Initiation à la programmation en C

TP n°2

Antoine Miné

22 février 2007

Site du cours: `http://www.di.ens.fr/~mine/enseignement/prog2006/`

Dans ce TP (et les suivants) nous indiquerons souvent le prototype des fonctions à programmer. Ceci permet de se mettre d'accord sur le nom de la fonction, le type et l'ordre de ses arguments, etc. Vous n'êtes pas obligé de recopier ce prototype dans votre programme[1] puisque vous allez définir la fonction, et qu'une définition a valeur de déclaration...

Exercice 1. Affichage.
Programmez une fonction de prototype suivant :

```
void ligne(char c,int n);
```

qui affiche `n` fois le caractère `c` à l'écran.

Servez-vous de la fonction `ligne` pour programmer une fonction :

```
void boite(int largeur,int hauteur,int epaisseur);
```

qui dessine un cadre de `largeur` par `hauteur` caractères blancs, entouré d'un cadre d'`epaisseur` caractères `#`. L'appel à `boite(5,4,2)` devra donc afficher :

```
##########
##########
##␣␣␣␣␣##
##␣␣␣␣␣##
##␣␣␣␣␣##
##␣␣␣␣␣##
##########
##########
```

Exercice 2. Exponentielle.
Programmez une fonction de prototype suivant :

```
double puissance(double x,int n);
```

qui renvoie x^n, c'est à dire, x multiplié n fois par lui-même.

Afin de limiter le nombre de multiplications à effectuer, on fait la remarque suivante : si n est pair, alors x^n est le carré de $x^{n/2}$, si n est impair, alors $x^n = (x^{n/2})^2 \times x$ (car $n = 2 \times (n/2) + 1$, par troncature de $n/2$ dans les entiers). Dans les deux cas, le calcul de x^n se ramène au calcul de

[1] Sauf, évidement, dans le cas de déclarations en avance.

$x^{n/2}$. Il est alors commode d'utiliser, dans `puissance`, un appel à `puissance(n/2)` : on construit une *fonction récursive*. Pour ne pas boucler indéfiniment, il faut traiter à part le cas $\texttt{n} = 0$.

Exercice 3. Instrumentation de l'exponentielle.
Afin de déterminer le nombre exact de multiplications effectuées par un appel donné à `puissance`, remplacez les opérateurs `*` par un appel à une fonction `mul` à définir. Celle-ci aura les effets suivants :
– `mul` renvoie le produit de ses deux arguments,
– `mul` incrémente un compteur global, initialement nul.

Exercice 4. Fibonacci récursif.
On rappelle que la suite de Fibonacci est définie par $X_0 = X_1 = 1$ et $X_{n+2} = X_{n+1} + X_n$. L'exercice 2 suggère une programmation plus directe de X_n que celle du TP précédent : la boucle est remplacé par des appels par récurrence.

Programmez une fonction de prototype suivant :

```
int fibo(int n);
```

qui calcule X_n de cette manière.
Écrivez un programme qui affiche X_1 à X_{43}. Que constate-t-on ?

Exercice 5. Booléens.

1. Que calcule `1-(expr !=0)` ?
2. Donnez une expression qui calcule le ou exclusif de deux expressions `expr1` et `expr2` (celui-ci vaut 1 si exactement une des deux expressions est non nulle, 0 sinon).
3. Que signifie `a<=b<=c` ? (Note : `<=` associe à gauche.)

Exercice 6. Tortue Logo.
Une *tortue* possède une position (`x` et `y` de type `double`) et une direction (un entier `dir` où `0` signifie nord, `1` est, `2` sud et `3` ouest).
Programmez un fonction `avance` permettant de déplacer la tortue dans la direction courante, ainsi que `tourne_droite` et `tourne_gauche` permettant de changer son orientation de 90 degrés (sans la déplacer) :

```
void avance(double pas);
void tourne_droite();
void tourne_gauche();
```

On propose la méthode suivante pour dessiner effectivement avec la tortue :
– le programme commence par afficher à l'écran une ligne contenant le mot :

`%!postscript`

– chaque appel à `avance` affiche une ligne de la forme :

x_1 y_1 `moveto` x_2 y_2 `lineto stroke`

où x_1 et y_1 sont les coordonnées de la tortue (en flottant) avant déplacement et x_2 y_2 ses coordonnées après déplacement,
– le programme termine en affichant la ligne

`showpage`

– lors de son exécution, la sortie du programme est *redirigée* vers un fichier d'extension `.ps` : on tapera par exemple

`./a.out > toto.ps`

– après exécution du programme, le fichier `toto.ps` contient une image au format PostScript, visualisable avec l'outil `gv` :

```
gv toto.ps &
```

On propose enfin la fonction (récursive) suivante qui dessine quelque chose de joli :

```
void courbe(int n)
{
  if (n<=0) avance(3);
  else {
    tourne_droite();
    courbe(n-1);
    tourne_gauche();
    courbe(n-1);
  }
}
```