Projet de programmation: TP

1 BAC Informatique à horaire décalé

Année académique 2014-2015

1. C

Exercice 1 : Ecrivez le programme C le plus petit possible.

Exercice 2 : Ecrivez un programme C qui lit un nombre au clavier et affiche sa valeur opposée.

Exercice 3 : Ecrivez un programme qui affiche le chiffre des dizaines, ainsi que le chiffre des centaines, d'un nombre saisi au clavier.

Exercice 4 : Implémentez en C l'algorithme qui permet de saisir deux nombres A et B et qui affiche VRAI si les deux nombres sont de même signe, et FAUX dans le cas contraire.

Exercice 5 : Écrivez un programme qui effectue les tâches suivantes :

1. Il lit deux entiers n_1 et n_2 au clavier ;
2. Il affiche la partie entière de leur quotient (div) ;
3. Il affiche la partie fractionnaire q de leur quotient ;
4. Il lit un nombre réel l au clavier ;
5. Il calcule la partie entière (trunc) du produit de l par q modulo 256 (mod), puis convertit le résultat en un caractère (chr) ;
6. Il affiche finalement le caractère obtenu.

Testez votre programme avec les valeurs suivantes : $n_1 = 1321$, $n_2 = 500$ et $l = 500$.

2. Boucles

Exercice 6 : Ecrivez les programmes qui permettent de saisir un entier positif x et affichent les x premiers éléments des suites de terme général suivantes :

1. $S_n = 2n$
2. $T_n = \frac{1}{n}$
3. $U_n = n^2$
4. $V_n = \frac{n(n-1)}{n}$

Exercice 7 : Ecrivez un programme pour découvrir un nombre entier choisi aléatoirement (random) entre 1 et 100.

Exercice 8 : Ecrivez un programme qui permet de saisir un entier positif n et calcule le résultat de la série :

$$\sum_{i=0}^{n} \frac{1}{2^i}$$

Exercice 9 : On se propose d'étudier la suite de terme général :

$$U_n = \frac{1}{n^2}$$

définie pour tout n entier positif non nul, ainsi que la suite définie par :

$$V_n = \sum_{k=1}^{n} U_k$$

définie pour tout n entier positif non nul. Pour ce faire, on souhaite écrire un algorithme qui affiche, ligne par ligne, la valeur de n, celle de U_n et celle de V_n et ce, tant que le terme U_n est supérieur à $0,001$. Écrivez cet algorithme en C.

3. Tableaux

Exercice 10: Ecrivez un programme qui initialise chaque élément tab[i] d'un tableau tab à la valeur 2^i. La valeur maximale de l'indice i est 31.

Exercice 11 : Ecrivez un programme qui remplit un tableau comprenant 50 cellules avec des valeurs aléatoires comprises entre 0 et max. max est saisi par l'utilisateur au début du programme. Le programme affiche ensuite la valeur minimale, la valeur maximale et la valeur moyenne stockées dans le tableau.

Exercice 12 : Le crible d'Eratosthène permet de déterminer les nombres premiers inférieurs à une certaine valeur N. On place dans un tableau unidimensionnel Tab les nombres entiers compris entre 1 et N. L'algorithme consiste, pour chaque élément Tab(i) du tableau, à rechercher parmi tous les suivants ceux qui en sont des multiples et à les éliminer (i.e. : en les remplaçant par 0).

Lorsque l'ensemble du tableau a subi ce traitement, seuls les nombres premiers du tableau n'ont pas été éliminés. Écrivez l'algorithme de ce crible et traduisez-le en C en recherchant par exemple les nombres premiers inférieurs à 500.

Exercice 13 : Ecrivez un programme qui recherche les suites monotones croissantes dans un tableau et affichent les indices de début et fin de chacune de ces suites ainsi que le longueur de la plus longue d'entre elles. Testez votre programme pour un tableau comprenant 50 valeurs aléatoires comprises entre 0 et max. max est saisi par l'utilisateur au début du programme.

Exercice 14 : Ecrivez un programme remplissant un tableau indexé à l'aide d'un type d'énumération avec le nombre de jours de chacun des mois de l'année saisie par l'utilisateur. Pour rappel, les années bissextiles sont divisibles par 4 ; les années divisibles par 100 et non par 400 ne sont pas bissextiles (1900 n'est pas bissextile mais 2000 bien).

4. Fonctions

Exercice 15 : Ecrivez un programme qui appelle une fonction pour afficher le contenu d'un tableau déclaré comme variable globale.

Exercice 16 : Même énoncé que l'exercice 15 mais le tableau est passé en paramètre à la fonction.

Exercice 17 : La factorielle d'un nombre n (n!) peut être définie comme :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n(n-1) & \text{si } n > 0 \end{cases}$$

avec n $\in$ N. On vous demande d'écrire une fonction qui calcule la factorielle de n. La solution au problème doit s'obtenir en utilisant la récursivité. Votre programme principal permettra à l'utilisateur de saisir une valeur pour n et fera appel à cette fonction.

Exercice 18 : La suite de Fibonacci est définie comme :

$$v_n = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ v_{n-1} + v_{n-2} & \text{si } n \geq 2 \end{cases}$$

avec n $\in$ N. On vous demande d'écrire une fonction qui calcule le $n^{ième}$ terme de la suite de Fibonacci. La solution au problème doit s'obtenir en utilisant la récursivité. Le programme principal fera appel à cette fonction.

Exercice 19 : Même énoncé que l'exercice 17 mais sans utiliser la récursivité.

Exercice 20 : Même énoncé que l'exercice 18 mais sans utiliser la récursivité.

5. Pointeurs

Exercice 21 : Ecrivez un programme C manipulant (i.e. : incrémentation) le contenu d'un variable entière à l'aide d'un pointeur.

Exercice 22 : Ecrivez un programme pascal comprenant 2 fonctions changeant le signe d'un entier qui lui est passé en paramètre. Le programme affichera le contenu de l'entier avant et après l'appel aux 2 fonctions. Le paramètre de la première fonction sera passé par valeur, celui de la seconde fonction sera un pointeur vers l'entier. Qu'observez-vous ?

Exercice 23 : Ecrivez un programme C censé modifier la valeur d'un entier déclaré comme constante au début de programme. Ce programme va-t-il être accepté par le compilateur C et si oui va-t-il fonctionner ?

Exercice 24 : Ecrivez un programme C affichant « l'adresse » d'une variable globale entière en sachant que printf(&x) retourne une erreur de compilation.

Exercice 25 : Ecrivez une fonction C qui calcule la somme des éléments d'un tableau d'entiers positifs de longueur inconnue. La fin du tableau est indiquée par la valeur -1. Ecrivez également un programme appelant cette procédure.

6. Allocation dynamique

Exercice 26 : Ecrivez un programme C qui alloue dynamiquement une variable entière et y stocke un nombre saisi par l'utilisateur.

Exercice 27 : Ecrivez un programme C qui alloue dynamiquement l'espace mémoire nécessaire pour stocker les nombres entiers positifs entrés par l'utilisateur. Le programme s'arrête lorsque qu'un nombre strictement négatif est entré.

7. Structures et pointeurs

Exercice 28 : Ecrivez un programme C qui permet de stocker un nombre fini d'enregistrements contenant le nom, le prénom et l'âge d'un utilisateur dans un tableau.

Exercice 29 : Modifiez le programme obtenu dans l'exercice 28 afin que les enregistrements soient triés par ordre alphabétique sur le prénom.

Exercice 30 : Comment modifieriez-vous le programme obtenu à l'exercice 29 afin de pouvoir également trier les enregistrements par âge sans perdre les informations sur l'ordre alphabétique ?

8. Listes chainées

Exercice 31 : Même énoncé que l'exercice 27 mais en utilisant une liste simplement chaînée.

Exercice 32 : Ecrivez une procédure récursive inversant l'ordre d'une liste simplement chaînée.

Exercice 33 : Ecrivez un programme C qui lit des chaînes de caractères (*string*) entrées par l'utilisateur et stocke chacune de ces *strings* dans une liste chaînée triée. A la fin du programme, lorsque l'utilisateur entre le mot « quit », les *strings* sont affichés par ordre alphabétique.

Exercice 34 : Modifiez le programme obtenu à l'exercice 33 afin que si un mot est entré une seconde fois par l'utilisateur, il sera supprimé de la liste.

Exercice 35 : Même énoncé que l'exercice 33 mais en utilisant une sentinelle.

Exercice 36 : Ecrivez un programme qui permet à l'utilisateur d'entrer des informations sur des clients (nom, âge). Ces informations seront stockées dans une liste doublement chaînée. Une fois les informations encodées, le programme calcule et affiche l'âge moyen des clients.

Exercice 37 : Ecrivez une procédure qui fusionne deux listes simplement chaînées triées pour donner une liste simplement chaînée triée avec sentinelle mais sans allouer de mémoire supplémentaire.

9. Fichiers textes

Exercice 38 : Ecrivez un programme qui affiche le contenu d'un fichier texte dont le nom est passé en paramètre.

Exercice 39 : Ecrivez un programme qui affiche le carré des valeurs contenues dans un fichier texte. Ces valeurs sont des nombres entiers strictement positifs séparés par des espaces.

Exercice 40 : Ecrivez un programme qui lit des mots entrés par l'utilisateur et stocke chacun de ces mots dans un fichier texte (un mot par ligne).

Exercice 41 : Ecrivez un programme qui permet à l'utilisateur d'entrer des informations sur des clients (nom, âge). Quand l'utilisateur décide de quitter le programme, les informations à propos des clients sont enregistrées dans un fichier texte. Modifiez ensuite votre programme pour qu'au démarrage il charge depuis un fichier texte les informations entrées lors d'une exécution précédente du programme.

10. Fichiers à accès direct

Exercice 42 : Ecrivez un programme qui écrit dans un fichier les valeurs entières entrées par l'utilisateur. Le programme se termine lorsqu'il entre la valeur -1.

Exercice 43 : Ecrivez un programme qui lit un fichier comme celui produit à l'aide du programme de l'exercice 42 et affiche le carré des valeurs contenues dans ce fichier.

Exercice 44 : Ecrivez un programme qui stocke des informations entrées par l'utilisateur dans une liste chaînée simple (nom, âge). Avant de se terminer, le programme sauvegarde le contenu de la liste chaînée dans un fichier binaire. Au démarrage du programme, le contenu de la liste chaînée est initialisée avec le contenu du fichier.

11. Unité de code

Exercice 45 : Modifiez le programme de l'exercice 44 afin que les fonctions de gestion et d'accès à la liste chaînées soient placées dans une unité de code dédiée.

12. Types de données abstraits

Exercice 46 : Un *ring buffer* (aussi appelé *circular buffer* ou *cyclic buffer*) est un buffer de taille fixe dont les extrémités sont virtuellement connectées. Ce type de buffer est largement utilisé dans du code devant gérer des flux de données (souvent dans des drivers réseaux, séries, . . .). L'accès aux données contenues dans un *ring buffer* se fait selon le modèle FIFO. Un accès en lecture lorsque le buffer est vide ou un accès en écriture sur un buffer plein doivent pouvoir être détectés par le code appelant. Spécifiez et implémentez un type abstrait permettant d'utiliser un tel *ring buffer* stockant des nombres entiers.

Exercice 47 : Modifiez le type abstrait de l'exercice 46 afin que le code l'utilisant puisse choisir qu'une écriture dans un buffer plein soit est ignorée, soit résulte en la perte des informations les plus anciennes contenues dans le buffer et l'écriture des nouvelles à leur place.