TD n° 4 (Correction)

Sémantique des passages de paramètres

Exercice 1.

```
void ajouter (int a, int b, int c) {
  c = a + b;
}
int main () {
  int x, y, z;
  x = 2;
  y = 8;
  z = 7;
  cout << "avant, x =" << x << "y =" << y << "z =" << z << endl;
  ajouter(x, y, z);
  cout << "apres, x =" << x << "y =" << y << "z =" << z << endl;
  return 0;
}
```

1. Donnez les **tableaux d'activation** des procédure et fonction ci-dessus puis les états successifs de la **mémoire**, pour les principales étapes du programme. Précisez les affichages à l'écran lorsqu'il y en a.
2. Que remarquez vous ? Comment s'appelle le passage du paramètre c ?
3. Modifiez la procédure `ajouter` pour que z soit modifi à la sortie de la procédure. Comment le paramètre c est-il passé ?

Correction : *Tableaux d'activation :*

Pour la procédure ajouter :

ajouter
a
b
c

Pour le main:

main
x
y
z
return

Entrée dans le main

		6	*?*
		5	*?*
pile :	*main*	*4*	*ADM*
	x	*3*	*?*
	y	*2*	*?*
	z	*1*	*?*
	return	*0*	*?*

Affectation des variables

		6	*?*
		5	*?*
pile :	*main*	*4*	*ADM*
	x	*3*	*2*
	y	*2*	*8*
	z	*1*	*7*
	return	*0*	*?*

Affichage: avant, x = 2, y = 8, z = 7

Appel à ajouter, on empile :

pile :

	9	?
ajouter	8	ADM
a	7	2
b	6	8
c	5	7
	4	ADM
	3	2
	2	8
	1	7
	0	?

on évalue c dans ajouter

pile :

	9	?
ajouter	8	ADM
a	7	2
b	6	8
c	5	10
	4	ADM
	3	2
	2	8
	1	7
	0	?

A la fin d'ajouter, on sort, on dépile :

pile :

	9	?
	8	ADM
	7	2
	6	8
	5	10
main	4	ADM
x	3	2
y	2	8
z	1	7
return	0	?

Affichage: apres, x = 2, y = 8, z = 7
C'était un passage par valeur, donc z n'a pas changé !

Maintenant, avec un passsage où c est une donnée-résultat
L'en-tte de ajouter devient
`void ajouter (int a, int b, int &c)`
Dans le nouveau tableau d'activation de `ajouter`, *c s'écrit* &c

Appel à ajouter, on empile :

pile :

	9	?
ajouter	8	ADM
a	7	2
b	6	8
&c	5	•
	4	ADM
	3	2
	2	8
	1	7
	0	?

on évalue c dans ajouter

pile :

	9	?
ajouter	8	ADM
a	7	2
b	6	8
&c	5	•
	4	ADM
	3	2
	2	8
	1	10
	0	?

A la fin d'ajouter, on sort, on dépile :

pile :

	9	*?*
	8	*ADM*
	7	*2*
	6	*8*
	5	
main	*4*	*ADM*
x	*3*	*2*
y	*2*	*8*
z	*1*	*10*
return	*0*	*?*

Affichage: apres, x = 2, y = 8, z = 10

C'était un passage par référence, donc la valeur de z a effectivement changé !

Exercice 2.

1. Réalisez la procédure `retrancher` qui prend en paramètre deux entiers et qui modifie la valeur de son premier paramètre en retranchant de celui-ci la valeur de son deuxième paramètre.
2. Réalisez la fonction `modulo` qui utilise la procédure `retrancher` pour calculer puis renvoyer le reste de la division entière de son premier paramètre par son deuxième.
3. Réaliser le programme principal qui lit deux nombres entiers positifs et calcule le modulo du premier nombre par le deuxime et l'affiche.
4. Donnez les **tableaux d'activation** et les états successifs de la **mémoire** pour les principales étapes du programme. Précisez les affichages à l'écran lorsqu'il y en a. On supposera que les nombres lus sont 13 et 4.

Correction :

```
void retrancher(int &x, int y) {
   x = x - y;
}

int modulo (int a, int b) {
  while (a >= b) {
     retrancher(a,b);
  }
  return a;
}

int main () {
   int u, v, m;
   do {
       cout << "Entrez 2 nbs entiers positifs u, v, tels que u >= v" << endl;
       cin >> u >> v;
   } while ( (v <= 0) or (u < v)) ;
   m = modulo(u,v);
   cout << "le modulo de " << u << "par " <<  v << " est " << m << endl;
}
```

Tableaux d'activation :

retrancher
&x
y

modulo
a
b
return

main
u
v
m

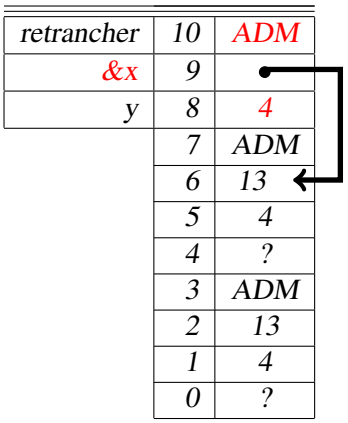
Entrée dans le main et Affectation des variables

pile :

	5	?
	4	?
main	3	ADM
u	2	13
v	1	4
m	0	?

Appel modulo, qui fait un 1er appel à retrancher, on empile :

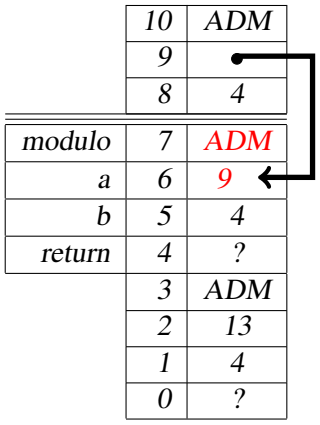
pile :



retrancher	10	ADM
&x	9	
y	8	4
	7	ADM
	6	13
	5	4
	4	?
	3	ADM
	2	13
	1	4
	0	?

calcul, modif puis retour à modulo

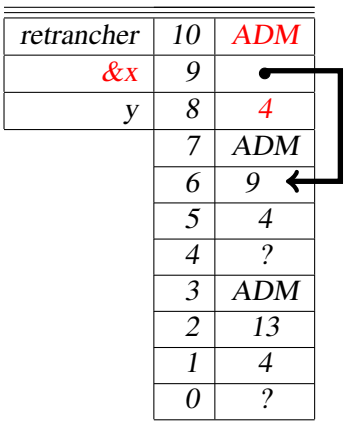
pile :



	10	ADM
	9	
	8	4
modulo	7	ADM
a	6	9
b	5	4
return	4	?
	3	ADM
	2	13
	1	4
	0	?

2eme appel à retrancher, on re-empile :

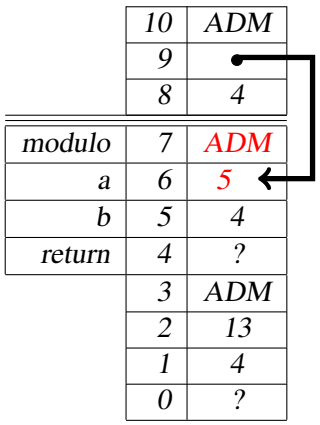
pile :



retrancher	10	ADM
&x	9	
y	8	4
	7	ADM
	6	9
	5	4
	4	?
	3	ADM
	2	13
	1	4
	0	?

calcul, modif puis retour à modulo

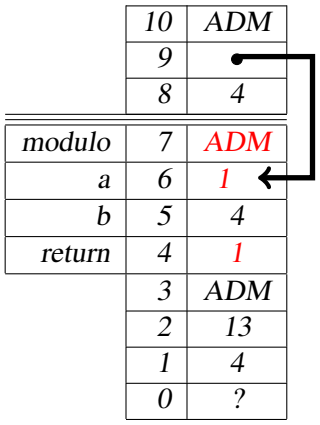
pile :



	10	ADM
	9	
	8	4
modulo	7	ADM
a	6	5
b	5	4
return	4	?
	3	ADM
	2	13
	1	4
	0	?

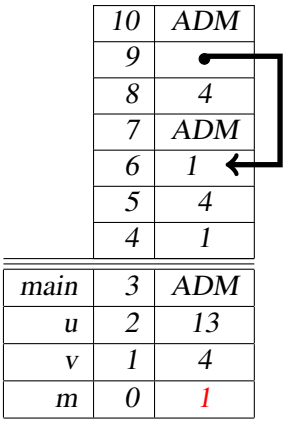
Apres le retour du 3eme appel à retrancher, avant le retour au main

pile :



	10	ADM
	9	
	8	4
modulo	7	ADM
a	6	1
b	5	4
return	4	1
	3	ADM
	2	13
	1	4
	0	?

le retour au main



	10	ADM
	9	
	8	4
	7	ADM
	6	1
	5	4
	4	1
main	3	ADM
u	2	13
v	1	4
m	0	1

Exercice 3.

1. Etant donnée la fonction `sqrt` qui calcule la racine d'un nombre, réalisez une fonction `p` qui prend en donnée un réel `x`, puis qui renvoie un booléen et transmet en résultat un réel. Si la racine carrée de (x-1)*(2-x) est définie au point `x`, le réel sera cette racine et le booléen vaudra `true`, sinon le réel sera nul et le booléen vaudra `false`.
2. Réalisez le programme principal qui, par deux fois, appelle la fonction `p` avec un paramètre réel dont la valeur est demandée à l'utilisateur et qui, si le booléen transmis est `true`, affiche le réel transmis.
3. Donnez les **tableaux d'activation** et les états successifs de la **mémoire** pour les principales étapes du programme. Précisez les affichages à l'écran lorsqu'il y en a. On supposera que l'utilisateur rentre successivement pour `x` les valeurs 1.1 puis 2.5.

Correction :

```
bool p(float x , float &r) {
  bool b;
  if ((x >= 1) and (x <= 2)  {
        b = true;
        r = sqrt((x-1)*(2-x));
  } else {
        b = false;
        r = 0;
  }
 return b;
}

int main() {
  float x, res;
  for (int i = 0; i < 2; i++) {
      cout << "Entrez x : " << endl;
      cin >> x ;       // on suppose qu'on lit x = 1.1 puis 2.5
      if (p(x, res)) {
        cout << "p(x) vaut " << res << endl;
      } else  {
        cout << " p( " << x << ") n est pas definie " << endl ;
      }
   }
  }
```

Tableaux d'activation :

p
x
&r
b
return

main
x
res
i

Exécution du main pour le 1er tour de boucle, avec x = 1,1 (à faire ensuite avec x = 2,5)

pile :

	4	*?*
main	*3*	*ADM*
x	*2*	*1.1*
res	*1*	*?*
i	*0*	*1*

initialisation

pile :

p	*8*	*ADM*
x	*7*	*1.1*
&r	*6*	●
b	*5*	*?*
return	*4*	*?*
	3	*ADM*
	2	*1.1*
	1	*?*
	0	*1*

les valeurs passées

pile :

	8	*ADM*
	7	*1.1*
	6	●
	5	*true*
	4	*true*
main	*3*	*ADM*
x	*2*	*1.1*
res	*1*	*0.3*
i	*0*	*1*

Partie sur les structures

Exercice 4. Considérons la définition ci-dessous du type structuré "Point", qui utilise deux champs x et y, qui sont les coordonnées d'un point dans le plan.

```
struct Point {
     float x, y ;
};
```

On va utiliser la procédure suivante qui permet de créer et de transmettre en résultat une variable de type `Point`, à partir de ses coordonnées lues auprès de l'utilisateur. Notez qu'au lieu de transmettre des `float`, ou des `int`, on renvoie un .... `Point` !!.

```
void litPoint (Point &p) {
    cout << "abscisse ? ";  cin  >> p.x ;
    cout << "ordonnee ? ";  cin  >> p.y ;
}
```

La notation pointée permet d'écrire dans les champs d'une structure, mais également d'accéder à ces champs, comme le montre la fonction suivante :

```
float abscisse (Point p) {
    return p.x ;
}
```

1. Ecrivez une fonction qui renvoie l'ordonnée d'un point.
2. Ecrivez une procédure qui affiche les coordonnées d'un point puis réalisez un programme principal utilisant cette procédure.
3. Réalisez une fonction `distance` qui renvoie la distance entre deux points donnés. Rappel : $dist_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
4. Ecrivez une fonction qui étant donnés deux points A et B, renvoie `true` si les deux points ne sont pas confondus, et transmet en résultat le point situé au milieu du segment AB.
5. Utilisez la fonction `EgalApproché` donnée ci-dessous pour vérifier que le milieu calculé dans la fonction précédente est bien à égale distance des deux extrémités du segment.

```
const float e = 1e-6 ;
bool EgalApproche( float x, float y){
   return (fabs(x-y) < e * fabs(x)) and (fabs(x-y) < e * fabs(y) );
}
```

où `fabs()` est la fonction valeur absolue pour les réels.

Correction :

```
float ordonnee (Point p) { return p.y ;}

void affichePoint (Point p) {
    cout << "abscisse = " <<  abscisse(p) << endl ;
    cout << "ordonnee = " <<  ordonnee(p) << endl ;
}

int main () {
    Point pp ;
    litPoint(pp);
    affichePoint(pp);
}

float distance (Point a, Point b) {
    float dx, dy ;
    dx = b.x - a.x ;
    dy = b.y - a.y ;
    return sqrt(dx * dx + dy * dy);
}

bool milieu (Point a, Point b, Point &m){
    bool res = ((a.x != b.x) or (a.y != b.y));
    m.x = (x.a+ x.b)/2 ;
    m.y = (y.a+ y.b)/2 ;
    return res ;
}

int main () {
   Point pa, pb, pm;
   litPoint(pa); litPoint(pb);
   if (not milieu(pa,pb,pm)) cout << "vos 2 points sont confondus" << endl;
   else {
       if (egalApproche( distance(pa, pm),  distance(pb, pm) )) {
         cout << " le milieu est en : " << endl;
         affichePoint(pm);
       } else {
            cout << "rate ! votre point est en : " << endl;
            affichePoint(pm);
       }
   }
}
```