Devoir en classe d'Informatique I,1

50 minutes – 30 points

Exercice 1

1.

a. Présentez l'algorithme d'*Euclide par soustraction* (code sans explication). [6 p.]

```
function pgcd(a, b : integer) : integer;
begin
  while (a <> b) do
    if (a > b) then
      a := a-b
    else
      b := b-a;
  result := a;
end;
```

b. Donnez un exemple d'exécution pour `pgcd(12, 66)`. [3 p.]

	a	b	result
initialisation	12	66	?
Après 1. itération	12	54	?
Après 2. itération	12	42	?
Après 3. itération	12	30	?
Après 4. itération	12	18	?
Après 5. itération	12	6	?
Après 6. itération	6	6	?
résultat	6	6	6

2.

a. Présentez l'algorithme d'*Euclide par division* (code sans explication). [8 p.]

```
function pgcd(a, b : integer) : integer;
var
  tmp : integer;
begin
  if (b > a) then
    begin
      tmp := a;
      a := b;
      b := tmp;
    end;
  while (b > 0) do
    begin
      tmp := a mod b;
      a := b;
      b := tmp;
    end;
  result := a;
end;
```

b. Donnez un exemple d'exécution pour `pgcd(12, 66)`. [3 p.]

	a	b	result
initialisation	12	66	?
échange	66	12	?
Après 1. itération	12	6	?
Après 2. itération	6	0	?
résultat	6	0	6

3. Expliquez, à l'aide des deux exemples d'exécution, pourquoi l'algorithme d'*Euclide par division* est plus rapide que l'algorithme d'*Euclide par soustraction*. [2 p.]

*L'algorithme d'Euclide par division est plus rapide que l'algorithme d'Euclide par soustraction parce que l'opération modulo fait que les deux nombres se rapprochent beaucoup plus vite qu'en utilisant l'opération soustraction. Comme le pgcd de deux nombres « a » et « b » est plus petit ou égal au plus petit des deux nombres, il faut d'abord ramener le plus grand des deux nombres à une valeur inférieure au plus petit des deux. En effet, il nous faut 5 itérations (car 66 = 5*12 + 6) dans l'algorithme d'Euclide par soustraction pour arriver à « 12 et 6 », alors que l'algorithme d'Euclide par division le fait en une seule itération (car 66 mod 12 = 6).*

Exercice 2

On donne la procédure définie par :

```
procedure exem (VAR a, b : integer);
var
  i : integer;
begin
  i := 1;
  while (b > 0) AND (b < 20) do
    begin
      a := a + b + i;
      b := b - 1;
      i := i + 1
    end
end;
```

Expliquez en détail (par exemple avec un tableau d'exécution) ce que les instructions suivantes vont afficher (x et y sont des variables de type *integer*) : [8 p.]

a) `x := 2; y := 0; exem(x, y); writeln (x, '-', y);`

	a	b	I
initialisation	2	0	1

Explication: Les variables « a » et « b » sont initialisées avec les valeurs de « x » et « y », alors que la variable « i » est initialisée avec 1. La condition de while n'est pas remplie, le programme n'entre même pas dans la boucle. Les valeurs sont donc inchangées. Le programme va donc afficher : « 2-0 ».

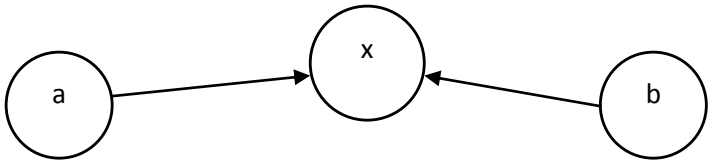
b) `x := 2; y := 1; exem(x, y); writeln (x, '-', y);`

	a	b	I
initialisation	2	1	1
1. itération	4	0	2

Explication: Les variables « a » et « b » sont initialisées avec les valeurs de « x » et « y », alors que la variable « i » est initialisée avec 1. La boucle est exécutée exactement une seule fois et la valeur des variables « a » et « b », respectivement de « x » et « y » sont « 4 » et « 0 ». Le programme va donc afficher : « 4-0 ».

c) `x := 2; y := 0; exem(x, x); writeln (x, '-', y);`

Explications préalables: Le passage par référence crée un lien étroit entre la variable référencée et la variable à l'intérieur de la fonction. Comme vous le savez, chaque variable utilise un emplacement dans la mémoire de l'ordinateur, dans lequel elle stocke ses valeurs. Or, si on utilise le passage par référence, la variable à l'intérieur de la fonction n'a pas d'emplacement propre. Elle est seulement un « alias », une « référence » à l'emplacement de la variable passée en argument. Donc, dans l'exemple suivant, les deux variables ou « références », pointent vers le même emplacement en mémoire. Si on assigne donc une nouvelle valeur à la variable « a », elle met à jour l'emplacement de la variable « x » et, en conséquence, aussi la valeur de la variable « b ». On peut s'imaginer la représentation suivante :



	a	b	i
initialisation	2	2	1
Après l'assignement à « a »	5	5	1
Après l'assignement à « b »	4	4	1
A la fin de la 1. itération	4	4	2
2. itération : après l'assignement de « a »	10	10	2
2. itération : après l'assignement de « b »	9	9	2
A la fin de la 2. itération	9	9	3
3. itération	20	20	4

Explication: Le programme va donc afficher : « 20-0 ».