R101 : Initiation au développement

TD 14 : révisions

TABLEAUX ET TRIS : tableau à une dimension, structures et tris

Considérons les déclarations suivantes :

```
constante entier MAXNUM :=3;        // nombre maximum de comptes par personne
constante entier MAXPERS :=10;      // nombre maximum de personnes gérées

type t_tabNumeros = tableau[MAXNUM] de entier ;

type t_personne = structure
début
        c_nom : chaine(30) ;        // nom de la personne
        c_liste : t_tabNumeros ;    // liste de ses n° de comptes bancaires
        c_nbComptes : entier;       // nombres de comptes actifs
fin

type t_tabPersonnes = tableau[MAXPERS] de t_personne ;
```

1. Question : Écrire la procédure *permuter()* qui permute les contenus respectifs de p1 et p2 de type t_personne avec un minimum d'instructions.
2. Question : Écrire une procédure *afficherPersonne()* qui affiche à l'écran le nom de la personne p et la liste de ses numéros de comptes bancaires.
3. Question Écrire une procédure *triNaif()* qui trie le tableau t sur le nom (par ordre croissant) suivant la méthode dite du « tri naïf » :
   - on cherche le plus petit élément du sous-tableau t[i…nbPers] et on l'échange avec en t[i] ;
   - on commence avec i=1, et on continue en faisant progresser i.
     remarque : le nombre d'éléments valides du tableau sera un paramètre de la procédure
4. Question : Écrire une procédure *afficherTableau()* qui affiche à l'écran les nbPers premiers éléments de t.
5. Question : Écrire un programme principal permettant :
   - de définir une constante BANQUE de type t_tabPersonne et de lui affecter les valeurs suivantes :
     "toto",{21,25,12}
     "dupont",{56}
     "albert ",{19,123,111}
     "alfred",{20,321}
   - d'afficher banque
   - de trier banque sur le nom
   - d'afficher banque triée

TABLEAUX ET TRIS : tableau à 2 dimensions

Considérons les déclarations suivantes :

```
constante entier MAX :=1000 ;
type t_matrice = tableau[MAX, MAX] de entier ;
```

1. La matrice m sera initialisée selon les règles suivante :
   - si l=c, m[l,c] = l
   - si l>c, m[l,c] = m[l-c , c] et m[c,l]=m[l,c]
2. Représenter la portion 10x10 d'une matrice de type t_matrice

l\c	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

a. *Que peut-on dire de cette matrice ?*

3. Écrire la procédure *initialiser()* qui permet initialiser une matrice de type t_matrice :

4. On définit le PGCD de trois nombres PGCD(n1,n2,n3) comme étant le PGCD du PGCD(n1,n2) et de n3.

En utilisant cette propriété écrire la fonction retournant le PGCD d'une série de nombres qui auront été préalablement saisis dans un tableau.

PILES

Lors de l'exécution d'un programme, une structure de pile est utilisée pour retenir dans quel ordre des fonctions (ou procédures) ont été appelées. Cela permet en particulier au système de savoir, une fois l'exécution d'une fonction terminée, à quelle fonction il doit retourner. Par exemple :

- Début systématique de l'exécution par la fonction main() ;
- Appel d'une fonction menu() ;
- Dans cette fonction menu, appel d'une fonction saisir() ;
- Une fois la fonction saisir() terminée, retour à la fonction menu() ;
- Une fois la fonction menu() terminée, retour au main() ;
- Une fois le main() terminé, il n'y a plus de fonction dans la pile et le programme s'achève.

Chaque fonction est décrite à l'aide d'une structure de type t_fct de la manière suivante :

```
constante entier N := 30 ;
type t_fct = structure
début
        c_nomF : chaîne(N);
        c_nbParametres : entier ;
fin
```

On propose la définition suivante d'une pile statique permettant de gérer l'exécution d'un programme :

```
constante entier MAXF := 20 ;

type t_tabFct := tableau [MAXF] de t_fct ;

type t_tpile = structure
début
        c_pile : t_tabFct ;
        c_ip : entier ; // indice de la fonction située au sommet de la pile représentée par le tableau pile
fin
```

1. Proposez l'écriture des procédures et fonctions ci-dessous pour gérer cette pile de fonctions.
    a. *Question : Fonction init() permettant la création et l'initialisation d'une pile.*
    b. *Question : Fonction pileVide() retournant vrai si la pile est vide, faux sinon.*
    c. *Question : Procédure depiler() permettant de supprimer une fonction f dans une pile p, charge à cette procédure de vérifier si la pile p n'est pas vide et d'en informer l'utilisateur le cas échéant. Si la fonction dépilée est le « main », la procédure vérifiera que la pile est désormais vide. Dans ce cas, elle affichera le message « exécution terminée ». Dans le cas inverse elle affichera « erreur d'exécution ».*
    d. *Question : Procédure fonctionsEnCours() permettant d'afficher le nom de la fonction en cours d'exécution*

RECURSIVITE

1. Proposer une procédure récursive inverse() qui lit une suite d'entiers (terminée par –1) au clavier et les affiche dans l'ordre inverse de l'ordre dans lequel ils ont été saisis.
2. Écrire et tester une fonction qui délivre le PGCD de deux entiers naturels en appliquant l'algorithme d'Euclide :

- si n1>n2 PGCD(n1, n2)=PGCD(n1-n2, n2)
- si n2>n1 PGCD(n1,n2) =PGCD(n1, n2-n1)

    a. *Question : Proposez une solution itérative avec une boucle tant que. La boucle s'arrête quand n1 et n2 deviennent égaux.*
    b. *Question :Proposez une solution récursive.*

3. récursivité croisée :
Soient les suites (un) et (vn) définies par :

```
u0 = 1
u1 = vn-1 + 2un-1

v0 = 1
vn =4vn-1 + un-1
```

    a. *Question : Calculer u3 et v3 « à la main ».*
    b. *Question :Écrire une procédure récursive qui permet de calculer un et vn. L'entête de la procédure sera le suivant :*

```
Procédure suite (sortF un : entier, sortF vn: entier, entF n : entier)
```