Année 2017-2018
1ère année
Durée : 2h

Examen de programmation et algorithmes

Tous documents interdits, calculatrice interdite
Remarques

- Les questions sont indépendantes dans une large mesure. Vous pouvez utiliser les fonctions écrites dans les questions précédentes sans les réécrire.
- Si une question vous semble pouvoir être interprétée de plusieurs manières, indiquez celle que vous avez choisie.
- Toutes les fonctions et programmes doivent être rédigés **en langage C**.

Question 1 Calcul de série : 1+1+1points

.La valeur de π peut être calculée par la formule de Nilakantha :

$$\pi = 3 + \frac{4}{2\times3\times4} - \frac{4}{4\times5\times6} + \frac{4}{6\times7\times8} - \ldots = 3 + \sum_{i=1}^{n}(-1)^{i+1}\frac{4}{(2\times i)\times(2\times i+1)\times(2\times i+2)}$$

- Ecrire une fonction **`double Sn(int n)`** qui calcule et retourne le terme S_n de la série. Il est **interdit** d'utiliser la fonction **`pow(x,y)`**.
- Ecrire une fonction **`double Pi(double eps)`** qui calcule et retourne la limite de la série et donne donc la valeur de π. La limite est atteinte quand l'écart **relatif** **`|a-b|/(|a|+|b|)`** entre 2 termes consécutifs S_n et S_{n+1} est inférieur à la valeur du paramètre de précision **`eps`**. La fonction retournant la valeur absolue d'un réel est **`double fabs(double x)`**.
- Ecrire un **programme** qui lit la valeur de la précision voulue au clavier, calcule puis affiche la valeur de π pour cette valeur de paramètre de précision

Question 2 Paramètres de fonctions : 1+1+1+1 points

- Ecrire une fonction qui calcule le produit de 2 réels **`a`** et **`b`**. Cette fonction prend 3 paramètres : les deux réels **`a`** et **`b`**, ainsi qu'une variable pour stocker le produit. **La fonction ne retourne aucune valeur**. Cette fonction ne fait aucun affichage à l'écran.
- Ecrire une fonction qui calcule et **retourne** la somme de 2 réels **`a`** et **`b`**. Les deux réels **`a`** et **`b`** sont passés en paramètres. La fonction retourne la somme et ne fait aucun affichage à l'écran.
- Ecrire une fonction qui calcule la somme et le produit de 2 réels **`a`** et **`b`**. Cette fonction prend 4 paramètres : les deux réels **`a`** et **`b`**, ainsi que deux variables pour stocker les deux résultats (somme et produit). Cette fonction ne fait aucun affichage à l'écran. **Vous devez utiliser les deux fonctions précédentes pour faire cette fonction**.
- Ecrire un programme qui lit 2 réels au clavier, calcule la somme et le produit de ces 2 réels **en utilisant la fonction précédente**, puis affiche ces valeurs.

Question 3 Matrice et carré latin: 1+0,5+1+1+1,5+1,5+1+1,5+2+2 points

Un carré latin d'ordre **`n`** est une matrice **`n`** x **`n`** d'entiers compris entre 0 et **`n-1`**, et telle que chaque ligne et chaque colonne contiennent tous les entiers de 0 à **`n-1`** une fois et une seule. La matrice ci contre est un carré latin d'ordre n=4.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 3 & 2 & 0 & 1 \\ 2 & 3 & 1 & 0 \end{bmatrix}$$

Fonctions d'allocation/libération de matrice :

- Ecrire la fonction **`int** alloueMatrice(int nl, int nc)`** qui alloue une matrice de **`nl`** x **`nc`** d'entiers. Cette fonction retourne l'adresse du tableau ainsi alloué ou NULL si l'allocation est impossible. L'allocation des données se fera de manière contiguë.
- Ecrire la fonction **`void libereMatrice(int** mat)`** qui libère la matrice **`mat`** de **`nl`** x **`nc`** d'entiers.

Test des lignes et colonnes
Pour savoir si tous les nombres de **`0`** à **`n-1`** sont présents une fois et une seule sur une ligne (**`n=nc`**), respectivement sur une colonne (**`n=nl`**), on utilise un histogramme : c'est un tableau appelé **`hist`** de **`n`** entiers qui contient le nombre d'occurrences de chaque entier dans la ligne (resp colonne). **`hist[i]`** doit contenir le nombre de fois que le nombre **`i`** est trouvé dans la ligne (resp. colonne). Ce tableau est alloué

dynamiquement au début de la fonction qui teste si une ligne contient une fois et seule tous les nombres et est libéré à la fin de cette fonction.

- Ecrire la fonction **`int testeLigne(int** mat, int nl, int nc, int k)`** qui teste si la ligne **`k`** de la matrice **`mat`** de **`nl`** x **`nc`** entiers contient une fois et une seule les nombres de **`0`** à **`nc-1`**. Elle retourne 1 si cette condition est vérifiée, 0 sinon. Elle utilise un histogramme intermédiaire alloué dynamiquement au début et libéré à la fin de la fonction.
- Ecrire la fonction **`int estLatin(int** mat, int n)`** qui teste si la matrice carrée **`mat`** de **`n`** x **`n`** entiers est un carré latin. Elle retourne 1 si cette condition est vérifiée, 0 sinon. **ATTENTION** : La fonction **`int testeColonne(int** mat, int nl, int nc, int k)`** qui teste si la colonne **`k`** de la matrice **`mat`** de **`nl`** x **`nc`** entiers contient une fois et une seule les nombres de **`0`** à **`nl-1`** est quasiment identique à la précédente et **n'est** donc **pas** à écrire. Vous pouvez l'utiliser **SANS** l'avoir écrite

Génération d'un carre latin trivial :
On peut engendrer un carré latin trivial par permutation circulaire des lignes. On peut remarquer que la valeur de l'élément (**`i,j`**) est égale à **`i+j`** modulo **`n`** (**`%`** est l'operateur modulo en C)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}$$

- Ecrire la fonction **`int** engendreCarre(int n)`** qui retourne un carré latin de **`n`** x **`n`** entiers. La matrice est allouée dynamiquement. La fonction retourne NULL si l'allocation est impossible.

Sauvegarde dans un fichier

- Ecrire la fonction **`void sauveTexte(char* nomfic, int** mat, int n)`** qui sauvegarde dans le fichier texte dont le nom est donné par **`nomfic`** la matrice carrée **`mat`** de **`n`** x **`n`** entiers. Le format du fichier texte sera le suivant : sur la première ligne, on écrira la valeur de **`n`**. Puis on écrira les valeurs de la matrice carrée **`mat`** ligne par ligne. Ne pas oublier de fermer le fichier à la fin de la fonction

Programme principal

- Ecrire un programme principal qui réalise les actions suivantes
  - Lecture au clavier de la dimension du carré
  - Génération du carre trivial
  - Afficher le résultat de la vérification du carré par la fonction estLatin.
  - Sauvegarder le carré dans le fichier "moncarre.txt".

Un peu plus difficile

- Ecrire la fonction **`int testeLignePointeur(int** mat, int nl, int nc, int k)`** : cette fonction est identique à la fonction **`testeLigne`**, mais vous ne devez pas utiliser d'entiers pour l'écrire. Les parcours de la ligne et de l'histogramme se font **uniquement avec des indices pointeurs, aucun entier n'est autorisé**.

- Ecrire la fonction **`int testeColonnePointeur(int** mat, int nl, int nc, int k)`** : cette fonction est identique à la fonction **`testeLignePointeur`**, mais pour les colonnes.

- Ecrire la fonction **`int** chargeBinaire(char* nomfic, int* pn)`** qui réalise la lecture d'une matrice carrée stockée dans un fichier de nom **`nomfic`**, en mode binaire et non en mode texte. Le format du fichier est le suivant : un entier contenant la taille du carré, puis les valeurs entières de la matrice carrée. Elle utilise donc la fonction **`fread`** pour la lecture. Elle retourne la matrice allouée dynamiquement. Elle utilise le paramètre **`pn`** qui sert à obtenir la dimension de cette matrice carrée. Cette fonction retourne **`NULL`** si la lecture est impossible.

Résumé de la syntaxe du langage C

Structure d'un programme

```
int main() {
  déclaration des variables ;
  instructions ;
}
```

Déclaration des variables : type_de_variable nom_de_variables ;
Exemples : int i ; double x ; char c ; char s1[256] ; int* p1 ; double* p2 ;

Entrées : scanf("%d ",&i) ; scanf("%lf", &x) ; scanf("%c", &c) ; scanf("%s", s1) ;
Sorties: printf("%d ",i) ; printf("%lf", x) ; printf("%c", c) ; printf("%s", s1) ;

Boucle POUR:

```
for (<initialisation>; <test_continuation>; <incrementation>) {
    <instructions_a_repeter> ;
}
```

Exemple : for (i=0 ; i< 100 ; i++) { t1[i] = 2*i ; }

Boucle TANTQUE:

```
<initialisation>
while (<test_continuation>) {
    <instructions_a_repeter> ;
    <incrementation> ;
}
```

Exemple : while (i< 100) { t1[i] = 2*i ; i= i+1 ;}

Boucle FAIRE..TANTQUE:

```
<initialisation>
do {
    <instructions_a_repeter> ;
    <incrementation> ;
} while (<test_continuation>) ;
```

Exemple : do { t1[i] = 2*i ; i= i+1 ;} while (i< 100) ;

Instruction condionnelle: if (test) { instructions_si_test_est_vrai ; } else { instructions_si_test_est_faux; }
Exemple : if (a !=0) { c = b/a ; } else { printf("La division est interdite\n") ; }

Pointeurs et adresse: l'opérateur & permet d'obtenir l'adresse d'une variable, l'opérateur * permet d'obtenir la valeur située à une adresse; Exemple :

```
int i ;
int* p =NULL; /* p est une variable pointeur contenant une adresse*/
p=&i ; /* p contient maintenant l'adresse de i*/
*p = 9 ; /*On met 9 dans la variable situé à l'adresse donnée par p, ie on met 9 dans i*/
```

Fonctions : déclaration

```
type_retourné nom_de_la_fonction(declaration des parametres de la fonction) {
    Declaration des variables ;
    Instructions de la fonction ;
}
```

Exemple :

```
double max( double a, double b) { double c ;
    if (a>b) c =a ;
    else c=b ;
    return c ;
}
```

Tableau 1D: déclaration : type_des_elements nom_du_tableau[Nombre d'element] ;
Exemple : int t1[100] ; double x1[256] ;
Le nom du tableau est son adresse
L'adresse d'un élément du tableau s'écrit &t[i] ou t+i

Tableau 2D: déclaration : type_des_elements nom_du_tableau[Nombre_d'element][Nombre d'element] ;
Exemple : int t2d[100] [20]; double x2d[256] [3];

Déclaration d'un type structure : struct nom_de_la_structure { déclarations des champs ; } ;
Déclaration d'une variable de type structuré: struct nom_de_la_structure nom_de_la_variable;
Exemple : struct complex { double reel ; double imaginaire ; } ; /* Un type complexe*/
struct complex x ; /* x est une variable de type complexe */

Déclaration d'un nouveau type : typedef ancien_type nouveau_type ;
Exemple : typedef struct complex MesComplexes ;

Fichiers :
Declaration d'une variable : FILE* nom_de_variable ;
Ouverture :
FILE* fopen(char* nom_du_fichier_a_ouvrir, char* mode_d'ouverture) ;

Lecture fichier texte :
int fscanf(FILE* variable_fichier, char* format_des données_a_lire, adresses_des_données) ;

Ecriture fichier texte :
int fprintf(FILE* variable_fichier, char* format_des_variables_a_ecrire , variables_a_ecrire) ;

Lecture fichier binaire :
int fread(void* adresse_donnees, int taille_d'un_element, int nombre_d'element, FILE* variable_fichier_dans lequel on lit) ;

Ecriture fichier binaire :
int fwrite(void* adresse_donnees, int taille_d'un_element, int nombre_d'element, FILE* variable_fichier_dans lequel_on_ecrit) ;

Allocation dynamique
Allocation d'un espace :
void* calloc(int nombre_d'element_a_allouer, int taille_d'un_element) ;

Libération mémoire :
void free(void* adresse_de_la_zone_a_liberer) ;

Quelques fonctions sur les chaines de caractères et les copies mémoires
Lecture d'une chaine : scanf("%s",mot) : lecture d'une chaine au clavier. Remplace le caractère '\n' (« retour chariot ») par le caractère '\0' (marque de fin de chaine).

Longueur d'une chaine : int strlen(char* mot) : retourne le nombre de caractères de la chaine

Comparaison de 2 chaines : int strcmp(char* s1, char* s2) : retourne une valeur négative si s1<s2, nulle si s1 et s2 sont identiques, et positive si s1 > s2 : c'est l'ordre lexicographique qui est utilisé.

Copie d'une chaine : char* strcpy(char* s1, char* s2) : copie de la chaine s2 dans la chaine s1. La chaine s1 doit comporter suffisamment de place pour contenir la chaine s2.

Copie d'un tableau : void* memcpy(void* dst, void* src, int n) : copie n octets du tableau src dans le tableau dst.