CSC 3002 - Corrigé du Contrôle Final 1

Exercice d'algorithmique (sur 7 points)

Enoncé :

- **données** nb : Naturel
- **résultat** affichage dans la fenêtre *shell* des nombres unitairement parfaits inférieurs ou égaux à *nb*.

Algorithme :

Définition de la fonction *principale* :

```
fonction principale ( ) : CodeRetour

    nb : Naturel

    afficher ( "Saisir un nombre supérieur à 1 : " )
    saisir ( nb )

    si nb < 1 alors
        afficher ( "La saisie est incorrecte" )
        allerAlaLigne ( )
        retourner KO
    fsi

    afficherNombresUnitairementParfaits ( nb )

    retourner OK

ffct principale
```

Définition de la procédure *afficherNombresUnitairementParfaits* :

```
procédure afficherNombresUnitairementParfaits ( donnée nb : Naturel )

    pour i = 1 : Naturel à nb
        si estUnitairementParfait ( i ) alors
            afficher ( i )
        fsi
    fpour i

    allerAlaLigne ( )

fproc afficherNombresUnitairementParfaits
```

Définition de la fonction *estUnitairementParfait* :

```
fonction estUnitairementParfait ( nombre : Naturel ) : Booléen

    somme = 0 : Naturel // somme des diviseurs
```

```
    pour diviseur = 1 : Naturel à nombre div 2

        si nombre mod diviseur = 0 et pgcd ( diviseur, nombre div diviseur ) = 1 alors
            somme ← somme + diviseur

        fsi
    fpour diviseur

    si somme = nombre alors
        retourner VRAI
    sinon
        retourner FAUX // ou plus simplement retourner ( somme = nombre )
    fsi

ffct estUnitairementParfait
```

Définition de la fonction *pgcd* :

```
fonction pgcd ( a , b : Naturel ) : Naturel

    si b = 0 alors
        retourner a
    sinon
        retourner pgcd ( b , a mod b )
    fsi

ffct pgcd
```

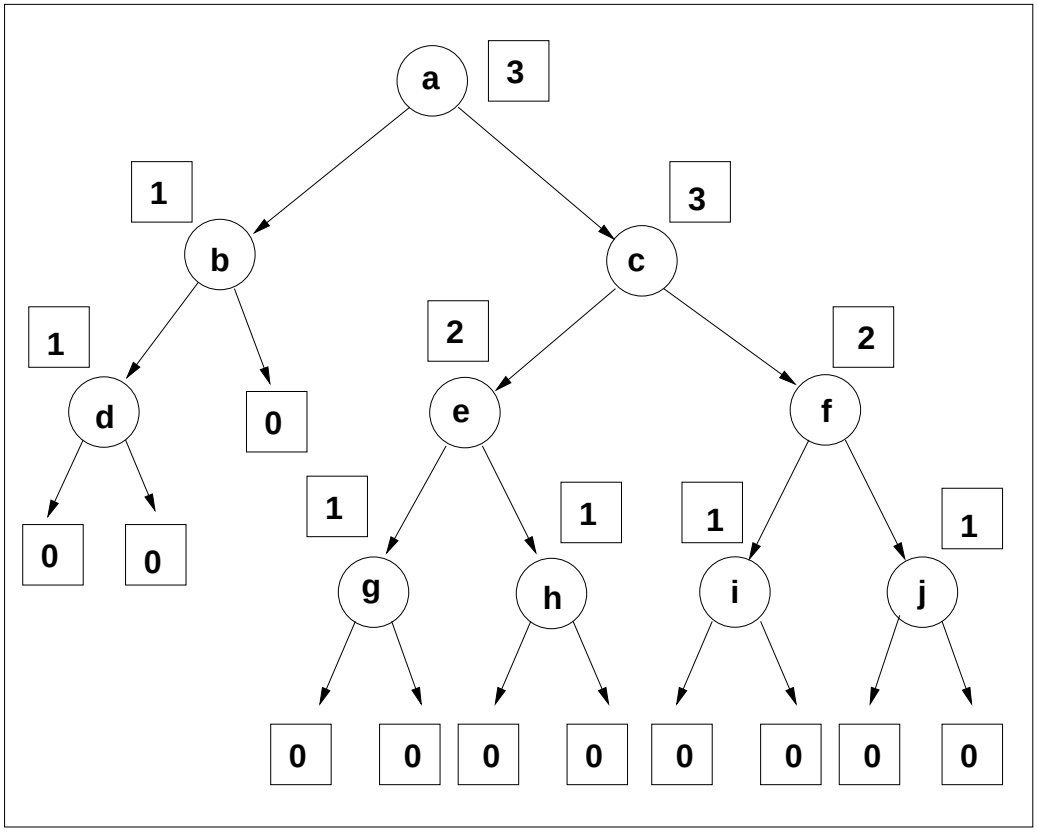
Exercice de langage C (sur 7 points)

Affichage produit par l'exécutable issu du fichier expressions.c

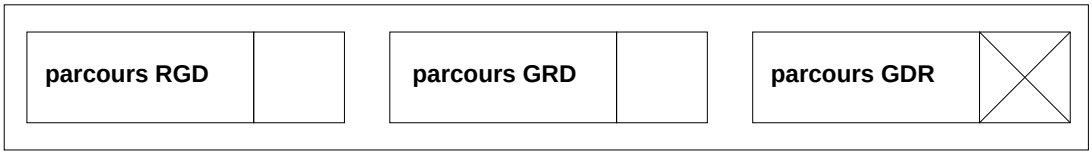
```
*p + 2 = 3

*( p + 2 ) = 7

*(p + *p + 2 ) = 10

*(p + *( p + 2 ) ) = 22

*(p + *( p + 2 ) - t[1] ) = 10

&t[7] - p + 1 = 8

&t[7] - &( *( p + 1 ) ) = 6

&t[7] - &( *( p + *( p + 1 ) ) ) = 3
```

Exercice de structures de données (sur 6 points)

1. Les valeur de Φ de l'arbre figurant ci-dessous.



2. Le parcours à utiliser pour calculer le nombre de Strahler d'un arbre binaire.



Il faut bien sûr descendre dans le sous-arbre gauche, puis dans le sous-arbre droit pour pouvoir calculer le nombre de Strahler du nœud courant.

3. La fonction *nombreDeStrahler*.

```
static int max ( int n, int m ) {

  if ( n > m )
    return n;
  return m;
}
```

```
int nombreDeStrahler ( Arbre a ) {

  if ( estArbreVide ( a ) )
    return 0;

  int nbGauche = nombreDeStrahler ( a->sag );
  int nbDroit = nombreDeStrahler ( a->sad );

  if ( nbGauche == nbDroit )
    return 1 + nbGauche;
  else
    return max ( nbGauche, nbDroit );
}
```