

Année universitaire : 2016 / 2017

LIFAP3 : Algorithmique et programmation avancée

Contrôle mi-parcours
24 octobre 2016
Durée : 1h30

Note :

/ 20

coller ici

Nom :
Prénom :
N° d'étudiant :
Signature :

coller ici

Documents et téléphones portables interdits. Le barème est donné à titre indicatif. Répondez dans les emplacements prévus à cet effet. Travaillez au brouillon d'abord de sorte à rendre une copie propre – nous ne pouvons pas vous garantir une copie supplémentaire. **Il se tenu compte de la présentation et de la clarté de vos réponses.**

Exercice 1 : Trace mémoire (5 points)

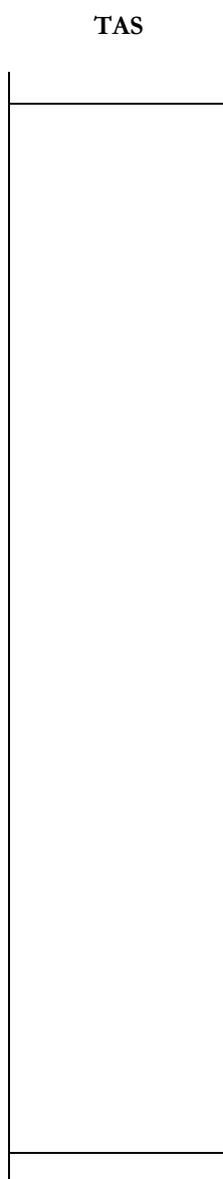
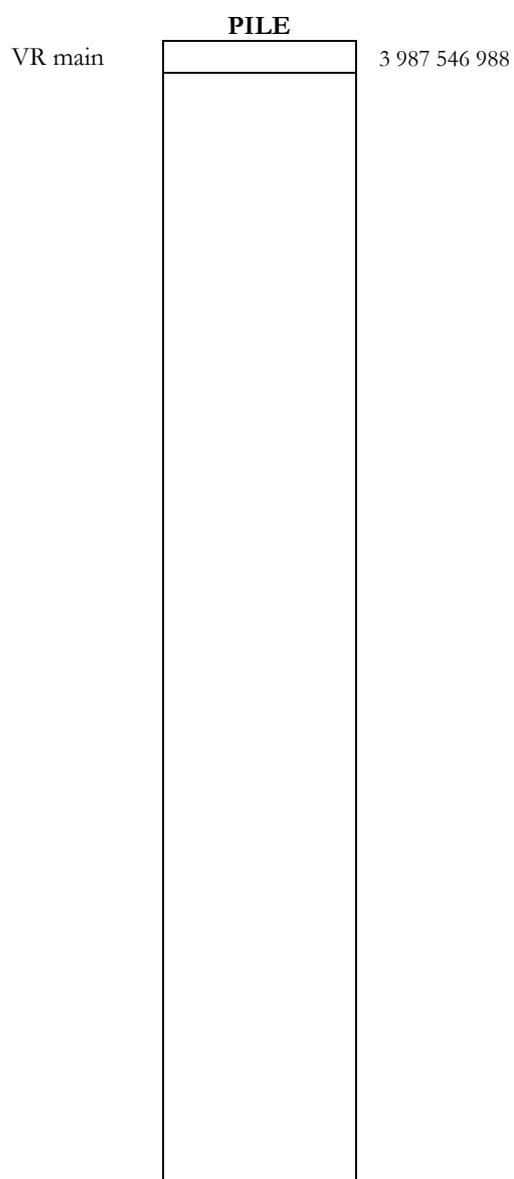
Soit le programme C++ suivant :

```
1  #include <iostream>
2  using namespace std;
3
4  unsigned int safeModulo (unsigned int a, unsigned int b) {
5      if (b != 0) return a % b;
6      else return 0;
7  }
8
9  unsigned int * estTabPair (const unsigned int * t, unsigned int a) {
10     unsigned int * res = new unsigned int [a];
11     /* Dessiner l'état de la mémoire */
12     for (unsigned int i=0; i<a; i++)
13         res[i] = safeModulo(t[i],2);
14     return res;
15 }
16
17 int main () {
18     unsigned int tab [5] = {5,6,2,10,7};
19     unsigned int * resultat;
20     resultat = estTabPair(tab,5);
21     /* Dessiner l'état de la mémoire */
22     for (unsigned int i=0; i<5; i++)
23         cout << "(" << tab[i] << "," << resultat[i] << ") ";
24     delete [] resultat;
25     return 0;
26 }
```

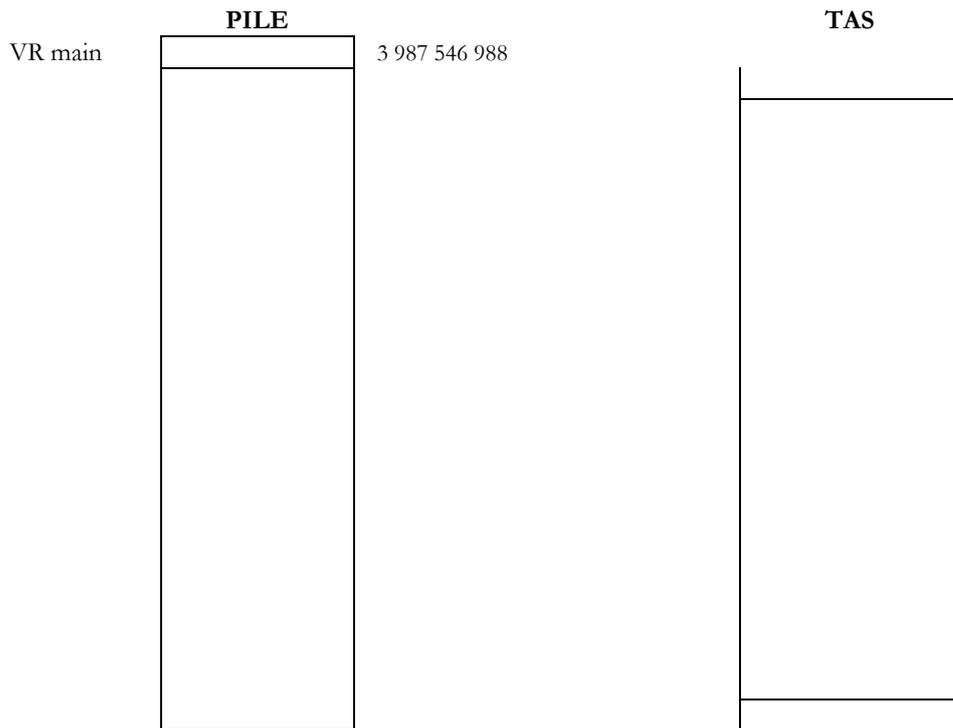
Question 1.1 : Dessinez l'état de la pile et du tas aux deux endroits indiqués en commentaire (lignes 11 et 21). On vous demande pour chaque dessin une « photo » de l'état de la mémoire. Vous utiliserez le modèle théorique de pile vu en cours et en TD. Vous supposerez que la valeur de retour du programme principal est stockée à l'adresse 3 987 546 988, et que les entiers (signés ou non) sont codés sur 4 octets.



DESSIN 1



DESSIN 2



Question 1.2 : Donner la trace écran de l'exécution du programme principal.

Exercice 2 : Invariant de boucle (5 points)

Soit la fonction `pgcd` suivante qui retourne le plus grand diviseur commun de deux entiers strictement positifs :

```
1 | unsigned int pgcd (unsigned int a, unsigned int b) {  
2 |     while (a != b) {  
3 |         if (a > b) a = a - b;  
4 |         else b = b - a;  
5 |     }  
6 |     return a;  
7 | }
```

Question 2.1 : Proposer un invariant de boucle (`while` de la ligne 2) de cette fonction calculant de PGCD.
Indication : l'invariant décrira l'évolution du PGCD de `a` et `b` entre deux itérations.

Question 2.2 : Vérifier la propriété d'initialisation et de terminaison de cet invariant. On ne demande pas de vérifier la propriété de conservation.

Exercice 3 : Calcul de complexité (5 points)

Considérons l'algorithme suivant :

```
Variables locales  
tab : tableau[n][m] de réels  
i, j, k, compteur : entiers  
Début  
1   compteur ← 0  
2   Pour i allant de 1 à n par pas de 1 faire  
3     Pour j allant de 1 à m par pas de 1 faire  
4       tab[i][j] ← 0  
5       Pour k allant de i à j par pas de 1 faire  
6         tab[i][j] ← tab[i][j] + 1  
7         compteur ← compteur + 1  
8       Fin pour  
9     Fin pour  
10  Fin pour  
Fin
```

Question 3.1 : Combien de fois exécute-t-on le corps de la boucle Pour des lignes 5 à 8 en fonction de i et j ?

Question 3.2 : Complétez la formule mathématique suivante qui compte le nombre de fois où la variable **compteur** est incrémentée. Développez ensuite cette formule jusqu'à l'exprimer en fonction de n et m uniquement.

Rappel : $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

$$\sum_{i=\dots}^{\dots} \sum_{j=\dots}^{\dots} \sum_{k=\dots}^{\dots} 1 =$$

Question 3.3 : Ecrivez le contenu d'un tableau `tab` de taille 3×4 obtenu après l'exécution de cet algorithme.

<code>tab</code>	<code>j = 1</code>	<code>j = 2</code>	<code>j = 3</code>	<code>j = 4</code>
<code>i = 1</code>				
<code>i = 2</code>				
<code>i = 3</code>				

Question 3.4 : Supposons maintenant que $n = m$. Combien de fois aura-t-on incrémenté la variable `compteur` ? En quoi cela peut-il être contre intuitif en regardant la structure générale de l'algorithme ?

Exercice 4 : Modélisation en classe (5 points)

Le but de cet exercice est de modéliser des classes `Point2D` et `Ligne` pouvant servir au calcul géométrique et à l'algèbre linéaire. Nous donnons ci-dessous un exemple de programme principal que l'on souhaite pouvoir exécuter.

```

1  int main () {
2      Point2D A (1.0,2.5);
3      Point2D B (-4.0,1.1);
4      Point2D C (5.0,-3.4);
5      Point2D O (0.0,0.0);
6      Ligne * l = new Ligne (&A,&B);
7      l->tracer();
8      bool CsurL = C.estSur(l);
9      Point2D vecOA = A - O;
10     Point2D vecOB = B - O;
11     if (vecOA == vecOB)
12         vecOB.normaliser();
13     else
14         l->setPoints(A,C);
15     delete l;
16     return 0;
17 }
```

Question 4.1 : Pour chaque ligne du programme principal, remplissez le tableau suivant indiquant les membres qu'il va falloir implémenter dans les classes `Point2D` et `Ligne`.

Ligne	Type de membre (fonction, procédure, opérateur, constructeur etc.)	Nom du membre	Classe (Point2D ou Ligne)
2-5			
6			
7			
8			
9-10			
11			
12			
14			
15			

Question 4.2 : Donner la déclaration minimale, en C++, des classes `Point2D` et `Ligne` pour que le programme ci-dessus puisse s'exécuter (l'implémentation des membres des classes n'est pas demandée).