

TD10 – Nombre variable d’arguments, pointeurs vers des fonctions

Langage C (LC4)

semaine du 8 avril

1 Fonctions à nombre variable d’arguments

La page « `man stdarg` » vous est donnée en document attaché.

Question 1. Écrire une fonction `moyenne` à nombre variable d’arguments prenant un premier argument fixe de type entier indiquant le nombre d’arguments qui suivent, supposés de type `double`, et renvoyant leur moyenne arithmétique. On rappelle que la moyenne arithmétique d’un ensemble de nombres $(x_i)_{1 \leq i \leq n}$ est donnée par :

$$\frac{1}{n} \sum_{i=1}^n x_i$$

Question 2. Écrire une fonction `concatenation` qui prend en argument fixe un caractère `sep` suivi de chaînes de caractères puis d’un pointeur `NULL` et affiche les chaînes dans l’ordre, séparées par le caractère `sep`.

Pour la question suivante on suppose que `void print_objet_bizarre(objet_bizarre o)` sait afficher des `objet_bizarre`.

Question 3. Écrire une fonction `monprintf` prenant en argument une chaîne de caractères pouvant contenir un certain nombre de fois le motif `%O` et affichant sur la sortie standard la chaîne où les différentes occurrences de `%O` sont remplacées par les arguments suivants de l’appel à `monprintf`, qui seront du type `objet_bizarre`.

2 Pointeurs vers des fonctions

2.1 Utilisation simple

Question 4. Écrire une fonction `int fois_deux(int i)` qui renvoie le double de l’entier qui lui est donné en argument.

Question 5. Écrire une fonction `void appliquer_tableau(int (*f)(int), int *t, int n)` qui applique une fonction `f`, au tableau `t` de taille `n`.

2.2 Tris

Voici un algorithme de tri des éléments d'un tableau dans l'ordre croissant :

```
void tri_croissant(int *t, int n)
{
    int i, i_min, j, tmp;
    for (i = 0; i < n - 1; i++) {
        i_min = i;
        for (j = i; j < n; j++)
            if (t[j] < t[i_min])
                i_min = j;
        tmp = t[i_min];
        t[i_min] = t[i];
        t[i] = tmp;
    }
}
```

Voici un algorithme de tri des éléments d'un tableau dans l'ordre décroissant :

```
void tri_decroissant(int *t, int n)
{
    int i, i_min, j, tmp;
    for (i = 0; i < n - 1; i++) {
        i_min = i;
        for (j = i; j < n; j++)
            if (t[j] > t[i_min])
                i_min = j;
        tmp = t[i_min];
        t[i_min] = t[i];
        t[i] = tmp;
    }
}
```

Ces deux algorithmes sont identiques, à l'exception de l'opérateur de comparaison. Afin d'éviter de dupliquer du code, nous allons utiliser les pointeurs de fonctions.

Question 6. Écrire une fonction `int superieur(int a, int b)` qui renvoie 1 si `a` est supérieur à `b`, 0 s'ils sont égaux et `-1` sinon.

Question 7. Écrire une fonction `int inferieur(int a, int b)` qui renvoie 1 si `a` est inférieur à `b`, 0 s'ils sont égaux et `-1` sinon.

Question 8. Écrire une fonction `tri`, qui trie un tableau `t` de taille `n` selon l'ordre donné par une fonction `compare`.

Question 9. Modifier le programme pour qu'il classe les éléments du tableau en mettant les entiers pairs au début et les entiers impairs à la fin.