RAPPORT  
TECHNIQUES DE PROGRAMMATION  
LOGICIEL D’AIDE À LA GESTION DES SERVICES DE SECOURS  
GOLOVAN Mykola et LACROIX Rémi  
GROUPE 2 (B)

Introduction

Les structures mises en place

Pour pouvoir bien structurer et gérer le développement du logiciel, nous avons décidé de séparer la structure correspondante aux alertes, la structure correspondante aux unités et toutes les données nécessaires qui traitent chacune d’elles. Autrement dit, nous avons modulé deux fichier d’interface *alerte.h* et *unite.h*.   
   
***Le fichier d’interface alerte.h***

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

#ifndef ALERTE\_H

#define ALERTE\_H

#include <stdio.h>

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

#endif

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

typedef struct {

int iCode;

char cType[TAILLE\_BUFFER];

char cNiveau[TAILLE\_BUFFER];

char cLieu[TAILLE\_BUFFER];

int iNombreVictimes;

char cDescription[TAILLE\_BUFFER];

int iEstTraiteParUnite;

int iCodeUniteQuiTraite;

} Alerte;

Nous avons déclaré au total 8 membres de la structure *Alerte.* Selon le cahier de charges, chaque alerte 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 cType[TAILLE\_BUFFER];

char cNiveau[TAILLE\_BUFFER];

char cLieu[TAILLE\_BUFFER];

char cDescription[TAILLE\_BUFFER];

sont pour but de stocker information contenant le type, niveau, lieu et description d’alerte. Ils ont une TAILLE\_BUFFER qui contient 150 cases vides pour les chaînes de caractères. Ensuite, il un entier int iNombreVictimes qui stock le nombre de victimes et deux autres entiers :

int iEstTraiteParUnite;

int iCodeUniteQuiTraite;

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 traitee par une unite int iEstTraiteParUnite;. Et une autre variable qui stock le code d'unité qui traite une alerte donnée.

La liste de fonctions

Nous avons déclaré 11 fonctions au total qui permettent de traiter une alerte :

void ajouterAlerte(Alerte \*\*alerte, int \*compteurAlerte);

void afficherUneAlerte(Alerte \*\*alerte, int \*compteurAlerte);

void afficherToutesAlertes(Alerte \*\*alerte, int \*compteurAlerte);

void modifierAlerte(Alerte \*\*alerte, int \*compteurAlerte);

void supprimerAlerte(Alerte \*\*alerte, int \*compteurAlerte);

char \* typeAlerte();

char \* niveauAlerte();

int genererCodeAlerte();

void alertesStatiques(Alerte \*\*alerte, int \*iCompteurAlerte);

void imprimerLesAlertes(Alerte \*\*alerte, int \*iCompteurAlerte);

void chargerLesAlertes(Alerte \*\*alerte, int \*iCompteurAlerte);

La fonction ajouterAlerte(Alerte \*\*alerte, int \*compteurAlerte); sert pour la création d’une nouvelle alerte. Elle de type void et donc elle ne retourne pas de valeurs. Elle prend en paramètre un pointeur de pointeur de la structure *Alerte* Alerte \*\*alerte et un pointeur de type entier int \*iCompteurAlerte qui décrits le nombre de cases total dans le tableau dynamique d’*Alerte* 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 alertes. Ces sont les fonctions utilisées à l'intérieur de la fonction :

char \* typeAlerte();

char \* niveauAlerte();

int genererCodeAlerte();

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 \*typeAlerte(); et \*niveauAlerte(); retournent un pointeur de type de chaîne de caractères selon la choix de l'utilisateur.  
 Les fonctions

void afficherUneAlerte(Alerte \*\*alerte, int \*iCompteurAlerte);

void afficherToutesAlertes(Alerte \*\*alerte, int \*iCompteurAlerte);

servent pour l’affichage des alertes. En utilisant un entier int \*iCompteurAlerte ces fonctions parcourent le tableau de la structure *Alerte* 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 *Alerte* et un entier int \*iCompteurAlerte qui sert pour décrire la quantité de cases dans le tableau dynamique. Dans la première fonctions la recherche est basée sur le code donnee par l’utilisateur alors que la deuxième fonctions affiche toutes les alertes du tableau.

La fonction void modifierAlerte(Alerte \*\*alerte, int \*compteurAlerte); retourne aucun valeur et prend en paramètre un pointeur de pointeur de la structure *Alerte* Alerte \*\*alerte et un pointeur d’entier int \*compteurAlerte decompteur qui correspond au nombre de cases du tableau. Cette fonction parcourt le tableau de la structure alerte et affiche une alerte selon le code qu’il a saisi. Si le code est faux ou l’alerte n’existe pas, le message correspondante sera affiché à l'écran. Sinon, l'utilisateur modifie l’information de l’alerte trouvée sauf le code qui a été donnée à la création de cette alerte.   
 La fonction void supprimerAlerte(Alerte \*\*alerte, int \*compteurAlerte); enlève une alerte 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 *Alerte* Alerte \*\*alerte et un pointeur d’entier int \*compteurAlerte 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 alertesStatiques(Alerte \*\*alerte, int \*iCompteurAlerte); ne retourne aucun valeur et prend en paramètre un pointeur de pointeur de la structure *Alerte* et un pointeur d’entier int \*iCompteurAlerte. Le but de cette fonction est initialisation de quelques alertes statiques pour faciliter les testes du logiciel. De cette façon, en lançant le logiciel il y aura déjà quelques données prédéfinies.