Université Paris-Nord — L1
Institut Galilée — Année 2010-2011
Licence 1 - 1er semestre

Travaux dirigés 11 : fonctions, fonctions récursives

1 Fonctions récursives

```
1  /* Declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* EXIT_SUCCESS */
3  #include <stdio.h> /* printf() */
4
5  /* Declarations constantes et types utilisateurs */
6
7  /* Declarations de fonctions utilisateurs */
8  int factorielle(int n);
9
10 /* Fonction principale */
11 int main()
12 {
13     /* Declaration et initialisation des variables */
14     int x = 3; /* argument */
15     int res;   /* resultat */
16
17     /* calcul */
18     res = factorielle(x);
19
20     /* affichage */
21     printf("%d! = %d\n", x, res);
22
23     /* Valeur fonction */
24     return EXIT_SUCCESS;
25 }
26
27 /* Definitions de fonctions utilisateurs */
28 int factorielle(int n)
29 {
30     int res; /* resultat */
31     if (n > 1) /* cas récursif */
32     {
33         res = n * factorielle(n - 1);
34     }
35     else /* cas de base */
36     {
37         res = 1;
38     }
39     return res;
40 }
41
```

1. Faire la trace du programme précédent.
2. La suite de Fibonacci est définie récursivement par la relation $u_n = u_{n-1} + u_{n-2}$. Cette définition doit être complété par une condition d'arrêt, par exemple : $u_1 = u_2 = 1$. Écrire

une fonction qui calcule et renvoie le n-ième terme de la suite de Fibonacci ($n \in \mathbb{N}^*$ donné en argument de la fonction).

3. Il n'est parfois pas suffisant d'avoir un bon cas de base, voici un exemple. En C, que vaut `Morris(1, 0)` ?

```
int Morris(int a, int b)
{
    if (a == 0)
    {
        return 1;
    }
    else
    {
        return Morris(a - 1, Morris(a, b));
    }
}
```

4. Les fonctions récursives mêmes simples donnent parfois des résultats difficiles à prévoir. Pour s'en convaincre voici un exemple. Pour $n > 100$ la fonction 91 de McCarthy vaut $n - 10$. Mais pour $n < 100$? (Tester sur un exemple... pas trop mal choisi).

```
int McCarthy(int x)
{
    if (x > 100)
    {
        return (x-10);
    }
    else
    {
        return McCarthy(McCarthy(x + 11));
    }
}
```

2 PGCD

Rappel : pour a et d deux entiers naturels, on dit que d divise a, et on note $d \mid a$, s'il existe un entier q tel que $a = dq$. En particulier $d \mid 0$ quel que soit d. Le plus grand diviseur commun (PGCD) de deux entiers naturels a et b est le plus grand entier d tel que $d \mid a$ et $d \mid b$. Cette définition donne un moyen effectif de trouver le PGCD de deux entiers (un algorithme) : il suffit de chercher tous les diviseurs de a et de b et de prendre le plus grand d'entre eux.

1. Que vaut $\mathrm{PGCD}(0, 0)$?
2. Si seulement un des entiers a et b est nul, que vaut $\mathrm{PGCD}(a, b)$?
3. Si aucun des deux entiers a et b n'est nul, jusqu'a quelle valeur doit-on chercher des diviseurs de a et de b ?
4. Comment tester si un nombre en divise un autre, en C ?
5. Écrire une fonction non récursive qui calcule le PGCD de deux entiers naturels.
6. Rappeler l'algorithme d'Euclide pour le calcul du PGCD.
7. Écrire une fonction récursive pour le calcul du PGCD de deux entiers.
8. À votre avis quelle est la méthode la plus rapide ?

3 Conjecture de Syracuse

Soit la fonction mathématique $f : \mathbb{N} \to \mathbb{N}$ définie par :

$$f : x \mapsto \begin{cases} x/2 \text{ si } x \text{ pair} \\ 3x + 1 \text{ sinon.} \end{cases}$$

On appelle cette fonction la fonction de Collatz et comme vous pourrez le vérifier sur internet elle a donné et donne toujours du fil à retordre aux mathématiciens. Voici pourquoi.

Soit un entier positif x. Si on calcule $f(x)$, puis $f(f(x))$, puis $f(f(f(x)))$ etc. on finit toujours par tomber sur la valeur 1 quelle que soit la valeur de $x \in \mathbb{N}^*$. Aucun mathématicien n'est arrivé à ce jour à le démontrer (et on ne sait même pas démontrer que c'est indécidable, c'est à dire non démontrable à l'aide des mathématiques usuelles). Cela reste une conjecture.

On peut également formuler le problème avec une suite définie par récurrence. On prend u_0 égal à x, et $u_{n+1} = f(u_n)$. Les termes successifs de la suite sont alors : x, $f(x)$, $f(f(x))$ etc. La conjecture est que la suite (u_n) finit par atteindre la valeur 1, pour tout $u_0 = x \in \mathbb{N}^*$. Si on poursuit le calcul après avoir trouvé une première fois 1, la suite devient périodique, de période 4, 2, 1. On dit que la suite est entrée dans un puit. Par exemple en partant de 15 on a la suite :

$$15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, (4, 2, 1, 4, 2, 1, \ldots)$$

Les mathématicien parlent de la trajectoire de x pour désigner la suite des valeurs prises par (u_n) avant d'atteindre le premier 1 lorsque la suite part de $u_0 = x$. Ils définissent également le temps de vol comme étant le nombre de termes avant le premier 1 (ici 17) et ils désignent par altitude maximale, la valeur maximale prise par la suite (ici 160).

1. Écrire en langage C une fonction `Collatz` calculant la valeur de la fonction f sur son argument.
2. Écrire une procédure (un fonction ne renvoyant pas de résultat) *récursive* `Syracuse` qui calcule les itérations successives de la fonction de Collatz sur un entier donné en argument, jusqu'à trouver 1. Pour le moment la fonction `Syracuse` n'affichera rien et ne renverra pas de résultat (mais elle doit calculer chacun des termes successifs de Collatz !). Si vous ne trouvez pas, vous pouvez écrire une procédure non récursive qui utilise plus classiquement un while pour calculer tour à tour toutes les valeurs de la suite.
3. Ajouter ensuite un affichage des valeurs successives trouvées dans cette fonction. Dans l'ordre direct. Puis dans l'ordre inverse. Comment feriez-vous pour obtenir l'ordre direct avec un while ? et l'ordre inverse ?
4. En reprenant le code de Syracuse écrire une fonction récursive `temps_de_vol` qui renvoie le nombre d'itérations de la fonction de Collatz nécessaire pour atteindre 1.
5. Question facultative. Comment faire en sorte que le temps de vol soit affiché plutôt que renvoyé comme valeur de retour ?
6. Question facultative. En reprenant le code précédant écrire une nouvelle fonction `altitude` qui renvoie la valeur maximum prise par les termes successifs de l'itération (x compris).

Université Paris-Nord — L1
Institut Galilée — Année 2010-2011
Licence 1 - 1er semestre

Travaux pratiques 11 : Syracuse au menu

Résumé de l'épisode précédent. On choisit un entier $u_0 > 0$ comme premier terme. Si il est pair on le divise par deux si il est impair on le multiplie par trois et on ajoute un. On obtient ainsi le terme suivant. Et on recommence. La conjecture est qu'on atteindra 1. Le nombre d'itérations nécessaires pour atteindre pour la première fois la valeur 1 est appelé *temps de vol* et la valeur maximale prise par la suite est appelée *altitude maximale*.

1. La fonction `Collatz` du TD calcule le terme suivant de la suite à partir du terme courant. Créer un nouveau programme *collatz.c* dans le répertoire du TP11 et ajoutez tout de suite la déclaration et la définition de cette fonction aux bons endroits.
2. Déclarer et définir une procédure `Syracuse` qui calcule les termes successifs d'une suite de Syracuse construite à partir d'un premier terme fourni en argument, jusqu'à rencontrer pour la première fois 1. Cette procédure devra afficher les termes succesifs de cette suite. *Vous pouvez programmer de manière récursive si vous êtes à l'aise avec la récursivité et avec les précédents chapitres (notamment les boucles), sinon faite le de manière itérative, c'est à dire avec une boucle... à votre avis,* `for` *ou* `while` *?* Tester votre procédure en l'appelant dans le `main` sur quelques valeurs.
   Exemple : `Départ 15 : 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1`.
3. Dans la procédure, trouver et afficher l'altitude maximale et le temps de vol (Tester). Ajouter un argument booléen à votre procédure `Syracuse` qui déterminera s'il faut afficher ou non les termes de la suite. Exemple (avec le booléen à faux) :

   ```
   Départ 15 : temps de vol 17, altitude maximale 160
   ```

4. Écrire une fonction `conjecture` qui exécute tour à tour la procédure `Syracuse` sur les entiers 2 à n, pour n donné en argument, sans afficher les termes de la suite.

   ```
   Tester de 2 a 4
   Depart 2 : temps de vol 1, altitude maximale 2
   Depart 3 : temps de vol 7, altitude maximale 16
   Depart 4 : temps de vol 2, altitude maximale 4
   ```

5. Ajouter un menu à votre programme proposant d'afficher les termes de la suite pour une valeur saisie par l'utilisateur ou de tester la conjecture jusqu'à une valeur saisie par l'utilisateur. Pensez à réutiliser des fonctions que vous avez déjà écrites en TP !
6. Ajouter à votre menu la possibilité de tester la conjecture sur tous les entiers indéfiniement (`ctrl-C` pour arrêter).

Fonction 91

Vérifier que la fonction 91 de McCarthy renvoie bien 91 pour les valeurs de n telles que $50 \leq n \leq 100$.