|  |  |
| --- | --- |
| Ponctuel 3 | **La récursivité (algorithmes)** |

Nom :Antonin Tritz Groupe SIM

1. Soit l'algorithme récursif suivant:

procédure **mystere** ( n )

si n > 0 alors

afficher "Ici on a n=", n

mystere ( n – 1 )

afficher "Et là on a n=", n

fin si

fin procédure

Dites ce qui sera affiché à la suite des appels suivants:

* 1. mystere ( 0 )

RIEN NE S’AFFICHE

RIEN NE S’AFFICHE

* 1. mystere ( 1 )

Ici on a n= 1

Et là on a n= 1 // pas 0

* 1. mystere ( 2 )

Ici on a n=2

Ici on a n=1

Et là on a n=1

Et là on a n=2

* 1. mystere ( 3 )

Ici on a n=3

Ici on a n=2

Ici on a n=1

Et là on a n=1

Et là on a n=2

Et là on a n=3

1. Soit l'algorithme récursif suivant:

fonction entière **surprise** ( n )

si n <= 0 alors

retourner 55

sinon

retourner ***1 +***surprise ( n – 1 )

fin si

fin fonction

a) Dites ce qui sera affiché à la suite des appels suivants:

afficher surprise ( 1 )

56

afficher surprise ( 3 )

58

b) Ensuite, supposez qu’on modifie l’appel récursif ainsi

*retourner surprise ( n – 1 ))*

et répondez de nouveau à la question a).

55

55

1. Dites ce que retourne la fonction récursive suivante après chacun des appels donnés.

fonction entière **quiz** ( base, limite)

si ( base = limite ) alors

retourne ( 1 )

sinon

si ( base > limite ) alors

retourne ( 0 )

sinon

retourne ( base + quiz ( base + 1, limite ) )

fin si

fin si

fin fonction

a) r1 🡨 quiz ( 5, 5 ) r1 vaut 1

b) r2 🡨 quiz ( 10, -7 ) r2 vaut 0

c) r3 🡨 quiz ( 0, 3 ) r3 vaut 4

1. Considérez l’algorithme de la fonction récursive suivante qui calcule *n1 – n2*. L'instruction « afficher » n'est là que pour vous aider à suivre le déroulement de l'exécution.

fonction entière **difference** ( n1, n2 )

afficher n1, n2

si (n2 = 0) alors

retourner( n1 )

sinon

retourner ( difference ( n1 – 1, n2 – 1) )

fin

fin fonction

1. Indiquez ce qui sera affiché à la suite de l’appel à *difference ( 7, 4 )*

*7 4 puis 6 3 puis 5 2 puis 4 1 puis 3 0*

1. Donnez les deux premiers et les deux derniers affichages qui seront produits par l'appel de *difference (238, 100)*

*238 100 puis 237 99 puis 236 98 … 139 1 puis 138 0*

1. Qu'arrive-t-il si n2 <= 0 ?

Si n2 est égal à 0, le programme retourne n1, s’il est inférieur le programme roulera sans s’arrêter.

1. Soit la fonction suivante, qui prend en paramètre un vecteur (*vec*) ainsi que le nombre d’éléments qu’on veut considérer (*nb*).

fonction entière **calcul** ( vec, nb )

si nb = 0 alors

retourner 0

sinon

si vec [nb-1] > 0 alors

retourner 1 + calcul ( vec, nb-1 )

sinon

retourner calcul ( vec, nb-1 )

fin si

fin si

fin fonction

1. En supposant que le tableau ventes contient les valeurs suivantes :

200 -250 100 -400  500 650 175

Que donnait l’expression *calcul ( ventes, 7 )* ?

b) Décrivez en quelques mots seulement ce que fait la fonction *calcul* :

Cette fonction calcul le nombre de valeur positive que contient le vecteur.

1. En vous inspirant de l’exercice précédent, écrire l’algorithme d’une fonction récursive *somme* qui calcule et retourne la somme des éléments d’un vecteur, dont le nombre d’éléments à considérer est passé en paramètre.

fonction entière **somme** ( vec, nb )

if (nb = 1)

|  |
| --- |
| Return vec[nb-1] |
| Sinon |
| Return vec[nb-1]+somme (vec, nb-1); |
| Fin Si |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

fin fonction

1. On désire écrire une fonction récursive booléenne nommé *identique* qui retourne vrai si et seulement si tous les éléments du vecteur passé en paramètre sont identiques.

Parmi les solutions proposées ci-dessous, dite laquelle ou lesquelles sont correctes.

|  |
| --- |
| a)  fonction booléenne **identique** ( vec, nb )  si **nb=1**  alors  retourner **vrai**  sinon  retourner vec [nb-1] = vec [nb-2] **ou**  identique ( vec, nb-1 )  fin si  fin fonction |
| b)  fonction booléenne **identique** ( vec, nb )  si **nb=0**  alors  retournrer **vrai**  sinon  retourner vec [nb-1] = vec [nb-2] **et**  identique ( vec, nb-1 )  fin si  fin fonction |
| c)  fonction booléenne **identique** ( vec, nb )  si **nb=1**  alors  retourner **vrai**  sinon  retourner vec [nb-1] = vec [nb-2] **et** identique ( vec, nb-1 )  fin si  fin fonction |
| d)  fonction booléenne **identique** ( vec, nb )  si **nb=1** alors  retourner **faux**  sinon  retourner vec [nb-1] = vec [nb-2] **et**  identique ( vec, nb-1 )  fin si  fin fonction |

La C.

1. Les tours de Hanoï

Ce jeu est composé d'un plateau sur lequel se trouvent trois tours (appelées A, B et C) et *n* disques avec un trou au milieu. Les disques sont empilés par taille décroissante (c'est-à-dire d'abord celui de rayon 8 cm, puis celui de 7 cm, et ainsi de suite sur la tour A.  
  
On ne peut déplacer qu’un disque à la fois et il ne peut être placé sur un disque plus petit de la même tour. L'objectif est déplacer tous les disques de la tour A vers la tour C, un disque à la fois, en respectant la règle énoncée précédemment.  
  
Écrire l'algorithme qui permet d’afficher chacun des déplacements qui permettront de déplacer un nombre quelconque de disques de la tour de départ à la tour d'arrivée.

Main{

entier nb = "Entrer le nombre de bloc de votre tour : "

hanoi (nb, "A", "B", "C");

}

void hanoi(int nb, String de, String s, String a) {

si (nb = 0)

return

fin si

sinon

hanoi(nb - 1, de, a, s)

Afficher "Déplacer le disque supérieur de " + de + " à " + a

hanoi(nb - 1, s, de, a)

fin sinon

}

}