUN;

while(continuer){

while(SDL\_PollEvent(&event)){

case SDL\_MOUSEBUTONUP:

Bout\_Med\_App = Bout\_Appuye\_Med(event.buton.x, event.buton.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’accueille

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 boutton){

char\* aretourner;

switch(boutton.keysym.sym){

case SDLK\_a:

strcpy(aretourner, "q");

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 boutton){

char\* aretourner;

switch(boutton.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 entrée sont identiques. Si oui, elles créé 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 saisi. 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 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 créé 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 cherche 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 créé 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 accueille médecin :* L’espace médecin une page d’accueille 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 Acceuille\_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’accueille médecin, les disponibilités et les rendez-vous (ces dernière font appelle 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:

Acceuille\_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 avions 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);

//récuperation 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\_Survole = AUCUN;

//Ce sont les variables permettant de stocker les bouttons appuyés dans l'accueille des medecins

Bout\_Acc Bout\_Surv\_AccMed = NONE;

Bout\_Acc Bout\_App\_AccMed = NONE;

Bout\_Acc Bout\_AppPrec\_AccMed = NONE;

//Ici les bouttons appuyés lors de l'identification (connexion)

Bout\_ID Bout\_For\_Iden = RIEN;

Bout\_ID Bout\_For\_IdenBefore = RIEN;

//ici les bouttons 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;

boutton Pass\_Sur\_Bout = NULL\_PART;

Lieu Zone = ACCUEILLE;

* On récupère leurs valeur lors d’un déplacement de la souri

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 bouttons 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 = Boutton\_AccSurv(event.motion.x, event.motion.y, Fenetre\_Long, Fenetre\_Larg, Bout\_Surv\_AccMed);

Bout\_NouvelDispo\_Surv = Boutton\_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 bouttons 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 an 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 créé 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 somme 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 appui 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;

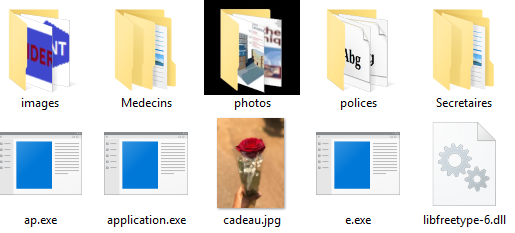
break;

}

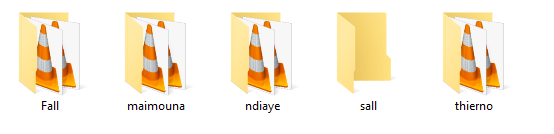
* Il y’a tellement d’autres fonctions : comme celle qui anime les photos, celle qui affiche la date, et récemment nous avons créé celles qui ordonnent les disponibilités et les rendez-vous. Toutes ces fonctions sont appelées dans la fonction main.

***Démonstration***

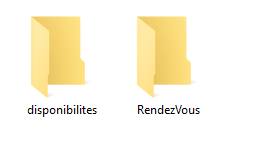
Fini la théorie, voyons tous cela en pratique. Commençons par avoir un dossier ‘’Medecins’’.

le voici avec l’exécutable.

Créons dedans un dossier du nom de ‘’sall’’.

voilà

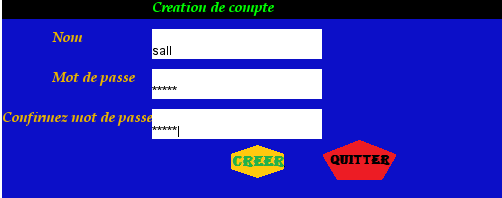
Dans le dossier ‘’sall’, créer les deux sous dossiers ‘’disponibilités’’ et ‘RendezVous’’

Sall est maintenant prêt à utiliser l’application.

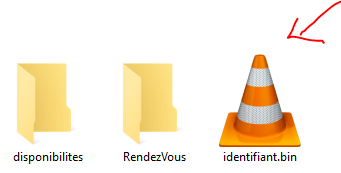
* Création de compte :

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

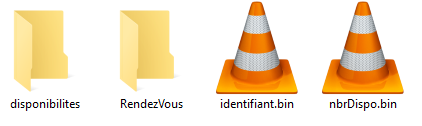
voici ces identifiant

* Connexion





Revoyons le dossier

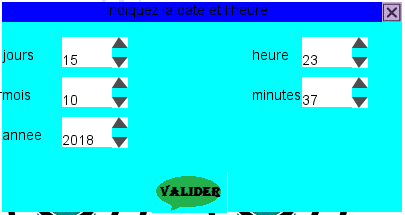


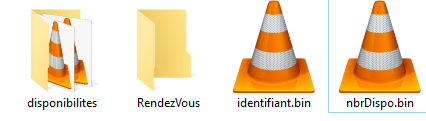
Un fichier pour stocker les disponibilités est créé.

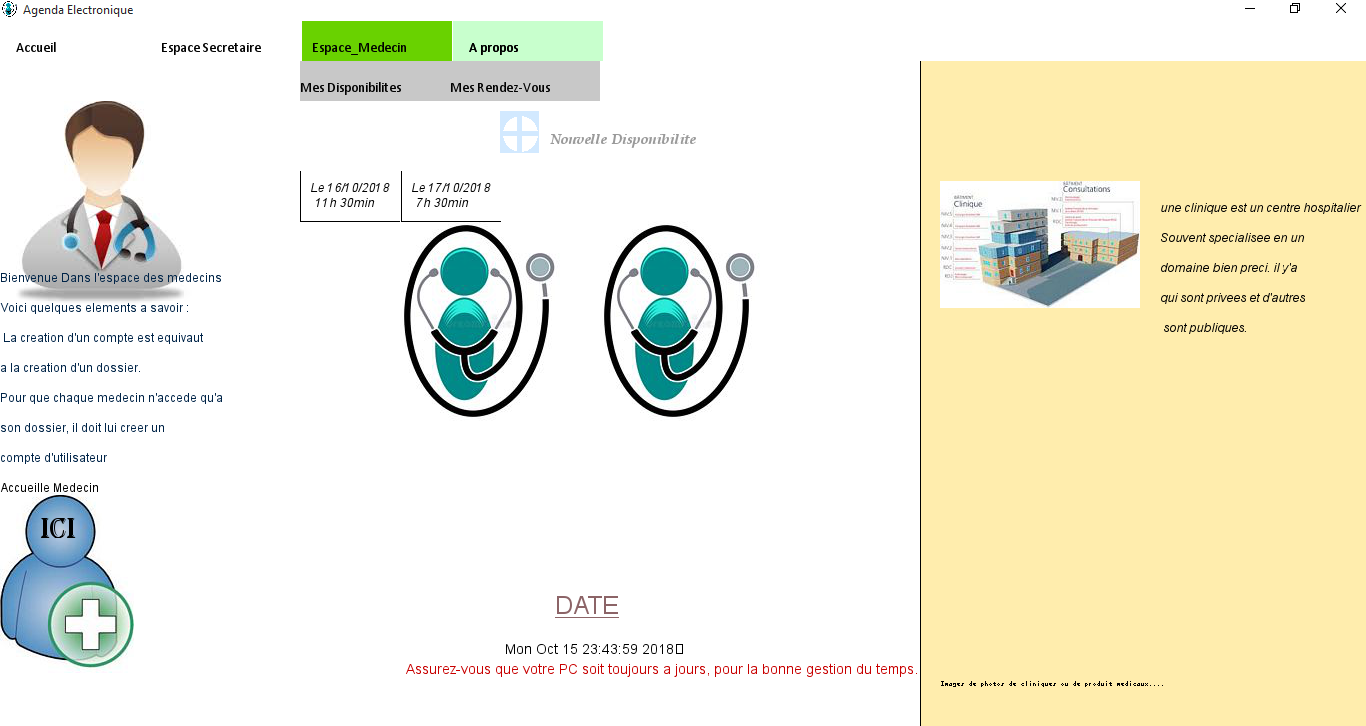
* Désormais, M. Sall a accès à son dossier. Il peut créer ses disponibilités (initialement vide).



Créons en deux.

fonction permettant de créer des disponibilités

voilà des disponibilités ont été créées

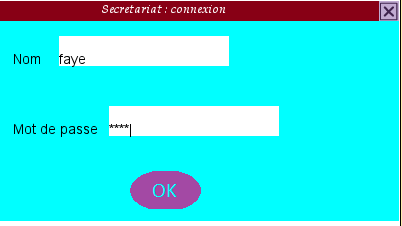


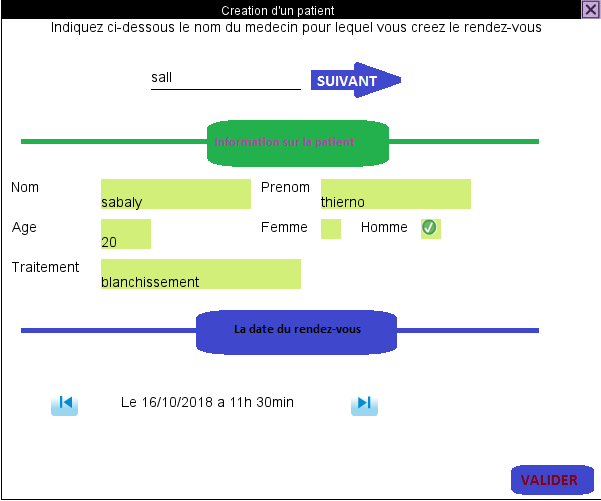
* Les rendez-vous

C’est au secrétariat que se créer les rendez-vous. On va utilisé un compte que j’ai déjà créé pour remplir le dossier de rendezVous de M. Sall

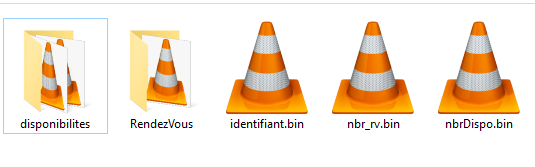
Secrétariat.

Connexion

fonction connexion

 fonction pour la création de rendez-vous.

Voyons le dossier de sall.

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