Language de programmation L2

Le langage C

TD2

Fonction récursive

La suite de *Fibonacci* définie par :

```
Fibonacci (N) = 0, Si N=0;
Fibonacci(N) = 1, Si N=1;
Fibonacci(N) = Fibonacci(N-2) + Fibonacci(N-1), Si N ≥ 2;
```

Programmer la fonction *Fibonacci qui prend un entier en argument et retourne sa valeur de Fibonacci.*

Calculette avec *scanf* et *printf*

Ecrire le programme *calculette* en décomposant l'application en fonctions pour la saisie, le calcul et l'affichage du résultat.

Tri à Bulle

Il s'agit du plus simple des algorithmes de tri. Le tri fonctionne par parcours successifs du tableau.

A chaque parcours, l'algorithme compare les couples d'éléments successifs. Si deux éléments successifs ne sont pas dans l'ordre alors ils sont échangés dans le tableau. Après un premier parcours, un autre est exécuté mais pour une taille de tableau diminuée de 1. Si aucun échange n'est effectué pendant un parcours, cela signifie que le tableau est trié.

Exercice 1

Ecrire la fonction *echange* qui permet d'échanger la valeur de deux entiers passés en argument (par adresse).

Exercice 2

En utilisant *echange*, programmez les fonctions nécessaires au tri à bulle d'un tableau d'entiers.

Exercice 3

On considère le type t_point suivant :

```
typedef struct {
      int x, y
} t_point ;
```

Adaptez vos fonctions de tri pour un tableau de points, triés par ordre d'abscisse.

Opérations sur les pointeurs

Exercice 1

Voici le tableau T :

```
int T[] = {56,67,34,1,0,6,34,56,98,45,12};
```

On considère que T est à l'adresse 100.

Donnez la valeur ou adresse de chacune de ces expressions :

- T[5]

- T+5
- *T+5
- *(T+5)
- &T+4
- &T[3]-4
- T+(*T-1)
- *(T * *(T+2)-T[3])

Exercice 2

Compléter le tableau en indiquant les valeurs des différentes variables au terme de chaque instruction du programme suivant (pour les pointeurs on peut indiquer sur quoi ils pointent).

Programme	***a***	***b***	***c***	***p1***	****p1***	***p2***	****p2***
`int a,b,c,*p1,*p2;`							
`a = 5; b = 3; c = 6;`							
`p1 = &a; p2 = &c;`							
`*p1 = (*p2)++;`							
`p1 = p2;`							
`p2 = &b;`							
`*p1 -= *p2;`							
`++*p2;`							
`*p1 *= *p2;`							
`a = ++*p2 * *p1;`							
`p1 = &a;`							
`*p2 = *p1 /= *p2;`							

Allocation dynamique et tableau

Il est souvent utile de disposer de tableaux maintenus triés : lorsqu'on ajoute un élément au tableau, la position du nouvel élément est calculé de façon à ce que le tableau demeure trié après l'ajout.

Exercice 1

Voici une définition d'un type t_tableau_trie. L'élément *tab* pointe sur le premier entier du tableau et *nb* contient le nombre d'éléments ajoutés au tableau:

```
typedef struct {
      int * tab ;
      int nb ;
} t_tableau_trie ;
```

Programmez les fonctions :

```
void tableau_trie_creer (t_tableau_trie * self) ;
void tableau_trie_detruire (t_tableau_trie * self) ;
void tableau_trie_ajouter (t_tableau_trie * self, int v) ;
void tableau_trie_afficher (t_tableau_trie * self) ;
```

Exercice 2

Adaptez les fonctions de l'exercice 1 pour un tableau d'éléments de type *t_personne* triés par nom et numéro de téléphone.

```
typedef struct {
      char nom[MAX_NOM];
      char tel[MAX_TEL];
} t_personne;
```