***Le fichier d’interface unite.h***

Nous avons protégé ce fichier contre multiples inclusions en utilisant l’approche suivante :

#ifndef UNITE\_H

#define UNITE\_H

#include <stdio.h>

/\* corps du fichier interface unite.h \*/

#endif

Ensuite, nous avons défini la structure *Unite*. Elle a pour but de faire des traitements sur les unités dans le logiciel de secours. La représentation de cette structure est la suivante :

typedef struct {

int iCode;

char cNom[TAILLE\_BUFFER];

char cMoyenDeplacement[TAILLE\_BUFFER];

char cNiveauDisponibilite[TAILLE\_BUFFER];

char cStatut[TAILLE\_BUFFER];

char cBase[TAILLE\_BUFFER];

int iCompteurRepos;

int iEstEnRepos;

int iUniteDisponible;

int iDeployeeSurAlerte;

} Unites;

Nous avons déclaré au total 10 membres de la structure *Unites.* Selon le cahier de charges, chaque unité doit contenir un code individuel généré par le logiciel. Le premier membre est donc int iCode qui sert pour le stockage du code aléatoire généré par le logiciel. Les membres suivants :

char cNom[TAILLE\_BUFFER];

char cMoyenDeplacement[TAILLE\_BUFFER];

char cNiveauDisponibilite[TAILLE\_BUFFER];

char cStatut[TAILLE\_BUFFER];

char cBase[TAILLE\_BUFFER];

ont pour but de stocker l'information contenant le nom, moyen de déplacement, niveau de disponibilité, statut et base d'unité. Ils ont une TAILLE\_BUFFER qui contient 150 cases vides pour les chaînes de caractères. Ensuite, il un entier int iCompteurRepos qui stocke le nombre de missions effectuées et un autre entier :

int iDeployeeSurAlerte qui stocke le code d'alerte que traite cette unité.

Puisque nous n’avons pas vraiment le type boolean dans le langage C et pour éviter d’utiliser d’autres librairies qui contiennent ce type, nous avons décidé de simplifier les choses et ne pas surcharger le logiciel. Nous utilisons une variable de type entier qui va définir avec les valeurs 0 (faux) et 1 (vrai) si une alerte est en repos int iEstEnRepos et si elle est disponible int iUniteDisponible.

La liste de fonctions

Nous avons déclaré 14 fonctions au total qui permettent de traiter une unité :

void creerUnite(Unites \*\*unite, int \*compteurUnite);

void afficherUneUnite(Unites \*\*unite, int \*compteurUnite);

void afficherToutesUnites(Unites \*\*unite, int \*compteurUnite);

void modifierUnite(Unites \*\*unite, int \*compteurUnite);

void supprimerUnite(Unites \*\*unite, int \*compteurUnite);

char \* moyenDeplacement();

char \* niveauDisponibilite();

char \* statutUnite();

int genererCodeUnite();

void mettreUniteEnRepos(Unites \*\*unite, int \*iCompteurUnite);

void unitesStatiques(Unites \*\*unite, int \*iCompteurUnite);

void consulterUnitesDeployees(Unites \*\*unite, int \*iCompteurUnite);

void imprimerLesUnites(Unites \*\*unite, int \*iCompteurUnite);

void chargerLesUnites(Unites \*\*unite, int \*iCompteurUnite);

La fonction creerUnite(Unites \*\*unite, int \*compteurUnite); sert pour la création d’une nouvelle unité. Elle de type void et donc elle ne retourne pas de valeurs. Elle prend en paramètre un pointeur de pointeur de la structure *Unites Unites \*\*unite* et un pointeur de type entier int \*compteurUnite qui décrits le nombre de cases total dans le tableau dynamique d’*Unites* et qui sert également pour le parcours du tableau et le traitement des cases. Cette fonction fait aussi appel aux fonctions supplémentaires qui contiennent informations par rapport aux unités. Ces sont les fonctions utilisées à l'intérieur de la fonction :

char \* moyenDeplacement();

char \* niveauDisponibilite();

char \* statutUnite();

int genererCodeUnite();

Elles prennent pas de paramètres mais elles retournent un pointeur de type char, ces pointeurs contiennent une chaîne de caractères selon le choix de l’utilisateur qu’il a saisi et un entier int qui est le code aléatoire d’alerte qui vient d'être créée. L’ensemble de déroulement de ces fonctions ont une forme schématique suivante :



Les fonctions char \* moyenDeplacement(); char \* niveauDisponibilite(); et char \* statutUnite(); retournent un pointeur de type de chaîne de caractères selon la choix de l'utilisateur.  
 Les fonctions

void afficherUneUnite(Unites \*\*unite, int \*compteurUnite);

void afficherToutesUnites(Unites \*\*unite, int \*compteurUnite);

servent pour l’affichage des alertes. En utilisant un entier int \*iCompteurUnite ces fonctions parcourent le tableau de la structure *Unites* et affichent les résultats trouvés ou un message correspondante s’il n’y a aucune alerte enregistrée. Elle ne retournent pas de valeurs et elles prennent en paramètre un pointeur de pointeur de la structure *Unites* et un entier int \*iCompteurUnite qui sert pour décrire la quantité de cases dans le tableau dynamique. Dans la première fonction la recherche est basée sur le code donnee par l’utilisateur alors que la deuxième fonctions affiche toutes les unités du tableau.

La fonction void modifierUnite(Unites \*\*unite, int \*compteurUnite); retourne aucun valeur et prend en paramètre un pointeur de pointeur de la structure *Unites Unites \*\*unite* et un pointeur d’entier int \*compteurUnite decompteur qui correspond au nombre de cases du tableau. Cette fonction parcourt le tableau de la structure unité et affiche une unité selon le code qu’il a saisi. Si le code est faux ou l'unité n’existe pas, le message correspondante sera affiché à l'écran. Sinon, l'utilisateur modifie l’information de l'unité trouvée sauf le code qui a été donnée à la création de cette unité.   
 La fonction void supprimerUnite(Unites \*\*unite, int \*compteurUnite); enlève une unité du tableau dynamique selon le code saisi par l’utilisateur. Elle ne retourne aucun valeur et elle prend en paramètre un pointeur de pointeur de la structure *Unites Unites \*\*unite* et un pointeur d’entier int \* compteurUnite qui indique le nombre de cases du tableau et qui sert à parcourir ce-dernière. Après la suppression, la mémoire et réallouée de nouveau et tous les éléments du tableau sont bougés pour éviter des trous.   
  


La fonction void unitesStatiques(Unites \*\*unite, int \*iCompteurUnite); ne retourne aucun valeur et prend en paramètre un pointeur de pointeur de la structure *Unites* et un pointeur d’entier int \*iCompteurUnite . Le but de cette fonction est initialisation de quelques unités statiques pour faciliter les testes du logiciel. De cette façon, en lançant le logiciel il y aura déjà quelques données prédéfinies.