CTP SDA2

Décembre 2013

2h avec fichiers et documents

téléphone interdit, messageries interdites

CODE BLOCKS autorisé

PARTIE 1 : listes chainées (Pour faciliter la correction, vous utiliserez le fichier exo1.c et vous ajouterez tout votre code dans ce fichier unique : Pas de compilation séparée lors de ce ctp)

1 seul fichier à rendre SVP

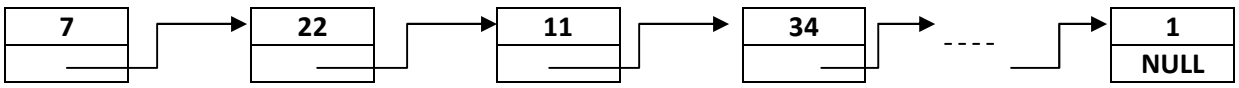
La conjecture tchèque (ou conjecture de Syracuse) forme une suite de nombres à partir d'une valeur V donnée en suivant la règle suivante :

- **si V est paire alors V reçoit V/2**
- **si V est impaire alors V reçoit (3*V)+1**
- **on applique cette règle jusqu'à obtenir V=1**

Ainsi à partir de V=7 on obtient V=22 puis V=11 puis V=34 puis V=17 puis ... cette suite de nombre se termine TOUJOURS par 1

Complétez exo1.c

Question 1 : à partir d'une valeur V saisie par l'utilisateur, construire la liste chainée (en utilisant insérer en fin de liste) contenant la suite conjecturée à l'aide de la règle ci-dessus. Pour V=7, on aura en mémoire :



Question 2 : afficher la liste ainsi formée. On souhaite un affichage proche de la figure suivante :

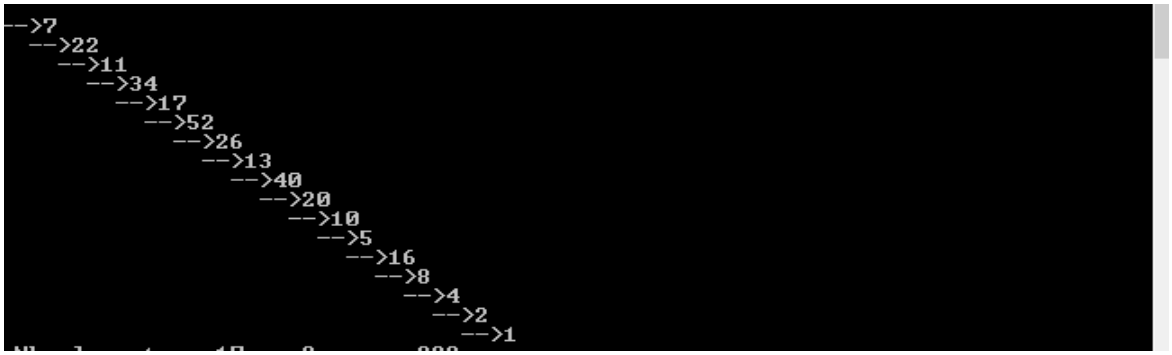


figure 1

Question 3 : afficher le nombre d'éléments dans la liste ainsi que la somme des éléments de cette liste. On souhaite un affichage proche de la figure suivante :

```
Nb elements = 17 -- Somme = 288
```

Question 4 : supprimez toutes les valeurs paires de la liste puis affichez la liste modifiée. Vous libérerez proprement la mémoire des mailles contenant des valeurs paires.

En utilisant alors les choix 2 et 3 du menu et à partir de la liste de la figure 1, on aura l'affichage suivant :

```
-->7
  -->11
    -->17
      -->13
        -->5
          -->1
Nb elements = 6 -- Somme = 54
```

PARTIE 2 : piles (Pour faciliter la correction, vous utiliserez le fichier exo2.c et vous ajouterez tout votre code dans ce fichier unique : Pas de compilation séparée lors de ce ctp)

1 seul fichier à rendre SVP

Le fichier exo2.c offre le code des permutations vu en TP1 sur les piles. Il sera utile pour traiter la question 2 de cette partie.

Concernant la question 1 de cette partie, vous pouvez jouter vos fonctions de traitement de pile si vous le souhaitez.

On considère un tableau 10X10 de float qui symbolise un labyrinthe.

On entre dans celui-ci par la case en haut à gauche (case 0,0 contenant 0.25) et on en sortira par la case en bas à droite (case 9,9 contenant 9.5). Pour avancer vers la sortie, il faut parcourir un chemin de valeurs réelles STRICTEMENT CROISSANTES.

On ne peut PAS se déplacer EN DIAGONALE. (uniquement à droite ou à gauche ou au dessus ou en dessous)

Question 1 : labyrinthe à chemin unique (avec ou sans solution)

Le chemin en vert est dans ce labyrinthe, le seul chemin existant vers la sortie.

Tout au long de ce chemin, il n'y a pas de fausse piste : c'est à dire qu'aucune case verte ne possède plus d'une case voisine contenant une valeur strictement supérieure. Il n'y a donc qu'une seule façon de poursuivre son chemin (chemin unique).

Il suffirait de modifier une seule valeur de ce chemin vert pour empêcher de sortir du labyrinthe (par exemple : en changeant la valeur 3.201 (bas de la colonne 0) en une valeur inférieure à 3.12)

tableau Laby1 :

0,25	0,2	0,4	0,03	7,4	5	7,5	7,8	7	8,22
0,4	0,12	0,44	0,08	7,41	7,49	7,54	7,88	7,9	6,3
0,55	0,47	0,09	7,27	7,3	7,09	7,01	6,99	7,91	7,88
0,64	0,51	0,14	0,45	7,25	7,22	7,21	5,99	7,99	8
2,81	2,7	2,9	1,98	4,8	7,1	7	8,34	7	8,4
2,9	2,14	3,24	3,5	2,4	7,04	7,03	7,01	8	8,44
3,12	1,43	3,1	3,09	6,88	6,911	6,09	8,79	8,601	8,6
3,201	3,5	3,49	3,08	6,8	2,34	8,1	8,8	8,501	8,57
0,15	3,72	3,11	2,98	6,79	4,31	8,8	8,88	9	9,31
5,3	5,8	6,4	6,55	6,58	5,8	9,8	8,85	8,99	9,5

Complétez exo2.c permettant de trouver et d'afficher le chemin unique (s'il existe) conduisant vers la sortie.

Vous mémoriserez le chemin parcouru en empilant des structures T_Case mémorisant la valeur de la case parcourue, son indice de ligne et son indice de colonne. On donne :

```
typedef struct
{
float valeur;
int ligne;
int colonne;
} T_Case;
```

La pile ressemblera lors des premiers pas à :

Data	0,64 3 0 0,55 2 0 0,4 1 0 0,25 0 0
Sp	4

L'affichage du résultat se fera après recherche complète :

```
 DEPART :

 BAS BAS BAS BAS BAS BAS BAS DROITE BAS BAS DROITE DROITE DROITE HAUT HAUT HAUT
DROITE HAUT HAUT HAUT GAUCHE HAUT HAUT DROITE DROITE DROITE DROITE BAS BAS DROIT
E BAS BAS BAS GAUCHE GAUCHE BAS BAS DROITE DROITE BAS


 VOUS ETES SORTI !! BRAVO
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Cas du labyrinthe sans chemin (3.201 remplacé par 3.11)

```
 DEPART :

 BAS BAS BAS BAS BAS BAS


 PAS DE CHEMIN !! FIN
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Question 2 : labyrinthe à chemins multiples (avec ou sans solution)

Il peut exister maintenant jusqu'à 3 possibilités d'avancer à partir d'une case.

Pour la première case (0.25), il existe 2 possibilités d'avancer (0.3 : fausse piste jaune car elle conduit à une impasse et 0.41).

Pour la case (0.84) située 3 lignes plus bas, il existe 2 possibilités d'avancer (1 : fausse piste violette car elle conduit à une impasse et 2.81).

Nous proposons de modifier le type T_Case:

```
typedef struct
{
float valeur;
int ligne;
int colonne;
int nbPossibilites;
int suite[4];
} T_Case;
```

tableau Laby2 :

0,25	0,3	0,4	0,03	7,5	8	7,9	8,32	8	8,22
0,41	0,12	0,44	0,08	7,41	7,49	7,54	7,88	7,9	8,5
0,75	0,69	0,09	7,27	7,3	7,09	7,01	6,99	7,91	7,88
0,84	1	0,14	0,45	7,25	7,22	9,2	8,15	7,99	8
2,81	2,85	2,9	1,98	4,8	7,1	7	8,34	9	8,4
2,9	2,14	3,24	3,5	2,4	7,04	7,03	7,01	8	8,44
3,12	1,43	3,1	3,09	6,88	6,911	6,09	8,79	8,601	8,6
3,201	3,5	3,77	3,08	6,8	2,34	8,87	8,8	8,501	8,57
0,15	3,72	3,11	2,98	6,79	4,31	9,4	8,88	9	9,31
6,2	5,8	6,4	6,55	6,58	5,8	9,8	9,4	9,09	9,5

Pour la première case (0.25), nous empilerons la T_Case suivante :

0,25 0 0 2			
-1	0,3	0,41	-1

// il existe 2 pistes possibles (2 possibilités)

La case 0 du tableau suite indique une piste <u>au dessus</u> de la case (-1 si il n'y en a pas)

La case 1 du tableau suite indique une piste <u>à droite</u> de la case (il n'y en a une : c'est 0.3)

La case 2 du tableau suite indique une piste <u>en dessous</u> de la case (il n'y en a une : c'est 0.41)

La case 3 du tableau suite indique une piste <u>à gauche</u> de la case (-1 si il n'y en a pas)

L'exploration de la fausse piste jaune (à partir de la première case 0.25)du labyrinthe conduira à :

0,44 1 2 0			
-1	-1	-1	-1
0,4 0 2 0			
-1	-1	0,44	-1
0,3 0 1 1			
-1	0,4	-1	-1
0,25 0 0 2			
-1	0,3	0,41	-1

il n'y a plus de possibilités après 0.44, il faut donc dépiler et revenir à la première Case offrant une autre possibilité d'exploration : c'est la case 0.25 dans laquelle on écrasera 0.3 pour indiquer que cette piste est sans issue :

0,25 0 0 1			
-1	-1	0,41	-1

// il reste 1 possibilité désormais : elle se situe en dessous par
// la case 0.41

On empilera donc ensuite :

0,84 3 0 2			
-1	1	2,81	-1
0,75 2 0 1			
-1	-1	0,84	-1
0,41 1 0 1			
-1	-1	0,75	-1
0,25 0 0 2			
-1	0,3	0,41	-1

puis la piste violette (1 : nouvelle fausse piste) sera explorée . On dépilera encore pour explorer finalement la piste 2.81 ...

Travail : A l'aide de l'algorithme des permutations vu en TP Pile (fourni dans exo2.c), implémentez cette recherche de pistes dans le labyrinthe. Les macro-fonctions (remonterPere, FrereSuivant ...) seront à redéfinir. L'algorithme sera à adapter.