ANNEE UNIVERSITAIRE 2017 / 2018

SESSION 1 D'AUTOMNE

Parcours : IGM501 (Ingénieurie Mathématique)
UE : 4TGM502ETP, Programmation avancée pour le calcul scientifique
Date : 21/12/2017 **Heure : 14h30** **Durée :** 3h
Epreuve de : Mme Weynans
Documents autorisés : Oui
Nombre de pages : 3

Avant de commencer, lisez attentivement ce qui suit:

- A la fin de l'examen vous devrez envoyer vos programmes par email avec accusé de réception.
- Pour une bonne lisibilité, commentez systématiquement vos programmes. Une question sera considérée comme traitée si les instructions demandées et l'affichage de leurs résultats ont été programmés, sont compilables et fonctionnent à l'exécution. Tous les programmes doivent donc être validés par un test même si ce n'est pas demandé explicitement.
- Si vous avez traité certaines questions mais que votre programme ne produit pas les résultats demandés, ajoutez à l'endroit adéquat des commentaires pour décrire le problème rencontré.
- Chaque exercice sera traité dans un programme indépendant. Les programmes de chaque exercice seront structurés en trois fichiers: un fichier contenant le programme principal, un autre contenant éventuellement les fonctions associées et un troisième (header) les éventuelles classes et prototypes des fonctions.

Exercice 1 : **Pointeurs**

Ecrire une fonction qui a comme paramètres un tableau d'entiers de taille quelconque, la taille du tableau, et 2 pointeurs vers des entiers min et max. La fonction doit renvoyer dans les entiers pointés par min et max respectivement les plus petits et les plus grands entiers du tableau.

Exercice 2 : **Résolution d'équations différentielles**

1. Créer une classe `solution` qui contient comme données privées un tableau (vector) de réels `y` et un réel strictement positif `dt`.
2. Créer le constructeur par défaut, le constructeur par copie et le destructeur pour cette classe.
3. Créer un constructeur qui initialise la taille du tableau à 1 et affecte à l'unique case du tableau un réel passé en argument.
4. Ecrire une fonction membre de la classe qui affiche le contenu d'une instance de la classe.

5. Créer une fonction membre de la classe qui prend pour arguments un entier `nite` et une fonction `f`: $\mathbb{R} \to \mathbb{R}$, et qui renvoie l'instance de la classe elle-même. Cette fonction calcule les `nite` premières itérations de la méthode d'Euler explicite appliquée à la résolution de l'équation différentielle
$$y'(x) = f(x)$$
avec comme pas de temps `dt`, et comme terme initial le dernier élément du tableau `y`, et ajoute ces `nite` termes supplémentaires au tableau `y`.

6. Testez cette fonction sur un exemple.

7. Ecrire une fonction amie de la classe qui prend en argument une fonction `f`: $\mathbb{R} \to \mathbb{R}$, un temps T_f, deux pas de temps dt_1 et dt_2, une valeur initiale y_0. Cette fonction initialise deux instances de la classe avec la valeur y_0, remplit les tableaux `y` de ces deux instances avec les itérations successives de la méthode d'Euler explicite avec dt_1 et dt_2, jusqu'à atteindre le temps final T_f et renvoie la dernière itération calculée pour chaque instance.

8. Testez cette fonction sur un exemple.

Exercice 3 : **La suite de Syracuse**

Soit $\phi : \mathbb{N}^* \to \mathbb{N}^*$ l'application telle que $\phi(n) = \frac{n}{2}$ si n est pair et $\phi(n) = 3n + 1$ si n est impair. On appelle suite de Syracuse toute suite $(s_n)_{n\in\mathbb{N}}$ définie par son premier terme $s_0 \in \mathbb{N}^*$ et par la relation $\forall n \in \mathbb{N}$, $s_{n+1} = \phi(s_n)$.
La conjecture de Syracuse postule que $(s_n)_{n\in\mathbb{N}}$ finit toujours par atteindre la valeur 1 et répète le motif 1, 4, 2, 1, 4, 2.... En dépit de la simplicité de son énoncé, cette conjecture n'a pas encore été prouvée. Elle mobilisa tant les mathématiciens américains durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle ce problème faisait partie d'un complot soviétique visant à ralentir la recherche américaine (source Wikipedia).

1. Déclarer une classe `Syracuse` avec comme donnée privée un tableau (type vector) d'entiers nommé `suite`, qui contiendra les termes successifs d'une suite de Syracuse.

2. Définissez le constructeur par défaut et le destructeur pour cette classe.

3. Définissez un constructeur initialisant une instance de la classe, avec le tableau `suite` de taille 1, contenant un entier `init` qui sera passé en argument de ce constructeur.

4. Définissez une fonction membre de la classe qui prend en argument un entier `nite` et qui calcule les `nite` termes successifs de la suite de syracuse, calculés à partir du dernier élément du tableau `suite`. Cette fonction renvoie l'instance de la classe elle-même, avec le tableau `suite` auquel ont été rajoutés les `nite` nouveau termes de la suite qui viennent d'être calculés.

5. Définissez une fonction membre de la classe qui calcule les termes successifs de la suite de Syracuse, à partir du dernier terme du tableau `suite`, jusqu'à arriver à la valeur 1, et qui renvoie l'instance elle-même avec les termes ainsi calculés rajoutés au tableau.

6. Définir une surcharge de l'opérateur "-", qui prend en argument un entier `nb` et une instance de la classe, et qui renvoie une autre instance de la classe telle que les `nb` premiers termes de la suite ont été enlevés.

Exercice 4 : **Templates**

1. Créez une classe template, dont la donnée privée est un tableau pouvant contenir soit des entiers soit des réels.
2. Créez une fonction membre de la classe qui affiche les valeurs du tableau.
3. Créez une fonction membre de la classe qui renvoie l'instance de la classe elle-même avec son tableau trié par ordre croissant.
4. Testez ces différentes fonctions dans un programme principal, pour une instance contenant des entiers, et une autre contenant des réels.

Exercice 5 : Makefile

1. Créer un makefile basique pour compiler les programmes des exercices précédent.
2. Créer un makefile avancé (utilisant les variables vues en cours) pour compiler les programmes des exercices précédent.

Lors de l'envoi des programmes, vous pourrez sauvegarder ces deux makefiles différents sous les noms 'Makefile1' et 'Makefile2'.