Université Paris-Nord — L1
Institut Galilée — Année 2008-2009
Licence 1 - 1er semestre

# Partiel d'éléments d'informatique

**Durée :** 3 heures.

**Documents autorisés :** Aucun.

**Recommandations :** Un barême vous est donné à titre indicatif afin de vous permettre de gérer votre temps. La notation prendra en compte à la fois la syntaxe et la sémantique de vos programmes, c'est-à-dire qu'ils doivent compiler correctement. Une fois votre programme écrit, il est recommandé de le faire tourner à la main sur un exemple pour s'assurer de sa correction.

## 1 Trace d'un programme (*4 points*)

Le programme suivant calcul les `N` premiers termes d'une certaine suite. En faire la trace. *Attention : ici* `N` *vaut 1.*

```
1   #include <stdlib.h>             /* EXIT_SUCCESS */
2   #include <stdio.h>
3
4   #define N 1
5
6   int terme_suivant (int u);
7   int choix_utilisateur ();
8   void afficher_terme (int i, int u);
9
10  int
11  main ()
12  {
13      int t[N];
14      int u;
15      int i;
16
17      u = choix_utilisateur ();
18
19      for (i = 0; i < N; i = i + 1)
20      {
21          t[i] = u;
22          afficher_terme (i, u);
23          u = terme_suivant (u);
24      }
25
26      return EXIT_SUCCESS;
```

```
27  }
28
29  int
30  terme_suivant (int u)
31  {
32      int res;
33
34      if (u % 2 == 0)             /* si u est pair */
35      {
36          res = u / 2;
37      }
38      else
39      {
40          res = 3 * u + 1;
41      }
42      return res;
43  }
44
45  int
46  choix_utilisateur ()
47  {
48      int res;
49
50      printf ("Donner le premier terme : ");
51      scanf ("%d", &res);
52      return res;
53  }
54
55  void
56  afficher_terme (int i, int u)
57  {
58      printf ("terme %d = %d\n", i, u);
59  }
```

# 2 Exercices sur des tableaux, sans fonctions (*5 points*)

Pour les deux exercices suivants, tout le traitement sera effectué dans le `main`, sans faire appel à des fonctions utilisateurs.

## 2.1 Somme de deux vecteurs (*1,5 points*)

Rappel : la somme de deux vecteurs du plan se fait terme à terme comme ceci :

$$(x, y) + (x', y') = (x + x', y + y').$$

Nous voulons faire un programme qui réalise la somme de deux vecteurs de dimension $N$. Un vecteur sera représenté par un tableau.

Écrire un programme qui, étant donnés deux tableaux de réels, $u$ et $v$, de tailles $N$, calcule dans un tableau $w$ de taille $N$ la somme terme à terme de $u$ et de $v$, puis affiche le contenu de

$w$. Définir `N` comme une constante symbolique valant 2. Les tableaux $u$ et $v$ seront initialisés à des valeurs de votre choix.

*Tout le traitement sera effectué dans le* `main`*, sans faire appel à des fonctions utilisateurs.*

### 2.2 Unicité des éléments d'un tableau (*3,5 points*)

Nous disposons d'un tableau $t$ de $N$ caractères et nous souhaitons savoir si chaque caractère apparaissant dans le tableau n'y apparaît qu'une seule fois, autrement dit on veut savoir si chaque caractère est unique.

1. Écrire un programme qui, étant donné un tableau initalisé $t$, teste si le premier élément du tableau est unique et affiche `Vrai` si c'est le cas, `Faux` sinon.
2. Écrire un programme qui étant donné un tableau initialisé $t$, teste si tous les éléments sont uniques et affiche `Vrai` si c'est le cas, `Faux` sinon.

*Tout le traitement sera effectué dans le* `main`*, sans faire appel à des fonctions utilisateurs.*

## 3 Conjecture de Syracuse (*5 points*)

Une suite de Syracuse est définie de la manière suivante. 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 suite finit toujours par atteindre 1 (mais personne ne sait le démontrer !) auquel cas on s'arrête car à partir de 1 la suite devient périodique (1, 4, 2, 1, ...). Pour un premier terme fixé, le nombre d'itérations nécessaires à atteindre pour la première fois la valeur un est appelé *temps de vol* et la valeur maximale prise par la suite est appelée *altitude maximale*.

La fonction `terme_suivant` du premier exercice calcule le terme suivant à partir du terme courant.

1. 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 l'altitude maximale et le temps de vol de la suite obtenue (le temps de vol est égal à 0 si le premier terme vaut 1). Utiliser (sans rappeler sa déclaration et sa définition) la fonction `terme_suivant` pour calculer le terme suivant. *Répondre en faisant bien apparaître d'une part la déclaration, d'autre part la définition.*
2. Écrire un `main` qui exécute tour à tour la procédure `syracuse` sur les entiers 2 à $n$, pour $n$ choisi par l'utilisateur. La saisie sera réalisée dans une fonction à declarer et définir (inutile de vérifier que l'utilisateur saisit bien un entier $n \geq 2$).

Exemple d'affichage (l'utilisateur saisit 7 à la première ligne) :

```
Tester de 2 a 7
Depart : 2, altitude maximale : 2, temps de vol : 1
Depart : 3, altitude maximale : 16, temps de vol : 7
Depart : 4, altitude maximale : 4, temps de vol : 2
Depart : 5, altitude maximale : 16, temps de vol : 5
Depart : 6, altitude maximale : 16, temps de vol : 8
Depart : 7, altitude maximale : 52, temps de vol : 16
```

# 4 Nombres rationnels (*6 points*)

Dans cet exercice, vous devrez écrire un programme calculant sur les nombres rationnels. Un nombre rationnel est toujours donné par une fraction $\frac{p}{q}$, définie par deux entiers $p$ et $q$. Le nombre $q$ est nécessairement non nul et sera toujours pris positif, le nombre $p$ peut être négatif ou nul.

1. Définir un type utilisateur pour les fractions.
2. Déclarer et définir une fonction `multiplier_fractions` qui prend deux fractions en argument et renvoie le produit des deux fractions (ne pas chercher à simplifier la fraction obtenue). *Répondre en faisant bien apparaître d'une part la déclaration, d'autre part la définition.*
3. Même question pour la somme de deux fractions `additionner_fractions`.
4. Déclarer et définir une procédure affichant une fraction rationnelle comme dans l'exemple suivant où les fractions $\frac{34}{26}$, $\frac{34}{26}$, $-\frac{34}{1}$, $\frac{0}{1}$ sont affichées tour à tour (quatre affichages, sans saut de ligne, séparés par des espaces) :

   ```
   34/26 -34/26 34 0
   ```

La représentation d'un nombre rationnel sous la forme d'une fraction $\frac{p}{q}$ est normalisée lorsque $p$ et $q$ sont premiers entre eux : c'est à dire lorsque le PGCD (plus grand commun diviseur) de $p$ et $q$ vaut 1 (autrement il faut *simplifier* par le PGCD). Pour la suite de l'exercice, on supposera donnée une fonction `int pgcd(int a, int b)` qui calcule et renvoie le PGCD de deux entiers **positifs** $a$ et $b$.

5. Déclarer et définir une fonction `normaliser_fraction` qui met une fraction sous forme normale.

**Correction.**

```
/* Declaration */
struct fraction_s normaliser_fraction(struct fraction_s x);
...
/* Definition */
struct fraction_s normaliser_fraction(struct fraction_s x)
{
  int d; /* pgcd */
  struct fraction_s res; /* fraction normalisee */
  /* Calcul du pgcd */
  if (x.p < 0)
  {
     d = pgcd(-x.p, x.q);
  }
  else
  {
     d = pgcd(x.p, x.q);
  }
  /* Simplification */
  res.p = x.p / d;
  res.q = x.q / d;
```

```
    return res;
}
```

6. Écrire un `main` qui réalise la somme $\frac{1}{6} + \frac{1}{3}$, normalise le résultat et l'affiche, à l'aide des fonctions précédentes.

**Correction.**

```
int main()
{
   struct fraction_s x = {1, 6};
   struct fraction_s y = {1, 3};
   struct fraction_s res;

   res = additionner_fractions(x, y);
   res = normaliser_fraction(res);
   printf("Resultat : ")
   afficher_fraction(res);
   printf("\n");

   return EXIT_SUCCESS;
}
```

## Question bonus

Définir la fonction `int pgcd(int a, int b)` utilisée dans l'exercice précédent.