L'objectif du TP d'aujourd'hui est d'apprendre à manipuler les pointeurs et les structures. En particulier dans ce TP, l'intérêt des pointeurs devrait devenir beaucoup plus clair. Dans ce TP vous devez au moins terminer les exercices 1 et 2 qui sont au programme du contrôle TP, ainsi que les TPs précédents.

Exercice 1 : Utilisation des tableaux sans erreurs de segmentation

Comme vous avez déjà pu le constater, la manipulation des tableaux est plus simple en java. Vous pouvez par exemple demander la taille d'un tableau, chose qui n'est pas possible en C. Mais aussi, lorsque vous demandez une case qui est en dehors du tableau, en java vous avez l'erreur explicite :
`Exception in thread "main" java.lang.IndexOutOfBoundsException`,
alors qu'en C vous avez simplement un : `Segmentation fault (core dumped)`.

Aujourd'hui il est temps de comprendre ce que veut dire le fait de "pouvoir tout faire" en C, mais aussi de comprendre ce que les autres langages de programmation cachent aux codeurs que ne cache pas le C.

Q1. Jusqu'à maintenant, si l'on voulait créer un tableau de caractères de taille 4, on demandait `char tab[4];`. Si cela fonctionne effectivement, il pose le problème que ce tableau à une taille qui doit être une constante. Comme nous avons vu en cours, en utilisant `malloc`, il est possible de réserver dynamiquement de la place en mémoire pour stocker par exemple un tableau (cf. Cours 3 Slide 6). Cet espace mémoire qui peut ensuite être utilisé comme un tableau (cf. Cours 2 Slide 15).

Pour commencer, nous allons manipuler rapidement cette nouvelle manière de voir les tableaux :

- En utilisant `malloc`, créez un tableau de caractères de taille 4 (rappel : la taille en mémoire d'un caractère est donnée par `sizeof(char)`).
- Remplissez-le, en utilisant l'approche par pointeur, des valeurs `'a'`, `'b'`, `'c'`, `'d'`.
- Affichez ce tableau.

Q2. Pour effectuer ce genre d'opérations basiques, il convient d'avoir des fonctions qui permettent de les automatiser.

- Écrivez une fonction `char* nouveau_tableau (int taille);` qui crée un tableau de caractères de taille `taille` et retourne un pointeur vers le premier élément.
- Écrivez une fonction `void initialise_tableau (char* tableau, int taille, char car);` qui initialise toutes les cases du tableau `tableau` de taille `taille` avec la valeur de `car`.
- Écrivez une fonction `void affiche_tableau (char* tableau, int taille);` qui affiche les éléments du tableau `tableau` de taille `taille` séparés par des espaces.
- Testez vos fonctions en créant un tableau de taille 4, en initialisant ses valeurs à `'a'` puis en modifiant la case d'indice 2 pour qu'elle ait la valeur `'e'` et enfin en affichant le résultat.

Q3. Dans la question précédente, vous avez réservé de la mémoire que vous n'avez pas libérée. On rappelle (cf. Cours 3 Slide 9) qu'il est indispensable de libérer la mémoire que vous réservez. Écrivez une fonction `void liberation_du_tableau (char* tableau);` qui libère l'espace mémoire réservé pour le tableau passé en argument. Notez que vous n'avez pas besoin de connaitre sa taille, le système d'exploitation se charge de récupérer tout ce qui lui revient comme un grand.

Corriger votre test pour correctement libérer toute la mémoire que vous avez réservée. Essayez d'afficher le tableau après l'avoir libéré.

Q4. Lorsque vous allez lire ou écrire dans un tableau, vous devez vous assurer que l'indice que vous donnez est bien un indice appartenant au tableau. Ce test n'est pas fait par défaut en C (moins de tests implique un code plus rapide... mais aussi plus de possibilités d'erreurs), mais rien n'empêche de l'implémenter.

- Écrivez `void place_dans_tableau (char* tableau, int taille, int indice, char car);` qui place le caractère `car` dans la case d'indice `indice` du tableau `tableau` de taille `taille`. Si `indice` correspond à une case n'appartenant pas au tableau, la fonction écrira (via `printf`) un message d'erreur et arrêtera le programme via l'invocation de la commande `exit(1)`. Attention : `exit` est une fonction présente dans une librairie autre que `stdio`, il va donc falloir ajouter cette librairie. Trouvez le nom de cette dernière en regardant `man 3 exit`.
- Écrivez une fonction `char lecture_du_tableau (char* tableau, int taille, int indice);` qui retourne le caractère de la case d'indice `indice` du tableau `tableau` de taille `taille`. Si `indice` correspond à une case n'appartenant pas au tableau, la fonction écrira (via `printf`) un message d'erreur et arrêtera le programme (via `exit`).
- Testez vos fonctions en plaçant dans le tableau `'t'` dans la case d'indice 1 et `'u'` dans la case d'indice 5. Puis en essayant de lire les cases de l'indice 0 à l'indice 6.

Exercice 2 : Une implémentation des tableaux dans d'autres langages

Q1. En utilisant les fonctions de l'exercice précédent, vous obtenez une certaine protection sur l'utilisation de vos tableaux. En pratique, ces protections sont implémentées par défaut dans de nombreux langages comme par exemple... le java ! Vous constaterez aussi que les fonctions précédentes demandent (presque) toutes le pointeur vers la première case du tableau et la taille du tableau sur lequel elles travaillent. Si en C, on a toujours le pointeur, la taille elle, c'est le codeur qui doit la redonner à chaque fois, ce qui n'est pas le cas en java. En réalité l'astuce du java est le fait que dans l'objet `array` il y a un attribut correspondant au tableau lui-même, mais aussi un attribut correspondant à sa taille ! En C, il est tout à fait possible de faire la même chose en utilisant des structures.

Créez une structure que vous nommerez `super_tableau_t` qui contient deux attributs, le premier, `char* tableau`, est le pointeur vers le premier élément du tableau de `char` et le deuxième, `int taille`, est un entier qui stockera la taille d'un tableau. (AIDE : Pour vous aider avec les structures, pensez à regarder l'exemple "structure" sur le moodle.)

Notes : La définition des structures doit être placée entre les `#include` des bibliothèques standard (stdio, stdlib, etc.) et les signatures des fonctions de votre code.

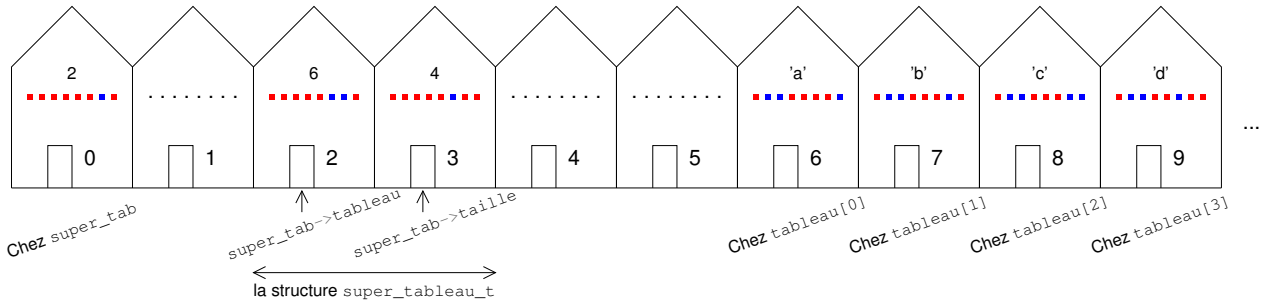
Q2. Nous allons procéder à un test d'utilisation du type `super_tableau_t` :

- Créez une variable `super_tab` qui est un pointeur vers un élément de type `super_tableau_t`.
- À l'aide de malloc, réservez un espace de taille adaptée à stocker un élément de type `super_tableau_t` et stockez l'adresse dans la variable `super_tab`.

Une fois ses étapes faites, vous avez réserver l'espace pour stocker la taille de `super_tab` (accessible en utilisant `super_tab->taille`) et un pointeur vers le tableau qui stockera les données (accessible en utilisant `super_tab->tableau`). ATTENTION : Il n'y a actuellement pas d'espace mémoire réservé pour le tableau, uniquement de l'espace mémoire pour stocker le pointeur vers ce tableau, il faut donc utiliser à nouveau malloc pour réserver cet espace, ou mieux encore, la fonction que nous avons créé précédemment : `nouveau_tableau`.

- Initialisez `super_tab->taille` pour qu'il contienne cette valeur 4.
- Déclarez un tableau de caractères de taille 4 qui sera ciblé par `super_tab->tableau`.
- Remplissez ce tableau de caractères.
- Affichez ce tableau de caractères.

AIDE : Pour bien comprendre comment tout est stocké en mémoire, je vous ai fait une représentation de la mémoire en dessous. Notez que `super_tab` est un pointeur vers une structure de type `super_tableau_t` et que `tableau` est aussi un pointeur car un tableau. Le contenu du tableau en lui-même est choisi au hasard (du copier-coller des dessins du cours).



Q3. Comme précédemment, il est d'usage d'avoir des fonctions qui créent correctement un élément de la structure, qui l'initialise, qui l'affiche et qui libère la mémoire à la fin de son utilisation. Pensez à utiliser les fonctions que vous avez déjà écrites.

- Écrivez une fonction `super_tableau_t* nouveau_super_tableau (int taille);` qui crée un élément de type `super_tableau_t` et renvoie un pointeur vers cet élément. Pensez à correctement réserver l'espace nécessaire pour le tableau qui doit être un tableau de caractères de taille `taille` ainsi qu'à correctement initialiser les attributs `tableau` et `taille` de l'élément de type `super_tableau_t` que vous venez de créer.
- Écrivez une fonction `void initialise_super_tableau (super_tableau_t* super_tab, char car);` qui initialise, avec la valeur de `car`, toutes les cases du tableau correspondant à l'attribut `tableau` du `super_tableau_t` ciblé par `super_tab`.
- Écrivez une fonction `void affiche_super_tableau (super_tableau_t* super_tab);` qui affiche les éléments du tableau correspondant à l'attribut `tableau` du `super_tableau_t` ciblé par `super_tab`.
- Écrivez une fonction `void liberation_du_super_tableau (super_tableau_t* super_tab);` qui libère l'espace mémoire réservé pour stocker le `super_tableau_t` ciblé par `super_tab`. Pensez à libérer explicitement aussi l'espace réservé pour le tableau ciblé par l'attribut `tableau`.
- Testez vos fonctions en créant un `super_tableau_t` de taille 4, en initialisant les valeurs de son tableau à `'a'` puis en modifiant la case d'indice 2 pour qu'elle ait la valeur `'e'` et enfin en affichant le resultat.

Q4. Comme précédemment, il faut aussi une fonction pour lire et écrire dans votre super tableau.

- Écrivez `void place_dans_super_tableau (super_tableau_t* super_tab, int indice, char car);` qui place le caractère `car` dans la case d'indice `indice` du tableau correspondant à l'attribut `tableau` du `super_tableau_t` ciblé par `super_tab`. Si `indice` correspond à une case n'appartenant pas au super tableau, la fonction écrira (via `printf`) un message d'erreur et arrêtera le programme (via `exit`).

— Écrivez une fonction `char lecture_du_super_tableau (super_tableau_t* super_tab, int indice);` qui retourne le caractère de la case d'indice `indice` du tableau correspondant à l'attribut `tableau` du `super_tableau_t` ciblé par `super_tab`. Si `indice` correspond à une case n'appartenant pas au super tableau, la fonction écrira (via `printf`) un message d'erreur et arrêtera le programme (via `exit`).
— Testez vos fonctions en plaçant dans le super tableau `'t'` dans la case d'indice 1 et `'u'` dans la case d'indice 5. Puis en essayant de lire les cases de l'indice 0 à l'indice 6.

Q5. Et maintenant la question que vous attendez tous : Écrivez une fonction `int taille_super_tableau(super_tableau_t* super_tab);` qui renvoit la taille du tableau correspondant à l'attribut `tableau` du `super_tableau_t` ciblé par `super_tab`.

Q6. Vous voilà désormais avec une implémentation des tableaux qui ressemble à celle que vous pouvez connaitre en java. Il faut ainsi être bien conscient que les différents langages de programmation que vous pouvez utiliser vous cachent un certain nombre de choses que le codeur peut être content de ne pas avoir à se préoccuper mais qui sont tout de même gérées par le langage. Mais ce contrôle qui est ajouté dans les différents langages, même s'il est très pratique pour le codeur car il résout rapidement de nombreux problèmes de codage, vient avec un prix. En effet, plus de contrôle implique plus de code à exécuter (même si ce n'est pas vous qui l'écrivez) et donc plus de temps de calcul.

La raison qui fait que le C est un des langages les plus rapides en exécution est que ces contrôles sont inexistants, mais c'est aussi ce qui fait qu'il est beaucoup plus dur de programmer en C et en particulier de débugger les erreurs.

(Non non, il n'y a rien à faire dans cette question, juste lire ce petit texte.)

Exercice 3 : Chaine de caractères

Une chaine de caractère en C est simplement un tableau dans lequel sont stockés des caractères et la fin de la chaine est symbolisée par le caractère `'\0'` (ce caractère est le caractère 0 de la table ascii. cf. `man ascii`).

À la différence d'un tableau classique, lorsque l'on déclare une chaine de caractère, on déclare souvent un tableau de taille beaucoup plus grande que la chaine en question. Pour rappel, dans le TP précédent, lorsque vous demandiez une entrée utilisateur, vous déclariez un tableau de taille 10 alors que vous attendiez 3 caractères seulement, la colonne, la ligne et le caractère `'\0'`. Cette précaution est souvent nécessaire et on va voir ici comment on peut facilement manipuler ces chaines.

Dans cet exercice nous présenterons une manière de gérer et manipuler des chaines de caractères.

Q1. Dans l'exercice précédent, vous avez créé le type `super_tableau_t`. Ce type permet de gérer et d'utiliser des tableaux de caractères. Cela tombe bien, c'est exactement ce que vous allez vouloir utiliser pour manipuler des chaines de caractères. Néanmoins vous allez apprécier d'avoir un type plus explicite pour vos chaines, typiquement `chaine_t`. Utilisez l'instruction `typedef super_tableau_t chaine_t;` pour créer un type `chaine_t` qui est simplement un nouveau nom pour le type `super_tableau_t`.

Q2. À nouveau, pour travailler avec le type `chaine_t`, vous allez écrire des fonctions. Gardez en tête que les types `chaine_t` et `super_tableau_t` étant strictement les mêmes, vous pouvez utiliser vos fonctions prévues pour les supers tableaux sur vos chaines.

On rappelle qu'en C, une chaine de caractères doit toujours terminer par un `'\0'`, et donc en particulier le tableau qui stocke votre chaine doit contenir ce caractère. Écrivez une fonction `int chaine_est_valide (chaine_t* chaine);` qui retourne 1 si ma chaine contient effectivement le caractère `'\0'` et 0 si elle ne le contient pas.

Q3. Maintenant que vous êtes capable de savoir si une chaine est valide, l'étape suivante est d'en connaitre sa taille. Comme expliqué précédemment, cette taille ne correspond pas forcément à la taille du tableau qui la contient. La taille d'une chaine est donc le nombre de caractères présent avant la première occurrence de `'\0'`. Notez bien que le caractère `'\0'` lui-même ne fait pas partie de la chaine.

Écrivez une fonction `int taille_de_la_chaine (chaine_t* chaine);` qui retourne la taille de la chaine de caractères stockée dans `chaine`. Cette fonction retournera -1 si la chaine n'est pas valide. Testez cette fonction sur les exemples créez pour les supers tableaux (qui ne contiennent pas le caractère `'\0'`), ajoutez ce caractère et recommencez.

Q4. Écrivez une fonction `void affiche_chaine (chaine_t* chaine);` qui vous permet d'afficher la chaine de caractère stockée dans `chaine` dans le cas où la chaine est valide et qui n'affiche rien si ce n'est pas le cas.

Q5. Un point intéressant des chaines de caractères est de pouvoir les allonger si le besoin s'en fait sentir.

Écrivez une fonction `void ajoute_caractere (chaine_t* chaine, char car);` qui ajoute le caractère `car` à la fin de votre chaine de caractère. Pour le moment vous supposerez que le tableau est assez grand pour tout stocker. Testez votre fonction en créant une chaine dont le tableau est de taille `10` et qui contient `"tot"` (donc `'t'`, `'o'`, `'t'`, `'\0'` puis le reste peut être n'importe quoi). Affichez cette chaine, ajoutez un `'o'` grâce à la fonction puis réaffichez la chaine. N'oubliez pas de vous assurer qu'il y a bien un caractère `'\0'` à la fin de votre chaine.

Q6. Dans la question précédente, on est parti du principe qu'il y avait toujours de la place pour ajouter des caractères. En pratique vous réalisez bien que ça ne sera pas toujours le cas. Il existe plusieurs manières de gérer ce problème, mais la plus classique consiste à travailler avec des tableaux dynamiques. En pratique cela veut dire que si vous détectez que le tableau n'est pas assez grand (et vous avez créé plein de fonctions exprès pour ça), alors vous allez créer un tableau un peu plus grand et copier l'ancien tableau dans le nouveau de telle sorte que dans le nouveau tableau vous ayez la place pour stocker le caractère à ajouter.

Écrivez une fonction `void allonge_tableau_de_la_chaine (chaine_t* chaine);` qui augmente la taille du tableau de `1` en maintenant son contenu et en plaçant `'\0'` dans la case ajoutée. Pensez à bien libérer la mémoire de l'ancien tableau que vous n'utiliserez plus.

Q7. Réécrivez votre fonction `void ajoute_caractere (chaine_t* chaine, char car);` qui agrandit de `1` le tableau de la chaine dans le cas où ce dernier n'était pas assez grand.

Q8. Testez vos fonctions en initialisant une chaine de caractère de taille `1` contenant uniquement le caractère `'\0'` puis en ajoutant successivement les lettres `'a'`, `'b'`, `'c'` et `'d'`. Après chaque ajout, affichez la chaine ainsi que la taille du tableau qui la contient (`chaine->taille`).

Q9. Maintenant que vous êtes capable d'ajouter des caractères les uns après les autres, vous pouvez désormais concaténer deux chaines de caractères. Pour des questions de mémoire et de temps de calcul il est souvent plus efficace si l'on a deux chaines `chaine1` et `chaine2` de placer `chaine2` à la suite de `chaine1`, mais il est aussi possible de construire une troisième chaine qui contiendrait la concaténation des deux chaines.

Écrivez la fonction `void ajoute_chaine (chaine_t* chaine1, chaine_t* chaine2);` qui ajoute la chaine `chaine1` à la suite de la chaine `chaine2`. Testez votre fonction avec une première chaine contenant `"aaa"` et une deuxième chaine contenant `"bbb"`. Affichez les deux chaines avant puis après l'utilisation de `ajoute_chaine`.

Q10. Lorsque vous voulez récupérer du texte saisi par l'utilisateur, vous avez vu, pendant le TP sur le démineur, que vous pouviez utiliser la fonction `scanf`. En réalité cette fonction est déjà une fonction évoluée qui utilise une fonction plus basique qui lit caractère par caractère : `getchar`. Pour comprendre comment fonctionnent ces fonctions, il faut commencer par le début. Lorsque l'utilisateur rentre du texte au clavier, celui-ci s'affiche sur son écran et... c'est tout, en tout cas tant qu'il n'a pas validé en appuyant sur entrée. Au moment où l'utilisateur appuie sur entrée, le texte est envoyé à un buffer qui se remplit de l'ensemble du texte saisi. C'est dans ce buffer que vos fonctions `scanf` ou `getchar` vont lire. En pratique `getchar` va lire le premier caractère de ce buffer puis le supprimer du buffer (avec une fonction similaire à `supprime_caractere` et va retourner ce caractère en retour de votre fonction. Maintenant que vous savez gérer des chaines de taille dynamique, il vous est facile de récupérer la totalité du texte saisi par l'utilisateur.

Écrivez une fonction `void lire_entree_utilisateur (chaine_t* chaine);` qui prend en entrée une chaine, la réinitialise avec que des `'\0'` pour qu'il n'y ait plus rien dedans puis va insérer, en utilisant `getchar()` les caractères entrés par l'utilisateur. La fonction s'arrêtera de lire lorsque l'utilisateur aura entré le caractère `'\n'` de retour à la ligne qu'elle n'insèrera pas dans la chaine. Testez votre fonction en demandant à l'utilisateur d'écrire du texte et en le réaffichant.

Q11. Lorsque vous travaillez avec des chaines de caractères, vous aimez aussi avoir la possibilité d'insérer un caractère au milieu de votre chaine ou encore mieux une autre chaine.

- Écrivez une fonction `void insere_caractere (chaine_t* chaine, int index, char car);` qui insère le caractère `car` à la position `index` dans la chaine `chaine`. Pensez bien que l'on veut ici insérer un caractère et non pas remplacer un caractère par un autre. N'oubliez pas de vérifier que le tableau est assez grand et si ce n'est pas le cas de l'agrandir en fonction.
- Écrivez une fonction `void insere_chaine (chaine_t* chaine1, int index, chaine_t* chaine2);` qui insère la chaine `chaine2` à la position `index` dans la chaine `chaine1`.

Q12. Si l'on est capable d'insérer des caractères, on doit aussi être capable d'en supprimer.

- Écrivez une fonction `void supprime_caractere (chaine_t* chaine, int index);` qui supprime le caractère à la position `index` dans la chaine `chaine`. Pensez bien qu'il faut ensuite décaler le reste de la chaine
- Écrivez une fonction
  `void supprime_sous_chaine (chaine_t* chaine, int index_debut, int index_fin);` qui supprime, dans la chaine `chaine`, les caractères compris entre la position `index_debut` et la position `index_fin`.

Q13. Pour finir ce TP, voici une liste de fonctions supplémentaires qui existent en java et que vous pouvez désormais implémenter vous-même en C.

- Écrivez une fonction `int chaines_sont_egales (chaine_t* chaine1, chaine_t* chaine2);` qui teste si les deux chaines de caractères contiennent la même chaine. Elle renvoit `1` si c'est le cas et `0` sinon. Attention : Les tableaux n'ont pas besoin d'être de la même taille et le contenu après le `'\0'` ne doit pas être pris en compte non plus.
- Écrivez une fonction `int chaine_est_vide (chaine_t* chaine);` qui teste si la chaine est vide. Elle renvoie `1` si c'est le cas et `0` sinon.
- Écrivez une fonction `void remplace_dans_la_chaine (chaine_t* chaine, char car_ancien, char car_n` qui remplace dans la chaine chaque caractère `car_ancien` par un caractère `car_nouveau`.
- Écrivez une fonction `void chaine_en_minuscule (chaine_t* chaine);` qui transforme toutes les majuscules de la chaine `chaine` en minuscule.
- Écrivez une fonction `void chaine_en_majuscule (chaine_t* chaine);` qui transforme toutes les minuscules de la chaine `chaine` en majuscule.
- Écrivez une fonction `void bonne_case(chaine_t* chaine);` qui transforme le texte tel que la première lettre de chaque mot soit une majuscule et les autres lettres soient en minuscule.
- Écrivez une fonction `int cherche_caractere (chaine_t* chaine, char car);` qui renvoie l'indice du premier caractère `car` de la chaine, et renvoie `-1` sinon.