Algorithmique et structures de données
TP2 – récursivité et tris performants

Lamia BENAMARA

Consigne :
Pour les séances TP, le langage de programmation utilisé sera le java.
Pensez toujours à soigner vos traces d'exécution.
Pensez à tester chaque fonction dans votre programme principal.

Exercice 1 : Itératif vs récursif

1. Ecrire la fonction int somme_recusif(int n) qui prend en paramètre un entier n, et retourne la somme des n premiers entiers si n > 0, -1 sinon.

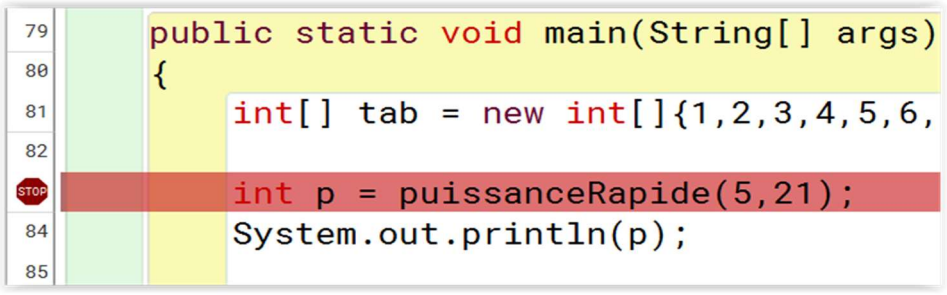
2. Ecrire une nouvelle fonction récursive int somme_recursif_bis(int n, int res) qui calcule la même somme, mais cette fois-ci, la valeur retournée ne doit pas faire l'objet d'un calcul intermédiaire(e.g. return n + somme_recursif(..)), mais seulement d'un appel récursif « pur » (e.g. return somme_recursif(...n-1...);

3. Une fois que cette fonction est testée et fonctionne bien, observer le comportement de la pile d'appels des fonctions.
   - le comportement est-il le même qu'avec la première fonction récursive?
   - si non, comment expliquer la différence de comportement ?

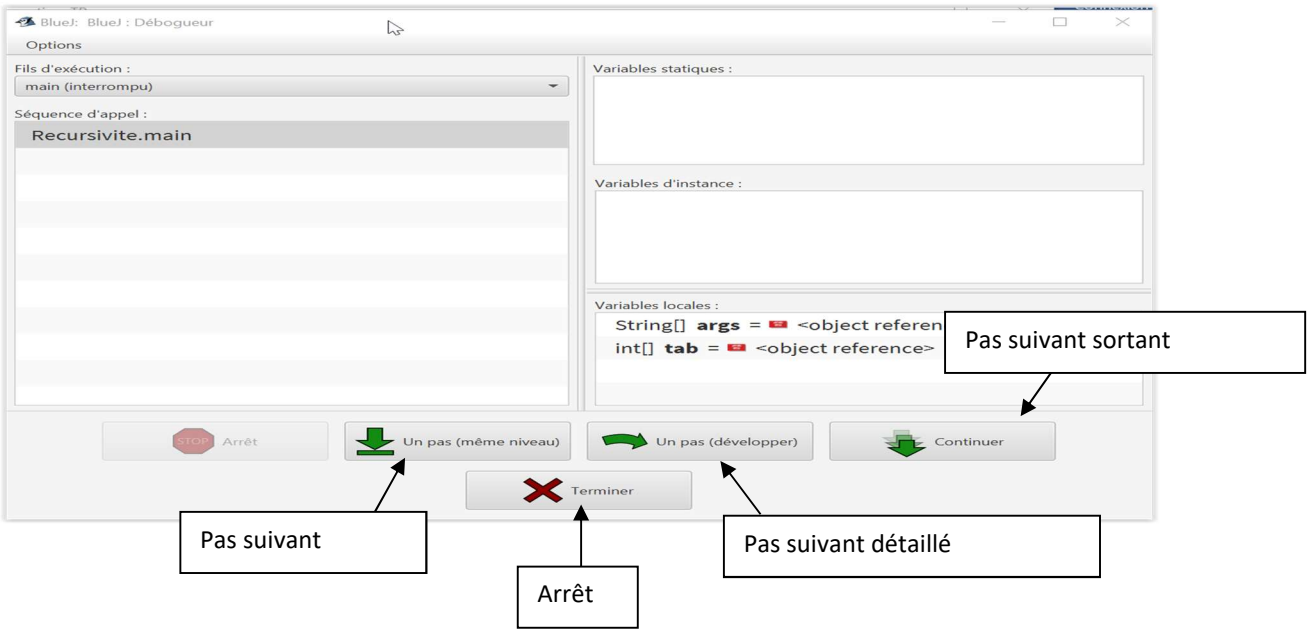
Utilisation du Débogueur

Le débogueur avec BlueJ : suivre les étapes suivantes :

- positionner un breakpoint (point d'arrêt) devant l'appel de la fonction souhaitée (juste à côté du numéro des lignes), pour enlever le point d'arrêt il faut recliquer sur le même point.



- Allez sur l'interface principale : Menu -> voir -> voir debogueur, vous aurez l'interface suivante :



- Lancer l'exécution de votre programme et choisissez entre les trois options pour avancer pas à pas :
  - "un pas (développer)" pour « rentrer » à l'intérieur de la fonction
  - "un pas (même niveau)" pour avancer d'une instruction sans détailler (i.e. rester au même niveau hiérarchique dans le code)

Pour les fonctions récursives demandées:

- lancer le programme en mode "Debug", puis avancer en pas à pas détaillé pour rentrer dans chaque appel successif de fonction
- observer l'état des variables et plus particulièrement la pile d'appel

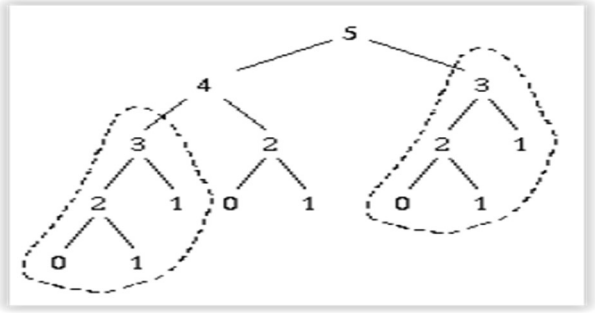
Exercice 2 : Suite de Fibonacci

1. Fibonacci récursif naïf

Concevoir un algorithme récursif **Fibo(*n* : entier) : entier** qui calcule et retourne le nième terme de la suite de Fibonacci.
Pour rappel : F0 = F1 = 1, Pour n > 1, Fn = Fn1+ Fn2

2. Fibonacci récursif malin

On peut constater qu'un algorithme récursif naïf pour calculer le nième terme de la suite de Fibonacci calcule plusieurs fois les mêmes valeurs.
Ex : F5 = F4 + F3 = (F3 + F2) + F2 = ((F2 + F1) + F2) + F2 = ...



Proposer une seconde solution utilisant algorithme récursif terminal.

Exercice 3 : Récursivité et tableaux

1. Écrire une fonction récursive recherche_max_recusif qui prend en paramètre un tableau d'entiers et sa taille, et retourne la position de l'élément maximum dans le tableau.
2. Écrire les deux fonctions somme_recusif et somme_recusif_terminal qui prennent en paramètre un tableau de réels et sa taille, et retourne la somme des éléments du tableau.

Exercice 5 : Tris et recherche en récursif

Implémentez en les algorithmes vus en TD :

1. Recherche dichotomique récursive
2. Tri par fusion
3. Tri rapide