Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes - RABAT

Rapport de Projet de programmation : Gestion citoyens/patients de COVID-19

Réalisé par :

Anass Dkheila
Amine Ait Oufkir

Encadré par :

Pr. A. El Hassouny

Année académique 2020/2021

Remerciements :

Nous voudrons tout d'abord adresser toute notre gratitude à notre professeur A. El Hassouny , pour sa confiance, sa disponibilité et surtout cette opportunité pour bien maîtriser le langage de programmation C et initier notre carrière par un aussi beau sujet.

Nous désirons aussi remercier tous ceux qui contribuaient à la réussite de ce projet, et bien d'autres pour leurs conseils et leurs connaissances qu'ils nous ont partagé.

Enfin, nos remerciements s'adressent aux professeurs et au corps administratif de l'Ecole Nationale Supérieure de l'Informatique et de l'Analyse des Systèmes-ENSIAS

Table des matières

INTRODUCTION GENERALE :

De nos jours, beaucoup de gens reviennent aux outils informatiques afin de faciliter leurs vies quotidiennes. Ces outils peuvent être utilisés dans plusieurs domaines, à savoir : le domaine professionnel, domaine sanitaire et d'autres domaines.

Lors des événements actuelles ;la pandémie de COVID-19. la direction de l'épidémiologie et de Lutte contre les maladies communiquent régulièrement au sujet de cette infection, ceci n'est possible que par la gestion des patients/citoyens du COVID-19.

Pour cela, après avoir étudié plusieurs matières de « langage C » et de « Structures de données » durant le premier semestre, et afin de nous permettre de mettre en pratique les différentes notions acquises durant les cours, les TDs et les TPs, nous étions invités à développer une application en C pour la Gestion citoyens/patients de COVID-19.

Chapitre 1

Problématique et cadre général du Projet

1.1 Introduction

PROJET C s'agit de produire un programme d'environ une centaines de lignes de code afin de valider les compétences descours : « Algorithmique », « Technique de programmation » et « Structures de données ».Le programme correspond à 20h heures de travail effectives en langage C.
Le travail se fait en binome en bénéficant des conseils d'un professeur encadrant. [1]

1.2 Problématique

Pour anticiper la propagation de la pandémie Covid-19, la direction de l'épidémiologie et de Lutte contre les maladies, a adopté une série de procédures préventives dont l'entrée en vigueur de l'état d'urgence sanitaire et la restriction de la circulation dans les tous coins du pays.
Afin de faciliter la tache de la gestion des citoyens/patients du COVID-19, on a conçu informatiser ce processus pour optimiser la prise en charge des patients, pour une meilleure exploitation des listes des patients en cours, patients guéris et pour contrôler les contacts des patients, statistique, etc.
Notre problématique est axée, principalement, sur la création d'une application gestion. Le besoin exprimé se formule comme suit :

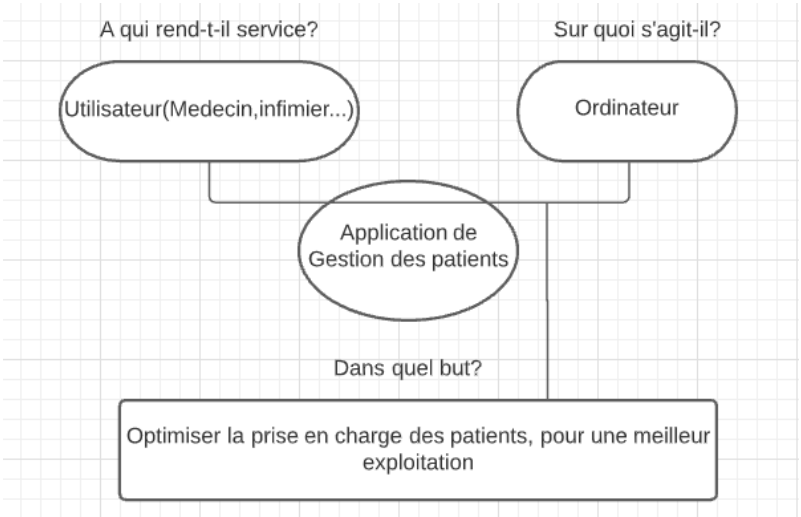

FIGURE 1.1 – Diagramme bête à corne

1. Notice du projet de programmation 2019/2020

1.3 Cahier de charges

Le cahier de charges présente l'ensemble des instructions et contraintes qui cadrent la réalisation du jeu. Le cahier de charges disponible donne plusieurs instructions qu'on va élaborer dans cette partie.

1.3.1 Contexte et objectifs du projet

Il s'agit de développer une application en langage C pour la gestion Gestion citoyens/patients de COVID-19. Pour le réaliser, nous avons besoin d'utilisé des structures de différentes natures, dont les champs contiennent les informations proposées ci-dessous :

1. Un patient est caractérisé par la structure ci-dessous :

```c
typedef struct patient{
    char nom_complet[30]; //Sous la forme: Prénom_Nom
    char etat[30];// L'état du patient: Normal,Critique,Besoin aide respiratoire
    char typechambre[30];//Type de chambre occupée: Normal,Reanimation
    char ville[30];
    char CIN[30];//CIN unique
    char jourinfection[30]; //le jour de l'infection (Lundi,...,Dimanche)
    int age;
    int chambre;// Le nombre du chambre occupée
    int occupelit; // 0 s'il n'occupe pas un lit 1 sinon
}patient;
```

FIGURE 1.2 – Structure patient

2. Un Contact est caractérisé par la structure ci-dessous :

```c
typedef struct contact_patient{
    char nom_complet[30]; //Sous la forme: Prénom_Nom
    char ville[30];
    char secteur[30];
    char CIN[30]; //CIN unique
    char tel[30]; //Une chaine de caractére pour conserver le 0 au début
    int age;
}contactpatient;
```

FIGURE 1.3 – Structure contact

3. Un Patient guéri est caractérisé par la structure ci-dessous :

```c
typedef struct patient_gueri{
    char nom_complet[30]; //Sous la forme: Prénom_Nom
    char ville[30];
    char secteur[30];
    char CIN[30]; //CIN unique
    char jourgueri[30]; //le jour de guérison (Lundi,...,Dimanche)
    char tel[30]; //Une chaine de caractére pour conserver le 0 au début
    int age;
}patientgueri;
```

FIGURE 1.4 – Structure guéri

4. Un Patient décédé est caractérisé par la structure ci-dessous :

```c
typedef struct patient_mort{
    char nom_complet[30]; //Sous la forme: Prénom_Nom
    char ville[30];
    char secteur[30];
    char CIN[30]; //CIN unique
    char jourmort[30];//le jour de décés (Lundi,...,Dimanche)
    char tel[30];//Une chaine de caractére pour conserver le 0 au début
    int age;
}patientmort;
```

FIGURE 1.5 – Structure décés

5. Une ville et un secteur caractérisé par les structures ci-dessous :

```
typedef struct _ville{
char nomville[30];
}ville;
```

FIGURE 1.6 – Structure ville

```
typedef struct _secteur{
char nomsecteur[30];
}secteur;
```

FIGURE 1.7 – Structure secteur

1.3.2 Principe du programme :

Notre programme devra propose à l'utilisateur un menu pour pouvoir manipuler tous les options possibles souhaité, et naviguer entre les différentes fonctionnalités réalisées, comme expliquons au-dessous :

1. **Gestion des patients :**
   - Ajouter, modifier ou supprimer un patient ou Afficher la liste des patients.
   - Lister les patients selon plusieurs critères : ville/secteur, état (normal, critique, besoin d'aide respiratoire), nombre des cas par ville, nombre des cas critiques, taux d'occupation des lits.
   - Retour au menu principal.
2. **Gestion des Contacts :**
   - Ajouter, modifier ou supprimer un contact ou Afficher la liste contacts.
   - Lister les contacts des patients selon plusieurs critères : ville, secteur, nombre par ville.
   - Retour au menu principal.
3. **Gestion des patients guéris :**
   - Ajouter, modifier ou supprimer un patient guéri ou Afficher la liste patients guéris.
   - Lister patients guéris selon plusieurs critères : ville, secteur, nombre par ville, taux de guérison par (jour, total).
   - Retour au menu principal.
4. **Gestion des décès :**
   - Ajouter, modifier ou supprimer un décès ou Afficher la liste décès.
   - Lister décès selon plusieurs critères : ville, secteur, nombre par ville, taux de décès par (jour,ville,total).
   - Retour au menu principal.

1.4 Procédure et étapes :

1. Etape 1 :Choix de type de gestion voulue ou quitter le programme :

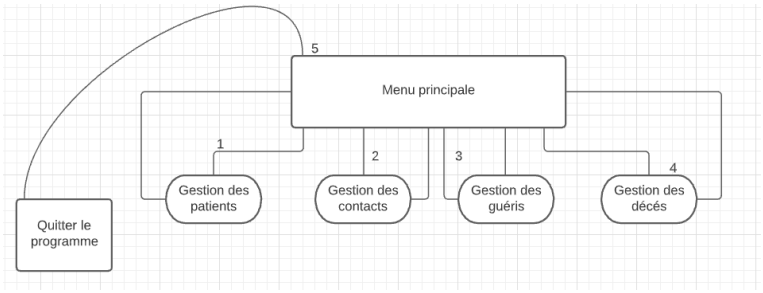

FIGURE 1.8 – Schématisation de l'étape 1

2. Etape 2 :Choix de l'opération voulue ou retourner au menu principal :

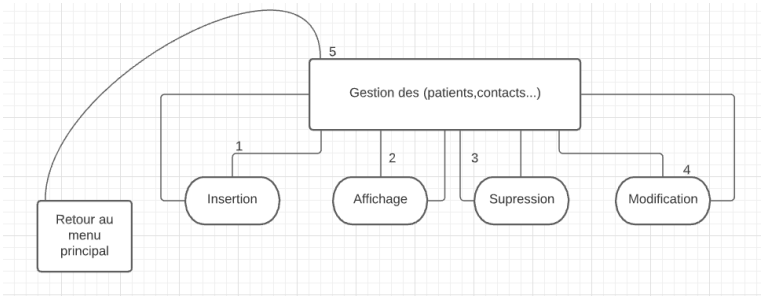

FIGURE 1.9 – Schématisation de l'étape 2

1.5 Conclusion

Après avoir dégagé les besoins fonctionnels et opérationnels et tous les critères qu'on doit prendre en considération, et afin de modéliser les besoin attendus de notre application et que les objectifs soient atteinte, on va suivre la démarche du processus unifié qu'on va le détailler dans la prochaine partie.

Chapitre 2

Analyse théorique et conception

Dans ce chapitre, nous présentons la partie réalisation et mise en oeuvre de notre travail.

Pour cela, nous présentons,les méthodes de manipulation de données, l'environnement de travail et les outils de développement utilisés.

2.1 Les étapes de la résolution du probléme

Dans cette partie, nous discuterons quelques méthodes de résolution pour travaille exigé dans le cahier de charges.

La gestion du programme se base essentiellement sur la gestion des listes chaînées pour simplifier les opérations d'insertion, de supression et de modification et l'affichage des données.

2.1.1 Gestion des données

Tout d'abord, on tient à commenter que la manipulation des données d'un patient, contact, guéri et décédé poursuivent le même déroulement. Et comme mentionnait ci-dessus, nous avons choisi les listes chaînées car ils sont simple à manipuler. Donc on aura besoin des structures mentionnée auparavant qui stockent les informations du patient/contact/guéri/décédé puis une liste associé à ces structures.

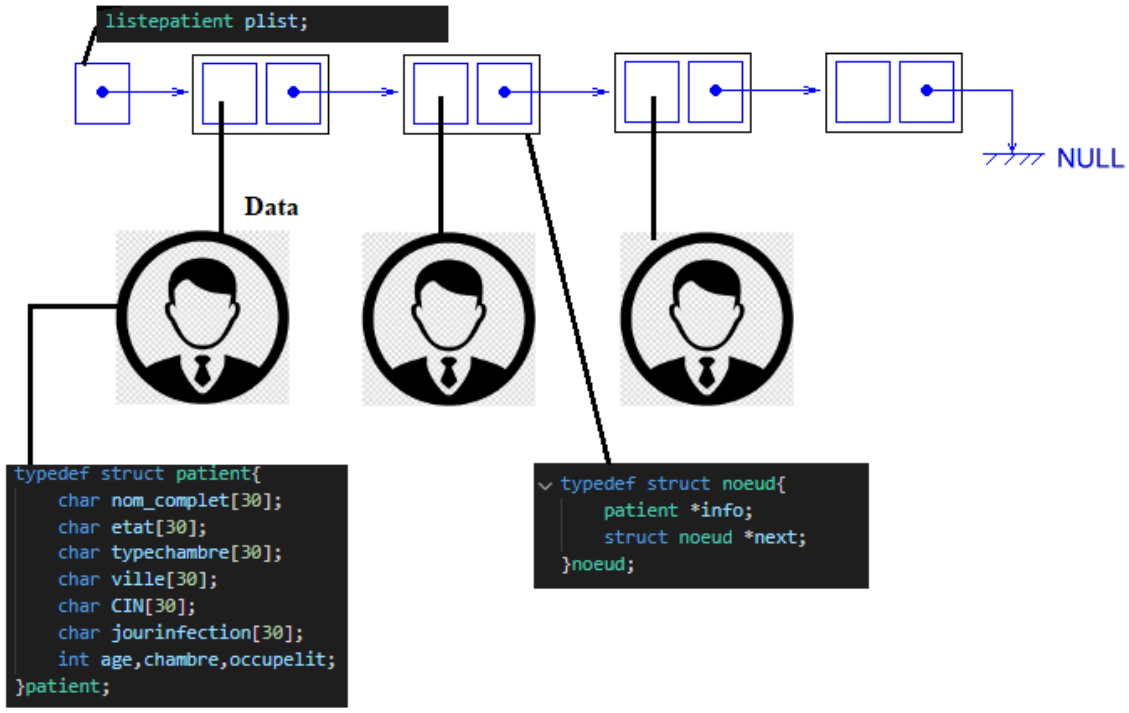

FIGURE 2.1 – Representation des données patient

2.1.2 Sauvegarde et charge des données

Afin de pouvoir sauvegarder automatiquement les données taper et les recharger en cours de l'utilisation , il faut faire usage au méthodes de stockages des fichiers.En effet, pour enregistrer les informations des patients/-contact/guéri/décédé , on a besoin des fichiers binaires (ou textes) pour stocker ces informations. Pour cela, nous avons utilisé quatres fichiers binaires ListePatients.data et ListeContactpatients.data et Listepatientsgueri.data et Listepatientsmort.data .

Pour atteindre cet objectif, nous avons définis des fonctions que l'on verra dans le prochaine chapitre.

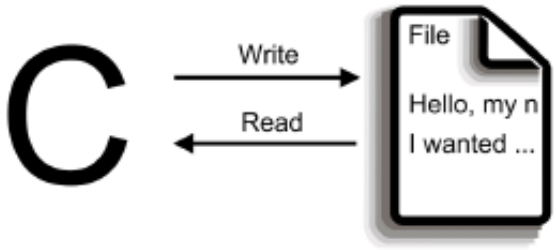

FIGURE 2.2 – Representation des données patient

2.2 Les outils et techniques de développement

2.2.1 Construction de l'application

Visual Studio Code est un éditeur de code source développé par Microsoft pour Windows, Linux et MacOs, il inclut la prise en charge du débogage, du contrôle Git intégré et de GitHub, gratuit, supportant une dizaine de langages dont langage C qu'on a utilisé dans ce projet.

FIGURE 2.3 – Logo de Visual Code

2.2.2 Redaction du rapport

La documentation professionelle nécessite la manipulation du logiciel de traitement de texte LaTex. Travailler avec ce dernier est inévitable tôt ou tard, donc nous avons voulu exploiter cette opportunité et explorer LaTex pour la premier fois.

FIGURE 2.4 – Logo du logiciel laTEX

Chapitre 3

Réalisations et résultas

3.1 Difficultés rencontrées

Cette section présente les soucis que nous avons affronté durant la réalisation de projet.

3.1.1 Compréhension du sujet

Parmi les difficultés que nous avons rencontrées est la comprehension du sujet, il y'avait qulques taches qui étaient facile à réaliser mais d'autres qui ont pris du temps pour les comprendre puis les réaliser sur code (les fonctions lister...)

3.1.2 Rédaction et organisation du code

Pour realiser ce programme, il fallut écrire plusieurs fonction des centaines des lignes de codes en utilisant notre connaissance, comme c'est notre premier projet avec le langage C c'était indispensablent difficile (les erreurs de compilations ...).De plus, la division du code en fichier " .c " et un fichier " .h ".

3.1.3 Contrainte de temps et situation actuelle

La travaille avec plusieurs technologies durant ces conditions ecxeptionel cette année sur ce projet a consommé pas mal de temps. Il y'avait aussi la contrainte du travail en groupe ceci était difficile en ligne.

3.2 Les fonctions du code source

Pour réaliser ce travail nous avons besoin de réaliser plusieurs fonctions. Dans cette partie nous allons élaborer chacune des fonctions utilisés dans le code source.

— **Menu principal :void menu()**

La premiére fenétre affiche le Menu Principal, ce dernier contient les options pour selon le type de gestion voulue.

Sinon, il y a la possibilité de quitter le logiciel.

FIGURE 3.1 – Menu principal

3.2.1 Les fonctions des menus

— void menu1()
Cette fonction permet de naviguer tous les différentes fonctions qui permettent la gestion des patients.

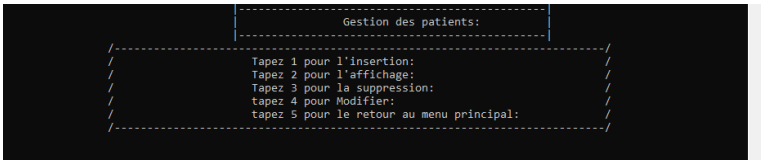

FIGURE 3.2 – Menu gestion des patients

— void menu2()
Cette fonction permet de naviguer tous les différentes fonctions qui permettent la gestion des contacts.

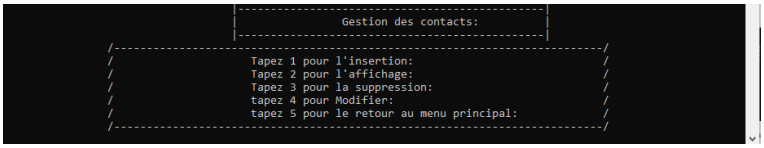

FIGURE 3.3 – Menu gestion des contacts

— void menu3()
Cette fonction permet de naviguer tous les différentes fonctions qui permettent la gestion des guéris.

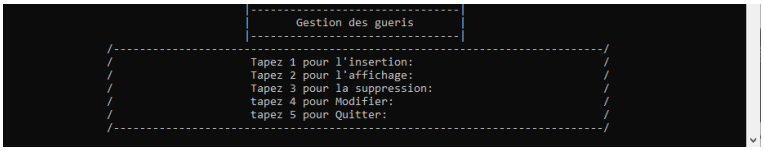

FIGURE 3.4 – Menu gestion des guéris

— void menu4()
Cette fonction permet de naviguer tous les différentes fonctions qui permettent la gestion des décés.

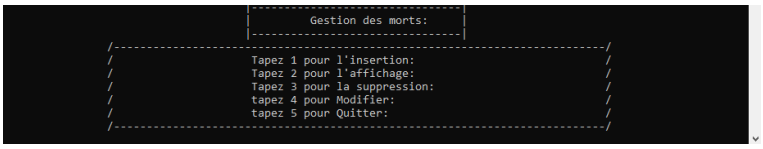

FIGURE 3.5 – Menu gestion des décés

— void modificationmenu()
Aprés choisir l'élément à modifier par son CIN cette fonction permet de choisir lequelle des données du patient on veut modifier.

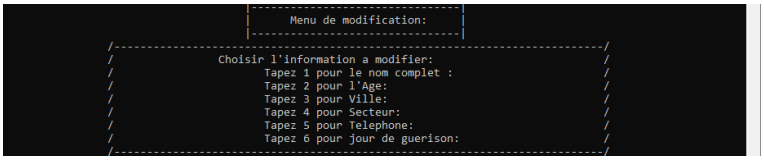

FIGURE 3.6 – Menu des modifications possibles.

— void modificationmenucontact()
Aprés choisir l'élément à modifier par son CIN cette fonction permet de choisir lequelle des données du contact on veut modifier.

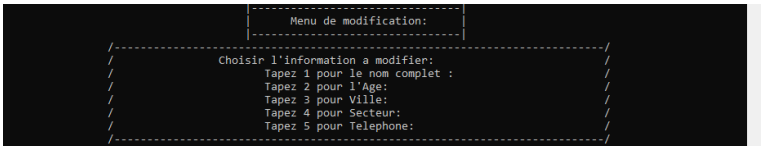

FIGURE 3.7 – Menu des modifications possibles.

— void modificationmenugueri()
Aprés choisir l'élément à modifier par son CIN cette fonction permet de choisir lequelle des données du guéri on veut modifier.

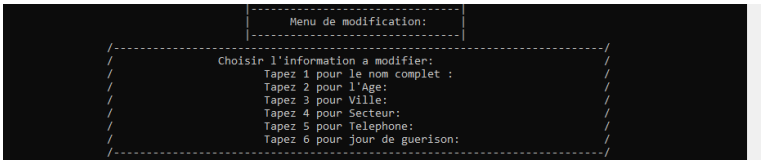

FIGURE 3.8 – Menu des modifications possibles.

— void modificationmenumort()
Aprés choisir l'élément à modifier par son CIN cette fonction permet de choisir lequelle des données du décédé on veut modifier.

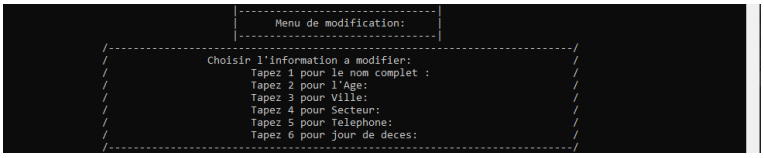

FIGURE 3.9 – Menu des modifications possibles.

— void menuaffichage()
Cette fonction permet de naviguer tous les différentes fonctions d'affichage pour le patient.

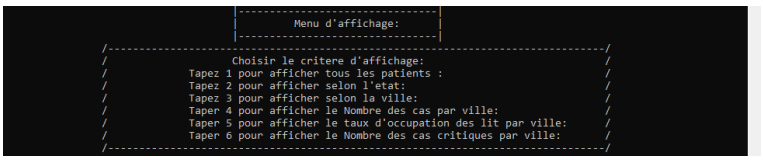

FIGURE 3.10 – Menu des affichages possibles.

— void menuaffichagecontact()
Cette fonction permet de naviguer tous les différentes fonctions d'affichage pour le contact.

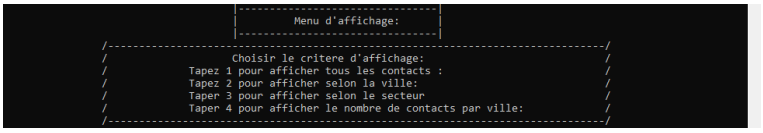

FIGURE 3.11 – Menu des affichages possibles.

— void void menuaffichagegueri()
Cette fonction permet de naviguer tous les différentes fonctions d'affichage pour le guéri.

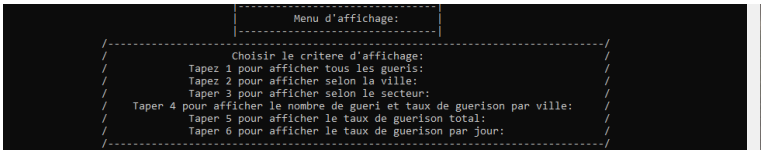

FIGURE 3.12 – Menu des affichages possibles.

— void menuaffichagemort()
Cette fonction permet de naviguer tous les différentes fonctions d'affichage pour le décédé.

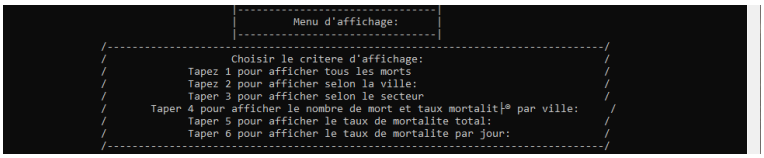

FIGURE 3.13 – Menu des affichages possibles.

3.2.2 Les fonctions du patient

— listepatient add(patient *d,listepatient l)
Cette fonction permet d'ajouter un patient.

— listepatient supprimer(patient *d,listepatient l)
Cette fonction permet de supprimer un patient.

— listepatient modifier(patient *d,listepatient l)
Cette fonction permet de modifier les données d'un patient selon le choix.

— void tauxoccuplit(listepatient l)
Cette fonction affiche le taux d'occupation de lit par ville.

— void afficheretat(listepatient l)
Cette fonction affiche les patients selon l'état.

— int length(listepatient l)
Cette fonction retourne la longueur de la liste patient.

— void afficherparville(listepatient l, char choixville[30])
Cette fonction affiche les patients par ville.

— void nbcascritiqueparville(listepatient l)
Cette fonction affiche le nombres de cas critique par ville.

— void nbcasparville(listepatient l)
Cette fonction affiche le nombres de cas par ville.

— void afficher(listepatient l)
Cette fonction affiche tous les patients.

3.2.3 Les fonctions du contact

— listecontactpatient ajoutercontact(contactpatient *d,listecontactpatient l)
Cette fonction permet d'ajouter un contact.

— listecontactpatient supprimercontact(contactpatient *d,listecontactpatient l)
Cette fonction permet de supprimer un contact.

— listecontactpatient modifiercontact(contactpatient *d,listecontactpatient l)
Cette fonction permet de modifier les données d'un contact selon le choix.

— int lengthcontact(listecontactpatient l)
Cette fonction retourne la longueur de la liste contact.

— void affichercontactparville(listecontactpatient l, char choixville1[30])
Cette fonction affiche les contacts par ville.

— void affichercontactparsecteur(listecontactpatient l, char choixsecteur[30])
Cette fonction affiche les contacts par secteur.

— void nbcontactparville(listecontactpatient l)
Cette fonction affiche le nombres de contacts par ville.

— void affichercontact(listecontactpatient l)
Cette fonction affiche tous les contacts.

3.2.4 Les fonctions du guéri

— listepatientgueri ajoutergueri(patientgueri *d,listepatientgueri l)
Cette fonction permet d'ajouter un guéri.

— listepatientgueri supprimergueri(patientgueri *d,listepatientgueri l)
Cette fonction permet de supprimer un guéri.

— listepatientgueri modifiergueri(patientgueri *d,listepatientgueri l)
Cette fonction permet de modifier les données d'un guéri selon le choix.

— int lengthgueri(listepatientgueri l)
Cette fonction retourne la longueur de la liste guéri.

— void affichergueriparville(listepatientgueri l, char choixville2[30])
Cette fonction affiche les guéris par ville.

— void affichergueriparsecteur(listepatientgueri l, char choixsecteur2[30])
Cette fonction affiche les guéris par secteur.

— void nbgueriettauxparville(listepatientgueri l,listepatient plist)
Cette fonction affiche le nombres de guéris et le taux de guérison par ville.

— void tauxdeguerisontot(listepatientgueri l,listepatient plist)
Cette fonction affiche le taux de guérison total.

— void tauxgueriparjour(listepatientgueri l,listepatient plist)
Cette fonction affiche le taux de guérison par jour.

— void affichergueri(listepatientgueri l)
Cette fonction affiche tous les guéris.

3.2.5 Les fonctions du décédé

— listepatientmort ajoutermort(patientmort *d,listepatientmort l)
Cette fonction permet d'ajouter un décédé.

— listepatientmort supprimermort(patientmort *d,listepatientmort l)
Cette fonction permet de supprimer un décédé.

— listepatientmort modifiermort(patientmort *d,listepatientmort l)
Cette fonction permet de modifier les données d'un décédé selon le choix.

— int lengthmort(listepatientmort l)
Cette fonction retourne la longueur de la liste décé.

— void affichermortparville(listepatientmort l, char choixville3[30])
Cette fonction affiche les décés par ville.

— void affichermortparsecteur(listepatientmort l, char choixsecteur3[30])
Cette fonction affiche les décés par secteur.

— void nbmortettauxparville(listepatientmort l,listepatient plist1)
Cette fonction affiche le nombres de décés et le taux de mortalité par ville.

— void tauxdemortsontot(listepatientmort l,listepatient plist)
Cette fonction affiche le taux de mortalité total.

— void tauxmortparjour(listepatientmort l,listepatient plist1)
Cette fonction affiche le taux de mortalité par jour.

3.2.6 Les fonctions du général du code

— float division(int A,int B)
Cette fonction retourne le division de deux nombres.

3.3 Organisation du projet et les fichiers

1. **Les fichiers utilisé :**
   Comme nous avons déja mentionné précedament, pour enregistrer les informations des offres, des candidats,et des recrutements, on a besoin des fichiers binaires (ou textes) pour stocker ces informations.Pour cela, nous avons utilisé quatres fichiers binaires ListeContactpatients.data ,ListePatient.data ,Listepatientsmort.data et Listepatientsgueri.

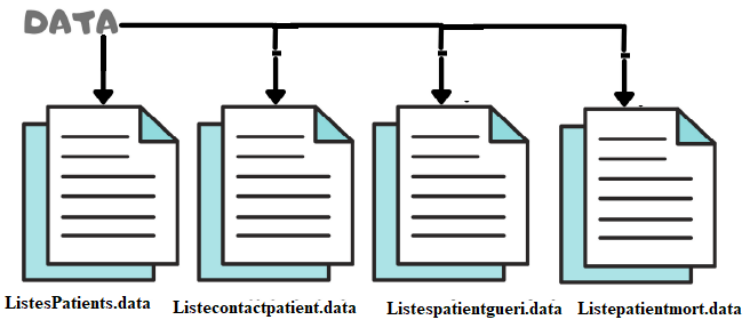

FIGURE 3.14 – Les fichiers binaires.

— void writeListePatientToFile(listepatient l)
Cette fonction permet de sauvegarder la liste des patient dans le fichier ListePatients.data.

— listepatient Chargerpatient(listepatientl)
Cette fonction permet de charger la liste des patient du fichier ListePatients.data.

— void writeListeContacttToFile(listecontactpatient l)
Cette fonction permet de sauvegarder la liste des contact dans le fichier ListeContactpatients.data.

— listecontactpatient Chargercontact(listecontactpatient l)
Cette fonction permet de charger la liste des contacts du fichier ListeContactpatients.data.

— void writeListegueriToFile(listepatientgueri l)
Cette fonction permet de sauvegarder la liste des guéris dans le fichier Listepatientsgueri.data.

— listepatientgueri Chargergueri(listepatientgueri l)
Cette fonction permet de charger la liste des guéris du fichier Listepatientsgueri.data.

— void writeListemortToFile(listepatientmort l)
Cette fonction permet de sauvegarder la liste des guéris dans le fichier Listepatientsmort.data.

— listepatientmort Chargermort(listepatientmort l)
Cette fonction permet de charger la liste des guéris du fichier Listepatientsmort.data.

2. **Organisation du projet :**
Pour y parvenir, nous avons répartis notre programme en un fichier (.c) et un fichier en-tête (.h) (head-file)

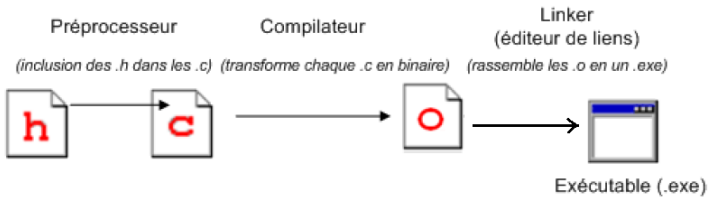

FIGURE 3.15 – La programmation modulaire.

— main.c
Ce fichier contient le Menu principal et toutes les fonctions créés.
— header.h
Ce fichier contient les éléments que devra connaître un utilisateur du module (définition de constantes, Macro-instructions, déclarations de variables globales, déclaration des fonctions (prototypes) définies dans le fichier .c). Ce fichier constitue l'interface du module et doit être inclut dans tous le fichier (.c) .

Conclusion générale

Notre projet a consisté à la réalisation une application qui automatise la gestion des patients COVID-19. Ce projet nous a permis d'approfondir nos connaissances théoriques, acquises tous le long de notre formation.Nous sommes arrivés à développer la majorité des fonctionnalités du système dans les temps. L'amélioration qu'on peut voir dans ce projet et le faite qu'on peut impléments une interface graphique quelconque et le mettre sous forme d'application.

Bibliographie

[1] Le cours de Structure de données de Monsieur EL FAKER, 2021. [Online].

[2] `https://www.infovac-maroc.com/covid19/Preesentation_COVID19.pdf`, 2021. [Online].

[3] Le cours de « Technique de programmation » de Monsieur Hatim GUERMAH, 2021. [Online].