**SOMMAIRE**

**INTRODUCTION page 2**

1. **Présentation globale des sujets page 3**
2. **Sujet 1 page 4**

* **Analyse………**
* **Solution algorithmique**
* **Programmation en langage C**

1. **Sujet 2**

* **Analyse****………**
* **Solution algorithmique**
* **Programmation en langage C**

1. **Sujet 3 page 19**

* **Analyse………**
* **Solution algorithmique**

1. **Programmation en langage C**
2. **Sujet 4 page 21**

* **Analyse………**
* **Solution algorithmique**
* **Programmation en langage C**

1. **Sujet 5 page 25**

* **Analyse**
* **Solution algorithmique**
* **Programmation en langage C**

1. **Sujet 6 page 31**

* **Analyse**
* **Solution algorithmique**
* **programmation en langage C**

1. **Sujet 7 page 36**

* **Analyse**
* **Solution algorithmique**
* **Programmation en langage C**

*INTRODUCTION*

*Le Bureau d’Etude Informatique (BE informatique) est un module inscrit dans le programme de formation des étudiants en TS STIC 1*

Dans le cadre de la fin du premier semestre de notre première année de Technicien Supérieur en Science et Technologie de l’Information et de la Communication (STIC) à l’Institut National Polytechnique Felix Houphouët-Boigny (INPHB), il nous est proposé un projet de 3 semaine nous permettant de mettre en pratique nos connaissances en base de programmation.

*Les structures de données consistent à construire un ensemble de bloc contenant plusieurs éléments(données) qui doivent être manipuler au fur et à mesure*

*La langage C est un langage de programmation informatique utilisé dans plusieurs domaines. Ce langage nous a été enseigné au premier semestre, d’où l’intérêt de l’utiliser pour nos résolutions.*

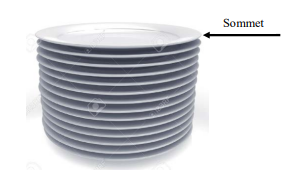
Ayant une passion commune pour l’informatique notre groupe composé de HOUEDANDE Yves, KARIDIOULA Sie, TOURE Mamadou et de CODJEAU Neloum, a saisi l’opportunité d’exploiter cet intérêt commun au travers de ce projet dont l’ébauche à soumettre à M.KPO LOUA.

*.*

1. Présentation globale des sujets

Les sujets qui nous ont été attribué seront traités à partir de structures de données. Les sujets traitent des cas particuliers dans la programmation qui sont les listes chainées, les piles et les files.

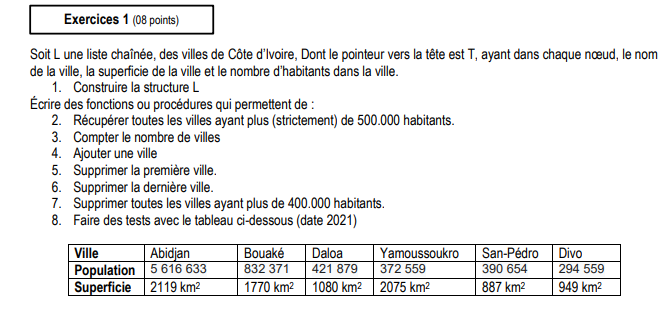
* Une liste chaînée est un ensemble de cellules(blocs) contenant des informations reliées l’une à la suite de l’autre, formant ainsi une liste. Une liste chaînée est donc une structure de données dans laquelle les éléments sont rangés linéairement.
* Une pile (stack, en anglais) est une suite d’éléments superposés les uns au-dessus des autres comme une suite d’assiettes superposées. L’on ne peut y accéder que par le sommet.



* Une file est un ensemble de valeurs qui a un début (Début) et une fin (Queue). Parlant de file, l’on peut faire allusion à une file d’attente.



1. **Sujet1**

****

* **Analyse du sujet :**

Dans l’exercice 1, nous sommes amenés à mettre en place une liste chainée L, des villes de la côte d’ivoire dont le pointeur T pointe vers la tête. Chaque nœud contient des informations sur : le nom de la ville, la superficie, et le nombre d’habitants.

Pour la résolution de l’exercice, nous utiliserons pour la programmation modulaire. Pour ce-faire, nous allons utiliser trois fichiers :

* principal.c qui contiendra la fonction main et qui est le fichier principal.
* fonctions.c qui contiendra nos fonctions et procédures.
* prototype.c qui contiendra nos structures et prototypes.

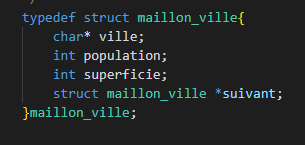
Ce découpage se fera pour l’ensemble des autres sujets qui suivront.

* **Concernant la question 1 :**

La structure L sera représenté par une structure de donnée appelé « maillon\_ville » qui représente chaque nœud avec toutes les informations concernant la population, la superficie, le nombre d’habitants et un pointeur qui nous permettra de mettre en place les différentes liaisons de notre liste chainée.

Le type Enregistrement de « maillon\_ville » en langage algorithmique est le suivant et sa représentation en langage c:

Type enregistrement



ville : chaîne de caractére

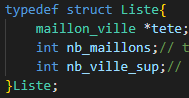
population : entier

superficie : entier

suivant : ^maillon\_ville

Fin enregistrement maillon\_ville;

De plus, nous avons crée une structure de contrôle pour notre liste chaînée. Elle contient les informations essentielles concernant la liste chainée. On y trouve les informations concernant la taille de la liste chaînée, le nombre de ville ayant une superficie supérieur ou égal à 500.000, ce qui nous sera utile au niveau de la question 2, pour la récupération des villes ayant plus de 500.000 habitants.



Type enregistrement

tete : ^maillon\_ville

nb\_maillons : entier

nb\_ville\_sup :

Fin enregistrment

Les champs « tete », « nb\_maillons », « nb\_ville\_sup » représentent respectivement la tête de la liste chainée, le nombre d’éléments de la liste chainée chaînée et le nombre de villes dont la superficie est supérieur à 500.000 habitants.

* **Concernant la question 2 :**

Au niveau de la question 2, il est question de compter le nombre de ville de la liste chainée. Pour cela, nous allons utiliser une boucle « tant que » et variable « temp » qui pour le premier va nous permettre de parcourir la liste en passant de nœud en nœud avec « temp » qui a chaque tour représente le nœud courant. Nous initialisons également dès le début un compteur « compt » que nous incrémentons à chaque nœud . lorsque le champ « suivant » de l’élément courant est Nil, la boucle s’arrête et on retourne la valeur du compteur.

**Solution algorithmique**

|  |
| --- |
| Fonction compter\_nb\_villes(liste :^Liste) :entier  Var  temp :^maillon\_ville  compt:entier  Debut  temp🡨liste.^tete  Tant que temp <> Nil Faire  compt 🡨compt+1  temp = temp^.suivant  FinTanque  Retourner(compt)  Fin |

* ***Programmation en c***

|  |
| --- |
| // permet de compter le nombre de ville  int compter\_nb\_villes(Liste \* liste){      maillon\_ville \*temp = liste->tete;      int compteur = 0;      while(temp != NULL){          compteur++;          temp = temp->suivant; // temp pour dire temporaire:          /\* A chaque tour de boucle, temp prend la valeur du noeud suivant\*/      }      return compteur;    } |

* **Concernant la question 3**

Pour la question 3, il s’agira pour nous de mettre en place une fonction qui retournera un tableau qu’on pourra peut-être réutiliser. Pour la fonction « recuperer\_ville », nous définissons un un pointeur « tab » vers la structure « maillon\_ville » et une variable « taille\_tab » pour enregistrer la taille actuelle. On définit également un pointeur temporaire « temp » pour parcourir la liste chainée. Le principe est que, pour la chaque élément de la liste, si la population est supérieur à 500.000, il est ajouté à la fin de « tab ».Mais si aucun élément ne respecte la condition de population supérieur à 500.000, la fonction va retourner Nil

* **Solution algorithmique**

|  |
| --- |
| Fonction ^recuperer\_ville(liste :^Liste) :  Var  tab :^maillon\_ville  taille\_tab :^entier  temp :^maillon\_ville  Debut  Tab🡨 Nil  Taille🡨0  Temp🡨liste^.tete  Tant que (tem<>Nil) Faire  Si (temp^.population > 500.000) alors  Allouer(tab)  Si (tab = Nil) alors  Ecrire(‘’ Erreur de réalloaction de mémoire ‘’)  Retourner(Nil)  Finsi  Tab[taille\_tab]🡨temp  Taille\_tab🡨taille\_tab+1  Finsi  Temp🡨temp^.suivant  Si(taille\_tab = 0)  Liberer(tab)  Retouner(Nil)  Finsi  liste^.nb\_ville\_sup = taille.tab//on enregitre la taille du tab  //pour la parcourir plus tard  Retourner(tab) |

* **Programmation en Langage C**

maillon\_ville \*recuperer\_villes(Liste \*liste) {

    // Initialisation de variables

    maillon\_ville \*tab = NULL;

    int taille\_tab = 0;

    maillon\_ville \*temp = liste->tete;

    // Parcours de la liste

    while (temp != NULL) {

        if (temp->population > 500000) {

            // Réallocation de mémoire pour le tableau

            tab = realloc(tab, (taille\_tab + 1) \* sizeof(maillon\_ville));

            if (tab == NULL) {

                printf("Erreur de réallocation de mémoire");

                return NULL;

            }

            // Ajout de l'élément à la fin du tableau

            tab[taille\_tab] = \*temp;

            taille\_tab++;

        }

        temp = temp->suivant;

    }

    if(taille\_tab==0){

        free(tab);

        return NULL;

    }

    liste->nb\_ville\_sup = taille\_tab;// on garde une trace sur la longeur de la lsite pour pouvoir la pacourrir ultérieurement.

    return tab;//on retourne le pointeur vers le premier élement de la liste

}

* **Concernant la question 4:**

Pour la question 4, il nous ait demandé d’écrire une fonction qui permet d’ajouter une ville. Pour notre cas, nous faisons le choix d’ajouter les élément au début de notre liste chainée .Pour ce faire, nous utiliserons une procédure qui permettra de le faire . la procédure prendra des données en entrées, c’est-à-dire, le nom de la ville « ville » le nombre de population « population » et la superficie « superficie » Nous prendrons dans notre procédure une fonction une variable « nv\_ville » pour dire nouvelle ville qui nous permettra d’ajouter un nœud .

* **Solution algorithmique**

|  |
| --- |
| Procedure ajouter\_ville(liste :^Liste, ville :chaine de caractéres, population :entier, superficie :entier)   Var  nv\_ville : ^maillon\_ville  Debut  allouer(nv\_ville)  Si (nv\_ville^.ville == Nil) alors  Ecrire(‘’erreur d’allocation ‘’)  nv\_ville^.ville = allouer(chaine de caractéres)  nv\_ville^ .ville = ville  nv\_ville^.population = population  nv\_ville^.supperficie = superficie  nv\_ville^ .suivant = list^.tete  liste^.tete= nv\_ville  liste^.nb\_maillon++  Finsi  Fin  FinProcedure |

* **Programmation en langage c**

void ajouter\_ville(Liste \* liste, char \*ville, int population, int superficie){

    maillon\_ville \*nv\_ville =  (maillon\_ville \*) malloc(sizeof(\*nv\_ville));

    if(nv\_ville == NULL){

        printf("Erreur d'allocation \n");

        exit(1);

    }

    nv\_ville->ville = (char\*) malloc(sizeof(char) \* (strlen(ville) + 1));

    if(nv\_ville->ville == NULL){

        printf("Erreur d'allocation\n");

        free(nv\_ville);

        exit(1);

    }

    strcpy(nv\_ville->ville, ville);

    nv\_ville->population = population;

    nv\_ville->superficie = superficie;

    nv\_ville->suivant = liste->tete;

    liste->tete = nv\_ville;

    liste->nb\_maillons++;

}

* **Concernant la question 5**:
* **Solution algorithmique**

Ayant choisi l’option d’ajouter chaque ville en début , cela signifie pour nous que la première ville est en fait le dernier élément de notre liste. Nous devons donc aller jusqu’au bout de la liste. Pour cela, nous utilisons deux pointeurs « temp » et « prec » et une boucle « tant que ». Notons que à chaque suppression, nous décrementons notre compteur « nb\_maillons » de liste à -1.

* **Solution algorithmique**

|  |
| --- |
| Procedure supprimer\_ville\_1(liste :^Liste)  Var  temp :^maillon\_ville  prec : ^maillon\_ville  Debut  temp🡨liste^.tete  prec🡨Nil  tant que (temp^.suivant<>Nil) Faire  prec🡨temp  temp🡨temp^.suivant  Fintanque  Si(prec<>Nil) alors  Prec^.suivant 🡨Nil  liste^.tete🡨Nil  liberer(temp)  FinDebut  FinProcedure |

* **Programmation en langage C**

// supprimer la premiere ville

void supprimer\_ville\_1(Liste \*liste){

    maillon\_ville \*temp = liste->tete;

    maillon\_ville \*prec = NULL;

    //parcours la liste jusqu'au dernier élément

    while (temp->suivant != NULL) {

        prec = temp;

        temp = temp->suivant;

    }

    if (prec != NULL){

        prec->suivant = NULL;

    }

    else{

        liste->tete = NULL;

    }

    free(temp->ville);

    free(temp);

    liste->nb\_maillons--;

}

* **Concernant la question 6**

La derniere ville ajoutée à la liste sera en début de la liste. Il suffit donc de supprimer le premier élément de notre liste. Pour cela, nous utilisons un pointeur « temp » qui pointe vers la tête de la liste. De plus , après l’opération de suppression, nous décrémentons le champ « nb\_maillons » de notre structure de contrôle « liste ». Pour garder la taille exacte du nombre de maillons ou éléments.

* **Solution algorithmique**

|  |
| --- |
| Procedure supprimer\_ville\_fin(liste :^Liste)  Var  temp :^maillon\_ville  Debut  Si(liste^.tete = Nil) alors  ecrire(‘’ La mémoire est vide ‘’)  Retourner(Nil)  temp 🡨liste^.tete  liste^.tete🡨liste^.tete^.suivant  Liberer(temp)  liste^.nb\_maillons🡨list^.nb\_maillons+1  Fin  FinProcedure |

* **Programmation en c**

//supprimer la derniere ville ajouter

void supprimer\_ville\_fin(Liste \*liste) {

    if (liste->tete == NULL) {

        printf(" La liste est vide\n");

        return;

    }

    maillon\_ville \*temp = liste->tete;

    liste->tete = liste->tete->suivant;

    free(temp->ville);// on libere l'espace allouer en debut pour stocker la chaines de caractère

    free(temp);

    liste->nb\_maillons--;

}

* **Concernant la question 7**:

Le principe est que nous allons parcourir la liste chainée et lorsqu’une ville respecte la condition c’est à dire si sa population est supérieure a 400000 nous la supprimons de la liste chainée tout en maintenant la liste dans son bon ordre en réalisant les liens entre les nœuds dans le cas. Pour se faire nous utiliserons deux pointeurs <<temp>> et « prec » que nous utilisons

La variable « temp » pointe vers la tête.

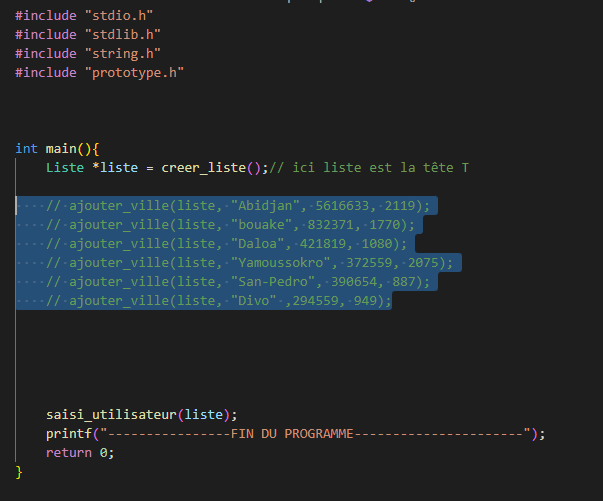
Le nom de la fonction est «sup\_plus\_de\_400milles»

* **Solution algorithmique**

|  |
| --- |
| Procédure sup\_ville\_400\_mille (liste : ^Liste)  Var  Temp : ^maillon\_ville  Prec : ^maillon\_ville  Début  Temp🡨liste^.tête  Prec🡨Nil  Tant que (temp<>Nil)  Si (temp^. population>400000) alors  Si (prec<>Nil) alors  prec^. suivant🡨temp^. suivant  libérer (temp)  liste^.nb\_maillon🡨liste^.nb\_maillon-1  temp🡨prec^.suivant  Sinon  Liste^.tete🡨temp^.suivant  libérer (temp)  liste^.nb\_maillon-1  temp🡨liste^.tete  finsi  sinon  prec🡨suivant  temp🡨temp^. suivant  finsi  fin  finprocedure |

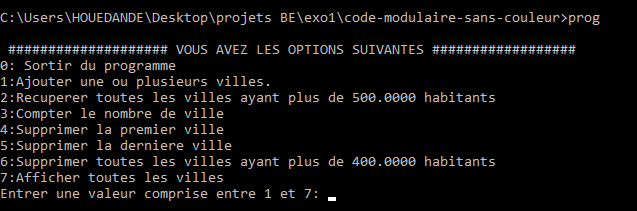
* **Concernant la question 8 :**

Nous avons ajouté d’autres fonctions et procédures qui rendent le programme intérractif avec l’utilisateur. On a une fonction « saisi\_utilisateur » qui permet cette intéractivité et est optionnel. Ainsi, on a la possibilité de l’utiliser comme ne pas l’utilisée en la mettant tout simplement en commentaire dans le programme principal « principal.c ».



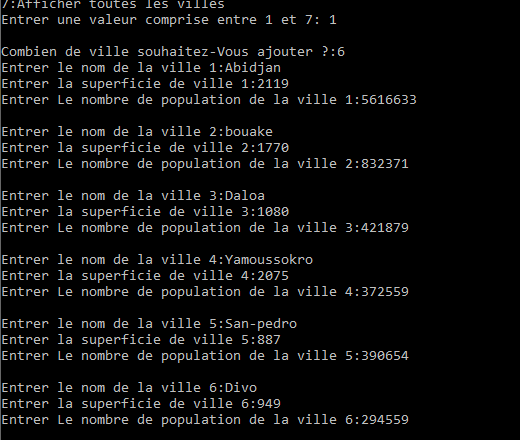
On constate que, nous avons choisi le mode interractif, car la fonction  « siasi\_utilisateur » n’est pas en commentaire.

Au lancement du programme, nous obtenons :



Le programme est fait de tel sorte que l’utilisateur face entré les valeur de 0 à 9.

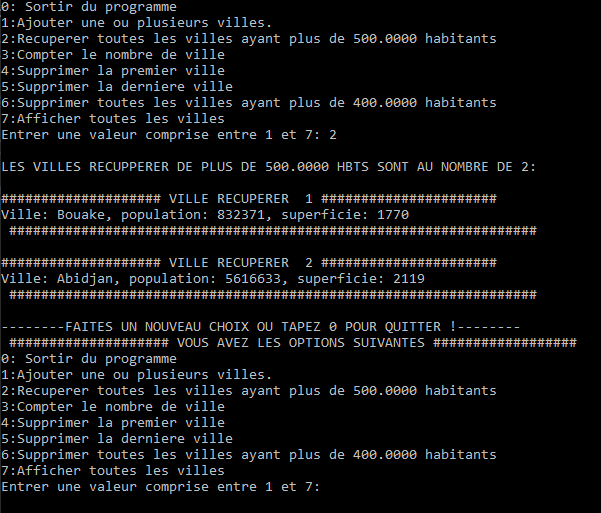
Si l’utlisateur entre une valeur différente( chaine de caractère ou nombre supérieur) le programme, lui demande d’entrée la valeur demander. Ainsi , notre programme tourne en boucle tant qu’on obtenions pas de valeur exacte.

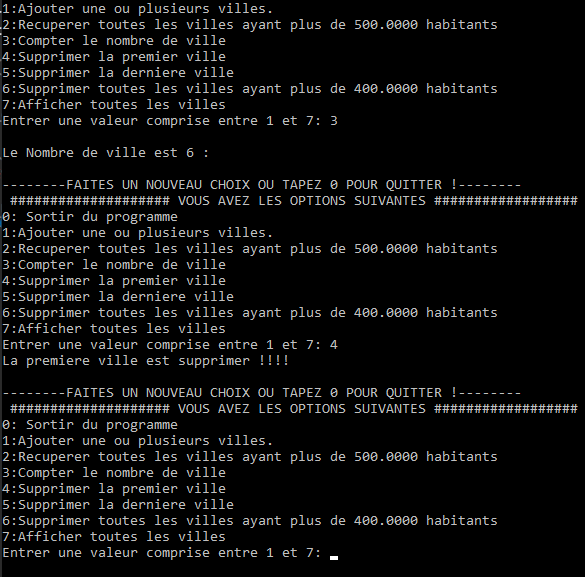
Pour L’option 1 :

Nous entrons ainsi les valeur pour le test souhaitez

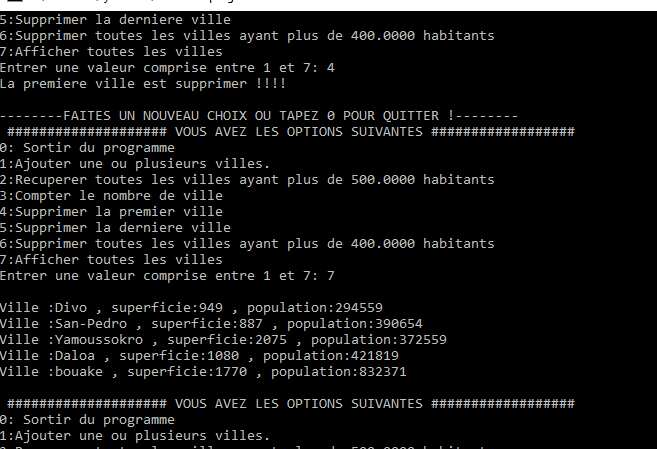
**Pour l’option 2 :**

La fonction recupére toutes les viles ayant plus de 500.000 habitants, les stocke dans une liste, puis les affiche

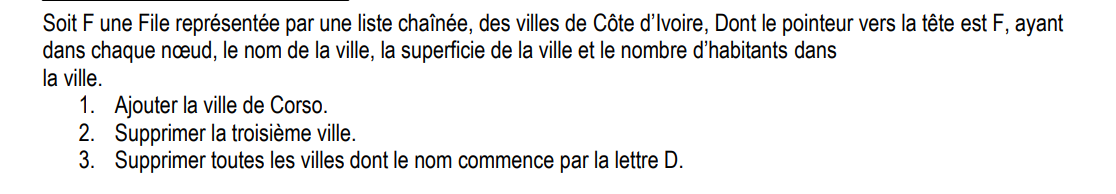


Pour l’option 3 et 4 :

Pour l’option 7 :



* **Sujet 3**

****

* **Analyse du sujet**

Pour ce sujet, nous allons d’abord construire une file représenter par une liste chainée.

typedef struct Ville {

    char nom\_ville[100];

    int superficie;

    int population;

    struct Ville \*suivant;

} Ville;

typedef struct File {

    Ville \*premier;

    Ville \*dernier;

} File;

* **Concernant la question 1**
* **Analyse algorithmique**

Pour ajouter la ville de Corso, il nous suffira d’abord de d’écrire la fonction enfiler

|  |
| --- |
| Fonction enfiler(file :^File, nom\_ville :chaîne de caractères, population , superficie : entiers)  Var  Nv\_ville :^Ville  Début  Allouer(nv\_ville)  Si (nv\_ville = Nil) alors  écrire (‘’erreur d’allocation de mémoire ‘’)  sortie(0)  sinon  copier(nv\_ville^.nom\_ville, nom\_ville)  nv\_ville^.superficie🡨superficie  nv\_ville^.population🡨population  nv\_ville^.suivant🡨NIL    Si (file^.dernier <> NIL) alors  file^.dernier^.suivant 🡨nv\_ville  sinon file^.dernier🡨 nv\_ville  fin si  Si (file^.premier <> NIL) alors  file^.premier^.suivant 🡨nv\_ville  fin si  fin si  fin fonction |

* **Programmation en C**

// Ajoute une ville à la fin de la file

void enfiler(File \*file, char \*nom\_ville, int population, int superficie) {

    /\* On gere deux cas lors de l'enfilage de la liste.

     le cas ou la file est vide et le cas ou la file est non vide.

    \*/

    Ville \*nv\_ville = (Ville \*)malloc(sizeof(\*nv\_ville));

    if (nv\_ville == NULL) {

        printf("Erreur d'allocation de mémoire\n");

        exit(1);

    }

    strcpy(nv\_ville->nom\_ville, nom\_ville);

    nv\_ville->superficie = superficie;

    nv\_ville->population = population;

    nv\_ville->suivant = NULL;

    if (file->dernier != NULL) {

        file->dernier->suivant = nv\_ville;

    }

    file->dernier = nv\_ville;

    if (file->premier == NULL) {

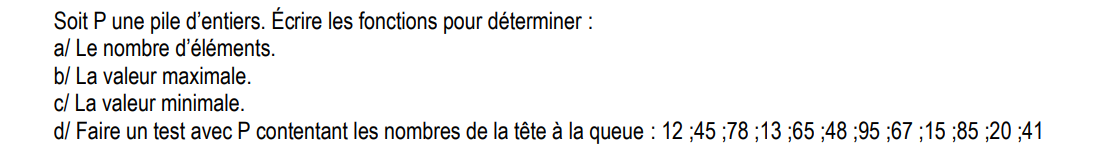
        file->premier = nv\_ville;

    }

}

**Pour ajouter cette ville nous aurons la procedure**

* **Sujet 4**

****

* **Analyse du sujet**

Pour ce sujet, nous somme amenés à créer une suite de fonctions.

* **Concernant la fonction qui détermine le nombre maximal d’éléments**

Pour cette partie nous créerons un compteur ‘’ compt ’’ qui vas parcourir la liste et vas afficher le nombre d’éléments de la liste. Lorsque le pointeur nous aurons un pointeur qui stockera la valeur du compteur jusqu'à la fin et affichera à la fin, la valeur qui est dernièrement stockée dans le pointeur.

* **Solution algorithmique**

|  |
| --- |
| Fonction nombreDepile(pile :^Pile)  Var  compt :entiers  temp :^NombreDepile  début  si(pile^.sommet =NIL) alors  écrire (‘’ La pile est vide ’’)  retouner ( 0 )  sinon  temp🡨 pile^.sommet  compt🡨0  tant que (temp<> NIL)  compt🡨compt+1  temp🡨temp^.suivant  fin tant que  fin si  fin si  retourner compt+1  Fin |

* **Programmation en C**

int nombreDePile(Pile \*pile) {

    if (pile->sommet == NULL) {

        printf("La pile est vide !\n");

        return 0;

    }

    NombrePile \*temp = pile->sommet;

    int compt = 0;

    while (temp != NULL) {

        compt++;

        temp = temp->suivant;

    }

    return compt++;

}

* **Concernant la fonction pour déterminer la valeur maximale**

Dans cette partir , un pointeur ‘’temp’’ sera créer et va s parcourir la liste et à la fin le nombre maximal d’éléments sera afficher.

* **Solution algorithmique**

|  |  |
| --- | --- |
| Fonction val\_max(pile :^Pile)  Var  Max : entiers  temp :^NombreDepile  début  si(pile^.sommet = NIL) alors  écrire (‘’ La pile est vide ’’)  retouner ( -1 )  sinon  max🡨temp^.valeur  tant que (temp<> NIL)  si(temp^.valeur > max) alors  max🡨temp^.valeur  sinon temp🡨^.suivant  fin tant que  fin si  fin si  retourner max  Fin |  |

* **Programmation en C**

int val\_max(Pile \*pile) {

    if (pile->sommet == NULL) {

        printf("La pile est vide !\n");

        return -1;

    }

    NombrePile \*temp = pile->sommet;

    int max = temp->valeur;

    while (temp != NULL) {

        if (temp->valeur > max) {

            max = temp->valeur;

        }

        temp = temp->suivant;

    }

    return max;

}

* **Concernant la détermination de la valeur minimale**
* **Solution algorithmique**

|  |
| --- |
| Fonction val\_min(pile :^Pile)  Var  max : entiers  temp :^NombreDepile  début  si(pile^.sommet = NIL) alors  écrire (‘’ La pile est vide ’’)  retouner ( -1 )  sinon  max🡨temp^.valeur  tant que (temp<> NIL)  si(temp^.valeur < max) alors  max🡨temp^.valeur  sinon temp🡨^.suivant  fin tant que  fin si  fin si  retourner max  Fin |

* **Programmation en C**

int val\_min(Pile \*pile) {

    if (pile->sommet == NULL) {

        printf("La pile est vide !\n");

        return -1;

    }

    NombrePile \*temp = pile->sommet;

    int max = temp->valeur;

    while (temp != NULL) {

        if (temp->valeur < max) {

            max = temp->valeur;

        }

        temp = temp->suivant;

    }

    return max;

}

* **Résultat après compilation**

E:\MAIN FOLDER\Desktop\be\exo4>prog

Les piles sont: 300 201 56 39 34 25 19 16 13 9 74 1

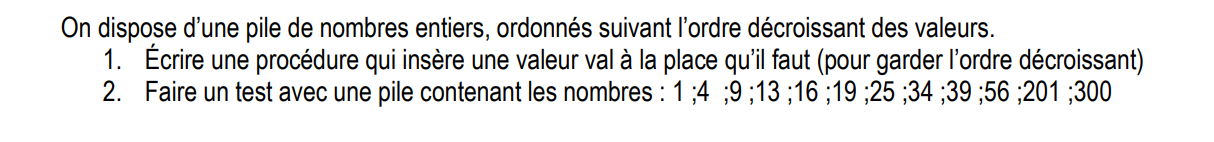
la valeur min est: 1

la valeur max est: 300

Le nombre de piles est 12

E:\MAIN FOLDER\Desktop\be\exo4>

* **Sujet 5**



* **Analyse du sujet**

Dans ce sujet il est question d’utiliser les piles

Données : piles d’entier

Traitement : - Demander à l’utilisateur d’entrer des valeurs en tenant compte du nombre max que la pile peut prendre

-Avec **la procédure insérer les nombres les valeurs** seront rangé dans l’ordre décroissant

Pour ce sujet, nous aurons à créer trois fichiers :

* prototypes.h
* principal.c
* fonctions.c
* **Solution algorithmique**
* fonctions.c

|  |
| --- |
| Procedure créer\_pile()  Var  pile :^Pile  Début  Allouer(pile)  pile^.sommet 🡨NUL  retourner pile  fin  procedure empiler(pile :^Pile, valeur :entier)  var  nb\_a\_empiler :^NombrePile  debut  allouer(nb\_a\_empiler)  si(nb\_a\_empiler = Nul) alors  écrire (« erreur d’allocation)  sortie(1)  fin si  nb\_a\_empiler^.valeur 🡨valeur  nb\_a\_empiler^.suivant 🡨 pile^.sommet  pile^.sommet 🡨 nb\_a\_empiler  Fin  Procedure inserer\_nb(pile :^Pile, nombre :entier)  Var  temp :^NombrePile  prec:^NombrePile  nouveau:^NombrePile  debut  temp 🡨 pile^.sommet  prec🡨NUL  allouer(nouveau)  tant que (temp <> NUL et temp^.valeur > nombre) faire  prec🡨temp  temp🡨temp^.suivant  fin tant que  si (prec = NUL) alors  nouvaeu^.suivant 🡨 prec^.somment  pile^.sommet 🡨 nouveau  sinon  nouveau^.suivant🡨 prec^.suivant  prec^.suivant🡨nouveau  fin si  fin si  Fin  Procedure afficherPile(pile :^Pile)  Var  temp :^NombrePile  debut  si(pile^.sommet = Nul) alors  ecrire( ‘’ la pile est vide !’’ )  sinon  temp🡨pile^.sommet  écrire (‘’ Les piles : ‘’)  tant que (temp <> NUL) faire  écrire(‘’ temp^.sommet)  temp🡨 temp^.suivant  fin tant que  fin si  fin si  Fin  Procedure saisi\_untilisateur(pile :^Pile)  Var  nb\_a\_inserer, nb\_choix : entiers  Début  Ecrire (‘’combien de nombre voulez vous inserer ? :’’)  Lire(nb\_choix)  nb\_a\_insere 🡨0  pour i allant de 1 à nb\_choix faire  ecrire(‘’entrez une valeur à inserer ‘’  lire(nb\_a\_inserer)  inserer\_nb (pile, nb\_a\_inserer)  fin pour  Fin |

* prototypes.h

|  |
| --- |
| Type enregistrement  valeur :entier  suivant :^NombrePile  fin enregistrement NombrePile  type enregistrement  sommet :^NombrePile  fin enregistrement NombrePile  Pile^ creer\_pile()  procedure empiler(pile :^Pile, valeur :entier)  procedure inserer\_nb(pile :^Pile, nombre:entier)  procedure saisi\_untilisateur(pile :^Pile) |

* principal.c

|  |
| --- |
| Fonction principale  var  pile :^Pile  début  pile 🡨 creer\_pile  empiler(pile, 1)  empiler(pile, 4)  empiler(pile, 9)  empiler(pile, 13)  empiler(pile, 16)  empiler(pile, 19)  empiler(pile, 25)  empiler(pile, 34)  empiler(pile, 39)  empiler(pile, 56)  empiler(pile, 201)  empiler(pile, 300)  ecrire(‘’----------Avant insertion---------‘’)  afficherPile(pile)  saisi\_utilisateur(pile)  ecrire(« -----------Apres insertion------------‘’)  afficherPile(pile)  retourner (0)  Fin |

* **Programmation en c**
* fonctions.c

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "prototypes.h"

Pile \*creer\_pile() {

    Pile \*pile = (Pile \*)malloc(sizeof(Pile));

    pile->sommet = NULL;

    return pile;

}

void empiler(Pile \*pile, int valeur) {

    NombrePile \*nb\_a\_empiler = (NombrePile \*)malloc(sizeof(NombrePile));

    if(nb\_a\_empiler == NULL){

        printf("Erreur d'allocation !");

        exit(1);

    }

    nb\_a\_empiler->valeur = valeur;

    nb\_a\_empiler->suivant = pile->sommet;

    pile->sommet = nb\_a\_empiler;

}

void inserer\_nb(Pile \*pile, int nombre) {

    NombrePile \*temp = pile->sommet;

    NombrePile \*prec = NULL;

    NombrePile \*nouveau = (NombrePile \*)malloc(sizeof(NombrePile));

    nouveau->valeur = nombre;

    while (temp != NULL && temp->valeur > nombre) {

        prec = temp;

        temp = temp->suivant;

    }

    if (prec == NULL) {

        nouveau->suivant = pile->sommet;

        pile->sommet = nouveau;

    }

    else {

        nouveau->suivant = prec->suivant;

        prec->suivant = nouveau;

    }

}

void afficherPile(Pile \*pile) {

    if (pile->sommet == NULL) {

        printf("La pile est vide !\n");

        return;

    }

    NombrePile \*temp = pile->sommet;

    printf("Les piles: ");

    while (temp != NULL) {

        printf("%d ", temp->valeur);

        temp = temp->suivant;

    }

    printf("\n");

}

void saisi\_utilisateur(Pile \*pile){

    int nb\_choix;

    printf("Combien de nombre voulez vous inserer?:");

    scanf("%d", &nb\_choix);

    int nb\_a\_inserer = 0;

    for(int i=1; i<=nb\_choix; i++){

        printf("Entrer une valeur a inserer");

        scanf("%d", &nb\_a\_inserer);

        inserer\_nb(pile, nb\_a\_inserer);

    }

}

* prototypes.h

#ifndef \_\_EXO5\_\_

#define \_\_EXO5\_\_

// les structures de base

typedef struct NombrePile{

    int valeur;

    struct NombrePile \*suivant;

}NombrePile;

typedef struct Pile{

    NombrePile \*sommet;

}Pile;

/\*---------------------------Prototype des fonctions-------------------------------\*/

Pile \*creer\_pile();

void empiler(Pile \*pile, int valeur);

void inserer\_nb(Pile \*pile, int nombre);

void afficherPile(Pile \*pile);

void saisi\_utilisateur(Pile \*pile);

#endif

* principal.c

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "prototypes.h"

int main(){

    Pile \*pile = creer\_pile();

    empiler(pile, 1);

    empiler(pile, 4);

    empiler(pile, 9);

    empiler(pile, 13);

    empiler(pile, 16);

    empiler(pile, 19);

    empiler(pile, 25);

    empiler(pile, 34);

    empiler(pile, 39);

    empiler(pile, 56);

    empiler(pile, 201);

    empiler(pile, 300);

    printf("--------------------Avant insertion-----------------------");

    afficherPile(pile);

    saisi\_utilisateur(pile);

    printf("-----------------------------Apres Insertion-----------------------");

    afficherPile(pile);

    return 0;

}

* **Resultat après compilation**

E:\MAIN FOLDER\Desktop\be\exo5\inside>prog

--------------------Avant insertion-----------------------Les piles: 300 201 56 39 34 25 19 16 13 9 4 1

Combien de nombre voulez vous inserer?:5

Entrer une valeur a inserer45

Entrer une valeur a inserer54

Entrer une valeur a inserer54

Entrer une valeur a inserer56

Entrer une valeur a inserer356

-----------------------------Apres Insertion-----------------------Les piles: 356 300 201 56 56 54 54 45 39 34 25 19 16 13 9 4 1

E:\MAIN FOLDER\Desktop\be\exo5\inside>

* **Sujet 6**



* **Analyse du sujet**

Pour le sujet 6, il nous ait demandé de ranger une pile par ordre croissant. Pour ce faire, nous allons créer d’abord une pile. Ensuite écrire la procédure pour trier

Données : pile d’entiers non rangée.

Traitement : Nous utiliserons deux pointeurs(pointeur\_1, pointeur\_2) et une valeur ‘’tempo’’ de type entier. Pointeur\_1 sera à la ‘’tete’’ et vas parcourir la liste. Si la valeur du pointeur\_1 est supérieure à la valeur du pointeur\_2, l’on fait des échanges entre la valeur de pointeur\_1 et pointeur\_2 par l’intermédiaire de ‘’tempo’’.

Résultat : liste triée

* **Solution algorithmique**

|  |
| --- |
| Type enregistrement  valeur : entier  suivant : ^PILE ;  Fin enregistrement PILE ;  Procedure empiler(tete :^^PILE, element : entier)  Var  nouveau\_elm :^PILE;  Début  allouer(taille de nouveau\_elm)  nouveau\_elm^.valeur 🡨 element  nouveau\_elm^.suivant 🡨 ^tete  ^tete 🡨 nouveau\_elm  Fin  Procédure tri\_pile(tete :^PILE)  Var  pointeur\_1, pointeur\_2 :^PILE ;  tempo : entier  Début  pour pointeur\_1 allant de tete à pointeur^.suivant faire  pour pointeur\_2 allant de pointeur\_1^.suivant à pointeur\_2^.suivant faire  si(pointeur\_1^.valeur > pointeur\_2^.valeur) faire  tempo 🡨 pointeur\_1^.valeur  pointeur\_1^.valeur 🡨 pointeur\_2^.valeur  pointeur\_2^.valeur 🡨 tempo  fin si  fin pour  fin pour  fin  procedure afficher\_liste(tete :^PILE)  Début  Tant que (tete différent de NULL) faire  écrire ( tete^.valeur)  tete 🡨 tete^.suivant  fin tant que  écrire(‘’ fin de la liste. ’’)  fin  Fonction principale  Var  pile :^PILE ;  Début  Pile 🡨 NULL ;  empiler(&pile, 16) ;  empiler(&pile, 19) ;  empiler(&pile, 34) ;  empiler(&pile, 4) ;  empiler(&pile, 25) ;  empiler(&pile, 9) ;  empiler(&pile, 130) ;  empiler(&pile, 1) ;  empiler(&pile, 201) ;  empiler(&pile, 56) ;  empiler(&pile, 300) ;  empiler(&pile, 39) ;  écrire (‘’LA LISTE AVANT LE TRI : ‘’)  afficher\_liste(pile)  tri\_bulle\_pile(pile) // dans le programme, il se passe un tri  écrire(‘’VOICI LA LISTE EN ORDRE CROISSANT APRES LE TRI : ‘’)  afficher\_liste(pile) // la liste s’affiche à nouveau  retourner 0 ;  } |

* **Programmation en C**

#include <stdio.h>

#include <stdlib.h>

// CONSTRUCTION DE LA STRUCTURE PILE

typedef struct PILE {

    int valeur;

    struct PILE \*suivant;

};

typedef struct PILE PILE;

            // PROCEDURE POUR AJOUTER UN ELEMENT

void empiler(PILE \*\*tete, int element) {

    // ALLOCATION DE MEMOIRE POUR UN ELEMENT

    PILE \*nouveau\_elm = (struct PILE \*)malloc(sizeof(struct PILE));

    nouveau\_elm->valeur = element;

    nouveau\_elm->suivant = (\*tete);

    (\*tete) = nouveau\_elm;

}

// PROCEDURE POUR TRIER LA PILE

void tri\_pile(PILE \*tete) {

     PILE \*pointeur\_1, \*pointeur\_2;

    int tempo;

            // PARCOURIR LA PILE

    for (pointeur\_1 = tete; pointeur\_1 != NULL; pointeur\_1 = pointeur\_1->suivant) {

        for (pointeur\_2 = pointeur\_1->suivant; pointeur\_2 != NULL; pointeur\_2 = pointeur\_2->suivant) {

            if (pointeur\_1->valeur > pointeur\_2->valeur) {  // ECHANGER LES VALEURS

                tempo = pointeur\_1->valeur;

                pointeur\_1->valeur = pointeur\_2->valeur;

                pointeur\_2->valeur = tempo;

            }

        }

    }

}

            // PROCEDURE POUR AFFICHER LA LISTE

void afficher\_liste(PILE \*tete) {

    while (tete != NULL) {

        printf("%d ; ", tete->valeur);

        tete = tete->suivant;

    }

    printf(" fin de liste.\n\n ");

}

int main() {

    PILE \*pile = NULL; //Initialisation de la liste

            // CREATION DE LA LISTE EN AJOUTANT DES ELEMENTS

    empiler(&pile, 16);

    empiler(&pile, 19);

    empiler(&pile, 34);

    empiler(&pile, 4);

    empiler(&pile, 25);

    empiler(&pile, 9);

    empiler(&pile, 130);

    empiler(&pile, 1);

    empiler(&pile, 201);

    empiler(&pile, 56);

    empiler(&pile, 300);

    empiler(&pile, 39);

    printf("\nLA LISTE AVANT LE TRI : "); // AVANT LE TRI

    afficher\_liste(pile);

    tri\_pile(pile); // TRI DE LA LISTE

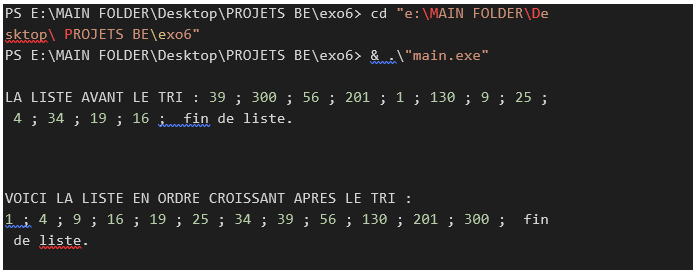
    printf("\n\nVOICI LA LISTE EN ORDRE CROISSANT APRES LE TRI : \n"); //APRES LES TRI

    afficher\_liste(pile);

    return 0;

}

* **Resultat après compilation**



* **Sujet 7**
* **Analyse du sujet**

Dans cet exercice, il nous ai demander de créer d’abord deux piles d’entiers et ensuite les fusionner dans une pile résultante tout en sachant que les deux piles et la pile résultante sont en en ordre croissant.

Données : deux piles d’entiers, A et B rangées en ordre croissant.

Traitement : Lors de l’exécution, le programme nous avons deux pointeurs, un sur la pile A et l’autre sur la pile B. A chaque fois, la valeur du pointeur (a) est comparée à celle du pointeur (b) et ensuite la plus petite valeur rentre dans la pile résultante.

Résultat : fusion de deux piles dans une seul pile triée par ordre croissant.

* **Solution algorithmique**

|  |
| --- |
| Type enregistrement  Valeur : entier  Suivant : ^nœud  Fin enregistrement nœud  Prcedure empiler(tete\_ref :^^nœud, nouvelle\_valeur : entiers )  Var  nouveau\_noeud :^nœud ;  Début  Allouer(nouveau\_noeud )  Si (nouveau\_noeud = NUL)  écrire (‘’ Erreur d’allocation de mémoire ’’)  sortir ;  fin si  nouveau\_noeud ^.valeur 🡨nouvelle\_valeur  nouveau\_^.suivant 🡨 ^tete\_ref  \*tete\_ref 🡨 nouveau\_noeud ;  Fin  Fonction fusionTriee(a, b :^nœud)  Var  résultat :^nœud  Début  Nœud 🡨 NUL  Si( a= Nul) retourner b ;  Sinon si (b = NUL) retourner a ;  Si ( a^.valeur <= b^.valeur) alors  resultat 🡨 a ;  resultat^.suivant 🡨 fusionTriee( a^.suivant, b)  sinon  resultat = b ;  resultat^.suivant 🡨 fusionTriee(a, b^.suivant)  fin si  retourner resultat ;  fin  procédure afficherListe(nœud :^noued)  début  tant que(noued <> NUL) faire  écrire ( nœud^.valeur)  nœud 🡨 nœud^.suivant  fin tant que  fin  fonction principale  var  resultat : ^nœud ;  pileA: ^nœud ;  pileB: ^nœud ;  Début  //ajouter des elements a la pile ‘pileA’  empiler(&pileA, 85)  empiler(&pileA, 85)  empiler(&pileA, 45)  empiler(&pileA, 17)  empiler(&pileA, 17)  empiler(&pileA, 11)  empiler(&pileA, 2)  écrire (‘’La liste 1 est : ‘’)  afficherListe(pileA) ;  // saut de ligne  // ajouter des elements a la pile ‘pileB’  Empiler(&pileB, 100)  Empiler(&pileB, 78)  Empiler(&pileB, 43)  Empiler(&pileB, 32)  Empiler(&pileB, 20)  Empiler(&pileB, 13)  Empiler(&pileB, 17)  écrire(‘’ La liste 2 est : ’’)  afficherListe(pileB)  //fusion des deux piles ‘pileA’ et ‘pileB’  resultat 🡨 fusionTriee(pileA, pileB)  écrire (‘’LA fusion des deux listes est : ’’)  afficherListe(resultat)    retourner 0 ;  Fin |

* **Programmation en C**

#include <stdio.h>

#include <stdlib.h>

// Structure de pile

struct noeud {

    int valeur;

    struct noeud \*suivant;

};

// Fonction pour fusionner les piles

struct noeud \*fusionTriee(struct noeud \*a, struct noeud \*b)

{

struct noeud \*resultat = NULL;

if (a == NULL)

    return b;

else if (b == NULL)

    return a;

if (a->valeur <= b->valeur) {

    resultat = a;

    resultat->suivant = fusionTriee(a->suivant, b);

} else {

    resultat = b;

    resultat->suivant = fusionTriee(a, b->suivant);

}

return resultat;

}

// Fonction pour afficher la pile

void afficherListe(struct noeud \*noeud)

{

while (noeud != NULL) {

printf("%d ", noeud->valeur);

noeud = noeud->suivant;

}

}

// Fonction pour ajouter un �l�ment � la pile

void empiler(struct noeud \*\*tete\_ref, int nouvelle\_valeur)

{

    struct noeud \*nouveau\_noeud = (struct noeud \*)malloc(sizeof(struct noeud));

    if (nouveau\_noeud == NULL) {

        printf("Erreur d'allocation de m�moire");

        exit(0);

    }

    nouveau\_noeud->valeur = nouvelle\_valeur;

    nouveau\_noeud->suivant = (\*tete\_ref);

    (\*tete\_ref) = nouveau\_noeud;

}

int main()

{

    struct noeud \*resultat = NULL;

    struct noeud \*pileA = NULL;

    struct noeud \*pileB = NULL;

    // Ajouter des �l�ments � la pile 'pileA'

    empiler(&pileA, 85);

    empiler(&pileA, 45);

    empiler(&pileA, 26);

    empiler(&pileA, 17);

    empiler(&pileA, 17);

    empiler(&pileA, 11);

    empiler(&pileA, 2);

    printf("\n La liste 1 est : ");

    afficherListe(pileA);

    printf("\n");

    // Ajouter des �l�ments � la pile 'pileB'

    empiler(&pileB, 100);

    empiler(&pileB, 78);

    empiler(&pileB, 43);

    empiler(&pileB, 32);

    empiler(&pileB, 20);

    empiler(&pileB, 17);

    empiler(&pileB, 13);

    printf("\n La liste 2 est : ");

    afficherListe(pileB);

    printf("\n");

    // Fusionner les piles 'pileA' et 'pileB'

    resultat = fusionTriee(pileA, pileB);

    printf("\nLa fusion des deux listes : ");

    afficherListe(resultat);

    printf("\n");

    return 0;

}

* **Resultat après compilation**

PS E:\MAIN FOLDER\Desktop\be\exo7> cd "e:\MAIN FOLDER\Desktop\be\exo7"

PS E:\MAIN FOLDER\Desktop\be\exo7> & .\"main.exe"

 La liste 1 est : 2 11 17 17 26 45 85

 La liste 2 est : 13 17 20 32 43 78 100

La fusion des deux listes : 2 11 13 17 17 17 20 26 32 43 45 78 85 100

PS E:\MAIN FOLDER\Desktop\be\exo7>