Ex1. Tableaux dynamiques en OCaml - 1h

Il est impossible d'ajouter ou supprimer un élément dans tableau (`array`) en OCaml. Les tableaux dynamiques (ou : redimensionnables) permettent d'ajouter un élément `e`, en recréant un tableau plus grand dans lequel on recopie tous les éléments ainsi que `e`. Voici la définition de tableau dynamique que nous allons utiliser :

```
type 'a array_dyn = { mutable t : 'a array ; mutable n : int };;
```

L'entier `n` indique le nombre de cases du tableau `t` que l'on considère comme faisant partie du tableau dynamique. Les indices au delà de `n` dans `t` sont simplement ignorés. À chaque fois que l'on voudra ajouter un élément `e` à un tableau dynamique `d` (de type `'a array_dyn` pour un type `'a` précis et fixé, par exemple `'a = float`) :

- s'il reste de la place dans le tableau, que l'on accède via `d.t` (c'est à dire si `d.n < Array.length d.t`), on met `e` dans `d.t.(n)` et on met à jour `d.n` (comment ?) ;
- sinon, on créé un nouveau tableau `t2` de taille deux fois plus grande `2*n` (on pensera à utiliser `Array.make longueur valeur_defaut` pour créer un tableau d'une longueur donnée avec une valeur par défaut), on recopie `d.t` dedans (une simple boucle `for` est votre amie), on ajoute `e` en fin du tableau `t2` en position `n+1`, puis on remplace `d.t` par `t2` dans la structure enregistrement `d`.

1. Écrire une fonction `add` ajoutant un élément `e` dans un `array_dyn d`.
2. Quelle est la complexité dans le pire cas de `add` ?
3. On suppose que l'on ajoute `n` éléments avec `add` à un tableau dynamique de taille initiale `1`. Montrer que la complexité totale de ces `n` opérations est $\mathcal{O}(n)$ (autrement dit, la complexité moyenne ou complexité amortie d'une opération est $\mathcal{O}(1)$).

Ex.2 Des petits calculs trigonométriques en C

Écrire une fichier C `TP7.exe` important `stdio.h` et `math.h`.

On rappelle qu'on compile ce fichier avec `COMPILATEUR = gcc` ou `clang` avec la ligne de commande suivante, puis on exécute le binaire produit avec la dernière ligne :

```
$ COMPILATEUR -O0 -Wall -Wextra -Wvla -Werror -fsanitize=address \
              -fsanitize=undefined -o TP7.exe TP7.c -lm
$ ./TP7.exe
```

Si une ligne affiche un résultat qui vous semble bizarre, commenter le.

1. Votre fichier contiendra une fonction `int main()` qui commence par calculer une constante `pi` valant une approximation numérique de π, de type `double` (flottant en double précision), en utilisant une formule de trigonométrie et les fonctions trigonométriques ou anti-trigonométriques qui viennent avec la bibliothèque `math`, et un affichage sur une ligne ressemblant à l'affichage suivant (en remplaçant `?` par la valeur de votre constante `pi`) :


```
pi = ?
```

Si besoin, vous consulterez la documentation de la bibliothèque `math.h` en ligne sur le site de référence `https://www.cplusplus.com/reference/cmath/`.

2. Définir une fonction `long int factorial(int n)` qui calcule la factorielle de `n`. Elle sera au choix récursive ou impérative avec une boucle `for` ou `while`. Dans votre fonction main, écrivez une boucle `for` et des `printf("%i ! = %li\n", i, factorial(i));` pour afficher des lignes comme suit, jusqu'à trouver un résultat incohérent d'une factorielle typiquement négative ou n'étant pas assez grande (à cause d'un dépassement de capacité, aussi appelé *integer overflow*).

```
0 ! = 1
1 ! = 1
2 ! = 2
3 ! = 6
etc...
```

Est-ce qu'un `runtime error` était là pour vous prévenir de ce dépassement de capacité ?

3. En utilisant votre fonction `factorial`, et une somme partielle de `k=0` à `k=nMax` de la série définissant l'exponentielle (c'est-à-dire $\exp(x) = \sum_{k=0}^{+\infty} \frac{x^k}{k!}$), écrire une fonction `double exponential(double x, int nMax)`. Testez la sur des valeurs bien choisies dans le `main` avec un `nMax` qui correspond au plus grand `n` tel que votre `factorial(n)` fonctionne sans erreur (sans *overflow*) ;
4. Faites de même avec les fonctions sinus sin et cosinus cos, en utilisant des sommes partielles de leurs séries, à savoir $\cos(x) = \sum_{k=0}^{+\infty} (-1)^k \frac{x^{2k}}{(2k)!}$ et $\sin(x) = \sum_{k=0}^{+\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$. Attention à l'alternance des signes, sinon vous définissez les fonctions hyperboliques cosh et sinh. Définissez aussi une fonction tangente tan et vérifiez que $\tan(\pi/4) = 1$.

Tester les sur quelques valeurs et faites des affichages du style `printf("cos2(%f) = %f\n", x, cos2(x))` pour des valeurs de x bien choisies dans $[-\pi/2, \pi/2]$ pour vérifier votre implémentation.

Attention : sous peine de recevoir un avertissement du compilateur (qui sera donc une erreur grâce à l'option `-Werror`), il faudra appeler vos fonctions d'un autre nom que celles de la bibliothèque `math.h`, je vous suggère donc de les appeler `cos2`, `sin2`, `tan2`, etc.

5. Que se passe-t-il pour $\tan(\pi/2)$? Essayez de vérifier les deux comportements de la limite à gauche et à droite. Expliquer vos observations.
6. *Bonus* : trouvez sur Internet le développement en série entière des fonctions arccos, arcsin et arctan, et écrivez des fonctions `double arccos(double arccos x)` avec un `nMax` valant entre 10 et 20. Tester les sur quelques valeurs et faites des affichages du style `printf("arccos2(%f) = %f\n", x, arccos2(x))` pour des valeurs de x bien choisies pour vérifier votre implémentation.
7. *Bonus* : faites de même pour les fonctions hyperboliques cosh, sinh et tanh. Comparez les