Travaux dirigés 4 : fonctions et procédures (1)

1 Trace de fonctions

Faire la trace du programme suivant et dire ce que calcule la fonction `est_xxx`.

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf() */
3  #include <stdbool.h> /* booleens */
4  /* Declaration constantes et types utilisateurs */
5
6  /* Declaration de fonctions utilisateurs */
7  bool est_xxx(int n);
8
9  /* Fonction principale */
10 int main()
11 {
12     /* Declaration et initialisation des variables */
13     int n = 9;
14
15     if (est_xxx(n)) {
16         printf("L'entier %d est xxx\n", n);
17     }
18     else {
19         printf("L'entier %d n'est pas xxx\n", n);
20     }
21
22     /* Valeur fonction */
23     return EXIT_SUCCESS;
24 }
25
26 /* Definition de fonctions utilisateurs */
27 bool est_xxx(int n)
28 {
29     int i;
30
31     for (i = 2; i < n; i = i + 1){
32         if (n % i == 0) {
33             return false;
34         }
35     }
36     return true;
37 }
```

2 Déclaration et définition de fonctions

Les fonctions `valeur_absolue()`, `factorielle()` et `minimum()` ne sont pas fournies avec le programme (`fonctions1.c`) suivant. Compléter le programme avec le prototype et le code des fonctions (le `main` doit rester inchangé) et faire la trace du programme complet.

```c
int main()
{
    int x = -3;
    int y = 5;
    int z;

    /* Un calcul sans signification particulière */
    x = valeur_absolue(x); /* valeur absolue de x */
    z = minimum(x, y);     /* minimum entre x et y */
    z = factorielle(z);    /* z! */
    z = minimum(2, z);     /* minimum entre 2 et z */

    /* Valeur fonction */
    return EXIT_SUCCESS;
}
```

3 Écriture de fonctions

Pour les questions suivantes il faut donner la déclaration et la définition de chaque fonction. Vous pouvez faire l'exercice une première fois en donnant uniquement les déclarations, puis le reprendre pour les définitions. Répondre dans un seul programme, `fonctions2.c`, dans lequel vous écrirez une fonction principale (`main`) faisant appel à toutes les fonctions pour les tester.

1. Écrire la fonction `carre` qui prend en entrée un entier et qui renvoie le carré de cet entier.
2. Écrire la fonction `cube` qui prend en entrée un entier et qui renvoie le cube de cet entier.
3. Écrire la fonction `est_majeur` qui prend en entrée un entier représentant l'age en années d'une personne et renvoie `true` si cette personne est majeure et `false` sinon.
4. Écrire la fonction `somme` qui prend en entrée un entier n et qui renvoie $\sum_{i=1}^{i=n} i$. Où faut-il déclarer la variable de boucle ?
5. Si vous ne l'avez pas fait à l'exercice précédent, écrire la fonction `factorielle` qui prend en entrée un entier n et qui renvoie $\Pi_{i=1}^{i=n} i$.
6. Écrire la procédure `afficher_rectangle` qui prend en entrée deux entiers, largeur et hauteur, et affiche un rectangle d'étoiles de ces dimensions.
7. Écrire la fonction `saisie_utilisateur` sans argument, qui demande à l'utilisateur de saisir un nombre entier et le retourne.
8. Écrire la fonction `binomial` qui prend en entrée un entier n et un entier p et retourne le nombre de tirages différents, non ordonnés et sans remise, de p éléments parmi n, c'est à dire le coefficient binomial :

$$\binom{n}{p} = \mathrm{C}_n^p = \frac{n!}{p!(n-p)!}.$$

   Faire appel à la fonction `factorielle`.
9. Optionnel. Écrire la fonction `saisie_dans_intervalle` à deux paramètre entiers a et b, qui demande à l'utilisateur de saisir un nombre entier jusqu'à ce qu'il soit dans l'intervalle $[a, b]$ et le retourne. On pourra fixer un nombre maximum de tentatives après quoi le nombre de l'intervalle le plus proche de la saisie utilisateur sera renvoyé.

```c
#include <stdlib.h>
#include <stdio.h>

void afficher_triangle(int cote);

int main() {
    int i; /* var de boucle */

    /* testons différentes tailles de triangles */
    for (i = 2; i <= 10; i += 2) {
        printf("\n\nafficher_triangle(%d)\n", i);
        afficher_triangle(i);
    }
    return EXIT_SUCCESS;
}

void afficher_triangle(int cote) {
    int i; /* var de boucle */

    /* il faudrait deja arriver a faire un carre ! */
    for (i = 0; i < cote; i += 1) {/* pour chaque ligne */
        /* afficher une ligne de cote etoiles. Faire une sous-procedure ? */
    }
}
```

FIGURE 1 – Programme de Pippo pour mettre au point la fonction afficher triangle

4 Programmer avec des fonctions et procédures

Dans les exercices suivants, l'objectif est de mettre au point des programmes entiers en y apportant vos propres fonctions et procédures. La méthode pour y arriver : **découper le programmes en tâches simples à mettre en œuvre et progresser par étapes en testant régulièrement votre code, même s'il est incomplet**.

Pour chaque exercice, vous pouvez faire plus ou moins de fonctions et procédures selon si vous découpez le travail en de plus ou moins gros morceaux. Lorsqu'on découpe une tâche en beaucoup de morceaux on parle de *code ravioli*. Par exemple, pour afficher des motifs d'étoiles vous pouvez déclarer et définir les procédures suivantes :

```c
void afficher_ligne(int nb_etoiles); /* afficher_ligne(5) affiche "*****\n" */
void afficher_etoile();              /* affiche "*" */
void afficher_espace();              /* affiche " " */
void finir_ligne();                  /* affiche "\n" */
```

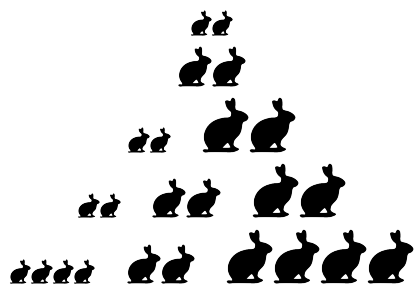
Lorsque c'est possible, exercez vous à produire du code ravioli en rajoutant des fonctions et procédures (par exemple, utiliser `afficher_ligne(n)` au lieu d'écrire directement une boucle for pour afficher n étoiles et un retour à la ligne). Puis, réciproquement, à supprimer les sous-tâches qui vous paraissent exagérées (par exemple, remplacer un appel à `afficher_espace()` par un appel direct à `printf(" ")`.)

4.1 Encore un triangle d'étoiles

Pippo a commencé à écrire un procédure qui permet d'afficher un triangle d'étoiles. Comme c'est un peu compliqué, il progresse par étapes et fait en sorte de pouvoir tester son code le plus tôt possible, même s'il n'est pas terminé. Compléter son programme `triangle3.c`, figure 1.

4.2 Population de lapins

Un couple de lapins a sa première portée à deux mois, puis une portée tous les mois. Chaque portée est un couple de lapins. Tous les couples ainsi obtenus se reproduisent de la même manière. [1]



1er septembre (deux jeunes lapins)

1er octobre (deux lapins d'un mois)

1er novembre (deux vieux et leurs deux petits)

1er décembre

1er janvier

1. On distinguera le nombre de couples de nouveaux lapins `nouveaux`, le nombre de couples de lapins ayant un mois, `un_mois`, et le nombre de couples de lapins ayant 2 mois ou plus, `vieux`. Calculer (à la main) le nombre de couples de lapins de chaque type, ainsi que leur nombre total, pour les 10 premiers mois.
2. Écrire un algorithme `nb_lapins(nb_mois, nb_couples)` calculant le nombre de lapins obtenus au bout de `nb_mois` mois à partir de `nb_couples` couples jeunes, et renvoyant le résultat.
3. À combien s'élève la population au bout de 24 mois ?
4. Écrire un algorithme `lapins_un_milliard` calculant au bout de combien de temps les lapins sont plus d'un milliard (on supposera qu'aucun lapin ne meurt pendant cette période), en partant d'un couple de jeunes lapins.
5. Mettre en œuvre ces algorithmes dans un programme C, `lapins.c`.

Les lapins disparaissent ! Cinq couples de jeunes lapins se sont installés sur une île déserte et ont fondé une communauté. Les premiers mois tout se passe normalement, mais à partir du septième mois, des lapins disparaissent après avoir donné naissance à leurs dernières portées : exactement un cinquième des vieux lapins, quatre cinquièmes des lapins venant d'avoir deux mois et neuf dixièmes des nouveaux lapins survivent (par arrondi inférieur), tous les autres disparaissent. Ceci tous les mois, sans que la cause ne soit connue (certains scientifiques lapins mettent en cause la nourriture transgénique ou les antibiotiques large spectre, d'autres lapins évoquent une famille de renards, et d'autres encore pensent que les services secrets humains ont découvert le plan d'invasion de la Terre par les lapins et ont pris des contre-mesures).

6. Tenir compte de ces nouvelles données dans la simulation. Que se passe t'il ?
7. Afficher la population de lapin obtenue à chaque nouveau mois sous forme graphique sur une ligne. Par exemple, un couple de nouveaux lapins sera affiché par le caractère point `"."`, un couple de lapins d'un mois par le caractère `"o"`, un vieux couple de lapins par le caractère `"#"`.
8. Combien de lapins sont nés sur l'île ?

1. Merci à Laure Petrucci pour cet exercice et à Lionel Allorge pour le dessin de lapin en cc-by-sa.