T.P. 6

LE VOYAGEUR DE COMMERCE

Projet réalisé par

MBUMA MONA Nathan

HULUTE-TE-LANGATE

Année universitaire 2018/2019

UPEM

I. Présentation

Le problème du voyageur de commerce décrit une situation qui prend en entrée un ensemble de villes avec leur position géographique, et qui consiste à fournir en réponse un ordre de visite de ces villes qui soit le plus court possible.

II.Fonctionnement

Pour lancer le programme commencer par de zipper le fichier , ensuite ouvrer votre terminal et taper le mot «make» . Saisir «./pvc» sur le même terminal vous avez ainsi lancé le programme voici la page d'accueil:
On peut ainsi visualiser un message qui nous informe de l'état de fonctions . Ce message est un élément important .

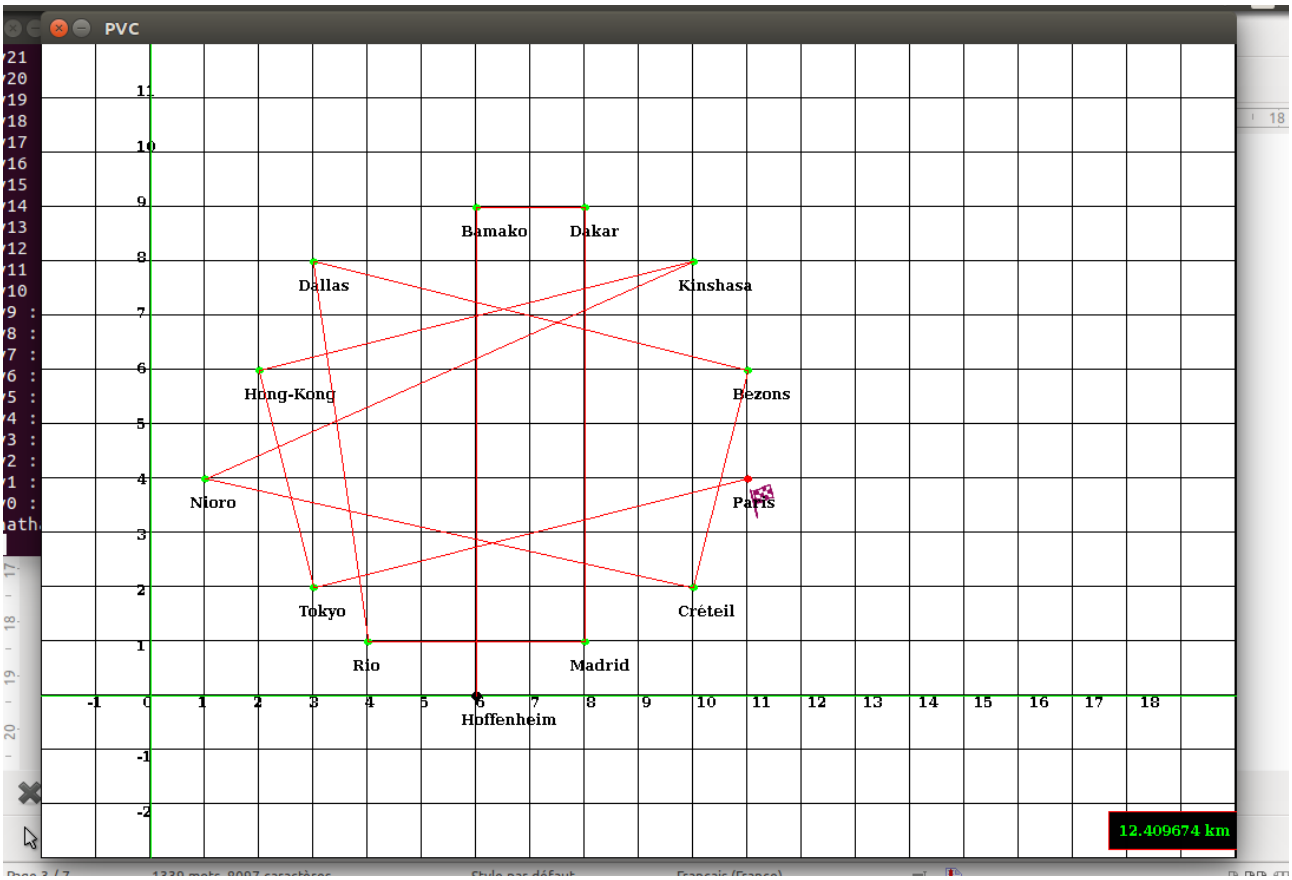
Voici un message positif en cas d'erreur l'exécution s'arrête

Après on affiche le menu l'utilisateur doit faire un choix s'il souhaite charger un fichier ou s'il souhaite créer une carte avec les noms entrés au clavier .

voici le menu d'accueil

Si vous choisissez l'option «charger un fichier» vous devez fournir un nom valide du fichier par exemple «villes» ensuite vous entrez successivement trois entiers attention la somme doit faire cent vingt-huit .
Si vous choisissez l'option «saisir vos villes» vous devez entrer le nom de la ville ensuite cliquer sur l'écran vous avez ainsi créé une ville faîtes l'opération au moins trois fois . Entrer le mot **«stop»** pour arrêter et lancer le programme.

Pour arrêter l'exécution du programme appuyer sur la barre d'espacement



voici un résultat

Cette image a été capturée pendant la recherche du chemin. Le point noir indique le début et le drapeau la fin.
Le **12.409674km** c'est le kilomètre à parcourir

III. Développement

Plusieurs petites fonctions utilisées pour réaliser ce projet nous allons passer en revue quelque fonction importante.
Nous tenons à préciser que ce projet utilise une très bonne gestion d'erreur (assert et structure de contrôle) .

Sans plus tarder voici les structures utilisées dans ce projet.

```
typedef struct ville{
        int x;
        int y;
        char libelle[30];
    }Ville;
```

ce type gère une ville (ses coordonnées et son nom).

```
typedef struct visite{
        Ville **tab;
        int longueur;
}Visite;
```

une visite est un pointeur sur un tableau de ville et la longueur de la visite

```
typedef struct population{
        Visite tab[K];
}Population;
```

Une population est un tableau de visite.

```
typedef struct mut{
    int j, i, l;
}Mutation;
```

une mutation gère trois entiers i ,j, l qui sont indispensable pour réaliser une mutation.

Voici les fonctions utilisées.

void recupererVille(FILE* fic, Ville tab[], int *i);

fonction : **recupererVille**
paramètres : un fichier , un tableau de ville , un pointeur sur un entier
retour : rien
description : fonction qui lit les éléments du fichier fic le place dans le tableau ville et un incrémente le compteur qui compte le nombre des éléments lus dans le fichier. Avec un algorithme très simple une fonction récursive qui fait avancer le curseur du tableau de ville à chaque appel on écrit les coordonnées à la première position.

void recupere_coordonnées(Ville villes[] , int *taille);
fonction : **recupere_coordonnées**
paramètres : un tableau de ville et un pointeur sur un entier
retour :rien
description : cette fonction récupère les données que l'utilisateur entrent (le nom et les coordonnées de ville) remplie le tableau de ville et incrémente la taille . Elle s'arrête dès l'utilisateur entre le nom «**stop**»

Visite générer_visite(Ville tab[],const int taille);
fonction : générer_**visite**
paramètres : un tableau de ville et une taille
retour : une visite
description :renvoie une visite aléatoire générée en fonction du tableau de ville tab

Mutation generateur(const int taille);
fonction : **generateur**
paramètres : une taille
retour : une mutation
description :génère trois entiers nécessaires pour la mutation et réalise les tests nécessaires pour éviter une collision.

void mutation(Visite visit, const int taille);
fonction : **mutation**
paramètres : une visite et une taille
retour : rien
description : réalise la mutation sur la
permutation de villes visit.

void Nouvelle_population(Population* pop, Ville tab[], const int alpha, const int bêta, const int gamma, const int taille);
fonction : **Nouvelle_population**
paramètres : un pointeur sur une population ,un tableau de ville
quatre entiers
retour :rien
description :crée notre nouvelle population en réalisant un ensemble de copies et un ensemble de génération aléatoire par rapport aux entiers alpha, bêta et gamma.

void Nouvelle_populationM(Population pop, const int alpha, const int taille);
fonction : **Nouvelle_populationM**
paramètres : une population et deux entiers
retour :rien
description :Elle poursuit la création de notre nouvelle population en réalisant un ensemble de mutation par rapport à l'entier alpha.

MLV_Color fabriqueCouleur(void);
fonction : **fabriqueCouleur**
paramètre : rien
retour : une couleur

description : cette fonction crée une couleur avec un contraste très faible pour éviter de cacher l'image sur l'arrière plan.

V. Découpage

Nous avons écrit un fichier **Makefile** plus besoin de taper à la main les différentes commandes d'exécution.
Nous avons respecté les règles de découpage , diviser pour mieux régner , pas d'économie d'inclusion et un maximum des fichiers .h dans le .c .
Il y a 11 fichiers affiche.c ,affiche.h ,Population.c, Population.h, Villes.h, Villes.c , Visite.c , Visite.h , test.c , test.h et main.c
- Dans le module **Villes** on gère l'insertion de villes dans le tableau (par lecture du fichier et par récupération de coordonnées)
- Dans le module **Visite** on effectue la création de visite(le parcours de toutes les visites) . On inclut villes.h
-Dans le module **Population** c'est ici qu'on effectue les mutations de visite. On inclut Visite.h et Villes.h
-Dans le module **affichage** on fait appel à toutes les fonctions du projet et on ajoute un message pour chaque appel .On inclut Population.h et Villes.h
-Dans le module **test** on fait un test de quatre fonctions du projet ces fonctions font appel à 90% des fonctions du projet sauf les fonctions qui utilisent le fichier.On inclut tous les modules
- Dans **main.c** on inclut affichage.h et test.h

V. Défis relevé et conclusion

Dans ce T.P j'ai répondu à 100% de questions posées et j'ai même répondu aux questions bonus . Un travail très réussi pas d'erreur de segmentation ,pas de warnings .
J'ai appris beaucoup de chose ce T.P est un puissant exercice sur le célèbre problème du voyageur de commerce j'ai pris beaucoup du plaisir.