

TD 4 : Fonctions et récursivité

Cette feuille de TD a pour but l'apprentissage des fonctions en langage C. Les objectifs sont d'une part le découpage du code suivant la logique du problème abordé, et d'autre part l'utilisation de la récursivité pour le traitement de données.

Dans votre répertoire nommé TDC/, créez un sous-répertoire TDC/td4/ (avec la commande `mkdir` dans un terminal). Positionnez-vous dans ce sous-répertoire (avec la commande `cd`), vous y resterez toute cette séance.

1 Premières fonctions

Exercice 1:

Citez des exemples de fonctions que vous avez utilisées dans vos programmes précédents. Pour chacune de ces fonctions, donnez sa définition (ou prototype) en vous servant de la commande `man`.

Exercice 2: Calcul du carré d'un nombre

Écrivez une fonction `sqr` de prototype :

```
double sqr(double x);
```

qui renvoie la valeur x^2 . Vous écrirez ensuite une fonction `main` qui affiche la valeur de x^2 pour une valeur de x lue au clavier.

Exercice 3:

Généralisez le programme de l'exercice précédent pour qu'il calcule la puissance nième de x . Pour cela vous écrirez une fonction `pow` de prototype :

```
double pow(double x, unsigned int n);
```

qui renvoie la valeur x^n . Votre fonction principale lira les valeurs de x et de n et affichera le résultat.

Exercice 4: Fonctions sur les chaînes de caractères

Écrivez les fonctions suivantes de manipulation de chaînes de caractères. Attention à bien faire en sorte que les résultats produits par vos fonctions sont bien des chaînes de caractères (le cas échéant) : n'oubliez la butée `'\0'`.

- `int strlen(char s[])` qui retourne la longueur de la chaîne de caractères `s`
- `int strcmp(char s1[], char s2[])` qui retourne 0 si les deux chaînes sont identiques, un nombre négatif si `s1` est lexicographiquement inférieure à `s2`, et un nombre positif dans le cas contraire
- `void str_cat(char s1[], char s2[])` qui concatène `s2` à `s1`
- `void str_ncat(char s1[], char s2[], unsigned int n)` qui concatène au plus les `n` premiers caractères de `s2` à `s1`, puis ajoute la butée `'\0'`
- `void strcpy(char s1[], char s2[])` qui copie la chaîne `s2` dans `s1`
- `void strncpy(char s1[], char s2[], unsigned int n)` qui copie au plus les `n` premiers caractères de `s2` dans `s1`, puis ajoute la butée `'\0'`

2 Portée et visibilité

Exercice 5:

On considère le programme suivant :

```
#include <stdio.h>

int n=0;

void incrementer();

void incrementer(){
    n++;
    printf("appel numero %d\n",n);
}

int main(){
    int i;
    for(i=0; i<5; i++){
        incrementer();

    return 0;
}
```

1. Quelles sont les variables locales de ce programme ? À quelles lignes sont-elles déclarées ? *i, L3;*
2. Quelles sont les variables globales de ce programme ? À quelles lignes sont-elles déclarées ? *n, L2;*
3. À quelle(s) ligne(s) est déclarée la fonction `incrementer` ? À quelle(s) ligne(s) est-elle définie ? Mêmes questions pour la fonction `main`
4. Copiez-collez ce programme dans un buffer `emacs`, compilez-le et exécutez-le
5. Déclarez une variable `n` locale à la fonction `incrementer` et compilez puis exécutez à nouveau le programme. Que constatez-vous ?
6. Déclarez une variable `t` locale à la fonction `main`, et incrémentez `t` dans la fonction `incrementer`. Compilez puis exécutez à nouveau ce programme. Que constatez-vous ?

3 Récursivité

Exercice 6: Factorielle

Écrivez une fonction récursive `fact` de prototype : `unsigned int fact(unsigned int n)` qui calcule la factorielle de `n`. Écrivez une fonction `main` qui lit une valeur de `n` au clavier, puis qui affiche la valeur de la factorielle de `n`.

Exercice 7: Puissance

En remarquant que :

$$x^n = \begin{cases} 1 & n = 0 \\ (x^{n/2})^2 & n \text{ pair (non nul)} \\ x \times (x^{(n-1)/2})^2 & n \text{ impair} \end{cases}$$

Écrivez une fonction récursive `power` de prototype : `double power(double x, unsigned int n)` qui renvoie la puissance nième de `x`. Vous écrirez une fonction `main` qui lit les valeurs de `x` et de `n` au clavier, qui appelle `power` et affiche le résultat.

Exercice 8: Fibonacci

La suite de Fibonacci est définie par :

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$

Écrivez une fonction `fibonacci` de prototype : `unsigned int fibonacci(unsigned int n)` qui renvoie la valeur de f_n . Vous écrirez une fonction `main` qui lit une valeur de `n` au clavier et affiche la valeur calculée.

Exercice 9: Décomposition binaire

Écrivez une fonction `decompose2` de signature : `void decompose2(unsigned int n)` et qui affiche la décomposition binaire de `n`. Vous écrirez une fonction principale qui lit la valeur de `n` et appelle `decompose2`.

Exercice 10: Décomposition en base b

Généralisez l'exercice précédent pour que la fonction `decompose` affiche la décomposition en base $b \leq 10$.

Exercice 11: Décomposition en base 16

Même exercice que la décomposition binaire pour obtenir cette fois-ci une décomposition hexadécimale

Exercice 12: Fonctions mutuellement récursives

Écrivez deux fonctions pair et impair qui prennent chacune une valeur de type unsigned int en paramètre et qui renvoie 1 lorsque leur paramètre est pair/impair respectivement, et 0 sinon. Vous écrirez une fonction principale main qui lit un entier non signé au clavier et qui affiche la parité de celui-ci

Exercice 13: Récursivité terminale

Copiez-collez le programme suivant dans un buffer emacs, compilez-le et exécutez-le.

```
#include <stdio.h>

unsigned int f1(unsigned int n) {
    if (n == 0)
        return 1;
    else
        return (n*f1(n-1));
}

unsigned int f2(unsigned int n, unsigned int r) {
    if (n == 0)
        return r;
    else
        return f2(n-1, n*r);
}

unsigned int f2_stub(unsigned int n) {
    return f2(n, 1);
}

int main() {
    unsigned int n;
    printf("n: ");
    scanf("%u", &n);
    printf("f1(%u)=%u\n", n, f1(n));
    printf("f2(%u)=%u\n", n, f2_stub(n));

    return 0;
}
```

Que font les fonctions f1 et f2? Que pensez-vous de ces deux implantations?

*elles calculent le factoriel
f2 est plus rapide car on stocke le résultat dans une nouvelle variable
à chaque appel récursif.*

Exercice 14: Retour sur la suite de Fibonacci

En vous inspirant de l'exercice précédent, écrivez une fonction récursive **fibonacci** qui calcule le n ème terme de la suite de Fibonacci, sans jamais calculer deux fois un même terme