TD n° 2 (Correction)

Sémantique : environnement, mémoire et pile

Dans les trois exercices ci-dessous, vous devez donner les états successifs de l'**environnement**, la **mémoire**, les **tableaux d'activation** pour les principales étapes du programme et préciser les affichages à l'écran lorsqu'il y en a. On suppose que le segment de pile est constitué des adresses de 0 à 1000 et que le segment de données statiques est constitué des adresses de 1000 à 2000.

Correction : *Il est conseillé, pour chaque exercice, de recopier le code au tableau afin de pouvoir simuler son exécution de facon plus démonstrative.*

Exercice 1.

```
1  #include <iostream>
2  using namespace std;
3  int main () {
4      int n1, n2;
5      cout << "Saisissez n1:" << endl;
6      cin >> n1;      // on suppose que l'on tape 2
7      cout << "Saisissez n2:" << endl;
8      cin >> n2;      // on suppose que l'on tape 3
9      n1 = n2;
10     n2 = n1;
11     cout << "n1 vaut " << n1 << endl;
12     cout << "n2 vaut " << n2 << endl;
13     return 0;
14 }
```

Modifiez ensuite ce code pour que les variables n1 et n2 échangent effectivement leur contenu, puis refaire les tableaux d'activation correspondant à cette nouvelle version du code.

Correction : *Exo 1*

Ne pas oublier de corriger l'exo 6 du TD1 au début

Ce TD passe plus ou moins bien... Pour les motiver, il faut **leur dire AU DEBUT DU TD** *que quand on passera à des programmes compliqués avec des références, ils auront besoin de la pile pour comprendre ce qu'il se passe. Et que comme on ne veut pas tout leur faire faire d'un coup, on commence par la pile sans références, puis dans les TDs suivants, les références sans la pile, puis les deux ensemble.*

On commence par l'état du programme à la compilation. Le tableau d'activation construit pour cette fonction main() prévoit 4 emplacements:

Main
n1
n2
return

Lors de l'exécution du programme, l'espace nécessaire prévu par le tableau d'activation de la fonction est alloué dans la mémoire, aux premières places disponibles de celle-ci :
On initialise ADM (donnée administrative, adresse de retour): ici on retient que à la fin du main on quittera le programme.

pile :

	5	?
	4	?
Main	3	ADM
n1	2	?
n2	1	?
return	0	?

Les effets de ce programme sont les suivants : on affecte à n1 la valeur saisie par l'utilisateur en ligne (6), de même pour n2, ligne (8).
Par exemple, 2 et 3.

pile :

	5	?
	4	?
Main	3	ADM
n1	2	2
n2	1	3
return	0	?

Puis on affecte à n1 la valeur de n2 : *pile :*

	5	?
	4	?
Main	3	ADM
n1	2	3
n2	1	3
return	0	?

Enfin, on affecte à n2 la valeur de n1 : *pile :*

	5	?
	4	?
Main	3	ADM
n1	2	3
n2	1	3
return	0	?

Après les impressions, n1 = 3, n2 = 3, la valeur 0 est affectée au return puis on dépile le tout.

Question Exo 1. Q2
Le code modifié :

```
(1) #include <iostream>
(2) using namespace std;
(3) int main(){
(4bis)  int n1, n2, temp;
(5)     cout << "Saisissez n1:" << endl;
(6)     cin >> n1;
(7)     cout << "Saisissez n2:" << endl;
(8)     cin >> n2;
(8bis)  temp = n1;
(9)     n1=n2;
(10bis) n2=temp;
(11)    cout << "n1 vaut " << n1 << endl;
(12)    cout << "n2 vaut " << n2 << endl;
(13)    return 0;
(14) }
```

Le tableau d'activation construit à la compilation pour cette nouvelle fonction main() prévoit 5 emplacements qui sont alloués dans la mémoire:

pile :

	5	?
Main	4	ADM
n1	3	?
n2	2	?
temp	1	?
return	0	?

toujours avec 2 et 3 comme valeurs saisies par l'utilisateur.

pile :

	5	?
Main	4	ADM
n1	3	2
n2	2	3
temp	1	?
return	0	?

Puis on affecte à temp la valeur de n1 :

	5	?
Main	4	ADM
n1	3	2
n2	2	3
temp	1	2
return	0	?

Puis, on affecte à n1 la valeur de n2 :

	5	?
Main	4	ADM
n1	3	3
n2	2	3
temp	1	2
return	0	?

Comme on a sauvegardé la valeur de n1 dans temp on peut l'affecter à n2 :

	5	?
Main	4	ADM
n1	3	3
n2	2	2
temp	1	2
return	0	?

Les impressions obtenues sont : n1 = 3 et n2 = 2

fin de l'exécution :

5	?
4	ADM
3	3
2	2
1	2
0	0

Exercice 2.

```
1 #include <iostream>
2 using namespace std;
3
4 int y;
5
6 int P1() {
7     int x = y * 3;
```

```
 8       return x;
 9   }
10
11   int main() {
12       int x;
13
14       x = 5;
15       y = 2*x;
16       x = P1() + 3;
17       cout << "x = " << x << endl;
18       return 0;
19   }
```

Correction : *A la compilation, on décide de l'adresse des variables globales dans le segment de données statiques. On planifie aussi l'utilisation de la mémoire de chaque fonction : la compilation construit le tableau d'activation de chaque procédure ou fonction.*

Pour la fonction `p1`*:*

P1
x
return

Pour la fonction `main`*:*

Main
x
return

La variable `y` *est une variable globale et sera stockée à l'adresse 1000.*

Puis on démarre l'exécution en commencant par le main :

- *Au début du programme :*
  données statiques :

y	*1000*	*?*

pile :

6	*?*
5	*?*
4	*?*
3	*?*
2	*?*
1	*?*

- *Entrée dans la fonction main:*
  données statiques:

y	*1000*	*?*

pile :

	6	*?*
	5	*?*
	4	*?*
	3	*?*
Main	*2*	*ADM*
x	*1*	*?*
return	*0*	*?*

Note: ADM en case mémoire 2 dénote les informations administatives : ici on retient que à la fin du main on quittera le programme.

- *Ligne 14 :*
  données statiques:

y	*1000*	*?*

pile:

	6	*?*
	5	*?*
	4	*?*
	3	*?*
Main	*2*	*ADM*
x	*1*	*5*
return	*0*	*?*

- *Ligne 15 :*
  données statiques:

y	*1000*	*10*

pile:

	6	*?*
	5	*?*
	4	*?*
	3	*?*
Main	*2*	*ADM*
x	*1*	*5*
return	*0*	*?*

- *Appel à P1 en ligne 16 : l'espace nécessaire prévu par le tableau d'activation de la fonction (son bloc d'activation) est alloué sur la pile.*
  Le bloc du Main n'est plus visible
  données statiques:

y	1000	10

pile:

	6	?
P1	5	ADM
x	4	?
return	3	?
	2	ADM
	1	5
	0	?

Note: ADM en case mémoire 5 retient que à la fin de P1, il faudra revenir ligne 16 du main exécuter l'instruction `x=P1() + 3;` *après avoir remplacé P1() par la valeur retournée.*

- *Dans P1 en ligne 8 :*
  données statiques:

y	1000	10

pile:

	6	?
P1	5	ADM
x	4	30
return	3	30
	2	ADM
	1	5
	0	?

- *Ligne 17 on affiche* `x = 33`

- *Ligne 18*
  données statiques:

y	1000	10

pile:

	6	?
	5	ADM
	4	30
	3	30
Main	2	ADM
x	1	33
return	0	0

- *Dans P1 en ligne 7 :*
  données statiques:

y	1000	10

pile:

	6	?
P1	5	ADM
x	4	30
return	3	?
	2	ADM
	1	5
	0	?

- *Return au main ligne 16 où l'on a remplacé l'appel* `P1()` *par la valeur retournée. On exécute donc* `x = 30 + 3;` *:*
  données statiques:

y	1000	10

pile:

	6	?
	5	ADM
	4	30
	3	30
Main	2	ADM
x	1	33
return	0	?

- *Retour du main et fin du programme*
  données statiques:

y	1000	10

pile:

6	?
5	ADM
4	30
3	30
2	ADM
1	33
0	0

Exercice 3.

```
1  #include <iostream>
2  using namespace std;
3  int y = 10;
4  int P1() {
5    int x;
6    x = 7;
7    return x + 1;
8  }
9  void P2() {
10   int x, y;
11   x = 3;
12   y = P1();
13   x = y;
14 }
15 void main() {
16   int x;
17   x = 1;
18   y = P1() + 4;
19   P2();
20   cout << "x vaut " << x << endl;
21 }
```

Correction :

On commence par l'état du programme à la compilation. La compilation construit le tableau d'activation de chaque procédure ou fonction.

Pour la fonction P1:

P1
x
return

Pour la procédure P2:

P2
x
y

Pour la procédure main:

main
x

La variable y est une variable globale et sera stockée à l'adresse 1000.

Puis on démarre l'exécution en commencant par le main :

- *Au début du programme :*
  données statiques :

y	*1000*	*10*

  pile :

5	*?*
4	*?*
3	*?*
2	*?*
1	*?*
0	*?*

- *Entrée dans la fonction main:*
  données statiques:

y	*1000*	*10*

  pile :

5	*?*	
4	*?*	
3	*?*	
2	*?*	
Main	*1*	*ADM*
x	*0*	*?*

- *Ligne 17 :*
  données statiques:

y	*1000*	*10*

pile:

	5	?
	4	?
	3	?
	2	?
Main	1	ADM
x	0	1

Appel à la fonction P1 :

l'espace nécessaire prévu
par le tableau d'activation de la fonction
(son bloc d'activation)
est alloué sur la pile.

Le bloc du Main n'est plus visible

- *Ligne 6, Dans P1:*
  données statiques:

y	1000	10

pile :

	5	?
P1	4	ADM
x	3	7
return	2	8
	1	ADM
	0	1

- *sortie de P1, retour au main :*
  données statiques:

y	1000	12

pile:

	5	?
	4	ADM
	3	7
	2	8
Main	1	ADM
x	0	1

- *Ligne 19, appel de P2,*
  on alloue le bloc de P2 sur le sommet de Pile
  ATTENTION :
  tant que les variables n'ont pas été initialisées
  elles ont les valeurs qui se trouvaient là !

  Le y global n'est plus visible
  car masqué par le y de P2

- *Ligne 18, appel de P1:*
  données statiques:

y	1000	10

pile :

	5	?
P1	4	ADM
x	3	?
return	2	?
	1	ADM
	0	1

On affecte à x la valeur 7,
puis on calcule la valeur de retour de P1, 8

le bloc de P1 est désalloué
le bloc visible est maintenant le main
mais les valeurs restent dans les cases
On affecte à y, la variable globale,
la valeur de retour de P1, 8, + 4 donc 12

données statiques:

	1000	12

pile :

	5	?
P2	4	ADM
x	3	7
y	2	8
	1	ADM
	0	1

- *Ligne 11, puis 12, dans P2,*
  données statiques:

y	*1000*	*12*

pile :

P1	*7*	*ADM*
x	*6*	*?*
return	*5*	*?*
	4	*ADM*
	3	*3*
	2	*8*
	1	*ADM*
	0	*1*

Après l'affectation de x par 3,
on passe à l'appel de P1
et on alloue le bloc de P1 sur le sommet de Pile

Le bloc de P2 n'est plus visible

Le y global redevient visible

données statiques:

y	*1000*	*12*

pile :

P1	*7*	*ADM*
x	*6*	*7*
return	*5*	*8*
	4	*ADM*
	3	*3*
	2	*8*
	1	*ADM*
	0	*1*

- *Ligne 6, dans P1:*

  On affecte à x la valeur 7,
  puis on calcule la valeur de retour de P1, 8

données statiques:

1000	*12*

pile :

	7	*ADM*
	6	*7*
	5	*8*
P2	*4*	*ADM*
x	*3*	*8*
y	*2*	*8*
	1	*ADM*
	0	*1*

- *Sortie de P1, retour à P2, en ligne 12, :*
  le y de P2 se voit affecté la valeur de retour de P1, toujours 8
  puis x recoit y, donc 8 aussi

données statiques:

y	*1000*	*12*

pile :

	7	*ADM*
	6	*7*
	5	*8*
	4	*ADM*
	3	*8*
	2	*8*
Main	*1*	*ADM*
x	*0*	*1*

- *Sortie de P2, retour au Main, en ligne 20, :*

  Le x qui est affiché est celui du Main qui vaut 1

  puis on sort,
  avec le sommet de pile qui revient sous 0.

S'il vous reste du temps, et que vous avez fini tous les exos du TD1,

Pour faire la transition avec les exos de la semaine suivante, faites les réfléchir à une fonction qui permettrait de simplifier une fraction. Quelles sont les données (numérateur, dénumérateur), les résultats, (nNew, dNew), combien de résultats (2) donc pas une fonction qui ne renvoie qu'un résultat...

```
1 #include <iostream>
2 using namespace std ;
3
4 int pgcd(int a, int b){
```

```
5     int reste;
6     do {
7           reste = a%b;
8           a = b;
9           b = reste;
10    } while (reste != 0);
11    return a;
12 }
13
14 void simplifier( int n, int d, int &nNew, int &dNew) {
15    int div = 1;
16    if (n > d) div = pgcd(n,d); else div = pgcd(d,n);
17    nNew = n / div ; dNew = d / div ;
18 }
19
20 int main(){
21   int a,b, c, d;
22   cout << " numerateur et denominateur ?" ;
23   cin >> a >> b ;
24   simplifier( a,b,c,d);
25   cout << c << " " << d << endl;
26   return 0;
27 }
```