Licence 1 — Université de Nantes
X2I0010 Algorithmique et programmation — 2016/2017

Travaux dirigés n^{o} 2
Sous-algorithmes

(4 séances)

Exercice 2.1 (Simulation)

Simuler l'exécution des algorithmes suivants.

1. suite de Syracuse

```
fonction syracuse(d entier n):entier
debut
   si (n MOD 2 = 0) alors
      retourner(n DIV 2)
   sinon
      retourner(3*n + 1)
   fin si
fin
```

```
entier nb, temps
debut
   nb ← 4
   temps ← 0
   tant que (nb > 1) faire
      nb ← syracuse(nb)
      temps ← temps + 1
      ecrire (nb)
   fin tant que
   ecrire ("temps : ", temps)
fin
```

2.

```
// Role : choix de la langue, "francais" par defaut
fonction choix_langue() : chaine de caracteres
caractere choix
debut
   ecrire ("1 : francais")
   ecrire ("2 : english")
   ecrire ("3 : Deutsch")
   lire(choix)
   selon choix dans
      '1' : retourner ("fre")
      '2' : retourner ("eng")
      '3' : retourner ("deu")
      'defaut' : retourner ("fre")
   fin selon
fin

// Role : saisie d'un nombre
fonction saisie_nombre(d chaine lang) : entier
entier nombre
debut
   selon langue dans
      "fre" : ecrire ("un nombre positif ? (0 pour arreter)")
      "eng" : ecrire ("a positive number ? (0 to stop)")
      "deu" : ecrire ("eine positive Zahl? (0 zu stoppen)")
   fin selon
   lire(nombre)
   retourner(nombre)
fin
```

```
// Role : saisie d'une serie de valeurs entieres , retourne la valeur maximale
//          et le nombre de valeurs positives saisies
procedure saisie_serie(d chaine lang , m max entier , m cpt entier)
entiers nb
debut
  cpt ← 0
  nb ← saisie_nombre(lang)
  max ← nb
  tant que  (nb > 0) faire
     si (nb > max) alors
        max  ← nb
     fin si
     cpt ← cpt + 1
     nb  ← saisie_nombre(lang)
  fin tant que
fin

chaine langue
entier nombre , maximum
debut
  langue ← choix_langue()
  saisie_serie(langue , maximum, nombre)
  ecrire(nombre, " valeurs ont ete saisis. Le maximum est : ", maximum)
fin
```

Exercice 2.2 (Dessin)

Il s'agit de dessiner une maison comme un triangle sans base au-dessus d'un carré.
- — Découper le problème en sous-problèmes. Spécifier les sous-algorithmes correspondant aux sous-problèmes.
- — Pour chaque sous-algorithme, faire l'analyse puis écrire son algorithme

```
  *
 / \
/   \
+---+
|   |
|   |
|   |
+---+
```

Exercice 2.3 (Saisies contrôlées)

1. Il s'agit d'écrire une procédure qui force l'utilisateur à saisir un entier positif.
   - — Spécifier la procédure.
   - — Faire l'analyse.
   - — Ecrire son algorithme.
   - — Ecrire un algorithme qui appelle la procédure de saisie d'un entier positif et qui affiche la valeur saisie.
   - — Si c'est possible, transformer la procédure en fonction, modifier l'algorithme appelant en conséquence.
2. Adapter les sous-algorithmes déjà écrits à la question 1 afin de forcer l'utilisateur à saisir un entier compris dans l'intervalle d'entiers $[a, b]$.

Exercice 2.4 (Géométrie du plan)

On considère les points du plan représentés par leurs coordonnées cartésiennes x et y. Une translation d'un point se définit par une quantité δ_x à ajouter à x et une quantité δ_y à ajouter à y. .

1. Représenter un point du plan à l'aide d'un enregistrement.
2. Spécifier trois procédures dont les rôles sont, respectivement de
   - — afficher un point,
   - — faire saisir les coordonnées d'un point,
   - — translater un point,
3. Effectuer l'analyse pour chaque procédure puis écrire son algorithme.

4. Ecrire un algorithme principal demandant à l'utilisateur un point de départ, puis lui proposant répétitivement deux choix :
   — appliquer une translation ; l'utilisateur fournit les quantités et l'algorithme affiche le nouveau point.
   — terminer ; l'algorithme affiche le dernier point obtenu et se termine.

Exercice 2.5 (Distance)

Un algorithme demande à l'utilisateur une série de points (coordonnées réelles x et y) et affiche la longueur totale de la polyligne ainsi décrite.

1. Représenter un point du plan à l'aide d'un enregistrement.
2. Effectuer l'analyse pour écrire cet algorithme. Si nécessaire, vous spécifierez des sous-algorithmes. Vous disposez d'une fonction nommée `sqrt` prenant en paramètre un réel positif et retournant sa racine carrée.
3. Ecrire les algorithmes.

Exercice 2.6 (Saisie contrôlée d'un entier quelconque)

Même si l'utilisateur doit saisir un entier, il arrive qu'il ne respecte pas les consignes et saisisse des caractères non autorisés ce qui provoque généralement un arrêt brutal de l'exécution du programme.

1. Soit la procédure $chaine\_en\_entier$(d chaine ch, m entier val, m booleen $reussi$) qui convertit une chaîne ch en un entier val si c'est possible, dans ce cas $reussi$ vaut $vrai$; si la conversion n'est pas possible $reussi$ prend la valeur $faux$.

   Exemples : L'exécution de $chaine\_en\_entier("XR3", val, reussi)$ donne la valeur $faux$ à $reussi$. L'exécution de $chaine\_en\_entier("1984", val, reussi)$ donne la valeur $vrai$ à $reussi$ et val a pour valeur 1984.
   — Effectuer l'analyse de la procédure $chaine\_en\_entier$.
   — En déduire des jeux d'essais.
   — Ecrire son algorithme.
2. Il s'agit de réaliser un sous-algorithme qui force l'utilisateur à saisir une chaîne de caractères composée de chiffres puis qui convertit cette chaîne de caractères en un entier.
   — Spécifier le sous-algorithme.
   — Effectuer l'analyse. Celle-ci fera éventuellement émerger d'autres sous-algorithmes qu'il faudra également spécifier.
   — Ecrire les algorithmes.

Exercice 2.7 (Prems')

Algorithme qui demande à l'utilisateur un entier et affiche le plus petit nombre premier directement supérieur ou égal à l'entier saisi.

1. Effectuer l'analyse. Si nécessaire, spécifier des sous-algorithmes pour lesquels vous effectuerez également une analyse.
2. Ecrire les algorithmes.

Exercice 2.8 (Calcul de suite)

Soit la suite mathématique $u_n = 2 * u_{n-1} + 1$, avec $u_0 = 1$. Calculer le n^{ieme} terme de cette suite peut se faire récursivement au moyen de la fonction suivante :

```
fonction suite(d entier n) : entier
debut
  si n = 0 alors
     retourner 1
  sinon
     retourner 2*suite(n-1) + 1
  fin si
fin
```

1. Identifier le cas de base et de récurrence dans cet algorithme, puis simuler son appel pour $n = 3$.

2. Écrire un algorithme récursif qui calcule le n^{ieme} élément de la suite $u_n = 3 * u_{n-1} + 2 * v_{n-1}$ avec $v_n = 2 * u_{n-1} + 3 * v_{n-1}$, $u_0 = 1$ et $v_0 = 2$. Le simuler pour calculer u_2. Déterminer combien de fois chaque terme intermédiaire des suites u et v est calculé pour obtenir le n^{ieme} terme de la suite u.

Exercice 2.9 (Procédure récursive)

Écrire deux versions d'une procédure affichant la table de multiplication de 1 à 10 d'un entier saisi par l'utilisateur : une version itérative puis une version récursive. À chaque fois, effectuer une analyse préalable.

Exercice 2.10 (Nombre de chiffres d'un nombre)

Trouvez l'algorithme récursif donnant le nombre de chiffres d'un nombre entier positif. Exemples : 1434 possède quatre chiffres ; 0 et 3 en possèdent un seul.

Exercice 2.11 (Palindrome)

Écrire une fonction récursive qui décide si une chaîne de caractères est un palindrome.

Exercice 2.12 (Algorithme d'Euclide)

L'algorithme d'Euclide est un algorithme récursif très efficace qui permet de trouver le plus grand diviseur commun $\mathrm{pgcd}(x, y)$ à deux nombres entiers positifs x et y. Il s'obtient en remarquant que si le nombre a est un diviseur commun à x et y alors il divise aussi le reste de la division de x par y. Écrire un algorithme récursif qui calcule $\mathrm{pgcd}(x, y)$ (vous pouvez supposer que $x \geq y$).

Exercice 2.13 (Série)

Soit la suite mathématique $u_n = 2 * u_{n-1} + 1$, avec $u_0 = 1$ déjà rencontrée lors de l'exercice 2.11.
En vous inspirant de cette fonction, écrire une procédure récursive qui calcule et affiche le k^{ime} terme de la série $S_u(k) = \Sigma_{n=0}^{n=k} u_n$
Simuler l'appel de cette procédure pour $k = 3$.

Exercice 2.14 (temps)

Pour toutes les écritures d'algorithme, une analyse préalable est, évidemment, indispensable.

1. Définir un type T_date (avec trois entiers : $jour$, $mois$, $annee$) à l'aide d'un enregistrement.
2. Définir un sous-algorithme $nb\_jours\_dans\_annee$ qui retourne le nombre de jours d'une année (365 ou 366 cf. TD1).
3. Définir un sous-algorithme $lendemain$ qui donne le lendemain d'une date (définir des sous-algorithmes supplémentaires si nécessaire).
4. Ecrire un algorithme qui affiche le nombre de jours séparant deux dates en utilisant le sous-algorithme $lendemain$.
5. Ecrire une nouvelle version, plus rapide, de l'algorithme précédent.