TD n° 3 : Pointeurs, tableaux et tris élémentaires

Exercice 1 : Pointeurs

Soit le programme C suivant :

```
double deux_fois(const double * i) {return 2*(*i);}

double * allouer_en_initialisant(double val) {
   double *p = (double*) malloc(sizeof(double));
   *p = val; return p;
}

void liberer_en_mettant_a_zero(double ** pa) {free(*pa); *pa = 0x0;}

int main()
{
   double e = 3.1;
   double k = deux_fois(&e);

   double * p1 = allouer_en_initialisant(-8.0);

   double tab[4];
   double * p2;
   for(p2=tab; p2 < tab+4; p2++) { *p2 = (*p1) + k; }
   p2 = &e;
   liberer_en_mettant_a_zero(&p1);

   double **pp2 = &p2;
   return 0;
}
```

a) Pour chaque apparition du caractère '*', indiquez s'il s'agit d'une définition de variable de type pointeur, de l'opérateur de déréférencement d'une adresse, ou bien de l'opérateur de multiplication.

b) Faîtes la trace mémoire de ce programme, en supposant que la valeur de retour du main est située à l'adresse 3 678 556 960.

c) Indiquez pour chaque paramètre s'il est passé en mode donnée ou en mode donnée-résultat.

Exercice 2 : Cryptage d'une chaîne de caractères

Considérons un algorithme de cryptage de chaînes de caractères qui consiste à « additionner » les caractères du texte à crypter avec ceux d'une clé de chiffrement. Par exemple, le cryptage de la chaîne « Cherchez au pied de l'arbre » avec la clé « indice » peut s'illustrer ainsi :

- on place la clé en regard du texte à crypter, en répétant la clé autant de fois que nécessaire pour couvrir le texte, et en ignorant les caractères qui ne sont pas des lettres (ils seront laissés inchangés par l'algorithme de cryptage) :

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
i	n	d	i	c	e	i	n		d	i		c	e	i	n		d	i		c	'	e	i	n	d	i

- on remplace chaque lettre de la clé par sa position dans l'alphabet (0 pour 'a', 1 pour 'b'…),

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
8	13	3	8	2	4	8	13		3	8		2	4	8	13		3	8		2	'	4	8	13	3	8

- on remplace chaque lettre du texte à crypter par la lettre située *d* positions plus loin dans l'alphabet, où *d* est le nombre indiqué par la clé (si on dépasse z, on reboucle sur a, b etc) :

C	h	e	r	c	h	e	z		a	u		p	i	e	d		d	e		l	'	a	r	b	r	e
K	u	h	z	e	l	m	m		d	c		r	m	m	q		g	m		n	'	e	z	o	u	m

a) Si *x* est le code ASCII d'une lettre à crypter et *y* le code ASCII de la lettre de la clé située en regard, quelle formule permet de calculer le code ASCII résultant ? On supposera pour cette question que les deux lettres sont des minuscules. Indice : que vaut *y* – 'a' ?

b) Ecrire en langage algorithmique la procédure de cryptage, dont voici l'entête :

> **Procédure** crypter (texte : tableau de caractères, cle : tableau de caractères, result : tableau de caractères)
> **Préconditions** : texte contient un ou plusieurs caractères suivis d'un '\0'. cle contient une ou plusieurs lettres minuscules suivies d'un '\0'.
> **Postcondition** : result contient la version cryptée de texte jusqu'au '\0' exclu, puis un '\0'. Seules les lettres (majuscules ou minuscules) non accentuées sont cryptées, les autres caractères sont recopiés tels quels. Les éventuels caractères situés derrière le '\0' sont ignorés.
> **Paramètres en mode donnée** : texte, cle
> **Paramètre en mode résultat** : result

c) Quel serait le prototype de cette procédure en C ?

Exercice 3 : Tri par insertion

Le tri par insertion est l'algorithme utilisé par la plupart des joueurs lorsqu'ils trient leur « main » de cartes à jouer. Le principe consiste à prendre le premier élément du sous-tableau non trié et à l'insérer à sa place dans la partie triée du tableau.

a) Dérouler le tri par insertion du tableau {5.1, 2.4, 4.9, 6.8, 1.1, 3.0}.

b) Ecrire le corps de la procédure de tri par insertion, par ordre croissant, d'un tableau de réels :

> **Procédure** tri_par_insertion (tab : tableau [1..n] de réels)
> **Précondition** : tab[1], tab[2], … tab[n] initialisés
> **Postcondition** : tab[1] ≤ tab[2] ≤ … ≤ tab[n]
> **Paramètres en mode donnée** : aucun
> **Paramètre en mode donnée-résultat** : tab

c) Donner l'invariant de boucle correspondant à cet algorithme, en démontrant qu'il vérifie bien les 3 propriétés d'un invariant de boucle : initialisation, conservation, terminaison.

d) **Question à faire chez soi, pour le prochain TD** : Evaluer le nombre de comparaisons de réels et le nombre d'affectations de réels pour un tableau de taille *n*, dans le cas le plus défavorable (tableau trié dans l'ordre décroissant). Cet algorithme est-il meilleur que le tri par sélection (ou tri du minimum), vu en cours magistral ?