GIF 1301 Langage C

Table des matières

I. Portée du projet

1. Objectif du projet

On souhaite réaliser un logiciel permettant de suivre et faire évoluer l'état d'un réseau électrique. On dispose de centrales qui délivrent une certaine capacité d'électricité à des villes qui la consomment.

Notre logiciel devra permettre à l'utilisateur d'ajouter ou de retirer des centrales et des villes. D'ajouter ou de retirer des lignes électriques entre une centrale et une ville. Il sera aussi nécessaire de connaître la puissance distribuée (en kW) pour chacune des centrales du réseau ainsi que la puissance fournie à chaque ville du réseau. De plus, il devra être possible d'enregistrer le réseau construit ou d'en charger un précédemment enregistrer.

2. Contraintes

Nous devrons utiliser une liste doublement chaînée pour les centrales. Les villes seront mémorisées dans une liste simplement chaînée. A chaque centrale, il sera attaché une liste simplement chaînée de toutes les villes qu'elle alimente avec la puissance qu'elle leur met à disposition (cf. Figure 3.

Les structures de données nous ont été données en début de projet afin que nous les suivions avec tout de même la possibilité d'y ajouter quelques éléments.

La structure ville contient trois informations (liste simplement chaînée) :

- Le code postal de la ville (entier naturel)
- Le nom de la ville (chaîne de 50 caractères)
- Un pointeur sur une structure Ville suivante (type PTville)

En voici la déclaration en Figure 1 :

```
//////////////////////////////////// Liste des villes ////////////////////////////
    typedef struct ville
    {
        int codePostal;
        char nomVille[50];
        struct ville * villeSuivante;
    } Tville;

    typedef Tville * PTville;
```

Figure 1 : Structure des villes

La structure centrale contient elle cinq informations (liste doublement chaînée) :

- Le code de la centrale (entier)
- Un pointeur sur la première ligne électrique dépendante (type PTLigneElectrique)
- Un pointeur sur le bidon des ligne électrique (type PTLigneElectrique)
- Un pointeur sur la structure centrale précédente (type PTcentrale)

- Un pointeur sur la structure centrale suivante (type PTcentrale)

En voici la déclaration en Figure 2 :

```
//////////////////////////////////// Liste des centrales ////////////////////////////
    typedef struct centrale
    {
        int codeCentrale;
        PTligneElectrique villeDependante; // Pointeur sur la liste des lignes
        // Pointeur sur le bidon des lignes electriques de la centrale.
        // pfinLigne NE DESSER PAS DE VILLES !!!
        PTligneElectrique pfinLigne;
        struct centrale * ptsuivant;
        struct centrale * ptprecedent;
    }Tcentrale;

    typedef Tcentrale * PTcentrale;
```

Figure 2 : structure des centrales

Les structures lignes électriques contiennent trois informations (liste doublement chaînée) :

- La puissance attribuée à la ville en kW (entier)
- Un pointeur sur la structure de la ville desservie (type PTville)
- Un pointeur sur la structure ligne électrique suivante (type PTLigneElectrique)

En voici la déclaration en Figure 3 :

```
//////////////////////////////////// Liste des lignes electriques /////////////
    typedef struct lignesElectrique
    {
        int puissance;
        PTville villeDesservie;
        struct lignesElectrique * ligneSuivante ;
    } TlignesElectrique;

    typedef TlignesElectrique * PTligneElectrique;
```

Figure 3 : structure des lignes électriques

II. Analyse du projet

1. Simplifications possibles

Bien que les trois structures de données (ville, centrale et lignes électriques) soient toutes les trois différentes, nous avons pu optimiser notre code en reprenant et en réadaptant certains sous-programmes. Par exemple, la fonction de l'affichage des lignes électriques par rapport aux villes et l'affichage par rapport à la liste des centrales, proviennent d'une unique fonction réadaptée pour chacune.

2. Solution existante

Plusieurs solutions étaient à notre disposition pour le développement du projet demandé.

L'utilisation de tableaux de structures auraient pu être envisagés, mais demandent une taille initialement définie.

Voilà pourquoi il est préférable d'utiliser des listes chainées pour ce projet.

III. Conception

1. Découpage du projet

Pour ce projet, nous nous sommes réparti les tâches en découpant le projet en plusieurs fonctions.

Pour travailler à deux sur le code et pouvoir coder simultanément, nous avons utilisé la plateforme GitHub. GitHub nous a permis de déposer nos codes sur un espace partagé dédié au projet et de les développer dans une autre application comportant un débugger et un compilateur comme DevC++, on a donc un fichier.c principal. A chaque nouvelle étape ou sous-programme à développer, la personne du binôme devant travailler dessus créait une « branche » (nouveau fichier.c à partir du programme principal) dédié à la modification. Un fois le code secondaire développé, c'est l'autre personne du binôme qui avait à observer les modifications et à remplacer le programme principal par ce-dernier. Nous pouvions aussi développer une branche ou plusieurs chacun de notre côté et le logiciel faisait automatiquement la mise en commun des programmes. Enfin, nous avons aussi pu développer sur le logiciel « Visual Studio Code » si nous voulions travailler en simultané sur une même partie du projet.

2. Principe de fonctionnement

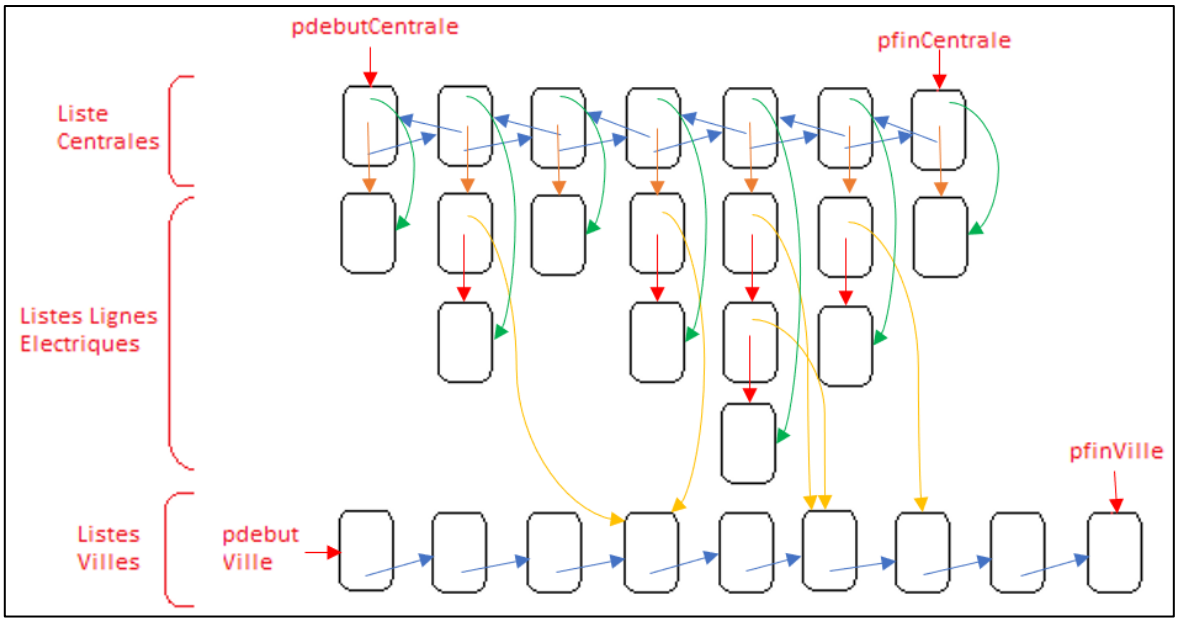

Figure 4 : représentations des maillons d'un réseau et de leurs pointeurs

Le réseau que nous avons développé comprend une liste doublement chaînée de centrales ainsi qu'une liste simplement chaînée de villes (cf. Figure4, flèches bleues). Chacune de ces deux listes s'est vu être attribuée des bidons de début et de fin de liste, simplifiant la gestion de ces dernières. Ainsi, les pointeurs « pfinVille » et de « pfinCentrale » pointant sur une structure, ont un pointeur suivant nul. Même représentation pour le bidon « pdebutCentrale » aillant un pointeur vers une structure le précédent de valeur nulle.

Chaque centrale faisant partie de la liste, pointe vers une ligne électrique (cf. Figure 4, flèches orange) elle-même pointant sur une ville du réseau (cf. Figure 4, flèches jaunes) en plus de pointer sur la ligne suivante (cf. Figure 4, flèches rouge) si ce n'est pas la dernière. Pour faciliter la gestion des lignes électriques, on utilise ici encore des représentations de bidons. Ainsi, chaque centrale pointera vers le dernier élément de sa liste de ligne électrique (cf. Figure 4, flèches vertes). Celui-ci ne desservira aucune ville et aura un pointeur ligneSuivante NULL.

Il faut préciser que pour faciliter la gestion du réseau, nous avons fait le choix d'utiliser les codePostal/codeCentrale des maillons pointé par les pointeurs pDebut. En effet, nous avons mis à l'intérieur le nombre d'élément que la liste possède.

Par exemple, s'il y a 8 villes dans le réseau, le « pdebutVille->codePostal = 8 ».

Pour cela nous avons dû veiller à bien faire attention de mettre à jour cette information lors des insertions et des suppressions.

Remarque : nous avons fait la même chose pour toutes les listes poubelles

3. Description des éléments

Une liste des sous-programmes du code, classé par ordre de déclaration dans le code est présente en annexe (voir Tableau 1).

a. Création lignes électriques

La création des lignes électriques est l'une des parties qui nous a posé le plus de problème. En effet, au début nous n'étions pas au clair sur l'agencement de ces dernières. Pour être plus précis, la création des lignes électriques n'était pas en accord avec les manipulations que l'on faisait dessus. La difficulté venait du fait que, d'une part, les listes de lignes électriques ne possédaient pas de bidons de début et de fin de liste et d'autre part, il y a plusieurs listes de ligne électrique au sein du réseau. On ne peut donc pas se permettre de déclarer un pointeur pour chacune de ces listes puisqu'il peut y en avoir autant que le nombre de centrale sur le réseau.

Ainsi pour clarifier un peu les choses nous avons fait le choix d'ajouter un bidon signant la fin d'une liste de ligne électrique dans la structure des centrales (cf. Figure 4). Pour rappelle ces pointeurs sur les bidons d'une fin de liste de lignes électriques d'une centrale a pour nom « pfinLigne » et il est représenté par les flèches verte en Figure 4.

D'ici on a pu enfin commencer à créer les lignes électriques.

Pour commencer on a d'abord créé la fonction « PtPreLigne » (cf. Tableau 1, fonction n°19). Puis nous avons créé la fonction « InsertionLigneElec » (cf. code, ligne 389).

Celle-ci prend en paramètre la centrale sur laquelle on veut insérer une nouvelle ligne électrique et un structure poubelle qui va nous permettre de réutiliser les éléments précédemment jetés.

On commence par déclarer un pointeur sur une ligne électrique « pL ».

Puis on regarde s'il n'y a rien dans la poubelle des lignes électriques. Si c'est le cas on fait un malloc() (allocation de mémoire) sur « pL ». Sinon, on pointe « pL » sur un élément de la poubelle des lignes électriques et on l'enlève de la poubelle (cf. code, lignes 393 à 399).

Ensuite il faut regarder si la centrale « pC » à déjà des lignes électriques. Pour cela on compare le pointeur sur la première ligne électrique (villeDependante) et celui sur le bidon des lignes électriques (pfinLigne) de la centrale « pC » (cf. code, ligne 402).

- S'ils sont différents cela veut dire que la centrale a déjà des lignes électriques. On peut alors utiliser la fonction « PtPreLigne » qui va renvoyer le pointeur se trouvant avant « pfinLigne » et le mettre dans « pPrePL ». On pourra alors faire pointer le « ligneSuivante » de « pPrePL » sur « pL » (cf. code lignes 404 et 405).
- Sinon, cela veut dire que les pointeurs sur la première ligne électrique (villeDependante) et celui sur le bidon des lignes électriques, pointent sur le même objet. Dans ce cas, la nouvelle ligne sera pointée par « villeDependante » (cf. code lignes 408).

Après ça on a plus qu'à mettre le pointeur « ligneSuivante » de « pL » sur « pfinLigne » et renvoyer « pL » (cf. code, ligne 410).

Remarque : On met le pointeur « villeDesservie » de « pL » à NULL par simple précaution (cf. code, ligne 411). Cela n'est pas forcément nécessaire puisque nous le mettons sur la ville à desservir juste après (en dehors de la fonction).

Il faut maintenant crée une fonction qui entre la puissance de la ville que la ligne électrique dessert et qu'elle pointe sur cette dernière.

Pour cela nous avons créé la fonction « CreationLigneElec » (cf. code, ligne 676). Elle prend en paramètre tous les bidons des listes de ville et de centrale ainsi que le code central duquel part la ligne électrique, le code postal de la ville à desservir et la puissance qu'on lui dessert. Il y a aussi une structure poubelle car dans la fonction « CreationLigneElec » on appelle la fonction « InsertionLigneElec ».

On commence par déclarer trois pointeurs, un pour chaque type de structure. On récupère ensuite le pointeur sur la centrale avec le code centrale correspondant grâce à la fonction « NumCentrale » (cf. Tableau 1, fonction n°14). Après on ajoute une ligne électrique à cette centrale à l'aide de « InsertionLigneElec ». Cette nouvelle ligne électrique sera directement pointée par un pointeur « pLigne » (cf. code, ligne 685). On va ensuite mettre la bonne puissance et pointer sur la ville à desservir (cf. code, lignes 687 et 688).

b. Réutilisation de la mémoire

Précédemment, nous avons mentionné les poubelles à plusieurs reprises. Elles ont été implémenter dans le code afin d'avoir une gestion beaucoup plus propre de la mémoire. Car cela permet de retrouver les éléments supprimés et de réutilisé l'espace mémoire qu'ils occupent.

Pour mettre en place cette poubelle, nous avons créé une structure contenant tous les bidons des listes poubelles (cf. Figure 5) :

```
////////////////////////// Structure contenant tout les bidons des listes poublelles //////////////
typedef struct ben
{
    PTville pdebutV;
    PTville pfinV;
    PTcentrale pdebutC;
    PTcentrale pfinC;
    PTligneElectrique pdebutL;
    PTligneElectrique pfinL;
}Tben;

typedef Tben *PTben;
```

Figure 5 : structure de la poubelle

Nous avons fait cela afin de réduire considérablement le nombre de paramètre à entrer dans une fonction de suppression. Mais aussi pour les fonctions d'insertion puisqu'on met aussi la poubelle en paramètre dans ces fonctions. Si on n'avait pas fait ça, les paramètres de la fonction « SupprimerReseau » (cf. Tableau 4, fonction n°26) serait :

(PTcentrale pdebutC, PTcentrale pfinC, PTville pdebutV, PTville pfinV, PTville BENpdebutV, PTville BENpfinV, PTcentrale BENpdebutC, PTcentrale BENpfinC, PTligneElectrique BENpdebutL, PTligneElectrique BENpfinL)

Mais grâce à la structure « ben », les paramètre en rouge ci-dessus peuvent simplement être remplacés par : « PTben BEN ».

Il faut ensuite allouer à l'ensemble de la structure un espace mémoire dans le main() (cf. code, lignes 992 à 999). Puis comme pour les listes du réseau, il faut initialiser correctement les différents bidons de poubelle (cf. code, lignes 1001 à 1015).

Ainsi quand on veut, par exemple, accéder au pointeur du début de la liste de la poubelle des villes il nous suffit de faire « BEN->pdebutV ».

Pour crée les fonctions de suppression, il nous suffit de mettre l'élément à supprimer dans la poubelle correspondante. Prenons l'exemple des villes où il faut, bien sûr, mettre à la poubelle la ville en question mais aussi toutes les lignes électriques pointant sur cette ville.

Pour cela, on va mettre en paramètre les bidons de la liste des villes, ceux de la liste des centrales puisqu'il va falloir les parcourir pour enlever des lignes électriques, le code postal de la ville à supprimer et la structure poubelle.

Pour commencer on déclare les pointeurs et variables nécessaires et on fait pointer « pV » sur la ville à supprimer à partir du code postale (cf. code, ligne 522 à 525).

Ensuite on enlève la ville de la liste des villes en s'occupant du maillon précédent et en mettant à jour le nombre d'élément dans la liste. Pour cela, on décrémente de 1 le code postal du bidon de début de liste des villes (cf. code, lignes 528 à 530).

Puis on va chercher toutes les lignes qui pointent sur la ville à supprimer en faisant une double boucle while. L'une va parcourir la liste des centrales tant que le pointeur « pC » sera différent du bidon de fin de liste des centrales (cf. code, lignes 533 à 543). Et l'autre boucle va parcourir les lignes électriques de la centrale « pC » tant que « pLx » sera différent du bidon de fin des lignes électriques de la centrale « pC » (cf. code, lignes 536 à 541). Pour chaque passage de boucle on vérifie si la ville que dessert la ligne électrique « pLx » est la même que la ville qu'il faut supprimer. Si c'est le cas, on met la ligne électrique dans la poubelle en la déliant proprement de sa liste grâce à la fonction « SupLigne » (cf. code, ligne 474). Ici on utilise deux pointeurs pour parcourir les lignes au lieu d'un seul afin de ne pas se retrouver à parcourir la poubelle. En effet, « pLy » permet de garder en mémoire la position de la ligne électrique à laquelle on était rendu lorsque l'on parcourt les lignes électriques de la centrales « pC ».

Après avoir mis à la poubelle toutes les lignes électriques qu'il fallait, il ne reste plus qu'à insérer la ville à supprimer dans la poubelle des villes en pensant bien à incrémenter de 1 le code postal du bidon de début de liste des villes poubelle (cf. code, lignes 547 à 549).

c. Principe de fonctionnement des sélections d'objets

On stocke pour les listes centrales et villes le nombre d'éléments de la liste dans le bidon de début. Ainsi, l'information du nombre de villes enregistrées dans le réseau est contenue dans le code postal de la structure pointée par le pointeur pdebutVille. L'information s'incrémente à chaque insertion d'une ville dans la fonction InsertionVille et on lui soustrait un à chaque passage par la fonction SupVilles (cf. Tableau 1, fonction n°25).

Même principe pour la liste des centrales, ou le nombre de centrales du réseau est inscrit dans le code centrale de la structure pointée par le pointeur pdebutCentrale.

Ces informations stockées permettent d'avertir l'utilisateur que le réseau est vide s'il veut par exemple supprimer un élément ou créer une ligne électrique.

On utilise des fonctions de sélection des villes et des centrales pour la suppression de centrales, villes, lignes électriques ainsi que pour l'affichage des lignes électriques, permettant à l'utilisateur de sélectionner l'objet à partir des flèches du clavier.

Dans les programmes de suppressions de villes (cf. code, lignes 1099 à 1125), d'ajout de lignes électriques (cf. code, lignes 1146 à 1187) et dans la fonction AfficherLignesVilles (cf. code, lignes 752 à 816), on fait intervenir la fonction SelectVille prenant en arguments les deux bidons de la liste des villes. Dans cette fonction (cf. code, lignes 632 à 673), on affiche premièrement un cadre avec la fonction cadre() d'une taille et de position définie. La fonction pose un pointeur sur pdebutVille->villeSuivante et demande à l'utilisateur un caractère avec la fonction lireCaract() tant que la touche pressée n'est pas 'Entrer' de code ASCII '13', dans ce cas elle renverra le code postal de la ville. Si la touche pressée est la flèche vers la gauche '<' (code ASCII '475'), le pointeur pointe sur la ville précédent la ville actuelle grâce à la fonction ptPreVille(), uniquement si cette ville n'est pas pointée par le pointeur pdebutVille. Manipulation inverse avec la touche '>' (code ASCII '477'). Enfin, la touche échap, de code ASCII '27' permet-elle de renvoyer '-1', signifiant au programme qu'il faut annuler la suppression.

Le fonctionnement est très proche pour la sélection de centrales avec la fonction SelectCentrale.

d. Enregistrement du réseau dans un fichier texte

Il s'est présenté comme indispensable le fait de pouvoir charger ou enregistrer un réseau, permettant à un utilisateur de sauvegarder son avancée dans la configuration / gestion de son réseau électrique, de le partager ou d'en charger un deuxième.

Pour cela, nous avons défini une structure de fichier texte (fichier.txt) permettant le bon chargement / enregistrement d'un réseau.

On sépare le code postal et le nom d'une ville par une tabulation, puis on effectue un retour à la ligne pour différencier chaque ville.

La liste des villes et celle des centrales électriques sont séparées par un « $ ». On différencie ensuite les centrales et leurs lignes électriques par de « # » (cf. Figure 6).

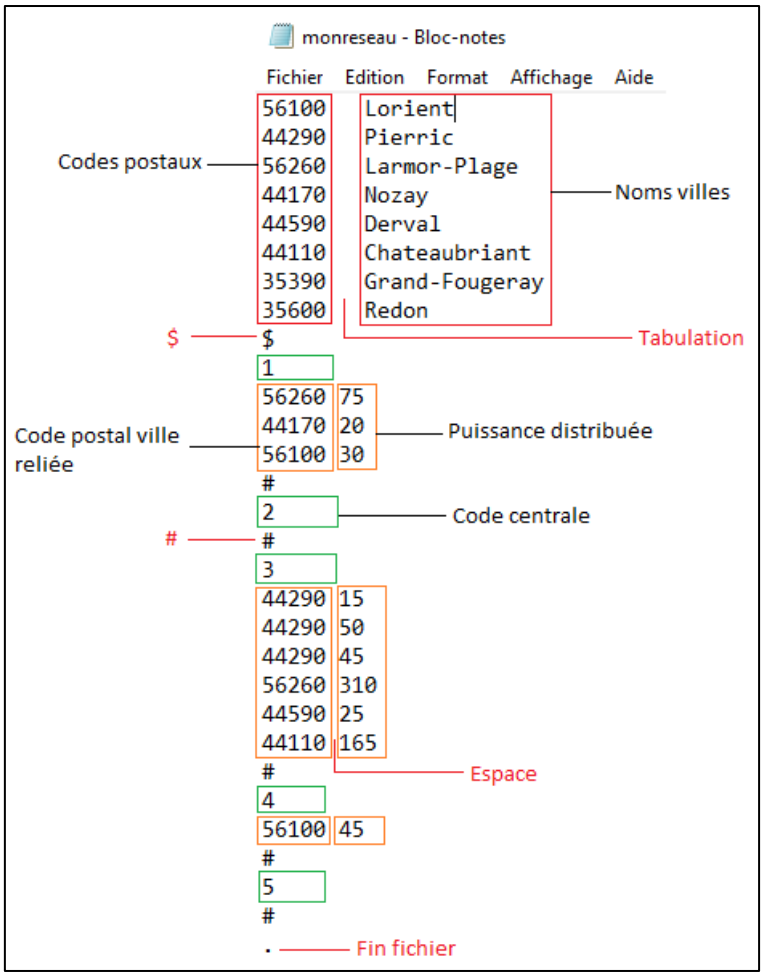

Figure 6 : structure du fichier texte

Pour enregistrer un fichier nous avons utilisé la fonction se trouvant à la ligne 911 du code. Celle- ci prend en paramètre tous les bidons de liste du réseau avec le nom du fichier que l'utilisateur à choisi.

Après avoir déclaré toutes les variables nécessaires, on commence par voir si le nom de fichier qu'a entré l'utilisateur existe déjà en essayant de l'ouvrir en mode lecture. Si on ne parvient pas à l'ouvrir en mode lecture, cela veut dire qu'il n'existe pas. On peut donc l'ouvrir en mode écriture et commencer l'enregistrement (cf. code, lignes 919 à 922).

Ensuite on peut commencer à enregistrer nos villes en parcourant la liste des villes et en écrivant leurs informations au bon endroit comme indiqué en Figure 6 (cf. code, lignes 924 à 926).

Remarque : Le caractère « $ » indique que l'enregistrement des villes est terminé. Donc s'il n'y a pas de villes on entre directement « $ » (cf. code, ligne 928).

Après les villes vient les centrales (cf. code, lignes 930 à 939) et leurs lignes électriques (cf. code, lignes 934 à 936) séparé du caractère « # » (cf. code, ligne 938). Ici on entre d'abord le code de la centrale puis on entre ses lignes électriques (en indiquant le code postal de la ville desservie et la puissance) si elle en possède.

Pour finir on met un point pour dire que le fichier est terminé et on ferme le fichier.

e. Chargement du réseau dans un fichier texte

La fonction « Enregistrer » ne sert à rien si elle n'est pas accompagnée de la fonction « Charger » (cf. code, ligne 822) qui elle va permettre de charger un réseau déjà enregistrer.

La fonction « Charger » va consister à récupérer un par un les caractères dans le fichier (grâce au « fscanf ») afin de les analyser un par un et de les utiliser pour recrée le réseau enregistrer.

Elle va prendre en paramètre tous les bidons de liste du réseau, le nom du fichier que l'utilisateur souhaite charger et une structure poubelle permettant de recycler (lors des insertions) les éventuels éléments déjà présents dans la poubelle.

Comme toujours on commence par définir les variables nécessaires. On essaye d'ouvrir en mode lecture le fichier qui a pour nom « nomFichier » et on regarde s'il existe (cf. code, lignes 831 à 833).

On ajoute une première boucle « while » qui regardera si le caractère actuel est la fin du fichier et une autre se trouvant à l'intérieur de cette dernière qui regarde si le caractère actuel est celui de la fin de déclaration des villes (cf. code, lignes 836 à 838). Si ce n'est pas le cas, on commence par insérer une nouvelle ville (cf. code, ligne 839) puis on lit le code postal de la ville en indiquant dans une boucle while, le caractère à la fin des code postales (cf. code, ligne 841 à 846). Puis, on converti la chaine de caractère en entier et on la met dans le codePostal de la ville insérée (cf. code, ligne 849). On fait la même chose avec le nom de la ville mais ici on n'a pas besoin de convertir la chaine de caractère récupéré car le nom de la ville est lui aussi une chaine de caractère. On finit par mette un dernier « fscanf » dans cette boucle « while » pour récupérer le caractère se trouvant après le saut de ligne (cf. code, ligne 859).

Ensuite avant de passer aux centrales, il ne faut pas oublier de mettre le caractère actuel au caractère en dessous de « $ » (fin des villes). Pour cela on fait deux « fscanf » (cf. code, lignes 861 et 862). Puis avant d'insérer une centrale on regarde si le caractère actuel n'est pas un « . » grâce à une boucle while. Si ce n'est pas le cas on insert une centrale et on lit le code de la centrale. Comme pour le code postal des villes, on indique le caractère de fin de codeCentrale dans une boucle « while » pour récupérer le code de la centrale afin de le mettre dans la centrale fraichement insérée (cf. code, lignes 865 à 873). Après ça on regarde s'il y a des lignes en récupérant le caractère actuel et en le comparant (grâce à un « while » car il peut y avoir plusieurs lignes électriques) au caractère « # » qui sépare les centrales (cf. code, ligne 875 et 877). Si ce n'est pas un « # » on récupère les infos de la ligne électrique et on les met dans des variables prévues à cet effet (cf. code lignes 879 à 894). On peut alors appeler la fonction « CreationLigne » qui va s'occuper d'insérer la ligne électrique comme il faut (cf. code, ligne 895).

Une fois qu'on a fini de récupérer les lignes électriques, ça veut dire que le curseur de lecture du fichier est sur un « # ». Il faut donc passer au caractère du dessous en faisant deux « fscanf » (cf. code, ligne 897).

Si c'est un point on ferme le fichier et on sort de la fonction charger. Sinon on recommence.

4. Interface utilisateur

Nous avons configuré l'IHM (Interface Homme Machine) en différents sous-menus, pour aérer l'affichage. On demande à l'utilisateur de sélectionner le domaine de l'action à exécuter (centrales, villes, lignes électriques ou la gestion du réseau) en pressant les touches '1', '2', '3' ou '4' au clavier. Pour quitter le programme, l'utilisateur doit presser la touche échap.

Chaque commande nous envoie ainsi vers un sous-menu spécifique.

Remarque : Pour sortir d'un sous menu et ainsi revenir à l'affichage du menu principal, l'utilisateur doit appuyer sur la touche échap 'ESC'.

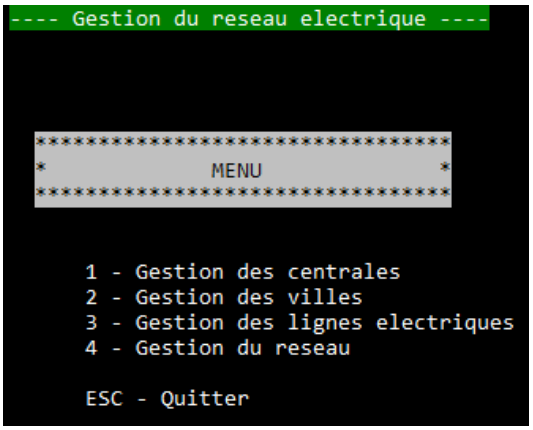

Figure 7 : affichage du menu principal

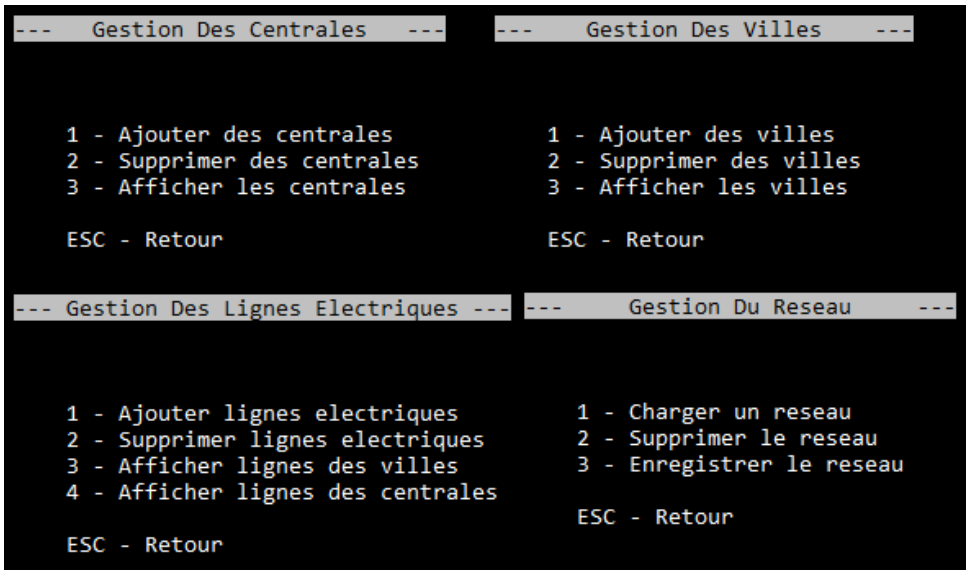

Figure 8 : affichage des sous-menus

Les menus dans le main() du programme sont dans une boucle « while(1) ». L'utilisation de la commande « getch() » demande un caractère à l'utilisateur. Une boucle tant que, redemande un caractère si le caractère rentré n'est pas '1', '2', '3', '4' ou 'ESC' (cf. code, ligne 1024).

L'affichage du réseau électrique (lignes électriques raccordées à une ville / centrale) peut se faire en affichant les lignes propres à une centrale ou en affichants les lignes électriques par rapport à chaque ville, en rentrant respectivement '3' et '4' dans le sous menu « Gestion des lignes électriques » (cf. Figure 10).

On peut alors naviguer d'une ville à une autre ou d'une centrale à une autre et ainsi s'informer sur son intégration au réseau, en se basant sur l'affichage des commandes (voir Figure 8). Ainsi, la navigation sera permise en utilisant les flèches du clavier et on pressera la touche échap 'ESC' pour revenir au menu principal.

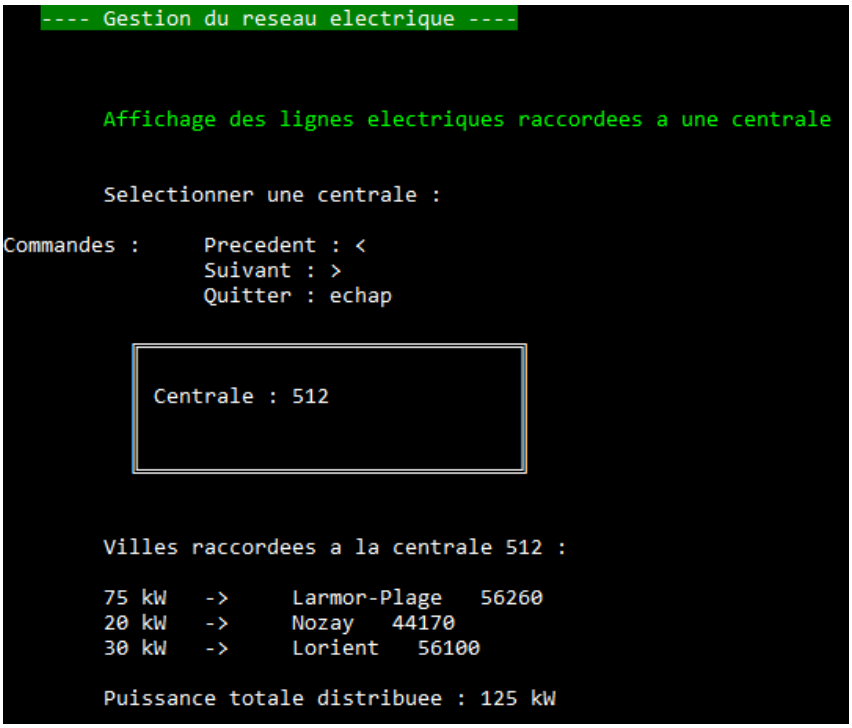

Figure 9 : interface affichage des lignes électriques

Pour toute suppression d'un élément du réseau, intervient la fonction « Validation() » (cf. code, lignes 571 à 582). Cette dernière demande à l'utilisateur de rentrer '0' ou 'o' (oui) pour valider l'action et 'N'ou 'n'(non) dans le cas contraire (cf. Figure 11). Si l'action est validée la fonction renvoie 1 et 0 dans le cas inverse. Même principe pour la suppression complète du réseau électrique.

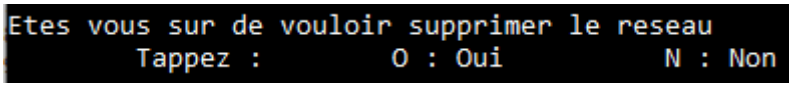

Figure 11 : Validation d'une action

IV. Tests

Des tests ont été réalisés régulièrement pour permettre la bonne avancée du logiciel. Des sauvegardes journalières, nommées par la date de dernière modification du code ont été faites, permettant de pouvoir revenir sur une version de code antérieur à tout moment.

Lors de bugs en phases de test (boucle sans fin, interruption du programme …), nous avons ajouté des commandes « printf() », permettant de connaître la partie du code faisant défaut (une boucle…) et ainsi optimiser notre temps sur la résolution du problème.

V. Résultats

Finalement, au bout de ces 5 semaines de projet, nous arrivons à un code fonctionnel où quelques améliorations restent tout de même à voir.

Le code a été rendu par mail au format .c, de nombreuses références à ce fichier ont été introduites dans ce compte rendu.

Il peut être exécuté directement depuis une application acceptant le langage C et comprenant un compilateur C générant un exécutable (.exe sur Windows, .elf sur Linux), ou bien directement en lançant l'exécutable depuis un terminal de commandes.

Attention, chaque système d'exploitation ne gèrera pas l'affichage de la même façon.

VI. Perspectives

Améliorations envisageables :

- Possibilité de supprimer une unique ligne électrique du réseau
- Navigation dans les menus avec les flèches du clavier, surligner la ligne sélectionnée et valider la sélection par la touche entrée.
- Définir une puissance maximale pour une centrale électrique.
- Définir une puissance maximale à recevoir pour une ville
- Offrir la possibilité à l'utilisateur d'enregistrer son réseau avant d'en charger un autre
- Vérifier que le nom du fichier entré par l'utilisateur se termine par .txt avant de l'enregistrer ou faire l'ajout automatique de l'extension .txt

VII. Conclusion

Ce projet de programmation en binôme autour d'un réseau électrique a été pour le moins passionnant. Il nous a permis d'approfondir et de développer les notions vues en CM / TP, notamment l'usage des pointeurs, des structures et des listes simplement et doublement chaînées. De plus, nous avons à travers ce projet développer des compétences sur le travail en équipe, chose qui n'est pas toujours évidente de travailler à plusieurs, sur un même code en même temps.

Nous avons donc pris soin d'enrichir le code de commentaires sur le fonctionnement de chaque partie, permettant non seulement à la deuxième personne de l'équipe de comprendre l'avancée du code, mais aussi de pouvoir revenir facilement sur le code après quelques jours.

VIII. Annexe

n°	Nom	Paramètres	Description
1	gotoligcol	int lig , int col	Met le curseur à la ligne "lig" et à la colonne "col"
2	ClearConsole	HANDLE hConsole	Effacer tout ce qu'il y a à l'écran
3	cadre	int x, int y, int haut, int larg	Affiche un cadre d'une hauteur "haut", d'une largeur "larg" à partir de la ligne "lig" et de la colonne "col"
4	Affiche	HANDLE hConsole , char c , int fond , int couleur	Affiche le texte dans la console avec une couleur de texte et de fond définie
5	AfficheMenu	HANDLE hConsole	Affiche le menu principale
6	AfficheMenuCentrales	HANDLE hConsole	Affiche le menu de gestion des centrales
7	AfficheMenuVilles	HANDLE hConsole	Affiche le menu de gestion des villes
8	AfficheMenuLignes	HANDLE hConsole	Affiche le menu de gestion des lignes électriques
9	AfficheMenuReseau	HANDLE hConsole	Affiche le menu de gestion du réseau en générale (enregistrer, charger un réseau, supprimer le réseau ...)
10	AfficherCentrales	PTcentrale pDebut , PTcentrale pFin	Affiche la liste de centrale comprise entre les bibons "pDebut" et "pFin"
11	AfficherVilles	PTville pDebut , PTville pFin, int ligne	Affiche la liste de ville comprise entre les bibons "pDebut" et "pFin" à partir de la ligne "ligne"
12	ExistanceCentrale	PTcentrale pDebut , PTcentrale pFin , int num	Renvoie 1 si la centrale avec le codeCentrale = "num" existe dans une liste de centrale donnée. Sinon renvoie 0.
13	ExistanceVille	PTville pDebut , PTville pFin , int cp	Renvoie 1 si la ville avec le codePostale = "cp" existe dans une liste de ville donnée. Sinon renvoie 0.
14	NumCentrale	PTcentrale pDebut , PTcentrale pFin , int num	Renvoie le pointeur sur la centrale qui possède le codeCentrale = "num" dans une liste de centrale donnée. Si la centrale n'existe pas, la fonction renvoie un pointeur NULL.
15	NumVille	PTville pDebut , PTville pFin , int num	Renvoie le pointeur sur la ville qui possède le codePostale = "num" dans une liste de ville donnée. Si la ville n'existe pas, la fonction renvoie un pointeur NULL.
16	PtPreVille	PTville pDebut , PTville pX	Renvoie le pointeur sur la ville qui précède le pointeur sur la ville "pX"
17	InsertionVille	PTville pDebut , PTville pFin , PTben BEN	Renvoie le pointeur sur une NOUVELLE ville se trouvant juste avant "pFin"
18	InsertionCentrale	PTcentrale pDebut , PTcentrale pFin , PTben BEN	Renvoie le pointeur sur une NOUVELLE centrale se trouvant juste avant "pFin"
19	PtPreLigne	PTligneElectrique pDebut , PTligneElectrique pLx	Renvoie le pointeur sur la ligne électrique qui précède le pointeur sur la ligne électrique "pLx".

20	InsertionLigneElec	PTcentrale pC , PTben BEN	Renvoie le pointeur sur une NOUVELLE ligne électrique se trouvant juste avant "pC->pfinLigne"
21	RemplirCentrales	PTcentrale pDebut , PTcentrale pFin , int n , PTben BEN	Ajoute "n" centrales dans la liste se trouvant entre les bidons "pDebut" et "pFin"
22	RemplirVilles	PTville pDebut , PTville pFin, int n , PTben BEN	Ajoute "n" villes dans la liste se trouvant entre les bidons "pDebut" et "pFin"
23	SupLigne	PTcentrale pC , PTligneElectrique pL , PTben BEN	Met dans la poubelle la ligne électrique se trouvant au bout de "pL"
24	SupCentrales	PTcentrale pDebutC , PTcentrale pFinC , int num , PTben BEN	Met dans la poubelle la centrale ayant le codeCentrale = "num" avec les lignes électriques liées à la centrale
25	SupVilles	PTville pDebutV , PTville pFinV , PTcentrale pDebutC , PTcentrale pFinC , int cp , PTben BEN	Met dans la poubelle la ville ayant le codePostale = "cp" avec les lignes électriques pointant sur la ville
26	SupprimerReseau	PTcentrale pdebutC , PTcentrale pfinC , PTville pdebutV , PTville pfinV , PTben BEN	Met dans la poubelle tous les éléments du réseau
27	Validation		Renvoie 1 si l'utilisateur entre "o" ou "O" et 0 si l'utilisateur entre "n" ou "N" sinon il attend que ce soit un de ces 4 caractères
28	lireCaract		Lit un caractère entré au clavier et retourne son code ASCII
29	SelectCentrale	PTcentrale pDebut , PTcentrale pFin	Renvoie la centrale sélectionnée dans le cadre
30	SelectVille	PTville pDebut, PTville pFin	Renvoie la ville sélectionnée dans le cadre
31	CreationLigne	PTcentrale pDebutC, PTcentrale pFinC, PTville pDebutV, PTville pFinV, int numC, int cp, int pow , PTben BEN	Ajoute une ligne électrique (grâce à "InsertionLigneElec") puis met les pointeurs et les variables de la ligne électriques aux valeurs indiquées en paramètre
32	AfficherLignesCentrale	PTcentrale pdebutCentrale, PTcentrale pfinCentrale	Affiche les centrales une à une dans un cadre avec en dessous la liste des villes qu'elle dessers
33	AfficherLignesVille	PTcentrale pdebutCentrale, PTcentrale pfinCentrale, PTville pdebutVille, PTville pfinVille	Affiche les villes une à une dans un cadre avec en dessous la liste des centrales qu'ils l'alimentent

34	Charger	PTcentrale pdebutC , PTcentrale pfinC , PTville pdebutV , PTville pfinV , char nomFichier[100] , PTben BEN	Charge un réseau se trouvant dans le fichier qui a pour nom "nomFichier[100]"
35	Enregistrer	PTcentrale pdebutC , PTcentrale pfinC , PTville pdebutV , PTville pfinV , char nomFichier[100]	Enregistre un réseau dans un fichier qui a pour nom "nomFichier[100]"

Tableau 1 : liste des sous-programmes rangé par ordre de déclaration dans le code