Corrigé-type de l'EMD2.
PSDD. le 12/06/2017

Exercice n° 1: (05 pts)

```
algorithme ex_01;
type hotel = enregistrement
        id:    chaine[10];
        nom:   chaine[30];
        ville: chaine[50];
        nbre:  entier;
        nbdis: entier;
        prix : reel;
     fin;
thote = tableau [1..100] de hotel;
Var   T: thote ; n : entier ;
```

(0,25) (0,25)

1/ Procedure remplir (Var T: thote, n: entier);
```
Var i: entier;
debut
   pour i ← 1 à n faire
      avec T[i] faire
         lire (id, nom, ville, nbre, nbdis, prix);
      favec;
   fpour;
fin;
```
(0,75)

2/ Procedure Reserver (Var T: thote; n: entier; idh: chaine[10])
```
Var i : entier ; trouve: booleen ;
debut
   i ← 1; trouve ← faux;
   tque (i ≤ n) et non trouve faire
      si (T[i].id = idh) et (T[i].nbdis ≠ 0)
      alors
         trouve ← vrai;
      Sinon
         i ← i + 1;
      fsi;
   ftque;
   si (trouve = vrai) alors T[i].nbdis ← T[i].nbdis - 1;
   Sinon
      ecrire (' chambres non disponible');
```
(0,25)

3/ Procedure afficher1 (T: thote, n: entier);
```
Var i : entier;
debut
   pour i ← 1 à n faire
      si ((T[i].ville = 'Bejaia') et
          (T[i].nbdis ≠ 0)) alors
         ecrire (T[i].id, ..... );
      fsi;
   fpour;
fin;
```
(0,75)

4/ fonction retourner (T: thote; n: entier): chaine[10]
```
Var hote : hotel; i : entier ;
debut
   i ← 1; hote ← T[1]; {initialisation}
   pour i ← 2 à n faire
      si (T[i].prix < hote.prix) alors
         hote ← i;
      fsi;
   fpour;
   Retourner ← hote.prix;
fin;
```
(1)

5/ Procedure afficher2 (T: thote; n: entier);
```
Var i : entier;
debut
   pour i ← 1 à n faire
      si (T[i].nbre ≥ 2) et (T[i].prix = 15000)
      alors
         ecrire (T[i].id, ..... );
      fsi;
   fpour;
fin;
```
(0,75)

Exercice n°2 (3pts)

1/ Procedure nbr_occ (car: caractère, Var p: pile; Var nb: entier);

```
Var p1 : pile;
debut
    nb ← 0; vider_pile (p1);
    tque non pile_vide (p) faire
        si (sommet (p) = car) alors nb ← nb+1;
        Sinon
            empiler (p1, sommet (p));
        fsi;
        depiler (p);
    ftque;
    tque non pile_vide (p1) faire
        empiler (p, sommet (p1));
        depiler (p1);
    ftque;
fin;
```

(1,5)

2/ Procedure Cmcat (p: pile; Var ch: chaine);

```
Var p1 : pile;
debut
    vider_pile (p1); ch ← '';
    tque non pile_vide (p) faire
        empiler (p1, sommet (p));
        depiler (p);
    ftque;
    tque (non pile_vide (p1)) faire
        ch ← ch + sommet (p1);
        empiler (p, sommet (p1));
        depiler (p1);
    ftque;
fin;
```

(1,5)

Exercice n°3: (3pts)

```
fonction pgcd (a, b : entier) : entier;
debut
    Si (a mod b = 0) alors pgcd ← b;
    Sinon
        Si (b mod a = 0) alors pgcd ← a;
        Sinon
            Si (a > b) alors pgcd ← pgcd (a mod b, b)
            Sinon
                pgcd ← pgcd (a, b mod a);
            fsi;
        fsi;
    fsi;
fin;
```

(0,5) (0,5) (0,5) (0,75) (0,75)

Exercice n°4: (09 points)

```
algorithme exo_04;
type liste = ↑ noeud;
     noeud = enregistrement
         info : entier;
         suivant: liste;
     fin;
Var
```

(0,25)

% fonction à utiliser pour les 6p demandés

```
fonction nbnoeud (nbre : entier) : entier;
Var d : entier;
debut
    nbnoeud ← 1;
    Si (nbre ≥ 10) alors
        d ← 10;
        tque (nbre div d > 0) faire
            d ← d * 10;
            nbnoeud ← nbnoeud + 1;
        ftque
    fsi;
fin.
```

(0,75)

P02

```
fonction puis10 ( nbre : entier ) : entier ;
Var i : entier ;
debut
    puis10 ← 1 ;
    pour i ← 1 à nbre faire
        puis10 ← puis10 * 10 ;
    fpour ;
fin ;
```

(0,5)

3/
```
fonction Compter ( L : liste ) : entier ;
Var p : liste ;
debut
    Compter ← 0 ; p ← L ;
    tque (p ≠ Nil) faire
        p ← p↑.suivant ;
        Compter ← Compter + 1 ;
    ftque ;
fin ;
```

(0,5)

1/
```
procedure creer (Var L : liste, nbre : entier) ;
Var N, x, puis : entier ; p : liste ;
debut
    N ← nbrechiffre (nbre) ;
    puis ← puis10 (N - 1) ;
    x ← nbre ; L ← Nil ;
    pour i ← 1 à N faire
        allouer (p) ;
        avec p↑ faire
            info ← x div puis ;
            suivant ← L ;
        favec ;
            L ← p ;
            x ← x mod puis ;
            puis ← puis div 10 ;
    fpour ;
fin ;
```

(1)

```
fonction reconstituer ( L : liste ) : entier ;
Var cp, puis : entier ; p : liste ;
debut
    cp ← compter ( L ) ;
    puis ← puis10 (cp - 1) ;
    reconstituer ← 0 ;
    p ← L ;
    tque (p ≠ Nil) faire
        reconstituer ← p↑.info * puis + reconstituer ;
        puis ← puis div 10 ;
        p ← p↑.suivant ;
    ftque ;
fin ;
```

(0,75)

2/
```
fonction Max ( L : liste ) : entier ;
Var L1, p, q : liste ;
debut
    L1 ← L ;
    p ← L1 ;
    repeter
        q ← q↑.suivant ;
        repeter
            si (p↑.info < q↑.info) alors
                p↑.info ← q↑.info ;
            fsi ;
            q ← q↑.suivant ;
        jusqu'à (q = Nil) ;
        p ← p↑.suivant ;
    jusqu'à (p↑.suivant = q) ;
Max ← reconstituer (L1) ;
fin ;
```

(1)

```
procedure Inverser (Var L : liste);
Var p, q : liste;
debut
    p ← L;
    p↑.suivant ← Nil;
    L ← L↑.suivant;
    q ← L;
    tque (L ≠ Nil) faire
        q↑.suivant ← p;
        p ← q;
        L ← L↑.suivant;
    ftque;
    q ← L;
    L ← p;
fin;
```

(0,75)

```
fonction puissance (a : entier ; L : liste) : entier;
Var i : entier ; nb : entier ;
debut
    Inverser (L);
    nb ← remettre (L);
    puissance ← 1;
    pour i ← 1 à a faire
        puissance ← puissance * nb;
    fpour;
fin;
```

(0,5)

```
procedure somme (L1, L2 : liste ; Var SL : liste);
Var pre : entier ; new, p, q : liste ; R : liste;
debut
    SL ← Nil;
    p ← L1, q ← L2;
    tque (p ≠ Nil) et (q ≠ Nil) faire
        allouer (new);
        si (p = L1) alors new↑.info ← (p↑.info + q↑.info) mod 10;
                          new↑.suivant ← Nil;
                          SL ← new;
        Sinon
            new↑.info ← ((p↑.info + q↑.info) + pre) mod 10;
            new↑.suivant ← Nil; R↑.suivant ← new;
        fsi;
        pre ← (p↑.info + q↑.info) div 10;
        R ← new; p ← p↑.suivant; q ← q↑.suivant;
    ftque
    si (p ≠ Nil) et (q = Nil) alors
        p1 ← p;
        tque (p1 ≠ Nil) faire
            allouer (new);
            new↑.info ← (p↑.info + pre) mod 10;
            new↑.suivant ← Nil;
            R↑.suivant ← new;
            R ← new;
            pre ← (p↑.info + pre) div 10;
            p1 ← p↑.suivant;
        ftque;
    Sinon
        si (p = Nil) et (q ≠ Nil) faire
            /* les mêmes etapes
        fsi;
    fsi;
    si (pre ≠ 0) alors allouer (new);
                       avec new↑ faire
                           info ← pre;
                           suivant ← Nil;
                       favec;
                       R↑.suivant ← new;
    fsi;
fin;
```

(1,75)

```
fonction Composer (L1, L2 : liste) : entier;
Var NL1, NL2 : entier; p, q : liste;
debut
    inverser (L1); inverser (L2);
    NL1 ← compter (L1); NL2 ← compter (L2);
    si (NL1 > NL2) alors composer ← 1;
    Sinon
        si (NL1 < NL2) alors composer ← -1;
        Sinon
            p ← L1; q ← L2;
            tque (p ≠ Nil) et (p↑.info = q↑.info) faire
                p ← p↑.suivant; q ← q↑.suivant;
            ftque;
            si (p = Nil) alors composer ← 0;
            Sinon si (p↑.info < q↑.info) alors composer ← -1;
                Sinon composer ← 1;
                fsi;
            fsi;
        fsi;
    fsi;
```

(1,25)

P04