Concours d'accès en première année du cycle d'ingénieurs pour les filières :

- **Génie du Logiciel et des Systèmes Informatiques Distribués (GLSID)**
- **Ingénierie Informatique, Big Data et Cloud Computing (II-BDCC)**

Session : Juillet 2018
Epreuve d'Informatique

Durée : 3 heures

Nom :	**Prénom :** .....
CIN :	**N° d'examen :** .....

Remarques importantes :

- Il est obligatoire de choisir votre ordre de préférence pour les deux filières **GLSID** et **II-BDCC** en précisant votre filière de premier choix et votre filière de deuxième choix dans le formulaire ci-dessous :

1er choix :	**2ème choix** : .....
Signature :	

- L'épreuve se compose de deux parties :
  - Partie QCM, **(Notée sur 60 points)**
  - Partie Algorithmique et Programmation, **(Notée sur 40 points)**
- L'usage de la calculatrice ou de tout autre appareil électronique est interdit.
- L'utilisation du blanco est strictement interdite.
- Aucun document n'est autorisé.
- Les extraits de code fournis dans l'épreuve sont écrits en langage C ou en langage Java.
- Aucune explication supplémentaire ne sera fournie aux candidats au cours de l'examen.
- Chaque question du QCM ne peut avoir qu'une seule réponse possible parmi les quatre choix. La réponse est à reporter dans la grille de réponses fournie (**page 15**) en cochant la case correspondante .
- Pour les exercices 1 et 2 :
  - Les réponses aux questions doivent être rédigées dans des feuilles de rédaction.
  - Les solutions algorithmiques peuvent être rédigées en utilisant un pseudo langage ou l'un des langages de programmation suivants : C, C++, C#, Java.
  - La clarté et la précision de votre solution algorithmique sera prise en considération.
- Sont à rendre la page de garde (**page 1**), la grille de reponses (**page 15**) et les feuilles de rédaction.

Partie 1- QCM (60 points)

1. Lequel des opérateurs suivants a la priorité la plus faible ?

   A) || B) |

   C) && D) &

2. Lequel des opérateurs suivants a une associativité de droite à gauche ?

   A) , B) <<

   C) && D) <<=

3. L'expression **`11U/22L*(3.75F-2)+3./6+.25/1.F`** est évaluée à :

   A) 0.5 B) 0.25

   C) 0.0 D) 0.75

4. Le type de l'expression **`3U/2*3.14F +1./1UL`** est :

   A) unsigned int B) double

   C) float D) long double

5. La variable **`x`** est de type **`short`**, l'expression **`x= 30*1000+2768`** est évaluée à :

   A) 32768 B) -32767

   C) -32768 D) 0

6. Quel affichage sera produit par l'exécution du programme suivant :

```
#include<stdio.h>
int main(){
     int x = 0x10+ 010+10;
     printf ("x = 0x%x", x);
     return 0;
}
```

   A) x = 022 B) x = 0x22

   C) x =x22 D) Erreur de compilation

7. Quel affichage sera produit par l'exécution du programme suivant :

```
#include<stdio.h>
int main(){
     int x =4, y =3*x--;
     if( x = !5|| y++>12)printf("x=%d,  y=%d\n",x,y);
     else printf("y= %d,  x=%d\n",y,x);
     return 0;
}
```

A) y=13, x=0 B) x= 1, y=13

C) x=1, y=10 D) y=10, x=0

8. Quel affichage sera produit par l'exécution du programme suivant ?

```
#include<stdio.h>
int main(){
     char x=2;
     for(;x++;);
     printf("x=%d\n",x);
}
```

A) x = 0 B) x = 2

C) x =1 D) x = -1

9. On considère la procédure suivante :

```
void f(int n){
     int i,a,b,c;
     a=0;b=2;c=0 ;
     for(i=0;i<n;i++){
          c=a+b;
          a=b+1;
          b=c+2;
     }
     System.out.print(a + " "+ b + " " + c);
     }
```

Pour n=5, l'appel de cette procédure permet d'afficher:

A) 23 37 35 B) 17 33 26

C) 48 78 76 D) 29 47 45

10. Soit la procédure suivante :

```
void f(int n) {
     int i, a=0, b=2, c=1;
     for (i = 0; i < n; i=i+2) {
          if (c % 2 == 1) i--;
          else c = a + b;
          a = 2 * b;
          b = c + 2;
     }
     System.out.print(a + " " + b + " " + c);
}
```

Pour n=8, le résultat affiché par la procédure est :

A) 8 3 1
B) 6 3 1
C) 12 5 3
D) 8 5 3

11. Soit la procédure suivante :

```
void f(int n){
     int i,j,a=0,b=0,c;
     for(i=0;i<n;i++){
          for(j=0;j<i;j++)
              a=a+i;
          b=b+a;
     }
     System.out.print(a +" "+ b);
}
```

Pour n=4, le résultat affiché par la procédure est :

A) 14 20
B) 15 22
C) 16 20
D) 14 22

12. Soit la fonction suivante :

```
int f(int n) {
     int[] T = new int[n + 2];
     T[0] = T[1] = 1;
     for (int i = 2; i <= n; i++) {
          T[i] = 0;
          for (int j = 0; j < i; j++)
              T[i] += T[j] * T[i - j - 1];
     }
     return T[n];
}
```

L'exécution de l'instruction :

```
for (int i = 0; i < 6; i++)
     System.out.print(f(i) + " ");
```

dans la fonction principale permet d'afficher:

A) 1 1 2 5 14 42
B) 1 1 2 7 16 42
C) 1 1 2 6 15 42
D) 1 2 2 6 14 44

13. Soit la procédure suivante :

```
void f() {
     int T[] = { 10, 20, -35, 35, -10, 5, -10, 20};
     int a = 0, b = 0, n = 8;
     for (int i = 0; i < n; i++) {
          a = a + T[i];
          b = b + (i * T[i]);
     }
     int c = b;
     for (int j = 1; j < n; j++) {
          b = b + a - n * T[n - j];
          if (b > c) c = b;
     }
     System.out.print(a + " " + b + " " + c);
}
```

Le résultat affiché par la procédure est :

A) 35 165 290
B) 35 168 290
C) 35 160 320
D) 45 180 340

14. Soit la fonction suivante :

```
int f(int T[], int n) {
     int a = -1;
     for (int i = 0; i < n; i++) {
          int b = 0;
          for (int j = 0; j < n; j++) {
              int index = (i + j) % n;
              b += j * T[index];
          }
          a =Math.max(a, b);
     }
     return a;
}
```

Pour T[] = { 1, 20, 2, 10 } et n=4, l'appel à la fonction permet de retourner :

A) 78
B) 102
C) 70
D) 72

15. Soit la procédure suivante :

```
void f(int T[], int n, int x) {
     int i;
     for (i = 0; i < n - 1; i++)
          if (T[i] > T[i + 1]) break;
     int a = (i + 1) % n, b = i;
     while (a != b) {
          if (T[a] + T[b] == x)
              System.out.print("A");
          if (T[a] + T[b] < x) {
              a = (a + 1) % n;
              System.out.print("C");
          } else
              b = (n + b - 1) % n;
     }
     System.out.print("B");
}
```

Pour T[] = { 11,15,6,8,9,10 }, n=6 et x=18, l'appel à la procédure permet d'afficher:

A) A C B A C
B) C A C A B
C) B A C B C
D) C A C B

16. Soit la fonction suivante :

```
int f(int T[], int n) {
     int a , b;
     for (int i = 0; i < n; i++) {
          if (T[i] <= 0 || T[i] > n) continue;
          a = T[i];
          while (T[a - 1] != a) {
              b = T[a - 1]; T[a - 1] = a; a = b;
              if (a <= 0 || a > n) break;
          }
     }
     for (int i = 0; i < n; i++)
          if (T[i] != i + 1) return i + 1;
     return n + 1;
}
```

Pour T[]={ 1, 2, 7, 6, 9, -1, -10, 15 } et n=8, l'appel à la fonction permet de retourner :

A) 2
B) 7
C) 3
D) 5

17. Soit la fonction suivante :

```
int f(int n) {
     int[] T = new int[n + 2];
     T[0] = 0;    T[1] = 1;
     for (int i = 2; i <= n; i++)
          T[i] = T[i - 1] + T[i - 2];
     return T[n];
}
```

Pour n=7, l'appel à la fonction permet de retourner :

A) 21 B) 8

C) 13 D) 17

18. Soit la procédure suivante :

```
void f(int T[], int n) {
     int a =0, b = 0;
     for (int k = 1; k <= n; k++) {
          a = b = -1;
          for (int i = 0; i <= n - k; i++) {
              b = T[i];
              for (int j = 1; j < k; j++)
                  if (T[i + j] < b) b = T[i + j];
              if (b > a)a = b;
          }
          System.out.print( a + " "+b);
     }
}
```

Pour T= { 10, 20, 30, 50, 10, 70, 30 } et n=7, l'appel à la procédure permet d'afficher:

A) 70 70 B) 10 10

C) 70 30 D) 50 10

19. Les variables x et y du code suivant auront respectivement les tailles :

```
struct str1{
     signed char a;
     unsigned short b;
     float c;
}x;
```

```
union str2{
      signed char a;
      unsigned short b;
      float c;
}y;
```

A) 7 Octets et 4 Octets B) 8 Octets et 4 Octets

C) 9 Octects et 4 Octets D) 8 Octets et 8 Octets

20. Dans une itération conditionnelle, l'invariant de boucle est :

A) une instruction à exécuter à chaque passage dans la boucle.

B) une expression booléenne vérifiée pendant toute l'éxcution de la boucle.

C) une expression constante définie avant d'entrer dans la boucle.

D) une instruction à exécuter en fin de boucle.

21. Que vaut la paire (x, y) à la fin de l'itération conditionnelle suivante :

```
int x=24, y=30,temp;
while(y){
      temp=x; x=y; y=temp%y;
}
```

A) (3, 0) B) (24, 6)

C) (30, 6) D) (6, 0)

22. Soit **f** une fonction récursive et soit **g** une fonction itérative équivalente à **f**. Laquelle des affirmations suivantes est vraie ?

A) **f** est toujours meilleure que **g.**

B) **f** utilise plus de mémoire que **g**.

C) **f** utilise moins de mémoire que **g**.

D) **g** est toujours plus meilleure et simple à écrire que **f**.

23. Soit **f** une fonction récursive utilisant une récursivité terminale. Laquelle des affirmations suivantes est vraie concernant la dérécusion de **f** (la version itérative équivalente à **f**)?

A) Elle peut être implémée en utilisant une boucle.

B) Elle peut être implémentée en utilisant un ensemble d'instructions **if**.

C) Elle peut être implémentée par une suite d'instructions dans un bloc.

D) Elle n'est pas possible.

24. Soit la fonction récursive **f** suivante où **g** est une fonction non récursive qui modifie les valeurs des parmaètres **x** et **y** :

```
void  f( int x,int y, int n ){
     if( n > 0 ){ g(&x,&y); f(x,y,n-1); g(&x,&y);}
}
```

La fonction **f** reprsénte un exemple typique de la récursivité :

A) Imbriquée B) terminale

C) non terminale D) croisée

25. Soient les fonctions **f** et **g** suivantes. Quelle est la valeur reoutrnée par l'appel **g(f(2,1))** ?

```
int f( int x, int y ){
     if( x = = 0 )return y;
     return f( x - 1, 2*y);
}
int g( int x ){
     if( x <= 1 ) return 1;
     return 3*g( x-1 ) + g( x-2 );
}
```

A) 43 B) 44

C) 42 D) 41

26. Soit la procédure récursive suivante :

```
void f(int n) {
     if (n > 0) {
          System.out.print(n+ " ");
          f(n - 1);
          f(n - 1);
     }
}
```

pour **n=4**, l'appel à la procédure permet d'afficher:

A) 3 3 3 B) 4 3 2 1 1 2 1 1 3 2 1

C) 4 3 2 1 1 2 1 1 3 2 1 1 2 1 1 D) 1 2 3 4 4 3 2 1

27. Soit la procédure récursive suivante :

```
void f(int n) {
     if (n > 0) {
          System.out.print(n+" ");
          f(n - 1);
          System.out.print(n-1+" ");
     }
}
```

pour **n=4**, l'appel à la procédure permet d'afficher:

A) 4 3 3 1 2 3
B) 1 2 3 4 0 1 2 3 4
C) 4 3 2 1 1 2 3 4
D) 4 3 2 1 0 1 2 3

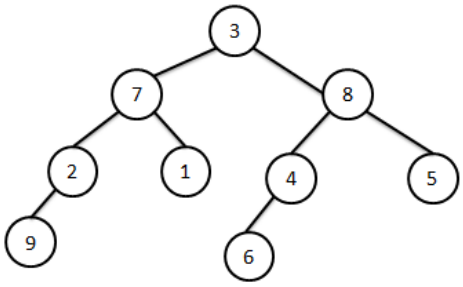
28. On insère les éléments 4, 3, 12, 7, 9 (dans cet ordre) dans un arbre binaire de recherche non équilibré (dont les éléments les plus petits se trouvent à gauche et les plus grands se trouvent à droite). Dans quel ordre vont-ils ressortir pour un affichage en préfixé?

A) 4 3 12 7 9
B) 3 4 7 9 12
C) 9 7 12 3 4
D) 4 3 7 9 12

29. Quelle est la complexité dans le pire des cas de la recherche d'un élément dans un arbre binaire de recherche équilibré de n éléments ?

A) $\Theta(n)$
B) Θ (n log n)
C) Θ (log n)
D) Θ (n^2)

30. Soit l'arbre binaire A suivant :



Et soit la procédure suivante :

```
void afficherContenuArbre(Noeud *a){
     if(a){
          afficherContenuArbre(a->filsGauche) ;
          afficherContenuArbre(a->filsDroit) ;
          printf("%d",a->contenu) ;
     }
}
```

L'appel à cette procédure, avec le paramètre l'arbre de la figure ci-dessus, permet d'afficher :

A) 9 2 7 1 6 4 8 5 3
B) 3 7 8 2 1 4 5 9 6
C) 9 2 1 7 6 4 5 8 3
D) 9 2 7 1 3 6 4 8 5

Partie 2 – Algorithmique et programmation (40 points)

Exercice 1 (20 points) :

On souhaite développer une application qui permet d'effectuer des traitements sur des images en 256 niveaux de gris ; Chaque pixel de l'image est représenté par un entier compris entre 0 et 255. Chaque image de dimension **`W x H`** ou **`W`** représente la largeur de l'image et **`H`** représente la hauteur de l'image, devrait être stockée, ligne par ligne, sous forme d'un vecteur représenté par un tableau à une seule dimension de taille **`W x H`**.

Exemple d'une image de taille **`W=4`** et **`H=3`** dans sa forme Matricielle :

12	33	55	12
3	11	3	16
20	1	0	11

Structure de la même image stockée sous forme d'un vecteur :

12	33	55	12	3	11	3	16	20	1	0	11

Questions :

1. Ecrire les déclarations des variables globales de l'application permettant de représenter l'image.

2. Ecrire une fonction « **`initData`** », qui reçoit en paramètres la largeur et la hauteur de l'image, et qui permet de créer une image en l'initialisant avec des données aléatoires.

3. Ecrire une fonction « **`afficherImage`** » qui permet d'afficher les données de l'image sous forme matricielle.

4. Ecrire une fonction qui permet de calculer et retourner l'histogramme de l'image. L'histogramme d'une image est une courbe représentée par un tableau d'entiers de taille 256. Chaque élément d'indice **`m`** de l'histogramme contient la fréquence de répétition du pixel de couleur **`m`** dans l'image. C'est-à-dire le nombre pixels de l'image ayant la couleur **`m`.**

5. Ecrire une fonction « **`getHistogramme`** », qui permet de remplacer l'image créée par une autre image dont chaque pixel représente la moyenne des 8 pixels qui l'entourent.

6. Evaluer l'ordre de complexité algorithmique de chaque fonction de cette application.

7. Ecrire le code du programme principal qui permet de tester l'ensemble des fonctions de cette application pour le cas d'une image de dimension **`W=20`** et **`H=10`**.

Exercice 2 (20 points) :

Le text mining est une technique permettant d'automatiser le traitement de gros volumes de contenus textuels. Le texte mining peut notamment être utilisé dans le cadre des études de marketing, de la veille, de la social intelligence, des études de satisfaction, etc.

C'est dans ce cadre qu'on se propose dans un premier temps de stocker et analyser les mots d'un corpus de textes constituant l'ensemble des échanges des commerciaux d'une entreprise avec leurs clients. Pour simplifier, on suppose que les documents textes du corpus sont tous exprimés en français. On suppose aussi avoir la liste des URLs des documents du corpus permettant de les localiser et d'y acceder en ligne.

Un premier niveau d'analyser qu'on se propose de faire, consiste à determiner la fréquence de chaque mot par document en omettant d'analyser les mots inutiles (Articles, déterminants, Préposition, Conjonction, etc). On suppose disposer d'une fonction de prétraitement permettant de faire le nettoyage d'un document texte en produisant la liste des mots ramenés dans leur forme de base. Une structure mémoire doit être définie pour le stockage des résultats de l'analyse. Pour cela, la structure d'une table d'association associant à un ensemble de clefs (les mots) à un ensemble correspondant de valeurs (liste de fréquences) est adoptée. Pour une meilleure efficacité, cette structure sera implémentée par une table de hachage pour mettre en cache l'ensemeble de mots et leurs fréquences comme l'illustre la figure 1.

Le principe d'une table de hachage consiste à calculer pour chaque clef **`k`** (mot) un entier de hachage h**`(k)`** compris entre **`0`** et **`n-1`** (où n est la taille de la table). On utilise ensuite un tableau **`T`** de taille **`n`** pour stocker les couples (clef, valeur) ayant le même entier de hachage sous forme d'une liste simplement chainée. La valeur **`T[h(k)]`** est l'adresse du noeud en tête de la liste des couples ayant le même code **`h(k)`**. Les listes chainées sont utilisées comme solution au problème

de collision. Le nœud d'une liste est une structure composée de trois champs : Le mot, sa liste des fréquenes (un tableau de taille **m**, où **m** est le nombre de documents du corpus).

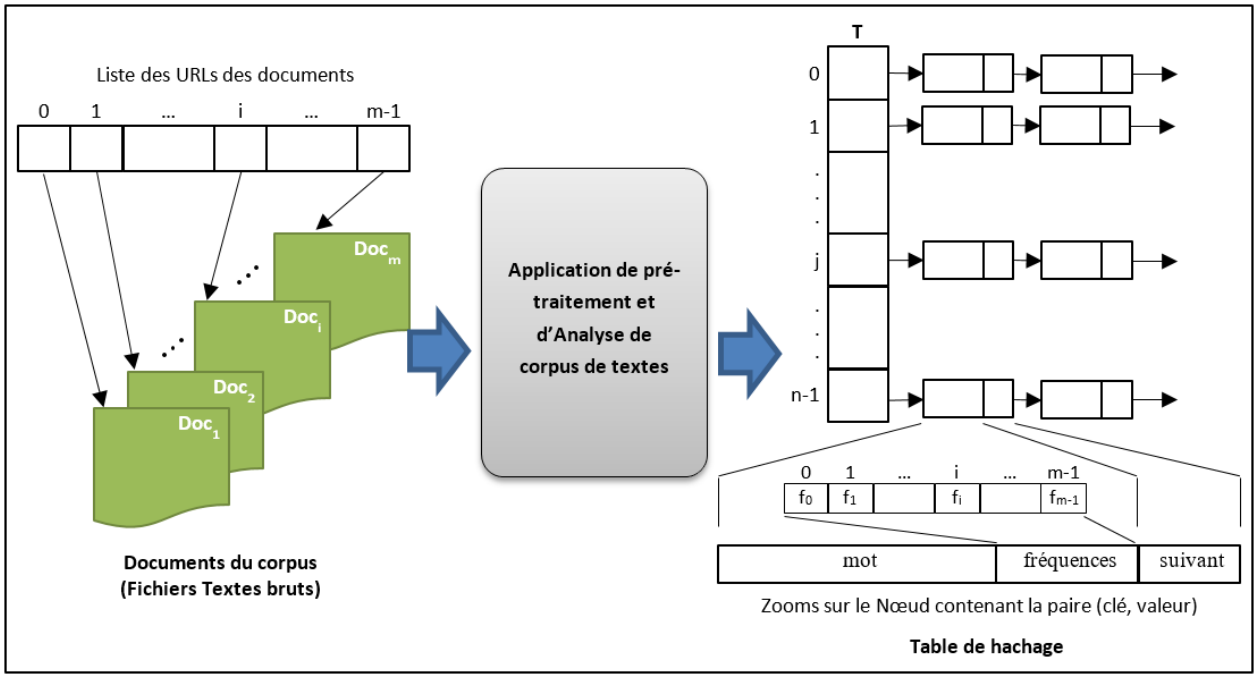


Figure 1 : Schéma de spécification de l'application d'analyse de corpus de textes

La fonction de hachage **h(k)** est une fonction qui transforme la clef (le mot) en un entier **i** compris entre **0** et **n-1** (l'indice dans le tableau **T**). Etant donné que les clefs sont des mots et pour définir une bonne fonction de hachage, on utilise la méthode suivante :

On transforme le mot **k** par une valeur entière v_k calculée par :

$$v_k = \sum_{i=0}^{n_k} c_i a^i$$

Où c_i est le code numérique (code ascii) du ième caractère du mot **k** et **a** un paramètre entier strictement positif choisi arbitrairement sauf qu'il ne doit être une puissance de 2.

La fonction de hachage est alors définie par :

$$h(k) = v_k \% n$$

Où **n** est la taille de la table choisi comme nombre premier suffisament grand.

Questions :

1. Donner les declarations nécessaires pour contruire la structure de la table de hachage. On supposera que le nombre de documents du corpus noté **m** est constant.

2. Soit **di** le ième document du corpus connu par son **URL**. Donner le jeu des fonctions (méthodes) nécessaire pour extraire les mots de **di** et les placer avec leurs fréquences dans la table **T**. On supposera que **T** est une variable static déjà déclarée et on utilisera la fonction de pré-traitement dont le prototype est :

```
char** preTraitement(char* url, int *nbMots) ; //langage C
String[] preTraitement(String url ) ; // langage Java
```

3. Ecrire la fonction qui calcule le code de hachage d'un mot **k**.

4. Ecrire la fonction de construction d'un nouveau nœud (constructeur) pour un mot **k** extrait du document **di**. Les fréquences d'un mot **k** pour les autres documents sont initialisées à la valeur **0**.

5. Ecrire la fonction qui ajoute un mot **k** à la table **T**. La fonction doit mettre à jour la fréquence si le mot **k** est déjà placé dans **T**. Les nouveaux nœuds sont toujours ajoutés en tête.

6. Ecrire la fonction qui permet de traiter l'extration et le placement dans la table **T** de tous les documents du corpus.

7. Ecrire la fonction qui permet de rechercher la liste des mots les plus fréquents dans chaque document.

ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSET

المدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء

Concours d'accès en 1ère année des Cycles d'Ingénieurs GLSID et II-BDCC

Numéro de l'examen :

Nom :

Prénom :

Signature :

Grille de réponses pour la partie QCM

- **Cocher, avec stylo à encre, la case correspondante à la bonne réponse.**
- **Réponse juste : 2 pts**
- **Réponse fausse : 0 pt**

Note :

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

	A	B	C	D	E
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					