TP 2

Exercice 1

Soit un étudiant représenté par la structure suivante :

```
struct etudiant{
 int numEtudiant;
 int age;
 float moyenne;
};
```

1. Créez les étudiants suivants :
   — de manière statique : seb (numéro étudiant : 125872) agé de 18 ans et ayant $10,25$ de moyenne.
   — de manière dynamique : pauline (numéro étudiant : 4875564) agée de 20 ans et ayant $12,36$ de moyenne.
   — de manière dynamique : laure (numéro étudiant : 4148784) agée de 19 ans et ayant $8,41$ de moyenne.
   — de manière dynamique : Valentin (numéro étudiant : 4785129) agée de 20 ans et ayant $15,47$ de moyenne.
2. Ecrivez la procédure `afficherEtudiant` permettant l'affichage à l'écran des informations relatives à un étudiant passé en paramètre.
   — Vous passerez en paramètre l'objet étudiant lui même :
   `void afficherEtudiant(etudiant etu);`
   — Vous passerez en paramètre l'adresse d'un objet étudiant :
   `void afficherEtudiant(etudiant *etu);`

   Vous utiliserez chacune des procédures pour afficher les informations relatives à Pauline et à Seb.
3. Ecrivez la procédure `echangerEtudiant` permettant d'échanger les informations de deux étudiants passés en paramètre.
4. Modifiez la structure *etudiant* afin de chaîner les étudiants les uns aux autres. Ecrivez (dans le main) les instructions permettant de chaîner les étudiants créés dynamiquement par ordre croissant de moyenne.
5. Soit une liste chainée d'étudiant. Vous écrirez les procédures :
   — `afficherEtudiant(etudiant *listeEtu)` qui affiche les informations relatives à chaque étudiant de la liste chainée passée en paramètre.
   — `ajouterEtudiant(etudiant **listeEtu, int numEtu, int age, float moy);` qui ajoute en tête de la liste chainée un étudiant dont les informations sont passées en paramètre.

Exercice 2

Dans cet exercice on se propose de modéliser le caddie d'une ménagère. Un produit est caractérisé par un nom (une chaîne de caractères représentée par une variable de type string), un prix et un poids.

1. Expliquez pourquoi il n'est pas judicieux de gérer le caddie sous la forme d'un tableau de produits. Proposez une autre structure de données et faites un schéma de cette dernière.
2. Ecrivez la fonction `creer_produit` qui se charge de retourner un pointeur sur un produit dont le nom, le prix et le poids sont passés en paramètres. Les chaînes de caractères seront dupliquées. Donnez un exemple d'appel de cette fonction.
3. Ecrivez la procédure `ajouter_produit` qui se charge d'ajouter un produit passé en paramètre au caddie de la ménagère.
4. Ecrivez la fonction `payer` retournant le prix total du caddie de la ménagère.
5. Ecrivez la procédure `vider_caddie` vidant un caddie passé en paramètre.

   Pour ranger ses courses dans le caddie, la ménagère préfère disposer les objets lourds au fond du caddie et les objets légers au dessus.

6. Proposez une modification de la procédure `ajouter_produit` afin de maintenir triés par poids croissants les produits dans le caddie de la ménagère.
7. Ecrivez la procédure `retirer_produit` qui retire le ou les produits dont le nom est passé en paramètre (sous la forme d'une variable de type string).

   Pour ranger ses courses dans sa voiture, la ménagère range en premier les objets lourds et ensuite les objets légers.
8. Expliquez quel problème cela pose lors de l'utilisation du caddie.
9. Proposez une modification de la structure de données et faites un schéma de cette dernière.
10. Quelles modifications sont à apporter à la procédure `ajouter_produit` et `retirer_produit` ?
11. Faites un programme de test permettant de valider l'utilisation de vos fonctions et procédures.

Exercice 3

Dans cet exercice on se propose de ≪ tokeniser ≫ (i.e. découper une chaine de caractères en mots) une chaine passée en paramètre. La chaine de caractères est représentée par un tableau de caractères terminé par le caractère $'0'$. Pour vos tests, vous pouvez utiliser la chaine de caractères suivante :

```
char texte [] = "   bonjour j'espere que  vous allez bien.  "
```

Un mot sera représenté par une structure comportant un pointeur sur le premier caractère du mot et un pointeur sur le caractère suivant le dernier caractère du mot.

```
struct mot {
 char *debut;
 char *fin;
};
```

1. Ecrivez une procédure `visualiserMot` affichant une suite de caractères. Cette procédure prendra en paramètre un mot.

   ```
   void visualiserMot(mot m);
   ```
2. Ecrivez une fonction `tokeniserChaine` retournant un tableau de mots (les champs du dernier mot pointeront sur *nullptr* indiquant la fin du tableau). Les mots sont séparés les uns des autres par un (ou plusieurs) espaces.

   ```
   mot *tokeniserChaine(char *chaine);
   ```
3. Ecrivez une fonction `estPresent` retournant *vrai* si un caractère donné, est présent dans le mot passé en paramètre.

   ```
   bool estPresent(char c,mot m);
   ```
4. Ecrivez une fonction `indexerChaine` retournant un tableau de listes chainées de mots. Pour une lettre 'a' de l'alphabet, ce tableau fournira une liste (chainée) de pointeurs sur les mots dans lesquels apparaissent la lettre 'a'. Vous déclarerez la liste chainée de la manière suivante :

   ```
   struct eltMot {
     mot *m;
     eltMot *suivant;
   };
   ```

   Cette procédure prendra en paramètre le tableau de mots retourné par la fonction `tokeniserChaine`. **Vous penserez à réutiliser les fonctions et/ou procédures précédentes.**

   ```
   eltMot** indexerChaine(mot *mots);
   ```
5. Ecrivez la procédure `afficherMotsCarac` affichant pour un caractère de l'alphabet passé en paramètre, la liste des mots dans lesquels ce caractère apparait.

   ```
   void afficherMotsCarac(char c,eltMot **index);
   ```
6. Ecrivez la procédure `afficherMots` affichant pour chaque caractère, la liste des mots dans lesquels ce caractère apparait.

   ```
   void afficherMotsCarac(eltMot **index);
   ```
7. Ecrivez la procédure `desallouerStructures` permettant de libérer les structures allouées aux questions 2 et 4.

   ```
   void desallouerStructure(mot *mots,eltMot **index);
   ```