Visite 50 points

Charlie habite une belle grande ville. Chacune des **N** rues et chacun des **C** croisements de la ville porte un numéro unique. Charlie reçoit la visite de Jill et à cette occasion, il veut lui montrer toutes les rues mais pour qu'elle ne s'ennuie pas, il se dit qu'il vaut mieux ne passer qu'une seule fois par chacune des rues.

Tâche

Votre tâche consiste à aider Charlie en écrivant un programme qui indique l'ordre dans lequel il doit visiter toutes les **N** rues de la ville sans passer deux fois par la même rue. Charlie habite dans la rue portant le numéro 1. À la fin de la visite, Charlie et Jill ne doivent pas nécessairement se retrouver dans la rue de départ. Comme il y a éventuellement beaucoup de manières différentes d'effectuer cette visite, il faut uniquement produire la visite passant par toutes les rues mais de telle manière que, si à un des **C** croisements plusieurs rues sont possibles, la rue portant le numéro le plus petit soit systématiquement choisie.

Restrictions

$1 \leq \mathbf{N} \leq 2\,000$ Chaque rue porte un numéro entier unique compris entre 1 et 2000

$1 \leq \mathbf{C} \leq 200$ Chaque croisement porte un numéro entier unique compris entre 1 et 200

Chaque rue donne sur exactement deux croisements qui ne sont pas nécessairement différents.

Entrée et sortie du programme

Entrée

- Le programme affiche à la console le message « Nom du fichier : » et attend que l'utilisateur entre le nom du fichier contenant la liste non vide de toutes les rues terminée par la valeur -1. Chaque ligne du fichier se compose de trois entiers séparés par exactement un espace : les deux premiers entiers désignent les numéros des croisements suivis par le numéro de la rue reliant ces deux croisements. Dans le fichier, exactement une rue porte le numéro 1. Charlie habite dans la rue numéro 1 et il quitte sa rue par le croisement qui porte le plus grand numéro.

Sortie

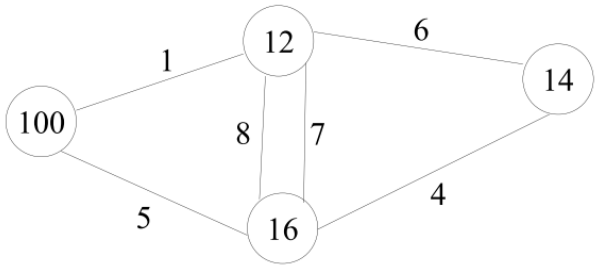
- Si le programme trouve une solution, il affiche à la console le message « Solution : » suivi du détail de la visite, c'est-à-dire des numéros de rue séparés par exactement un espace. Dans le cas contraire, il affiche le message « Pas de solution ».

Exemples d'exécution

Exemple 1

Contenu du fichier *parcours1.txt* :

```
12 14 6
16 14 4
12 16 7
100 12 1
12 16 8
100 16 5
-1
```



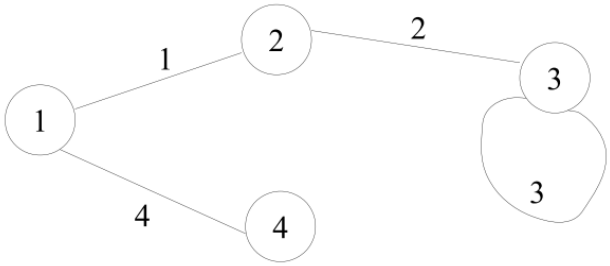
Affichage à la console :

```
Nom du fichier : parcours1.txt
Solution : 1 5 4 6 7 8
```

Exemple 2

Contenu du fichier *parcours2.txt* :

```
1 2 1
2 3 2
3 3 3
4 1 4
-1
```



Affichage à la console :

```
Nom du fichier : parcours2.txt
Pas de solution
```

Remettez le programme sous le nom VISITE.xxx, avec xxx=PAS ou C(PP).

GPS 50 points

Vous êtes le nouveau programmeur en chef auprès d'un constructeur de GPS. Votre première tâche consiste en l'optimisation de la recherche d'un nom de ville, chose qui actuellement met beaucoup trop de temps. Pour rechercher un nom de ville, l'utilisateur utilise le clavier tactile pour entrer lettre par lettre le nom recherché. Cependant, afin de faciliter

cette recherche, uniquement les lettres, pour lesquelles il existerait au moins une ville avec ce préfixe, sont activées.

Exemple de l'image ci-dessus : La lettre « O » est active parce qu'il existe une ville qui commence par « NEW YO ». La lettre « X » par exemple n'est pas active car aucune ville ne commence par « NEW YX ». Si maintenant le GPS connaissait une ville qui s'appelait « New Yuke », alors la lettre « U » serait aussi active.

Tâche

Afin de faciliter les essais du GPS, on vous fournit déjà un programme (*english : grader*) qui contient une librairie dans laquelle se trouvent quatre fonctions (pseudo-code) :

- `void init(string filename)` : cette fonction est appelée une seule fois au lancement du programme. Elle reçoit en argument le nom du fichier texte **F** qui contient **N** noms de villes.
- `boolean[27] getKeyboard(char C)` : cette fonction prend en argument le prochain caractère entré par l'utilisateur et retourne un tableau de 27 valeurs booléennes. Les 26 premières valeurs définissent respectivement si les 26 lettres de l'alphabet sont actives ou inactives. La 27ième valeur définit si la touche pour l'espace (*english : space*) est active.
  Exemple : si la deuxième valeur du tableau est « true », alors la touche pour la lettre « B » du clavier est active.
- `string[] getCityNames(int X)` : cette fonction est appelée par le programme après chaque appel de `getKeyboard()`. Elle retourne un tableau des noms de toutes les villes qui commencent par le texte que l'utilisateur a déjà entré si le nombre de ce ces villes est inférieur ou égal à **X**. Si par contre, le nombre de villes est supérieur à **X**, la fonction renvoie un tableau vide. Les noms de ce tableau doivent être triés et le tableau ne doit pas contenir de doublons !
- `void clear()` : cette fonction efface le texte que l'utilisateur a déjà entré.

Pour la définition exacte des fonctions, veuillez consulter la page 6 de ce document ainsi que le code-source fourni.

Restrictions

- Le nom de fichier **F** est un chemin relatif vers ce fichier. À la limite, vous pouvez supposer que le nom de fichier se trouve toujours dans le répertoire du programme. Exemples :
  - `init('cities.txt')`
  - `init('cities/cities1.txt')`
- La lettre **C** passée en argument à la fonction `getKeyboard()` appartient à l'ensemble

suivant :{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, _, #}. Cet ensemble est constitué des 26 lettres minuscules, du signe « _ » pour l'espace et du symbole « # ». Le symbole « # » signifie que l'utilisateur a utilisé la touche « Backspace » sur le clavier tactile et a donc enlevé le dernier caractère du texte que l'utilisateur a déjà entré.

Attention :

- Cette fonction sera aussi appelée avec des lettres non-actives du clavier !
- Utiliser la touche « Backspace » alors qu'aucun texte n'a encore été entré est possible mais n'a pas d'effet.

- Le nombre **X** passé en argument à la fonction `getCityNames()` est un nombre entier strictement positif.
- Le nombre **N** de villes contenues dans le fichier texte est un nombre strictement positif qui peut être très grand.

Sous-tâche 1 [5 Points]

Pour 5 points, les restrictions suivantes sont respectées :

$1 \leq \mathbf{N} \leq 1\,000$ Le nombre de villes **N** dans le fichier

$1 \leq \mathbf{X} \leq 3$ Paramètre pour `getCityNames()`

$1 \leq$ **nombre d'appels à `getKeyboard()` et `getCityNames()`** $\leq 1\,000$

Sous-tâche 2 [10 Points]

Pour 10 points, les restrictions suivantes sont respectées :

$1 \leq \mathbf{N} \leq 100\,000$ Le nombre de villes **N** dans le fichier

$1 \leq \mathbf{X} \leq 100$ Paramètre pour `getCityNames()`

$1 \leq$ **nombre d'appels à `getKeyboard()` et `getCityNames()`** $\leq 100\,000$

Sous-tâche 3 [35 Points]

Pour 35 points, les restrictions suivantes sont respectées :

$1 \leq \mathbf{N} \leq 1\,000\,000$ Le nombre de villes **N** dans le fichier

$1 \leq \mathbf{X} \leq 1\,000$ Paramètre pour `getCityNames()`

$1 \leq$ **nombre d'appels à `getKeyboard()` et `getCityNames()`** $\leq 1\,000\,000$

Entrée du programme

- La première ligne du fichier texte **F** contient un seul nombre entier positif : le nombre **N** de noms de villes.
- Les **N** lignes suivantes contiennent les différents noms de villes, un nom par ligne. Les noms sont

tous écrits en minuscules et sont formés exclusivement des 26 lettres de l'alphabet et de l'espace « ». Le nom d'une ville a une longueur maximale de 35 caractères.

- **Attention** : les noms des villes ne sont pas triés et il peut y avoir des doublons dans la liste.

Exemple d'exécution

Soit le contenu du fichier texte CITIES.TXT le suivant :

```
6
york
yake
yike
york
ying yang
yom
```

Voici une séquence possible d'appels de fonctions ainsi que leurs valeurs de retour. Afin de simplifier la mise en page, la valeur « true » est représentée par 1 et la valeur « false » par 0:

```
init('CITIES.TXT');
```

`getKeyboard('y');`	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(3);`	[] // retourne un tableau vide car il y a 5 villes potentielles et 5 > 3
`getKeyboard('i');`	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(3);`	['yike', 'ying yang']
`getKeyboard('#');`	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(4);`	[] // retourne un tableau vide car il y a 5 villes potentielles et 5 > 4
`getKeyboard('o');`	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(3);`	['yom', 'york'] // retourne un tableau trié sans doublons
`clear();`	
`getKeyboard('y');`	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(5);`	['yake', 'yike', 'ying yang', 'yom', 'york']
`getKeyboard('x');`	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(5);`	[]
`getKeyboard('#');`	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
`getCityNames(5);`	['yake', 'yike', 'ying yang', 'yom', 'york']

Remettez le programme sous le nom GPS.xxx, avec xxx=PAS, C(PP).

Delphi

Le programme qui vous est fourni contient les fichiers suivants :

1. Le fichier « GPS_Grader.dpr » sert à tester la librairie que vous devez compléter. Ce fichier lit le fichier IN.TXT et compare votre résultat au résultat fichier.
2. Le fichier « GPS.pas » est une librairie que vous devez compléter. Elle contient la définition du type TKeyboard ainsi que la déclaration des quatre fonctions/procédures. Vous ne devez en aucun cas modifier le type ou la déclaration des fonctions/procédures. Cependant, vous devez implémenter les fonctions/procédures après le mot-clé « implementation ». En-dessous du mot-clé « implementation », vous pouvez rajouter autant de types, variables ou fonctions/procédures que vous le jugez nécessaire.

Déclarations importantes :

```
type
  TKeyboard = array[0 .. 26] of boolean;

procedure init(s : string);
function getKeyboard(c : char) : TKeyboard;
function getCityNames(x : integer) : TStringList;
procedure clear();
```

C/C++

Le programme qui vous est fourni contient les fichiers suivants :

1. Le fichier « grader.c(pp) » sert à tester la librairie que vous devez compléter. Utilisez la commande ci-dessous pour lancer le grader correctement.
2. Le fichier « gps.c(pp) » (avec « grader.h ») est une librairie que vous devez compléter. Elle contient la déclaration des quatre fonctions. Vous ne devez en aucun cas modifier la déclaration des fonctions. Cependant, vous devez implémenter les fonctions dans le fichier « gps.c(pp) ». Vous pouvez rajouter autant de code que vous le jugez nécessaire.

Pour éviter le problème des données de retour, la fonction `init()` alloue de la mémoire pour les variables globales « letters » et « cities ». Les fonctions retournent un pointeur vers ces variables. Ceci est une manière possible de procéder, mais vous pouvez en utiliser une autre aussi longtemps que le type de retour correspond à la déclaration des fonctions.

Pour compiler :

- `C :  gcc gps.c grader.c`
- `C++ : g++ gps.cpp grader.cpp`

Pour lancer le grader, utilisez la commande suivante depuis une console: grader.exe <IN.TXT

Déclarations importantes :

```
void init(char *filename);      // code this function in gps.c(pp)
int* getKeyboard(char C);       // code this function in gps.c(pp)
char** getCityNames(int X);     // code this function in gps.c(pp)
void clear();                   // code this function in gps.c(pp)
```