TD2 : Vie et mort des variables en mémoire (2/2)

Exercice 1 : Pointeurs et tableaux en C++, arithmétique des pointeurs

```
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]

a) monTab[0] + 2, soit la valeur -23
b) adresse de monTab[2]
c) monTab[2], soit la valeur 8 (monTab[i] ⇔ *(monTab + i))
d) adresse de la variable p
e) adresse de monTab[1]
f) adresse de monTab[3]
g) valeur (indice) 7, &monTab[7]-&monTab[0]
h) valeur 17 (98-81)

Exercice 2 : Pointeurs et allocation dynamique de mémoire

Soit le programme C++ suivant :

```
#include <iostream>  /* entrées-sorties avec cin et cout */
using namespace std;

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

  p1 = new int [5];
  if (p1 == NULL) { cout << "Allocation ratee \n"; exit(1); }

  p2 = new double;
  if (p2 == NULL) { cout << "Allocation ratee \n"; exit(1); }
```

```
  for (i=0; i < 5; i++) { p1[i] = e-i; }
  *p2 = r;
  /* Faire la trace mémoire (1) */
  (*p1)*=5;
  *p2=p1[3]*2.0;
  /* Faire la trace mémoire (2) */
  e = 25;
  r = -8.3;
  /* Faire la trace mémoire (3) */
  delete [] p1;
  delete p2;
  /* Faire la trace mémoire (4) */
  return 0;
}
```

a. Expliquez ce que signifie l'instruction `p1 = new int [5];`

En supposant qu'un int est stocké sur 4 octets, new réserve 5*4 = 20 octets contigus dans le segment « Tas » de l'espace d'adressage du programme, c'est-à-dire dans un segment mémoire qui n'est pas automatiquement désalloué lorsqu'on sort de la fonction ou de la procédure. Cela permet de stocker un tableau de 5 entiers. new renvoie l'adresse du premier octet réservé, qui est aussi l'adresse de l'élément 0 du tableau. Cette adresse est stockée dans la variable p1 qui est de type pointeur sur int.

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

`p1[0] = p1[0] * 5;  , ou : p1[0] *= 5;`

c. Faites les quatre traces mémoire de ce programme, en supposant que la valeur de retour du main est stockée à l'adresse 3 987 546 988 et qu'il n'y a pas de problème d'allocation mémoire.

Pour rappel, globalement le tas augmente vers le haut (adresses de plus en plus grandes), mais les blocs réservés par des appels différents à un new ne produisent pas forcément des blocs contigus en mémoire. Les données d'un bloc (par exemple les 5 entiers ici) sont elles forcément contigues.

Trace #1

PILE

VR main		3 987 546 988
e	10	3 987 546 984
r	3.14	3 987 546 976
p1	11 456	3 987 546 968
p2	11 528	3 987 546 960
i	5	3 987 546 956

TAS

3.14	11 528
6	11 472
7	11 468
8	11 464
9	11 460
10	11 456

Trace #2

	PILE	
VR main		3 987 546 988
e	10	3 987 546 984
r	3.14	3 987 546 976
p1	11 456	3 987 546 968
p2	11 528	3 987 546 960
i	5	3 987 546 956

TAS	
14.0	11 528
6	11 472
7	11 468
8	11 464
9	11 460
50	11 456

Trace #3

	PILE	
VR main		3 987 546 988
e	25	3 987 546 984
r	-8.3	3 987 546 976
p1	11 456	3 987 546 968
p2	11 528	3 987 546 960
i	5	3 987 546 956

TAS	
14.0	11 528
6	11 472
7	11 468
8	11 464
9	11 460
50	11 456

Trace #4

	PILE	
VR main		3 987 546 988
e	25	3 987 546 984
r	-8.3	3 987 546 976
p1	11 546	3 987 546 968

p2	11 528	3 987 546 960
i	5	3 987 546 956

Exercice 3 : Appel à une fonction qui retourne un tableau

Dessinez l'état de la pile et du tas aux endroits indiqués en commentaires. Vous utiliserez le modèle théorique de pile vu en cours et au TD précédent. Vous supposerez que la valeur de retour du main est stockée à l'adresse 3 987 546 988 et qu'il n'y a pas de problème d'allocation dynamique de mémoire.

```
/* Précondition: tab1 et tab2 sont de même taille */
/* Postcondition: de la mémoire est allouée dans le tas, charge à l'utilisateur de la
                  libérer quand il n'en a plus besoin */
float * sommeTab(const float * tab1, const float * tab2, const int taille) {
  int i;
  float * resultat = new float [taille];
  for (i=0; i < taille; i++) resultat[i] = tab1[i] + tab2[i];
  return resultat;
  /* Faire la trace mémoire (1) */
}

int main() {
  float notes1[] = {8.5, 12.6, 14.5, 10.0, 9.1};
  float notes2[] = {12.2, 5.8, 17.3, 11.7, 7.6};
  float * somme = NULL;
  somme = sommeTab(notes1,notes2,5);
  /* Faire la trace mémoire (2) */
  delete [] somme;
  return 0;
}
```

Trace mémoire #1 (avant le retour de sommeTab)

PILE

VR main		3 987 546 988
notes1[4]	9.1	3 987 546 984
notes1[3]	10.0	3 987 546 980
notes1[2]	14.5	3 987 546 976
notes1[1]	12.6	3 987 546 972
notes1[0]	8.5	3 987 546 968
notes2[4]	7.6	3 987 546 964
notes2[3]	11.7	3 987 546 960
notes2[2]	17.3	3 987 546 956
notes2[1]	5.8	3 987 546 952
notes2[0]	12.2	3 987 546 948
somme	NULL	3 987 546 940
	appel à sommeTab	
VR	1 020	3 987 546 932

TAS

16.7	1 036
21.7	1 032
31.8	1 028
18.4	1 024
20.7	1 020

tab1	3 987 546 968	3 987 546 924
tab2	3 987 546 948	3 987 546 916
taille	5	3 987 546 912
i	5	3 987 546 908
resultat	1 020	3 987 546 900

Trace mémoire #2 (après le retour de sommeTab)

PILE

VR main		3 987 546 988
notes1[4]	9.1	3 987 546 984
notes1[3]	10.0	3 987 546 980
notes1[2]	14.5	3 987 546 976
notes1[1]	12.6	3 987 546 972
notes1[0]	8.5	3 987 546 968
notes2[4]	7.6	3 987 546 964
notes2[3]	11.7	3 987 546 960
notes2[2]	17.3	3 987 546 956
notes2[1]	5.8	3 987 546 952
notes2[0]	12.2	3 987 546 948
somme	1 020	3 987 546 940

TAS

16.7	1 036
21.7	1 032
31.8	1 028
18.4	1 024
20.7	1 020

Pour information, l'instruction suivante (delete) supprime le tableau somme du tas.
Et à la sortie du main, le reste de la pile est libérée.