Série IV

BONNAZ Aymeric, MOUILLET Claude

10 mars 2021

Cette série d'exercices appartient au corpus testant vos connaissances et votre compréhension des concepts étudiés lors de l'initiation **C# INTM Strasbourg**.
Les réponses écrites doivent être données sous leurs questions.

Nom apprenant : _____

Date de réalisation : _____

Exercice I — Code Morse

Le **Morse** est un code permettant de communiquer un texte à l'aide d'enchaînements d'impulsions courtes et longues. Chaque lettre, chiffre ou symbole de ponctuation est associé à une combinaison déterminée de ces signaux intermittents. Ce dernier s'applique à de nombreuses applications notamment dans l'armée et la marine.
Le présent exercice consiste à établir des méthodes de traduction et d'écriture du code international Morse.

FIGURE 1 – Code Morse en morse

Quelques règles de fonctionnement du Morse :

- Deux types d'impulsion sont utilisées :
  - Impulsion longue, nommée **"taah"** : ===
  - Impulsion courte, nommée **"ti"** : =
- Le point **.** décrit un espace entre deux symboles.
  - Séparation entre deux lettres : **...**
  - Séparation entre deux mots : **.....**

Pour simplifier la situation, seules les vingt-six lettres de l'alphabet sont considérées. Pour chacune des méthodes, une combinaison inconnue est associée au symbole + et une chaîne en morse contenant autre chose que des = et **.** doit lever une exception de type **ArgumentException**.

1. (1 point) Un dictionnaire de couple clé-valeur (code, lettre) est fourni afin d'implémenter les différentes méthodes. Justifier ce choix de structure de donnée ?

   _____

   _____

   _____

2. Caractérisation d'un code morse :
   (a) (1 point) Détermination du nombre de lettres présentes dans la chaîne de caractères d'entrée. Implémenter la méthode **LettersCount**(string code) : int .
   (b) (1 point) Détermination du nombre de mots présents dans la chaîne de caractères d'entrée. Implémenter la méthode **WordsCount**(string code) : int .

Après avoir caractérisé la chaîne de caractères d'entrée, il est temps de transcrire le contenu de celle-ci en langage naturel. Les lettres sont en majuscule et deux mots sont séparés par un espace. Supposez pour l'instant que les seules configurations de points sont le point **isolé**, le **triplet** et le **quintuplet**.

3. (3 points) Implémenter la méthode **MorseTranslation**(string code) : string .

```
Traduction Morse :
===.=.===.=...===.===.===...===.=.=...=.....===.===...===.===.===...=.===.=...=.=.=...= : CODE MORSE
=...=.===.=...==...===.===.===...=.===.= : ER+OR
```

FIGURE 2 – Exemples de codes en morse

Dans des conditions d'utilisation réelles, de nombreuses imperfections peuvent se produire dans la transmission du message. Un programme traitant des cas réels se doit de prendre en compte ces impuretés. Supposons ici que celles-ci se manifestent par des ajouts de points inopinés ainsi que des points au début et à la fin du message.

Mettons à jour les règles du Morse :

— Séparation entre des impulsions : **.** ou **..**

— Séparation entre deux lettres : **...** ou **....**

— Séparation entre deux mots : **.....** ou plus.

4. (2 points) Implémenter la méthode **EfficientMorseTranslation**(string code) : string .

```
Traduction Morse :
...===.=.===.=...===.===.===...===.=.=...=..... : CODE
===.=.===.=....===..===..===...===.=.=...=..... : CODE
```

FIGURE 3 – Exemples de codes imparfaits en morse

En bonus, finissons par l'encodage d'une phrase écrite avec des lettres de l'alphabet latin en langage morse.

5. (2 points (bonus)) Implémenter la méthode **MorseEncryption**(string sentence) : string .

Exercice II — Contrôle des parenthèses

Un éditeur de texte pour développeur se doit d'implémenter une panoplie de fonctionnalités dans le but de faciliter l'écriture de programmes informatiques. L'une de ses fonctionnalités est la vérification de la fermeture des parenthèses. L'objectif de cet exercice est d'implémenter une méthode nous retournant si oui ou non la chaîne de caractères d'entrée satisfait cette propriété.

Quelques règles de fonctionnement :

- Les types de parenthèses **ouvrantes** et **fermantes** sont respectivement ({, (, [) et (},),]).
- Une parenthèse **ouvrante** doit être associée à une parenthèse **fermante** de même type.
- Les couples de parenthèses peuvent être imbriquées les uns dans les autres.
- En plus des parenthèses, la chaîne de caractères contient des lettres de l'alphabet latin et des chiffres.

  Astuce : Se concentrer sur les parenthèses, ignorer les autres caractères. Une parenthèse fermante est comparée avec la **dernière** parenthèse ouvrante stockée. Une fois fermée, une parenthèse ouvrante n'est plus testée.

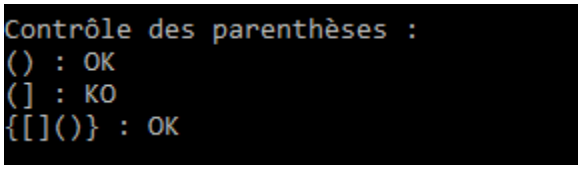

FIGURE 4 – Exemples de contrôles de parenthèses

1. (1 point) Quelle structure de données est la plus adaptée à la résolution de ce problème ? *Justifier votre choix.*

2. (3 points) Implémenter la méthode **BracketsControls**(string sentence) : bool.

Exercice III — Liste des contacts téléphoniques

Dans cet exercice, l'objectif est d'implémenter un annuaire téléphonique avec ajout, suppression et utilisation de contacts téléphoniques.

FIGURE 5 – Un annuaire téléphonique

Cet exercice consiste à implémenter la structure de données **PhoneBook**.

PhoneBook :

- **IsValidPhoneNumber**(string phoneNumber) : bool
- **ContainsPhoneContact** (string phoneNumber) : bool
- **PhoneContact**(string phoneNumber) : void
- **AddPhoneNumber**(string phoneNumber, string name) : bool
- **DeletePhoneNumber**(string phoneNumber) : bool
- **DisplayPhoneBook**() : void

Choix de la structure de donnée :

Quelques indications sur les contraintes de l'annuaire :

- Un numéro téléphonique est toujours associé à un nom et est **unique**.
- Un contact téléphonique n'est **pas** nécessairement **unique**.
- L'annuaire fonctionne seulement dans le sens numéro vers nom du contact.

1. (1 point) En analysant les contraintes, quelle structure de données doit être utilisée ? *Justifier votre choix.*

Méthodes de l'annuaire :

Quelques indications sur les contraintes pesant sur les numéros et contacts :

- Un numéro téléphonique est valide s'il contient exactement 10 chiffres, le premier et le second étant respectivement égal et différent de 0.
- Ce nom est aussi une chaîne de caractères quelconque non vide.

2. (1 point) Implémenter la méthode suivante :
   Est-ce que le numéro est valide ? : **IsValidPhoneNumber**(string phoneNumber) : bool

Quelques indications sur le comportement de l'annuaire :

- Les méthodes d'ajout et de suppression d'un numéro téléphonique renvoient le statut de l'opération.
- L'accès direct à un numéro téléphonique entraine une exception si ce dernier n'est pas présent.

3. Implémenter simplement les méthodes suivantes :
   (a) (½ point) Est-ce que l'annuaire contient ce numéro ? :
   ContainsPhoneContact (string phoneNumber) : bool
   (b) (½ point) Contact associé au numéro :
   PhoneContact(string phoneNumber) : void
   (c) (½ point) Ajout d'un numéro :
   AddPhoneNumber(string phoneNumber, string name) : bool
   (d) (½ point) Suppression d'un numéro :
   DeletePhoneNumber(string phoneNumber) : bool

Affichage de l'annuaire :

L'affichage de l'emploi du temps s'effectue de la manière suivante (figure (6)) :

- L'en-tête consiste uniquement en la phrase "Annuaire téléphonique :".
- Chaque ligne du contenu présente le numéro puis le contact téléphonique séparé par un espace.
- Pour un annuaire vide, on écrit "Pas de numéros téléphoniques".

```
Annuaire téléphonique :
-----------------------
0601020304 : Giselle
0486019304 : Coiffeur
-----------------------
```

FIGURE 6 – Annuaire téléphonique.

4. (2 points) Implémenter la méthode : **DisplayPhoneBook**() : void

Exercice IV — Emploi du temps professionnel

Dans la vie professionnelle, la mise en place des réunions de travail est une tâche fastidieuse impliquant de déterminer la disponibilité de chacun des participants potentiels. Cet exercice a pour volonté de mettre en place une structure de données adaptée à la gestion des réunions d'un salarié.

FIGURE 7 – Le monde merveilleux des réunions

Cet exercice consiste à implémenter la structure de données **BusinessSchedule**.

BusinessSchedule :

- **IsEmpty**() : bool
- **SetRangeOfDates**(DateTime begin, DateTime end) : void
- **AddBusinessMeeting**(DateTime date, TimeSpan duration) : bool
- **DeleteBusinessMeeting**(DateTime date, TimeSpan duration) : bool
- **ClearMeetingPeriod**(DateTime begin, DateTime end) : int
- **DisplayMeetings**() : void

Choix de la structure de donnée :

Lors de la résolution d'un problème, le choix de la **judicieuse** structure de données est salutaire et facilite celle-ci. Les prochaines questions représentent le cheminement intellectuel nécessaire au choix fait pour le reste de l'exercice.

1. (2 points) Supposons une liste C# de N éléments, quel est le nombre d'inversions d'éléments nécessaires afin d'insérer un élément dans la liste au début, au milieu ou à la fin de celle-ci ?

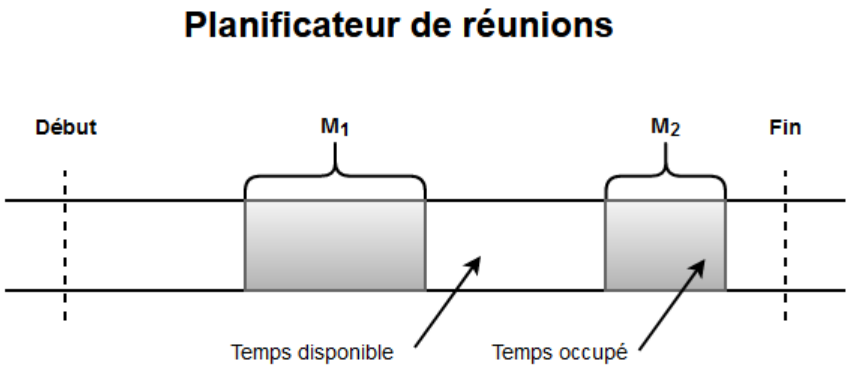

FIGURE 8 – Schéma représentant le planificateur de réunions

Justifier votre choix.

2. (2 points) Supposons toujours une liste C# de N éléments, quel est le nombre de comparaisons **dans le pire cas** nécessaire à la recherche d'un élément donné dans celle-ci si elle est non-triée ou triée ?
Justifier votre choix.

Dans le reste de l'exercice, le choix s'est porté sur **SortedDictionary<TKey,TValue>**. Le tableau suivant vous présente ses principales méthodes d'utilisation et leur coût d'utilisation.

Méthodes ou Propriétés	**Explications**	**Coût**
ContainsKey (TKey key) : bool	Test sur l'existence de la clef.	$O(log(N))$
Add (TKey key, TValue value) : void	Ajout de la valeur associée à une clef.	$O(log(N))$
Remove (TKey key) : void	Suppression de la valeur associée à la clef spécifiée.	$O(log(N))$
Count : int	Nombre d'éléments.	$O(1)$
Item[TKey key] : TValue	Accès à la valeur associée à la clef spécifiée.	$O(log(N))$
Keys : Type énumérable	Ensemble des clefs **triées**.	$O(1)$

État de l'emploi du temps :

L'emploi du temps ne fonctionne que sur une période de temps donnée. Celle-ci est par défaut [01/01/2020 - 31/12/2030[. Elle ne peut être modifiée que lorsque l'emploi du temps est vide de toute réunion et doit être valide (sinon renvoi d'une exception).

3. Implémenter les méthodes suivantes :
   (a) (1 point) Est-ce que l'emploi du temps est vide ? : **IsEmpty**() : bool
   (b) (1 point) Période étudiée : **SetRangeOfDates**(DateTime begin, DateTime end) : void

L'affichage de l'emploi du temps s'effectue de la manière suivante :

- La date et l'heure sont présentés selon le format **dd/MM/YYYY HH :mm :ss**.
- L'en-tête contient la période étudiée et le contenu la période de chaque réunion.
- Pour un emploi du temps vide, on écrit "Pas de réunions programmées".

On trouve deux exemples avec les figures (9) et (10).

```
Emploi du temps : 10/03/2021 00:00:00 - 12/03/2021 00:00:00
------------------------------------------------------------
Réunion 1       : 10/03/2021 14:00:00 - 10/03/2021 15:30:00
Réunion 2       : 10/03/2021 17:00:00 - 10/03/2021 18:00:00
------------------------------------------------------------
```

FIGURE 9 – Emploi du temps contenant des réunions.

```
Emploi du temps : 10/03/2021 00:00:00 - 12/03/2021 00:00:00
------------------------------------------------------------
Pas de réunions programmées
------------------------------------------------------------
```

FIGURE 10 – Emploi du temps vide.

4. (1 point) Suivant les indications précédentes, implémenter : **DisplayMeetings**() : void

Gestion des réunions :

Une réunion est acceptée dans l'emploi du temps dans les cas suivants :

- La période de la réunion est incluse entièrement dans la période étudiée par l'emploi du temps.
- Aucune intersection avec une réunion déjà présente dans l'emploi du temps.

Recherche des conflits potentiels :

Afin de faciliter l'implémentation de l'ajout d'une réunion, une méthode intermédiaire retournant le couple de réunions les plus proches de la potentielle réunion à ajouter doit être implémentée. Quelques précisions :

- La première valeur correspond à la date de début de la réunion la plus proche mais inférieure ou égale.
- La seconde valeur correspond à la date de début de la réunion la plus proche mais supérieure strictement.
- Si l'une des deux valeurs n'existe pas, renvoyez **null** à la place de celle-ci.

5. (2 points) Implémenter la méthode intermédiaire :
   ClosestElements(DateTime beginMeeting) : KeyValuePair<DateTime,DateTime>

6. Ajout et suppression d'une réunion :
   (a) (1 point) Ajout d'une réunion, renvoi du statut de l'opération :
   AddBusinessMeeting(DateTime date, TimeSpan duration) : bool
   (b) (1 point) Suppression d'une réunion, renvoi du statut de l'opération :
   DeleteBusinessMeeting(DateTime date, TimeSpan duration) : bool

Le nettoyage d'une période donnée de l'emploi du temps implique de supprimer toutes les réunions incluses au moins en partie dans celle-ci. Cette période doit être valide et incluse dans la période étudiée par l'emploi du temps (sinon renvoi d'une exception).

7. (1 point) En réutilisant astucieusement la méthode **ClosestElements**, implémenter :
   ClearMeetingPeriod(DateTime begin, DateTime end) : int