## Consignes générales pour les TDs

FAITES BIEN LES SCHEMAS DEMANDES, **CE N’EST PAS OPTIONNEL** ET VOUS AIDERA À VISUALISER ET COMPRENDRE LES ALGORITHMES

POUR LES EXERCICES COMPORTANT DE LA REDACTION DE CODE EN C, **UTILISEZ UN EDITEUR DE TEXTE OU UN IDE**, AFIN DE POUVOIR RECUPERER VOS TRAVAUX POUR LES TPs.

Les exercices précédés d’un symbole losange ◆ sont obligatoires.

# TD1 : rappels sur les structures, les pointeurs et les listes

### Thème 1 – Structures – objectif : manipulation de structures et des champs. Distinguer les notations . et ->. Passage par valeur et par adresse

#### Exercice 1) Soit la définition de type suivante

**struct s\_product**

**{**

**char name[30] ;**

**float base\_price ; // prix Hors taxe, Excl. Taxes price**

**float VAT\_rate ; // taux de TVA**

**} ;**

**typedef struct s\_product t\_product ;**

Et la définition de variable associée :

**t\_product prod ;**

question 1) faites un schéma pour visualiser la variable **prod** et ses champs

question 2) écrivez les instructions pour initialiser la variable **prod** avec « Smartphone » pour le champ **name** , 453 pour le champ **base\_price** et 20 pour le champ **VAT\_Rate**

Attention, le champ name est une chaine de caractères, il faut donc utiliser une fonction spécifique à ce type particulier.

Question 3) écrivez une fonction **displayProduct()** qui affiche un produit, en ajoutant l’affichage du prix TTC.

Question 4) écrivez une fonction **displayProductPtr()** qui affiche un produit à partir d’un pointeur sur un produit, en ajoutant l’affichage du prix TTC.

Question 5) dans le programme suivant, faites les appels corrects à **displayProduct()** et **displayProductPtr()**

**struct s\_product**

**{**

**char name[30] ;**

**float base\_price ; // prix Hors taxe, Excl. Taxes price**

**float VAT\_rate ; // taux de TVA**

**} ;**

**typedef struct s\_product t\_product ;**

**int main()**

**{**

**t\_product prod ;**

**// initialisation vue en question 1**

**// appelez displayProduct()**

**// appelez displayProductPtr()**

**return 0;**

**}**

### Thème 2 – Pointeurs – objectif : initialisation valide, notations \* . et ->

#### Exercice 2) soit le morceau de programme suivant :

**int \*ptr1 ;**

**float \*ptr2 ;**

**t\_product \*ptr3 ;**

**float x ;**

**x = 1.70732 ;**

**ptr1 = NULL ;**

**ptr2 = &x ;**

**ptr3 = (t\_product \*)malloc(1 \* sizeof(t\_product)) ;**

question 1) En utilisant le type de visualisation de variables vu en cours, faites le schéma de visualisation des variables de ce programme.

question 2) Quels sont, parmi les pointeurs de ce programme, ceux dont on peut utiliser le contenu ? justifiez votre réponse.

Question 3) on veut mettre la valeur 5.50 dans le champ **VAT\_rate** du contenu de **ptr3**. En utilisant la visualisation, écrivez l’instruction correspondante.

### Thème 3 - Listes simplement chaînées – objectif : comprendre les structures utilisées, faire des schémas avec les pointeurs

Rappel : comme indiqué en cours, on utilisera, pour les exercices, les types :

**struct s\_cell  
{  
 int value;  
 struct s\_cell \*next;  
};  
  
typedef struct s\_cell t\_cell;**

et

**typedef struct s\_std\_list  
{  
 t\_cell \*head;  
} t\_std\_list;**

#### Exercice 3)

Écrivez la fonction **t\_cell \*createCell(int);** utilisée en cours pour le chaînage en tête de liste

#### Exercice 4)

Faites la visualisation / représentation d’une liste contenant 3 cellules avec les valeurs 5, 4, et 8 (de la première à la dernière)

#### Exercice 5)

Faites la visualisation/représentation des étapes de chaînage en tête pour le programme suivant :

**t\_std\_list mylist ;**

**t\_cell \*newcell ;**

**mylist.head = NULL;**

**newcell = createCell(8);**

**newcell->next = mylist.head;**

**mylist.head = newcell;**

**newcell = createCell(4);**

**newcell->next = mylist.head;**

**mylist.head = newcell;**

#### Exercice 6)

On rappelle que l’instruction **free(un\_pointeur)** libère l’espace préalablement alloué par **malloc()** désigné par le pointeur. Son contenu n’est donc plus accessible.

On utilisera la fonction de chaînage en tête de liste vue en cours : **addHeadStd()**

Note : **addHeadStd()** appelle **createCell()** , qui appelle à son tour **malloc()** . Il y a donc bien allocation d’une cellule à chaque passage dans la boucle for du programme suivant :

Considérons ce programme :

**t\_std\_list mylist ;**

**int cpt ;**

**mylist.head = NULL ;**

**for (cpt = 3 ; cpt < 10 ; cpt++)**

**{**

**addHeadStd(&mylist, cpt) ;**

**}**

**free(mylist.head);**

question 1) Dans quel état est la liste juste après la boucle for ? (Vous pouvez utiliser un schéma)

question 2) Dans quel état est la liste après l’instruction **free(mylist.head)** ? Pourquoi est-ce potentiellement un problème ?

*Exercice 7)* Expliquez comment on peut ‘perdre la tête’ d’une liste. Comment y remédier ?

### Thème 4 : Fonctions standard sur les listes - objectifs : rappels sur les traitements usuels

**Rappel : pour écrire correctement les fonctions, posez-vous les questions suivantes :**

* Quelles sont les valeurs dont elle a besoin (ce sont ses paramètres) ?
* La fonction modifie-t-elle des paramètres ? (si oui il faut transmettre leur adresse et le paramètre est un pointeur)

Quelle valeur retourne la fonction (c’est son type de retour) ? Si la fonction ne retourne rien, son type de retour est **void**

#### Exercice 8)

Écrire une fonction itérative nommée **flipVals()**, à qui l’on transmet une liste, qui ajoute 1 à toutes les valeurs paires et enlève 1 à toutes les valeurs impaires(). **Attention, ici la liste n’est pas modifiée (la liste stocke juste où se trouve la première cellule), on ne modifie que les valeurs des cellules !**

#### Exercice 9)

Écrire une fonction itérative nommée **countVal()** , à qui l’on fournit une liste, une valeur, et qui indique combien de fois la valeur se trouve dans cette liste.

#### Exercice 10)

Écrire une fonction nommée **isValInList()** à qui l’on fournit une liste et une valeur, et qui indique si la valeur est stockée dans la liste.

#### Exercice 11)

Écrire une fonction nommée **removeValFromList()** à qui l’on fournit une liste (à modifier) et une valeur, et qui supprime la première occurrence de la valeur dans la liste, si elle y est stockée.

**Note :**

On testera d’abord que la liste n’est pas vide

On testera ensuite le cas particulier où la valeur est en tête de liste, et donc il faudra modifier le champ ‘head’

Il ne reste ensuite que le cas ‘général’

#### Exercice 12)

Écrire une fonction nommée **removeAllValFromList()** à qui l’on fournit une liste (à modifier) et une valeur, et qui supprime toutes les occurrences de la valeur dans la liste.

Vous utiliserez les fonctions présentées dans les exercices 10 et 11, même si vous ne les avez pas rédigées.

### Thème 4 - Fonctions récursives sur les listes – objectif : connaître la forme récursive, placer les appels récursifs

#### Exercice 13)

Question 1) Écrire une fonction récursive **int** **checkOrderedCellRec(t\_cell\*) ;** qui indique si les éléments rangés dans les cellules de la liste sont classés en ordre croissant.

Question 2) Écrire une fonction **int check0rderedListRec(t\_std\_list) ;** qui utilise **checkOrderedCellRec()** pour faire le premier appel récursif.