TD 1

Exercice 1 Le programme suivant calcule le PGCD de deux entiers strictement positifs :

```
int pgcd(int a, int b){
   int x = a ; int y = b ;
   while (x != y)
      if (x > y)
         x = x-y ;
      else
      y = y-x ;
   return x ;
}
```

Montrer que l'algorithme `pgcd` termine et qu'il est correct, *i.e.* `pgcd(`n`,` m`)` calcule bien PGCD de n et de m.

Montrer que l'algorithme (d'Euclide) suivant calcule aussi le PGCD de deux entiers :

```
int euclide(int a, int b){
   int r ; int x = a ; int y = b ;
   while (y != 0){
      r = x%y ;
      x = y ;
      y = r ;
   }
   return x ;
}
```

Exercice 2 La suite de Fibonacci définie par :

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_n = u_{n-2} + u_{n-1}, \text{ pour } n > 1 \end{cases}$$

est une suite linéaire récurrente d'ordre 2. On peut calculer chacun de ses termes en temps constant, en calculant la racine de l'équation caractéristique associée $r^2 - r - 1 = 0$, mais considérons ici un calcul moins sophistiqué pour chaque term u_n.

```
int fib(int n){
   int t ; int x = 0 ; int y = 1 ; int i = 2 ;

   if (n == 0)
      return x ;

   if (n == 1)
      return y ;

   while (i <= n){
      t = x ;
      x = y ;
      y = y + t ;
      ++i ;
   }
   return y ;
}
```

Montrer que l'algorithme `fib` termine et qu'il est correct, *i.e.* `fib(`n`)` calcule bien le terme u_n.

Exercice 3 Le programme suivant calcule la factotielle d'un entier :

```
int fact(int n){
   int r ;
   if (n == 0)
      r = 1 ;
   else
      r = n*fact(n-1) ;
   return r ;
}
```

Montrer que l'algorithme `fact` termine et qu'il est correct, *i.e.* `fact(`n`)` calcule bien $n!$.

Exercice 4 Prouver la terminaison de la fonction d'Ackermann implantée par :

```
int ack(int m, int n){
   if (m == 0)
      return n+1 ;
   else
      if (n == 0)
         return ack(m-1, 1) ;
      else
         return ack(m-1, ack(m, n-1)) ;
}
```

Exercice 5 Prouver la terminaison de la fonction 91 de McCarthy implantée par :

```
int f91(int x){
   if (x > 100)
      return r-10 ;
   else
      return f(f(x+11)) ;
}
```

Exercice 6 Soit l'algorithme suivant du tri fusion :

```
procédure tri_fusion(t : tableau[1..n] de réels ; g, d : entiers)
début
   si g < d
   alors
      k <- ((g+d) div 2)
      tri_fusion(t, g, k)
      tri_fusion(t, k+1, d)
      fusion(t, g, k, d)
   finsi
fin
```

1. Donner l'algorithme de la procédure de fusion :

   `procédure fusion(tableau[1..n] de réels ; g, k, d : entiers)`

   Cette procédure permute trie les éléments de `tableau[g..d]`, sachant que `tableau[g..k]` et `tableau[k..d]` sont déjà triés.

2. Montrer que votre procédure `fusion` termine.

3. Montrer que votre procédure `fusion` est correcte.

4. Montrer la correction ainsi que la terminaison de `tri_fusion`.

Exercice 7 Soit l'algorithme suivant du tri par sélection :

```
procédure tri_sélection(t : tableau[1..n] de réels ; g, d : entiers)
début
   si d > g
   alors
      m <- min_index(t, g, d)
      échanger(t, m, g)
      tri_selection(t, g+1, d)
fin
```

1. Ecrire l'algorithme de la fonction `min_index` :

`procédure min_index(tableau[1..n] de réels ; g, d : entiers)`
qui retourne l'indice du plus petit élément de `t[g..d]`.

2. Montrer que votre fonction `min_index` est correcte et termine.

3. Montrer la correction ainsi que la terminaison de `tri_sélection`.

Exercice 8 L'algorithme du tri rapide est le suivant :

```
procédure partition (t : tableau[1..n] de réels ; i, j : entiers)
début
   échanger(t, i, (i+j) div 2) // mieux si le tableau est déjà trié
   pivot <- t[i] ; g <- i+1 ; d <-j

   tantque g < d faire
      tantque g <= j et t[g] <= pivot faire
         g++
      fintantque

      tantque d >= i et t[d] > pivot faire
         d--
      fintantque

      si g < d
      alors
         échanger(t, g, d)
      finsi
   fintantque

   si t[d] > pivot
   alors
      d--
   finsi
   échanger(t, i, d)
   retourner d
fin

procédure tri_rapide (t : tableau[1..n] de réels ; g, d : entiers)
début
   si g < d
   alors
      k <- partition(t, g, d)
      tri_rapide(t, g, k-1)
      tri_rapide(t, k+1, d)
fin
```

1. Montrer que la procédure `partition` est correcte et termine.

2. Montrer que la procédure `tri_rapide` est correcte et termine.