TP C#2
My name is Gutenberg

Rendu

Archive

Vous devez rendre un fichier zip avec l'architecture suivante:

```
rendu-tp-firstname.lastname.zip
    |-- rendu-tpcs2-firstname.lastname/
        |-- AUTHORS
        |-- README
        |-- HelloWorlds/
            |-- HelloWorlds.sln
            |-- HelloWorlds/
                |-- everything except bin/ and obj/
        |-- Add/
            |-- Add.sln
            |-- Add/
                |-- everything except bin/ and obj/
        |-- Skyscraper/
            |-- Skyscraper.sln
            |-- Skyscraper/
                |-- everything except bin/ and obj/
        |-- JollyRoger/
            |-- JollyRoger.sln
            |-- JollyRoger/
                |-- everything except bin/ and obj/
        |-- Morse/
            |-- Morse.sln
            |-- Morse/
                |-- everything except bin/ and obj/
```

- Vous devez bien sûr remplacer *firstname.lastname* avec votre propre login (qui est de la forme *prenom.nom*).
- Votre code doit compiler.
- Pour ce TP, vous êtes autorisés à implémenter d'autres fonctions que celles demandées, elles seront considérées comme des bonus. Cependant, vous devez garder une archive la plus légère

possible.

AUTHORS

Ce fichier doit contenir une ligne formatée comme il suit : une étoile (*), un espace, votre login et un retour à la ligne, RIEN DE PLUS.
Voici un exemple (où $ représente un retour à la ligne et ␣ un espace):

```
*␣firstname.lastname$
```

Notez que le nom du fichier est AUTHORS avec AUCUNE extension.

Pour créer simplement un fichier AUTHORS valide, vous pouvez taper la commande suivante dans un terminal:

```
echo "* firstname.lastname" > AUTHORS
```

README

Vous devez écrire dans ce fichier tout commentaire sur le TP, votre travail, ou plus généralement vos forces / faiblesses. Un README vide sera considéré comme une archive invalide (malus).

1 Introduction

1.1 Objectifs

Durant ce TP, nous allons aborder les notions suivantes:

- La Console,
- ASCII art,
- Manipulation basique de string,
- Morse.

2 Cours

2.1 La Console

Le terminal

Jusqu'à maintenant, la plupart des programmes que vous utilisiez fonctionnaient avec la souris. Ces applications (appelées applications GUI), composées de fenêtres, onglets, boutons, images, etc., ne sont, comme vous pouvez penser, pas les plus basiques qui existent. Vous ne devrez pas implémenter de programme de ce type durant vos TPs de C# cette année.

À la place, nous vous demanderons de créer des programmes CLI (pour Command Line Interface). Ces programmes fonctionnent dans ce que l'on appelle un terminal, et sont exécutables via un autre programme, appelé shell. Un shell est simplement un programme qui permet à un utilisateur d'interagir avec sa machine en utilisant des commandes textuelles. Une fois un terminal ouvert un shell est automatiquement appelé et attend les entrées. Le shell (bash par défaut sur vos machines) vous affiche quelques informations à propos de vous, votre machine et le dossier où vous êtes. Ces informations sont contenues dans une chaîne appelée *prompt* (voir ci-dessous).

```
[sparrow@sea ~/blackpearl]$
```

Dans cet exemple, l'utilisateur s'appelle `sparrow`, il/elle a appelé son ordinateur `sea` et est actuellement dans le dossier `~/blackpearl`.

Utiliser la ligne de commande

Après avoir affiché le prompt, le shell attend que vous entriez une commande. Une commande valide peut avoir plusieurs aspects, c'est généralement le nom d'un programme, éventuellement suivi d'arguments. Lisez l'exemple ci-dessous:

```
1 [sparrow@sea ~]$ ls
2 foo  acdc  sups
3 [sparrow@sea ~]$ ls foo
4 bar  baz
5 [sparrow@sea ~]$
```

Dans cet exemple, le programme `ls` est d'abord exécuté tout seul (ligne 1). `ls`, affiche le contenu d'un répertoire, si aucun argument n'est donné, il affiche le contenu du répertoire courant (ici `foo`, `acdc` et `sups`). À la troisième ligne, un argument est donné à `ls`, `foo`, son contenu est donc affiché. Comme vous le voyez, pour donner un argument à un programme, vous devez seulement écrire l'argument

après le nom du programme, précédé par un espace. À la ligne 5, le prompt est de nouveau affiché et le shell attend pour une nouvelle commande.

Bien sûr, plus d'un argument peuvent être passés à un programme. Des arguments spéciaux peuvent souvent être donnés aux programmes pour modifier leur comportement. Ces arguments sont appelés options et commencent habituellement avec un `'-'`. Généralement, sur les systèmes Unix, vous pouvez obtenir la liste des arguments possibles en utilisant l'argument `--help`, ou en lisant la page de `man`.

Flux standards

Maintenant que vous savez comment utiliser la ligne de commandes, abordons des notions un peu plus complexe. Les terminaux utilisent des *flux*. Un flux est simplement une voie qu'utilise votre système pour lire ou écrire des données, notamment du texte. Par exemple, sur les systèmes UNIX, chaque fichier peut être utilisé comme un flux. Il y a 3 principaux flux que vous devez connaître:

- Entrée standard: Ce flux est celui dans lequel vos programmes lisent ce que vous tapez. Tout ce que vous tapez dans un terminal est envoyé sur ce flux. On l'appelle souvent `stdin`.
- Sortie standard: Ce flux est l'opposé de `stdin`, c'est le flux dans lequel votre shell et vos programmes écrivent. Ce que vous, en tant qu'utilisateur, lisez dans votre terminal est ce que la sortie standard envoie. On l'appelle souvent `stdout`.
- Sortie d'erreur standard: Ce flux est est très similaire à `stdout`, il fonctionne exactement de la même manière, sauf que par convention seuls des messages relatifs à des erreurs sont envoyés sur ce flux. Pour vous quand vous lisez dans la console cela ne change rien du tout. On l'appelle souvent `stderr`.

Il est important que vous compreniez correctement comment vous pouvez interagir avec vos programmes via ces flux. Si vous avez des doutes, demandez à vous ACDCs adorés.

2.2 Récupérer des arguments

Comme vous le savez déjà, la méthode `Main` est automatiquement appelée lorsque votre programme est lancé. Regardons son prototype:

```
1 static int Main(string[] args);
```

Le préfixe `static` est un mot-clé C#, vous apprendrez plus tard son intérêt. `int` peut être remplacé par `void`, il est utilisé pour envoyer à votre système un code à propos de l'état de votre programme à la fin de son exécution, vous pouvez l'ignorer pour le moment. `Main` est l'identifiant de la méthode. Et... `string[] args`. Comme vous savez, dans les parenthèses d'une méthode, nous listons ses paramètres. Mais étant donné que la méthode `Main` est automatiquement appelée quand votre programme commence, qui peut changer ces arguments et quel est leur intérêt ? La réponse est simple, quand votre

système appelle la méthode `Main`, il met automatiquement tous les arguments donnés à votre programme en ligne de commande dans le tableau `args`. Par exemple, écrivez ce petit programme et compilez le en un programme appelé `MyEcho.exe`:

```
1  static void Main(string[] args)
2  {
3      if (args.Length < 1)
4          Console.WriteLine("Not enough arguments.");
5      else
6          Console.WriteLine(args[0]);
7  }
```

Lançons-le dans la console:

```
1  [sparrow@sea ~]$ ./MyEcho.exe
2  Not enough arguments.
3  [sparrow@sea ~]$ ./MyEcho.exe Test.
4  Test.
5  [sparrow@sea ~]$ ./MyEcho.exe I love my ACDCs
6  I
7  [sparrow@sea ~]$ ./MyEcho.exe "I love my ACDCs"
8  I love my ACDCs
9  [sparrow@sea ~]$
```

Comme vous voyez, ce programme vérifie si des arguments lui sont passés. Si aucun argument n'est donné, il affiche le message `"Not enough arguments."` sur `stdout`. Si des arguments sont donnés, il affiche le premier (index 0 du tableau).

Remarquez le comportement du programme aux lignes 7 et 8 de l'exemple de la console. Ici, nous utilisons des doubles guillemets (") lorsque nous donnons des arguments à notre programme. Si vous faites ça, les mots entre guillemets sont vus par le shell comme un seul token et mis dans le tableau `args` comme un seule chaîne de caractères. Si vous n'écrivez pas de doubles guillemets, le shell découpe les arguments au niveau des espaces, et chaque mot est mis seul à l'index correspondant à sa position dans la console dans le tableau.

2.3 La classe Console

Vous allez apprendre à lire et écrire sur les flux standards. Pour cela, la bibliothèque C# contient la classe Console[1]. Cette classe contient un grand nombre de méthodes, qui vous permettent d'écrire dans la console, de lire depuis celle-ci, récupérer des informations sur son état, changer ses propriétés.

[1] `https://msdn.microsoft.com/en-us/library/system.console`

Voici un résumé de ce que vous pouvez faire avec celle classe et quelles méthodes utiliser. Vous pouvez bien sûr avoir de plus amples informations sur cette classe en lisant la page MSDN à son sujet. Toutes ces méthodes peuvent être appelées en écrivant dans votre code `Console.MethodName(args)`.

Écriture

Deux méthodes vous permettent d'écrire dans la console:

- `Console.Write(arg)`: Cette méthode convertit le paramètre `arg` en `string` et l'affiche sur `stdout`. L'argument peut être de nombreux types, la méthode peut même prendre plus d'un paramètre, lisez la page MSDN pour avoir plus d'informations sur cette méthode.
- `Console.WriteLine(arg)`: Cette méthode fait la même chose que `Console.Write`, et concatène un retour à la ligne à la fin de la string affichée.
  => `Console.WriteLine(arg) == Console.Write(arg + "\n")`

Ces deux méthodes ont des variantes, `Console.Error.Write` et `Console.Error.WriteLine`, elles font la même chose que celles ci-dessus, sauf qu'elles affichent le message résultant sur la sortie d'erreur (`stderr`) au lieu de `stdout`.

Voici quelques exemples d'utilisation de ces fonctions, n'hésitez pas à les tester (la sortie est écrite est en commentaire):

```
1  Console.WriteLine("The Code is the Law.");
2  // The Code is the Law.
3
4  Console.Write("Jack");
5  Console.Write("Sparrow\n");
6  // JackSparrow
7
8  Console.WriteLine("a" + "b" + "c" + "d");
9  // abcd
10 Console.WriteLine("42");
11 // 42
12 Console.WriteLine(42);
13 // 42
14 Console.WriteLine(6 * 7);
15 // 42
16
17 string beloved = "ACDC";
18 bool best = !false;
19 Console.WriteLine("My {0}s are the {1} best ones."
20                   , beloved, best);
21 // My ACDCs are the True best ones.
```

```
22 Console.Error.Write("An error occurred.\n");
23 // on stderr: An error occurred.
```

Rappellez-vous que même si pour vous tout est affiché de la même façon sur la console, quand vous utilisez `Console.Error`, le résultat est envoyé sur un autre flux, `stderr`.

Lecture

Lire depuis la console est l'opération inverse, elle permet à un programme de récupérer une valeur tapée par un utilisateur. Deux méthodes permettent de lire depuis l'entrée standard:

- `Console.Read()`: Cette méthode attend que l'utilisateur tape un caractère. Dès que l'utilisateur en tape un dans la console, la méthode retourne le caractère tapé (casté en `int`) et le programme reprend son exécution.
- `Console.ReadLine()`: Cette méthode attend que l'utilisateur tape une ligne entière, terminée par un retour à la ligne (quand la touche `Enter` est tapée). Dès que l'utilisateur en a tapé une dans la console, la méthode retourne la ligne entière dans une string (sans le `'\n'` de fin) et le programme reprend.

Pour essayer ces méthodes, lisez et lancez le programme suivant:

```
1  static void Main(string[] args)
2  {
3      Console.Write("First char: ");
4      int c = Console.Read();
5      Console.WriteLine();
6      Console.WriteLine("You typed: '{0}'", (char)c);
7
8      Console.Write("Second char: ");
9      c = Console.Read();
10     Console.WriteLine();
11     Console.WriteLine("You typed: '{0}'", (char)c);
12
13     Console.WriteLine("Type a line:");
14     string str = Console.ReadLine();
15     Console.WriteLine("You typed: \"{0}\"", str);
16     Console.WriteLine("Length == " + str.Length);
17 }
```

Notez que `Console.Error.Read()` n'existe pas. En effet, cela n'aurait aucun sens car `stderr` n'est qu'un flux de sortie.

Apparence

Vous pouvez donc lire et écrire dans la console. Ça c'est cool. Mais c'est un peu ennuyeux. En réalité, la classe Console vous permet de faire beaucoup d'opérations pour modifier la sortie de vos programmes, comme le repositionnement du curseur, le changement des couleurs d'arrière-plan ou de premier plan, etc.:

Méthodes:

- `Console.Clear()`: Nettoie la console, cela veut dire qu'après un appel à cette méthode, tout le texte sur la console est effacé et le curseur utilisé pour écrire est placé dans le coin supérieur gauche.
- `Console.SetCursorPosition()`: Place le curseur d'écriture à une position donnée de la console.

Attributs:

- `Console.Title`: Obtient ou définit le titre à afficher dans la barre de titre de la console.
- `Console.BackgroundColor`: Obtient ou définit la couleur d'arrière-plan de la console.
- `Console.ForegroundColor`: Obtient ou définit la couleur de premier plan de la console.

Beaucoup d'autres méthodes et attributs pour utiliser la console existent. De nouveau, lisez la page MSDN[2] de la classe Console pour récupérer de plus amples informations et des exemples de leur utilisation.

Passons à quelques exercices pour utiliser ce que vous avez appris sur la console !

3 Exercice 0: Hello WorldS

Commençons avec les bases. Vous connaissez certainement le "Hello world". Cet exercice va vous demander d'en implémenter des versions plus étoffées. Vous devez implémenter votre travail dans un projet nommée `HelloWorlds`.

Écrivez la méthode `Hello`:

```
1 static void Hello(string str);
```

Cette méthode prend un nom en paramètre sous la forme d'une chaîne de caractères et affiche une ligne dans la sortie standard saluant d'un "Hello" le nom indiqué, avec un point d'exclamation à la fin. Si ce nom est vide, celui-ci doit être remplacé par "World". Vous ne devez pas utiliser l'opérateur "+" pour cette méthode. En commentaire se trouvent les sorties attendues.

[2] `https://msdn.microsoft.com/en-us/library/system.console`

```
Hello("Jack");
// Hello Jack!
Hello("");
// Hello World!
```

Ensuite, écrivez la méthode `HelloWorldErr`:

```
static void HelloWorldErr();
```

Cette méthode doit écrire une ligne "Hello World!" dans la sortie standard d'*erreur* de la console.

```
HelloWorldErr();
// Hello World!
```

Enfin, écrivez la méthode `ImportantHello`:

```
static void ImportantHello();
```

Cette méthode écrit une ligne dans la sortie standard de la console "This is an important announcement: Hello World!" avec un fond jaune et un texte rouge. Une fois le message affiché, les couleurs doivent reprendre leur état d'origine. Pour faire cette méthode, vous aurez besoin des propriétés `Console.BackgroundColor` et `Console.ForegroundColor`. N'oubliez pas que vous pouvez les lire ou les écrire. Ces propriétés ont un type spécial, `ConsoleColor`. `ConsoleColor` est une `enum` (cette notion sera abordée dans un prochain TP). Étant donné que `ConsoleColor` est un type, vous pouvez créer des variables de ce type. Pour obtenir des valeurs de ce type, vous pouvez utiliser la syntaxe `ConsoleColor.macouleur`, où "macouleur" est une des couleurs listées sur la page MSDN de `ConsoleColor`.

Le projet de cet exercice ne doit PAS contenir de méthode `Main`.

4 Exercice 1: Add

Le but de cet exercice est de récupérer les arguments passés à votre programme via la ligne de commande. Par conséquent, le projet de cet exercice doit avoir une méthode Main. Vous devez implémenter votre travail dans un projet nommée `Add`.

Votre programme devra additionner les deux premiers arguments qui lui sont passés via la ligne de commande, et devra afficher le résultat sur une ligne dans la sortie standard. S'il n'y a pas assez d'arguments passés, un message sur la sortie standard d'erreur devra être affiché. Vous pourriez être intéressés par la page MSDN de `Int32.Parse`. Vous pouvez supposer que les deux premiers arguments seront toujours des entiers valides.

```
[sparrow@sea ~]$ ./Add.exe
Invalid number of operands
```

```
[sparrow@sea ~]$ ./Add.exe 42 24
66
[sparrow@sea ~]$ ./Add.exe -1 -3 8
-4
```

5 Exercise 2: Skyscraper

Pour le dernier TP, on vous a demandé de dessiner une maison. Aujourd'hui vous allez nous dessiner un gratte-ciel. Votre gratte-ciel aura un nombre variable d'étages. Chaque étage sera la propriété de la personne y vivant, et chaque propriétaire aura des goûts décoratifs (représentés par des couleurs) aléatoires. Bien sûr, le rez-de-chaussée et le toit de l'immeuble ne sont pas la propriété d'un particulier et auront donc une couleur ordinaire.

Vous devez implémenter votre travail dans un projet nommée `Skyscraper`.

5.1 Rez-de-chaussée et toit

Tout d'abord, faisons les parties simples et invariables.

Écrivez la méthode `Ground`:

```
static void Ground();
```

Cette méthode dessine le rez-de-chaussée, le rez-de-chaussée a la même couleur que celles par défaut de votre terminal, vous ne devez pas les modifier pour le moment. Voici un exemple de ce à quoi doit ressembler votre rez-de-chaussée (les longues lignes horizontales sont des suites de underscore `'_'`), la largeur des deux niveaux est de 14 caractères:

```
|     _____     |
|____|-||-|____|
```

Une fois faite, implémentez la méthode `Roof`:

```
static void Roof();
```

Cette méthode est très similaire à la précédente, sauf que celle-ci dessine le toit au lieu du rez-de-chaussée. Voici un exemple de ce à quoi votre toit doit ressembler:

```
 _____________
|             |
```

5.2 Mon bel étage

Maintenant que vous avez implémenté les parties triviales, passons à la partie amusante !

Implémentez la méthode `Floor`:

```
1 static void Floor();
```

`Floor` affiche un étage de l'immeuble, avec des couleurs d'arrière-plan et de premier plan aléatoires, un étage doit ressembler à ça:

```
|   |0|   |0|   |
|               |
```

Pour obtenir une `ConsoleColor` aléatoire, vous devez utiliser l'objet `Random`. Pour cela, nous vous fournissons une méthode:

```
1 static Random random = new Random();
2
3 static ConsoleColor GetRandomColor()
4 {
5     return (ConsoleColor)random.Next(0, 16);
6 }
```

Copiez ces 6 lignes dans votre code, à côté de vos autres méthodes. Vous n'avez pas besoin de comprendre ce que fait ce code. La méthode fournie, `GetRandomColor`, retourne une couleur aléatoire utilisable pour définir l'attribut `Console.BackgroundColor` ou `Console.ForegroundColor`. Si vous êtes curieux, vous pouvez lire la page MSDN du mot-clé `new`, de la classe `Random` et du type `enum`.

Vous êtes désormais capable de récupérer une `ConsoleColor` aléatoire. Utilisez cela pour réaliser votre méthode `Floor`.

La méthode `Floor` doit se comporter comme suit:

- Sauvegarder les couleurs précédentes de la console,
- Définir les couleurs de la console à des valeurs aléatoires,
- Afficher les 2 lignes de votre étage, seuls les 14 premiers caractères, correspondants à la largeur de l'immeuble, doivent être colorés, pas le reste de la console, pensez à la différence entre `Console.Write` et `Console.WriteLine`,
- Redéfinir les couleurs de la console aux valeurs qu'elles avaient avant l'appel à votre méthode.

5.3 Construisons quelque chose

Vous êtes maintenant capable de dessiner un rez-de-chaussée, un étage coloré et un toit. Implémentez maintenant la méthode `Floors`:

```
1 static void Floors(int n);
```

Cette méthode prend un entier en paramètre et affiche `n` étages colorés. Elle doit être récursive et appeler votre méthode `Floor`.

Implémentez maintenant la méthode finale, nommée `Skyscraper`:

```
1 static void Skyscraper(int n);
```

Cette méthode appelle simplement vos méthodes `Roof`, `Floors` et `Ground`.

Le projet de cet exercice ne doit PAS contenir de méthode `Main`.

6 Exercice 3: Jolly Roger

Salut, mousaillon! J'ai perdu le pavillon de mon bâtiment. Si tu peux m'en faire un nouveau, je te donnerai quelques pièces d'or issues de ma dernière prise.

Dans cet exercice, vous devrez dessiner en ASCII Art[3] un crâne au milieu de la longueur de la console.

Vous devez implémenter votre travail dans un projet nommée `JollyRoger`.

Vous devez premièrement implémenter la méthode `PrintSkull`.

```
1 static void PrintSkull();
```

Cette méthode affichera un crâne en ASCII art sur la sortie standard de la console. La console doit être vidée avant d'afficher le crâne. Ce dernier doit être affiché depuis la première ligne du haut de la console, et doit être centré sur la longueur de celle-ci.

Vous devriez regarder les pages de la MDSN pour `Console.BufferWidth`, `Console.BufferHeight`, et `Console.SetCursorPosition` pour écrire cette méthode. Vous pouvez trouver des crânes en ASCII Art ici: `http://www.asciiworld.com/-Death-Co-.html`. Si la console est trop petite pour accueillir le crâne, rien ne devrait être affiché.

Ensuite, vous implémentrez la méthode `MyJack`.

```
1 static void MyJack();
```

[3] `https://fr.wikipedia.org/wiki/Art_ASCII`

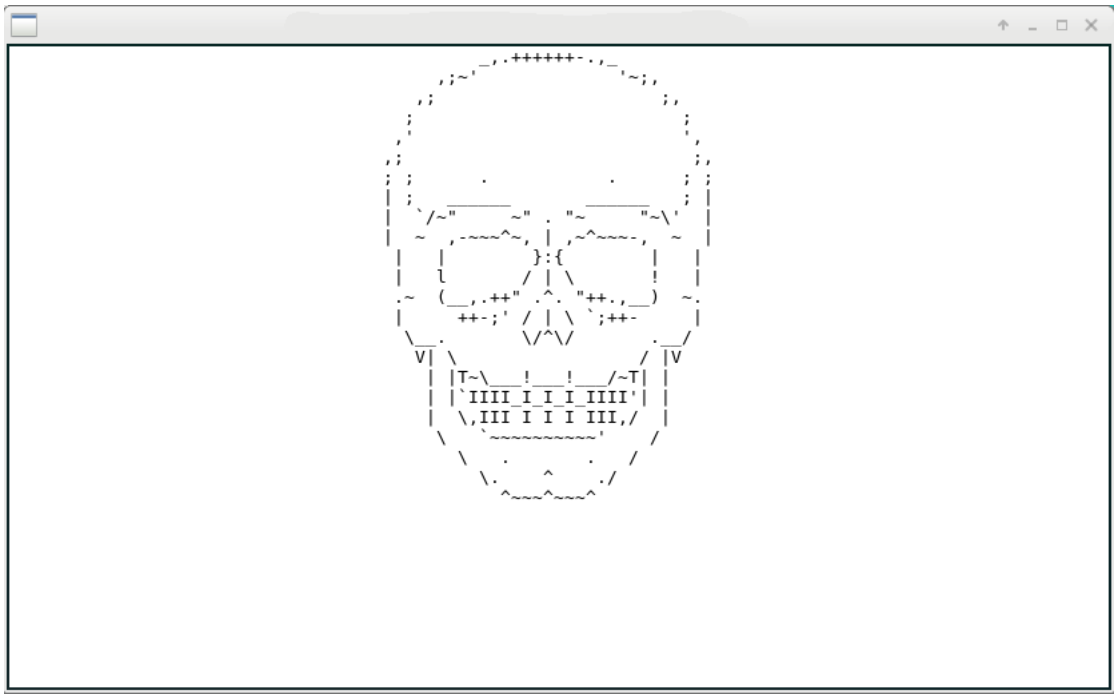

Figure 1: Affichage attendu

Cette méthode va d'abord afficher votre crâne une première fois. Puis elle va attendre une entrée de l'utilisateur avant d'afficher le crâne de nouveau, mais avec un fond blanc et un texte noir, plutôt que les couleurs précédentes de la console. Elle attendra alors encore une entrée utilisateur pour finalement afficher, après avoir effacé la console "The code is law" (avec les couleurs de la console avant l'affichage noir sur blanc).

Le projet de cet exercice ne doit PAS contenir de méthode `Main`.

7 Exercise 4: -.-. .- -. ... .-.. .- - --- .-.

Dans cet exercice, vous devrez écrire un petit programme qui traduit un message écrit en utilisant l'alphabet latin (aka. le français) en code morse. Votre programme prendra un argument, un entier indiquant le nombre de lettres que votre programme doit lire puis traduire. Jusqu'à ce que toutes les lettres soient lues par le programme, celui-ci doit attendre que l'utilisateur écrive le caractère suivant. À la fin, il doit afficher le message traduit sur la sortie standard. Si aucun argument n'est donné, ou trop, le programme doit écrire un message sur la sortie *d'erreur* standard.

Vous devez implémenter cet exercice dans un projet nommé `Morse`. Ce projet doit contenir une méthode `Main`.

Ci-dessous, un exemple d'utilisation du programme que vous devez créer:

```
[sparrow@sea ~]$ ./Morse.exe
Invalid arguments.
[sparrow@sea ~]$ ./Morse.exe 42 51
Invalid arguments.
[sparrow@sea ~]$ ./Morse.exe 3
SOS
... --- ...
[sparrow@sea ~]$ ./Morse.exe 11
Votai Test.
...- --- - .- .. / - . ... - .
[sparrow@sea ~]$ ./Morse.exe 19
The code is the law
- .... . / -.-. --- -.. . / .. ... / - .... . / .-.. .- .--
[sparrow@sea ~]$
```

7.1 Morse

"L'alphabet morse ou code morse, est un code permettant de transmettre un texte à l'aide de séries d'impulsions courtes et longues, qu'elles soient produites par des signes, une lumière, un son ou un geste. L'alphabet morse international encode l'alphabet latin basic ISO, certains caractères latin complémentaires, les chiffres arabes, et quelques caractères de ponctuation et signaux procéduraux sous la forme de suites standardisées de signaux courts et longs appelés *points* et *tirets*." [4]

Plus simplement, le code morse ne change pas les mots, il s'agit uniqument d'une autre manière d'écrire / prononcer les lettres d'un message. Chaque caractère d'un message est encodé comme une suites de *points* (`'.'`) et *tirets* (`'-'`). Pour rendre le message produit plus lisible, chaque caractère encodé est séparé de son suivant par un *espace* (`' '`). Le caractère *espace*, séparant les mots dans le message original, est un peu particulier. Étant donné qu'il donne des informations sur le message original (les mots), il ne peut pas être ignoré, mais ne peut pas être encodé comme un simple *espace* non plus, car ce caractère est déjà utilisé pour séparer les lettres. Vous représenterez donc les caractères *space* avec un *slash* (`'/'`). Par exemple, le message `"foo bar"` serait affiché en code morse comme `"..-. --- --- / -... .- .-."`.

7.2 Convertir les caractères

Implémentez la méthode `LetterToMorse`, qui convertit un caractère en string représentant son encodage en code morse:

[4] `https://en.wikipedia.org/wiki/Morse_code`

```
static string LetterToMorse(char c);
// LetterToMorse('f') -> "..-."
// LetterToMorse('F') -> "..-."
// LetterToMorse('7') -> "--..."
// LetterToMorse(' ') -> "/"
// LetterToMorse('#') -> "#"
```

Cette méthode prend en paramètre un caractère à traduire et retourne la string représentant l'encodage du caractère. Si le caractère est une des 26 lettres de l'alphabet (majuscule ou minuscule) ou un chiffre, utilisez la table disponible sur la page Wikipédia[5] du code morse comme référence pour la conversion. Si le caractère est un espace, retournez la string `"/"`. Si le caractère a une toute autre valeur, vous devez retourner une string ne contenant que le caractère donné.

7.3 Lire et traduire

Implémentez la méthode `Translate`:

```
static string Translate(int n);
```

Cette méthode doit être récursive. Elle lit `n` caractères entrés par l'utilisateur sur l'entrée standard et retourne la string contenant la traduction entière en code morse du message. Vous devez bien sûr utiliser votre méthode `LetterTomorse`. N'oubliez pas de séparer chaque traduction de lettre par un espace (`' '`) dans la chaîne résultante.

7.4 Exécutons ça

Vous allez maintenant écrire le reste de votre algorithme comme un programme fonctionnel. Pour cela, vous devez coder dans la méthode `Main`.

Vous devez d'abord vérifier le nombre d'arguments passés à votre programme. S'il n'y a pas d'argument, ou si plus d'un sont donnés, vous devez écrire un message sur la sortie *d'erreur* standard (`stderr`)et s'arrêter.

Ensuite, celui-ci convertit l'argument en `int` (souvenez-vous de `Int32.Parse`), appelle votre méthode `Transale` et affiche ensuite sur la ligne suivante de la console le résultat de la traduction en code morse.

[5] `https://upload.wikimedia.org/wikipedia/commons/b/b5/International_Morse_Code.svg`

8 Boni:

Si vous avez fini tous les exercices et souhaitez essayer plus de choses avec la console, vouz pouvez améliorer n'importe lequel des exercices précédents comme vous le souhaitez. Voic une liste d'améliorations possibles:

- Rendre le crâne de l'exercice 3 animé,
- Changer le titre de la fenêtre de la console selon le programme lancé,
- Utiliser la méthode `Console.Beep` pour jouer le message en code morse après la traduction,
- Écrire une variante du traducteur, qui traduit des messages dans l'autre sens: de code morse vers l'alphabet latin,
- N'importe quoi d'autre, soyez créatifs.

N'oubliez pas de lister et expliquer tous vos bonii dans votre fichier README.

The code is the law.