Épreuve d'informatique de l'X - 2009 - PSI/PT

Question 1 : Si n admet un diviseur dans l'intervalle entier $\llbracket 2, n-1 \rrbracket$, c'est que n n'est pas un nombre premier (détecté en ligne 5) et donc le résultat sera `false`. Si n est premier, la variable `bool` n'est pas modifiée dans la boucle et la valeur retournée sera `true`.

```

1  estPremier := proc(n :: posint)
2  local bool, i;
3  bool := true;
4  for i from 2 to n-1 do
5      if mod(n,i)=0 then bool := false fi
6  od;
7  if bool then return(1) else return(0) fi;
8  end;
```

Question 2 : On parcourt tous les entiers de 2 à n (boucle des lignes 5 à 11). À chaque fois que l'on trouve un nombre premier, on incrémente le compteur de nombres premiers (ligne 8) et on le stocke dans le tableau des nombres premiers (ligne 9). Le tableau `premier` est supposé être indexé à partir de 1. La variable `nbPremiers` permet de compter le nombre de nombres premiers.

```

1  petitsPremiers := proc(n :: posint)
2  global premier;
3  local i, nbPremiers;
4  nbPremiers := 0;
5  for i from 2 to n do
6      if estPremier(i) = 1
7          then
8              nbPremiers := nbPremiers+1;
9              premier[nbPremiers] := i
10         fi;
11  od;
12  return(nbPremiers);
13  end;
```

Question 3 : On stocke 2 dans le tableau des nombres premiers (ligne 4) puis on examine les entiers de deux en deux (ligne 19) à partir de 3 (boucle des lignes 7 à 20). Pour un entier i donné, on teste la divisibilité de i avec les entiers premiers p ($p \geq 3$) déjà trouvés et qui vérifient $p^2 \leq i$ (boucle conditionnelle des lignes 11 à 15). Si on n'a trouvé aucun diviseur pour i , on incrémente le compteur de nombres premiers et on ajoute i au tableau des nombres premiers (lignes 16 à 18).

```

1  petitsPremier2 := proc(n :: posint)
2  global premier;
3  local i, j, bool, nbPremiers, p;
4  premier[1] := 2;
5  nbPremiers := 1;
6  i := 3;
7  while i <= n do
8      j := 2;
9      bool := true;
10     p := 3;
11     while p*p<=i && bool do
12         if mod(i,p) = 0 then bool := false fi;
13         j := j+1;
14         p := premier[j];
15     od;
16     if bool then nbPremiers := nbPremiers + 1;
17                 premier[nbPremiers] := i;
18     fi;
19     i := i+2;
20 od;
21 return(nbPremiers);
22 end;
```

Question 4 : On écrit la fonction `petitsPremiers3` qui recherche les nombres premiers en suivant la méthode du crible d'Ératosthène. L'étape 1 correspond à la boucle des lignes 9 à 11. L'étape 2 correspond à la boucle conditionnelle de la ligne 16. L'étape 3 correspond aux lignes 17 à 23. Quand on n'a coché aucune case, on quitte la boucle puis on compte et on place les nombres premiers trouvés dans le tableau `premier` (boucle des lignes 28 à 31).

```

1 petitsPremiers3 := proc(n :: posint)
2 global premier;
3 local i, nbPremiers, caillou1, caillou2, pas, raye, tab;
4 raye := true;
5 tab := array(1..n-1);
6 for i from 1 to n-1 do
7   tab[i] := 0;
8 od;
9 caillou1 := 0;
10 while raye do
11   raye := false;
12   caillou1 := caillou1 + 1;
13   # on positionne la première pierre (étape 2).
14   while tab[caillou1]  $\diamond$  0 do caillou1 := caillou1+1; od;
15   pas := caillou1 + 1;
16   # on positionne la deuxième pierre.
17   caillou2 := caillou1 + pas;
18   # on biffe les cases (étape 3).
19   while caillou2  $\leq$  n-1 do
20     if tab[caillou2] = 0
21       then tab[caillou2] := 1; raye := true fi;
22     caillou2 := caillou2+pas;
23   od;
24 od;
25 nbPremiers := 0;
26 # on récupère les nombres premiers
27 for i from 1 to n-1 do
28   if tab[i]  $\diamond$  1 then nbPremiers := nbPremiers+1;
29     premier[nbPremiers] := i+1 fi;
30 end;
31 return(nbPremiers)
32 end;

```

Question 5 : L'étape 1, qui est une étape d'initialisation, est effectuée en lignes 4 et 5. L'étape 2 de l'algorithme de l'énoncé correspond à la boucle conditionnelle des lignes 7 à 14. On sort de cette boucle dès que m vaut 1. Les divisions successives ont lieu dans la boucle des lignes 8 à 12. Cette boucle correspond à l'étape 3. L'étape 4 correspond à la ligne 13.

```

1 factoriser := proc(n :: posint)
2 global premier, facteur;
3 local i, m, nbDiviseurs;
4 m := n; i := 1;
5 petitesPremiers3(n);
6 nbDiviseurs := 0;
7 while m  $\diamond$  1 do
8   while mod(m, premier[i]) = 0 do
9     nbDiviseurs := nbDiviseurs + 1;
10    facteur[nbDiviseurs] := premier[i];
11    m := m/premier[i];
12  od;
13  i := i+1;
14 od;
15 return(nbDiviseurs)
16 end;

```

Question 6 : Dans cette fonction de factorisation, on n'utilise plus le tableau des nombres premiers. Le test des lignes 14 à 18 correspond à la seconde remarque de l'énoncé. Si le dernier m calculé est différent de 1 c'est que c'est un facteur premier de la décomposition de n , on le rajoute alors aux facteurs de n .

```

1 factoriser2 := proc(n :: posint)
2 global facteur;
3 local i, j, m, nbDiviseurs;
4 m := n;
5 i := 2;
6 j := 1;
7 nbDiviseurs := 0;
8 while m  $\diamond$  1 and i*i  $\leq$  m do
9   while mod(m,i)  $\diamond$  0 do j := j+2; i := j od;
10  nbDiviseurs := nbDiviseurs + 1;
11  facteur[nbDiviseurs] := i;
12  m := m/i;
13 od;
14 if m  $\diamond$  1
15   then
16     nbDiviseurs := nbDiviseurs + 1;
17     facteur[nbDiviseurs] := m;
18   fi;
19 return(nbDiviseurs)
20 end;

```

Question 7 : Avec la fonction `factoriser`, on a rangé les facteurs premiers de n en ordre croissant dans le tableau `facteur` (ligne 4). On exploite ce tableau pour remplir le tableau `alpha`. C'est l'objet de la boucle des lignes 8 à 16. On traite le cas du dernier facteur premier de n aux lignes 17 et 18.

```

1 calculerAlpha := proc(n :: posint)
2 global facteur , alpha;
3 local i , m , p , nbFacteurs , compt;
4 m := factoriser(n);
5 p := facteur[1];
6 nbFacteurs := 0;
7 compt := 0;
8 for i from 1 to m do
9   if facteur[i] = p
10    then compt := compt+1;
11    else nbFacteurs:=nbFacteurs+1;
12         alpha[nbFacteurs] := compt;
13         compt := 1;
14         p := facteur[i];
15   fi;
16 od;
17 nbFacteurs:=nbFacteurs+1;
18 alpha[nbFacteurs] := compt;
19 return(nbFacteurs);
20 end;

```

Question 8 : Pour tester si n est de la forme a^b , on commence par calculer les puissances des facteurs premiers de la décomposition de n (ligne 4), puis on vérifie que tous les ordres de multiplicité des facteurs premiers de la décomposition de n sont divisibles par b (boucle des lignes 6 à 8).

```

1 estPuissance := proc(n::posint ,b::posint)
2 global alpha;
3 local i , m , bool;
4 m := calculerAlpha(n);
5 bool := true;
6 for i from 1 to m do
7   bool := bool and (mod(alpha[i],b) = 0)
8 od;
9 if bool then return(1) else return(0) fi;
10 end;

```

Question 9 : Pour détecter si n est un carré ou pas, on peut appliquer une méthode de Newton modifiée pour rechercher une racine de l'équation $x^2 = n$. La méthode de Newton converge très rapidement. Ici on effectue des divisions entières au lieu d'effectuer des divisions réelles. On arrête la boucle dès que la distance entre deux termes consécutifs de la suite est inférieure ou égale à 1. Il ne reste plus cas vérifier si la dernière valeur calculer est la racine carrée de n (ligne 9).

```

1 estCarre := proc(n :: posint)
2 local x , y;
3 x:= n;
4 y := iquo(x^2+n,2*x);
5 while abs(y - x) > 1 do
6   x := y;
7   y := iquo(x^2+n,2*x)
8 od;
9 return(evalb(y*y=n));
10 end;

```

Ou bien, on peut tester si n est égal au carré de sa racine carrée entière :

```

      rac := sqrt(n); evalb(n=rac*rac);

```

Question 10 : On applique directement la définition donnée par l'énoncé des nombres de Carmichael.

- on vérifie que c est impair non premier (ligne 4),
- on vérifie que les ordres de multiplicité des facteurs premiers de la décomposition de c sont égaux à 1 (ligne 7),
- on vérifie que les $p_i - 1$ divise $c - 1$ pour tous les facteurs premiers p_i entrant dans la décomposition de c (ligne 8).

```

1 estCarmichael := proc(c :: posint)
2 global facteur , alpha;
3 local i , m , bool;
4 bool := ((c mod 2 = 1) and (not(estPremier(c))));
5 m := calculerAlpha(c);
6 for i from 1 to m do
7   bool := bool and ((alpha[i] = 1)
8                     and ((c-1) mod (facteur[i]-1) = 0));
9 end;
10 if bool then return(1) else return(0) fi;
11 end;

```

Question 11 : On suit les consignes de l'énoncé. On calcule les nombres premiers de 3 à n (ligne 4). On calcule tous les produits de trois de ces nombres et on teste si ce sont des nombres de Carmichael (boucle des lignes 6 à 21). Si on a bien un nombre de Carmichael, on le stocke dans le tableau `carmichael` et on incrémente le compteur de nombres de Carmichael (ligne 13 à 18).

```

1 calculerCarmichael3 := proc(n::posint)
2 global premier, carmichael;
3 local nbPrem, nbCarmichael, i, j, k, p1, p2, p3, c;
4 nbPrem := petitsPremiers3(n);
5 nbCarmichael := 0;
6 for i from 1 to nbPrem-2 do
7   for j from i+1 to nbPrem-1 do
8     for k from j+1 to nbPrem do
9       p1 := premier[i];
10      p2 := premier[j];
11      p3 := premier[k];
12      c := p1*p2*p3;
13      if (c <= n) and ((c-1) mod (p1-1)= 0)
14        and ((c-1) mod (p2-1)= 0)
15        and ((c-1) mod (p3-1)= 0)
16        then
17          carmichael[nbCarmichael] := c;
18          nbCarmichael := nbCarmichael + 1;
19        fi;
20      od;
21    od;
22  od;
23 return(nbCarmichael)
24 end;
```

Question 12 : On applique la méthode de criblage donnée par l'énoncé. On crée le tableau `t` (ligne 6) que l'on initialise (boucle des ligne 7 à 9). Dans la boucle des lignes 15 à 18, on effectue la division de `t[i]` par p pour tous les entiers i de la forme $p^2 + k.p(p-1)$ plus petits que n . Dans la boucle des lignes 27 à 28, on récupère et on compte tous les nombres de Carmichael compris entre 1 et n .

```

1 calculerCarmichael := proc(n)
2 global premier, carmichael;
3 local nbPrem, nbCarmichael, i, j, t, p, pas;
4 nbPrem := petitsPremiers3(sqrt(n));
5 nbCarmichael := 0;
6 t := array(0..n);
7 for i from 0 to n do
8   t[i] := i;
9 od;
10 i := 2;
11 p := premier[i]; # on commence avec p=3
12 while p*p < n do
13   j := p*p;
14   pas := p*(p-1);
15   while j <= n do
16     t[j] := iquo(t[j],p);
17     j := j+pas
18   od;
19   i := i+1;
20   p := premier[i];
21 od;
22 nbCarmichael := 0;
23 # on récupère les nombres de Carmichael
24 for i from 2 to n-1 do
25   if t[i] = 1 then nbCarmichael := nbCarmichael+1;
26     carmichael[nbCarmichael] := i;
27   fi;
28 end;
29 return(nbCarmichael)
30 end;
```

Voici les nombres de Carmichael inférieurs ou égaux à 100000 : 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973, 75361