TD n° 2 : Vie et mort des variables en mémoire

Exercice 1 : Appels récursifs

Donnez la trace mémoire et la trace écran du programme suivant. Pour la trace mémoire, utilisez le modèle théorique de pile vu en cours. Vous supposerez que la valeur de retour du programme principal est stockée à l'adresse 3 965 281 684 et qu'un entier occupe 4 octets.

```
Procédure mystere (i : entier)
Précondition : i initialisé avec valeur valide
Postcondition : à votre avis ?
Paramètres en mode donnée : i
Paramètres en mode donnée-résultat : aucun
Variables locales : aucune
Début
  Si (i < 10) Alors
      afficher(i)
  Sinon
      afficher(i mod 10)
      i ← i / 10
      mystere(i)
      afficher("ici, i vaut : ", i)
  FinSi
Fin mystere
```

```
{Programme principal}
Début
    v : entier
    v ← 972
    mystere(v)
Fin
```

Exercice 2 : Premier contact avec les pointeurs

Soit le programme C suivant :

```
void proc(int * ad, int j)
/* Précondition : ad adresse valide non nulle
   Postcondition : à votre avis? */
{
       *ad = (*ad + j)*3;
       /* dessiner l'etat de la memoire */
}

int main()
{
       double a = 0.001, b = 0.003;
       double c, *pa, *pb;
       int i=0;
       /* dessiner l'etat de la memoire */
       pa = &a; *pa *= 2;
       pb = &b;
       c = 3*(*pb - *pa);
       /* dessiner l'etat de la memoire */
       proc(&i,4);
       /* dessiner l'etat de la memoire */
       return 0;
}
```

a) En utilisant le modèle de pile vu en cours, dessinez l'état de la mémoire à chaque fois que c'est indiqué en commentaire. Vous supposerez que les variables sont stockées à partir de l'adresse 0xbffff3a0 (note : bffff3a0 en base 16 = 3 221 222 304 en base 10). Veillez à faire des dessins distincts mais cohérents entre eux.

b) Qu'en déduisez-vous au sujet du mode de passage du premier paramètre de proc ?

Exercice 3 : Pointeurs et allocation dynamique de mémoire en C

Considérons le programme C suivant :

```
#include <stdio.h>   /* entrées-sorties avec printf et scanf  */
#include <stdlib.h>  /* malloc, free, exit */

int main()
{
 int e = 10;
 double r = 3.14;
 int * p1;
 double * p2;
 int i;

 p1 = (int*) malloc (5*sizeof(int));
 if(p1 == NULL) { printf("Allocation ratee \n"); exit(1); }

 p2 = (double*) malloc (sizeof(double));
 if(p2 == NULL) { printf("Allocation ratee \n"); exit(1); }

 for(i=0 ; i < 5 ; i++) {p1[i] = e-i;}
 *p2 = r;

 (*p1)*=5;  *p2=p1[3]*2.0;

 e = 25;    r = -8.3;

 free(p1); free(p2);
 return 0;
}
```

a) Expliquez ce que signifie l'instruction `p1 = (int*) malloc (5*sizeof(int));`

b) Proposez une autre façon d'écrire l'instruction `(*p1)*=5;`

c) Faites la trace mémoire de ce programme, en supposant que la valeur de retour du main est stockée à l'adresse 0xD4986C36.

A faire chez soi pour le prochain TD :

Exercice 4 : Pointeurs et tableaux en C, arithmétique des pointeurs

L'exercice suivant nécessite d'avoir vu l'arithmétique des pointeurs (chapitre 4 du cours).

```
int monTab[] = {-25, -6, 8, 15, 38, 50, 72, 81, 98};
int * p = monTab;
```

Quelles valeurs ou adresses fournissent les expressions suivantes ?

a) `*p+2`
b) `*(p+2)`
c) `p+2`
d) `&p`
e) `&monTab[4]-3`
f) `monTab+3`
g) `&monTab[7]-p`
h) `*(p+8)-monTab[7]`