Stage d'algorithmique n°2

Cours d'Informatique de Deuxième Année

—DEUG Sciences M3-MI—

Procédure et fonction, appel par valeur

Lors de cette séance, nous allons introduire la notion de fonction en langage C, et plus précisément les points suivants :

- déclaration d'une fonction ;
- appel par valeur ;
- valeur de retour d'une fonction.

▸ **Exercice 1. (factorielle d'un entier)** *La fonction factorielle est définie par les relations suivantes :*

$$n! = \begin{cases} 1 & \text{si } n = 0 \ , \\ n \times (n-1) \times \cdots \times 3 \times 2 \times 1 & \text{si } n > 0 \ . \end{cases}$$

Ecrire la fonction `Factorielle` *qui prend en argument* `n` *et qui calcule la factorielle de* `n`.

▸ **Exercice 2. (coefficients binomiaux)** *Nous cherchons à calculer les coefficients binomiaux C_n^p définis par les relations suivantes :*

$$C_n^p = \begin{cases} \dfrac{n!}{p!(n-p)!} & \text{si } 0 \leq p \leq n \ , \\ \\ 0 & \text{sinon.} \end{cases}$$

Ecrire la fonction `Binomial` *qui prend comme arguments* `n` *et* `p` *et qui calcule le coefficient binomial C_n^p.*

▸ **Exercice 3.** *Nous cherchons à étudier les propriétés des coefficients binomiaux. Nous nous intéressons à la somme suivante :*

$$\sum_{i=0}^{n} C_n^i \quad .$$

Ecrire la fonction `Sum` *qui prend en argument* `n` *et qui calcule cette somme. Essayer cette fonction pour $n = 2, 3, 4, 5, \ldots$. En déduire une évaluation de la somme ci-dessus.*

▸ **Exercice 4.** *Ecrire une fonction* `SommeEntier` *qui prend en argument un entier et qui calcule la somme des chiffres composant cet entier.*

▸ **Exercice 5.** *Ecrire une fonction* `PremierCarre` *qui prend en argument un entier* `n` *et qui calcule le premier carré supérieur à* `n`*.*

▸ **Exercice 6.** *Un nombre est dit parfait s'il est égal à la somme de ses diviseurs propres. Par exemple,* 6 *est parfait* ($6 = 1 + 2 + 3$)*. Ecrire une fonction* `EstParfait` *qui prend en argument un entier* `n` *et qui retourne* 1 *si* `n` *est parfait et* 0 *sinon. En déduire une fonction* `Parfait` *qui affiche les nombres parfaits inférieurs à* 1000*.*

▸ **Exercice 7.** *On cherche à afficher la décomposition en base* 2 *d'un entier* `n`*. Le principe est très simple. On effectue la division entière de* `n` *par* 2*. On trouve un quotient* q_0 *et un reste* r_0*. On recommence avec* q_0*. On construit ainsi une suite de reste* $(r_0, r_1, r_2, \ldots)$*. On arrête le processus dès que l'on trouve un quotient* q_k *égal à* 0*. La décomposition de* `n` *en base* 2 *est alors* $(r_k, r_{k-1}, \ldots, r_2, r_1, r_0)$*. Par exemple pour* $n = 10$ *:*

$$\begin{array}{rcrcrcl} 10 & = & 5 \times 2 & + & 0 & \longrightarrow & r_0 = 0 \\ 5 & = & 2 \times 2 & + & 1 & \longrightarrow & r_1 = 1 \\ 2 & = & 1 \times 2 & + & 0 & \longrightarrow & r_2 = 0 \\ 1 & = & 0 \times 2 & + & 1 & \longrightarrow & r_3 = 1 \end{array}$$

La représentation de 10 *en base* 2 *est donc* 1010*.*

En déduire un algorithme de passage d'une écriture en base 10 *vers une écriture en base* 2*. Ecrire un programme qui lit un entier entré au clavier et qui affiche cette décomposition. Pour l'exemple donné, on voudrait voir l'affichage suivant :*

```
0*2^0 + 1*2^1 + 0*2^2 + 1*2^3
```

▸ **Exercice 8. (valeur de retour d'une fonction)** *On cherche à lire un entier strictement positif. On propose le programme suivant :*

```
#include <stdio.h>
void main(void)
{ int n;

  n=0;
  while (n<=0)
  { printf("entrer un nombre : ");
    scanf("%d",&n);
    printf("J'ai lu :%d\n",n);
  }
}
```

Que se passe-t-il lors d'une erreur de frappe ? Comment modifier le programme pour que la saisie continue même en cas d'erreurs de frappe ?