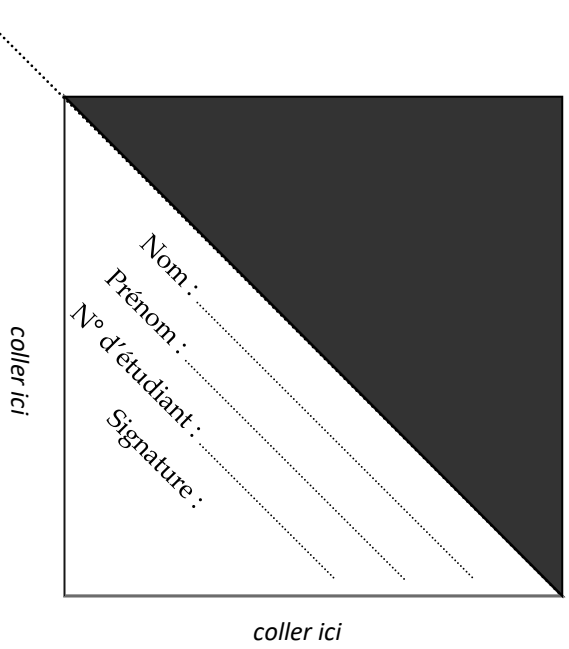
Université Claude Bernard Lyon 1

Année universitaire : 2016 / 2017

LIFAP3 : Algorithmique et programmation avancée

Contrôle mi-parcours
24 octobre 2016
Durée : 1h30

Note : / 20



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)

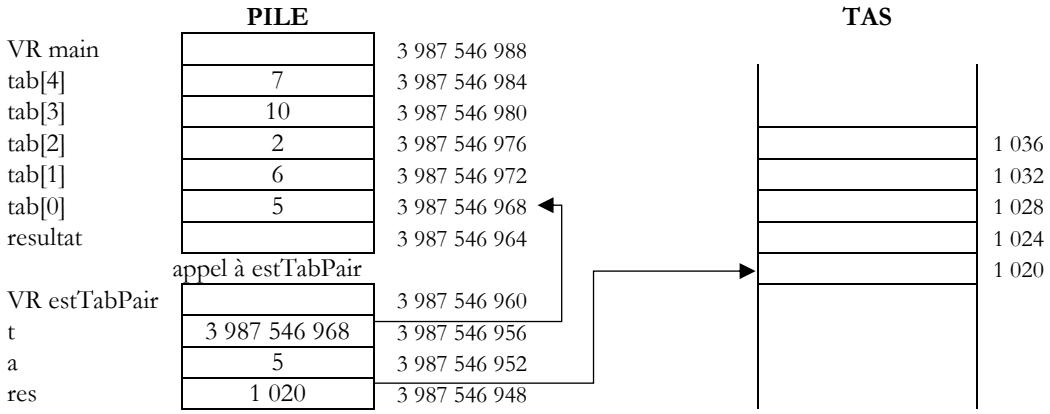
Soit le programme C++ suivant :

```
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 }
```

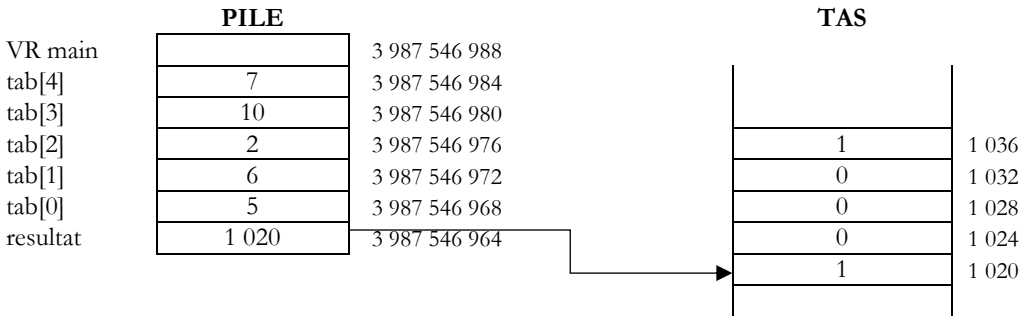
Question 1.1 : Dessinez l'état de la pile et du tas aux deux endroits indiqués en commentaire (lignes 11 et 21). On vous demande pour chaque dessin une « photo » de l'état de la mémoire. Vous utiliserez le modèle théorique de pile vu en cours et en TD. Vous supposerez que la valeur de retour du programme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

[redacted]

DESSIN 1



DESSIN 2



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

```
(5,1) (6,0) (2,0) (10,0) (7,1) (22,0)
```

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 décriera l'évolution du PGCD de a et b entre deux itérations.

Un invariant pour cette fonction est « Pour toute itération k, on a $PGCD(a_k, b_k) = PGCD(a, b)$ ». C'est-à-dire que la valeur du pgcd des deux nombres ne change pas 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.

Initialisation : A la première itération, les valeurs des paramètres sont directement utilisées dans la boucle, donc on a $PGCD(a_0, b_0) = PGCD(a, b)$ et donc la propriété est vraie avant l'exécution de la boucle.

Terminaison : En sortie de boucle, après n itérations, la propriété du `while` est fausse donc on a $a_n = b_n$ et la valeur a_n est retournée. Or lorsque $a_n = b_n$ on a $a_n = PGCD(a_n, b_n)$ et ici on a aussi $PGCD(a_n, b_n) = PGCD(a_k, b_k) = PGCD(a, b)$ (invariant de boucle), donc la fonction retourne bien $PGCD(a, b)$.

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 ?

Le corps de la boucle Pour des lignes 5 à 8 est exécuté pour des valeurs de k allant de i à j compris donc (j-i+1) fois.

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

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

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=i}^{j} 1 = \sum_{i=1}^{n}\sum_{j=1}^{m}(j-i+1) = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} j + \sum_{j=1}^{m} 1 - i\right) = \sum_{i=1}^{n}\left(\frac{m(m+1)}{2} + m - mi\right)$$

$$= n\left(\frac{m(m+1)}{2} + m\right) - m\sum_{i=1}^{n} i = n\left(\frac{m(m+1)}{2} + m\right) - m\frac{n(n+1)}{2} = \frac{nm^2 - mn^2}{2} + nm$$

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

`tab`	j = 1	j = 2	j = 3	j = 4
i = 1	1	2	3	4
i = 2	0	1	2	3
i = 3	0	0	1	2

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 ?

Si $n = m$, alors la formule devient

$$\frac{nm^2 - mn^2}{2} + nm = \frac{nn^2 - nn^2}{2} + nn = n^2$$

La complexité de l'algorithme est quadratique en fonction de la taille n du tableau 2D, alors que l'algorithme comprend trois boucles imbriquées qui pourraient suggérer une complexité en $O(n^3)$.

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	Constructeur	Point2D	Point2D
6	Constructeur	Ligne	Ligne
7	Procédure	tracer	Ligne
8	Fonction	estSur	Point2D
9-10	Opérateur Constructeur (par copie)	- (soustraction) Point2D	Point2D Point2D
11	Opérateur	== (égalité)	Point2D
12	Procédure	normaliser	Point2D
14	Procédure	setPoints	Ligne
15	Destructeur	Ligne	Ligne

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).

```
class Point2D {
public :
   float posX, posY;

   Point2D (float x, float y);
   Point2D (const Point2D & p);
   bool estSur (const Ligne * l) const;
   void normaliser ();

   const Point2D operator - (const Point2D & pt);
   bool operator == (const Point2D & pt);
};



class Ligne {
public :
   Point2D * point1, * point2;

   Ligne (Point2D * p1, Point2D * p2);
   ~Ligne ();

   void tracer () const;
   void setPoints (const Point2D & p1, const point2D & p2);
};
```