TP 5 et 6 : Un peu d'algorithmique

Avertissement : Cette planche est à faire sur deux séances. C'est à la fois un entraînement à la conception d'algorithmes et une autoévaluation.

1. Elémentaire mon cher Lock Olms… en apparence !

En se limitant uniquement à deux variables (voir les définitions ci-dessous)

```
int cpt; char carcou;
```

Ecrivez le programme qui lit une suite de caractères terminée par un point (`'.'`) et compte le nombre d'apparition de la séquence "*le*".

Vérifiez que votre programme trouve bien 3 séquences "*le*" dans la phrase :

on a vole le sac de billes de Watson.

2. Gérer des indices :

L'objectif est de trouver l'ensemble des nombres premiers compris entre 2 et 999 (un nombre premier est divisible uniquement par 1 et par lui-même). Pour cela, on appliquera la méthode suivante :

- déclarer un tableau **`t`** de 1000 cellules
- remplir ce tableau **`t`** dans l'ordre avec les entiers de 0 à 999 (chaque entier i sera stocké en **`t[i]`**. On négligera par la suite les deux premières cellules d'indice 0 et 1)
- pour chaque entier i à partir de 2, « effacer » dans **`t`** tous les multiples de i (par exemple en mettant la cellule **`t[m]`** à 0 si `m` est un multiple de i). A la fin du processus, les entiers non nuls restants dans **`t`** sont premiers.

0	1	2	3	0	5	0	7	0	0	0	11	0	13
0	1	2	3	4	5	6	7	8	9	10	11	12	13

- « tasser » ensuite le contenu du tableau de sorte que les nombres premiers soient regroupés au début du tableau (rappel : 1 ne fait pas partie des nombres premiers)

2	3	5	7	11	13								
0	1	2	3	4	5	6	7	8	9	10	11	12	13

- afficher la liste des nombres premiers.

Quelques remarques pour être efficace :

Remarque 1 : si i a déjà été effacé, il est inutile de chercher à éliminer les multiples de i (c'est déjà fait !)

Remarque 2 : à l'étape i, les multiples de i inférieur à i^2 ont déjà été effacés par les itérations précédentes.

Remarque 3 : les multiples de i peuvent être obtenus successivement par itération avec une simple addition (beaucoup moins coûteuse qu'une multiplication).

3. Calcul : approximer le sinus d'un angle en radian

La fonction sinus peut être approximée à partir de son développement limité ci-dessous en négligeant le terme $o(x^{2n+2})$:

$$\sin(x) = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1} + o(x^{2n+2})$$

Ecrivez le programme qui :

- lit un angle en radian
- calcule le développement limité à l'ordre 6 (on négligera le terme en *o*)
- affiche le résultat

Avant d'écrire l'algorithme, vous répondrez aux questions suivantes :

- pour i=0 , quelles sont les valeurs des 3 termes $(-1)^i$, $(2i+1)!$ et x^{2i+1} ?
- si les valeurs de ces 3 termes sont connues à l'étape i et stockées respectivement dans Si, Qi et Pi, comment ces variables évoluent pour l'itération suivante ?

Testez votre algorithme en générant tous les angles de 0 à 360° par pas de 10° (attention à convertir en radian) et en comparant vos résultats à celui donné par la fonction `sin` de `<math.h>` (pensez à ajouter l'option –lm à la compilation : `gcc tp.c -o tp -lm`)
A partir de quelle valeur ces 2 fonctions divergent significativement ?
Sans le programmer, mais seulement en le précisant dans un commentaire en tête de votre programme, que proposeriez-vous pour avoir une meilleure approximation du calcul du sinus quelque soit l'angle dans l'intervalle [0, 2π] et sans en augmenter l'ordre ?

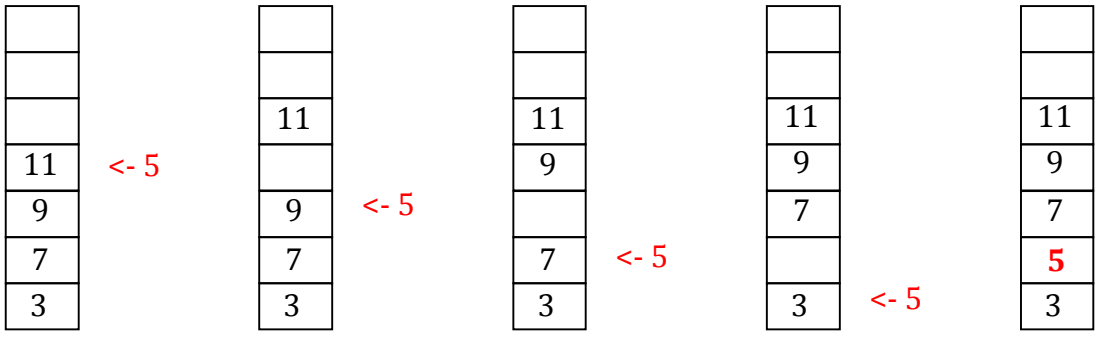
4. Tri par insertion

On considère un tableau **`t`** de taille **N** qui permettra de stocker une suite d'entiers dans l'ordre croissant. La particularité de ce tri est de permettre une arrivée des données échelonnée dans le temps : initialement le tableau est vide, et on y insère régulièrement des valeurs tout en conservant sa relation d'ordre .
Le principe de tri est le suivant :

- on considère que **n** éléments sont déjà stockés dans l'ordre croissant
- trouver le rang d'insertion d'une nouvelle valeur **x** lue au clavier consiste à comparer successivement x au $n^{ème}$ élément, $(n-1)^{ème}$ élément, … jusqu'à vérifier l'inégalité $x \geq i^{ème}$ élément
- l'insertion doit se faire au rang i+1 après avoir décalé d'une case les éléments de rang > i (ce décalage sera effectué au fur et à mesure des comparaisons)
- on dispose alors de n+1 éléments triés (*`n <- n+1`*)

Remarque : la comparaison ne doit en aucun cas dépasser le bas du tableau (par exemple, lors de la comparaison d'un élément plus petit que tous ceux déjà stockés). Pour simplifier le test d'arrêt, on pourra utiliser la technique de la *sentinelle* qui consiste à stocker dans la case **`t[0]`** une valeur fictive « butoir » qui sera toujours plus petite que les valeurs à insérer (par exemple pour un tableau d'`int`, la constante `INT_MIN` définie dans la bibliothèque `<limits.h>` et qui est le plus petit entier de type `int` pour la configuration de votre ordinateur)

Processus d'insertion de la valeur 5



Comparaisons et glissement d'une case d'insertion *Insertion*

5. Tri et indirection

On suppose qu'il y a n élèves inscrits et qu'ils auront deux notes chacun : une en math et une en français. Chaque élève est enregistré à la scolarité sous un numéro unique (de 0 à n-1) qui l'identifie.
En utilisant les structures de données suivantes :

```
#define MATH 0       // n° de colonne des maths dans note
#define FRAN 1       // n° de colonne du français dans note
int note[100][2] ;   // le n° d'un élève est son indice dans note
int n=0 ;            // nombre d'élèves enregistrés
int clMath[100], clFran[100] ; /*recevront les n° d'élèves dans
                              l'ordre du classement de chaque matière */
```

Ecrire le programme qui propose par menu :

- ajouter un élève dans le tableau notes en saisissant ses notes
- construire la table de classement des élèves en mathématiques
- construire la table de classement des élèves en français
- afficher les deux classements avec les numéros des élèves et leurs notes. Exemple :

```
Classement  en français          en math.
1er         élève n° 3 (18/20)   élève n°5(17/20)
2eme        élève n°12 (15/20)   élève n°3(16/20)
```

etc.

Vous utiliserez au choix un tri par insertion ou par permutation.