IF2B – Examen final
Durée : 1h30

Seule une feuille A4 manuscrite nominative recto-verso est autorisée comme document.

Tous les autres documents ou tout autre dispositif électronique sont interdits.

Barème donné à titre indicatif (±1)
Remarque : vous pouvez rédiger les algorithmes demandés en pseudo-code ou en langage C (sauf si le langage C est imposé par l'exercice). Le prototype complet des sous-programmes (ou fonctions) doit être fourni. Il n'est pas nécessaire de préciser les « includes » pour les exercices en C.

Exercices 1 et 2 à rendre sur des copies séparées.

Exercice 1 – Tri par comptage (8 points)

Le **tri par comptage** est un algorithme de tri très rapide, mais il ne fonctionne idéalement que pour des entiers dont les valeurs sont limitées et peut nécessiter beaucoup de mémoire. L'objectif de l'exercice est donc de programmer un tri par comptage capable de trier un tableau T d'entiers de taille N, contenant des valeurs comprises entre 0 et 100 inclus.

Le principe de l'algorithme consiste ensuite à :

- Compter le nombre de fois que chaque valeur (entre 0 et 100) apparaît dans le tableau T
- Générer le tableau trié à partir du comptage

Par exemple, soit le tableau T :

T[index]	1	5	5	3	2	1
index	0	1	2	3	4	5

On génère le **tableau de comptage** C suivant :

C[index]	0	2	1	1	0	2	0		0	0
index	0	1	2	3	4	5	6		99	100

La case C[0] contient le nombre de 0 dans le tableau T, la case C[1] le nombre de 1, …, la case 100 le nombre de 100.

Une fois le tableau de comptage généré, il suffit de le parcourir pour générer un nouveau tableau : pour chaque case C[i] du tableau, on ajoute dans un nouveau tableau U[i] fois la valeur i (par rapport à l'exemple, on ajoute 0 fois la valeur 0, 2 fois la valeur 1, 1 fois la valeur 2, etc.).

U[index]	1	1	2	3	5	5
index	0	1	2	3	4	5

1) **Ecrire le sous-programme qui produit le tableau de comptage pour un tableau T donné et une taille N donné (on supposera que les valeurs du tableau T sont comprises entre 0 et 100)**

4 points.

Exemple de correction :

```
void comptage(int tailleT, int T[tailleT], int C[101]) {
    for(int i = 0; i < tailleT; i++) {
```

```
            C[T[i]]++;
        }
}
```

2) **Ecrire le sous-programme qui à partir du tableau de comptage génère un tableau trié selon l'algorithme précédemment décrit**

4 points.

Exemple de correction :

```
void tri_par_comptage(int C[101], int tailleU, int U[tailleU])
{
    int position = 0;
    for(int i = 0; i < 101; i++) {
        for(int j = 0; j < C[i]; j++) {
            U[position] = i;
            position = position + 1;
        }
    }
}
```

Exercice 2 – Algorithme de compression (12 points)

Un algorithme de compression permet de diminuer le nombre d'octets utilisés pour représenter des données. Dans cet exercice nous allons implémenter un algorithme de compression simple capable de réduire la taille d'un fichier texte.

Principe de l'algorithme : *Soit une chaîne de caractère C quelconque, par exemple C = « AAAAABBCCCCCDDDDDEEEEEEAAA ». La technique de compression que nous allons utiliser par consister à compter le nombre de fois qu'un même caractère est mentionné de suite, et à les remplacer par un couple de valeurs composé du nombre d'occurrence et du caractère concerné. Par exemple, dans la chaîne C nous avons 5 fois le A, donc les 5 occurrences de A doivent être remplacées par « 5A ». Si l'algorithme est entièrement appliqué sur C, nous obtiendrons la chaîne compressée « 5A2B5C5D6E3A ».*

***Attention** : si un caractère est présent 10 fois ou plus d'affilée, nous décomposerons le remplacement en plusieurs décomposition de telle sorte que la valeur représentant la quantité de caractères ne dépassera jamais 9, ainsi on n'écrira jamais « 13A » mais plutôt « 9A4A », ou jamais « 28X » mais « 9X9X9X1X ».*

Afin de mener à bien l'implémentation de cet algorithme de compression, nous divisons le travail en plusieurs étapes à réaliser :

1) **Définir un type structuré « Element » capable de représenter un couple de valeurs correspondant à un caractère et son nombre d'apparitions successives**

```
typedef struct {
    char caractere;
    int nb_occurrence;
} Element;
```

2) **Définir le sous-programme qui, étant donné une chaîne de caractère quelconque, et une position dans cette chaîne de caractère, retourne le caractère présent à la position donnée et le nombre de fois qu'il apparaît suite à cette position (si le caractère apparaît plus de 9 fois, on s'arrêtera à 9)**

Exemple de correction :

```
Element comptage(int position, char chaine[]) {
    Element resultat;
    int i = position;
    resultat.caractere = chaine[i];
    while (chaine[i] != '\0' && chaine[i] ==
resultat.caractere && position - i < 9) {
        i = i + 1;
    }
    resultat.nb_occurrence = i - position;
    return resultat;
}
```

Il faut faire particulièrement attention à contrôler le caractère de fin de chaîne pour éviter un dépassement du tableau. Attention : en C il n'est pas possible de retourner deux valeurs, mais il est possible de retourner une variable composée de deux valeurs, ici Element (l'utilisation du passage par adresse est aussi possible).

3) **Définir le sous-programme qui, étant donné une chaîne de caractère quelconque va enregistrer le résultat de la compression dans un tableau. Le prototype de cette fonction sera void compression(char chaine_a_compresser[], char resultat[]); *(vous pouvez changer les noms des paramètres par souci de place sur votre copie)***

Exemple de correction :

```
void compression(char chaine_a_compresser[], char resultat[])
{
    int position_in = 0;
    int position_out = 0;
    Element e = comptage(position_in, chaine_a_compresser);
    while (e.caractere != '\0') {
        // Remplissage du resultat
        resultat[position_out] = (e.nb_occurrence + '0');
        resultat[position_out + 1] = e.caractere;
        position_out = position_out + 2;
        // Passage à la partie suivante de la chaine
        position_in = position_in + e.nb_occurrence;
        e = comptage(position_in, chaine_a_compresser);
    }
}
```

4) **Ecrire en C le programme *main()* qui ouvre deux fichiers « input.txt » et « output.txt », qui lit, ligne par ligne, le contenu de « input.txt », le compresse, et écrit la chaîne compressée dans le fichier « output.txt » (on supposera que le résultat de la compression ne dépassera jamais 100 caractères).**

Le plus important est le respect de la procédure de manipulation de fichiers. Il faut utiliser les instructions correctes pour ouvrir les fichiers avec les modes appropriés, puis contrôler le succès de l'ouverture, faire les traitements adéquats et fermer les fichiers.

Exemple de correction :

```
int main() {
    FILE *input = fopen("input.txt", "r");
    FILE *output = fopen("output.txt", "w");

    if(input != NULL && output != NULL) {
        // Lecture d'une ligne en boucle
        char ligne[1000];
        while (fgets(ligne, 1000, input)) {
            ligne[strlen(ligne) - 1] = '\0';
            // Compression
            char resultat[100];
            compression(ligne, resultat);
            // Sauvegarde
            fprintf(output, "%s\n", resultat);
        }
    }

    fclose(input);
```

```
    fclose(output);

    return 0;
}
```

Remarque : les questions 1, 2 et 3 sont relativement dépendantes mais rien ne vous oblige à vous appuyer sur la réponse à une question pour répondre à la question suivante (c'est possible de faire autrement).

Mémo

Conversion d'une chaîne de caractères en entier : int atoi(char str[]);
Conversion d'une chaîne de caractères en flottants : float atof(char str[]);
Fonctions usuelles :

FILE * **fopen**(char filename[], char accessMode[]);
int **fclose**(FILE * stream);
int **fprintf**(FILE * stream, char format[], ...);
int **fscanf**(FILE * stream, char format[], ...);
char * **strcpy**(char destination[], char source[]);
unsigned int **strlen**(char theString[]);
char * **fgets**(char string[], int maxLength, FILE * stream);