TP C#2 : Escape from Ik

1 Consignes de rendu

L'architecture de rendu est la suivante :

```
-- rendu-tp-login_x.zip
   |-- login_x/
       |-- AUTHORS
       |-- Ex1.cs
       |-- Ex2.cs
       |-- Ex3.cs
       |-- Ecard.cs (facultatif)
```

Bien entendu, vous devez remplacer "`login_x`" par votre propre login.

N'oubliez pas de vérifier les points suivants avant de rendre :
- l'archive ne doit pas contenir de fichiers autres que ceux demandés ci-dessus
- le fichier `AUTHORS` doit être au format habituel
- et surtout, **le code doit compiler**

2 Cours

Comme son titre ne l'indique pas, ce TP porte sur le mode console. Nous avons tout à fait conscience que vous avez hâte de coder, mais il est tout de même important de faire un peu de théorie avant.

Cette section de cours se découpe en deux grandes parties : la première comprend une petite présentation de la console et les bases de son utilisation ; et la deuxième vous parlera de la boucle `while`.

À moins de déjà être un habitué de la console, nous vous recommandons de ne pas sauter la première partie : la console est un outil indispensable dont vous vous servirez toute votre vie.

2.1 La console

2.1.1 Qu'est-ce que c'est ?

Jusqu'ici, la plupart d'entre vous ont seulement utilisé leur ordinateur via une interface graphique : votre interaction avec la machine se faisait en utilisant des "images" (fenêtres, boutons, etc.) plutôt que du texte. En anglais, on parle de **GUI** (Graphical User Interface).

Il existe cependant des programmes n'ayant pas d'interface graphique, et qui doivent donc s'utiliser d'une autre manière : via la console. La console vous permet en effet de lancer et d'utiliser des programmes en tapant des lignes de texte ayant une signification particulière pour l'ordinateur : les **commandes**. C'est de là que vient le terme **CLI** (Command-Line Interface).

Un exemple : pour afficher le contenu d'un dossier en interface graphique, on utilise l'explorateur Windows. Dans la console, on utilise la commande `dir` :

```
C:\Users\ACDC> dir
 Directory of C:\Users\ACDC

31/08/2012  09:13     <DIR>          .
31/08/2012  09:13     <DIR>          ..
20/09/2012  05:42     <DIR>          tp3
30/09/2012  14:27     <DIR>          tp4
02/10/2012  20:32             16 317 rendu.txt
```

2.1.2 À quoi ça sert ?

Vous vous demandez peut-être à quoi sert la console, et pourquoi elle existe alors que les interfaces graphiques sont plus jolies et plus intuitives.

Tout d'abord, il faut savoir que ces jolies interfaces graphiques demandent tout de même une puissance de calcul qui n'était tout simplement pas disponible aux débuts de l'informatique. À l'époque, on n'avait donc pas le choix : le seul moyen d'utiliser un ordinateur était de passer par la console.

De nos jours, la console est encore très utilisée. En effet, bien que moins intuitive, elle reste plus rapide et plus pratique que l'interface graphique, pour peu que l'on sache s'en servir.

Un exemple très simple : mettons que vous vouliez compter le nombre d'images au format JPG dans un dossier. Avec un explorateur de fichiers, vous êtes probablement condamnés au mieux à trier les photos par type et les compter en les sélectionnant, et au pire à les compter manuellement.

Avec une console sur un système d'exploitation de type Unix (sous Linux par exemple), vous pouvez taper la commande suivante :

```
> ls -l | grep jpg | wc -l
103
```

Si on précise qu'il s'agit d'une commande Unix, c'est parce que la console de base de Windows est assez réduite comparée à ce qui se fait sur les autres OS. À vrai dire, elle risque même de vous donner une mauvaise image de la console, mais ne vous inquiétez pas : vous découvrirez toutes les capacités d'une vraie ligne de commande en temps voulu.

Sachez qu'il existe une autre console de Microsoft, Powershell, qui est assez proche des terminaux qu'on retrouve sous Unix. Cependant, elle n'est pas disponible sur tous les Windows. Nous allons donc nous contenter de la console de base pour ce TP.

2.1.3 Comment ça s'utilise ?

Sous Windows, la console de base est `cmd.exe`. Pour la lancer, appuyez sur WinKey + R, puis tapez "cmd" dans le champ de texte de la fenêtre qui s'affiche.

Elle possède un certain nombre de commandes de base qui permettent d'interagir avec la machine. Celles qui vous intéresseront aujourd'hui sont les suivantes :

dir : pour lister l'ensemble des fichiers et dossiers présents dans le dossier courant.

cd : pour se déplacer d'un dossier à l'autre.

Et pour votre culture (et au cas où) :

copy : pour copier des fichiers ou dossiers

move : pour déplacer des fichiers ou dossiers

del : pour supprimer un fichier

cls : pour remettre l'affichage de la console à zéro

Pour exécuter une commande, il suffit d'entrer le nom de la commande (ou le chemin vers l'exécutable) dans votre terminal suivi de zéro ou plusieurs arguments. `cd`, par exemple, prend comme argument le dossier dans lequel on veut se rendre.

Pour l'exemple, imaginons que dans le cadre du TP vous voulez lancer l'exécutable que vous venez de compiler.

Quand vous lancez la console, vous vous trouvez dans votre dossier utilisateur. La première étape est donc de naviguer vers le dossier où se trouve votre exécutable :

```
C:\Users\Thiel> cd Projects\sapin\bin\Debug
```

Bien entendu, vous remplacerez le chemin de l'exemple par celui que *vous* avez utilisé pour le TP.

Pour lister le contenu du dossier et vérifier que c'est bien là que se situe votre programme, utilisez la commande `dir` :

```
C:\Users\Thiel\Projects\sapin\bin\Debug> dir
 Directory of C:\Users\Thiel\Projects\sapin\bin\Debug

06-Nov-12  07:21 PM    <DIR>          .
06-Nov-12  07:21 PM    <DIR>          ..
06-Nov-12  07:22 PM             5,632 sapin.exe
               1 File(s)          5,632 bytes
               2 Dir(s)   3,359,334,400 bytes free
```

Enfin, pour exécuter votre programme, tapez simplement son nom :

```
C:\Users\Thiel\..\Projects\sapin\sapin\bin\Debug> sapin.exe
    *
   ***
  **o**
 *******
*o*o*o*o*
    #
    #
```

2.2 La boucle `while`

Vous verrez les boucles plus en détails dans le prochain TP, mais pour celui-ci nous allons avoir besoin d'au moins l'une d'entre elles. Nous allons donc parler un peu de la boucle `while`.

Pour faire simple, les boucles sont des structures de contrôle permettant de répéter des instructions (on parle d'**itération**). On leur donne une condition pour indiquer quand elles doivent cesser de se répéter.

La syntaxe de la boucle `while` est la suivante :

Boucle `while` en C#

```
while (<condition>) {
    /* instructions */
}
```

Équivalent en langage algo

```
tant que <condition> faire
    /* instructions */
fin tant que
```

Tant que la `<condition>` est vraie, les *`/* instructions */`* vont se répéter.

Un exemple typique de parcours de tableau :

```
string[] args = new string[] { "Nathalie", "Krisboul", "Xavier" };
int i = 0;

while (i < names.Length) {
    Console.WriteLine(names[i]);
    i++;
}
```

Vous verrez les tableaux plus en détails dans le prochain TP. Pour l'instant, retenez juste les points suivants :

- les tableaux sont des variables contenant plusieurs éléments.
- vous pouvez accéder à un élément en particulier en indiquant son numéro entre crochets après le nom du tableau.
- l'indexation des tableaux commence à 0. Par exemple, le troisième élément d'un tableau se situe à l'index 2.
- le champ `Length` contient le nombre d'éléments du tableau. Dans l'exemple ci-dessus, `names.Length` vaut donc 3.

Dans l'exemple, on utilise donc une variable `i` qui va servir d'index pour accéder aux éléments du tableau. On l'initialise à 0, et on l'incrémente jusqu'à atteindre le nombre d'éléments dans le tableau.

3 Exercices

Les exercices 1 et 2 sont découpés en paliers, qui sont là pour vous guider et pour guider la notation. Les paliers les plus élevés rapportent évidemment plus de points et sont votre objectif. Rien ne vous empêche d'y aller directement, sans passer par les paliers inférieurs, si vous vous en sentez capables.

Attention, on ne vous demande de rendre que le code du plus haut palier atteint ! Si vous montez jusqu'au palier 4, ne rendez que le code du programme demandé à ce palier. Ne gardez pas le code des paliers inférieurs en les mettant par exemple dans une méthode ou un fichier à part (ce qui vous fera d'ailleurs un rendu sale).

3.0 Exercice 0 : MyToolbox

Avant de passer à la programmation à proprement parler, votre première tâche consistera à apprendre à vous servir de la documentation officielle du langage C#, disponible sur le site du MSDN (MicroSoft Developer Network).

Cet exercice est très important puisque vous rencontrerez tout au long de l'année (et même de votre carrière) de nombreux problèmes, et vous n'aurez pas toujours l'aide de vos professeurs, de vos assistants ou de vos amis pour les résoudre ! Sans compter que ce ne sont pas forcément des documentations sur pattes (il peut arriver d'en rencontrer dans la nature, cela dit).

Votre mission, si vous l'acceptez :

1. Trouvez l'adresse du site du MSDN.
2. Trouvez la page "Library".
3. Utilisez le champ de recherche pour vous renseigner sur la méthode `Console.WriteLine(String)`. Que fait-elle ? Que prend-elle en argument ?
4. Trouvez la différence entre `Console.WriteLine(String)` et `Console.Write(String)`.
5. Passez à la méthode `Console.ReadLine()`. Que renvoie-t-elle ?

C'est tout pour le moment. Sachez cependant que pour terminer le TP, vous pouvez avoir besoin de vous renseigner sur les choses suivantes :

- `Console.Read()`
- `Console.ReadKey()`
- `Console.ReadLine()`
- `Console.Write(String)`
- `Console.WriteLine(String)`
- `Int32.Parse(String)`
- `Int.TryParse(String, Int32)` (celle-ci est un peu particulière, assurez-vous de bien lire toute la documentation avant d'appeler vos ACDCs !)
- `Console.Clear()`
- `Console.ForegroundColor`
- `Console.BackgroundColor`
- `ConsoleColor.DarkGreen`
- `System.Threading.Thread.Sleep(Int32)` (utile pour animer votre sapin)

3.1 Exercice 1 : Mini-calculatrice

Cet exercice a pour but de vous faire maîtriser la gestion des arguments passés à votre programme.

Lorsqu'un programme est lancé avec des arguments, ces arguments sont stockés dans un tableau de `string`s appelé `args`, passé en paramètre à votre fonction `Main`.

Pour connaître le nombre d'arguments passés à votre programme, vous avez accès au champ `Length` qui retourne la taille du tableau de `string`s.

3.1.1 Palier 1

Écrire un programme prenant un seul argument et l'affichant sur la sortie standard. S'il n'y a pas d'argument, le programme ne fait rien.

Exemples :

```
C:\tp> ex1.exe
C:\tp> ex1.exe 42
42
```

3.1.2 Palier 2

Modifier le programme pour qu'il prenne un nombre variables d'arguments et les affiche tous, un par ligne. S'il n'y a pas d'arguments, le programme ne fait rien.

Exemple :

```
C:\tp> ex1.exe ce tp dechire
ce
tp
dechire
```

3.1.3 Palier 3

Modifier le programme pour qu'il prenne en argument des nombres entiers et affiche leur somme. On ne vous demande pas de gestion d'erreur (autrement dit on considère –pour ce palier– que tous les arguments sont des nombres valides.)

Exemple :

```
C:\tp> ex1.exe 10 22 8 2
42
```

3.1.4 Palier 4

Modifier le programme pour qu'il affiche un message d'erreur (au format de votre choix) au lieu de la somme si l'un des arguments n'est pas un nombre.

Exemple :

```
C:\tp> ex1.exe 10 22 lol 2
Erreur: "lol" n'est pas un nombre
```

3.1.5 Palier 5

Modifier le programme pour qu'il prenne en argument une suite alternant nombres et symboles indiquant l'opération mathématique à effectuer, puis affiche le résultat de l'opération.

On ne vous demande pas de gestion d'erreur (les arguments seront toujours des nombres ou des opérateurs valides et seront toujours dans le bon ordre) ni de gestion de la précédence mathématique des opérateurs (vous effectuerez le calcul de la gauche vers la droite, dans l'ordre dans lequel se présentent les arguments.)

Exemples :

```
C:\tp> ex1.exe 4 + 5 * 3                    # (4 + 5) * 3
27
C:\tp> ex1.exe 2 * 12 - 5 / 2               # ((2 * 12) - 5) / 2
9
```

3.1.6 Palier 6 (bonus)

Modifier le programme pour qu'il gère la précédence des opérateurs.

Exemples :

```
C:\tp> ex1.exe 4 + 5 * 3                    # 4 + (5 * 3)
19
C:\tp> ex1.exe 2 * 12 - 5 / 2               # (2 * 12) - (5 / 2)
22
```

3.1.7 Palier 7 (bonus libres)

Quelques idées :
- Affichage d'une aide en cas d'arguments invalides
- Gestion des nombres flottants
- Opérateurs d'incrémentation/décrémentation

  Exemple :

  ```
  C:\tp> ex1.exe ++3 * --5
  8
  ```

- Surprenez-nous !

3.2 Exercice 2 : Rotations

3.2.1 Palier 1

Écrire la fonction suivante :

```
static char Rot13(char c)
```

qui applique un rot13 [1] au caractère spécifié (uniquement s'il s'agit d'une lettre minuscule ou majuscule non-accentuée) et le retourne.

3.2.2 Palier 2

Modifier la fonction précédente pour obtenir

```
static string Rot13(string s)
```

qui applique un rot13 à une chaîne entière et la retourne.

3.2.3 Palier 3

Écrire un programme qui lit une chaîne sur l'entrée standard et l'affiche.

Exemple :

```
C:\tp> ex2.exe
In ACDC you trust   # Cette ligne est rentree par l'utilisateur
In ACDC you trust   # Cette ligne est affichee par le programme
```

3.2.4 Palier 4

Écrire un programme qui lit une chaîne sur l'entrée standard, lui applique un rot13 et l'affiche.

Exemple :

```
C:\tp> ex2.exe
In ACDC you trust   # Cette ligne est rentree par l'utilisateur
Va NPQP lbh gehfg   # Cette ligne est affichee par le programme
```

3.2.5 Palier 5 (bonus)

Modifier le programme pour qu'il applique cette fois-ci un rot N, la valeur de N étant passée en argument.

Exemple :

```
C:\tp> ex2.exe 3
In ACDC you trust   # Cette ligne est rentree par l'utilisateur
Lq DFGF brx wuxvw   # Cette ligne est affichee par le programme
```

1. http ://fr.wikipedia.org/wiki/ROT13

3.3 Exercice 3 : Noyeux Joël, le Retour de la Vengeance Finale 2

3.3.1 Prologue

Souvenez-vous : vous êtes employé chez Ik, et c'est vous qui avez été chargé de recalculer les acomptes des clients de l'entreprise suite à la décision du PDG de les baisser de 10%.

Après avoir fièrement accompli cette tâche éreintante, vous vous apprêtez à rentrer chez vous pour goûter à un repos bien mérité lorsque votre boss vous rattrape dans le couloir et vous confie une nouvelle tâche : annoncer la bonne nouvelle aux clients en spammant leur boîte email.

Après avoir contacté les syndicats pour leur faire part de cette prise en otage honteuse, vous vous attelez à la tâche dans l'espoir de pouvoir rentrer chez vous avant midi parce que faut pas exagérer non plus, on a pas idée de retenir les gens au boulot comme ça.

Au bout d'une heure, vous avez pondu un petit programme OCaml pour envoyer des emails en masse à tous les clients de l'entreprise, et en prime vous avez même dessiné un petit sapin en ASCII art parce que c'est Noël. Malheureusement, à l'instant où vous vous apprêtez à appuyer sur Entrée pour lancer l'exécution du programme, votre boss surgit et annonce que OCaml c'est fini, c'est dépassé, à partir de maintenant on ne fera plus que du C#, puis se volatilise en pause déjeuner.

Après avoir brièvement sombré dans la dépression, vous vous rendez compte qu'il est 10h45 du matin et que vous n'avez toujours pas fini votre journée. C'est décidé : vous appelez la police.
Ça tombe bien, c'est Roger, votre cousin à la brigade des douanes, qui répond : il vous envoie immédiatement un car de CRS pour vous sortir de là.

En attendant, vous décidez de recoder votre programme en C# afin de ne pas éveiller les soupçons.

3.3.2 Partie 1 : Mon beau Sapin

Après avoir rapidement réécrit sendmail en C#, vous attaquez le plus dur : l'affichage d'un sapin ASCII.

Écrivez la fonction suivante :

```
static void Sapin(int n)
```

qui affiche un sapin de hauteur n avec un tronc de hauteur $n/2$ sur la sortie standard.

Exemples :

```
                    *
                   ***
                  *****
                 *******
                *********
               ***********
              *************
   *         ***************
  ***       *****************
 *****               #
*******              #
   #                 #
   #                 #

sapin(4);       sapin(9);
```

Le même sapin(4) avec les espaces mis en valeur :

```
␣␣␣*
␣␣***
␣*****
*******
␣␣␣#
␣␣␣#
```

3.3.3 Partie 2 : Régis est daltonien

Ouf ! C'est fait. Le car de CRS sera là d'une minute à l'autre et vous pourrez enfin rentrer chez vous. Pour maintenir une attitude non-suspecte vis-à-vis de l'assaut imminent des forces de l'ordre, vous décidez de continuer votre programme. À court d'idées, vous demandez son avis à Régis de la compta qui vous suggère d'ajouter un peu de couleurs.

Modifiez la fonction `Sapin` pour qu'elle affiche le tronc du sapin en `DarkYellow` et le reste en `DarkGreen`.

3.3.4 Partie 3 : "Il faut absolument que j'appelle Armand"

Après avoir contemplé votre œuvre pendant 3 bonnes minutes, vous êtes brusquement ramené à la réalité par la désagréable sensation d'être observé. En levant la tête, vous croisez le regard interrogateur de Josiane la secrétaire.

En panique, vous remplacez votre nom de famille dans la base de données des employés pour éviter qu'elle ne remonte à votre cousin douanier Roger, puis, pour échapper à son regard accusateur, vous décidez de changer de bureau.

Étant toujours entouré de potentiels agents doubles, il vous faut encore faire semblant de travailler. À court d'idées brillantes, vous décidez de vous contenter de décorer un peu plus votre sapin.

Ajoutez des boules colorées à votre sapin. Attention, il ne faut pas que les boules de Noël couvrent la totalité de la surface du sapin.

```
         *
        ***
       **o**
      *o*****
     *****o***
    **o*o***o**
   ********o****
  ***o***o***o***
*****************
         #
         #
         #
         #

    sapin(9);
```

3.3.5 Partie 4 : Christmas Lights

Votre sapin attire de plus en plus l'attention de vos collègues. Heureusement, vous avez une idée qui va vous permettre de retourner la situation en votre faveur : vous allez utiliser votre sapin pour faire diversion et vous permettre de prendre la fuite. Mais pour cela, il faut qu'il soit beaucoup plus flashy.

Animez votre sapin ! Faites en sorte que vos boules de Noël changent de couleur toutes les secondes. Si vous n'êtes pas épileptique, changez aussi la couleur du fond (oui oui, toutes les secondes).

3.3.6 Épilogue

Enfin ! L'heure d'asséner le coup de grâce. Vous modifiez votre programme précédent pour envoyer des emails en masse, non pas aux clients de l'entreprise mais à tous vos collègues, en y joignant votre sapin épileptique.

Au bout de quelques minutes seulement, les effets de votre plan diabolique se font sentir : vos collègues, hypnotisés par votre œuvre d'art, commencent à avoir la nausée et s'évanouissent par dizaines. C'est le moment que vous choisissez pour jaillir de votre fauteuil et vous précipiter vers la sortie.

Malheureusement, votre boss, qui revient tout juste de sa pause déjeuner et n'a donc pas encore lu ses emails, vous aperçoit et se lance à votre poursuite. En panique, vous changez de trajectoire et partez vers la fenêtre. Dans un superbe effet de `slow motion`*, vous vous élancez et traversez la vitre, la brisant au passage. Après une chute de plus de un étage, vous atterrissez dans une benne à ordures et perdez connaissance.*

Quand vous reprenez vos esprits, vous vous retrouvez dans une scène de dévastation. L'assaut des CRS ayant finalement eu lieu, l'immeuble de bureau est à feu et à sang. Plusieurs de vos collègues sont à l'hôpital, quelques-uns à l'asile, et votre boss a été violemment neutralisé par les forces de l'ordre lorsqu'il a brandi sa cravate comme une arme à feu.

Soudainement pris par un sentiment de culpabilité, vous décidez de vous faire pardonner auprès de vos collègues (ou tout du moins ceux encore sains d'esprit) en leur envoyant une jolie carte de Noël.

Bonus : écrivez dans le fichier `Ecard.cs` la meilleure carte de Noël interactive du monde. Faites-vous plaisir : tout est autorisé ! Couleurs, animations, mini-jeu, etc. En plus de voir s'attribuer des points bonus à la pelle, les meilleurs cartes de Noël seront inscrites dans le *Hall of Fame* des épitéens les plus créatifs [2].

4 Conclusion

In ACDC you trust.

2. Ce *Hall of Fame* peut ou peut ne pas exister