CORRIGE : Les procédures et les fonctions

Exercice I :

1 - Trouver le résultat fourni par l'algorithme :

```
Procédure SomCar ( →X1 : numérique, → X2 : numérique, ↔S : numérique)
Début
      X1 ← X1 * X1
      X2 ← X2 * X2
      S ← X1 + X2
Fin

Programme principal :
      Variables X, Y, Z : numériques
      X ← 3
      Y ← 4
      Z ← 0
      SomCar(X, Y, Z)
      Ecrire X,  « ^2 + », Y, «  ^2 = », Z
Fin du programme principal
```

Ce programme affiche pour résultat « 3^2+4^2 = 25 ». On remarquera que la procédure, bien qu'elle ait modifié les valeurs des paramètres formels X1 et X2, n'a pas modifié la valeur des paramètres effectifs X et Y car ils étaient en entrée (passés par valeur).

2 - Remplacer dans ce programme la procédure par une fonction.

Avec une fonction, ce programme devient :

```
Fonction SomCar ( →X1 : numérique, → X2 : numérique) : numérique
Début
      X1 ← X1 * X1
      X2 ← X2 * X2
      Résultat X1 + X2
Fin

Programme principal :
      Variables X, Y, Z : numériques
      X ← 3
      Y ← 4
      Z ← SomCar(X, Y)
      Ecrire X,  « ^2 + », Y, «  ^2 = », Z
Fin du programme principal
Remarque :
      On peut aussi écrire :
      Ecrire X,  « ^2 + », Y, «  ^2 = », SomCar(X, Y)
```

Exercice II : Une procédure est déclarée par :

Procédure Essai (→A : numérique, ↔B : numérique, ↔ C : numérique)
Début
    A ← A + 1
    B ← 22
    C ← C + 3
Fin

Parmi les appels suivants certains ne sont pas corrects, expliquer pourquoi. Pour les autres, trouver les valeurs des paramètres A, B, C au début et à la fin de son exécution, des variables X, Y et Z ensuite.

Avant chacun des appels, on effectue :

X ← 3
Y ← 7
Z ← 11

1 – Essai (1 , 2, 3)
Appel incorrect car les paramètres formels B et C sont en entrée / Sortie : il doit leur correspondre des variables, pas des constantes.
2 – Essai (X, Y, Z)
Au début de l'exécution de la procédure, les paramètres A, B et C valent respectivement 3, 7 et 11. A la fin de la procédure, leurs valeurs sont devenues 4, 22 et 14. Après l'exécution de la procédure, X, Y, Z ont pour nouvelles valeurs 3, 22 et 14.
3 – Essai (Z, Y, X)
Au début de l'exécution de la procédure, les paramètres A, B et C valent respectivement 11, 7 et 3. A la fin de la procédure, leurs valeurs sont devenues 12, 22 et 6. Après l'exécution de la procédure, X, Y, Z ont pour nouvelles valeurs 6, 22 et 11.
4 – Essai (1+X*10, Y, Z)
L'expression 1+X*10 est évaluée avant que sa valeur soit transmise au paramètre 1. Au début de l'exécution de la procédure, les paramètres A, B et C valent respectivement 31, 7 et 11. A la fin de la procédure, leurs valeurs sont devenues 32, 22 et 14. Après l'exécution de la procédure, X, Y, Z ont pour nouvelles valeurs 3, 22 et 14.
5 – Essai (X, X, Z)
Au début de l'exécution de la procédure, les paramètres A, B et C valent respectivement 3, 3 et 11. A la fin de la procédure, leurs valeurs sont devenues 4, 22 et 14. Après l'exécution de la procédure, X, Y, Z ont pour nouvelles valeurs 22, 7 et 14. On a pu utiliser deux fois le paramètre effectif X sans problème.
6 – Essai (X, Y, Y)
Au début de l'exécution de la procédure, les paramètres A, B et C valent respectivement 3, 7 et 7. A la fin de la procédure, leurs valeurs sont devenues 4, 25 et 25. Après l'exécution de la procédure, X, Y et Z ont pour nouvelles valeurs 3, 25 et 11. On a pu utiliser deux fois le paramètre effectif X, mais comme cette fois ci il correspondait à deux paramètres formels en entrée sortie, des résultats curieux ont été obtenus (on aurait pu croire que C aurait valu 7 + 3 = 10, mais l'affectation B ← 22 a modifié aussi la valeur de Y et celle de C). Bien sûr, ce genre de situation est à éviter absolument car l'algorithme ainsi conçu produit des résultats presque imprévisibles et se montre particulièrement peu clair.

- ⇨ **On peut utiliser plusieurs fois la même variable comme paramètre effectif sous réserve que, parmi les paramètres formels qui lui correspondent un seul soit en entrée / sortie.**

Exercice III : Quels sont les résultats produits par l'algorithme suivant :

```
Procédure Max (→X : numérique,  →Y : numérique, ↔ M : numérique)
Début
      A ← X
      Si A < Y  alors
            A ← Y
      Fsi
      M ← A
Fin // de la procédure

Début du programme principal
      Variables A, B, C : numériques

      A ← 3
      B ← 7
      C ← 0
      Max (A, B, C)
      Ecrire « Le maximum de », A, « et », B, « est  », C
Fin du programme principal
```

On aurait souhaité que cet algorithme fournisse pour résultat « Le maximum de 3 et 7 est 7 », mais il donne « Le maximum de 7 et 7 est 7 ». A n'est pas le nom d'un paramètre ni d'une variable locale pour la procédure qui utilise donc la variable A du programme principal comme variable globale et la modifie en lui affectant d'abord 3, puis 7. Si par malchance les valeurs initiales de A et B avaient été respectivement de 7 et 3, on ne se serait aperçu de rien et on n'aurait découvert ce problème que le jour où les valeurs de A et B auraient été dans l'ordre croissant.

La modification subreptice d'une variable globale par une procédure ou une fonction (effet de bord) est une cause de mauvais fonctionnement de certains algorithmes souvent difficile à repérer et même à déceler.

- ⇨ **Pour éviter tout effet de bord, il faut que toutes les variables utilisées dans les procédures et les fonctions soient soit des paramètres, soit des variables locales.**

Exercice IV : Les algorithmes suivants ont été écrits par un mauvais programmeur, particulièrement maladroit dans les choix des noms des paramètres et peu soucieux d'éviter les effets de bords. Seuls les modes de transmission des paramètres diffèrent entre ces quatre algorithmes. Que produisent ils ?

Version 1

```
Procédure Deux (↔A : numérique)
Début
      A ← A + 1
      Ecrire A
Fin

Procédure Un (↔ B : numérique)
Début
      A ← A + 1
      Deux (A)
      B ← B +1
      Deux (B)
      Ecrire A, B
Fin

Début programme principal
      Variables A, B : numériques
      A ← 10
      Deux (A)
      Un (A)
      B ← 10
      Deux (B)
      Un (B)
      Ecrire A, B
Fin du programme principal
```

Version 2

```
Procédure Deux (↔A : numérique)
Début
      A ← A + 1
      Ecrire A
Fin

Procédure Un (→ B : numérique)
Début
      A ← A + 1
      Deux (A)
      B ← B +1
      Deux (B)
      Ecrire A, B
Fin

Début programme principal
      Variables A, B : numériques
      A ← 10
      Deux (A)
      Un (A)
      B ← 10
      Deux (B)
      Un (B)
      Ecrire A, B
Fin du programme principal
```

Version 3

```
Procédure Deux (→A : numérique)
Début
      A ← A + 1
      Ecrire A
Fin

Procédure Un (→ B : numérique)
Début
      A ← A + 1
      Deux (A)
      B ← B +1
      Deux (B)
      Ecrire A, B
Fin

Début programme principal
      Variables A, B : numériques
      A ← 10
      Deux (A)
      Un (A)
      B ← 10
      Deux (B)
      Un (B)
      Ecrire A, B
Fin du programme principal
```

Version 4

```
Procédure Deux (→A : numérique)
Début
      A ← A + 1
      Ecrire A
Fin

Procédure Un (↔ B : numérique)
Début
      A ← A + 1
      Deux (A)
      B ← B +1
      Deux (B)
      Ecrire A, B
Fin

Début programme principal
      Variables A, B : numériques
      A ← 10
      Deux (A)
      Un (A)
      B ← 10
      Deux (B)
      Un (B)
      Ecrire A, B
Fin du programme principal
```

Les quatre algorithmes produisent des résultats différents en raison de la modification des paramètres effectifs lors des passages par adresse, et de leur conservation lors des passages par valeur.

Version 1	Version 2	Version 3	Version 4
11	11	11	11
13	13	12	12
15	13	12	13
15 15	13 13	11 11	12 12
11	11	11	11
17	15	13	14
13	13	12	12
17 13	15 13	12 11	13 11
17 13	15 11	12 10	13 11

ECRITURE DE PROCEDURES ET DE FONCTIONS

Exercice V : Ecrire une fonction qui permet de savoir si un entier est divisible par un autre. On pourra utiliser un nouveau type nommé logique afin de renvoyer le résultat

```
Solution
fonction logique Divise (var entier a, var entier b)
début

    si (a mod b = 0)
        retourner vrai;
    sinon
        retourner faux;

fin

qui peut s'utiliser : si (Divise (x, y) = vrai)
```

Exercice VI : Créer un petit ensemble de procédures et de fonctions permettant de manipuler facilement les heures et les minutes et composé de :

1- La fonction *Minutes*, qui calcule le nombre des minutes correspondant à un nombre d'heures et un nombre de minutes donnés.

```
Fonction Minutes ( →H : numérique, → M : numérique) : numérique
Début
      Résultat H * 60 + M
Fin
```

2- La fonction ou la procédure HeuresMinutes qui réalise la transformation inverse de la fonction Minute.
Pour HeuresMinutes, il y a deux résultats à fournir, une fonction ne peut convenir, il faut donc écrire une procédure comportant trois paramètres :
La durée (entrée), l'heure (sortie) et les minutes (sortie).

```
Procédure HeuresMinutes (→ Durée : numérique, ↔ H : numérique, ↔
M : numérique)
Début
   H ← Durée Div 60      // division entière
   M ← Durée – 60 * H
Fin
```

3- La procédure AjouteTemps qui additionne deux couples de données heures et minutes en utilisant les deux fonctions précédentes.

La procédure AjouteTemps reçoit quatre paramètres en entrée, fournit deux paramètres en sortie. La variable locale MinuteEnTout sert à stocker un résultat intermédiaire, mais elle n'est pas indispensable.

```
Procédure AjouteTemps (→H1 : numérique, → M1 : numérique, → H2 :
numérique,  → M2 : numérique, ↔Hsomme : numérique, ↔Msomme :
numérique)
      Variables MinuteEnTout : numérique
      Début
            MinuteEnTout ← Minutes (H1, M1) + Minutes (H2, M2)
            HeuresMinutes(MinuteEnTout, Hsomme , Msomme)
      Fin
```

ON PEUT INTRODUIRE LE TYPE BOOLEEN (VRAI ET FAUX)

Exercice VII : Cet exercice permet de compléter les procédures et fonctions de l'exercice précédent

1- Créer une fonction qui permet de dire si un mois a 30 jours ou non. Cette fonction renverra 1 si c'est le cas et 0 sinon.

1	2	3	4	5	6	7	8	9	10	11	12
Janv.	Fév.	Mars	Avril	Mai	Juin	Juil	Aout	Sept	Oct	Nov	Dec
31	28 ou 29	31	30	31	30	31	31	30	31	30	31

```
Fonction EstUnMoisDeTrenteJours ( → mois : numérique)  :
numérique
Début
      Si (mois = 4 ou mois = 6 ou mois = 9 ou mois = 11) alors
            Résultat 1
      Sinon
            Résultat 0
      Fsi
Fin
```

2- Créer une fonction qui permet de dire si un mois a 31 jours ou non. Cette fonction renverra 1 si c'est le cas et 0 sinon.

```
      Fonction EstUnMoisDeTrenteEtUnJours ( → mois : numérique)  :
      numérique
      Début
            Si (mois = 1 ou mois = 3 ou mois = 5 ou mois = 7 ou mois
      = 8 ou mois = 10 ou mois = 12) alors
            Résultat 1
      Sinon
            Résultat 0
      Fsi
      Fin
```

3- Créer une fonction qui permet de dire si une année est bissextile ou non. Cette fonction renverra 1 si c'est le cas et 0 sinon.
Pour qu'une année soit bissextile, il suffit que l'année soit un nombre divisible par 4 et non divisible par 100, ou alors qu'elle soit divisible par 400.

```
      Fonction EstUneAnneeBissextile ( → annee : numérique)  :
      numérique
      Début
```

```
        Si ((annee Mod 4 = 0 et annee Mod 100 <> 0) ou (annee Mod
400 = 0)) alors
              // Année bissextile
              Résultat 1
        Sinon
              Résultat 0
        Fsi
Fin
Si l'année est bissextile alors le mois de février à 29 jours.
Il en a 28 sinon
```

4- Utiliser ces fonctions pour écrire une fonction retournant le nombre de jours pour un mois et une année donnés.

```
Fonction Nombre_de_jours (→mois : numérique, → annee :
numérique)  : numérique
Variable est31, est30 : numériques

Début
      est31 ←EstUnMoisDeTrenteEtUnJours(mois)
      est30 ←EstUnMoisDeTrenteJours(mois)
      Si (est31 = 1) alors
            Résultat 31
      Sinon
            Si (est30 = 1) alors
                  Résultat 30
            Sinon
                  // mois de février
                  // regarder si bissextile
                  Si (EstUneAnneeBissextile(annee) = 1) alors
                        Résultat 29
                  Sinon
                        Résultat 28
                  Fsi
            Fsi
      Fsi
Fin
```

5- Ecrire un programme principal permettant à l'utilisateur d'entrer un numéro de mois (entre 1 et 12) et une année (entre 1582 et 2003), qui seront ensuite passés en paramètres à la fonction **`Nombre_de_jours()`**.

Il faut tester la validité des mois et années.

INTRODUIRE LE TYPE CHAINE DE CARACTERES

Exercice VIII : On désire gérer un tableau contenant une liste de noms. Pour cela, on décide de répéter l'affichage d'un menu et l'exécution de la commande choisie par l'utilisateur. Le menu sera de la forme qui suit :

Veuillez frapper :
+ la lettre V pour voir la liste
+ La lettre S pour supprimer un nom de la liste
+ La lettre A pour ajouter un nom à la liste
+ La lettre pour rechercher si un nom est dans la liste
+ La lettre T pour terminer.

Cette gestion exige d'abord qu'un même nom ne figure pas deux fois dans la liste, ensuite qu'un nouveau nom soit ajouté, au début de la liste, à la fin de la liste, ou après un autre nom, selon le choix de l'utilisateur.

On pourrait écrire un « gros » algorithme monolithique, mais pour que le programme principal soit le plus simple donc le plus clair possible, on y fera apparaître des appels de procédures. Les écritures séparées de ces procédures mettront en évidence l'intérêt de nouvelles procédures et fonctions qui leur rendront des services identiques. On ne détaillera pas les algorithmes des diverses procédures tant que le programme ne sera pas entièrement organisé.

La première étape d'écriture des programmes offrant un menu est toujours très simple, dans le style suivant :

```
Tableau Liste [Nmax] :  Chaînes de caractères
Variable N : numérique
Variable choix : caractère

Début
        N ← 0
        Répéter
                Ecrire « Tapez : »
                Ecrire « V pour voir la liste »
                Ecrire « S pour supprimer un nom de la liste »
                Ecrire «  A pour ajouter un nom au début de la liste »
                Ecrire « R pour rechercher si un nom est dans la liste »
                Ecrire « T pour terminer »

                Lire choix
                Si Choix = 'V' alors Voir(T,N) Fsi
                Si Choix = 'S' alors Supprimer (T,N) Fsi
                Si Choix = 'R' alors Recherche (T,N) Fsi
                Si Choix = 'A' alors Ajouter (T,N) Fsi
        Jusqu'à Choix = 'T'
Fin
```

On remarque qu'il n'est pas nécessaire dans la lecture de Choix que la réponse soit correcte (V, S, R, A ou T) car une autre réponse ne provoque l'exécution d'aucune procédure et la boucle répéter fait afficher de nouveau le menu et effectuer une nouvelle lecture de la variable Choix.

Commençons par bâtir le cadre des procédures :

```
Supprimer :
        Choisir le nom à supprimer
        Si ce nom existe alors le supprimer
        Sinon fournir un message d'erreur

Ajouter :
        Choisir le nom à ajouter
        Si ce nom existe
                Alors
                        fournir un message d'erreur
                Sinon
                        Choisir Début, Fin, Après
                        Si Début alors insérer en place 1
```

Si Fin alors insérer en place N+1
Si Après alors
Choisir après qui
Si qui existe alors insérer après
Sinon erreur

Voir :
Passer toute la liste en revue avec un Répéter pour

Rechercher :
Choisir le nom à rechercher
Si ce nom existe alors
Ecrire « Présent »
Sinon
Ecrire « Absent »

On a fait apparaître l'intérêt d'une fonction Existe, d'une procédure Insérer, d'une fonction Place donnant la place d'un nom dans la liste et aussi des procédures Erreur et LireNom. Détaillons les maintenant.

```
Procédure Erreur(→Message : chaîne de caractère)
Début
      Ecrire « Action impossible car », Message
Fin

Procédure LireNom (↔Nom : chaîne de caractères, → Question : chaîne
de caractères)
Début
      Ecrire Question
      Lire Nom
Fin

Procédure Insérer (↔tableau T[Nmax] : Chaînes de caractères, ↔N :
numérique, → Qui : Chaîne de caractère, → Endroit : numérique)
Variable i : numérique
Début
      // faire éventuellement un trou pour loger le nouveau nom
      Répéter pour i = N en descendant jusqu'à Endroit faire
            T[i+1] ← T[i] // boucle à faire absolument en descendant
      FinPour
      T[Endroit] ← Qui
      N ← N + 1
Fin

Fonction Place (→tableau T[Nmax] : Chaîne de caractère, → N :
numérique, → Qui : chaîne de caractère) : numérique
Variable i : numérique
Début
      // le résultats sera zéro si l'élément n'est pas trouvé
      T[N+1] ← Qui  // on utilise la méthode sentinelle
      i ← 1
      Tant que T[i] ≠ Qui faire
            i ← i + 1
      FinTantque
      Si i > N alors i ← 0 Fsi
      Résultat i
Fin
```

```
Fonction Existe (→ tableau T[Nmax] : Chaîne de caractère, ↔ N :
numérique, → Qui : Chaîne de caractère): numérique
Début
      Si Place (T, N, Qui) ≠ 0 alors Existe ← 1
      Sinon
            Existe ← 0
      Fsi
Fin

Procédure Ajouter (↔tableau T[Nmax] : Chaîne de caractère, ↔ N :
numérique)
Variables Qui, Après : Chaînes de caractères
Variable Mode : caractère
Début
      LireNom (Qui, « Qui voulez vous ajouter ?)
      Si Existe (T, N, Qui) = 1 alors
            Erreur (« ce nom existe déjà)
      Sinon
            Répéter
                  Ecrire « Début, Fin, ou après un autre (D,F,A) ?»
                  Lire Mode
            Jusqu'à (Mode = 'D' ou Mode = 'F' ou Mode = 'A'
            Si Mode = 'D' alors Insérer (T, N, Qui, 1) Fsi
            Si Mode = 'F' alors Insérer (T, N, Qui, N+1) Fsi
            Si Mode = 'A' alors
                  LireNom(Après, « Après Qui ? »)
                  Si Existe(T, N, Après) = 1 alors
                        Insérer (T, N, Qui, Place(T, N, Après)
                  Sinon
                        Erreur (« Nom inconnu »)
                  Fsi
            Fsi
      Fsi
Fin

Procédure rechercher (→tableau T[Nmax] : Chaîne de caractère, → N :
numérique)
Variable Qui : chaîne de caractère
Début
      LireNom(Qui, « Qui recherchez vous ? »)
      Si Existe(T, N, Qui) = 1 alors
            Ecrire « Présent dans la liste »
      Sinon
            Ecrire « Inconnu dans la liste »
      Fsi
Fin

Procédure Supprimer ( ↔ tableau T[Nmax] : Chaîne de caractère, ↔ N :
numérique)
Variables Qui : Chaîne de caractères
Variable i : numérique
Début
      LireNom(Qui, « Qui voulez vous supprimer ? »)
      Si Existe (T, N, Qui) = 1 alors
            N ← N - 1
            Répéter pour i = Place (T, N, Qui) jusqu'à N faire
                  T[i] ← T[i+1]
            FinPour
      Sinon
```

```
            Erreur(« Ce nom n'est pas dans la liste »)
      Fsi
Fin

Procédure Voir (→ tableau T[Nmax] : Chaîne de caractère,  → N :
numérique)
Variable i : numérique
Début
      Ecrire « Liste des noms »
      Répéter pour i = 1 à N faire
            Ecrire T[i]
      FinPour
Fin
```

Exercice IX (Exercice Complémentaire) : Calcul de l'impôt sur le revenu
Remarque : Pour les besoins de l'exercice, certains calculs ont été simplifiés.

Dans cet exercice, nous cherchons à simuler le calcul de l'impôt sur le revenu d'une famille ayant des revenus salariés. Cette famille est caractérisée par le nombre d'adultes et le nombre d'enfants à charge. Les autres données prises en compte sont le salaire et les abattements. Pour cela, on définit les fonctions ou procédures suivantes :

1- Fonction qui calcule le revenu imposable RI
Pour cela, vous devez :
   - Déduire du salaire 10 % pour frais professionnel (cet abattement est limité à un plafond de 77460F),
   - Ensuite déduire un abattement de 20 % (limité à un plafond de 141400F) pour obtenir le revenu imposable.
2- Fonction qui calcule le nombre de parts N qui est différent suivant qu'il s'agit d'une famille à un seul adulte ou d'une famille ayant plus de deux enfants à charge. Pour un adulte élevant seul ses enfants, les enfants comptent comme une part. Pour un couple marié (adulte = 2) les deux premiers enfants comptent pour ½ part et au delà un enfant compte une part comme pour une personne seule.

Les deux fonctions précédentes servent à déterminer le quotient familial : QF = RI / N. Ce quotient sert à déterminer la tranche d'imposition appropriée, de façon à appliquer la formule générale de calcul, qui est du type :
$I = RI * T_i – K_i * N$ avec T_i et K_i donnés dans le tableau suivant.

Barème d'imposition pour la déclaration des revenus 1998

Tranche	T_i	Abattement K_i
QF ≤ 26100	0	0
26100 < QF ≤ 51340	0,105	2740,50
51340 < QF ≤ 90370	0,24	9671,40
90370 < QF ≤ 146320	0,33	17804,70
146320 < QF ≤ 238080	0,43	32436,70
238080 < QF ≤ 293600	0,48	44340,70
QF > 293600	0,54	61956,70

3- **Procédure qui initialise un tableau** (à deux dimensions) contenant le barème de l'impôt

4- Dans le **programme principal**, les données sont lues pour calculer le quotient familial. Il faut ensuite rechercher la tranche correspondant au barème puis on calcule l'impôt suivant la formule décrite ci-dessus.

```
PROGRAM PIMPOT ;
Var Revenu, Impot  : real ;
    Adulte, Enfant, i: integer ;
    Bareme : ARRAY[1..7, 1..4] of real ;
    QF : real ;

PROCEDURE LIRE_BAREME ;
Var i, j : integer ;
BEGIN
FOR i := 1 TO 7 DO
      FOR j := 1 TO 4 DO
    BEGIN
      Writeln('Entrer la valeur se trouvant en ligne ', i, ' et en colonne ', j);
      READLN(Bareme[i,j]) ;
    END;
END ;

{Remarque sur la structure du tableau : la première colonne contient la
borne inférieure, la deuxième la borne supérieure, la troisième Ti et la
quatrième Ki }

FUNCTION REVENU_IMPOSABLE : real ;
Var temp, C : real ;
BEGIN
      temp := Revenu / 10 ; (* Abattement de 10% *)
      IF temp <= 77460
      THEN C := Revenu - temp
      ELSE C := revenu - 77460 ;
      temp := C * 2 / 10 ; (* Abattement de 20% *)
      IF temp <= 141400
      THEN REVENU_IMPOSABLE := revenu - temp
      ELSE REVENU_IMPOSABLE := revenu - 141400 ;
END;

FUNCTION PARTS(Adulte, Enfant : integer) : real ;
BEGIN
   IF Adulte = 1 THEN PARTS := Adulte + Enfant
               ELSE IF (Enfant = 1) OR (Enfant =2)THEN PARTS:= Adulte + Enfant / 2
                                    ELSE PARTS := Adulte + Enfant - 1 ;
END ;
{ Pour un couple marié (adulte = 2) les deux premiers enfants comptent pour
½ part et au delà un enfant compte une part
comme pour une personne seule. }


(* programme principal *)
BEGIN
LIRE_BAREME ;
writeln(' Entrez le nombre d''adultes et d''enfants : ');
READLN(Adulte, Enfant) ;
WRITELN('Entrez le revenu de la famille : ');
READLN(Revenu) ;
QF := REVENU_IMPOSABLE / PARTS(Adulte, Enfant);
i := 1 ;
WHILE (QF > Bareme[i,2]) DO
      i := i + 1 ;
Impot := Bareme[i,3] * REVENU_IMPOSABLE ;
Impot := Impot - Bareme[i,4] * PARTS(Adulte, Enfant);
WRITELN('IMPOT = ', Impot) ;
READLN;
END.
```