Examen d'informatique INFO304
Durée de l'épreuve : 1heure 30
Documents autorisés

21 02 11

Remarque générale : On tiendra compte dans la notation de la simplicité des programmes. TOUS LES PROGRAMMES, FONCTIONS ET PROCEDURES SERONT ECRITS EN LANGAGE ADA.

1. *Itératif ET récursif (4 points)*

On considère les déclarations suivantes

```
type noeud;
type ptnoeud is access noeud;
type noeud is
record
        x, y : integer;
        b : Boolean;
        suiv : ptnoeud;
end record;
```

Ecrire une fonction ***Function Complexe (TDL : in ptnoeud) return integer***
qui compte le nombre de nœuds pour lesquels le booléen b est vrai moins les valeurs de x quand elles sont strictement supérieures à y. Ces valeurs sont contenues dans la liste pointée par TDL. Effectuer ce calcul **de manière itérative puis récursive.**
Par exemple, si la liste de nombres contient les valeurs (3,2,false) (5,4, true) (1,6, true) (8,7, false) (4,5, true) (5,7,false) (4,0, false), la fonction rendra 3-(3+5+8+4) = -17

Itératif

Récursif

2. listes d'attente et piles (4 points)

On se propose d'appliquer des traitements particuliers à certaines suites de nombres positifs entrés au clavier par l'utilisateur. Une suite est terminée par -100.
Ces traitements sont appliqués dès que l'on rencontre -1, -2 ou -3.
-1 affiche dans l'ordre toutes les multiples de 10 rencontrés jusqu'à maintenant.
-2 affiche à l'envers toutes les nombres terminés par 5 rencontrées jusqu'à maintenant.
-3 supprime tous les nombres rencontrés jusqu'à maintenant et non traités

Exemple à lire attentivement pour voir si vous avez bien compris l'énoncé :
Si l'utilisateur entre ***10 15 20 -1 284 155 -2 15 10 20 -3 12 58 100 -1 -100***
Le programme affichera ***10 20 155 15 100***

Contraintes
Il est interdit de stocker la phrase dans un tableau. Pour simplifier le codage de cette application, vous devez utiliser les packages *listesAttente et Piles*.

Le package "listesAttente" est un package qui permet de disposer de
- un type ***listeAttente*** privé (et dont chaque élément est un entier)
- une procédure ***initListe (L : out listeAttente)*** qui crée une liste d'attente vide
- une fonction ***listeVide (L : in listeAttente) : boolean*** qui rend vrai si la liste est vide, faux sinon
- une procédure ***ajouter (L : in out listeAttente; x : in integer)*** qui permet d'ajouter l'entier x à la fin de la liste L
- une procédure ***retirer (L : in out listeAttente ; x : out integer)*** qui enlève l'entier le plus ancien de la liste L et qui le range dans x.

Le package "Piles" est un package qui permet de disposer de
- un type ***Pile*** privé (et dont chaque élément est un entier)
- une procédure ***initPile (P : out Pile)*** qui crée une liste d'attente vide
- une fonction ***PileVide (P : in Pile) : boolean*** qui rend vrai si la pile est vide, faux sinon
- une procédure ***empiler (P : in out Pile; x : in integer)*** qui permet d'empiler l'entier x sur la pile P
- une procédure ***depiler (L : in out Pile ; x : out integer)*** qui enlève l'entier au sommet de pile et qui le range dans x.

Ecrire la procédure réécrit qui réécrit une suite de nombres selon la méthode décrite ci-dessus tout en respectant les contraintes indiquées.
(N'oubliez pas d'initialiser la liste d'attente et la pile et de mettre quelques commentaires)

Procedure Reecrit is

3. *Spécification et corps de package (6 points)*

On donne le type

Vecteur is array(1..N) of float

Ecrire la spécification d'un package qui permet de traiter des listes circulaires (vues en TD) de « vecteur » et permettant de

- fournir le type liste_de_vecteurs sans que l'on sache comment il est constitué
- fournir la constante liste_vecteurs_vide
- fournir la fonction qui compte le nombre de vecteurs dont la première coordonnée est égale à la deuxième.

- fournir une procédure permettant d'insérer un vecteur en tête de liste

Package ListCircuVecteur is

Private

Ecrire le corps de ce package

4. *Arbres (6 points)*

On considère la structure suivante

```
Type nœud;
Type arbre is access nœud;
Type nœud is
Record
        Val : integer;
        Fg,fd : arbre;
End record;
```

Ecrire de manière récursive une fonction qui, étant donnés un objet de type arbre et une valeur entière x, donne la somme de toutes les valeurs de l'arbre qui sont supérieures ou égales à x.

Function somme_sup (a : arbre ; x : integer) return integer is

Begin

End somme_sup;

Ecrire de manière récursive une fonction qui compare deux objets de type arbre. Un arbre A est inférieur à un arbre B s'il a moins de nœuds que B. Dans le cas où les deux arbres ont le même nombre de nœuds, on compare la somme de tous les éléments de A et celle de tous les éléments de B.

Ecrire une function récursive qui affiche toutes les feuilles d'un arbre