# Programmation Impérative

Partiel session 2 25/06/2014, salle B134

Nom _____
Prénom _____
Login _____

Les questions sont à traiter dans l'ordre qui vous plaira.
Ce sujet doit être remis avec votre copie. Quand une fonction est demandée, c'est pas opposition à un programme, rien ne vous interdit d'utiliser et donc de définir deux fonctions ou plus.

## A) Exercices

1) Soit la déclaration de la table 2, donnez une fonction qui, pour une valeur, renvoie le nombre d'éléments du vecteur qui sont plus grands que cette valeur.
2) Donnez maintenant une fonction qui renvoie l'élément *médian* du vecteur (c'est-à-dire celui qui a autant d'éléments plus petits que lui que d'éléments plus grands, à un près).
3) Donnez toutes les étapes de l'exécution de la fonction *oups* (cf. table 3) avec comme paramètres 8 et 1000.

## B) Boîtes

1) Soit les déclarations de la table 1, donnez une fonction qui cherche, dans une *boite* l'élément dont l'indice est le plus grand.
2) Donnez une fonction qui permet de ranger la *boite* de façon à ce que tous les élements soient rangés dans l'ordre des *indices* croissants.
3) Donnez une fonction qui remplit le champ *total* par la valeur du champ *nbh* multipliée par celle du champ *indice* pour tous les éléments de la boite. 4) Combien d'octets utilise un paquet ?
Combien d'octets utilise une boite si son champ nb vaut 10 ?

## C) Arbres

Voici (en table 4) un type de données, `arbre`.

1. Faites une fonction qui affiche toutes les valeurs d'un tel arbre.
2. Faites tourner la fonction `valueadetax` (voir table 5) avec comme paramètre la valeur 98704. La mémoire est donnée dans le tableau 6.
   Attention, pour faire tourner cette fonction, il faut, comme en cours, montrer ce qui arrive à chaque case définie.

```
struct paquet {
  int num;
  char nom [20];
  double nbh,
  int indice;
  double total;
} ;
typedef struct paquet paquet
struct boite {
  int nb;
  paquet * champs;
} ;
typedef struct boite boite;
```

Table 1 – Type de données structurées

```
struct vecteur {
  int nb;
  float * vec;
} ;
typedef struct vecteur vecteur;
```

Table 2 – Type pour vecteur

```
void oups (int n, vecteur * v) {
  float z, t;
  z = -1.0;
  t = -1.0;
  v->nb = n;
  v->vec = (float *) malloc ((size_t) n * sizeof(float));
  while (n--) {
    v->vec[n] = z;
    z += t;
    t = z - t;
  }
}
```

Table 3 – Fonction oups

```
struct cell {
  int num;
  float val;
  struct cell * d[3];
} ;
typedef struct cell cell;
typedef struct cell * arbre;
```

TABLE 4 – Types arbres

```
float valuea (arbre a, float ref) {
  float tot;
  int i;

  if (! a)
     return 0.0;
  tot = ref - a->val;
  for (i= 0; i < 3; i++)
      tot += valuea (a->d[i], a->val);
  return tot;
}
float valueadetax(arbre a) {
  if ( !a)
    return 0.0;
  return valuea (a, 0.0);
}
```

TABLE 5 – Fonction valueadetax

3. Faites une fonction qui renvoie l'élément dont le champ `val` est le plus grand.
4. Faites une fonction qui calcule le nombre de noeuds de l'arbre.
5. Faites une fonction qui calcule le nombre de feuilles de l'arbre (une feuille est un noeud qui n'a aucun successeur).

## D) Fichiers

1. Écrire une fonction qui place dans un fichier tous les éléments d'un vecteur comme celui défini dans la table 1.
2. Faire une fonction qui permet de calculer combien d'éléments de type `paquet` (voir table 1) sont écrits dans le fichier donné en argument.
3. Faire une fonction qui permet de sauvegarder dans un fichier tous les éléments permettant de reconstruire un arbre (cf. table 4) donné en argument.

| adresse | x | adresse | x | adresse | x |
|---|---|---|---|---|---|
| 98704 | 12 | 98792 | 98800 | 98880 | 98896 |
| 98708 | -2.10 | 98800 | 4 | 98888 | 0 |
| 98712 | 98736 | 98804 | 13.10 | 98896 | 2 |
| 98720 | 98832 | 98808 | 0 | 98900 | 7.00 |
| 98728 | 0 | 98816 | 98864 | 98904 | 0 |
| 98736 | 14 | 98824 | 0 | 98912 | 0 |
| 98740 | 5.50 | 98832 | 7 | 98920 | 0 |
| 98744 | 98928 | 98836 | 11.20 | 98928 | 11 |
| 98752 | 0 | 98840 | 0 | 98932 | 12.30 |
| 98760 | 0 | 98848 | 98768 | 98936 | 0 |
| 98768 | 10 | 98856 | 0 | 98944 | 0 |
| 98772 | -7.60 | 98864 | 3 | 98952 | 0 |
| 98776 | 0 | 98868 | -5.30 | | |
| 98784 | 0 | 98872 | 0 | | |

TABLE 6 – Adresses et contenu de la mémoire