ÉCOLE CENTRALE DE LILLE

Algorithmique Avancée et Programmation

TEA AAP - TP2 2021

Types Abstraits de Données, Piles et Listes chaînées

Bruno SOARES ZIMMER

LILLE, FRANCE

2021

INTRODUCTION

Les vecteurs sont les formes les plus primordiales de création de liste, cette liste est appelée liste statique, puisque la taille est prédéfinie par le programmeur. Dans l'ordre, nous avons des vecteurs dynamiques, dans lesquels sa taille est définie dynamiquement, mais une fois qu'elle est allouée, il n'est plus possible de changer sa taille (seulement qu'elle vide tout de la mémoire et l'alloue à nouveau).

Enfin, nous avons le modèle le plus courant utilisé par les programmeurs du langage C, une liste créée de manière chaînée. Il a la particularité d'occuper la mémoire au fur et à mesure des besoins, tout comme il est possible de retirer des termes du milieu, et de le vider de sa mémoire (1).

Les objectifs de ces exercices sont multiples :

1) Nous devons appliquer le concept de pile utilisant la bibliothèque développée en classe.

2) Nous est demandé de créer 5 nouvelles fonctions à la manipulation de la liste, étant encore nécessaire de les développer sous forme récursive et itérative.

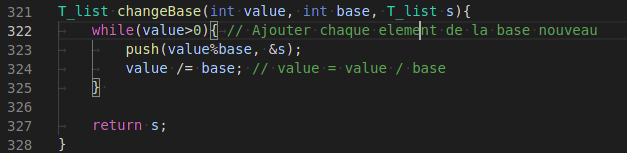
3) Il est demandé au programme de créer une image et cette image fournit tous les éléments de la liste liée.

4) L'exercice BONUS vise à créer une nouvelle gestion de liste, étant nécessaire de créer 4 nouvelles fonctions pour la gestion de celle-ci.

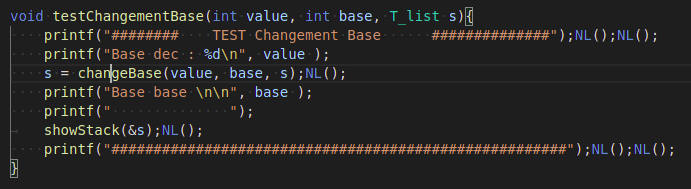
EXERCICE 1

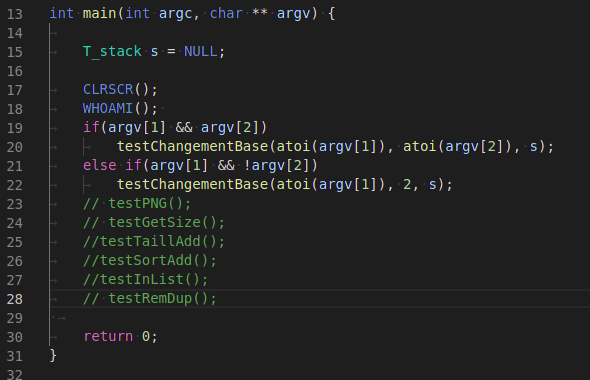
Dans ce premier exercice, nous devons résoudre le problème de l'échange de bases sur une base décimale.

La solution est dans l'ordre



Avec une fonction de teste dans l’ordre:

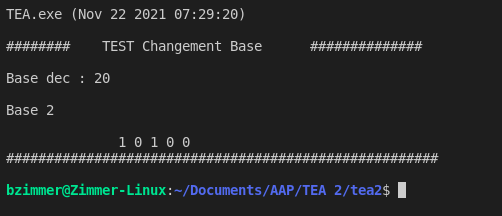




Étaient la fonction “testChagementBase” une fonction pour tester la fonction “changeBase” avec nombres aléatoires.

La fonction changeBase a qu’une boucle que arrété quand le value atteint zéro, qui ça veut dire que il n’y a plus de division, suivant la formule a chaque division du valeur par la base on va enregistrer le reste de la division dans la pile.

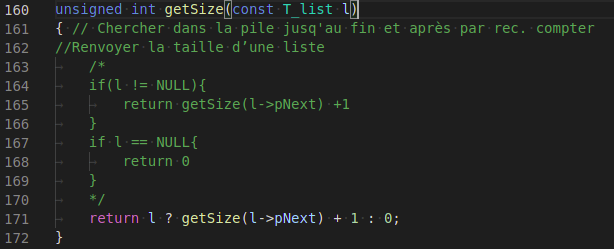
Après nous montrerons la liste de la formule showStack, avec le resultát:



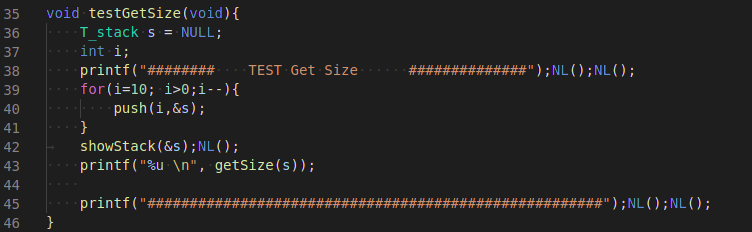
EXERCICE 2

2.1 Fonction “getSize”

La première fonction est appelée "getSize", en version récursive en utilisant le conditionnel ternaire:

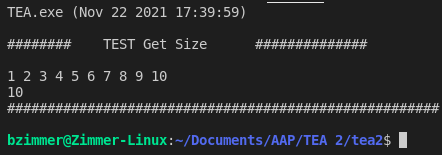


Avec une fonction de teste dans l’ordre:



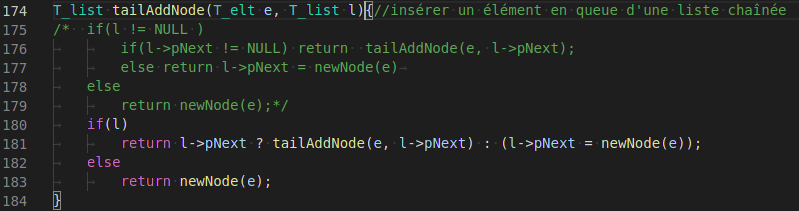
La fonction essentiellement fait la récursivité jusqu’au dernier élément et après retourne tout en augmentant un a chaque interaction de la fonction.

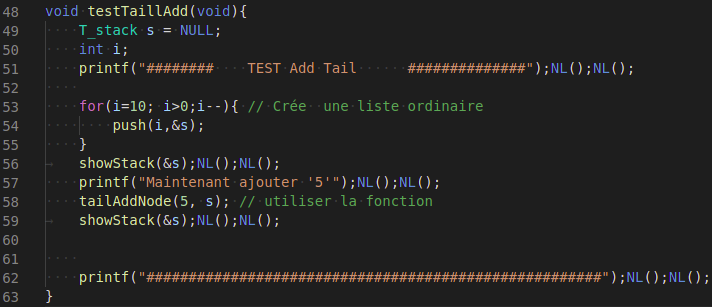
Le résultat:



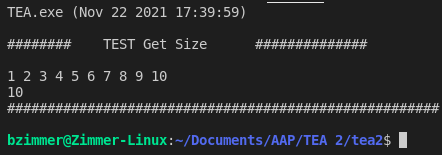
2.2 Fonction “tailAddNode”

Le code pour ces fonctions est dans l'ordre:

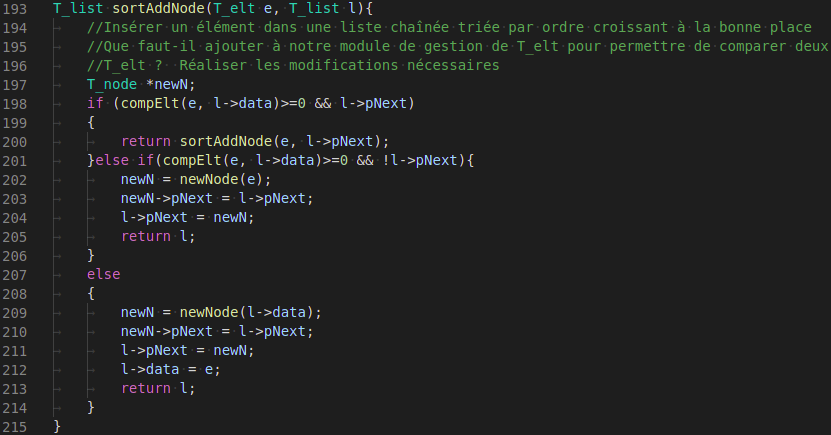


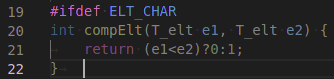
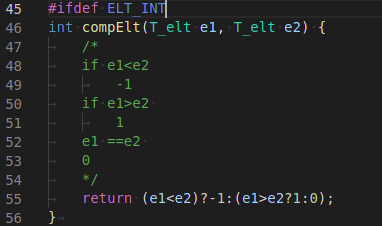
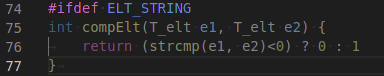
Le principe de la première fonction est assez simple, faites défiler toute la liste chaînée jusqu'au dernier terme, dès que vous le trouvez, un nouvel élément est ajouté à la liste. Pour vérifier le fonctionnement il y a la fonction de test est dans l’ordre:

Le résultat:



2.3 Fonction “sortAddNode”

Suivre, le code de cette fonction:

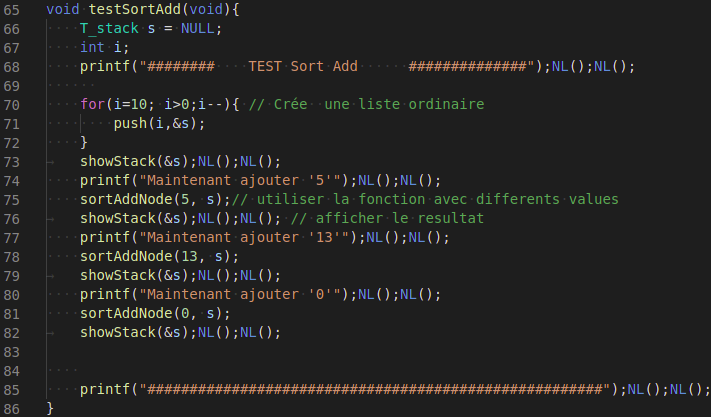
Le principe est essentiellement de passer par chaque vecteur, et à chaque avance, de faire une comparaison entre cet élément de la liste, et l'élément à placer. La fonction compElt est utilisée pour comparer. Le code de cette fonction est en séquence. La fonction comElt a trois formes définies par le variables: ELT\_INT, ELT\_CHAR, ELT\_STRING, de la forme:

Le principe est le même que la fonction "sortAddNode", cependant, au lieu

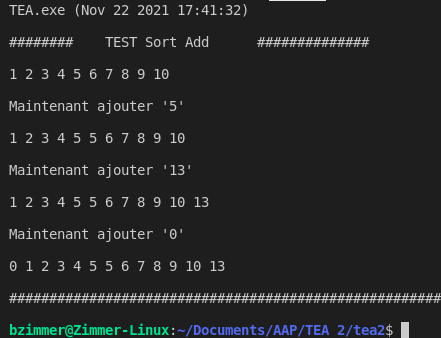
d'utiliser la boucle, nous utilisons la récursion pour faire défiler toute la liste.

Les résultats de ces deux demandes sont présentés dans l'ordre suivant.

Le dossier de test est ici:

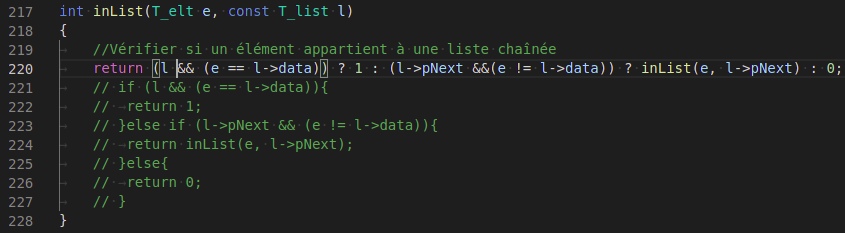


Le résultat:

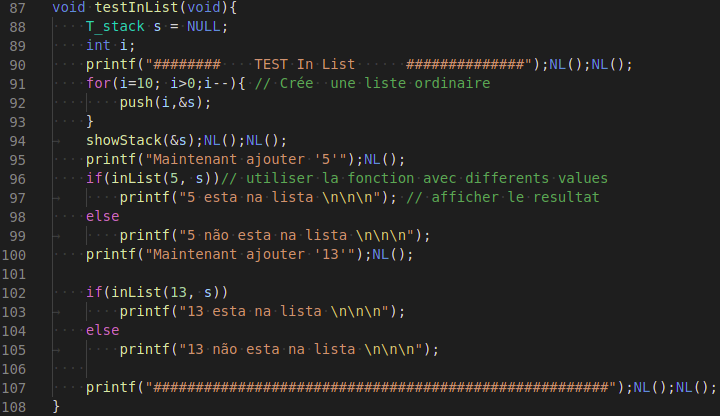


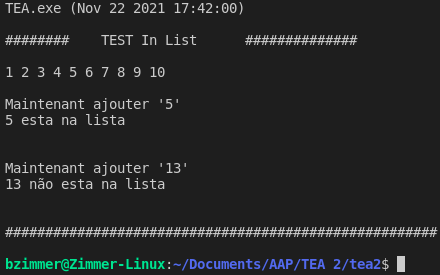
2.4 Fonction “inList”

Le code pour cette fonction est dans l'ordre.

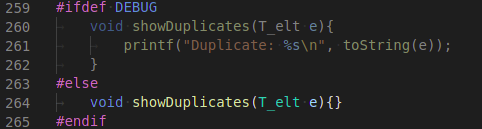


L'idée de base des deux fonctions est de vérifier toute la liste, jusqu'à ce que vous trouviez un élément similaire au paramètre précédent. Dans le premier cas, la liste est recherchée de manière itérative, et dans le second de manière récursive.

Le dossier de test est ici:  
 Le résultat:



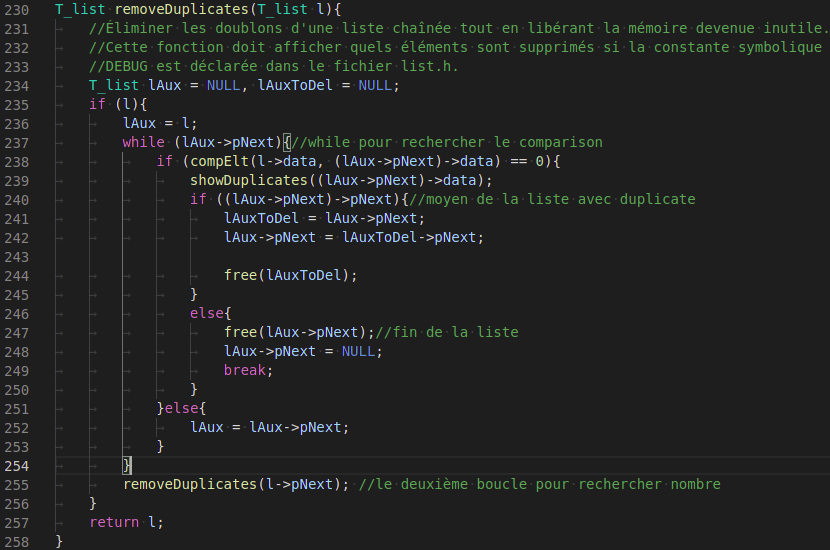
2.5 Fonction “removeDuplicates”

Avant d'afficher le code de la fonction, l'extrait de code suivant montre le code d'une fonction auxiliaire ‘showDuplicates“, qui a la fonctionnalité d'afficher un texte sur le terminal grâce à la création d'une constante.

Ainsi, si l'indicateur a défini la constante "DEBUG", la fonction "showDuplicates"

affichera un texte, et sinon, rien ne sera affiché.

La fonction "removeDuplicates" se déroule dans l'ordre.



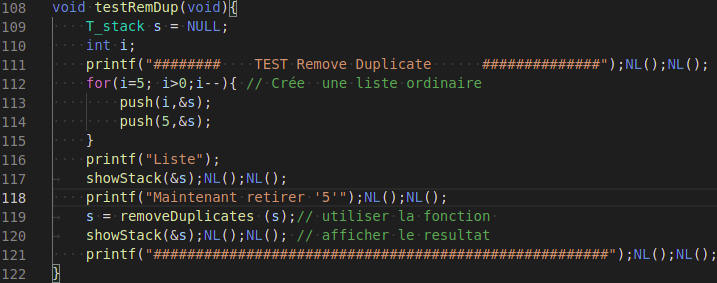
Le principe de cette fonction est de parcourir l'ensemble de la liste, et dans

chaque élément, de faire une vérification avec le reste de la liste, afin de connaître

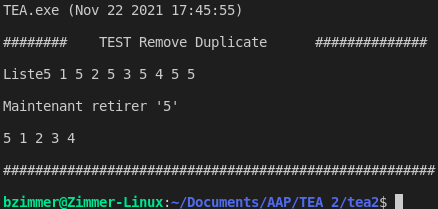
l'existence d'éléments égaux. Si c'est le cas, la fonction désactive de mémoire

l'élément en double et réactive les liens avec la liste.

La fonction de test:

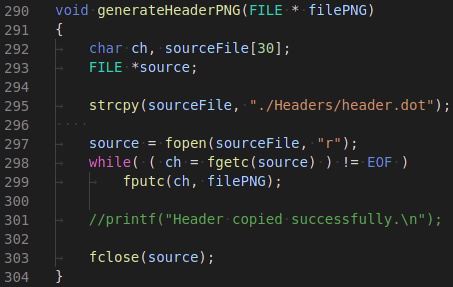


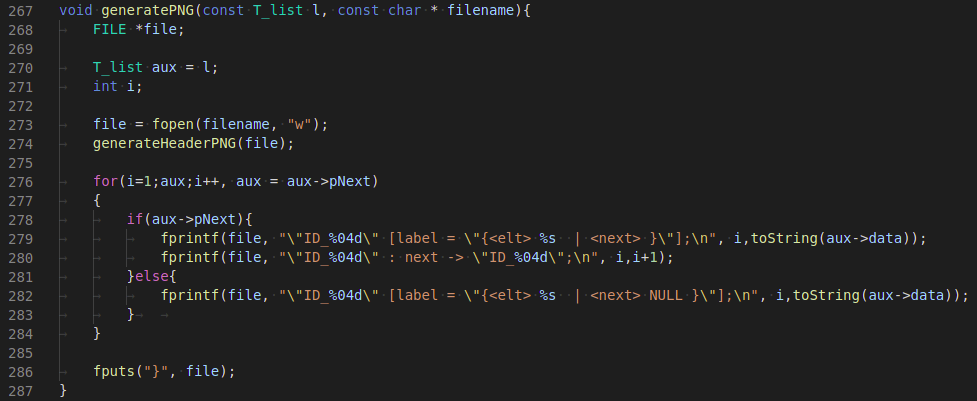
Et le résultat:



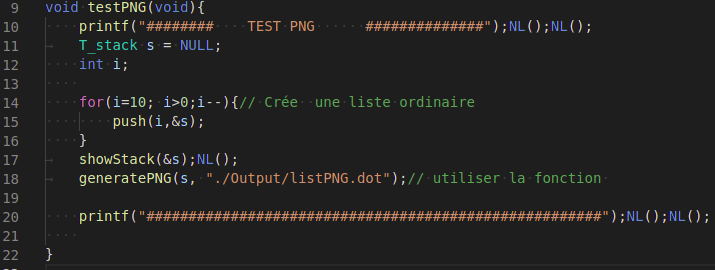
EXERCICE 3

L'objectif de cet exercice est de créer un fichier .dot contenant les informations d'une liste chaînée, et de convertir ensuite ce fichier en fichier .png. En principe, de la même manière que vous interagissez avec des fichiers .txt en C, vous manipulez des fichiers .dot. En utilisant des fonctions telles que "fopen", "fscanf", "fprintf".

La première fonction développée a été "generateHeaderPNG", qui vise à créer une le header du document .dot. Son code est séquentiel.

Dans la fonction principale, un texte est affiché dans le terminal, annonçant la fin de la création du fichier. Dans l'un des processus, la fonction "generatePNG" est appelée, qui est présentée en séquence.

Cette fonction a les principes suivants : créer le fichier, appeler la fonction "generateHeaderPNG" (une fonction qui crée l'en-tête par défaut du fichier .dot), recevoir la liste, et faire défiler la liste entière. Pendant que la fonction fait défiler la liste, à chaque processus, elle crée une ou deux nouvelles lignes dans le code, selon qu'il s'agit d'un élément du milieu ou de la fin de la liste. Chaque ligne qui est placée dans le fichier final est composée de parties très similaires.

Le dossier de test est:

Et le résultat:



CONCLUSION

Après tous ces développements, il est possible de remarquer la grande évolution que nous avons eue par rapport à la gestion des listes chaînées. Il existe maintenant une variété de fonctions déjà définies et extrêmement utiles dans l'utilisation des listes.

REFERENCES

(1) Demko, C. (16 de Avril de 2020). Structures de données en C/Les listes simples. Fonte: Wikibooks: https://fr.wikibooks.org/wiki/Structures\_de\_donn%C3%A9es\_en\_C/Les\_listes\_

simples