TD n° 4

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 ?

Exercice 2.

1. Réalisez la procédure `retrancher` 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.

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.

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.