Algorithmes et structures de données : TD 6 Corrigé

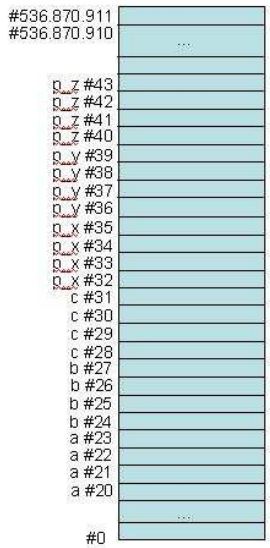
Tableaux statiques et dynamiques - Pointeurs - Complexité asymptotique

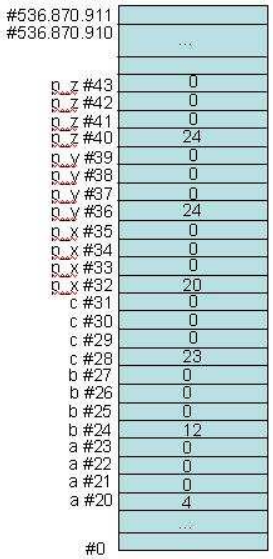
Exercice 6.1 *Pointeurs*

Considérer l'algorithme suivant :

```
var a, b, c : integer;
var p_x, p_y, p_z : ^integer;

{ Endroit 1 }

début

  a := 4;
  b := 12;
  c := 23;

  p_x := Addr(a);
  p_y := Addr(b);
  p_z := p_y;

{ Endroit 2 }

  p_x^ := p_x^ + 2;
  p_y^ := p_y^ + 1;
  c := c + 3;

fin
```

1. Ebaucher l'occupation de la mémoire dans un ordinateur de 1 Mo de mémoire vive à l'endroit 1, puis à l'endroit 2.

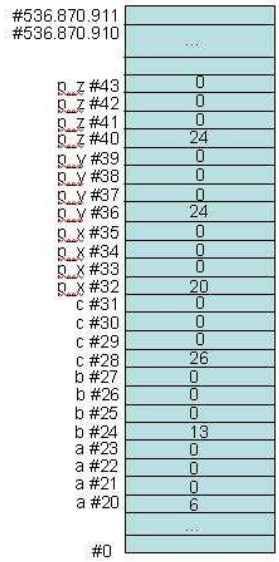




2. Faites tourner cet algorithme dans un tableau (de 6 colonnes bien sur).

a	b	c	p_x	p_y	p_z
4					
	12				
		23			
			20		
				24	
					24
6					
	13				
		26			

3. Ebaucher l'occupation de la mémoire dans un ordinateur de 1 Mo de mémoire vive à l'endroit 3.



Exercice 6.2 *Complexité asymptotique*

1. Considérer les algorithmes suivantes avec un temps d'exécution $T(n)$ pour une longueur de données n. Déterminer leur complexités asymptotiques respectives, et indiquer quel(s) règle(s) vous aviez appliqués.

Algorithme A1 $T(n) = 3n + 2$

Complexité asymptotique $O(n)$

Algorithme A2 $T(n) = 6$

Complexité asymptotique $O(1)$

Algorithme A3 $T(n) = 4n^2 + n + 2$

Complexité asymptotique $O(n^2)$

Algorithme A4

```
Exécuter A1;
Exécuter A2;
Exécuter A3;
```

Règle de somme : Complexité asymptotique $O(n^2)$

Algorithme A5

```
pour i de 1 à n faire
    Exécuter A3;
fin pour
Exécuter A1;
```

Règle de produit pour la boucle et ensuite règle de somme : Complexité asymptotique $O(n^3)$

Algorithme A6

```
pour i de 1 à 5 faire
    Exécuter A1;
fin pour
```

5 fois la règle de somme : Complexité asymptotique $O(n)$

Exercice 6.3 *Appels des fonctions par valeur*

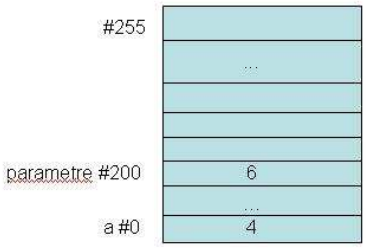
Considérer le programme suivant :

```
var a : byte;

procedure ajouter (parametre : byte)
début
    WriteLn('parametre (avant) ', parametre);
    { Endroit 2 }
    parametre := parametre + 2;
    WriteLn('parametre (après) ', parametre);
    { Endroit 3 }
fin

début
    a := 4;
    { Endroit 1 }
    WriteLn('a (avant) ', a);
    ajouter(a);
    WriteLn('a (après) ', a);
fin
```

1. Qu'est-ce qui est affiché à l'écran ?

```
a (avant) 4
parametre (avant) 4
parametre (après) 6
a (après) 4
```

2. Ebaucher l'occupation de la mémoire dans un ordinateur de 256 Octets de mémoire vive à l'endroit 3 (adressage 32 bits).



Exercice 6.4 *Appels des fonctions par référence*

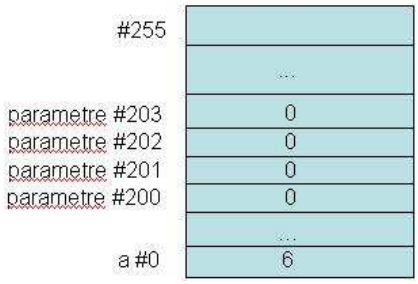
Considérer le programme suivant :

```
var a : byte;
type t_p_a = ^byte;

procedure ajouter (parametre : t_p_a);
begin
    WriteLn('parametre^ (avant)', parametre^);
    { Endroit 2 }
    parametre^ := parametre^ + 2;
    WriteLn('parametre^ (après)', parametre^);
   { Endroit 3 }
end;

début
    a := 4;
    { Endroit 1 }
    WriteLn('a (avant) ', a);
    ajouter(Addr(a));
    WriteLn('a (après) ', a);
fin
```

1. Qu'est-ce qui est affiché à l'écran ?

```
a (avant) 4
parametre (avant) 4
parametre (après) 6
a (après) 6
```

2. Ebaucher l'occupation de la mémoire dans un ordinateur de 256 Octets de mémoire vive à l'endroit 3 (adressage 32 bits).



Exercice 6.5 *Tableaux*

Considérer l'algorithme 1 qui remplit un tableau statique de taille n :

```
var tableau     :       array[0..n-1] of integer;
var i           :       integer;

début
  i:=0;
  tant que i<n-4;
        tableau[i] := i*i;
        i := i + 1;
  fin tant que
fin
```

1. Quelle est le temps d'exécution $T(n)$ de cet algorithme ? Quelle est la complexité asymptotique de cet algorithme (notation Grand-O) ?

1 affectation, est dans la boucle, $n-4$ comparaisons et $2*(n-4)$ affectations, alors :

$$T(n) = 1 + (n-4)*3 = 3n - 11$$

La complexité asymptotique est donc de $O(n)$.

2. Ecrire un algorithme qui insère un élément supplémentaire avec la valeur 1000 **au début (à l'index 0)** du tableau. Quelle est la complexité de votre algorithme ?

REMARQUE : Dans cet algorithme il s'agit de déplacer le contenu des autres cellules ...

```
  i:=n-4;
  tant que i>0 faire
        tableau[i] := tableau[i-1];
        i := i - 1;
  fin tant que
  tableau[0] := 1000;
```

La compexité est de O(n) car il faut déplacer le contenu de tous les cellules (comme on verra plus tard, ceci n'est pas nécessaire avec une liste linéaire simplement chaînée ..).